

CS4840 Embedded Systems

FPGA Raytracer — Source Code

May 13, 2026

Contents

1	Hardware (SystemVerilog + Qsys + Quartus)	2
	raytracer.sv	2
	raytracer_hw.tcl	57
	soc_system_assignment_defaults.qdf	60
	soc_system.qsys	60
	soc_system.tcl	73
2	Software (HPS Driver + Userspace Server)	78
	SW/raytracer_batch.h	78
	SW/raytrace_batch_drve.c	82
	SW/fpga/server.c	86
	SW/fpga/render_frame.c	92
	SW/fpga/Makefile	94
3	Desktop Client + Demos	94
	SW/desktop/client.py	94
	SW/desktop/camera.py	102
	SW/desktop/pong.py	107
	SW/desktop/sandbox.py	112
4	Python Reference + Cocotb Tests	117
	raytracer_fp.py	117
	raytracer_reflect_ref.py	124
	test_raytracer_batch.py	129
	test_cone.py	142
	test_fp.py	145
	test_fp_div.py	146
	test_fp_sqrt.py	147
	test_fp_inv.py	148
	Makefile.raytracer_batch_pipe	150
	Makefile.div	150
	Makefile.mul	150
	Makefile.sqrt	150
	Makefile.fp_inv	150

1 Hardware (SystemVerilog + Qsys + Quartus)

raytracer.sv

```
1 // =====
2 // Package: per-ray reflection metadata.
3 //
4 // `ray_meta_t` is an 11-bit packed struct that travels alongside each ray
5 // through every pipeline stage. For primary rays it's all zero; reflection
6 // rays inherit/derive their values when the parent's stage G spawns a child:
7 //
8 //   depth      -- bounce count. 0 = primary; incremented on each spawn.
9 //               Stage G refuses to spawn when depth >= max_depth.
10 //   excluded    -- one-hot bitmask of which primitive this ray was just
11 //                spawned off of. Stage C masks that primitive's hit so
12 //                the new ray doesn't immediately self-intersect at the
13 //                surface origin. excluded[0]=skip s0, [1]=skip s1,
14 //                [2]=skip cone. 000=none (primary).
15 //   total_shift -- cumulative power-of-2 attenuation along the chain. Each
16 //                spawn adds the parent-primitive's reflect_shift. The
17 //                accumulator emits (color * bright) >> total_shift, so
18 //                deep bounces contribute progressively less.
19 // =====
20 package raytracer_pkg;
21   localparam int RT_WORD = 27;
22   // Tag width sized for production BATCH=480.
23   localparam int RT_TAG_W = 9;
24
25   typedef struct packed {
26     logic [2:0] depth;      // 0 (primary) .. max_depth (terminal)
27     logic [2:0] excluded;   // {cone, s1, s0} one-hot self-skip bitmask
28     logic [4:0] total_shift; // running >> count for accumulator weight
29   } ray_meta_t;
30
31   typedef struct packed {
32     logic signed [RT_WORD-1:0] r, g, b;
33   } rgb_t;
34
35   // Sphere primitive - geometry + surface appearance + reflectivity.
36   // r_sq is pre-squared by SW.
37   typedef struct packed {
38     logic signed [RT_WORD-1:0] x, y, z;
39     logic signed [RT_WORD-1:0] r_sq;
40     rgb_t color;
41     logic [4:0] reflect_shift;
42   } sphere_t;
43
44   // Y-axis cone primitive. Apex on top at (apex_x, apex_y, apex_z),
45   // single-half (opens downward) over y [apex_y - height, apex_y].
46   // Surface  $(-xAx)^2 + (-zAz)^2 = k\_sq \cdot (-yAy)^2$ . Bottom is capped by a
47   // disc of radius2 =  $k\_sq \cdot height^2$ . norm_factor =  $\sqrt{k\_sq \cdot (1+k\_sq)}$ 
48   // is SW-precomputed to skip a sqrt in HW. reflect_shift = 0 → matte.
49   typedef struct packed {
50     logic signed [RT_WORD-1:0] apex_x, apex_y, apex_z;
51     logic signed [RT_WORD-1:0] k_sq;
52     logic signed [RT_WORD-1:0] height;
53     logic signed [RT_WORD-1:0] norm_factor;
54     rgb_t color;
55     logic [4:0] reflect_shift;
56   } cone_t;
57
58   // Floor (infinite-ish ground plane) - checker colors + pattern mode +
59   // bounded half-extends. Consumed only by rtl_hit_resolve.
60   typedef struct packed {
61     rgb_t pc1, pc2;
62     logic [1:0] mode;
63     logic signed [RT_WORD-1:0] lim_x, lim_z;
64   } floor_t;
65
66   // Top-level scene bundle. Each stage receives this and reads only the
67   // fields it needs; Quartus prunes unused field-driven logic.
68   typedef struct packed {
69     sphere_t s0, s1;
```

```

70     cone_t    cone;
71     floor_t   floor;
72 } scene_t;
73
74 // Per-ray ride-along through every stage handoff.
75 typedef struct packed {
76     logic      sv;
77     logic [RT_TAG_W-1:0] tag;
78     ray_meta_t meta;
79 } pipe_baggage_t;
80
81 localparam logic [2:0] HIT_TYPE_SKY    = 3'b000;
82 localparam logic [2:0] HIT_TYPE_S0    = 3'b001;
83 localparam logic [2:0] HIT_TYPE_S1    = 3'b010;
84 localparam logic [2:0] HIT_TYPE_PLANE = 3'b011;
85 localparam logic [2:0] HIT_TYPE_CONE  = 3'b100;
86
87 // Floor pattern modes; selects rtl_hit_resolve's chk_bit derivation.
88 localparam logic [1:0] FLOOR_MODE_1M_CHECK    = 2'b00;
89 localparam logic [1:0] FLOOR_MODE_SIERPINSKI = 2'b01;
90 localparam logic [1:0] FLOOR_MODE_DIAGONAL    = 2'b10;
91 localparam logic [1:0] FLOOR_MODE_FINE_CHECK  = 2'b11;
92
93 localparam int ADDR_CONTROL          = 7'd0;
94 localparam int ADDR_STATUS           = 7'd1;
95 localparam int ADDR_LIGHT_X          = 7'd4;
96 localparam int ADDR_LIGHT_Y          = 7'd5;
97 localparam int ADDR_LIGHT_Z          = 7'd6;
98 localparam int ADDR_R0_SQ            = 7'd7;
99 localparam int ADDR_R1_SQ            = 7'd8;
100 localparam int ADDR_DMA_CTRL         = 7'd9;
101 localparam int ADDR_DMA_STATUS       = 7'd10;
102 localparam int ADDR_FRAME_BUF_BASE   = 7'd15;
103 localparam int ADDR_SC1_X            = 7'd16;
104 localparam int ADDR_SC1_Y            = 7'd17;
105 localparam int ADDR_SC1_Z            = 7'd18;
106 localparam int ADDR_MAX_DEPTH        = 7'd19;
107 localparam int ADDR_REFLECT_SHIFT_0  = 7'd20;
108 localparam int ADDR_REFLECT_SHIFT_1  = 7'd21;
109 localparam int ADDR_CAM_X            = 7'd22;
110 localparam int ADDR_CAM_Z            = 7'd23;
111 localparam int ADDR_RIGHT_X          = 7'd24;
112 localparam int ADDR_RIGHT_Z          = 7'd25;
113 localparam int ADDR_UP_X             = 7'd26;
114 localparam int ADDR_UP_Y             = 7'd27;
115 localparam int ADDR_UP_Z             = 7'd28;
116 localparam int ADDR_FWD_X            = 7'd29;
117 localparam int ADDR_FWD_Y            = 7'd30;
118 localparam int ADDR_FWD_Z            = 7'd31;
119 // Visual-upgrades CSRs -(#0#5)
120 localparam int ADDR_SCO_X             = 7'd32;
121 localparam int ADDR_SCO_Y             = 7'd33;
122 localparam int ADDR_SCO_Z             = 7'd34;
123 localparam int ADDR_PC1_R             = 7'd35;
124 localparam int ADDR_PC1_G             = 7'd36;
125 localparam int ADDR_PC1_B             = 7'd37;
126 localparam int ADDR_PC2_R             = 7'd38;
127 localparam int ADDR_PC2_G             = 7'd39;
128 localparam int ADDR_PC2_B             = 7'd40;
129 localparam int ADDR_SCOLO_R           = 7'd41;
130 localparam int ADDR_SCOLO_G           = 7'd42;
131 localparam int ADDR_SCOLO_B           = 7'd43;
132 localparam int ADDR_SCOL1_R           = 7'd44;
133 localparam int ADDR_SCOL1_G           = 7'd45;
134 localparam int ADDR_SCOL1_B           = 7'd46;
135 localparam int ADDR_FLOOR_MODE        = 7'd47;
136 localparam int ADDR_FLOOR_LIM_X       = 7'd48;
137 localparam int ADDR_FLOOR_LIM_Z       = 7'd49;
138 localparam int ADDR_CONE_X            = 7'd50;
139 localparam int ADDR_CONE_Y            = 7'd51;
140 localparam int ADDR_CONE_Z            = 7'd52;
141 localparam int ADDR_CONE_K_SQ         = 7'd53;
142 localparam int ADDR_CONE_HEIGHT       = 7'd54;
143 localparam int ADDR_CONE_COL_R        = 7'd55;

```

```

144     localparam int ADDR_CONE_COL_G          = 7'd56;
145     localparam int ADDR_CONE_COL_B          = 7'd57;
146     localparam int ADDR_REFLECT_SHIFT_CONE  = 7'd58;
147     localparam int ADDR_CONE_NORM_FACTOR    = 7'd59;
148     localparam int ADDR_COLOR_0             = 7'd64;
149 endpackage : raytracer_pkg
150
151 // fp_mul: (a*b) >> FRAC_BITS. Output registered (Quartus infers the DSP's
152 // hard output FF). Latency: 1 cycle.
153 module fp_mul #(
154     parameter FRAC_BITS = 13,
155     parameter WORD      = 27
156 )(
157     input  logic          clk,          // clock; output registered on posedge
158     input  logic signed [WORD-1:0] a,    // multiplicand (Q14.13)
159     input  logic signed [WORD-1:0] b,    // multiplier (Q14.13)
160     output logic signed [WORD-1:0] result // a*b, registered, 1-cycle latency
161 );
162     logic signed [2*WORD-1:0] full;
163     assign full = a * b;
164     always_ff @(posedge clk) begin
165         result <= full[WORD-1+FRAC_BITS:FRAC_BITS];
166     end
167 endmodule
168
169 // Asymmetric-width fp_mul: lets basis_compute use 14-bit px_c against
170 // 27-bit basis vectors so Quartus infers a 14x27 (1 DSP) instead of 27x27.
171 module fp_mul_narrow #(
172     parameter int FRAC_BITS = 13,
173     parameter int WA        = 27,
174     parameter int WB        = 14,
175     parameter int WOUT      = 27
176 )(
177     input  logic          clk,          // clock; output registered
178     input  logic signed [WA-1:0] a,      // wide operand (WA bits, Q14.FRAC_BITS)
179     input  logic signed [WB-1:0] b,      // narrow operand (WB bits, Q*.FRAC_BITS) - small range
180     output logic signed [WOUT-1:0] result // a*b in WOUT bits; 1-cycle latency
181 );
182     logic signed [WA+WB-1:0] full;
183     assign full = a * b;
184     always_ff @(posedge clk) begin
185         result <= full[WOUT-1+FRAC_BITS:FRAC_BITS];
186     end
187 endmodule
188
189 // fp_sqrt: digit-by-digit. Latency = (WORD+FRAC_BITS)/2 cycles (20 at Q14.13).
190 // Reference: https://projectf.io/posts/square-root-in-verilog/
191 module fp_sqrt #(
192     parameter FRAC_BITS = 13,
193     parameter WORD      = 27
194 )(
195     input  logic          clk,          // clock
196     input  logic          rst_n,        // active-low reset
197     input  logic          start,        // 1-cycle pulse: latch a, begin sqrt
198     input  logic signed [WORD-1:0] a,    // radicand (Q14.13); a<=0 → result=0
199     output logic signed [WORD-1:0] result, // √a (Q14.13), valid when done=1
200     output logic          done          // 1-cycle pulse 21c after start (start latch + ITER=20 iterations)
201 );
202     localparam ITER = (WORD + FRAC_BITS) >> 1;
203
204     logic [WORD-1:0] x, x_next;
205     logic [WORD-1:0] q, q_next;
206     logic [WORD+1:0] ac, ac_next;
207     logic [WORD+1:0] test_res;
208     logic [$clog2(ITER):0] i;
209     logic          busy;
210
211     always_comb begin
212         test_res = ac - {q, 2'b01};
213         if (test_res[WORD+1] == 0) begin
214             {ac_next, x_next} = {test_res[WORD-1:0], x, 2'b0};
215             q_next = {q[WORD-2:0], 1'b1};
216         end else begin
217             {ac_next, x_next} = {ac[WORD-1:0], x, 2'b0};

```

```

218         q_next = q << 1;
219     end
220 end
221
222 always_ff @(posedge clk or negedge rst_n) begin
223     if (!rst_n) begin
224         result <= '0;
225         done <= 1'b0;
226         busy <= 1'b0;
227         q <= '0;
228         ac <= '0;
229         x <= '0;
230         i <= '0;
231     end else begin
232         done <= 1'b0;
233         if (start) begin
234             if (a <= 0) begin
235                 result <= '0;
236                 done <= 1'b1;
237                 busy <= 1'b0;
238             end else begin
239                 busy <= 1'b1;
240                 i <= '0;
241                 q <= '0;
242                 {ac, x} <= {{WORD{1'b0}}, a, 2'b0};
243             end
244         end else if (busy) begin
245             if (i == ITER - 1) begin
246                 busy <= 1'b0;
247                 done <= 1'b1;
248                 result <= q_next;
249             end else begin
250                 i <= i + 1;
251                 x <= x_next;
252                 ac <= ac_next;
253                 q <= q_next;
254             end
255         end
256     end
257 end
258 endmodule
259
260 // fp_div: (a << FRAC_BITS) / b. Sequential restoring divider, WIDE+1c
261 // latency. Returns 0 on b==0.
262 module fp_div #(
263     parameter FRAC_BITS = 13,
264     parameter WORD = 27
265 )(
266     input logic clk, // clock
267     input logic rst_n, // active-low reset
268     input logic start, // 1-cycle pulse: latch a/b, begin divide
269     input logic signed [WORD-1:0] a, // dividend (Q14.13)
270     input logic signed [WORD-1:0] b, // divisor (Q14.13); b==0 → result=0
271     output logic signed [WORD-1:0] result, // a/b (Q14.13), valid when done=1
272     output logic done // 1-cycle pulse 41c after start
273 );
274     localparam WIDE = WORD + FRAC_BITS;
275
276     logic sign_out;
277     logic [WIDE-1:0] numer;
278     logic [WIDE-1:0] denom;
279     logic [WIDE-1:0] quotient;
280     logic [WIDE-1:0] remainder;
281     logic [WIDE-1:0] rem_next;
282     logic [$clog2(WIDE):0] count;
283
284     typedef enum logic [1:0] {
285         IDLE,
286         BUSY,
287         FINISH
288     } state_t;
289     state_t state;
290
291     always_comb begin

```

```

292     rem_next = {remainder[WIDE-2:0], numer[count-1]};
293 end
294
295 always_ff @(posedge clk or negedge rst_n) begin
296     if (!rst_n) begin
297         result    <= '0;
298         done      <= 1'b0;
299         state     <= IDLE;
300         numer     <= '0;
301         denom     <= '0;
302         quotient  <= '0;
303         remainder <= '0;
304         count     <= '0;
305         sign_out  <= 1'b0;
306     end else begin
307         done <= 1'b0;
308         case (state)
309             IDLE: begin
310                 if (start) begin
311                     if (b == '0) begin
312                         result <= '0;
313                         done  <= 1'b1;
314                     end else begin
315                         sign_out <= a[WORD-1] ^ b[WORD-1];
316                         numer    <= {(a[WORD-1] ? -a : a), {FRAC_BITS{1'b0}}};
317                         denom    <= {{FRAC_BITS{1'b0}}, (b[WORD-1] ? -b : b)};
318                         quotient <= '0;
319                         remainder <= '0;
320                         count    <= WIDE[$clog2(WIDE):0];
321                         state    <= BUSY;
322                     end
323                 end
324             end
325             BUSY: begin
326                 if (rem_next >= denom) begin
327                     remainder <= rem_next - denom;
328                     quotient[count-1] <= 1'b1;
329                 end else begin
330                     remainder <= rem_next;
331                     quotient[count-1] <= 1'b0;
332                 end
333                 count <= count - 1;
334                 if (count == 1)
335                     state <= FINISH;
336             end
337             FINISH: begin
338                 result <= sign_out ? -quotient[WORD-1:0] : quotient[WORD-1:0];
339                 done  <= 1'b1;
340                 state <= IDLE;
341             end
342             default: state <= IDLE;
343         endcase
344     end
345 end
346 endmodule
347
348 // fp_inv: 1/b via fp_div with numerator = ONE. Stages C/F/H share one
349 // reciprocal across 3 numerators - collapses 3 fp_divs to 1 fp_inv + 3 muls.
350 module fp_inv #(
351     parameter FRAC_BITS = 13,
352     parameter WORD      = 27
353 )(
354     input logic          clk,          // clock
355     input logic          rst_n,        // active-low reset
356     input logic          start,        // 1-cycle pulse: latch b, begin inversion
357     input logic signed [WORD-1:0] b,   // divisor (Q14.13); b==0 → result=0
358     output logic signed [WORD-1:0] result, // 1/b (Q14.13), valid when done=1
359     output logic          done         // 1-cycle pulse 41c after start
360 );
361     localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;
362     fp_div #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_div (
363         .clk(clk), .rst_n(rst_n), .start(start),
364         .a(ONE), .b(b), .result(result), .done(done)
365     );

```

```

366 endmodule
367
368
369 module rtl_dma_master
370     import raytracer_pkg::*;
371     #(
372         parameter int N          = 480,
373         parameter int WIDTH      = 480,
374         parameter int FRAME_BYTES = WIDTH * 360 * 4
375     )(
376         input logic          clk,          // clock
377         input logic          reset,        // active-HIGH sync reset
378         /* CSR slave bus (only DMA_CTRL writes are decoded here). */
379         input logic          chipselect,   // slave chip-select
380         input logic          write,        // slave write strobe
381         input logic [9:0]    address,      // slave word address
382         input logic [31:0]    writedata,   // slave write data (bit0=ENABLE, 1=CLEAR_ERR, 2=
CLEAR_DONE)
383         /* From frame FSM */
384         input logic          start_pulse,  // batch-start kick; clears dma_done_sticky
385         input logic          core_done,    // compute pipeline done; triggers dma_kick when enabled
386         input logic [$clog2(WIDTH)-1:0] pixel_x_reg, // pixel-x of current batch (for addr_burst computation)
387         input logic [8:0]    pixel_y_reg,  // pixel-y of current batch (row index)
388         input logic          bank_sel,     // compute's BRAM bank - DMA reads the other (~bank_sel)
389         input logic          write_slot,   // 0/1 = DDR3 frame ring slot for this frame
390         /* CSR-staged frame-buffer base address (written via the slave). */
391         input logic [31:0]    frame_buf_base_reg, // base DDR3 phys addr of 2-slot frame ring
392         /* BRAM read port (color_mem provides the data; we drive the addresses). */
393         output logic [$clog2(N)-1:0] dma_rd_addr_lo, // BRAM read addr for even-lane pixel (low 32 bits of
beat)
394         output logic [$clog2(N)-1:0] dma_rd_addr_hi, // BRAM read addr for odd-lane pixel (high 32 bits of
beat)
395         input logic [23:0]    dma_rd_data_lo, // even-lane pixel data (24-bit BGR from color_mem)
396         input logic [23:0]    dma_rd_data_hi, // odd-lane pixel data
397         output logic [31:0]    m_avm_address, // Avalon-MM master: burst base address (byte)
398         output logic [7:0]    m_avm_byteenable, // Avalon-MM master: byte enables (hardcoded 0xFF)
399         output logic          m_avm_write,     // Avalon-MM master: held high during 240-beat burst
400         output logic [63:0]    m_avm_writedata, // Avalon-MM master: {pad,hi_pixel,pad,lo_pixel}
401         output logic [7:0]    m_avm_burstcount, // Avalon-MM master: hardcoded 240 beats per burst
402         input logic          m_avm_waitrequest, // Avalon-MM master: bridge back-pressure (hold beat)
403         output logic          dma_busy,        // 1 while a burst is in flight; goes to DMA_STATUS bit0
404         output logic          dma_err,         // sticky error flag; goes to DMA_STATUS bit1
405         output logic          dma_done_sticky, // last burst done; cleared on next core_done or CSR
406         output logic          dma_bank_sel,    // BRAM read bank ( ~bank_sel, latched at burst start)
407         output logic [2:0]    dma_dbg_state,  // FSM state for DMA_STATUS [5:3]
408         output logic [7:0]    dma_dbg_beats   // beats remaining for DMA_STATUS [13:6]
409     );
410     typedef enum logic [2:0] {
411         DMA_IDLE,
412         DMA_PREFETCH,
413         DMA_WRITE,
414         DMA_COMMIT,
415         DMA_DONE
416     } dma_state_t;
417
418     dma_state_t          dma_state;
419     logic [31:0]          addr_burst;
420     logic [$clog2(N+1)-1:0] lane_idx;
421     logic [7:0]          beats_remaining;
422     logic [3:0]          commit_delay;
423     logic                dma_enable;
424     logic                dma_kick;
425
426     assign dma_dbg_state = dma_state;
427     assign dma_dbg_beats = beats_remaining;
428
429     // BRAM read addresses: pre-fetch the NEXT pair when accepting a beat;
430     // hold the CURRENT pair under waitrequest so writedata stays stable.
431     always_comb begin
432         if (dma_state == DMA_WRITE && !m_avm_waitrequest) begin
433             dma_rd_addr_lo = lane_idx[$clog2(N)-1:0] + ($clog2(N))'(2);
434             dma_rd_addr_hi = lane_idx[$clog2(N)-1:0] + ($clog2(N))'(3);
435         end else begin
436             dma_rd_addr_lo = lane_idx[$clog2(N)-1:0];

```

```

437         dma_rd_addr_hi = lane_idx[$clog2(N)-1:0] + ($clog2(N))'(1);
438     end
439 end
440
441 always_ff @(posedge clk or posedge reset) begin
442     if (reset) begin
443         dma_state      <= DMA_IDLE;
444         addr_burst     <= '0;
445         lane_idx       <= '0;
446         beats_remaining <= '0;
447         dma_busy       <= 1'b0;
448         dma_err        <= 1'b0;
449         dma_done_sticky <= 1'b0;
450         dma_bank_sel   <= 1'b0;
451         commit_delay   <= '0;
452         dma_enable     <= 1'b0;
453         dma_kick       <= 1'b0;
454     end else begin
455         dma_kick <= core_done & dma_enable;
456
457         // dma_done_sticky clears on either a new kick or on core_done.
458         // The core_done clear MUST lead dma_kick by one cycle -
459         // clearing on dma_kick instead races FRAME_DRAIN.
460         if (start_pulse) dma_done_sticky <= 1'b0;
461         if (core_done)   dma_done_sticky <= 1'b0;
462
463         if (chipselct && write && address == ADDR_DMA_CTRL) begin
464             dma_enable <= writedata[0];
465             if (writedata[1]) dma_err      <= 1'b0;
466             if (writedata[2]) dma_done_sticky <= 1'b0;
467         end
468
469         case (dma_state)
470             DMA_IDLE: if (dma_kick) begin
471                 dma_bank_sel   <= bank_sel;
472                 addr_burst     <= frame_buf_base_reg
473                     + (write_slot ? FRAME_BYTES[31:0] : 32'd0)
474                     + ((pixel_y_reg * WIDTH + pixel_x_reg) << 2);
475                 lane_idx       <= '0;
476                 beats_remaining <= 8'd240;
477                 dma_busy       <= 1'b1;
478                 dma_state      <= DMA_PREFETCH;
479             end
480
481             DMA_PREFETCH: dma_state <= DMA_WRITE;
482
483             DMA_WRITE: if (!m_avm_waitrequest) begin
484                 lane_idx       <= lane_idx + ($clog2(N+1))'(2);
485                 beats_remaining <= beats_remaining - 8'd1;
486                 if (beats_remaining == 8'd1) begin
487                     commit_delay <= 4'd15;
488                     dma_state    <= DMA_COMMIT;
489                 end
490             end
491
492             DMA_COMMIT: begin
493                 if (commit_delay == 4'd0) dma_state <= DMA_DONE;
494                 else
495                     commit_delay <= commit_delay - 4'd1;
496             end
497
498             DMA_DONE: begin
499                 dma_busy       <= 1'b0;
500                 dma_done_sticky <= 1'b1;
501                 dma_state      <= DMA_IDLE;
502             end
503
504             default: dma_state <= DMA_IDLE;
505         endcase
506     end
507 end
508
509 assign m_avm_address      = addr_burst;
510 assign m_avm_byteenable   = 8'hFF;
511 assign m_avm_writedata    = {8'b0, dma_rd_data_hi, 8'b0, dma_rd_data_lo};

```



```

511     assign m_avm_burstcount = 8'd240;
512     assign m_avm_write      = (dma_state == DMA_WRITE);
513 endmodule
514
515 // =====
516 // sun_threshold_compute - produces K_const = |light|^2 · 63/64, the threshold
517 // used by rtl_hit_resolve's sun-billboard test ((d·light)^2 K_const·|d|^2).
518 // Time-muxes one fp_mul across a 3-phase rotation (lx^2, ly^2, lz^2); the 63/64
519 // fold is x - (x>>>6) (no DSP). Output updates continuously as light_{x,y,z}
520 // change. Cold-reset value matches light = (0, 1, 0) so the sun is
521 // unreachable until SW writes a real light direction.
522 // =====
523 module sun_threshold_compute #(
524     parameter int FRAC_BITS = 13,
525     parameter int WORD      = 27
526 )(
527     input logic          clk,
528     input logic          reset,
529     input logic signed [WORD-1:0] light_x,
530     input logic signed [WORD-1:0] light_y,
531     input logic signed [WORD-1:0] light_z,
532     output logic signed [WORD-1:0] K_const
533 );
534     localparam signed [WORD-1:0] ONE_FX = 1 <<< FRAC_BITS;
535
536     logic [1:0] kc_phase;
537     always_ff @(posedge clk or posedge reset) begin
538         if (reset)          kc_phase <= 2'd0;
539         else if (kc_phase == 2'd2) kc_phase <= 2'd0;
540         else                kc_phase <= kc_phase + 2'd1;
541     end
542
543     logic signed [WORD-1:0] kc_mul_a, kc_mul_b;
544     always_comb begin
545         case (kc_phase)
546             2'd0: begin kc_mul_a = light_x; kc_mul_b = light_x; end
547             2'd1: begin kc_mul_a = light_y; kc_mul_b = light_y; end
548             default: begin kc_mul_a = light_z; kc_mul_b = light_z; end
549         endcase
550     end
551
552     logic signed [WORD-1:0] kc_mul_result;
553     fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_l_sq (
554         .clk(clk), .a(kc_mul_a), .b(kc_mul_b), .result(kc_mul_result)
555     );
556
557     logic signed [WORD-1:0] lx_sq_r, ly_sq_r, lz_sq_r;
558     always_ff @(posedge clk or posedge reset) begin
559         if (reset) begin
560             lx_sq_r <= '0;
561             ly_sq_r <= ONE_FX;
562             lz_sq_r <= '0;
563         end else begin
564             case (kc_phase)
565                 2'd1: lx_sq_r <= kc_mul_result;
566                 2'd2: ly_sq_r <= kc_mul_result;
567                 default: lz_sq_r <= kc_mul_result;
568             endcase
569         end
570     end
571
572     logic signed [WORD-1:0] light_sq_w;
573     assign light_sq_w = lx_sq_r + ly_sq_r + lz_sq_r;
574
575     // K_const = light_sq - (light_sq >>> 6) light_sq · 63/64.
576     always_ff @(posedge clk or posedge reset) begin
577         if (reset) K_const <= ONE_FX - (ONE_FX >>> 6);
578         else      K_const <= light_sq_w - (light_sq_w >>> 6);
579     end
580 endmodule
581
582 module raytracer_batch_mm
583     import raytracer_pkg::*;
584     #(

```

```

585     parameter int N          = 480,
586     parameter int FRAC_BITS = 13,
587     parameter int WORD      = 27,
588     parameter int WIDTH     = 480,
589     parameter int HEIGHT    = 360
590 ) (
591     input logic          clk,          // divclk (125 MHz target)
592     input logic          reset,        // active-HIGH sync reset (inverted internally to rst_n)
593
594     input logic          chipselect,   // slave selected
595     input logic          read,         // master asserts to begin a read transaction
596     input logic          write,        // master asserts to begin a write transaction
597     input logic [9:0]    address,      // word address (1024-word window)
598     input logic [31:0]   writedata,    // write data; HW masks each CSR's relevant bits
599     output logic [31:0]  readdata,     // read data; fixed 2-cycle latency from read assertion
600
601     // Avalon-MM master → hps_0.f2h_axi_slave (qsys-bridged to AXI3)
602     output logic [31:0]  m_avm_address, // burst base byte address (DDR3 phys addr)
603     output logic [7:0]   m_avm_byteenable, // byte enables (hardcoded 0xFF)
604     output logic         m_avm_write,    // held high during DMA burst
605     output logic [63:0]  m_avm_writedata, // 64-bit beat = 2 packed BGRX pixels
606     output logic [7:0]   m_avm_burstcount, // hardcoded 240 beats per burst
607     input logic         m_avm_waitrequest // back-pressure from f2h bridge
608 );
609
610 /*
611  * Register map (word offsets - addresses live in raytracer_pkg as ADDR_*).
612  * SW writes land directly on the live regs; the per-frame loop is
613  * WAIT_FRAME → write all CSRs → FRAME_START so writes never tear an
614  * in-flight frame. Most fields are signed Q FRAC_BITS.
615  *
616  * 0    CONTROL          bit2 frame_start, bit3 clear_frame_done
617  * 1    STATUS           bit2 frame_done, bit3 frame_running, bit4 write_slot
618  * 4-6   LIGHT_{X,Y,Z}   light vector (also drives sun via |light|.63/64)
619  * 7-8   R{0,1}_SQ       sphere radii2 (SW pre-squares r)
620  * 9     DMA_CTRL        bit0 enable, bit1 clear_err, bit2 clear_done
621  * 10    DMA_STATUS      [0] busy, [1] err, [2] done, [5:3] state, [13:6] beats
622  * 11    reserved (formerly LIGHT_COMMIT; not needed post-collapse)
623  * 15    FRAME_BUF_BASE  DDR3 phys addr; HW writes pix_y·W+pix_x at slot offset
624  * 16-18 SC1_{X,Y,Z}     sphere 1 (red) center
625  * 19    MAX_DEPTH       3-bit recursion depth (default 3; 0 disables refl)
626  * 20-21 REFLECT_SHIFT_{0,1} 5-bit per-sphere bounce attenuation (>>SHIFT)
627  * 22-23 CAM_{X,Z}       camera origin (cam_y locked at 1.5)
628  * 24-31 RIGHT/UP/FWD_*   precomputed basis from SW (right.y=0, no roll)
629  * 32-34 SC0_{X,Y,Z}     sphere 0 (blue) center
630  * 35-40 PC{1,2}_{R,G,B} floor checker colors
631  * 41-46 SCOL{0,1}_{R,G,B} per-sphere base colors
632  * 47    FLOOR_MODE       00=1m checker, 01=Sierpinski, 10=stripes, 11=0.5m
633  * 48-49 FLOOR_LIM_{X,Z} bounded plane half-extents (default max-pos)
634  * 50-52 CONE_{X,Y,Z}    cone apex (default origin)
635  * 53    CONE_K_SQ        tan2(half-angle) Q14.13 (default 1.0 = 45°)
636  * 54    CONE_HEIGHT      finite cone height (default 0 = disabled)
637  * 55-57 CONE_COL_{R,G,B} cone surface color (default mid-grey)
638  * 58    REFLECT_SHIFT_CONE 5-bit cone bounce attenuation (default 0 matte)
639  * 59-63 reserved (former sun CSRs; sun is derived from light/|light| now)
640  * 64..  COLOR_0..N       BRAM mirror, packed {00,R,G,B} (DMA is the
641  *                          primary readback path)
642  */
643
644 logic [$clog2(WIDTH)-1:0] pixel_x_reg;
645 logic [$clog2(HEIGHT)-1:0] pixel_y_reg;
646 logic signed [WORD-1:0] light_x_reg;
647 logic signed [WORD-1:0] light_y_reg;
648 logic signed [WORD-1:0] light_z_reg;
649
650 // Reflection / recursion control
651 localparam int MAX_DEPTH_W = 3;
652 localparam int REFLECT_SHIFT_W = 5;
653 localparam int DEFAULT_MAX_DEPTH = 3;
654 localparam int DEFAULT_REFLECT_SHIFT_0 = 2; // 25%
655 localparam int DEFAULT_REFLECT_SHIFT_1 = 1; // 50%
656 logic [MAX_DEPTH_W-1:0] max_depth_reg;
657
658 // Camera state (cam_y locked in HW at CAM_Y_FIXED; right.y hardcoded 0).

```

```

659 logic signed [WORD-1:0] cam_x_reg, cam_z_reg;
660 logic signed [WORD-1:0] right_x_reg, right_z_reg;
661 logic signed [WORD-1:0] up_x_reg, up_y_reg, up_z_reg;
662 logic signed [WORD-1:0] fwd_x_reg, fwd_y_reg, fwd_z_reg;
663
664 // Scene primitives - struct-typed CSR storage. Each field below maps
665 // 1:1 to a CSR address; the case-arms inside the CSR write block
666 // assign to sc0_reg.x, sc0_reg.color.r, sc0_reg.reflect_shift, etc.
667 sphere_t sc0_reg, sc1_reg;
668 cone_t cone_reg;
669 floor_t floor_reg;
670
671 logic start_pulse;
672
673 // Frame FSM: →KICKWAIT(core_done)→KICK loop, then DRAIN(dma_done) →
674 // DONE_PULSE
675 typedef enum logic [2:0] {
676     FRAME_IDLE,
677     FRAME_KICK,
678     FRAME_WAIT,
679     FRAME_DRAIN,
680     FRAME_DONE_PULSE
681 } frame_state_t;
682
683 frame_state_t frame_state;
684 logic [$clog2(WIDTH)-1:0] pix_x_gen;
685 logic [$clog2(HEIGHT+1)-1:0] pix_y_gen; // 0..HEIGHT (post-last-advance)
686 logic frame_running_reg;
687 logic frame_done_sticky;
688 logic [31:0] frame_buf_base_reg;
689
690 // Frame-level double buffer. Driver allocates 2·FRAME_BYTES at
691 // FRAME_BUF_BASE; HW alternates slots so SW reads one while HW writes
692 // the other. write_slot toggles only at FRAME_DONE_PULSE.
693 localparam int FRAME_BYTES = WIDTH * HEIGHT * 4;
694 logic write_slot;
695
696 logic core_done;
697 logic core_wr_en;
698 logic [$clog2(N)-1:0] core_wr_addr;
699 logic [23:0] core_wr_data;
700
701 logic dma_busy;
702 logic dma_err;
703 logic dma_done_sticky;
704 logic dma_bank_sel;
705 logic [2:0] dma_dbg_state;
706 logic [7:0] dma_dbg_beats;
707 logic [$clog2(N)-1:0] dma_rd_addr_lo, dma_rd_addr_hi;
708 logic [23:0] dma_rd_data_lo, dma_rd_data_hi;
709
710 localparam signed [WORD-1:0] ONE_FX = 1 <<< FRAC_BITS;
711 localparam signed [WORD-1:0] DEFAULT_SCO_X = 0;
712 localparam signed [WORD-1:0] DEFAULT_SCO_Y = ONE_FX;
713 localparam signed [WORD-1:0] DEFAULT_SCO_Z = 0;
714 localparam signed [WORD-1:0] DEFAULT_RO_SQ = ONE_FX;
715 localparam signed [WORD-1:0] DEFAULT_SC1_X = (5 * ONE_FX) / 2;
716 localparam signed [WORD-1:0] DEFAULT_SC1_Y = ONE_FX;
717 localparam signed [WORD-1:0] DEFAULT_SC1_Z = 0;
718 localparam signed [WORD-1:0] DEFAULT_R1_SQ = ONE_FX;
719
720 localparam signed [WORD-1:0] CAM_Y_FIXED = (3 * ONE_FX) / 2;
721 localparam signed [WORD-1:0] DEFAULT_CAM_X = '0;
722 localparam signed [WORD-1:0] DEFAULT_CAM_Z = -8 * ONE_FX;
723 localparam signed [WORD-1:0] DEFAULT_RIGHT_X = -ONE_FX;
724 localparam signed [WORD-1:0] DEFAULT_RIGHT_Z = '0;
725 localparam signed [WORD-1:0] DEFAULT_UP_X = '0;
726 localparam signed [WORD-1:0] DEFAULT_UP_Y = ONE_FX;
727 localparam signed [WORD-1:0] DEFAULT_UP_Z = '0;
728 localparam signed [WORD-1:0] DEFAULT_FWD_X = '0;
729 localparam signed [WORD-1:0] DEFAULT_FWD_Y = '0;
730 localparam signed [WORD-1:0] DEFAULT_FWD_Z = ONE_FX;
731
732 localparam signed [WORD-1:0] DEFAULT_PC1 = (9 * ONE_FX) / 10;

```

```

733 localparam signed [WORD-1:0] DEFAULT_PC2      = (4 * ONE_FX) / 10;
734 localparam signed [WORD-1:0] DEFAULT_SCOLO_R  = (2 * ONE_FX) / 10;
735 localparam signed [WORD-1:0] DEFAULT_SCOLO_G  = (3 * ONE_FX) / 10;
736 localparam signed [WORD-1:0] DEFAULT_SCOLO_B  = (8 * ONE_FX) / 10;
737 localparam signed [WORD-1:0] DEFAULT_SCOL1_R  = (8 * ONE_FX) / 10;
738 localparam signed [WORD-1:0] DEFAULT_SCOL1_G  = (3 * ONE_FX) / 10;
739 localparam signed [WORD-1:0] DEFAULT_SCOL1_B  = (2 * ONE_FX) / 10;
740 localparam logic [1:0]      DEFAULT_FLOOR_MODE = 2'b00;
741 localparam signed [WORD-1:0] DEFAULT_FLOOR_LIM = {1'b0, {(WORD-1){1'b1}}};
742
743 // Sun-billboard threshold K_const = |light|^2 * 63/64. Time-muxed
744 // single-DSP compute lives in sun_threshold_compute.
745 logic signed [WORD-1:0] K_const_r;
746 sun_threshold_compute #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sun_threshold (
747     .clk      (clk),
748     .reset    (reset),
749     .light_x  (light_x_reg),
750     .light_y  (light_y_reg),
751     .light_z  (light_z_reg),
752     .K_const  (K_const_r)
753 );
754
755 // Bundle the four primitive registers into a single scene_t for the
756 // pipeline. Pure struct-literal - Quartus prunes unused fields at each
757 // consumer stage.
758 scene_t scene_active;
759 assign scene_active = '{s0:    sc0_reg,
760                        s1:    sc1_reg,
761                        cone:   cone_reg,
762                        floor:  floor_reg};
763
764 rtl_batch_pipe #(
765     .K      (4),
766     .BATCH  (N),
767     .FRAC_BITS (FRAC_BITS),
768     .WORD    (WORD),
769     .WIDTH   (WIDTH),
770     .HEIGHT  (HEIGHT)
771 ) core_inst (
772     .clk      (clk),
773     .rst_n    (~reset),
774     .start    (start_pulse),
775     .pixel_x  (pixel_x_reg),
776     .pixel_y  (pixel_y_reg),
777     .light_x  (light_x_reg),
778     .light_y  (light_y_reg),
779     .light_z  (light_z_reg),
780     .scene    (scene_active),
781     .max_depth (max_depth_reg),
782     .cam_x    (cam_x_reg),
783     .cam_y    (CAM_Y_FIXED),
784     .cam_z    (cam_z_reg),
785     .right_x  (right_x_reg),
786     .right_z  (right_z_reg),
787     .up_x     (up_x_reg),
788     .up_y     (up_y_reg),
789     .up_z     (up_z_reg),
790     .fwd_x    (fwd_x_reg),
791     .fwd_y    (fwd_y_reg),
792     .fwd_z    (fwd_z_reg),
793     .K_const  (K_const_r),
794     .done     (core_done),
795     .wr_en    (core_wr_en),
796     .wr_addr  (core_wr_addr),
797     .wr_data  (core_wr_data)
798 );
799
800 logic      bank_sel;
801 localparam int COLOR_MEM_DEPTH = 1 << ($clog2(N) + 1);
802 (* ramstyle = "no_rw_check, M10K" *)
803 logic [23:0] color_mem [0:COLOR_MEM_DEPTH-1];
804
805 always_ff @(posedge clk or posedge reset) begin
806     if (reset) bank_sel <= 1'b0;

```

```

807     else if (start_pulse) bank_sel <= ~bank_sel;
808 end
809
810 always_ff @(posedge clk) begin
811     if (core_wr_en)
812         color_mem[{bank_sel, core_wr_addr}] <= core_wr_data;
813 end
814
815 // =====
816 // CSR data-register bank - direct-mapped Avalon-MM slave writes.
817 // Each register is one FF; writes land at the matching address. Writes
818 // to ADDR_CONTROL (frame_start / clear_done) and ADDR_DMA_CTRL (DMA
819 // enable) are NOT data writes - they live in the Frame FSM block
820 // below and in rtl_dma_master respectively.
821 // =====
822 always_ff @(posedge clk or posedge reset) begin
823     if (reset) begin
824         light_x_reg    <= '0;
825         // light_y resets to ONE_FX so |light|^2 = 1 and K_const 63/64.
826         // Without this, K_const = 0 and the sun overruns the sky until
827         // SW writes a real light direction.
828         light_y_reg    <= ONE_FX;
829         light_z_reg    <= '0;
830         max_depth_reg  <= DEFAULT_MAX_DEPTH[MAX_DEPTH_W-1:0];
831         cam_x_reg      <= DEFAULT_CAM_X;
832         cam_z_reg      <= DEFAULT_CAM_Z;
833         right_x_reg    <= DEFAULT_RIGHT_X;
834         right_z_reg    <= DEFAULT_RIGHT_Z;
835         up_x_reg       <= DEFAULT_UP_X;
836         up_y_reg       <= DEFAULT_UP_Y;
837         up_z_reg       <= DEFAULT_UP_Z;
838         fwd_x_reg      <= DEFAULT_FWD_X;
839         fwd_y_reg      <= DEFAULT_FWD_Y;
840         fwd_z_reg      <= DEFAULT_FWD_Z;
841         // Sphere 0 (blue, 25% reflective by default)
842         sc0_reg <= '{x:  DEFAULT_SCO_X,
843                    y:  DEFAULT_SCO_Y,
844                    z:  DEFAULT_SCO_Z,
845                    r_sq: DEFAULT_RO_SQ,
846                    color: '{r: DEFAULT_SCOLO_R, g: DEFAULT_SCOLO_G, b: DEFAULT_SCOLO_B},
847                    reflect_shift: DEFAULT_REFLECT_SHIFT_0[REFLECT_SHIFT_W-1:0]};
848         // Sphere 1 (red, 50% reflective by default)
849         sc1_reg <= '{x:  DEFAULT_SC1_X,
850                    y:  DEFAULT_SC1_Y,
851                    z:  DEFAULT_SC1_Z,
852                    r_sq: DEFAULT_R1_SQ,
853                    color: '{r: DEFAULT_SCOL1_R, g: DEFAULT_SCOL1_G, b: DEFAULT_SCOL1_B},
854                    reflect_shift: DEFAULT_REFLECT_SHIFT_1[REFLECT_SHIFT_W-1:0]};
855         // Cone - height=0 disables (slab empty); SW writes a real height
856         // to enable. Default mid-grey, matte, k^2=1.0 (45° half-angle).
857         cone_reg <= '{apex_x: '0, apex_y: '0, apex_z: '0,
858                    k_sq: 27'sd8192,           // Q14.13: 1.0
859                    height: '0,
860                    norm_factor: 27'sd11585,    // sqrt(2)
861                    color: '{r: 27'sd4096, g: 27'sd4096, b: 27'sd4096}, // 0.5,0.5,0.5
862                    reflect_shift: '0};
863         // Floor (checker, max half-extents by default)
864         floor_reg <= '{pc1: '{r: DEFAULT_PC1, g: DEFAULT_PC1, b: DEFAULT_PC1},
865                    pc2: '{r: DEFAULT_PC2, g: DEFAULT_PC2, b: DEFAULT_PC2},
866                    mode: DEFAULT_FLOOR_MODE,
867                    lim_x: DEFAULT_FLOOR_LIM,
868                    lim_z: DEFAULT_FLOOR_LIM};
869         frame_buf_base_reg <= '0;
870     end else if (chipselct && write) begin
871         case (address)
872             ADDR_LIGHT_X:    light_x_reg <= writedata[WORD-1:0];
873             ADDR_LIGHT_Y:    light_y_reg <= writedata[WORD-1:0];
874             ADDR_LIGHT_Z:    light_z_reg <= writedata[WORD-1:0];
875             ADDR_FRAME_BUF_BASE: frame_buf_base_reg <= writedata;
876             ADDR_MAX_DEPTH:  max_depth_reg <= writedata[MAX_DEPTH_W-1:0];
877             ADDR_CAM_X:      cam_x_reg    <= writedata[WORD-1:0];
878             ADDR_CAM_Z:      cam_z_reg    <= writedata[WORD-1:0];
879             ADDR_RIGHT_X:    right_x_reg  <= writedata[WORD-1:0];
880             ADDR_RIGHT_Z:    right_z_reg  <= writedata[WORD-1:0];

```

```

881     ADDR_UP_X:      up_x_reg      <= writedata[WORD-1:0];
882     ADDR_UP_Y:      up_y_reg      <= writedata[WORD-1:0];
883     ADDR_UP_Z:      up_z_reg      <= writedata[WORD-1:0];
884     ADDR_FWD_X:      fwd_x_reg     <= writedata[WORD-1:0];
885     ADDR_FWD_Y:      fwd_y_reg     <= writedata[WORD-1:0];
886     ADDR_FWD_Z:      fwd_z_reg     <= writedata[WORD-1:0];
887     // Sphere 0
888     ADDR_SCO_X:      sc0_reg.x      <= writedata[WORD-1:0];
889     ADDR_SCO_Y:      sc0_reg.y      <= writedata[WORD-1:0];
890     ADDR_SCO_Z:      sc0_reg.z      <= writedata[WORD-1:0];
891     ADDR_RO_SQ:      sc0_reg.r_sq   <= writedata[WORD-1:0];
892     ADDR_SCOLO_R:    sc0_reg.color.r <= writedata[WORD-1:0];
893     ADDR_SCOLO_G:    sc0_reg.color.g <= writedata[WORD-1:0];
894     ADDR_SCOLO_B:    sc0_reg.color.b <= writedata[WORD-1:0];
895     ADDR_REFLECT_SHIFT_0: sc0_reg.reflect_shift <= writedata[REFLECT_SHIFT_W-1:0];
896     // Sphere 1
897     ADDR_SC1_X:      sc1_reg.x      <= writedata[WORD-1:0];
898     ADDR_SC1_Y:      sc1_reg.y      <= writedata[WORD-1:0];
899     ADDR_SC1_Z:      sc1_reg.z      <= writedata[WORD-1:0];
900     ADDR_R1_SQ:      sc1_reg.r_sq   <= writedata[WORD-1:0];
901     ADDR_SCOL1_R:    sc1_reg.color.r <= writedata[WORD-1:0];
902     ADDR_SCOL1_G:    sc1_reg.color.g <= writedata[WORD-1:0];
903     ADDR_SCOL1_B:    sc1_reg.color.b <= writedata[WORD-1:0];
904     ADDR_REFLECT_SHIFT_1: sc1_reg.reflect_shift <= writedata[REFLECT_SHIFT_W-1:0];
905     // Cone
906     ADDR_CONE_X:      cone_reg.apex_x <= writedata[WORD-1:0];
907     ADDR_CONE_Y:      cone_reg.apex_y <= writedata[WORD-1:0];
908     ADDR_CONE_Z:      cone_reg.apex_z <= writedata[WORD-1:0];
909     ADDR_CONE_K_SQ:    cone_reg.k_sq <= writedata[WORD-1:0];
910     ADDR_CONE_HEIGHT: cone_reg.height <= writedata[WORD-1:0];
911     ADDR_CONE_NORM_FACTOR: cone_reg.norm_factor <= writedata[WORD-1:0];
912     ADDR_CONE_COL_R:    cone_reg.color.r <= writedata[WORD-1:0];
913     ADDR_CONE_COL_G:    cone_reg.color.g <= writedata[WORD-1:0];
914     ADDR_CONE_COL_B:    cone_reg.color.b <= writedata[WORD-1:0];
915     ADDR_REFLECT_SHIFT_CONE: cone_reg.reflect_shift <= writedata[REFLECT_SHIFT_W-1:0];
916     // Floor
917     ADDR_PC1_R:      floor_reg.pc1.r <= writedata[WORD-1:0];
918     ADDR_PC1_G:      floor_reg.pc1.g <= writedata[WORD-1:0];
919     ADDR_PC1_B:      floor_reg.pc1.b <= writedata[WORD-1:0];
920     ADDR_PC2_R:      floor_reg.pc2.r <= writedata[WORD-1:0];
921     ADDR_PC2_G:      floor_reg.pc2.g <= writedata[WORD-1:0];
922     ADDR_PC2_B:      floor_reg.pc2.b <= writedata[WORD-1:0];
923     ADDR_FLOOR_MODE:  floor_reg.mode <= writedata[1:0];
924     ADDR_FLOOR_LIM_X:  floor_reg.lim_x <= writedata[WORD-1:0];
925     ADDR_FLOOR_LIM_Z:  floor_reg.lim_z <= writedata[WORD-1:0];
926     default: begin end
927 endcase
928 end
929 end
930
931 // =====
932 // Frame FSM - chains per-batch →→KICKWAITDRAINDONE_PULSE for an entire
933 // frame on a single SW kick. ADDR_CONTROL writes (frame_start /
934 // clear_done) are handled here; they mutate FSM state, not data regs.
935 // write_slot flips at FRAME_DONE_PULSE for frame-level double buffering.
936 // =====
937 always_ff @(posedge clk or posedge reset) begin
938     if (reset) begin
939         pixel_x_reg <= '0;
940         pixel_y_reg <= '0;
941         start_pulse <= 1'b0;
942         frame_state <= FRAME_IDLE;
943         pix_x_gen <= '0;
944         pix_y_gen <= '0;
945         frame_running_reg <= 1'b0;
946         frame_done_sticky <= 1'b0;
947         write_slot <= 1'b0;
948     end else begin
949         start_pulse <= 1'b0;
950
951         // ADDR_CONTROL writes: bit3 clears frame_done_sticky, bit2 kicks
952         // a new frame. Handled here (not in the CSR block) because every
953         // signal touched is FSM state.
954         if (chipselct && write && address == ADDR_CONTROL) begin

```

```

955         if (writedata[3]) frame_done_sticky <= 1'b0;
956     if (writedata[2]) begin
957         pix_x_gen         <= '0;
958         pix_y_gen         <= '0;
959         frame_running_reg <= 1'b1;
960         frame_done_sticky <= 1'b0;
961         frame_state       <= FRAME_KICK;
962     end
963 end
964
965 case (frame_state)
966     FRAME_IDLE: begin
967         /* wait for SW to write CONTROL[2] (handled above) */
968     end
969     FRAME_KICK: begin
970         // Latch this batch's pix coords (pre-edge - pix_gen
971         // holds the NEXT batch); fire start_pulse.
972         pixel_x_reg <= pix_x_gen;
973         pixel_y_reg <= pix_y_gen[$clog2(HEIGHT)-1:0];
974         start_pulse <= 1'b1;
975
976         if (pix_x_gen + N >= WIDTH) begin
977             pix_x_gen <= '0;
978             pix_y_gen <= pix_y_gen + 1;
979         end else begin
980             pix_x_gen <= pix_x_gen + N;
981         end
982
983         frame_state <= FRAME_WAIT;
984     end
985     // Advance on core_done (NOT dma_done_sticky) so batch N's
986     // DMA overlaps with batch N+1's compute.
987     FRAME_WAIT: if (core_done) begin
988         if (pix_y_gen >= HEIGHT)
989             frame_state <= FRAME_DRAIN;
990         else
991             frame_state <= FRAME_KICK;
992     end
993     FRAME_DRAIN: if (dma_done_sticky)
994         frame_state <= FRAME_DONE_PULSE;
995     FRAME_DONE_PULSE: begin
996         frame_running_reg <= 1'b0;
997         frame_done_sticky <= 1'b1;
998         // Flip write_slot AFTER the frame's DMA committed so the
999         // next frame writes the slot SW just released.
1000         write_slot         <= ~write_slot;
1001         frame_state         <= FRAME_IDLE;
1002     end
1003     default: frame_state <= FRAME_IDLE;
1004 endcase
1005 end
1006
1007 // Slave read path - single-beat, fixed readLatency=1. address_r
1008 // absorbs the qsys-side IC route; the 3-way case mux drives readdata
1009 // combinational within the same period.
1010 logic [9:0] address_r;
1011 always_ff @(posedge clk or posedge reset) begin
1012     if (reset) address_r <= '0;
1013     else       address_r <= address;
1014 end
1015
1016 logic [31:0] ctrl_rdata;
1017 always_comb begin
1018     case (address_r)
1019         ADDR_STATUS: ctrl_rdata = {27'b0, write_slot,
1020                                     frame_running_reg,
1021                                     frame_done_sticky,
1022                                     2'b00};
1023         ADDR_DMA_STATUS: ctrl_rdata = {18'b0, dma_dbg_beats,
1024                                         dma_dbg_state, dma_done_sticky,
1025                                         dma_err, dma_busy};
1026         ADDR_FRAME_BUF_BASE: ctrl_rdata = frame_buf_base_reg;
1027         default: ctrl_rdata = 32'b0;

```

```

1029         endcase
1030     end
1031
1032     assign readdata = ctrl_rdata;
1033
1034     always_ff @(posedge clk) begin
1035         dma_rd_data_lo <= color_mem[{dma_bank_sel, dma_rd_addr_lo}];
1036     end
1037     always_ff @(posedge clk) begin
1038         dma_rd_data_hi <= color_mem[{dma_bank_sel, dma_rd_addr_hi}];
1039     end
1040
1041     rtl_dma_master #(
1042         .N            (N),
1043         .WIDTH        (WIDTH),
1044         .FRAME_BYTES  (FRAME_BYTES)
1045     ) u_dma (
1046         .clk            (clk),
1047         .reset          (reset),
1048         .chipselect     (chipselect),
1049         .write          (write),
1050         .address        (address),
1051         .writedata      (writedata),
1052         .start_pulse    (start_pulse),
1053         .core_done      (core_done),
1054         .pixel_x_reg    (pixel_x_reg),
1055         .pixel_y_reg    (pixel_y_reg),
1056         .bank_sel       (bank_sel),
1057         .write_slot     (write_slot),
1058         .frame_buf_base_reg (frame_buf_base_reg),
1059         .dma_rd_addr_lo (dma_rd_addr_lo),
1060         .dma_rd_addr_hi (dma_rd_addr_hi),
1061         .dma_rd_data_lo (dma_rd_data_lo),
1062         .dma_rd_data_hi (dma_rd_data_hi),
1063         .m_avm_address  (m_avm_address),
1064         .m_avm_byteenable (m_avm_byteenable),
1065         .m_avm_write    (m_avm_write),
1066         .m_avm_writedata (m_avm_writedata),
1067         .m_avm_burstcount (m_avm_burstcount),
1068         .m_avm_waitrequest (m_avm_waitrequest),
1069         .dma_busy       (dma_busy),
1070         .dma_err        (dma_err),
1071         .dma_done_sticky (dma_done_sticky),
1072         .dma_bank_sel   (dma_bank_sel),
1073         .dma_dbg_state  (dma_dbg_state),
1074         .dma_dbg_beats  (dma_dbg_beats)
1075     );
1076
1077 endmodule
1078
1079 // Stage A - RAY GEN: select raw direction (basis_compute output for primaries,
1080 // latched parent reflection for spawns) and compute |raw|^2 for stage C's
1081 // quadratic `a` coefficient. Also precomputes a_cone = |d|^2 - (1+k^2)·D.y^2 so
1082 // that u_inv_a_cone can kick at stage C's t=0 alongside u_inv_a/u_inv_pd -
1083 // see docs/cone_fp_inv_timing.html.
1084 module rtl_ray_gen
1085     import raytracer_pkg::*;
1086     #(
1087         parameter int FRAC_BITS = 13,
1088         parameter int WORD      = 27,
1089         parameter int WIDTH     = 480,
1090         parameter int HEIGHT    = 360,
1091         parameter int TAG_W     = 7
1092     )(
1093         input logic          clk, rst_n,                // clock, async neg-edge reset
1094         input logic [5:0]    tick,                      // pipe-local tick counter (0..II-1)
1095         input pipe_baggage_t in_bag,                    // {sv, tag, meta} - latched at tick=0
1096         // Reflection-ray override path: in_use_override=1 selects dir*_lat
1097         input logic          in_use_override,           // 1=spawn (use in_dir), 0=primary (use basis
output)
1098         input logic signed [WORD-1:0] in_orig_x, in_orig_y, in_orig_z, // ray origin: camera for primaries, parent
hit for spawns
1099         input logic signed [WORD-1:0] in_dir_x, in_dir_y, in_dir_z,    // override direction (parent reflection);
used iff in_use_override=1

```



```

1100 // Shared basis_compute output from rtl_batch_pipe - settles by issue+5.
1101 input logic signed [WORD-1:0] in_basis_raw_x, in_basis_raw_y, in_basis_raw_z, // primary raw direction (
right_px + up_py + fwd)
1102 input scene_t in_scene, // scene CSR relay; only in_scene.cone.k_sq is
read here
1103 output logic signed [WORD-1:0] raygen_raw_x, raygen_raw_y, raygen_raw_z, // captured raw direction → Stage
C
1104 output logic signed [WORD-1:0] raygen_mag_sq, // |d|^2, used by sphere quadratic + sun
billboard
1105 output logic signed [WORD-1:0] raygen_a_cone, // |d|^2 - (1+k^2)·D.y^2 - feeds u_inv_a_cone at C-
t=0
1106 output logic signed [WORD-1:0] raygen_ox, raygen_oy, raygen_oz, // registered origin forwarded to Stage C
1107 output pipe_baggage_t raygen_bag // baggage forwarded to Stage C
1108 );
1109 localparam signed [WORD-1:0] ONE_FX = 1 <<< FRAC_BITS;
1110
1111 pipe_baggage_t bag_r;
1112 logic use_override_r;
1113 logic signed [WORD-1:0] orig_x_r, orig_y_r, orig_z_r;
1114 logic signed [WORD-1:0] dir_x_lat, dir_y_lat, dir_z_lat;
1115 logic signed [WORD-1:0] raw_x_r, raw_y_r, raw_z_r;
1116 logic signed [WORD-1:0] prod_x_r, prod_y_r, prod_z_r;
1117 logic signed [WORD-1:0] mag_sq_r;
1118 logic signed [WORD-1:0] one_plus_k_sq_r;
1119 logic signed [WORD-1:0] kp1_dy_sq_r, a_cone_r;
1120
1121 logic signed [WORD-1:0] mul_a, mul_b, mul_result;
1122 fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul (.clk(clk), .a(mul_a), .b(mul_b), .result(mul_result));
1123
1124 always_comb begin
1125 mul_a = '0; mul_b = '0;
1126 case (tick)
1127 6'd8: begin mul_a = raw_x_r; mul_b = raw_x_r; end
1128 6'd9: begin mul_a = raw_y_r; mul_b = raw_y_r; end
1129 6'd10: begin mul_a = raw_z_r; mul_b = raw_z_r; end
1130 6'd11: begin mul_a = prod_y_r; mul_b = one_plus_k_sq_r; end // kp1_dy_sq → cap t=12
1131 default: begin end
1132 endcase
1133 end
1134
1135 always_ff @(posedge clk or negedge rst_n) begin
1136 if (!rst_n) begin
1137 bag_r <= '0;
1138 use_override_r <= 1'b0;
1139 orig_x_r <= '0; orig_y_r <= '0; orig_z_r <= '0;
1140 dir_x_lat <= '0; dir_y_lat <= '0; dir_z_lat <= '0;
1141 raw_x_r <= '0; raw_y_r <= '0; raw_z_r <= '0;
1142 prod_x_r <= '0; prod_y_r <= '0; prod_z_r <= '0;
1143 mag_sq_r <= '0;
1144 one_plus_k_sq_r <= '0;
1145 kp1_dy_sq_r <= '0;
1146 a_cone_r <= '0;
1147 end else begin
1148 one_plus_k_sq_r <= in_scene.cone.k_sq + ONE_FX;
1149
1150 case (tick)
1151 6'd0: begin
1152 bag_r <= in_bag;
1153 use_override_r <= in_use_override;
1154 orig_x_r <= in_orig_x;
1155 orig_y_r <= in_orig_y;
1156 orig_z_r <= in_orig_z;
1157 dir_x_lat <= in_dir_x;
1158 dir_y_lat <= in_dir_y;
1159 dir_z_lat <= in_dir_z;
1160 end
1161 6'd5: begin
1162 raw_x_r <= use_override_r ? dir_x_lat : in_basis_raw_x;
1163 raw_y_r <= use_override_r ? dir_y_lat : in_basis_raw_y;
1164 raw_z_r <= use_override_r ? dir_z_lat : in_basis_raw_z;
1165 end
1166 6'd9: prod_x_r <= mul_result;
1167 6'd10: prod_y_r <= mul_result;
1168 6'd11: prod_z_r <= mul_result;

```

```

1169         6'd12: begin
1170             mag_sq_r    <= prod_x_r + prod_y_r + prod_z_r;
1171             kp1_dy_sq_r <= mul_result;    // (1+k²)·D.y² from t=11 issue
1172         end
1173         6'd13: a_cone_r <= mag_sq_r - kp1_dy_sq_r;
1174         default: begin end
1175     endcase
1176 end
1177 end
1178
1179 assign raygen_raw_x = raw_x_r;
1180 assign raygen_raw_y = raw_y_r;
1181 assign raygen_raw_z = raw_z_r;
1182 assign raygen_mag_sq = mag_sq_r;
1183 assign raygen_a_cone = a_cone_r;
1184 assign raygen_ox = orig_x_r;
1185 assign raygen_oy = orig_y_r;
1186 assign raygen_oz = orig_z_r;
1187 assign raygen_bag = bag_r;
1188 endmodule
1189
1190 // Stage B - ISECT PREP: half-discriminant form for both spheres + plane denom.
1191 // a = |raw|² (shared from stage A)
1192 // half_b = dot(dir, oc), oc = orig - sphere_center
1193 // c = dot(oc, oc) - r²
1194 // disc_sqrt = sqrt(half_b² - a·c) (per sphere)
1195 // inv_a, inv_pd: emitted as reciprocals so stage C multiplies instead of
1196 // divides (LAB-relax saves divs).
1197 module rtl_isect_prep
1198     import raytracer_pkg::*;
1199     #(
1200         parameter int FRAC_BITS = 13,
1201         parameter int WORD = 27,
1202         parameter int TAG_W = 7
1203     )(
1204         input logic clk, rst_n, // clock, async neg-edge reset
1205         input logic [5:0] tick, // pipe-local tick counter (0..II-1)
1206         input logic signed [WORD-1:0] raygen_raw_x, raygen_raw_y, raygen_raw_z, // ray direction from Stage A
1207         input logic signed [WORD-1:0] raygen_mag_sq, // |d|² from Stage A (= sphere "a")
1208         input logic signed [WORD-1:0] raygen_a_cone, // a_cone = |d|⁻²(1+k²)·D.y² (precomputed in
1209         Stage A)
1210         // Ray origin (CAM for primaries, parent hit for reflection spawns).
1211         input logic signed [WORD-1:0] raygen_ox, raygen_oy, raygen_oz,
1212         input scene_t in_scene, // scene relay (reads .s0/.s1/.cone geometry
1213     )
1214     input pipe_baggage_t raygen_bag, // {sv, tag, meta} forwarded
1215     output logic signed [WORD-1:0] isect_dx, isect_dy, isect_dz, // registered direction → Stage C
1216     output logic signed [WORD-1:0] isect_inv_sa, // 1/|d|² (1/a) - sphere t-solve via
1217     multiply
1218     output logic signed [WORD-1:0] isect_s0_nb, // sphere 0: -half_b (numerator term)
1219     output logic signed [WORD-1:0] isect_s0_dsq, // sphere 0: √(half_b² - a·c)
1220     output logic isect_s0_nh, // sphere 0: no_hit flag (disc < 0)
1221     output logic signed [WORD-1:0] isect_s1_nb, // sphere 1: -half_b
1222     output logic signed [WORD-1:0] isect_s1_dsq, // sphere 1: √disc
1223     output logic isect_s1_nh, // sphere 1: no_hit flag
1224     output logic signed [WORD-1:0] isect_inv_pd, // 1/dir.y for plane t-solve
1225     // Cone outputs. u_inv_a_cone kicks at t=0 (with a_cone_r from stage A);
1226     // result wire valid by t=42, captured at stage DE's t=0 boundary latch.
1227     output logic signed [WORD-1:0] isect_cone_nb, // cone: -b_cone
1228     output logic signed [WORD-1:0] isect_cone_dsq, // cone: √(b² - a·c)
1229     output logic isect_cone_nh, // cone: no_hit flag
1230     output logic signed [WORD-1:0] isect_inv_a_cone, // 1/a_cone - cone t-solve via multiply
1231     // Δ = origin - apex, forwarded for stage C's cone hit_y bounds check
1232     // and cap/normal math.
1233     output logic signed [WORD-1:0] isect_cone_dx, // cone Δ.x = orig.x - apex.x
1234     output logic signed [WORD-1:0] isect_cone_dy, // cone Δ.y (used for slab + cap)
1235     output logic signed [WORD-1:0] isect_cone_dz, // cone Δ.z
1236     // Registered |d|² for stage C's sun squared-compared. Must read the
1237     // captured a_r (not raw raygen_mag_sq) since A is K=4 pixels ahead of B.
1238     output logic signed [WORD-1:0] isect_sa, // registered |d|² for sun test
1239     // orig flows to C for hit-point reconstruction; orig_y for the plane numer.
1240     output logic signed [WORD-1:0] isect_ox, isect_oy, isect_oz, // registered origin → Stage C
1241     output pipe_baggage_t isect_bag // baggage forwarded to Stage C
1242 );

```

```

1240 localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;
1241
1242 logic signed [WORD-1:0] dx_r, dy_r, dz_r;
1243 logic signed [WORD-1:0] ox_r, oy_r, oz_r;
1244 logic signed [WORD-1:0] a_r;
1245 pipe_baggage_t bag_r;
1246
1247 sphere_t sc0_local, sc1_local;
1248 always_ff @(posedge clk or negedge rst_n) begin
1249     if (!rst_n) begin
1250         sc0_local <= '0;
1251         sc1_local <= '0;
1252     end else begin
1253         sc0_local <= in_scene.s0;
1254         sc1_local <= in_scene.s1;
1255     end
1256 end
1257
1258 // oc = orig - sphere_center, one combinational subtract then registered.
1259 logic signed [WORD-1:0] oc0_x_c, oc0_y_c, oc0_z_c;
1260 logic signed [WORD-1:0] oc1_x_c, oc1_y_c, oc1_z_c;
1261 assign oc0_x_c = ox_r - sc0_local.x;
1262 assign oc0_y_c = oy_r - sc0_local.y;
1263 assign oc0_z_c = oz_r - sc0_local.z;
1264 assign oc1_x_c = ox_r - sc1_local.x;
1265 assign oc1_y_c = oy_r - sc1_local.y;
1266 assign oc1_z_c = oz_r - sc1_local.z;
1267 logic signed [WORD-1:0] oc0_x_r, oc0_y_r, oc0_z_r;
1268 logic signed [WORD-1:0] oc1_x_r, oc1_y_r, oc1_z_r;
1269
1270 // Per-sphere products + intermediates
1271 logic signed [WORD-1:0] p0_hb0_r, p1_hb0_r, p2_hb0_r;
1272 logic signed [WORD-1:0] p0_oc0_r, p1_oc0_r, p2_oc0_r;
1273 logic signed [WORD-1:0] half_b0_r, c0_r;
1274 logic signed [WORD-1:0] half_b0_sqrt_r, a_c0_r;
1275 logic signed [WORD-1:0] disc_sqrt0_r;
1276 logic no_hit0_r;
1277
1278 logic signed [WORD-1:0] p0_hb1_r, p1_hb1_r, p2_hb1_r;
1279 logic signed [WORD-1:0] p0_oc1_r, p1_oc1_r, p2_oc1_r;
1280 logic signed [WORD-1:0] half_b1_r, c1_r;
1281 logic signed [WORD-1:0] half_b1_sqrt_r, a_c1_r;
1282 logic signed [WORD-1:0] disc_sqrt1_r;
1283 logic no_hit1_r;
1284
1285 // Cone state
1286 logic signed [WORD-1:0] cone_dx_r, cone_dy_r, cone_dz_r; // Δ
1287 logic signed [WORD-1:0] cdx_sq_r, cdy_sq_r, cdz_sq_r; // Δ.x2, Δ.y2, Δ.z2
1288 logic signed [WORD-1:0] k_cdy_sq_r; // Δk2·y2
1289 logic signed [WORD-1:0] cdx_dx_r, cdy_dy_r, cdz_dz_r; // Δ·{x,y,z}·D·{x,y,z}
1290 logic signed [WORD-1:0] k_cdy_dy_r; // k2·Δ(y·D·y)
1291 logic signed [WORD-1:0] a_cone_r, b_cone_r, c_cone_r;
1292 logic signed [WORD-1:0] b_cone_sq_r, a_c_cone_r;
1293 logic signed [WORD-1:0] disc_sqrt_cone_r;
1294 logic no_hit_cone_r;
1295
1296 // Two fp_muls: u_mul0 for sphere 0, u_mul1 for sphere 1.
1297 logic signed [WORD-1:0] m0_a, m0_b, m0_r;
1298 logic signed [WORD-1:0] m1_a, m1_b, m1_r;
1299 fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul0 (.clk(clk), .a(m0_a), .b(m0_b), .result(m0_r));
1300 fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul1 (.clk(clk), .a(m1_a), .b(m1_b), .result(m1_r));
1301
1302 logic signed [WORD-1:0] mc_a, mc_b, mc_r;
1303 fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul_cone (.clk(clk), .a(mc_a), .b(mc_b), .result(mc_r));
1304
1305 // Locally-registered `tick` for the operand mux. Same pattern as
1306 // tick_local_h in rtl_hit_resolve: `tick` has 400+ destinations and
1307 // the cross-die IC to u_mul0/u_mul1/u_mul_cone selector clusters was
1308 // a top-violator class. This single FF drives the combined case
1309 // (no preserve so Quartus is free to place / replicate near the DSPs);
1310 // it lags `tick` by 1 cycle so case arm indices shift -1.
1311 logic [5:0] tick_local_i;
1312 always_ff @(posedge clk or negedge rst_n) begin
1313     if (!rst_n) tick_local_i <= 6'd0;

```

```

1314         else          tick_local_i <= tick;
1315     end
1316
1317     always_comb begin
1318         m0_a = '0; m0_b = '0;
1319         m1_a = '0; m1_b = '0;
1320         mc_a = '0; mc_b = '0;
1321         case (tick_local_i)
1322             6'd1: begin                                // tick=2
1323                 m0_a = dx_r;    m0_b = oc0_x_r;
1324                 m1_a = dx_r;    m1_b = oc1_x_r;
1325                 // mc free - D.y2 and k2·D.y2 now precomputed in stage A.
1326             end
1327             6'd2: begin                                // tick=3
1328                 m0_a = dy_r;    m0_b = oc0_y_r;
1329                 m1_a = dy_r;    m1_b = oc1_y_r;
1330             end
1331             6'd3: begin                                // tick=4
1332                 m0_a = dz_r;    m0_b = oc0_z_r;
1333                 m1_a = dz_r;    m1_b = oc1_z_r;
1334                 mc_a = cone_dx_r; mc_b = cone_dx_r;    // Δ.x2 → cap t=5
1335             end
1336             6'd4: begin                                // tick=5
1337                 m0_a = oc0_x_r; m0_b = oc0_x_r;
1338                 m1_a = oc1_x_r; m1_b = oc1_x_r;
1339                 mc_a = cone_dy_r; mc_b = cone_dy_r;    // Δ.y2 → cap t=6
1340             end
1341             6'd5: begin                                // tick=6
1342                 m0_a = oc0_y_r; m0_b = oc0_y_r;
1343                 m1_a = oc1_y_r; m1_b = oc1_y_r;
1344                 mc_a = cone_dz_r; mc_b = cone_dz_r;    // Δ.z2 → cap t=7
1345             end
1346             6'd6: begin                                // tick=7
1347                 m0_a = oc0_z_r; m0_b = oc0_z_r;
1348                 m1_a = oc1_z_r; m1_b = oc1_z_r;
1349                 mc_a = in_scene.cone.k_sq; mc_b = cdy_sq_r; // Δk2·y2 → cap t=8
1350             end
1351             6'd7: begin                                // tick=8
1352                 m0_a = half_b0_r; m0_b = half_b0_r;
1353                 m1_a = half_b1_r; m1_b = half_b1_r;
1354                 mc_a = cone_dx_r; mc_b = dx_r;          // Δ.x·D.x → cap t=9
1355             end
1356             6'd8: begin                                // tick=9
1357                 mc_a = cone_dy_r; mc_b = dy_r;          // Δ.y·D.y → cap t=10
1358             end
1359             6'd9: begin                                // tick=10
1360                 m0_a = a_r; m0_b = c0_r;
1361                 m1_a = a_r; m1_b = c1_r;
1362                 mc_a = cone_dz_r; mc_b = dz_r;          // Δ.z·D.z → cap t=11
1363             end
1364             6'd10: begin                                // tick=11
1365                 mc_a = in_scene.cone.k_sq; mc_b = cdy_dy_r; // k2·Δ(.y·D.y) → cap t=12
1366             end
1367             6'd13: begin                                // tick=14
1368                 mc_a = b_cone_w; mc_b = b_cone_w;      // b2 → cap t=15
1369             end
1370             6'd14: begin                                // tick=15
1371                 mc_a = a_cone_r; mc_b = c_cone_r;      // a·c → cap t=16
1372             end
1373         default: begin end
1374     endcase
1375 end
1376
1377 logic signed [WORD-1:0] b_cone_w, c_cone_w;
1378 assign b_cone_w = cdx_dx_r + cdz_dz_r - k_cdy_dy_r;
1379 assign c_cone_w = cdx_sq_r + cdz_sq_r - k_cdy_sq_r;
1380
1381 // Two fp_sqrtfs in lockstep at tick=13.
1382 logic          sqrt_start;
1383 logic signed [WORD-1:0] sqrt0_in, sqrt1_in;
1384 logic signed [WORD-1:0] sqrt0_out, sqrt1_out;
1385 logic          sqrt0_done, sqrt1_done;
1386 assign sqrt_start = (tick == 6'd13);
1387 assign sqrt0_in   = half_b0_sq_r - a_c0_r;

```

```

1388 assign sqrt1_in = half_b1_sq_r - a_c1_r;
1389 fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt0 (
1390     .clk(clk), .rst_n(rst_n), .start(sqrt_start),
1391     .a(sqrt0_in), .result(sqrt0_out), .done(sqrt0_done)
1392 );
1393 fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt1 (
1394     .clk(clk), .rst_n(rst_n), .start(sqrt_start),
1395     .a(sqrt1_in), .result(sqrt1_out), .done(sqrt1_done)
1396 );
1397
1398 // Cone disc sqrt
1399 logic          sqrt_cone_start;
1400 logic signed [WORD-1:0] sqrt_cone_in;
1401 logic signed [WORD-1:0] sqrt_cone_out;
1402 logic          sqrt_cone_done;
1403 assign sqrt_cone_start = (tick == 6'd17);
1404 assign sqrt_cone_in = b_cone_sq_r - a_c_cone_r;
1405 fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt_cone (
1406     .clk(clk), .rst_n(rst_n), .start(sqrt_cone_start),
1407     .a(sqrt_cone_in), .result(sqrt_cone_out), .done(sqrt_cone_done)
1408 );
1409
1410 always_ff @(posedge clk or negedge rst_n) begin
1411     if (!rst_n) begin
1412         dx_r <= '0; dy_r <= '0; dz_r <= '0;
1413         ox_r <= '0; oy_r <= '0; oz_r <= '0;
1414         a_r <= '0;
1415         oc0_x_r <= '0; oc0_y_r <= '0; oc0_z_r <= '0;
1416         oc1_x_r <= '0; oc1_y_r <= '0; oc1_z_r <= '0;
1417         bag_r <= '0;
1418         p0_hb0_r <= '0; p1_hb0_r <= '0; p2_hb0_r <= '0;
1419         p0_oc0_r <= '0; p1_oc0_r <= '0; p2_oc0_r <= '0;
1420         half_b0_r <= '0; c0_r <= '0;
1421         half_b0_sq_r <= '0; a_c0_r <= '0;
1422         disc_sqrt0_r <= '0; no_hit0_r <= 1'b0;
1423         p0_hb1_r <= '0; p1_hb1_r <= '0; p2_hb1_r <= '0;
1424         p0_oc1_r <= '0; p1_oc1_r <= '0; p2_oc1_r <= '0;
1425         half_b1_r <= '0; c1_r <= '0;
1426         half_b1_sq_r <= '0; a_c1_r <= '0;
1427         disc_sqrt1_r <= '0; no_hit1_r <= 1'b0;
1428         cone_dx_r <= '0; cone_dy_r <= '0; cone_dz_r <= '0;
1429         cdx_sq_r <= '0; cdy_sq_r <= '0; cdz_sq_r <= '0; k_cdy_sq_r <= '0;
1430         cdx_dx_r <= '0; cdy_dy_r <= '0; cdz_dz_r <= '0; k_cdy_dy_r <= '0;
1431         a_cone_r <= '0; b_cone_r <= '0; c_cone_r <= '0;
1432         b_cone_sq_r <= '0; a_c_cone_r <= '0;
1433         disc_sqrt_cone_r <= '0; no_hit_cone_r <= 1'b0;
1434     end else begin
1435         case (tick)
1436             6'd0: begin
1437                 dx_r <= raygen_raw_x;
1438                 dy_r <= raygen_raw_y;
1439                 dz_r <= raygen_raw_z;
1440                 ox_r <= raygen_ox;
1441                 oy_r <= raygen_oy;
1442                 oz_r <= raygen_oz;
1443                 a_r <= raygen_mag_sq;
1444                 a_cone_r <= raygen_a_cone; // precomputed in stage A
1445                 bag_r <= raygen_bag;
1446             end
1447             6'd1: begin
1448                 oc0_x_r <= oc0_x_c; oc0_y_r <= oc0_y_c; oc0_z_r <= oc0_z_c;
1449                 oc1_x_r <= oc1_x_c; oc1_y_r <= oc1_y_c; oc1_z_r <= oc1_z_c;
1450                 cone_dx_r <= ox_r - in_scene.cone.apex_x;
1451                 cone_dy_r <= oy_r - in_scene.cone.apex_y;
1452                 cone_dz_r <= oz_r - in_scene.cone.apex_z;
1453             end
1454             6'd3: begin p0_hb0_r <= m0_r; p0_hb1_r <= m1_r; end
1455             6'd4: begin p1_hb0_r <= m0_r; p1_hb1_r <= m1_r; end
1456             6'd5: begin p2_hb0_r <= m0_r; p2_hb1_r <= m1_r; cdx_sq_r <= mc_r; end
1457             6'd6: begin
1458                 p0_oc0_r <= m0_r; p0_oc1_r <= m1_r;
1459                 half_b0_r <= p0_hb0_r + p1_hb0_r + p2_hb0_r;
1460                 half_b1_r <= p0_hb1_r + p1_hb1_r + p2_hb1_r;
1461                 cdy_sq_r <= mc_r;

```

```

1462         end
1463         6'd7: begin p1_oc0_r <= m0_r; p1_oc1_r <= m1_r; cdz_sq_r <= mc_r; end
1464         6'd8: begin p2_oc0_r <= m0_r; p2_oc1_r <= m1_r; k_cdy_sq_r <= mc_r; end
1465         6'd9: begin
1466             half_b0_sq_r <= m0_r; half_b1_sq_r <= m1_r;
1467             c0_r <= p0_oc0_r + p1_oc0_r + p2_oc0_r - sc0_local.r_sq;
1468             c1_r <= p0_oc1_r + p1_oc1_r + p2_oc1_r - sc1_local.r_sq;
1469             cdx_dx_r <= mc_r;
1470         end
1471         6'd10: cdy_dy_r <= mc_r;
1472         6'd11: begin a_c0_r <= m0_r; a_c1_r <= m1_r; cdz_dz_r <= mc_r; end
1473         6'd12: begin
1474             no_hit0_r <= (half_b0_sq_r < a_c0_r);
1475             no_hit1_r <= (half_b1_sq_r < a_c1_r);
1476             k_cdy_dy_r <= mc_r;
1477         end
1478         6'd14: begin
1479             b_cone_r <= b_cone_w;
1480             c_cone_r <= c_cone_w;
1481         end
1482         6'd15: b_cone_sq_r <= mc_r;
1483         6'd16: begin
1484             a_c_cone_r <= mc_r;
1485             no_hit_cone_r <= (b_cone_sq_r < mc_r);
1486         end
1487         default: begin end
1488     endcase
1489     if (sqrt0_done) disc_sqrt0_r <= sqrt0_out;
1490     if (sqrt1_done) disc_sqrt1_r <= sqrt1_out;
1491     if (sqrt_cone_done) disc_sqrt_cone_r <= sqrt_cone_out;
1492 end
1493 end
1494
1495 assign isect_dx = dx_r;
1496 assign isect_dy = dy_r;
1497 assign isect_dz = dz_r;
1498 assign isect_s0_nb = -half_b0_r;
1499 assign isect_s0_dsq = disc_sqrt0_r;
1500 assign isect_s0_nh = no_hit0_r;
1501 assign isect_s1_nb = -half_b1_r;
1502 assign isect_s1_dsq = disc_sqrt1_r;
1503 assign isect_s1_nh = no_hit1_r;
1504 // Clamp |dy| 1/2048 so plane t = -orig.y/dy can't blow up at horizon
1505 // rays (dy 0)
1506 localparam signed [WORD-1:0] PLANE_DENOM_EPS = 1 <<< (FRAC_BITS - 11);
1507 logic signed [WORD-1:0] plane_denom_c;
1508 assign plane_denom_c =
1509     (raygen_raw_y > -PLANE_DENOM_EPS && raygen_raw_y < PLANE_DENOM_EPS)
1510     ? ((raygen_raw_y >= 0) ? PLANE_DENOM_EPS : -PLANE_DENOM_EPS)
1511     : raygen_raw_y;
1512
1513 // 2 concurrent fp_invs at t=0 emit 1/sphere_a and 1/plane_denom for
1514 // stage C's t-muls (replaces 3 fp_divs in the old split). Finishes by t=40.
1515 logic inv_start;
1516 assign inv_start = (tick == 6'd0);
1517 logic signed [WORD-1:0] inv_a_w, inv_pd_w;
1518 logic inv_a_done, inv_pd_done;
1519 fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_a (
1520     .clk(clk), .rst_n(rst_n), .start(inv_start),
1521     .b(raygen_mag_sq), .result(inv_a_w), .done(inv_a_done)
1522 );
1523 fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_pd (
1524     .clk(clk), .rst_n(rst_n), .start(inv_start),
1525     .b(plane_denom_c), .result(inv_pd_w), .done(inv_pd_done)
1526 );
1527
1528 // Cone fp_inv kicks at tick=0 alongside u_inv_a/u_inv_pd. Input a_cone_r
1529 // is precomputed in stage A and latched here at t=0; result is wire-valid
1530 // during t=42 and captured by stage DE at its own t=0 boundary latch
1531 // (2 clock periods of →FFFF slack). See docs/cone_fp_inv_timing.html.
1532 logic inv_cone_start;
1533 assign inv_cone_start = (tick == 6'd0);
1534 logic signed [WORD-1:0] inv_a_cone_w;
1535 logic inv_a_cone_done;

```

```

1536 // Same-edge race fix: at tick=0 the boundary latch a_cone_r <=
1537 // raygen_a_cone fires on the SAME posedge that fp_inv samples b.
1538 // fp_inv would see the pre-edge a_cone_r (one II behind). Feed the
1539 // upstream wire instead - it's been stable since stage A's t=13 of
1540 // the previous II.
1541 fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_a_cone (
1542     .clk(clk), .rst_n(rst_n), .start(inv_cone_start),
1543     .b(raygen_a_cone), .result(inv_a_cone_w), .done(inv_a_cone_done)
1544 );
1545
1546 assign isect_inv_sa = inv_a_w;
1547 assign isect_inv_pd = inv_pd_w;
1548 assign isect_sa     = a_r;
1549 assign isect_inv_a_cone = inv_a_cone_w;
1550 assign isect_cone_nb   = -b_cone_r;
1551 assign isect_cone_dsq  = disc_sqrt_cone_r;
1552 assign isect_cone_nh   = no_hit_cone_r;
1553 assign isect_cone_dx   = cone_dx_r;
1554 assign isect_cone_dy   = cone_dy_r;
1555 assign isect_cone_dz   = cone_dz_r;
1556
1557 assign isect_ox        = ox_r;
1558 assign isect_oy        = oy_r;
1559 assign isect_oz        = oz_r;
1560 assign isect_bag       = bag_r;
1561 endmodule
1562
1563 // Stage C - HIT RESOLVE: intersect t-muls + winner mux (first half) fused
1564 // with hit point + base color + normal prep (second half). Long pole is the
1565 // diff_mag sqrt, completing by t=34. hit_type 3'b000=sky / 001=s0 / 010=s1 /
1566 // 011=plane / 100=cone.
1567 module rtl_hit_resolve
1568     import raytracer_pkg::*;
1569 #(
1570     parameter int FRAC_BITS = 13,
1571     parameter int WORD      = 27,
1572     parameter int TAG_W     = 7
1573 )(
1574     input logic          clk, rst_n, // clock, async neg-edge reset
1575     input logic [5:0]    tick,       // pipe-local tick counter
1576     input logic signed [WORD-1:0] isect_dx, isect_dy, isect_dz, // ray direction (from Stage B)
1577     input logic signed [WORD-1:0] isect_ox, isect_oy, isect_oz, // ray origin
1578     input logic signed [WORD-1:0] isect_inv_sa, // 1/|d|^2 for sphere t-solve
1579     input logic signed [WORD-1:0] isect_s0_nb, // sphere 0: -half_b
1580     input logic signed [WORD-1:0] isect_s0_dsq, // sphere 0: sqrt(disc)
1581     input logic          isect_s0_nh, // sphere 0: no_hit flag
1582     input logic signed [WORD-1:0] isect_s1_nb, // sphere 1: -half_b
1583     input logic signed [WORD-1:0] isect_s1_dsq, // sphere 1: sqrt(disc)
1584     input logic          isect_s1_nh, // sphere 1: no_hit flag
1585     input logic signed [WORD-1:0] isect_inv_pd, // 1/dir.y for plane t-solve
1586     input logic signed [WORD-1:0] isect_sa, // |d|^2 for sun squared-compare
1587     // Cone path inputs from rtl_isect_prep. inv_a_cone is captured later
1588     // (tick=14) since C's fp_inv kicked at C tick=15; the rest are stable
1589     // from start of II.
1590     input logic signed [WORD-1:0] isect_cone_nb, // cone: -b_cone
1591     input logic signed [WORD-1:0] isect_cone_dsq, // cone: sqrt(disc)
1592     input logic          isect_cone_nh, // cone: no_hit flag
1593     input logic signed [WORD-1:0] isect_inv_a_cone, // 1/a_cone (late settle, captured at
1594     tick=14)
1595     input logic signed [WORD-1:0] isect_cone_dx, // cone Δx = orig.x - apex.x
1596     input logic signed [WORD-1:0] isect_cone_dy, // cone Δy (for slab + cap)
1597     input logic signed [WORD-1:0] isect_cone_dz, // cone Δz
1598     input logic signed [WORD-1:0] light_x_local, light_y_local, light_z_local, // light pos (per-pipe relay) for
1599     sun + Lambert prep
1600     input scene_t          in_scene, // scene relay (this stage reads
1601     every field)
1602     input logic signed [WORD-1:0] K_const_local, // |light|^2·63/64 - sun-billboard
1603     threshold
1604     input pipe_baggage_t    isect_bag, // {sv, tag, meta} from Stage B
1605     output logic signed [WORD-1:0] hit_hx, hit_hy, hit_hz, // surface hit point
1606     output logic signed [WORD-1:0] hit_dfx, hit_dfy, hit_dfz, // diff = hit - sphere_center (sphere
1607     normal num)
1608     output logic signed [WORD-1:0] hit_tlx, hit_tly, hit_tlz, // to-light vector tl = light - hit

```

```

1604     output logic signed [WORD-1:0] hit_cr, hit_cg, hit_cb,           // base surface RGB (after checker/
sun mux)
1605     output logic [2:0] hit_hit_type,                               // 000 sky, 001 s0, 010 s1, 011 plane
, 100 cone
1606     output logic signed [WORD-1:0] hit_dmag,                       // |diff| for sphere - needed by
Stage D's 1/|n|
1607     // Forward parent direction to stage E (for refl r = d - 2·(d·n)·n).
1608     output logic signed [WORD-1:0] hit_dx, hit_dy, hit_dz,         // parent ray direction → Stage E
refl formula
1609     output pipe_baggage_t hit_bag                                  // baggage forwarded to Stage D
1610 );
1611     localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;
1612     localparam signed [WORD-1:0] SKY2_G = (7 * ONE) / 10;
1613     localparam signed [WORD-1:0] SUN_COL_R = ONE;                  // 1.0
1614     localparam signed [WORD-1:0] SUN_COL_G = (95 * ONE) / 100;    // 0.95
1615     localparam signed [WORD-1:0] SUN_COL_B = (55 * ONE) / 100;    // 0.55
1616
1617     // ---- Tick=0 latches: dir, orig, light, sphere data, tag, meta ----
1618     logic signed [WORD-1:0] dx_r, dy_r, dz_r;
1619     logic signed [WORD-1:0] ox_r, oy_r, oz_r;
1620     logic signed [WORD-1:0] lx_r, ly_r, lz_r;
1621
1622     logic signed [WORD-1:0] a_r;
1623     logic no_hit0_r, no_hit1_r;
1624     pipe_baggage_t bag_r;
1625
1626     // ---- ISECT (first half of stage C - intersect t-muls + winner mux) ----
1627     // Combinational numerators from upstream wires (race-fix).
1628     logic signed [WORD-1:0] num_t0_s0, num_t0_s1;
1629     assign num_t0_s0 = isect_s0_nb - isect_s0_dsq;
1630     assign num_t0_s1 = isect_s1_nb - isect_s1_dsq;
1631     logic signed [WORD-1:0] plane_numer_w;
1632     assign plane_numer_w = -isect_oy;
1633
1634     logic signed [WORD-1:0] num_t0_s0_r, num_t0_s1_r, plane_numer_r;
1635     logic signed [WORD-1:0] inv_a_r, inv_pd_r;
1636
1637     // t-mul outputs (registered after DSP).
1638     logic signed [WORD-1:0] t0_s0_r, t0_s1_r, tp_r;
1639
1640     // Winner mux output, then registered to break C-side comb chain off
1641     // the hit-mul DSP a-input.
1642     logic signed [WORD-1:0] hit_t_w, hit_t_r;
1643     logic [2:0] hit_type_w, hit_type_r;
1644
1645     // ---- HP + NORMAL PREP (second half of stage C) ----
1646     logic signed [WORD-1:0] m_dh_x_r, m_dh_y_r, m_dh_z_r;
1647     logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
1648     logic signed [WORD-1:0] diff_x_r, diff_y_r, diff_z_r;
1649     logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
1650     logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
1651     logic signed [WORD-1:0] sky_g_r;
1652     logic signed [WORD-1:0] prod0_r, prod1_r, prod2_r;
1653     logic signed [WORD-1:0] diff_mag_r;
1654     // Pipelined t_sky / inv_t_sky to lift col_*_r off the dy_r adder cascade.
1655     logic signed [WORD-1:0] t_sky_r, inv_t_sky_r;
1656
1657     // ---- Cone state (parallel to sphere/plane path) ----
1658     logic signed [WORD-1:0] cone_nb_r, cone_dsqr_r;
1659     logic cone_nh_r;
1660     logic signed [WORD-1:0] cone_dx_r, cone_dy_r, cone_dz_r;        // Δ (origin - apex)
1661     logic signed [WORD-1:0] inv_a_cone_r;
1662     logic signed [WORD-1:0] t_cone_r;
1663     logic signed [WORD-1:0] m_dh_cx_r, m_dh_cy_r, m_dh_cz_r;        // dir·t_cone
1664     logic signed [WORD-1:0] hit_x_cone_r, hit_y_cone_r, hit_z_cone_r;
1665     logic cone_in_slab_r;                                           // hit_y in [apex.y-h, apex.y]
1666     logic cone_wins_r;                                              // final mux decision
1667     logic signed [WORD-1:0] cone_norm_x_r, cone_norm_y_r, cone_norm_z_r; // unnormalized gradient
1668     logic signed [WORD-1:0] cone_kdy_r;                             // k²·(hit_y-apex_y) pre-neg
1669     logic signed [WORD-1:0] cone_diff_mag_r;                        // factor · (apex.y - hit.y)
1670
1671     // Single shared fp_mul scheduled across t-muls, hit-muls, sky, diff².
1672     logic signed [WORD-1:0] mul_a, mul_b, mul_result;
1673     fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul (

```



```

1674     .clk(clk), .a(mul_a), .b(mul_b), .result(mul_result)
1675 );
1676
1677 logic signed [WORD-1:0] mc_a, mc_b, mc_r;
1678 fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul_cone_e (
1679     .clk(clk), .a(mc_a), .b(mc_b), .result(mc_r)
1680 );
1681
1682 logic signed [WORD-1:0] sun_x_r, sun_y_r, sun_z_r;
1683 logic signed [WORD-1:0] sun_dot_c;
1684 logic signed [WORD-1:0] sun_dot_r;
1685 assign sun_dot_c = sun_x_r + sun_y_r + sun_z_r;
1686
1687 // Squared-compare regs (u_mul time-mux'd at t=9/10): K_const·a_r →
1688 // sun_thresh_a_r; sun_dot²; combinational sign+GE; 1c FF for safety.
1689 logic signed [WORD-1:0] sun_thresh_a_r;
1690 logic signed [WORD-1:0] sun_dot_sq_r;
1691 logic sun_active_w;
1692 // Sign gate (~sun_dot_r MSB) catches antipodal rays whose squared
1693 // compare would otherwise pass.
1694 assign sun_active_w = (~sun_dot_r[WORD-1])
1695     && (sun_dot_sq_r >= sun_thresh_a_r);
1696 logic sun_active_r;
1697
1698 // sky t = (dir_y + ONE) >>> 1 (combinational from registered dy_r).
1699 logic signed [WORD-1:0] t_sky_c;
1700 assign t_sky_c = (dy_r + ONE) >>> 1;
1701
1702 // Locally-registered `tick` for the operand muxes. `tick` has ~400
1703 // destinations across the pipe, and the cross-die IC from tick[3] to
1704 // the cone-mul DSP's selector cluster was the worst-slack path
1705 // (-0.477 ns @ SEED=2). This single FF lags `tick` by 1 cycle and
1706 // drives BOTH operand muxes (u_mul and u_mul_cone_e); no preserve
1707 // attribute so Quartus is free to place / replicate it as needed near
1708 // the consumer DSPs. Same pattern as the validated stage_e fix that
1709 // closed -0.091 ns at the visual-upgrades baseline. Case arm indices
1710 // shift by -1; always_ff captures still use `tick` directly so the
1711 // tick=0 input latch and downstream stage handshake are unchanged.
1712 logic [5:0] tick_local_h;
1713 always_ff @(posedge clk or negedge rst_n) begin
1714     if (!rst_n) tick_local_h <= 6'd0;
1715     else tick_local_h <= tick;
1716 end
1717
1718 always_comb begin
1719     mul_a = '0; mul_b = '0;
1720     case (tick_local_h)
1721         6'd0: begin mul_a = num_t0_s0_r; mul_b = inv_a_r; end // tick=1
1722         6'd1: begin mul_a = num_t0_s1_r; mul_b = inv_a_r; end // tick=2
1723         6'd2: begin mul_a = plane_numer_r; mul_b = inv_pd_r; end // tick=3
1724         6'd3: begin mul_a = SKY2_G; mul_b = t_sky_r; end // tick=4
1725         6'd4: begin mul_a = dx_r; mul_b = lx_r; end // tick=5
1726         6'd5: begin mul_a = dy_r; mul_b = ly_r; end // tick=6
1727         6'd6: begin mul_a = dz_r; mul_b = lz_r; end // tick=7
1728         6'd8: begin mul_a = K_const_local; mul_b = a_r; end // tick=9
1729         6'd9: begin mul_a = dx_r; mul_b = hit_t_r; end // tick=10
1730         6'd10: begin mul_a = dy_r; mul_b = hit_t_r; end // tick=11
1731         6'd11: begin mul_a = dz_r; mul_b = hit_t_r; end // tick=12
1732         6'd12: begin mul_a = sun_dot_r; mul_b = sun_dot_r; end // tick=13
1733         6'd15: begin mul_a = diff_x_r; mul_b = diff_x_r; end // tick=16
1734         6'd16: begin mul_a = diff_y_r; mul_b = diff_y_r; end // tick=17
1735         6'd17: begin mul_a = diff_z_r; mul_b = diff_z_r; end // tick=18
1736         default: begin end
1737     endcase
1738 end
1739
1740 // Cone fp_mul scheduler. inv_a_cone_r is now captured at t=0 (boundary
1741 // latch alongside inv_a_r/inv_pd_r), so the t=15 mux reads a settled
1742 // value with plenty of slack.
1743 logic signed [WORD-1:0] num_cone_w;
1744 assign num_cone_w = cone_nb_r - cone_dsqr_r; // = -b - sqrt_disc
1745
1746 // Apex cull: reject cone hits within CONE_APEX_CULL of the apex
1747 // (1/32 m at FRAC_BITS=13)

```

```

1748 localparam signed [WORD-1:0] CONE_APEX_CULL = 1 <<< (FRAC_BITS - 5);
1749
1750 always_comb begin
1751     mc_a = '0; mc_b = '0;
1752     case (tick_local_h)
1753         6'd14: begin mc_a = num_cone_w;      mc_b = inv_a_cone_r; end          // tick=15
1754         6'd16: begin mc_a = dx_r;           mc_b = t_cone_r;      end          // tick=17
1755         6'd17: begin mc_a = dy_r;           mc_b = t_cone_r;      end          // tick=18
1756         6'd18: begin mc_a = dz_r;           mc_b = t_cone_r;      end          // tick=19
1757         6'd21: begin mc_a = in_scene.cone.k_sq;                                     // tick=22
1758                 mc_b = (hit_y_cone_r - in_scene.cone.apex_y); end
1759         6'd22: begin mc_a = in_scene.cone.norm_factor;                             // tick=23
1760                 mc_b = (in_scene.cone.apex_y - hit_y_cone_r); end
1761         default: begin end
1762     endcase
1763 end
1764
1765 // ---- Per-candidate hit flags (read registered t values) ----
1766 logic s0_hit, s1_hit, plane_hit;
1767 assign s0_hit = !no_hit0_r && (t0_s0_r > 0);
1768 assign s1_hit = !no_hit1_r && (t0_s1_r > 0);
1769 assign plane_hit = (tp_r > 0);
1770
1771 // Closest-t winner (3 parallel pairwise compares + win-flag AND + 4-way
1772 // mux, ~5 LUT levels). Tie-breaking: s0 > s1 > plane.
1773 logic s1_lt_s0, tp_lt_s0, tp_lt_s1;
1774 assign s1_lt_s0 = t0_s1_r < t0_s0_r;
1775 assign tp_lt_s0 = tp_r < t0_s0_r;
1776 assign tp_lt_s1 = tp_r < t0_s1_r;
1777 logic s0_wins, s1_wins, plane_wins;
1778 assign s0_wins = s0_hit && (!s1_hit || !s1_lt_s0) && (!plane_hit || !tp_lt_s0);
1779 assign s1_wins = s1_hit && (!s0_hit || s1_lt_s0) && (!plane_hit || !tp_lt_s1);
1780 assign plane_wins = plane_hit && (!s0_hit || tp_lt_s0) && (!s1_hit || tp_lt_s1);
1781 always_comb begin
1782     unique case ({s0_wins, s1_wins, plane_wins})
1783         3'b100: begin hit_t_w = t0_s0_r; hit_type_w = HIT_TYPE_S0; end
1784         3'b010: begin hit_t_w = t0_s1_r; hit_type_w = HIT_TYPE_S1; end
1785         3'b001: begin hit_t_w = tp_r; hit_type_w = HIT_TYPE_PLANE; end
1786         default: begin hit_t_w = '0; hit_type_w = HIT_TYPE_SKY; end
1787     endcase
1788 end
1789
1790 // Floor pattern from hit_x_r/hit_z_r bits. Modes: 00=1m checker,
1791 // 01=Sierpinski (3-bit AND==0 on 8m tile), 10=diagonal stripes,
1792 // 11=0.5m checker. Two's-complement bit extract works across origin.
1793 logic chk_bit;
1794 always_comb begin
1795     unique case (in_scene.floor.mode)
1796         FLOOR_MODE_1M_CHECK: chk_bit = hit_x_r[FRAC_BITS] ^ hit_z_r[FRAC_BITS];
1797         FLOOR_MODE_SIERPINSKI: chk_bit = (hit_x_r[FRAC_BITS+:3] & hit_z_r[FRAC_BITS+:3]) == 3'b000;
1798         FLOOR_MODE_DIAGONAL: chk_bit = hit_x_r[FRAC_BITS+1] ^ hit_z_r[FRAC_BITS];
1799         FLOOR_MODE_FINE_CHECK: chk_bit = hit_x_r[FRAC_BITS-1] ^ hit_z_r[FRAC_BITS-1];
1800         default: chk_bit = hit_x_r[FRAC_BITS] ^ hit_z_r[FRAC_BITS];
1801     endcase
1802 end
1803
1804 logic chk_bit_r;
1805 always_ff @(posedge clk or negedge rst_n) begin
1806     if (!rst_n) chk_bit_r <= 1'b0;
1807     else chk_bit_r <= chk_bit;
1808 end
1809
1810 function automatic logic signed [WORD-1:0] clamp01(input logic signed [WORD-1:0] v);
1811     if (v < 0) return '0;
1812     else if (v > ONE) return ONE;
1813     else return v;
1814 endfunction
1815
1816 // ---- Sphere center select (registered hit_type_r drives the mux) ----
1817 logic signed [WORD-1:0] sc_sel_x, sc_sel_y, sc_sel_z;
1818 assign sc_sel_x = (hit_type_r == HIT_TYPE_S1) ? in_scene.s1.x : in_scene.s0.x;
1819 assign sc_sel_y = (hit_type_r == HIT_TYPE_S1) ? in_scene.s1.y : in_scene.s0.y;
1820 assign sc_sel_z = (hit_type_r == HIT_TYPE_S1) ? in_scene.s1.z : in_scene.s0.z;
1821

```

```

1822 logic signed [WORD-1:0] sc_sel_x_r, sc_sel_y_r, sc_sel_z_r;
1823 always_ff @(posedge clk or negedge rst_n) begin
1824     if (!rst_n) begin
1825         sc_sel_x_r <= '0;
1826         sc_sel_y_r <= '0;
1827         sc_sel_z_r <= '0;
1828     end else begin
1829         sc_sel_x_r <= sc_sel_x;
1830         sc_sel_y_r <= sc_sel_y;
1831         sc_sel_z_r <= sc_sel_z;
1832     end
1833 end
1834
1835 // Sky-gradient combinational
1836 logic signed [WORD-1:0] inv_t_sky_c;
1837 logic signed [WORD-1:0] sky_r_c, sky_g_full_c, sky_b_c;
1838 assign inv_t_sky_c = ONE - t_sky_c;
1839 assign sky_r_c = clamp01(inv_t_sky_r + (t_sky_r >>> 1));
1840 assign sky_g_full_c = clamp01(inv_t_sky_r + sky_g_r);
1841 assign sky_b_c = ONE;
1842
1843 // Base color mux on hit_type: 000=sky, 001=s0, 010=s1, 011=plane (chk_bit), 100=cone.
1844 logic signed [WORD-1:0] col_r_c, col_g_c, col_b_c;
1845 always_comb begin
1846     case (hit_type_r)
1847         HIT_TYPE_S0: begin
1848             col_r_c = in_scene.s0.color.r; col_g_c = in_scene.s0.color.g; col_b_c = in_scene.s0.color.b;
1849         end
1850         HIT_TYPE_S1: begin
1851             col_r_c = in_scene.s1.color.r; col_g_c = in_scene.s1.color.g; col_b_c = in_scene.s1.color.b;
1852         end
1853         HIT_TYPE_PLANE: begin
1854             if (!chk_bit_r) begin
1855                 col_r_c = in_scene.floor.pc1.r; col_g_c = in_scene.floor.pc1.g; col_b_c = in_scene.floor.pc1.b;
1856             end else begin
1857                 col_r_c = in_scene.floor.pc2.r; col_g_c = in_scene.floor.pc2.g; col_b_c = in_scene.floor.pc2.b;
1858             end
1859         end
1860         default: begin
1861             col_r_c = sky_r_c;
1862             col_g_c = sky_g_full_c;
1863             col_b_c = sky_b_c;
1864         end
1865     endcase
1866 end
1867
1868 logic          sqrt_start;
1869 logic signed [WORD-1:0] sqrt_in;
1870 logic signed [WORD-1:0] sqrt_out;
1871 logic          sqrt_done;
1872 assign sqrt_start = (tick == 6'd20);
1873 assign sqrt_in = prod0_r + prod1_r + prod2_r;
1874 fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt (
1875     .clk(clk), .rst_n(rst_n), .start(sqrt_start),
1876     .a(sqrt_in), .result(sqrt_out), .done(sqrt_done)
1877 );
1878
1879 // ---- All FFs ----
1880 always_ff @(posedge clk or negedge rst_n) begin
1881     if (!rst_n) begin
1882         dx_r <= '0; dy_r <= '0; dz_r <= '0;
1883         ox_r <= '0; oy_r <= '0; oz_r <= '0;
1884         lx_r <= '0; ly_r <= '0; lz_r <= '0;
1885         a_r <= '0;
1886         no_hit0_r <= 1'b0; no_hit1_r <= 1'b0;
1887         bag_r <= '0;
1888         num_t0_s0_r <= '0; num_t0_s1_r <= '0; plane_numer_r <= '0;
1889         inv_a_r <= '0; inv_pd_r <= '0;
1890         t0_s0_r <= '0; t0_s1_r <= '0; tp_r <= '0;
1891         hit_t_r <= '0; hit_type_r <= HIT_TYPE_SKY;
1892         m_dh_x_r <= '0; m_dh_y_r <= '0; m_dh_z_r <= '0;
1893         hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
1894         diff_x_r <= '0; diff_y_r <= '0; diff_z_r <= '0;
1895         tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;

```

```

1896     col_r_r <= '0'; col_g_r <= '0'; col_b_r <= '0';
1897     sky_g_r <= '0';
1898     prod0_r <= '0'; prod1_r <= '0'; prod2_r <= '0';
1899     diff_mag_r <= '0';
1900     t_sky_r <= '0'; inv_t_sky_r <= '0';
1901     sun_thresh_a_r <= '0';
1902     sun_dot_sq_r <= '0';
1903     sun_dot_r <= '0';
1904     sun_active_r <= 1'b0;
1905     sun_x_r <= '0'; sun_y_r <= '0'; sun_z_r <= '0';
1906     cone_nb_r <= '0'; cone_dsqr <= '0'; cone_nh_r <= 1'b0;
1907     cone_dx_r <= '0'; cone_dy_r <= '0'; cone_dz_r <= '0';
1908     inv_a_cone_r <= '0';
1909     t_cone_r <= '0';
1910     m_dh_cx_r <= '0'; m_dh_cy_r <= '0'; m_dh_cz_r <= '0';
1911     hit_x_cone_r <= '0'; hit_y_cone_r <= '0'; hit_z_cone_r <= '0';
1912     cone_in_slab_r <= 1'b0;
1913     cone_wins_r <= 1'b0;
1914     cone_norm_x_r <= '0'; cone_norm_y_r <= '0'; cone_norm_z_r <= '0';
1915     cone_kdy_r <= '0';
1916     cone_diff_mag_r <= '0';
1917 end else begin
1918     t_sky_r    <= t_sky_c;
1919     inv_t_sky_r <= inv_t_sky_c;
1920     case (tick)
1921     6'd0: begin
1922         dx_r <= isect_dx; dy_r <= isect_dy; dz_r <= isect_dz;
1923         ox_r <= isect_ox; oy_r <= isect_oy; oz_r <= isect_oz;
1924         lx_r <= light_x_local; ly_r <= light_y_local; lz_r <= light_z_local;
1925         a_r <= isect_sa;
1926         no_hit0_r <= isect_s0_nh || isect_bag.meta.excluded[0];
1927         no_hit1_r <= isect_s1_nh || isect_bag.meta.excluded[1];
1928         bag_r <= isect_bag;
1929         num_t0_s0_r <= num_t0_s0;
1930         num_t0_s1_r <= num_t0_s1;
1931         plane_numer_r <= plane_numer_w;
1932         inv_a_r <= isect_inv_sa;
1933         inv_pd_r <= isect_inv_pd;
1934         // Cone fp_inv result is captured here at the boundary
1935         // latch alongside inv_a_r/inv_pd_r (cone kicks at C=t=0)
1936         // now that a_cone is precomputed in stage A).
1937         inv_a_cone_r <= isect_inv_a_cone;
1938         // Cone Stage C outputs (stable from start of II).
1939         // excluded[2] is the cone self-skip bit on reflection rays.
1940         cone_nb_r <= isect_cone_nb;
1941         cone_dsqr <= isect_cone_dsqr;
1942         cone_nh_r <= isect_cone_nh || isect_bag.meta.excluded[2];
1943         cone_dx_r <= isect_cone_dx;
1944         cone_dy_r <= isect_cone_dy;
1945         cone_dz_r <= isect_cone_dz;
1946     end
1947     6'd2: t0_s0_r <= mul_result;
1948     6'd3: t0_s1_r <= mul_result;
1949     6'd4: tp_r <= mul_result;
1950     6'd5: begin
1951         hit_t_r <= hit_t_w;
1952         hit_type_r <= hit_type_w;
1953         sky_g_r <= mul_result;
1954     end
1955     6'd6: sun_x_r <= mul_result;
1956     6'd7: sun_y_r <= mul_result;
1957     6'd8: sun_z_r <= mul_result;
1958     6'd9: sun_dot_r <= sun_dot_c;
1959     6'd10: sun_thresh_a_r <= mul_result;
1960     6'd11: m_dh_x_r <= mul_result; // HP-x from t=10 issue
1961     6'd12: m_dh_y_r <= mul_result; // HP-y from t=11
1962     6'd13: m_dh_z_r <= mul_result; // HP-z from t=12
1963     6'd14: begin
1964         hit_x_r <= ox_r + m_dh_x_r;
1965         hit_y_r <= oy_r + m_dh_y_r;
1966         hit_z_r <= oz_r + m_dh_z_r;
1967         sun_dot_sq_r <= mul_result; // sun^2 from t=13 issue
1968     end
1969     6'd15: begin

```

```

1970         diff_x_r <= hit_x_r - sc_sel_x_r;
1971         diff_y_r <= hit_y_r - sc_sel_y_r;
1972         diff_z_r <= hit_z_r - sc_sel_z_r;
1973         tl_x_r   <= lx_r - hit_x_r;
1974         tl_y_r   <= ly_r - hit_y_r;
1975         tl_z_r   <= lz_r - hit_z_r;
1976         sun_active_r <= sun_active_w;
1977     end
1978     6'd16: begin
1979         if (hit_type_r == HIT_TYPE_SKY && sun_active_r) begin
1980             col_r_r <= SUN_COL_R;
1981             col_g_r <= SUN_COL_G;
1982             col_b_r <= SUN_COL_B;
1983         end else begin
1984             col_r_r <= col_r_c;
1985             col_g_r <= col_g_c;
1986             col_b_r <= col_b_c;
1987         end
1988         t_cone_r <= mc_r;                                // = num_cone · inv_a_cone
1989     end
1990     6'd17: prod0_r <= mul_result;                        // diff_x² from t=16
1991     6'd18: begin
1992         prod1_r   <= mul_result;                        // diff_y² from t=17
1993         m_dh_cx_r <= mc_r;                              // from t=17 mul (dx · t_cone_r settled)
1994     end
1995     6'd19: begin
1996         prod2_r   <= mul_result;                        // diff_z² from t=18
1997         m_dh_cy_r <= mc_r;                              // from t=18
1998     end
1999     6'd20: m_dh_cz_r <= mc_r;                            // from t=19
2000     6'd21: begin
2001         // Cone hit point. m_dh_c*_r are all settled by now.
2002         hit_x_cone_r <= ox_r + m_dh_cx_r;
2003         hit_y_cone_r <= oy_r + m_dh_cy_r;
2004         hit_z_cone_r <= oz_r + m_dh_cz_r;
2005     end
2006     6'd23: begin
2007         cone_kdy_r   <= mc_r;                            // from t=22 mul
2008         // Slab boundary + apex cull. Uses settled hit_y_cone_r
2009         // (registered at t=21, settled t=22, read at t=23 LHS).
2010         cone_in_slab_r <= ( (hit_y_cone_r <= (in_scene.cone.apex_y - CONE_APEX_CULL)) &&
2011             (hit_y_cone_r >= (in_scene.cone.apex_y - in_scene.cone.height)) );
2012         // Surface gradient X/Z components (Y handled at t=24
2013         // because it needs the freshly-captured cone_kdy_r).
2014         cone_norm_x_r <= hit_x_cone_r - in_scene.cone.apex_x;
2015         cone_norm_z_r <= hit_z_cone_r - in_scene.cone.apex_z;
2016     end
2017     6'd24: begin
2018         cone_wins_r   <= !cone_nh_r && (t_cone_r > 0) && cone_in_slab_r &&
2019             ((hit_type_r == HIT_TYPE_SKY) ||
2020             (t_cone_r < hit_t_r));
2021         // Y component of gradient = -k²·(hit_y - apex_y); cone_kdy_r
2022         // holds k²·(hit_y - apex_y) with sign so we just negate
2023         cone_norm_y_r <= -cone_kdy_r;
2024         cone_diff_mag_r <= mc_r;                        // from t=23 mul: factor · (apex_y - hit_y)
2025     end
2026     default: begin end
2027 endcase
2028 if (sqrt_done) diff_mag_r <= sqrt_out;
2029 end
2030
2031
2032 // ---- Output muxing: cone_wins_r overrides hit*_r and diff vector ----
2033 logic [2:0]          final_hit_type;
2034 logic signed [WORD-1:0] final_hx, final_hy, final_hz;
2035 logic signed [WORD-1:0] final_dfx, final_dfy, final_dfz;
2036 logic signed [WORD-1:0] final_dmag;
2037 logic signed [WORD-1:0] final_cr, final_cg, final_cb;
2038 logic signed [WORD-1:0] final_tlx, final_tly, final_tlz;
2039
2040 always_comb begin
2041     if (cone_wins_r) begin
2042         final_hit_type = HIT_TYPE_CONE;
2043         final_hx       = hit_x_cone_r;

```

```

2044         final_hy      = hit_y_cone_r;
2045         final_hz      = hit_z_cone_r;
2046         final_dfx     = cone_norm_x_r;
2047         final_dfy     = cone_norm_y_r;
2048         final_dfz     = cone_norm_z_r;
2049         final_dmag     = cone_diff_mag_r;
2050         final_cr      = in_scene.cone.color.r;
2051         final_cg      = in_scene.cone.color.g;
2052         final_cb      = in_scene.cone.color.b;
2053         final_tlx     = lx_r - hit_x_cone_r;
2054         final_tly     = ly_r - hit_y_cone_r;
2055         final_tlz     = lz_r - hit_z_cone_r;
2056     end else begin
2057         final_hit_type = hit_type_r;
2058         final_hx      = hit_x_r;
2059         final_hy      = hit_y_r;
2060         final_hz      = hit_z_r;
2061         final_dfx     = diff_x_r;
2062         final_dfy     = diff_y_r;
2063         final_dfz     = diff_z_r;
2064         final_dmag     = diff_mag_r;
2065         final_cr      = col_r_r;
2066         final_cg      = col_g_r;
2067         final_cb      = col_b_r;
2068         final_tlx     = tl_x_r;
2069         final_tly     = tl_y_r;
2070         final_tlz     = tl_z_r;
2071     end
2072 end
2073
2074 assign hit_hx = final_hx;
2075 assign hit_hy = final_hy;
2076 assign hit_hz = final_hz;
2077 assign hit_dfx = final_dfx;
2078 assign hit_dfy = final_dfy;
2079 assign hit_dfz = final_dfz;
2080 assign hit_tlx = final_tlx;
2081 assign hit_tly = final_tly;
2082 assign hit_tlz = final_tlz;
2083 assign hit_cr = final_cr;
2084 assign hit_cg = final_cg;
2085 assign hit_cb = final_cb;
2086 assign hit_hit_type = final_hit_type;
2087 assign hit_dmag = final_dmag;
2088 assign hit_dx = dx_r;
2089 assign hit_dy = dy_r;
2090 assign hit_dz = dz_r;
2091 assign hit_bag = bag_r;
2092 endmodule
2093
2094 // Stage D - NORMAL INV: emit 1/diff_mag (single fp_inv) + ride-along forward
2095 module rtl_normal_div
2096     import raytracer_pkg::*;
2097     #(
2098         parameter int FRAC_BITS = 13,
2099         parameter int WORD      = 27,
2100         parameter int TAG_W     = 7
2101     )(
2102         input logic          clk, rst_n,                // clock, async neg-edge reset
2103         input logic [5:0]    tick,                      // pipe-local tick counter
2104         input logic signed [WORD-1:0] hit_hx, hit_hy, hit_hz, // surface hit point (forwarded)
2105         input logic signed [WORD-1:0] hit_dfx, hit_dfy, hit_dfz, // diff = hit - sphere_center (sphere normal
2106         numerator)
2107         input logic signed [WORD-1:0] hit_tlx, hit_tly, hit_tlz, // to-light vector (forwarded)
2108         input logic signed [WORD-1:0] hit_cr, hit_cg, hit_cb,    // base surface RGB (forwarded)
2109         input logic [2:0]      hit_hit_type,                  // hit type (forwarded)
2110         input logic signed [WORD-1:0] hit_dmag,                // |diff| - sqrt result from Stage C
2111         input logic signed [WORD-1:0] hit_dx, hit_dy, hit_dz,   // parent ray direction (forwarded to Stage
2112         E)
2113         input pipe_baggage_t    hit_bag,                      // baggage
2114         output logic signed [WORD-1:0] norm_hx, norm_hy, norm_hz, // hit point (forwarded)
2115         output logic signed [WORD-1:0] norm_inv_dm,             // 1/|diff| - sphere normal magnitude
2116         inverse (THIS STAGE'S WORK)

```

```

2114     output logic signed [WORD-1:0]  norm_dfx, norm_dfy, norm_dfz,      // diff (forwarded to Stage E for n =
diff·inv_dm)
2115     output logic signed [WORD-1:0]  norm_tlx, norm_tly, norm_tlz,      // to-light vector (forwarded)
2116     output logic signed [WORD-1:0]  norm_cr, norm_cg, norm_cb,        // base color (forwarded)
2117     output logic [2:0]               norm_hit_type,                  // hit type (forwarded)
2118     output logic signed [WORD-1:0]  norm_dx, norm_dy, norm_dz,        // ray direction (forwarded)
2119     output pipe_baggage_t            norm_bag                         // baggage forwarded to Stage E
2120 );
2121 localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;
2122
2123 logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
2124 logic signed [WORD-1:0] diff_x_r, diff_y_r, diff_z_r;
2125 logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
2126 logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
2127 logic signed [WORD-1:0] dx_r, dy_r, dz_r;
2128 logic [2:0]             hit_type_r;
2129 pipe_baggage_t          bag_r;
2130
2131 logic                   inv_start;
2132 assign inv_start = (tick == 6'd0);
2133 logic signed [WORD-1:0] inv_dm_w;
2134 logic                   inv_dm_done;
2135 fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_dm (
2136     .clk(clk), .rst_n(rst_n), .start(inv_start),
2137     .b(hit_dmag), .result(inv_dm_w), .done(inv_dm_done)
2138 );
2139
2140 always_ff @(posedge clk or negedge rst_n) begin
2141     if (!rst_n) begin
2142         hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
2143         diff_x_r <= '0; diff_y_r <= '0; diff_z_r <= '0;
2144         tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
2145         col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
2146         dx_r <= '0; dy_r <= '0; dz_r <= '0;
2147         hit_type_r <= HIT_TYPE_SKY;
2148         bag_r <= '0;
2149     end else if (tick == 6'd0) begin
2150         hit_x_r <= hit_hx; hit_y_r <= hit_hy; hit_z_r <= hit_hz;
2151         diff_x_r <= hit_dfx; diff_y_r <= hit_dfy; diff_z_r <= hit_dfz;
2152         tl_x_r <= hit_tlx; tl_y_r <= hit_tly; tl_z_r <= hit_tlz;
2153         col_r_r <= hit_cr; col_g_r <= hit_cg; col_b_r <= hit_cb;
2154         dx_r <= hit_dx; dy_r <= hit_dy; dz_r <= hit_dz;
2155         hit_type_r <= hit_hit_type;
2156         bag_r <= hit_bag;
2157     end
2158 end
2159
2160 assign norm_hx = hit_x_r;
2161 assign norm_hy = hit_y_r;
2162 assign norm_hz = hit_z_r;
2163 assign norm_inv_dm = inv_dm_w;
2164 assign norm_dfx = diff_x_r;
2165 assign norm_dfy = diff_y_r;
2166 assign norm_dfz = diff_z_r;
2167 assign norm_tlx = tl_x_r;
2168 assign norm_tly = tl_y_r;
2169 assign norm_tlz = tl_z_r;
2170 assign norm_cr = col_r_r;
2171 assign norm_cg = col_g_r;
2172 assign norm_cb = col_b_r;
2173 assign norm_hit_type = hit_type_r;
2174 assign norm_dx = dx_r;
2175 assign norm_dy = dy_r;
2176 assign norm_dz = dz_r;
2177 assign norm_bag = bag_r;
2178 endmodule
2179
2180 // Stage E - REFLECT DIR + LIGHT PREP: recovers normal via fp_mul (diff·1/dm),
2181 // computes refl = d - 2(d·n)n, and the light_dist sqrt for stage F
2182 module rtl_reflect_dir
2183     import raytracer_pkg::*;
2184     #(
2185         parameter int FRAC_BITS = 13,
2186         parameter int WORD      = 27,

```

```

2187     parameter int TAG_W      = 7
2188 )(
2189     input  logic              clk, rst_n,                // clock, async neg-edge reset
2190     input  logic [5:0]        tick,                     // pipe-local tick counter
2191     input  logic signed [WORD-1:0] norm_hx, norm_hy, norm_hz, // surface hit point (forwarded)
2192     input  logic signed [WORD-1:0] norm_inv_dm,          // 1/|diff| from Stage D (used for sphere n
= diff·inv_dm)
2193     input  logic signed [WORD-1:0] norm_dfx, norm_dfy, norm_dfz, // diff = hit - center (for sphere normal)
2194     input  logic signed [WORD-1:0] norm_tlx, norm_tly, norm_tlz, // to-light vector (forwarded; |tl|2 used
for sqrt)
2195     input  logic signed [WORD-1:0] norm_cr, norm_cg, norm_cb,    // base color (forwarded)
2196     input  logic [2:0]          norm_hit_type,                 // hit type (plane override → n = (0,1,0))
2197     input  logic signed [WORD-1:0] norm_dx, norm_dy, norm_dz,   // parent ray direction (for reflection
formula)
2198     input  pipe_baggage_t       norm_bag,                     // baggage
2199     output logic signed [WORD-1:0] refl_hx, refl_hy, refl_hz,    // hit point (forwarded)
2200     output logic signed [WORD-1:0] refl_nx, refl_ny, refl_nz,    // unit normal n (computed here for spheres;
(0,1,0) for plane)
2201     output logic signed [WORD-1:0] refl_tlx, refl_tly, refl_tlz, // to-light vector (forwarded)
2202     output logic signed [WORD-1:0] refl_cr, refl_cg, refl_cb,    // base color (forwarded)
2203     output logic [2:0]          refl_hit_type,                 // hit type (forwarded)
2204     output logic signed [WORD-1:0] refl_ld,                    // |to-light| = √(tl·tl) - light distance (
sqrt computed here)
2205     // refl = d - 2·(d·n)·n, forwarded through H/IJ to K's spawn router.
2206     output logic signed [WORD-1:0] refl_rx, refl_ry, refl_rz,    // reflection vector r - child ray direction
if spawn
2207     output pipe_baggage_t       refl_bag                      // baggage forwarded to Stage F
2208 );
2209     localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;
2210
2211     logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
2212     logic signed [WORD-1:0] inv_dm_r;
2213     logic signed [WORD-1:0] diff_x_r, diff_y_r, diff_z_r;
2214     logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
2215     logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
2216     logic signed [WORD-1:0] dx_r, dy_r, dz_r;
2217     logic [2:0]          hit_type_r;
2218     pipe_baggage_t       bag_r;
2219
2220     // Surface-normal components from the t3..5 diff·inv_dm muls.
2221     logic signed [WORD-1:0] mul_nx_r, mul_ny_r, mul_nz_r;
2222
2223     // Plane override: hit_type==HIT_TYPE_PLANE forces n=(0,1,0).
2224     logic signed [WORD-1:0] norm_x_w, norm_y_w, norm_z_w;
2225     always_comb begin
2226         if (hit_type_r == HIT_TYPE_PLANE) begin
2227             norm_x_w = '0;
2228             norm_y_w = ONE;
2229             norm_z_w = '0;
2230         end else begin
2231             norm_x_w = mul_nx_r;
2232             norm_y_w = mul_ny_r;
2233             norm_z_w = mul_nz_r;
2234         end
2235     end
2236
2237     logic signed [WORD-1:0] prod0_r, prod1_r, prod2_r;
2238     logic signed [WORD-1:0] light_dist_r;
2239
2240     // Reflection-math state: dot prod, doubled dot, r_term, refl.
2241     logic signed [WORD-1:0] p_dn_0_r, p_dn_1_r, p_dn_2_r;
2242     logic signed [WORD-1:0] dot_dn_r, two_dot_r;
2243     logic signed [WORD-1:0] r_term_x_r, r_term_y_r, r_term_z_r;
2244     logic signed [WORD-1:0] refl_x_r, refl_y_r, refl_z_r;
2245
2246     logic signed [WORD-1:0] mul_a, mul_b, mul_result;
2247     fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul (.clk(clk), .a(mul_a), .b(mul_b), .result(mul_result));
2248
2249     always_comb begin
2250         mul_a = '0; mul_b = '0;
2251         case (tick)
2252             6'd2: begin mul_a = diff_x_r; mul_b = inv_dm_r; end
2253             6'd3: begin mul_a = diff_y_r; mul_b = inv_dm_r; end
2254             6'd4: begin mul_a = diff_z_r; mul_b = inv_dm_r; end

```



```

2255         6'd5: begin mul_a = dx_r;      mul_b = mul_nx_r; end
2256         6'd6: begin mul_a = dy_r;      mul_b = mul_ny_r; end
2257         6'd7: begin mul_a = dz_r;      mul_b = mul_nz_r; end
2258         6'd8: begin mul_a = tl_x_r;    mul_b = tl_x_r;    end
2259         6'd9: begin mul_a = tl_y_r;    mul_b = tl_y_r;    end
2260         6'd10: begin mul_a = tl_z_r;    mul_b = tl_z_r;    end
2261         6'd11: begin mul_a = two_dot_r; mul_b = mul_nx_r; end
2262         6'd12: begin mul_a = two_dot_r; mul_b = mul_ny_r; end
2263         6'd13: begin mul_a = two_dot_r; mul_b = mul_nz_r; end
2264         default: begin end
2265     endcase
2266 end
2267
2268 logic          sqrt_start;
2269 logic signed [WORD-1:0] sqrt_in;
2270 logic signed [WORD-1:0] sqrt_out;
2271 logic          sqrt_done;
2272 assign sqrt_start = (tick == 6'd12);
2273 assign sqrt_in    = prod0_r + prod1_r + prod2_r;
2274 fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt (
2275     .clk(clk), .rst_n(rst_n), .start(sqrt_start),
2276     .a(sqrt_in), .result(sqrt_out), .done(sqrt_done)
2277 );
2278
2279 always_ff @(posedge clk or negedge rst_n) begin
2280     if (!rst_n) begin
2281         hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
2282         inv_dm_r <= '0;
2283         diff_x_r <= '0; diff_y_r <= '0; diff_z_r <= '0;
2284         mul_nx_r <= '0; mul_ny_r <= '0; mul_nz_r <= '0;
2285         tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
2286         col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
2287         dx_r <= '0; dy_r <= '0; dz_r <= '0;
2288         hit_type_r <= HIT_TYPE_SKY;
2289         bag_r <= '0;
2290         prod0_r <= '0; prod1_r <= '0; prod2_r <= '0;
2291         light_dist_r <= '0;
2292         p_dn_0_r <= '0; p_dn_1_r <= '0; p_dn_2_r <= '0;
2293         dot_dn_r <= '0; two_dot_r <= '0;
2294         r_term_x_r <= '0; r_term_y_r <= '0; r_term_z_r <= '0;
2295         refl_x_r <= '0; refl_y_r <= '0; refl_z_r <= '0;
2296     end else begin
2297         case (tick)
2298             6'd0: begin
2299                 hit_x_r <= norm_hx; hit_y_r <= norm_hy; hit_z_r <= norm_hz;
2300                 inv_dm_r <= norm_inv_dm;
2301                 diff_x_r <= norm_dfx; diff_y_r <= norm_dfy; diff_z_r <= norm_dfz;
2302                 tl_x_r <= norm_tlx; tl_y_r <= norm_tly; tl_z_r <= norm_tlz;
2303                 col_r_r <= norm_cr; col_g_r <= norm_cg; col_b_r <= norm_cb;
2304                 dx_r <= norm_dx; dy_r <= norm_dy; dz_r <= norm_dz;
2305                 hit_type_r <= norm_hit_type;
2306                 bag_r <= norm_bag;
2307             end
2308             6'd3: mul_nx_r <= mul_result;
2309             6'd4: mul_ny_r <= mul_result;
2310             6'd5: mul_nz_r <= mul_result;
2311             6'd6: p_dn_0_r <= mul_result;
2312             6'd7: p_dn_1_r <= mul_result;
2313             6'd8: p_dn_2_r <= mul_result;
2314             6'd9: begin
2315                 automatic logic signed [WORD-1:0] dn_sum = p_dn_0_r + p_dn_1_r + p_dn_2_r;
2316                 dot_dn_r <= dn_sum;
2317                 two_dot_r <= dn_sum <<< 1;
2318                 prod0_r <= mul_result;
2319             end
2320             6'd10: prod1_r <= mul_result;
2321             6'd11: prod2_r <= mul_result;
2322             6'd12: r_term_x_r <= mul_result;
2323             6'd13: r_term_y_r <= mul_result;
2324             6'd14: begin
2325                 r_term_z_r <= mul_result;
2326                 refl_x_r <= dx_r - r_term_x_r;
2327             end
2328             6'd15: refl_y_r <= dy_r - r_term_y_r;

```

```

2329         6'd16: refl_z_r <= dz_r - r_term_z_r;
2330         default: begin end
2331     endcase
2332     if (sqrt_done) light_dist_r <= sqrt_out;
2333 end
2334 end
2335
2336 assign refl_hx = hit_x_r;
2337 assign refl_hy = hit_y_r;
2338 assign refl_hz = hit_z_r;
2339 assign refl_nx = norm_x_w;
2340 assign refl_ny = norm_y_w;
2341 assign refl_nz = norm_z_w;
2342 assign refl_tlx = tl_x_r;
2343 assign refl_tly = tl_y_r;
2344 assign refl_tlz = tl_z_r;
2345 assign refl_cr = col_r_r;
2346 assign refl_cg = col_g_r;
2347 assign refl_cb = col_b_r;
2348 assign refl_hit_type = hit_type_r;
2349 assign refl_ld = light_dist_r;
2350 assign refl_rx = refl_x_r;
2351 assign refl_ry = refl_y_r;
2352 assign refl_rz = refl_z_r;
2353 assign refl_bag = bag_r;
2354 endmodule
2355
2356 // Stage F - LIGHT INV: emit 1/light_dist (single fp_inv) + ride-along forward.
2357 // Stage G does the ldir = tl_inv_light_dist muls on its mul0 idle slots.
2358 module rtl_light_dir_div
2359     import raytracer_pkg::*;
2360 #(
2361     parameter int FRAC_BITS = 13,
2362     parameter int WORD      = 27,
2363     parameter int TAG_W     = 7
2364 )(
2365     input  logic          clk, rst_n,                // clock, async neg-edge reset
2366     input  logic [5:0]    tick,                      // pipe-local tick counter
2367     input  logic signed [WORD-1:0] refl_hx, refl_hy, refl_hz, // surface hit point (forwarded)
2368     input  logic signed [WORD-1:0] refl_nx, refl_ny, refl_nz, // unit normal (forwarded)
2369     input  logic signed [WORD-1:0] refl_tlx, refl_tly, refl_tlz, // to-light vector (forwarded; for ldir =
    tl_inv_ld)
2370     input  logic signed [WORD-1:0] refl_cr, refl_cg, refl_cb, // base color (forwarded)
2371     input  logic [2:0]    refl_hit_type,                // hit type (forwarded)
2372     input  logic signed [WORD-1:0] refl_ld,                // |to-light| from Stage E - divisor for
    inv_ld
2373     input  logic signed [WORD-1:0] refl_rx, refl_ry, refl_rz, // reflection vector r (forwarded)
2374     input  pipe_baggage_t refl_bag,                      // baggage
2375     output logic signed [WORD-1:0] ldir_hx, ldir_hy, ldir_hz, // hit point (forwarded)
2376     output logic signed [WORD-1:0] ldir_nx, ldir_ny, ldir_nz, // unit normal (forwarded)
2377     output logic signed [WORD-1:0] ldir_tlx, ldir_tly, ldir_tlz, // to-light vector (forwarded)
2378     output logic signed [WORD-1:0] ldir_inv_ld,            // 1/|to-light| - Stage G multiplies tl by
    this (THIS STAGE'S WORK)
2379     output logic signed [WORD-1:0] ldir_ld,                // |to-light| (forwarded for shadow t-
    comparison)
2380     output logic signed [WORD-1:0] ldir_cr, ldir_cg, ldir_cb, // base color (forwarded)
2381     output logic [2:0]    ldir_hit_type,                // hit type (forwarded for shadow self-skip)
2382     output logic signed [WORD-1:0] ldir_rx, ldir_ry, ldir_rz, // reflection vector r (forwarded)
2383     output pipe_baggage_t ldir_bag,                      // baggage forwarded to Stage G
2384 );
2385     logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
2386     logic signed [WORD-1:0] nx_r, ny_r, nz_r;
2387     logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
2388     logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
2389     logic signed [WORD-1:0] light_dist_r;
2390     logic signed [WORD-1:0] refl_x_r, refl_y_r, refl_z_r;
2391     logic [2:0]    hit_type_r;
2392     pipe_baggage_t bag_r;
2393
2394     logic          inv_start;
2395     assign inv_start = (tick == 6'd0);
2396
2397     logic signed [WORD-1:0] inv_ld_w;
2398     logic          inv_ld_done;

```

```

2399 fp_inv #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_inv_ld (
2400     .clk(clk), .rst_n(rst_n), .start(inv_start),
2401     .b(refl_ld), .result(inv_ld_w), .done(inv_ld_done)
2402 );
2403
2404 always_ff @(posedge clk or negedge rst_n) begin
2405     if (!rst_n) begin
2406         hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
2407         nx_r <= '0; ny_r <= '0; nz_r <= '0;
2408         tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
2409         col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
2410         light_dist_r <= '0; hit_type_r <= HIT_TYPE_SKY;
2411         refl_x_r <= '0; refl_y_r <= '0; refl_z_r <= '0;
2412         bag_r <= '0;
2413     end else if (tick == 6'd0) begin
2414         hit_x_r <= refl_hx; hit_y_r <= refl_hy; hit_z_r <= refl_hz;
2415         nx_r <= refl_nx; ny_r <= refl_ny; nz_r <= refl_nz;
2416         tl_x_r <= refl_tlx; tl_y_r <= refl_tly; tl_z_r <= refl_tlz;
2417         col_r_r <= refl_cr; col_g_r <= refl_cg; col_b_r <= refl_cb;
2418         light_dist_r <= refl_ld;
2419         refl_x_r <= refl_rx; refl_y_r <= refl_ry; refl_z_r <= refl_rz;
2420         hit_type_r <= refl_hit_type;
2421         bag_r <= refl_bag;
2422     end
2423 end
2424
2425 assign ldir_hx = hit_x_r;
2426 assign ldir_hy = hit_y_r;
2427 assign ldir_hz = hit_z_r;
2428 assign ldir_nx = nx_r;
2429 assign ldir_ny = ny_r;
2430 assign ldir_nz = nz_r;
2431 assign ldir_tlx = tl_x_r;
2432 assign ldir_tly = tl_y_r;
2433 assign ldir_tlz = tl_z_r;
2434 assign ldir_inv_ld = inv_ld_w;
2435 assign ldir_ld = light_dist_r;
2436 assign ldir_cr = col_r_r;
2437 assign ldir_cg = col_g_r;
2438 assign ldir_cb = col_b_r;
2439 assign ldir_hit_type = hit_type_r;
2440 assign ldir_rx = refl_x_r;
2441 assign ldir_ry = refl_y_r;
2442 assign ldir_rz = refl_z_r;
2443 assign ldir_bag = bag_r;
2444 endmodule
2445
2446 // Stage G - SHADOW TEST: shadow-ray quadratic + visibility check
2447 module rtl_shadow_test
2448     import raytracer_pkg::*;
2449 #(
2450     parameter int FRAC_BITS = 13,
2451     parameter int WORD      = 27,
2452     parameter int TAG_W     = 7
2453 )(
2454     input  logic          clk, rst_n,                // clock, async neg-edge reset
2455     input  logic [5:0]    tick,                      // pipe-local tick counter
2456     input  logic signed [WORD-1:0] ldir_hx, ldir_hy, ldir_hz, // surface hit point (used as shadow ray
origin + bias)
2457     input  logic signed [WORD-1:0] ldir_nx, ldir_ny, ldir_nz, // unit normal (for shadow-origin bias hit +
n/1024)
2458     input  logic signed [WORD-1:0] ldir_tlx, ldir_tly, ldir_tlz, // to-light vector (multiplied by inv_ld →
ldir)
2459     input  logic signed [WORD-1:0] ldir_inv_ld,            // 1/|to-light| (from Stage F)
2460     input  logic signed [WORD-1:0] ldir_ld,                // |to-light| - compared with shadow ray's t
to test occlusion
2461     input  logic signed [WORD-1:0] ldir_cr, ldir_cg, ldir_cb, // base color (forwarded)
2462     input  logic [2:0]    ldir_hit_type,                  // hit type (used for self-skip on shadow
ray)
2463     input  scene_t        in_scene,                      // scene relay (reads .s0/.s1/.cone geometry)
)
2464     input  logic signed [WORD-1:0] ldir_rx, ldir_ry, ldir_rz, // reflection vector r (forwarded)
2465     input  pipe_baggage_t ldir_bag,                      // baggage
2466     output logic signed [WORD-1:0] shdw_hx, shdw_hy, shdw_hz, // hit point (forwarded)

```

```

2467     output logic signed [WORD-1:0] shdw_nx, shdw_ny, shdw_nz, // unit normal (forwarded)
2468     output logic signed [WORD-1:0] shdw_lx, shdw_ly, shdw_lz, // ldir = tl·inv_ld - unit light direction (
computed here)
2469     output logic signed [WORD-1:0] shdw_cr, shdw_cg, shdw_cb, // base color (forwarded)
2470     output logic [2:0] shdw_hit_type, // hit type (forwarded)
2471     output logic shdw_in_shadow, // 1 = light blocked by any primitive (THIS
STAGE'S WORK)
2472     output logic signed [WORD-1:0] shdw_rx, shdw_ry, shdw_rz, // reflection vector r (forwarded to Stage H
)
2473     output pipe_baggage_t shdw_bag // baggage forwarded to Stage H
2474 );
2475     logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
2476     logic signed [WORD-1:0] nx_r, ny_r, nz_r;
2477
2478     logic signed [WORD-1:0] tl_x_r, tl_y_r, tl_z_r;
2479     logic signed [WORD-1:0] inv_ld_r;
2480     logic signed [WORD-1:0] lx_r, ly_r, lz_r;
2481     logic signed [WORD-1:0] light_dist_r;
2482     logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
2483     logic signed [WORD-1:0] refl_x_r, refl_y_r, refl_z_r;
2484     logic [2:0] hit_type_r;
2485     pipe_baggage_t bag_r;
2486
2487     sphere_t sc0_local, sc1_local;
2488     cone_t cone_local;
2489     always_ff @(posedge clk or negedge rst_n) begin
2490         if (!rst_n) begin
2491             sc0_local <= '0;
2492             sc1_local <= '0;
2493             cone_local <= '0;
2494         end else begin
2495             sc0_local <= in_scene.s0;
2496             sc1_local <= in_scene.s1;
2497             cone_local <= in_scene.cone;
2498         end
2499     end
2500
2501     logic signed [WORD-1:0] sh_ox_c, sh_oy_c, sh_oz_c;
2502     logic signed [WORD-1:0] oc0_x_c, oc0_y_c, oc0_z_c;
2503     logic signed [WORD-1:0] oc1_x_c, oc1_y_c, oc1_z_c;
2504     logic signed [WORD-1:0] coc_x_c, coc_y_c, coc_z_c;
2505     assign sh_ox_c = hit_x_r + (nx_r >>> FRAC_BITS);
2506     assign sh_oy_c = hit_y_r + (ny_r >>> FRAC_BITS);
2507     assign sh_oz_c = hit_z_r + (nz_r >>> FRAC_BITS);
2508     assign oc0_x_c = sh_ox_c - sc0_local.x;
2509     assign oc0_y_c = sh_oy_c - sc0_local.y;
2510     assign oc0_z_c = sh_oz_c - sc0_local.z;
2511     assign oc1_x_c = sh_ox_c - sc1_local.x;
2512     assign oc1_y_c = sh_oy_c - sc1_local.y;
2513     assign oc1_z_c = sh_oz_c - sc1_local.z;
2514     assign coc_x_c = sh_ox_c - cone_local.apex_x;
2515     assign coc_y_c = sh_oy_c - cone_local.apex_y;
2516     assign coc_z_c = sh_oz_c - cone_local.apex_z;
2517
2518     logic signed [WORD-1:0] oc0_x_r, oc0_y_r, oc0_z_r;
2519     logic signed [WORD-1:0] oc1_x_r, oc1_y_r, oc1_z_r;
2520     logic signed [WORD-1:0] coc_x_r, coc_y_r, coc_z_r;
2521
2522     // Cone shadow state
2523     logic signed [WORD-1:0] cone_a_r, cone_b_r, cone_c_r;
2524     // Pre-registered negation of cone_b_r. cone_b_r is captured at tick=10
2525     // and stable thereafter, but `~cone_b_r - cone_disc_sqrt_r` (used by
2526     // the cone-num mux at tick=38 and num_w comb) was synthesized as two
2527     // 27-bit carry chains in series, eating ~3 ns. A free-running
2528     // neg_cone_b_r FF breaks one of the chains; the comb consumers see
2529     // `neg_cone_b_r - cone_disc_sqrt_r` instead.
2530     logic signed [WORD-1:0] neg_cone_b_r;
2531     logic signed [WORD-1:0] cone_dy_sq_r, cone_k_dy_sq_r;
2532     logic signed [WORD-1:0] cone_cdx_sq_r, cone_cdy_sq_r, cone_cdz_sq_r, cone_k_cdy_sq_r;
2533     logic signed [WORD-1:0] cone_cdx_dx_r, cone_cdy_dy_r, cone_cdz_dz_r, cone_k_cdy_dy_r;
2534     logic signed [WORD-1:0] cone_b_sq_r, cone_a_c_r;
2535     logic signed [WORD-1:0] cone_disc_sqrt_r;
2536     logic cone_no_disc_r; // disc < 0 → no shadow hit
2537

```

```

2538 logic signed [WORD-1:0] cone_a_ld_r; // a · light_dist
2539 logic signed [WORD-1:0] cone_a_sh_oy_r; // a · sh_origin.y
2540 logic signed [WORD-1:0] cone_a_slab_bot_r; // a · (apex_y - height)
2541 logic signed [WORD-1:0] cone_a_slab_top_r; // a · (apex_y - CONE_APEX_CULL)
2542 logic signed [WORD-1:0] cone_num_ly_r; // num · ldir.y, where num = -b - sqrt_disc
2543 logic cone_in_shadow_r;
2544
2545 // Two fp_muls
2546 logic signed [WORD-1:0] m0_a, m0_b, m0_r;
2547 logic signed [WORD-1:0] m1_a, m1_b, m1_r;
2548 fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul0 (.clk(clk), .a(m0_a), .b(m0_b), .result(m0_r));
2549 fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul1 (.clk(clk), .a(m1_a), .b(m1_b), .result(m1_r));
2550
2551 // Third fp_mul dedicated to cone shadow quadratic
2552 logic signed [WORD-1:0] mc_a, mc_b, mc_r;
2553 fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul_cone_shdw (.clk(clk), .a(mc_a), .b(mc_b), .result(mc_r));
2554
2555 // Free-running neg_cone_b_r relay: track -cone_b_r with 1-cycle lag.
2556 // cone_b_r settles at tick=11 and is stable for the rest of the ray,
2557 // so neg_cone_b_r is valid from tick=12 onward - well before the
2558 // tick=38 cone-num mux that consumes it.
2559 always_ff @(posedge clk or negedge rst_n) begin
2560     if (!rst_n) neg_cone_b_r <= '0;
2561     else neg_cone_b_r <= -cone_b_r;
2562 end
2563
2564 // Per-DSP 1-cycle-delayed tick replicas
2565 (* preserve, dont_merge *) logic [5:0] tick_m0_r;
2566 (* preserve, dont_merge *) logic [5:0] tick_m1_r;
2567 (* preserve, dont_merge *) logic [5:0] tick_mc_r;
2568 always_ff @(posedge clk or negedge rst_n) begin
2569     if (!rst_n) begin
2570         tick_m0_r <= '0;
2571         tick_m1_r <= '0;
2572         tick_mc_r <= '0;
2573     end else begin
2574         tick_m0_r <= tick;
2575         tick_m1_r <= tick;
2576         tick_mc_r <= tick;
2577     end
2578 end
2579
2580 // No fp_inv needed - shadow check uses a-scaled cross-multiplication
2581 // (cone_a_r · light_dist_r, etc.) so the divide is avoided. fp_inv's
2582 // 39c latency would land past the in_shadow_r capture deadline.
2583
2584 // fp_sqrt on cone discriminant. Kicked at t=18 (b_sq_r valid from t=17,
2585 // a_c_r valid from t=18, disc combinational at t=18 - t=18 is the earliest
2586 // possible start without unwinding the cone_k_dy_sq_r race fix in 649c146).
2587 // 21c latency (cocotb-verified) → done at tick=39 → cone_disc_sqrt_r register
2588 // valid from t=40. The num_ly mul and cone_in_shadow_r capture sit at t=40
2589 // and t=42 respectively so they consume the fresh disc.
2590 logic sqrt_cone_start;
2591 logic signed [WORD-1:0] sqrt_cone_in;
2592 logic signed [WORD-1:0] sqrt_cone_out;
2593 logic sqrt_cone_done;
2594 assign sqrt_cone_start = (tick == 6'd18);
2595 assign sqrt_cone_in = cone_b_sq_r - cone_a_c_r;
2596 fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt_cone_shdw (
2597     .clk(clk), .rst_n(rst_n), .start(sqrt_cone_start),
2598     .a(sqrt_cone_in), .result(sqrt_cone_out), .done(sqrt_cone_done)
2599 );
2600 logic signed [WORD-1:0] p_b0_0_r, p_b1_0_r, p_b2_0_r;
2601 logic signed [WORD-1:0] p_c0_0_r, p_c1_0_r, p_c2_0_r;
2602 logic signed [WORD-1:0] half_b0_r, c0_r;
2603 logic signed [WORD-1:0] half_b0_sq_r;
2604 logic signed [WORD-1:0] disc_sqrt0_r;
2605 logic no_hit0_raw_r;
2606
2607 logic signed [WORD-1:0] p_b0_1_r, p_b1_1_r, p_b2_1_r;
2608 logic signed [WORD-1:0] p_c0_1_r, p_c1_1_r, p_c2_1_r;
2609 logic signed [WORD-1:0] half_b1_r, c1_r;
2610 logic signed [WORD-1:0] half_b1_sq_r;
2611 logic signed [WORD-1:0] disc_sqrt1_r;

```

```

2612 logic                                no_hit1_raw_r;
2613
2614 // mul0 t0..2 use upstream ldir_tlx/ldir_inv_ld at t=0 (locals not latched yet)
2615 // and locals at t=1/2 (upstream regs flipped to next ray). Same pattern
2616 // as stage E's diff_inv_dm.
2617 // m0 (sphere 0 shadow) operand mux - driven by tick_m0_r. Case arms
2618 // shift -1 vs the original tick-driven cases; t=0 wraps to II-1=42.
2619 always_comb begin
2620     m0_a = '0; m0_b = '0;
2621     case (tick_m0_r)
2622         6'd42: begin m0_a = ldir_tlx; m0_b = ldir_inv_ld; end // orig t=0
2623         6'd0:  begin m0_a = tl_y_r; m0_b = inv_ld_r; end // orig t=1
2624         6'd1:  begin m0_a = tl_z_r; m0_b = inv_ld_r; end // orig t=2
2625         6'd4:  begin m0_a = lx_r; m0_b = oc0_x_r; end // orig t=5
2626         6'd5:  begin m0_a = ly_r; m0_b = oc0_y_r; end // orig t=6
2627         6'd6:  begin m0_a = lz_r; m0_b = oc0_z_r; end // orig t=7
2628         6'd7:  begin m0_a = oc0_x_r; m0_b = oc0_x_r; end // orig t=8
2629         6'd8:  begin m0_a = oc0_y_r; m0_b = oc0_y_r; end // orig t=9
2630         6'd9:  begin m0_a = oc0_z_r; m0_b = oc0_z_r; end // orig t=10
2631         6'd10: begin m0_a = half_b0_r; m0_b = half_b0_r; end // orig t=11
2632         default: begin end
2633     endcase
2634 end
2635
2636 always_comb begin
2637     m1_a = '0; m1_b = '0;
2638     case (tick_m1_r)
2639         6'd1:  begin m1_a = lx_r; m1_b = oc1_x_r; end // orig t=2
2640         6'd2:  begin m1_a = ly_r; m1_b = oc1_y_r; end // orig t=3
2641         6'd3:  begin m1_a = lz_r; m1_b = oc1_z_r; end // orig t=4
2642         6'd4:  begin m1_a = oc1_x_r; m1_b = oc1_x_r; end // orig t=5
2643         6'd5:  begin m1_a = oc1_y_r; m1_b = oc1_y_r; end // orig t=6
2644         6'd6:  begin m1_a = oc1_z_r; m1_b = oc1_z_r; end // orig t=7
2645         6'd7:  begin m1_a = half_b1_r; m1_b = half_b1_r; end // orig t=8
2646         default: begin end
2647     endcase
2648 end
2649
2650 // mc (cone shadow) operand mux - driven by tick_mc_r
2651 always_comb begin
2652     mc_a = '0; mc_b = '0;
2653     case (tick_mc_r)
2654         6'd2:  begin // orig t=3
2655             mc_a = ly_r; mc_b = ly_r; // ldir.y² → cap t=4
2656         end
2657         6'd4:  begin // orig t=5
2658             mc_a = coc_x_r; mc_b = coc_x_r; // Δ.x² → cap t=6
2659         end
2660         6'd5:  begin // orig t=6
2661             mc_a = coc_y_r; mc_b = coc_y_r; // Δ.y² → cap t=7
2662         end
2663         6'd6:  begin // orig t=7
2664             mc_a = coc_z_r; mc_b = coc_z_r; // Δ.z² → cap t=8
2665         end
2666         6'd7:  begin // orig t=8
2667             mc_a = cone_local.k_sq;
2668             mc_b = cone_cdy_sq_r; // Δk².y² → cap t=9
2669         end
2670         6'd8:  begin // orig t=9
2671             mc_a = coc_x_r; mc_b = lx_r; // Δ.x·D.x → cap t=10
2672         end
2673         6'd9:  begin // orig t=10
2674             mc_a = coc_y_r; mc_b = ly_r; // Δ.y·D.y → cap t=11
2675         end
2676         6'd10: begin // orig t=11
2677             mc_a = coc_z_r; mc_b = lz_r; // Δ.z·D.z → cap t=12
2678         end
2679         6'd11: begin // orig t=12
2680             mc_a = cone_local.k_sq;
2681             mc_b = cone_cdy_dy_r; // k²·Δ(.y·D.y) → cap t=13
2682         end
2683         6'd12: begin // orig t=13
2684             mc_a = cone_local.k_sq;
2685             mc_b = cone_dy_sq_r; // k²·D.y² → cap t=14

```

```

2686         end
2687         6'd14: begin // orig t=15
2688             mc_a = cone_b_w_r; mc_b = cone_b_w_r; // b2 → cap t=16
2689         end
2690         6'd15: begin // orig t=16
2691             mc_a = cone_a_r; mc_b = cone_c_r; // a·c → cap t=17
2692         end
2693         6'd16: begin // orig t=17
2694             mc_a = cone_a_r; mc_b = light_dist_r; // a · ld → cap t=18
2695         end
2696         6'd18: begin // orig t=19
2697             mc_a = cone_a_r; mc_b = sh_oy_c; // a · sh_oy → cap t=20
2698         end
2699         6'd19: begin // orig t=20
2700             mc_a = cone_a_r;
2701             mc_b = apex_y_minus_height_r; // a · slab_bot → cap t=21
2702         end
2703         6'd20: begin // orig t=21
2704             mc_a = cone_a_r;
2705             mc_b = apex_y_minus_cull_r; // a · slab_top → cap t=22
2706         end
2707         6'd39: begin // orig t=40
2708             // Pushed +2 ticks vs the prior schedule: fp_sqrt is 21c (cocotb
2709             // verified), so sqrt_cone_done fires at t=39 and cone_disc_sqrt_r
2710             // only becomes valid during t=40. Reading it at t=38 (the old
2711             // slot) captured the previous ray's disc - visible as thin arcs
2712             // at cone-shadow boundaries.
2713             mc_a = neg_cone_b_r - cone_disc_sqrt_r; // -cone_b pre-registered (saves a carry chain)
2714             mc_b = ly_r; // num · ldir.y → cap t=41
2715         end
2716         default: begin end
2717     endcase
2718 end
2719
2720 localparam signed [WORD-1:0] CONE_APEX_CULL_SHDW = 1 <<< (FRAC_BITS - 5);
2721
2722 logic signed [WORD-1:0] apex_y_minus_height_r, apex_y_minus_cull_r;
2723 always_ff @(posedge clk or negedge rst_n) begin
2724     if (!rst_n) begin
2725         apex_y_minus_height_r <= '0;
2726         apex_y_minus_cull_r <= '0;
2727     end else begin
2728         apex_y_minus_height_r <= cone_local.apex_y - cone_local.height;
2729         apex_y_minus_cull_r <= cone_local.apex_y - CONE_APEX_CULL_SHDW;
2730     end
2731 end
2732
2733 logic signed [WORD-1:0] cone_a_w, cone_b_w, cone_c_w;
2734 localparam signed [WORD-1:0] ONE_FX_LOCAL = 1 <<< FRAC_BITS;
2735 assign cone_a_w = ONE_FX_LOCAL - cone_dy_sq_r - cone_k_dy_sq_r;
2736 assign cone_b_w = cone_cdx_dx_r + cone_cdz_dz_r - cone_k_cdy_dy_r;
2737 assign cone_c_w = cone_cdx_sq_r + cone_cdz_sq_r - cone_k_cdy_sq_r;
2738
2739 logic signed [WORD-1:0] cone_b_w_r;
2740 always_ff @(posedge clk or negedge rst_n) begin
2741     if (!rst_n) cone_b_w_r <= '0;
2742     else cone_b_w_r <= cone_b_w;
2743 end
2744
2745 // sqrt_start at t=15. fp_sqrt is 21c →startdone (cocotb-verified), so done
2746 // fires at tick=36 and disc_sqrt0/1_r capture at edge-end-36 (= valid from
2747 // t=37). The t=37 in_shadow_r capture then reads the new disc in the same
2748 // cycle. Was t=16 before, which made the disc-update and in_shadow_r-capture
2749 // share an edge - in_shadow_r captured the PREVIOUS ray's disc, causing
2750 // thin shadow-boundary arcs in reflections.
2751 logic sqrt_start;
2752 logic signed [WORD-1:0] sqrt0_in, sqrt1_in;
2753 logic signed [WORD-1:0] sqrt0_out, sqrt1_out;
2754 logic sqrt0_done, sqrt1_done;
2755 assign sqrt_start = (tick == 6'd15);
2756 assign sqrt0_in = half_b0_sq_r - c0_r;
2757 assign sqrt1_in = half_b1_sq_r - c1_r;
2758 fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt0 (
2759     .clk(clk), .rst_n(rst_n), .start(sqrt_start),

```

```

2760     .a(sqrt0_in), .result(sqrt0_out), .done(sqrt0_done)
2761 );
2762 fp_sqrt #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_sqrt1 (
2763     .clk(clk), .rst_n(rst_n), .start(sqrt_start),
2764     .a(sqrt1_in), .result(sqrt1_out), .done(sqrt1_done)
2765 );
2766
2767 always_ff @(posedge clk or negedge rst_n) begin
2768     if (!rst_n) begin
2769         hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
2770         nx_r <= '0; ny_r <= '0; nz_r <= '0;
2771         tl_x_r <= '0; tl_y_r <= '0; tl_z_r <= '0;
2772         inv_ld_r <= '0;
2773         lx_r <= '0; ly_r <= '0; lz_r <= '0;
2774         light_dist_r <= '0;
2775         col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
2776         refl_x_r <= '0; refl_y_r <= '0; refl_z_r <= '0;
2777         hit_type_r <= HIT_TYPE_SKY;
2778         bag_r <= '0;
2779         oc0_x_r <= '0; oc0_y_r <= '0; oc0_z_r <= '0;
2780         oc1_x_r <= '0; oc1_y_r <= '0; oc1_z_r <= '0;
2781         coc_x_r <= '0; coc_y_r <= '0; coc_z_r <= '0;
2782         p_b0_0_r <= '0; p_b1_0_r <= '0; p_b2_0_r <= '0;
2783         p_c0_0_r <= '0; p_c1_0_r <= '0; p_c2_0_r <= '0;
2784         half_b0_r <= '0; c0_r <= '0;
2785         half_b0_sq_r <= '0;
2786         disc_sqrt0_r <= '0; no_hit0_raw_r <= 1'b0;
2787         p_b0_1_r <= '0; p_b1_1_r <= '0; p_b2_1_r <= '0;
2788         p_c0_1_r <= '0; p_c1_1_r <= '0; p_c2_1_r <= '0;
2789         half_b1_r <= '0; c1_r <= '0;
2790         half_b1_sq_r <= '0;
2791         disc_sqrt1_r <= '0; no_hit1_raw_r <= 1'b0;
2792         cone_a_r <= '0; cone_b_r <= '0; cone_c_r <= '0;
2793         cone_dy_sq_r <= '0; cone_k_dy_sq_r <= '0;
2794         cone_cdx_sq_r <= '0; cone_cdy_sq_r <= '0; cone_cdz_sq_r <= '0;
2795         cone_k_cdy_sq_r <= '0;
2796         cone_cdx_dx_r <= '0; cone_cdy_dy_r <= '0; cone_cdz_dz_r <= '0;
2797         cone_k_cdy_dy_r <= '0;
2798         cone_b_sq_r <= '0; cone_a_c_r <= '0;
2799         cone_disc_sqrt_r <= '0; cone_no_disc_r <= 1'b0;
2800         cone_a_ld_r <= '0;
2801         cone_a_sh_oy_r <= '0;
2802         cone_a_slab_bot_r <= '0;
2803         cone_a_slab_top_r <= '0;
2804         cone_num_ly_r <= '0;
2805         cone_in_shadow_r <= 1'b0;
2806         in_shadow_r <= 1'b0;
2807     end else begin
2808         case (tick)
2809             6'd0: begin
2810                 hit_x_r <= ldir_hx; hit_y_r <= ldir_hy; hit_z_r <= ldir_hz;
2811                 nx_r <= ldir_nx; ny_r <= ldir_ny; nz_r <= ldir_nz;
2812                 tl_x_r <= ldir_tlx; tl_y_r <= ldir_tly; tl_z_r <= ldir_tlz;
2813                 inv_ld_r <= ldir_inv_ld;
2814                 light_dist_r <= ldir_ld;
2815                 col_r_r <= ldir_cr; col_g_r <= ldir_cg; col_b_r <= ldir_cb;
2816                 refl_x_r <= ldir_rx; refl_y_r <= ldir_ry; refl_z_r <= ldir_rz;
2817                 hit_type_r <= ldir_hit_type;
2818                 bag_r <= ldir_bag;
2819             end
2820             6'd1: begin
2821                 oc0_x_r <= oc0_x_c; oc0_y_r <= oc0_y_c; oc0_z_r <= oc0_z_c;
2822                 oc1_x_r <= oc1_x_c; oc1_y_r <= oc1_y_c; oc1_z_r <= oc1_z_c;
2823                 coc_x_r <= coc_x_c; coc_y_r <= coc_y_c; coc_z_r <= coc_z_c;
2824                 lx_r <= m0_r;
2825             end
2826             6'd2: ly_r <= m0_r;
2827             6'd3: begin
2828                 p_b0_1_r <= m1_r;
2829                 lz_r <= m0_r;
2830             end
2831             6'd4: begin p_b1_1_r <= m1_r; cone_dy_sq_r <= mc_r; end
2832             6'd5: begin p_b2_1_r <= m1_r; end
2833             // cone_k_dy_sq_r capture moved to t=14 (race fix - see mux comment)

```



```

2834         6'd6: begin
2835             p_b0_0_r <= m0_r; p_c0_1_r <= m1_r;
2836             half_b1_r <= p_b0_1_r + p_b1_1_r + p_b2_1_r;
2837             cone_cdx_sq_r <= mc_r;
2838         end
2839         6'd7: begin p_b1_0_r <= m0_r; p_c1_1_r <= m1_r; cone_cdy_sq_r <= mc_r; end
2840         6'd8: begin p_b2_0_r <= m0_r; p_c2_1_r <= m1_r; cone_cdz_sq_r <= mc_r; end
2841         6'd9: begin
2842             p_c0_0_r <= m0_r; half_b1_sq_r <= m1_r;
2843             half_b0_r <= p_b0_0_r + p_b1_0_r + p_b2_0_r;
2844             c1_r <= p_c0_1_r + p_c1_1_r + p_c2_1_r - sc1_local.r_sq;
2845             cone_k_cdy_sq_r <= mc_r;
2846         end
2847         6'd10: begin p_c1_0_r <= m0_r; cone_cdx_dx_r <= mc_r; end
2848         6'd11: begin p_c2_0_r <= m0_r; cone_cdy_dy_r <= mc_r; end
2849         6'd12: begin
2850             half_b0_sq_r <= m0_r;
2851             c0_r <= p_c0_0_r + p_c1_0_r + p_c2_0_r - sc0_local.r_sq;
2852             no_hit1_raw_r <= (half_b1_sq_r < c1_r);
2853             cone_cdz_dz_r <= mc_r;
2854         end
2855         6'd13: cone_k_cdy_dy_r <= mc_r;
2856         6'd14: begin
2857             cone_b_r <= cone_b_w;
2858             cone_c_r <= cone_c_w;
2859             cone_k_dy_sq_r <= mc_r; // race fix capture (was t=5)
2860         end
2861         6'd15: begin
2862             no_hit0_raw_r <= (half_b0_sq_r < c0_r);
2863             cone_a_r <= cone_a_w; // pushed +1 - needs settled cone_k_dy_sq_r
2864         end
2865         6'd16: cone_b_sq_r <= mc_r; // b² kick at tick_mc_r=14
2866
2867         6'd17: begin
2868             cone_a_c_r <= mc_r;
2869             cone_no_disc_r <= (cone_b_sq_r < mc_r);
2870         end
2871         6'd18: cone_a_ld_r <= mc_r;
2872         6'd20: cone_a_sh_oy_r <= mc_r;
2873         6'd21: cone_a_slab_bot_r <= mc_r;
2874         6'd22: cone_a_slab_top_r <= mc_r;
2875         6'd37: in_shadow_r <= in_shadow_w;
2876         6'd41: cone_num_ly_r <= mc_r; // num · ldir.y (mc op moved to t=40)
2877         6'd42: cone_in_shadow_r <= cone_blocks_w;
2878         default: begin end
2879     endcase
2880     if (sqrt0_done) disc_sqrt0_r <= sqrt0_out;
2881     if (sqrt1_done) disc_sqrt1_r <= sqrt1_out;
2882     if (sqrt_cone_done) cone_disc_sqrt_r <= sqrt_cone_out;
2883 end
2884
2885 // Shadow visibility check
2886 logic no_hit0_w, no_hit1_w;
2887 assign no_hit0_w = no_hit0_raw_r || (hit_type_r == HIT_TYPE_S0);
2888 assign no_hit1_w = no_hit1_raw_r || (hit_type_r == HIT_TYPE_S1);
2889
2890 logic signed [WORD-1:0] t0_s0_w, t0_s1_w;
2891 assign t0_s0_w = -half_b0_r - disc_sqrt0_r;
2892 assign t0_s1_w = -half_b1_r - disc_sqrt1_r;
2893
2894 logic s0_blocks_w, s1_blocks_w;
2895 assign s0_blocks_w = !no_hit0_w && (t0_s0_w > 0) && (t0_s0_w < light_dist_r);
2896 assign s1_blocks_w = !no_hit1_w && (t0_s1_w > 0) && (t0_s1_w < light_dist_r);
2897
2898 logic in_shadow_w;
2899 assign in_shadow_w = (hit_type_r != HIT_TYPE_SKY) && (s0_blocks_w || s1_blocks_w);
2900
2901 logic in_shadow_r;
2902
2903 logic signed [WORD-1:0] num_w;
2904 logic signed [WORD-1:0] cone_a_hit_y_w;
2905 assign num_w = neg_cone_b_r - cone_disc_sqrt_r;
2906 assign cone_a_hit_y_w = cone_a_sh_oy_r + cone_num_ly_r;

```

```

2908
2909 logic cone_a_pos, cone_a_neg;
2910 assign cone_a_pos = (cone_a_r > 0);
2911 assign cone_a_neg = (cone_a_r < 0);
2912
2913 logic cone_has_t_w;
2914 assign cone_has_t_w =
2915     (cone_a_pos && (num_w > 0) && (num_w < cone_a_ld_r)) ||
2916     (cone_a_neg && (num_w < 0) && (num_w > cone_a_ld_r));
2917
2918 logic cone_in_slab_w;
2919 assign cone_in_slab_w =
2920     (cone_a_pos && (cone_a_hit_y_w >= cone_a_slab_bot_r) &&
2921         (cone_a_hit_y_w <= cone_a_slab_top_r)) ||
2922     (cone_a_neg && (cone_a_hit_y_w >= cone_a_slab_top_r) &&
2923         (cone_a_hit_y_w <= cone_a_slab_bot_r));
2924
2925 logic cone_blocks_w;
2926 assign cone_blocks_w = (hit_type_r != HIT_TYPE_SKY) &&
2927     (hit_type_r != HIT_TYPE_CONE) && // self-skip
2928     !cone_no_disc_r && // disc 0
2929     cone_has_t_w &&
2930     cone_in_slab_w;
2931
2932 assign shdw_hx = hit_x_r;
2933 assign shdw_hy = hit_y_r;
2934 assign shdw_hz = hit_z_r;
2935 assign shdw_nx = nx_r;
2936 assign shdw_ny = ny_r;
2937 assign shdw_nz = nz_r;
2938 assign shdw_lx = lx_r;
2939 assign shdw_ly = ly_r;
2940 assign shdw_lz = lz_r;
2941 assign shdw_cr = col_r_r;
2942 assign shdw_cg = col_g_r;
2943 assign shdw_cb = col_b_r;
2944 assign shdw_hit_type = hit_type_r;
2945 assign shdw_in_shadow = in_shadow_r || cone_in_shadow_r;
2946 assign shdw_rx = refl_x_r;
2947 assign shdw_ry = refl_y_r;
2948 assign shdw_rz = refl_z_r;
2949 assign shdw_bag = bag_r;
2950 endmodule
2951
2952 // Stage H - SHADE: hit_type/in_shadow mux selects sky / ambient / lit.
2953 // sky: color = passed-through pre-baked gradient
2954 // shadow: color = clamp01(col·AMBIENT)
2955 // else: color = clamp01(col·AMBIENT + col·diff·coeff),
2956 // diff·coeff = max(0, dot(n, ldir))
2957 module rtl_shade_blend
2958     import raytracer_pkg::*;
2959     #(
2960         parameter int FRAC_BITS = 13,
2961         parameter int WORD = 27,
2962         parameter int TAG_W = 7
2963     )(
2964         input logic clk, rst_n, // clock, async neg-edge reset
2965         input logic [5:0] tick, // pipe-local tick counter
2966         input logic signed [WORD-1:0] shdw_hx, shdw_hy, shdw_hz, // hit point (forwarded for spawn origin)
2967         input logic signed [WORD-1:0] shdw_nx, shdw_ny, shdw_nz, // unit normal (used for Lambert dot)
2968         input logic signed [WORD-1:0] shdw_lx, shdw_ly, shdw_lz, // unit light direction (used for Lambert
2969         dot)
2970         input logic signed [WORD-1:0] shdw_cr, shdw_cg, shdw_cb, // base surface RGB
2971         input logic [2:0] shdw_hit_type, // hit type (sky/sphere/plane/cone - gates
2972         which branch)
2973         input logic shdw_in_shadow, // 1 = use ambient only; 0 = ambient +
2974         diffuse
2975         input logic signed [WORD-1:0] shdw_rx, shdw_ry, shdw_rz, // reflection vector r (forwarded for spawn
2976         direction)
2977         input pipe_baggage_t shdw_bag, // baggage
2978         output logic signed [WORD-1:0] shade_cr, shade_cg, shade_cb, // final shaded RGB (clamped) - THIS STAGE'S
2979         WORK
2980         output logic signed [WORD-1:0] shade_hx, shade_hy, shade_hz, // hit point (forwarded for spawn origin)

```

```

2976     output logic signed [WORD-1:0]  shade_rx, shade_ry, shade_rz,      // reflection direction (forwarded for spawn
child ray)
2977     output logic [2:0]                shade_hit_type,                  // hit type (forwarded for spawn-eligibility
check)
2978     output pipe_baggage_t             shade_bag                        // baggage (drop_out latched at tick=15 here
)
2979 );
2980     localparam signed [WORD-1:0] ONE      = 1 <<< FRAC_BITS;
2981     // AMBIENT 5/32 = (col>>3) + (col>>5). One extra adder vs 1/8, kept
2982     // per user pref for the less-crushed shadow tone.
2983     localparam signed [WORD-1:0] AMBIENT = (5 * ONE) / 32;
2984
2985     logic signed [WORD-1:0] hit_x_r, hit_y_r, hit_z_r;
2986     logic signed [WORD-1:0] nx_r, ny_r, nz_r;
2987     logic signed [WORD-1:0] lx_r, ly_r, lz_r;
2988     logic signed [WORD-1:0] col_r_r, col_g_r, col_b_r;
2989     logic signed [WORD-1:0] refl_x_r, refl_y_r, refl_z_r;
2990     logic [2:0]             hit_type_r;
2991     logic                   in_shadow_r;
2992     pipe_baggage_t          bag_r;
2993
2994     logic signed [WORD-1:0] p0_r, p1_r, p2_r;
2995     logic signed [WORD-1:0] diff_coeff_r;
2996     logic signed [WORD-1:0] amb_r_r, amb_g_r, amb_b_r;
2997     logic signed [WORD-1:0] dif_r_r, dif_g_r, dif_b_r;
2998     // pre_clamp*_r holds branch-combined value before clamp01. Splitting
2999     // combine and clamp into t=13/14 breaks a long →tickaddcompare path.
3000     logic signed [WORD-1:0] pre_clamp_r_r, pre_clamp_g_r, pre_clamp_b_r;
3001     logic signed [WORD-1:0] color_r_rg, color_g_rg, color_b_rg;
3002
3003     logic signed [WORD-1:0] mul_a, mul_b, mul_result;
3004     fp_mul #(.FRAC_BITS(FRAC_BITS), .WORD(WORD)) u_mul (.clk(clk), .a(mul_a), .b(mul_b), .result(mul_result));
3005
3006     logic [5:0] tick_local;
3007     always_ff @(posedge clk or negedge rst_n) begin
3008         if (!rst_n) tick_local <= 6'd0;
3009         else        tick_local <= tick;
3010     end
3011
3012     always_comb begin
3013         mul_a = '0; mul_b = '0;
3014         case (tick_local)
3015             6'd1: begin mul_a = nx_r;    mul_b = lx_r;    end
3016             6'd2: begin mul_a = ny_r;    mul_b = ly_r;    end
3017             6'd3: begin mul_a = nz_r;    mul_b = lz_r;    end
3018             6'd8: begin mul_a = col_r_r; mul_b = diff_coeff_r; end
3019             6'd9: begin mul_a = col_g_r; mul_b = diff_coeff_r; end
3020             6'd10: begin mul_a = col_b_r; mul_b = diff_coeff_r; end
3021             default: begin end
3022         endcase
3023     end
3024
3025     logic signed [2*WORD-1:0] amb_r_full_w, amb_g_full_w, amb_b_full_w;
3026     assign amb_r_full_w = col_r_r * AMBIENT;
3027     assign amb_g_full_w = col_g_r * AMBIENT;
3028     assign amb_b_full_w = col_b_r * AMBIENT;
3029     logic signed [WORD-1:0] amb_r_w, amb_g_w, amb_b_w;
3030     assign amb_r_w = amb_r_full_w[WORD-1+FRAC_BITS:FRAC_BITS];
3031     assign amb_g_w = amb_g_full_w[WORD-1+FRAC_BITS:FRAC_BITS];
3032     assign amb_b_w = amb_b_full_w[WORD-1+FRAC_BITS:FRAC_BITS];
3033
3034     function automatic logic signed [WORD-1:0] clamp01(input logic signed [WORD-1:0] v);
3035         if (v < 0)      return '0;
3036         else if (v > ONE) return ONE;
3037         else            return v;
3038     endfunction
3039
3040     always_ff @(posedge clk or negedge rst_n) begin
3041         if (!rst_n) begin
3042             hit_x_r <= '0; hit_y_r <= '0; hit_z_r <= '0;
3043             nx_r <= '0; ny_r <= '0; nz_r <= '0;
3044             lx_r <= '0; ly_r <= '0; lz_r <= '0;
3045             col_r_r <= '0; col_g_r <= '0; col_b_r <= '0;
3046             refl_x_r <= '0; refl_y_r <= '0; refl_z_r <= '0;

```

```

3047     hit_type_r <= HIT_TYPE_SKY; in_shadow_r <= 1'b0;
3048     bag_r <= '0;
3049     p0_r <= '0; p1_r <= '0; p2_r <= '0;
3050     diff_coeff_r <= '0;
3051     amb_r_r <= '0; amb_g_r <= '0; amb_b_r <= '0;
3052     dif_r_r <= '0; dif_g_r <= '0; dif_b_r <= '0;
3053     pre_clamp_r_r <= '0; pre_clamp_g_r <= '0; pre_clamp_b_r <= '0;
3054     color_r_rg <= '0; color_g_rg <= '0; color_b_rg <= '0;
3055 end else begin
3056     case (tick)
3057     6'd0: begin
3058         hit_x_r <= shdw_hx; hit_y_r <= shdw_hy; hit_z_r <= shdw_hz;
3059         nx_r <= shdw_nx; ny_r <= shdw_ny; nz_r <= shdw_nz;
3060         lx_r <= shdw_lx; ly_r <= shdw_ly; lz_r <= shdw_lz;
3061         col_r_r <= shdw_cr; col_g_r <= shdw_cg; col_b_r <= shdw_cb;
3062         refl_x_r <= shdw_rx; refl_y_r <= shdw_ry; refl_z_r <= shdw_rz;
3063         hit_type_r <= shdw_hit_type;
3064         in_shadow_r <= shdw_in_shadow;
3065         bag_r <= shdw_bag;
3066     end
3067     6'd3: p0_r <= mul_result;
3068     6'd4: p1_r <= mul_result;
3069     6'd5: p2_r <= mul_result;
3070     6'd6: begin
3071         automatic logic signed [WORD-1:0] dot_sum = p0_r + p1_r + p2_r;
3072         diff_coeff_r <= dot_sum[WORD-1] ? '0 : dot_sum;
3073     end
3074     6'd1: begin
3075         amb_r_r <= amb_r_w;
3076         amb_g_r <= amb_g_w;
3077         amb_b_r <= amb_b_w;
3078     end
3079     6'd10: dif_r_r <= mul_result;
3080     6'd11: dif_g_r <= mul_result;
3081     6'd12: dif_b_r <= mul_result;
3082     6'd13: begin
3083         if (hit_type_r == HIT_TYPE_SKY) begin
3084             pre_clamp_r_r <= col_r_r;
3085             pre_clamp_g_r <= col_g_r;
3086             pre_clamp_b_r <= col_b_r;
3087         end else if (in_shadow_r) begin
3088             pre_clamp_r_r <= amb_r_r;
3089             pre_clamp_g_r <= amb_g_r;
3090             pre_clamp_b_r <= amb_b_r;
3091         end else begin
3092             pre_clamp_r_r <= amb_r_r + dif_r_r;
3093             pre_clamp_g_r <= amb_g_r + dif_g_r;
3094             pre_clamp_b_r <= amb_b_r + dif_b_r;
3095         end
3096     end
3097     6'd14: begin
3098         color_r_rg <= clamp01(pre_clamp_r_r);
3099         color_g_rg <= clamp01(pre_clamp_g_r);
3100         color_b_rg <= clamp01(pre_clamp_b_r);
3101     end
3102     default: begin end
3103 endcase
3104 end
3105 end
3106
3107 assign shade_cr = color_r_rg;
3108 assign shade_cg = color_g_rg;
3109 assign shade_cb = color_b_rg;
3110 assign shade_hx = hit_x_r;
3111 assign shade_hy = hit_y_r;
3112 assign shade_hz = hit_z_r;
3113 assign shade_rx = refl_x_r;
3114 assign shade_ry = refl_y_r;
3115 assign shade_rz = refl_z_r;
3116 assign shade_hit_type = hit_type_r;
3117 assign shade_bag = bag_r;
3118 endmodule
3119
3120 // =====

```

```

3121 // rtl_pipe - one fixed-II=43 raytracer pipeline. Chains stages A..H with a
3122 // shared `tick` counter (wraps 0..II-1) and a per-pipe load_in handshake.
3123 // PHASE parameter offsets the counter at reset so K parallel rtl_pipe
3124 // instances can be staggered (PHASE = II·i/K). Output bus (drop_out,
3125 // out_tag, out_color_*, out_refl_*, out_meta) is registered at t=15 from
3126 // stage H's settled color + spawn side-band.
3127 // =====
3128 module rtl_pipe
3129     import raytracer_pkg::*;
3130     #(
3131         parameter int FRAC_BITS = 13,
3132         parameter int WORD      = 27,
3133         parameter int WIDTH     = 480,
3134         parameter int HEIGHT    = 360,
3135         parameter int TAG_W     = 7,
3136         parameter int II        = 43,
3137         parameter int PHASE     = 0 // initial tick offset (for K-staggering)
3138     )(
3139         input logic          clk, rst_n, // clock, async neg-edge reset
3140         input logic          load_in,   // 1c pulse to issue a new ray
3141         at t=0 (=sv bit)
3142         input logic signed [WORD-1:0] in_light_x, in_light_y, in_light_z, // light position (batch-stable)
3143         input scene_t          in_scene, // per-pipe scene relay (
3144         geometry + colors + reflect + floor)
3145         input logic [2:0]      max_depth_r, // reflection recursion cap (0
3146         disables)
3147         input logic signed [WORD-1:0] in_K_const, // |light|²·63/64 - sun
3148         threshold
3149         input logic [TAG_W-1:0] in_tag, // lane tag for accumulator BRAM
3150         input ray_meta_t        in_meta, // ray meta: depth, excluded,
3151         total_shift
3152         // Reflection override: in_use_override=1 supplies parent hit + refl dir,
3153         // bypassing stage A's pixel/basis path.
3154         input logic          in_use_override, // 1 = spawn (use in_dir +
3155         in_orig); 0 = primary
3156         input logic signed [WORD-1:0] in_orig_x, in_orig_y, in_orig_z, // ray origin (camera or parent
3157         hit point)
3158         input logic signed [WORD-1:0] in_dir_x, in_dir_y, in_dir_z, // override direction (parent
3159         reflection)
3160         // Shared basis_compute output from rtl_batch_pipe (single instance per
3161         // batch - replaces K-fold redundant per-pipe basis math).
3162         input logic signed [WORD-1:0] in_basis_raw_x, in_basis_raw_y, in_basis_raw_z, // primary ray direction (
3163         right·px + up·py + fwd)
3164         output logic          drop_out, // 1 when stage H emits a real
3165         ray (1-cycle pulse at tick=15)
3166         output logic [TAG_W-1:0] out_tag, // lane tag at the drop_out edge
3167         output logic signed [WORD-1:0] out_color_r, out_color_g, out_color_b, // shaded color at the drop_out
3168         edge
3169         output ray_meta_t        out_meta, // ray meta at the drop_out edge
3170
3171         output logic signed [WORD-1:0] out_hit_x, out_hit_y, out_hit_z, // parent hit point - child's
3172         origin if spawning
3173         output logic signed [WORD-1:0] out_refl_x, out_refl_y, out_refl_z, // reflection direction r -
3174         child's direction if spawning
3175         output logic [2:0]          out_hit_type, // hit type - selects which
3176         reflect_shift to use
3177
3178         output logic          out_spawn_push, // drop_out & spawn_decision (
3179         push into pipe's spawn_slot)
3180         output ray_meta_t        out_spawn_meta, // child's depth+1, excluded,
3181         total_shift+reflect_shift
3182         output logic          at_load_tick // 1 during the cycle tick==0 (
3183         arbiter signal)
3184     );
3185     // Tick counter (wraps at II-1 -> 0)
3186     logic [5:0] tick;
3187     always_ff @(posedge clk or negedge rst_n) begin
3188         if (!rst_n)
3189             tick <= PHASE[5:0];
3190         else if (tick == II[5:0] - 1)
3191             tick <= 6'd0;
3192         else
3193             tick <= tick + 6'd1;
3194     end
3195 end

```

```

3178 logic signed [WORD-1:0] light_x_local, light_y_local, light_z_local;
3179 always_ff @(posedge clk or negedge rst_n) begin
3180     if (!rst_n) begin
3181         light_x_local <= '0;
3182         light_y_local <= '0;
3183         light_z_local <= '0;
3184     end else begin
3185         light_x_local <= in_light_x;
3186         light_y_local <= in_light_y;
3187         light_z_local <= in_light_z;
3188     end
3189 end
3190
3191 // 1-cycle local relay of the scene struct, mirroring the existing
3192 // broadcast-relay pattern (replaces the prior cone_local /
3193 // cone_color_local / cone_norm_factor_local / vu_local quartet).
3194 scene_t scene_local;
3195 logic signed [WORD-1:0] K_const_local;
3196 always_ff @(posedge clk or negedge rst_n) begin
3197     if (!rst_n) begin
3198         scene_local <= '0;
3199         K_const_local <= '0;
3200     end else begin
3201         scene_local <= in_scene;
3202         K_const_local <= in_K_const;
3203     end
3204 end
3205
3206 logic at_load_tick_r;
3207 always_ff @(posedge clk or negedge rst_n) begin
3208     if (!rst_n) at_load_tick_r <= 1'b0;
3209     else at_load_tick_r <= (tick == II[5:0] - 1);
3210 end
3211 assign at_load_tick = at_load_tick_r;
3212
3213 pipe_baggage_t in_bag;
3214 assign in_bag.sv = load_in;
3215 assign in_bag.tag = in_tag;
3216 assign in_bag.meta = in_meta;
3217
3218 // ---- Stage A: RAY GEN ----
3219 logic signed [WORD-1:0] raygen_raw_x, raygen_raw_y, raygen_raw_z, raygen_mag_sq;
3220 logic signed [WORD-1:0] raygen_a_cone;
3221 logic signed [WORD-1:0] raygen_ox, raygen_oy, raygen_oz;
3222 pipe_baggage_t raygen_bag;
3223 rtl_ray_gen #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .WIDTH(WIDTH), .HEIGHT(HEIGHT), .TAG_W(TAG_W)) u_raygen (
3224     .*, .in_scene(scene_local)
3225 );
3226
3227 // ---- Stage B: ISECT PREP ----
3228 logic signed [WORD-1:0] isect_dx, isect_dy, isect_dz;
3229 logic signed [WORD-1:0] isect_ox, isect_oy, isect_oz;
3230 logic signed [WORD-1:0] isect_inv_sa;
3231 logic signed [WORD-1:0] isect_s0_nb, isect_s0_dsqr;
3232 logic isect_s0_nh;
3233 logic signed [WORD-1:0] isect_s1_nb, isect_s1_dsqr;
3234 logic isect_s1_nh;
3235 logic signed [WORD-1:0] isect_inv_pd;
3236 logic signed [WORD-1:0] isect_sa;
3237 logic signed [WORD-1:0] isect_cone_nb, isect_cone_dsqr;
3238 logic isect_cone_nh;
3239 logic signed [WORD-1:0] isect_inv_a_cone;
3240 logic signed [WORD-1:0] isect_cone_dx, isect_cone_dy, isect_cone_dz;
3241 pipe_baggage_t isect_bag;
3242 rtl_isect_prep #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_isect (
3243     .*, .in_scene(scene_local)
3244 );
3245
3246 // ---- Stage C: HIT RESOLVE (isect-resolve + hit-point/normal-prep fused) ----
3247 logic signed [WORD-1:0] hit_hx, hit_hy, hit_hz;
3248 logic signed [WORD-1:0] hit_dfx, hit_dfy, hit_dfz;
3249 logic signed [WORD-1:0] hit_tlx, hit_tly, hit_tlz;
3250 logic signed [WORD-1:0] hit_cr, hit_cg, hit_cb;

```

```

3252 logic [2:0] hit_hit_type;
3253 logic signed [WORD-1:0] hit_dmag;
3254 logic signed [WORD-1:0] hit_dx, hit_dy, hit_dz; // M2c+: parent d pass-through
3255 pipe_baggage_t hit_bag;
3256 rtl_hit_resolve #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_hit (
3257     .*, .in_scene(scene_local)
3258 );
3259
3260 // ---- Stage D: NORMAL INV ----
3261 logic signed [WORD-1:0] norm_hx, norm_hy, norm_hz;
3262 logic signed [WORD-1:0] norm_inv_dm;
3263 logic signed [WORD-1:0] norm_dfx, norm_dfy, norm_dfz;
3264 logic signed [WORD-1:0] norm_tlx, norm_tly, norm_tlz;
3265 logic signed [WORD-1:0] norm_cr, norm_cg, norm_cb;
3266 logic [2:0] norm_hit_type;
3267 logic signed [WORD-1:0] norm_dx, norm_dy, norm_dz;
3268 pipe_baggage_t norm_bag;
3269 rtl_normal_div #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_norm (*);
3270
3271 // ---- Stage E: REFLECT DIR + LIGHT PREP ----
3272 logic signed [WORD-1:0] refl_hx, refl_hy, refl_hz;
3273 logic signed [WORD-1:0] refl_nx, refl_ny, refl_nz;
3274 logic signed [WORD-1:0] refl_tlx, refl_tly, refl_tlz;
3275 logic signed [WORD-1:0] refl_cr, refl_cg, refl_cb;
3276 logic [2:0] refl_hit_type;
3277 logic signed [WORD-1:0] refl_ld;
3278 logic signed [WORD-1:0] refl_rx, refl_ry, refl_rz;
3279 pipe_baggage_t refl_bag;
3280 rtl_reflect_dir #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_refl (*);
3281
3282 // ---- Stage F: LIGHT INV ----
3283 logic signed [WORD-1:0] ldir_hx, ldir_hy, ldir_hz;
3284 logic signed [WORD-1:0] ldir_nx, ldir_ny, ldir_nz;
3285 logic signed [WORD-1:0] ldir_tlx, ldir_tly, ldir_tlz;
3286 logic signed [WORD-1:0] ldir_inv_ld;
3287 logic signed [WORD-1:0] ldir_ld;
3288 logic signed [WORD-1:0] ldir_cr, ldir_cg, ldir_cb;
3289 logic [2:0] ldir_hit_type;
3290 logic signed [WORD-1:0] ldir_rx, ldir_ry, ldir_rz;
3291 pipe_baggage_t ldir_bag;
3292 rtl_light_dir_div #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_ldir (*);
3293
3294 // ---- Stage G: SHADOW TEST ----
3295 logic signed [WORD-1:0] shdw_hx, shdw_hy, shdw_hz;
3296 logic signed [WORD-1:0] shdw_nx, shdw_ny, shdw_nz;
3297 logic signed [WORD-1:0] shdw_lx, shdw_ly, shdw_lz;
3298 logic signed [WORD-1:0] shdw_cr, shdw_cg, shdw_cb;
3299 logic [2:0] shdw_hit_type;
3300 logic shdw_in_shadow;
3301 logic signed [WORD-1:0] shdw_rx, shdw_ry, shdw_rz;
3302 pipe_baggage_t shdw_bag;
3303 rtl_shadow_test #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_shdw (
3304     .*, .in_scene(scene_local)
3305 );
3306
3307 // ---- Stage H: SHADE ----
3308 // H's outputs surface hit + refl + hit_type so rtl_pipe's emit register
3309 // can latch them on the drop edge for the spawn router.
3310 logic signed [WORD-1:0] shade_cr, shade_cg, shade_cb;
3311 logic signed [WORD-1:0] shade_hx, shade_hy, shade_hz;
3312 logic signed [WORD-1:0] shade_rx, shade_ry, shade_rz;
3313 logic [2:0] shade_hit_type;
3314 pipe_baggage_t shade_bag;
3315 rtl_shade_blend #(.FRAC_BITS(FRAC_BITS), .WORD(WORD), .TAG_W(TAG_W)) u_shade (*);
3316
3317 // Emit registers: capture color + spawn side-band at t=15
3318 pipe_baggage_t emit_bag;
3319 logic signed [WORD-1:0] emit_cr, emit_cg, emit_cb;
3320 logic signed [WORD-1:0] emit_hx, emit_hy, emit_hz;
3321 logic signed [WORD-1:0] emit_rx, emit_ry, emit_rz;
3322 logic [2:0] emit_hit_type;
3323 always_ff @(posedge clk or negedge rst_n) begin
3324     if (!rst_n) begin
3325         emit_bag <= '0;

```

```

3326         emit_cr <= '0; emit_cg <= '0; emit_cb <= '0;
3327         emit_hx <= '0; emit_hy <= '0; emit_hz <= '0;
3328         emit_rx <= '0; emit_ry <= '0; emit_rz <= '0;
3329         emit_hit_type <= HIT_TYPE_SKY;
3330     end else if (tick == 6'd15) begin
3331         emit_bag <= shade_bag;
3332         emit_cr <= shade_cr;
3333         emit_cg <= shade_cg;
3334         emit_cb <= shade_cb;
3335         emit_hx <= shade_hx; emit_hy <= shade_hy; emit_hz <= shade_hz;
3336         emit_rx <= shade_rx; emit_ry <= shade_ry; emit_rz <= shade_rz;
3337         emit_hit_type <= shade_hit_type;
3338     end else begin
3339         emit_bag.sv <= 1'b0;
3340     end
3341 end
3342
3343 assign drop_out      = emit_bag.sv;
3344 assign out_tag       = emit_bag.tag[TAG_W-1:0];
3345 assign out_color_r   = emit_cr;
3346 assign out_color_g   = emit_cg;
3347 assign out_color_b   = emit_cb;
3348 assign out_meta      = emit_bag.meta;
3349 assign out_hit_x     = emit_hx;
3350 assign out_hit_y     = emit_hy;
3351 assign out_hit_z     = emit_hz;
3352 assign out_refl_x    = emit_rx;
3353 assign out_refl_y    = emit_ry;
3354 assign out_refl_z    = emit_rz;
3355 assign out_hit_type  = emit_hit_type;
3356
3357 // Spawn decision: combinational. reflect_shift_for_hit is read from
3358 // the registered scene_local so it matches the rest of stage-K's data
3359 // path (and gives Quartus a placement handle near the consumer).
3360 logic [4:0] reflect_shift_for_hit;
3361 logic      spawn_decision;
3362 always_comb begin
3363     unique case (emit_hit_type)
3364         HIT_TYPE_S0: reflect_shift_for_hit = scene_local.s0.reflect_shift;
3365         HIT_TYPE_S1: reflect_shift_for_hit = scene_local.s1.reflect_shift;
3366         HIT_TYPE_CONE: reflect_shift_for_hit = scene_local.cone.reflect_shift;
3367         default:      reflect_shift_for_hit = '0;
3368     endcase
3369 end
3370 assign spawn_decision = (emit_bag.meta.depth < max_depth_r) &&
3371     ((emit_hit_type == HIT_TYPE_S0) ||
3372      (emit_hit_type == HIT_TYPE_S1) ||
3373      (emit_hit_type == HIT_TYPE_CONE)) &&
3374     (reflect_shift_for_hit != '0) &&
3375     ({1'b0, emit_bag.meta.total_shift} +
3376      {1'b0, reflect_shift_for_hit}) < 6'd8);
3377 assign out_spawn_push      = emit_bag.sv & spawn_decision;
3378 assign out_spawn_meta.depth = emit_bag.meta.depth + 3'd1;
3379 assign out_spawn_meta.excluded = emit_hit_type;
3380 assign out_spawn_meta.total_shift = emit_bag.meta.total_shift + reflect_shift_for_hit;
3381 endmodule
3382
3383 // =====
3384 // basis_compute - shared 4c-latency camera basis transform. One instance per
3385 // rtl_batch_pipe (was K replicas inside each per-pipe stage A; the K-fold
3386 // redundancy was unnecessary since all pipes consume the same basis per
3387 // frame). 5 dedicated DSPs (right.px + up.py + fwd, with right.y=0 for no
3388 // roll).
3389 // =====
3390 module basis_compute #(
3391     parameter int FRAC_BITS = 13,
3392     parameter int WORD      = 27,
3393     parameter int WIDTH     = 480,
3394     parameter int HEIGHT    = 360
3395 )(
3396     input logic          clk, rst_n,                // clock, async neg-edge reset
3397     input logic          kick,                      // 1c pulse: latch pix*_in, begin 5-
3398     cycle compute
3399     input logic [$clog2(WIDTH)-1:0] pix_x_in,      // pixel column (0..WIDTH-1)

```



```

3399     input logic [$clog2(HEIGHT)-1:0] pix_y_in, // pixel row (0..HEIGHT-1)
3400     input logic signed [WORD-1:0] right_x_in, right_z_in, // camera basis right.{x,z}; right.y=0 by
no-roll convention
3401     input logic signed [WORD-1:0] up_x_in, up_y_in, up_z_in, // camera basis up
3402     input logic signed [WORD-1:0] fwd_x_in, fwd_y_in, fwd_z_in, // camera basis fwd
3403     output logic signed [WORD-1:0] raw_x_out, raw_y_out, raw_z_out // un-normalized ray direction = right.px
+ up.py + fwd; valid 5c after kick
3404 );
3405     localparam signed [WORD-1:0] ONE = 1 <<< FRAC_BITS;
3406     localparam signed [WORD-1:0] HALF_W = ONE / 2;
3407     localparam signed [WORD-1:0] HALF_H = (HALF_W * HEIGHT) / WIDTH;
3408
3409     localparam int RECIP_SHIFT = 24;
3410     localparam longint INV_W_Q = ((longint'(2 * HALF_W)) <<< RECIP_SHIFT) / WIDTH;
3411     localparam longint INV_H_Q = ((longint'(2 * HALF_H)) <<< RECIP_SHIFT) / HEIGHT;
3412     localparam signed [31:0] INV_W = INV_W_Q[31:0];
3413     localparam signed [31:0] INV_H = INV_H_Q[31:0];
3414
3415     logic [$clog2(WIDTH)-1:0] pix_x_r;
3416     logic [$clog2(HEIGHT)-1:0] pix_y_local_r;
3417     always_ff @(posedge clk or negedge rst_n) begin
3418         if (!rst_n) begin
3419             pix_x_r <= '0;
3420             pix_y_local_r <= '0;
3421         end else if (kick) begin
3422             pix_x_r <= pix_x_in;
3423             pix_y_local_r <= pix_y_in;
3424         end
3425     end
3426
3427     localparam int PXC_W = 14;
3428     logic signed [WORD-1:0] px_s, py_s;
3429     logic signed [WORD+32-1:0] px_prod, py_prod;
3430     logic signed [PXC_W-1:0] px_c_w, py_c_w;
3431     assign px_s = signed'({(WORD-$clog2(WIDTH)){1'b0}}, pix_x_r));
3432     assign py_s = signed'({(WORD-$clog2(HEIGHT)){1'b0}}, pix_y_local_r));
3433     assign px_prod = px_s * INV_W;
3434     assign py_prod = py_s * INV_H;
3435     assign px_c_w = PXC_W'(WORD'(px_prod >>> RECIP_SHIFT) - HALF_W);
3436     assign py_c_w = PXC_W'(HALF_H - WORD'(py_prod >>> RECIP_SHIFT));
3437
3438     logic signed [PXC_W-1:0] px_c_r, py_c_r;
3439     always_ff @(posedge clk or negedge rst_n) begin
3440         if (!rst_n) begin
3441             px_c_r <= '0;
3442             py_c_r <= '0;
3443         end else begin
3444             px_c_r <= px_c_w;
3445             py_c_r <= py_c_w;
3446         end
3447     end
3448
3449     logic signed [WORD-1:0] m_rx_px, m_rz_px, m_ux_py, m_uy_py, m_uz_py;
3450     fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_rx_px (
3451         .clk(clk), .a(right_x_in), .b(px_c_r), .result(m_rx_px));
3452     fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_rz_px (
3453         .clk(clk), .a(right_z_in), .b(px_c_r), .result(m_rz_px));
3454     fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_ux_py (
3455         .clk(clk), .a(up_x_in), .b(py_c_r), .result(m_ux_py));
3456     fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_uy_py (
3457         .clk(clk), .a(up_y_in), .b(py_c_r), .result(m_uy_py));
3458     fp_mul_narrow #(.FRAC_BITS(FRAC_BITS), .WA(WORD), .WB(PXC_W), .WOUT(WORD)) u_mul_uz_py (
3459         .clk(clk), .a(up_z_in), .b(py_c_r), .result(m_uz_py));
3460
3461     logic signed [WORD-1:0] rx_px_r, rz_px_r, ux_py_r, uy_py_r, uz_py_r;
3462     logic signed [WORD-1:0] raw_x_r, raw_y_r, raw_z_r;
3463
3464     always_ff @(posedge clk or negedge rst_n) begin
3465         if (!rst_n) begin
3466             rx_px_r <= '0; rz_px_r <= '0;
3467             ux_py_r <= '0; uy_py_r <= '0; uz_py_r <= '0;
3468             raw_x_r <= '0; raw_y_r <= '0; raw_z_r <= '0;
3469         end else begin
3470             rx_px_r <= m_rx_px;

```

```

3471         rz_px_r <= m_rz_px;
3472         ux_py_r <= m_ux_py;
3473         uy_py_r <= m_uy_py;
3474         uz_py_r <= m_uz_py;
3475         raw_x_r <= rx_px_r + ux_py_r + fwd_x_in;
3476         raw_y_r <= uy_py_r + fwd_y_in;
3477         raw_z_r <= rz_px_r + uz_py_r + fwd_z_in;
3478     end
3479 end
3480
3481 assign raw_x_out = raw_x_r;
3482 assign raw_y_out = raw_y_r;
3483 assign raw_z_out = raw_z_r;
3484 endmodule
3485
3486 // =====
3487 // rtl_batch_pipe - K parallel rtl_pipe instances + issue arbiter + per-pipe
3488 // spawn slots + BRAM write port. Pipes have staggered PHASEs (II·i/K) so at
3489 // most one is at tick=0 each cycle. Issue arbiter picks the pipe at t=0 and
3490 // hands it either a primary (next pix_base + issue_count) or its own pipe's
3491 // spawn slot if valid. Each pipe's drop fires once per ray at its own t=15.
3492 // Spawn router writes reflection children with depth+1 / total_shift
3493 // accumulated into the dropping pipe's own slot; primaries retire to BRAM
3494 // via the wr_en/wr_addr/wr_data port.
3495 // =====
3496 module rtl_batch_pipe
3497     import raytracer_pkg::*;
3498     #(
3499         parameter int K          = 4,
3500         parameter int BATCH      = 120,
3501         parameter int FRAC_BITS  = 13,
3502         parameter int WORD       = 27,
3503         parameter int WIDTH      = 480,
3504         parameter int HEIGHT     = 360
3505     )(
3506         input  logic          clk,                // clock
3507         input  logic          rst_n,              // active-low async reset
3508         input  logic          start,              // 1c pulse: latch all inputs, kick batch
3509         input  logic [$clog2(WIDTH)-1:0] pixel_x, // batch's starting pixel-x (= pix_base)
3510         input  logic [$clog2(HEIGHT)-1:0] pixel_y, // batch's row (pix_y_r)
3511         input  logic signed [WORD-1:0] light_x, light_y, light_z, // light position (latched at start)
3512         input  scene_t        scene,             // bundled primitives + floor (latched at
start)
3513         input  logic [2:0]    max_depth,         // reflection recursion cap (0 disables)
3514         // cam_y is a constant from the wrapper (no Y-translation CSR).
3515         // right.y is hardcoded 0 (no roll) so it's not transmitted.
3516         input  logic signed [WORD-1:0] cam_x, cam_y, cam_z, // camera origin (cam_y locked at 1.5 by
wrapper)
3517         input  logic signed [WORD-1:0] right_x, right_z, // camera basis right.{x,z}; right.y=0 (no
roll)
3518         input  logic signed [WORD-1:0] up_x, up_y, up_z, // camera basis up
3519         input  logic signed [WORD-1:0] fwd_x, fwd_y, fwd_z, // camera basis fwd
3520         input  logic signed [WORD-1:0] K_const, // |light|^2·63/64 - sun-billboard
threshold
3521         output logic          done,              // 1c pulse: batch complete (all pixels
emitted)
3522         // BRAM write port: one indexed write per cycle when a pipe drops a
3523         // completed lane (1c spacing guaranteed by one-hot issue scheduling).
3524         output logic          wr_en,             // write enable to color_mem (in wrapper)
3525         output logic [$clog2(BATCH)-1:0] wr_addr, // lane tag = pixel-within-batch index
3526         output logic [23:0]   wr_data           // saturated 8-bit BGR pixel color
3527     );
3528     localparam int II      = 43;
3529     localparam int TAG_W = (BATCH <= 1) ? 1 : $clog2(BATCH);
3530
3531     logic [$clog2(WIDTH)-1:0] pix_base;
3532     logic [$clog2(HEIGHT)-1:0] pix_y_r;
3533     logic signed [WORD-1:0] in_light_x, in_light_y, in_light_z;
3534     scene_t in_scene;
3535     logic [2:0] max_depth_r;
3536     logic signed [WORD-1:0] cam_x_r, cam_y_r, cam_z_r;
3537     logic signed [WORD-1:0] right_x_r, right_z_r;
3538     logic signed [WORD-1:0] up_x_r, up_y_r, up_z_r;
3539     logic signed [WORD-1:0] fwd_x_r, fwd_y_r, fwd_z_r;

```

```

3540 logic signed [WORD-1:0]      in_K_const;
3541
3542 // Per-pipe broadcast relays. `(* preserve, dont_merge *)` keeps the
3543 // K copies physically distinct so Quartus can place each one near its
3544 // consumer instead of folding them into a single FF with high fanout.
3545 (* preserve, dont_merge *) logic signed [WORD-1:0]      in_light_x_p [K];
3546 (* preserve, dont_merge *) logic signed [WORD-1:0]      in_light_y_p [K];
3547 (* preserve, dont_merge *) logic signed [WORD-1:0]      in_light_z_p [K];
3548 (* preserve, dont_merge *) logic signed [WORD-1:0]      in_K_const_p [K];
3549
3550 (* preserve, dont_merge *) scene_t                      in_scene_p   [K];
3551
3552 // Shared basis_compute outputs (single instance below; broadcast to
3553 // every pipe's stage A).
3554 logic signed [WORD-1:0]      in_basis_raw_x, in_basis_raw_y, in_basis_raw_z;
3555
3556 logic [$clog2(BATCH+1)-1:0] issue_count;
3557 logic [$clog2(BATCH+1)-1:0] complete_count;
3558 logic                        busy;
3559
3560 // K-pipe issue + drop side-bands.
3561 logic                        pipe_load      [K];
3562 logic [$clog2(WIDTH)-1:0]   pipe_pix_x     [K];
3563 logic [TAG_W-1:0]           pipe_tag_in     [K];
3564 ray_meta_t                  pipe_meta_in   [K];
3565
3566 logic                        pipe_use_override [K];
3567 logic signed [WORD-1:0]     pipe_orig_x      [K];
3568 logic signed [WORD-1:0]     pipe_orig_y      [K];
3569 logic signed [WORD-1:0]     pipe_orig_z      [K];
3570 logic signed [WORD-1:0]     pipe_dir_x       [K];
3571 logic signed [WORD-1:0]     pipe_dir_y       [K];
3572 logic signed [WORD-1:0]     pipe_dir_z       [K];
3573 logic                        pipe_drop       [K];
3574 logic [TAG_W-1:0]           pipe_tag_out     [K];
3575 logic signed [WORD-1:0]     pipe_color_r     [K];
3576 logic signed [WORD-1:0]     pipe_color_g     [K];
3577 logic signed [WORD-1:0]     pipe_color_b     [K];
3578 ray_meta_t                  pipe_meta_out   [K];
3579 // Spawn side-band, stable on the drop edge.
3580 logic signed [WORD-1:0]     pipe_out_hit_x   [K];
3581 logic signed [WORD-1:0]     pipe_out_hit_y   [K];
3582 logic signed [WORD-1:0]     pipe_out_hit_z   [K];
3583 logic signed [WORD-1:0]     pipe_out_refl_x  [K];
3584 logic signed [WORD-1:0]     pipe_out_refl_y  [K];
3585 logic signed [WORD-1:0]     pipe_out_refl_z  [K];
3586 logic [2:0]                 pipe_out_hit_type [K];
3587 logic                        pipe_out_spawn_push [K];
3588 ray_meta_t                  pipe_out_spawn_meta [K];
3589 logic                        pipe_at_load    [K];
3590
3591 genvar gi;
3592 generate
3593     for (gi = 0; gi < K; gi++) begin : g_pipe
3594         localparam int PHASE = (II * gi) / K;
3595         rtl_pipe #(
3596             .FRAC_BITS(FRAC_BITS), .WORD(WORD),
3597             .WIDTH(WIDTH), .HEIGHT(HEIGHT),
3598             .TAG_W(TAG_W), .II(II), .PHASE(PHASE)
3599         ) u_pipe (
3600             .*,
3601             .in_light_x      (in_light_x_p[gi]),
3602             .in_light_y      (in_light_y_p[gi]),
3603             .in_light_z      (in_light_z_p[gi]),
3604             .in_K_const      (in_K_const_p[gi]),
3605             .in_scene        (in_scene_p[gi]),
3606             .load_in         (pipe_load[gi]),
3607             .in_tag           (pipe_tag_in[gi]),
3608             .in_meta          (pipe_meta_in[gi]),
3609             .in_use_override (pipe_use_override[gi]),
3610             .in_orig_x       (pipe_orig_x[gi]),
3611             .in_orig_y       (pipe_orig_y[gi]),
3612             .in_orig_z       (pipe_orig_z[gi]),
3613             .in_dir_x        (pipe_dir_x[gi]),

```

```

3614         .in_dir_y      (pipe_dir_y[gi]),
3615         .in_dir_z      (pipe_dir_z[gi]),
3616         .drop_out      (pipe_drop[gi]),
3617         .out_tag       (pipe_tag_out[gi]),
3618         .out_color_r   (pipe_color_r[gi]),
3619         .out_color_g   (pipe_color_g[gi]),
3620         .out_color_b   (pipe_color_b[gi]),
3621         .out_meta      (pipe_meta_out[gi]),
3622         .out_hit_x     (pipe_out_hit_x[gi]),
3623         .out_hit_y     (pipe_out_hit_y[gi]),
3624         .out_hit_z     (pipe_out_hit_z[gi]),
3625         .out_refl_x    (pipe_out_refl_x[gi]),
3626         .out_refl_y    (pipe_out_refl_y[gi]),
3627         .out_refl_z    (pipe_out_refl_z[gi]),
3628         .out_hit_type  (pipe_out_hit_type[gi]),
3629         .out_spawn_push(pipe_out_spawn_push[gi]),
3630         .out_spawn_meta(pipe_out_spawn_meta[gi]),
3631         .at_load_tick  (pipe_at_load[gi])
3632     );
3633     end
3634 endgenerate
3635
3636 // ---- Per-pipe spawn slots -----
3637 //
3638 // Each pipe owns one single-entry spawn slot. When stage H of pipe i
3639 // emits a drop for a ray that hit a sphere/cone with depth < max_depth,
3640 // the spawn router writes into spawn_slot[i] with the child entry
3641 // (same tag, parent hit point as origin, reflection direction, and
3642 // meta with depth+1, excluded=parent's hit_type, total_shift updated).
3643 // The issue arbiter consumes pipe i's slot at pipe i's load_tick,
3644 // preferring the slot over a new primary so chains progress promptly.
3645 //
3646 // Single-entry is sufficient because: (a) stage H of each pipe emits
3647 // at most one drop per II window (the drop is a 1-cycle pulse at a
3648 // fixed tick), and (b) push and pop on the same pipe occur at
3649 // different ticks of the pipe's local II schedule (push at the stage-K
3650 // emit tick, pop at the load_tick), so they never collide on the same
3651 // pipe in the same cycle.
3652 typedef struct packed {
3653     logic [TAG_W-1:0] tag;
3654     logic signed [WORD-1:0] orig_x, orig_y, orig_z;
3655     logic signed [WORD-1:0] dir_x, dir_y, dir_z;
3656     ray_meta_t meta;
3657 } spawn_entry_t;
3658
3659 spawn_entry_t spawn_slot [K];
3660 logic spawn_slot_valid [K];
3661
3662 logic spawn_push [K];
3663 spawn_entry_t spawn_push_data [K];
3664
3665 logic issue_fire;
3666 logic issue_is_spawn;
3667 logic [$clog2(K):0] issue_pidx;
3668 always_comb begin
3669     issue_fire = 1'b0;
3670     issue_is_spawn = 1'b0;
3671     issue_pidx = '0;
3672     for (int p = 0; p < K; p++) begin
3673         if (!issue_fire && pipe_at_load[p] && busy) begin
3674             if (spawn_slot_valid[p]) begin
3675                 issue_fire = 1'b1;
3676                 issue_is_spawn = 1'b1;
3677                 issue_pidx = p[$clog2(K):0];
3678             end else if (issue_count < BATCH) begin
3679                 issue_fire = 1'b1;
3680                 issue_is_spawn = 1'b0;
3681                 issue_pidx = p[$clog2(K):0];
3682             end
3683         end
3684     end
3685     for (int p = 0; p < K; p++) begin
3686         automatic logic use_spawn_p = issue_fire && (p == issue_pidx) && issue_is_spawn;
3687         pipe_load[p] = issue_fire && (p == issue_pidx);

```

```

3688     pipe_pix_x[p]      = pix_base + $clog2(WIDTH)'(issue_count);
3689     pipe_tag_in[p]     = use_spawn_p ? spawn_slot[p].tag
3690                          : issue_count[TAG_W-1:0];
3691     pipe_meta_in[p]    = use_spawn_p ? spawn_slot[p].meta : '0;
3692     pipe_use_override[p] = use_spawn_p;
3693     pipe_orig_x[p]     = use_spawn_p ? spawn_slot[p].orig_x : cam_x_r;
3694     pipe_orig_y[p]     = use_spawn_p ? spawn_slot[p].orig_y : cam_y_r;
3695     pipe_orig_z[p]     = use_spawn_p ? spawn_slot[p].orig_z : cam_z_r;
3696     pipe_dir_x[p]      = spawn_slot[p].dir_x;
3697     pipe_dir_y[p]      = spawn_slot[p].dir_y;
3698     pipe_dir_z[p]      = spawn_slot[p].dir_z;
3699 end
3700 end
3701
3702 logic bm_kick;
3703 assign bm_kick = issue_fire && !issue_is_spawn;
3704 basis_compute #(
3705     .FRAC_BITS(FRAC_BITS), .WORD(WORD),
3706     .WIDTH(WIDTH),         .HEIGHT(HEIGHT)
3707 ) u_basis (
3708     .clk(clk), .rst_n(rst_n),
3709     .kick(bm_kick),
3710     .pix_x_in(pipe_pix_x[issue_pidx]),
3711     .pix_y_in(pix_y_r),
3712     .right_x_in(right_x_r), .right_z_in(right_z_r),
3713     .up_x_in(up_x_r),      .up_y_in(up_y_r),      .up_z_in(up_z_r),
3714     .fwd_x_in(fwd_x_r),    .fwd_y_in(fwd_y_r),    .fwd_z_in(fwd_z_r),
3715     .raw_x_out(in_basis_raw_x), .raw_y_out(in_basis_raw_y), .raw_z_out(in_basis_raw_z)
3716 );
3717
3718 always_ff @(posedge clk or negedge rst_n) begin
3719     if (!rst_n) begin
3720         for (int p = 0; p < K; p++) spawn_slot_valid[p] <= 1'b0;
3721     end else if (start) begin
3722         for (int p = 0; p < K; p++) spawn_slot_valid[p] <= 1'b0;
3723     end else begin
3724         for (int p = 0; p < K; p++) begin
3725             if (spawn_push[p])
3726                 spawn_slot_valid[p] <= 1'b1;
3727             else if (issue_fire && issue_is_spawn && (issue_pidx == p))
3728                 spawn_slot_valid[p] <= 1'b0;
3729         end
3730     end
3731 end
3732 always_ff @(posedge clk) begin
3733     for (int p = 0; p < K; p++)
3734         if (spawn_push[p]) spawn_slot[p] <= spawn_push_data[p];
3735 end
3736
3737 localparam signed [WORD-1:0] COLOR_ONE = 1 <<< FRAC_BITS;
3738 function automatic logic [7:0] to_byte(input logic signed [WORD-1:0] v);
3739     return v[FRAC_BITS] ? 8'hff : v[FRAC_BITS-1 -: 8];
3740 endfunction
3741
3742 // Shared accumulator BRAM. Per-tag slot holds {pending, rgb sum}.
3743 typedef struct packed {
3744     logic    pending;
3745     logic [15:0] r, g, b;    // 16-bit per channel: covers MAX_DEPTH7
3746                             // with power-of-2 attenuation per bounce.
3747 } lane_acc_t;
3748
3749 localparam lane_acc_t LANE_ACC_INIT =
3750     '{pending: 1'b1, r: 16'd0, g: 16'd0, b: 16'd0};
3751
3752 logic    any_drop_w;
3753 logic [TAG_W-1:0] drop_tag_w;
3754 logic signed [WORD-1:0] drop_color_r_w, drop_color_g_w, drop_color_b_w;
3755 ray_meta_t drop_meta_w;
3756 logic [2:0] drop_hit_type_w;
3757 always_comb begin
3758     any_drop_w      = 1'b0;
3759     drop_tag_w      = '0;
3760     drop_color_r_w  = '0;
3761     drop_color_g_w  = '0;

```

```

3762     drop_color_b_w = '0';
3763     drop_meta_w     = '0';
3764     drop_hit_type_w = HIT_TYPE_SKY;
3765     for (int p = 0; p < K; p++) begin
3766         if (pipe_drop[p]) begin
3767             any_drop_w     = 1'b1;
3768             drop_tag_w     = pipe_tag_out[p];
3769             drop_color_r_w = pipe_color_r[p];
3770             drop_color_g_w = pipe_color_g[p];
3771             drop_color_b_w = pipe_color_b[p];
3772             drop_meta_w     = pipe_meta_out[p];
3773             drop_hit_type_w = pipe_out_hit_type[p];
3774         end
3775     end
3776 end
3777
3778 // Init on primary issue only - spawn children must not wipe the
3779 // parent's running sum.
3780 logic          init_lane_w;
3781 logic [TAG_W-1:0] init_tag_w;
3782 always_comb begin
3783     init_lane_w = issue_fire && !issue_is_spawn;
3784     init_tag_w  = issue_count[TAG_W-1:0];
3785 end
3786
3787 (* ramstyle = "no_rw_check, M10K" *)
3788 lane_acc_t acc_mem [BATCH];
3789
3790 // 1-cycle pipeline of drop info (covers BRAM read latency). The
3791 // always_ff below latches the _w signals into _d1 regs on cycle T and
3792 // fires the M10K read; the always_comb blocks consume the _d1 regs +
3793 // acc_rd_d1 on cycle T+1 to produce acc_new, which the same always_ff
3794 // writes back into acc_mem.
3795 logic          drop_d1;
3796 logic [TAG_W-1:0] drop_tag_d1;
3797 logic [7:0]      drop_byte_r_d1, drop_byte_g_d1, drop_byte_b_d1;
3798 ray_meta_t      drop_meta_d1;
3799 logic [2:0]      drop_hit_type_d1;
3800 lane_acc_t       acc_rd_d1;
3801 lane_acc_t       acc_new;
3802 logic [4:0]      drop_reflect_shift_d1;
3803 logic            is_spawn_d1;
3804 logic [7:0]      weighted_r, weighted_g, weighted_b;
3805
3806 always_ff @(posedge clk) begin
3807     drop_d1          <= any_drop_w;
3808     drop_tag_d1      <= drop_tag_w;
3809     drop_byte_r_d1   <= to_byte(drop_color_r_w);
3810     drop_byte_g_d1   <= to_byte(drop_color_g_w);
3811     drop_byte_b_d1   <= to_byte(drop_color_b_w);
3812     drop_meta_d1     <= drop_meta_w;
3813     drop_hit_type_d1 <= drop_hit_type_w;
3814
3815     if (any_drop_w)
3816         acc_rd_d1 <= acc_mem[drop_tag_w];
3817
3818     if (init_lane_w)
3819         acc_mem[init_tag_w] <= LANE_ACC_INIT;
3820     else if (drop_d1)
3821         acc_mem[drop_tag_d1] <= acc_new;
3822 end
3823
3824 always_comb begin
3825     unique case (drop_hit_type_d1)
3826         // Accumulator runs at the merge point - uses the wrapper-level
3827         // single-FF batch-state copy (in_scene), not the per-pipe relay.
3828         HIT_TYPE_S0: drop_reflect_shift_d1 = in_scene.s0.reflect_shift;
3829         HIT_TYPE_S1: drop_reflect_shift_d1 = in_scene.s1.reflect_shift;
3830         HIT_TYPE_CONE: drop_reflect_shift_d1 = in_scene.cone.reflect_shift;
3831         default:      drop_reflect_shift_d1 = '0;
3832     endcase
3833 end
3834
3835 assign is_spawn_d1 = (drop_meta_d1.depth < max_depth_r) &&

```

```

3836             ((drop_hit_type_d1 == HIT_TYPE_S0) ||
3837              (drop_hit_type_d1 == HIT_TYPE_S1) ||
3838              (drop_hit_type_d1 == HIT_TYPE_CONE)) &&
3839             (drop_reflect_shift_d1 != '0) &&
3840             ({1'b0, drop_meta_d1.total_shift} +
3841              {1'b0, drop_reflect_shift_d1}) < 6'd8);
3842
3843 always_comb begin
3844     if (drop_meta_d1.total_shift >= 5'd8) begin
3845         weighted_r = 8'd0;
3846         weighted_g = 8'd0;
3847         weighted_b = 8'd0;
3848     end else begin
3849         weighted_r = drop_byte_r_d1 >> drop_meta_d1.total_shift;
3850         weighted_g = drop_byte_g_d1 >> drop_meta_d1.total_shift;
3851         weighted_b = drop_byte_b_d1 >> drop_meta_d1.total_shift;
3852     end
3853 end
3854
3855 always_comb begin
3856     acc_new.r      = acc_rd_d1.r + 16'(weighted_r);
3857     acc_new.g      = acc_rd_d1.g + 16'(weighted_g);
3858     acc_new.b      = acc_rd_d1.b + 16'(weighted_b);
3859     acc_new.pending = is_spawn_d1;
3860 end
3861
3862 // Single emit pulse - fires on chain termination.
3863 logic      acc_emit_w;
3864 logic [TAG_W-1:0] acc_emit_tag_w;
3865 logic [15:0] acc_emit_r_w, acc_emit_g_w, acc_emit_b_w;
3866 assign acc_emit_w      = drop_d1 && !acc_new.pending;
3867 assign acc_emit_tag_w = drop_tag_d1;
3868 assign acc_emit_r_w    = acc_new.r;
3869 assign acc_emit_g_w    = acc_new.g;
3870 assign acc_emit_b_w    = acc_new.b;
3871
3872 // Spawn router: per-pipe drivers
3873 always_comb begin
3874     for (int p = 0; p < K; p++) begin
3875         spawn_push[p]      = pipe_out_spawn_push[p];
3876         spawn_push_data[p] = '0;
3877         if (pipe_out_spawn_push[p]) begin
3878             spawn_push_data[p].tag      = pipe_tag_out[p];
3879             spawn_push_data[p].orig_x    = pipe_out_hit_x[p];
3880             spawn_push_data[p].orig_y    = pipe_out_hit_y[p];
3881             spawn_push_data[p].orig_z    = pipe_out_hit_z[p];
3882             spawn_push_data[p].dir_x     = pipe_out_refl_x[p];
3883             spawn_push_data[p].dir_y     = pipe_out_refl_y[p];
3884             spawn_push_data[p].dir_z     = pipe_out_refl_z[p];
3885             spawn_push_data[p].meta      = pipe_out_spawn_meta[p];
3886         end
3887     end
3888 end
3889
3890 // 16-bit accumulator → 8-bit channel with high-byte saturation.
3891 function automatic logic [7:0] sat_byte(input logic [15:0] v);
3892     return v[15:8] != 8'd0 ? 8'hff : v[7:0];
3893 endfunction
3894
3895 // Completion counts on chain-terminal emits, not raw drops, so spawn
3896 // children don't double-increment.
3897 logic any_emit;
3898 assign any_emit = acc_emit_w;
3899
3900 // ----- Batch scene state latch -----
3901 // Captures all batch-stable broadcasts on `start` and holds them for
3902 // the duration of the batch. SW invariant: CSRs are only rewritten
3903 // between batches (WAIT_FRAME → write CSRs → FRAME_START), so once
3904 // these latch they don't change until the next start.
3905 always_ff @(posedge clk or negedge rst_n) begin
3906     if (!rst_n) begin
3907         pix_base <= '0; pix_y_r <= '0;
3908         in_light_x <= '0; in_light_y <= '0; in_light_z <= '0;
3909         in_scene  <= '0;

```

```

3910     max_depth_r <= '0;
3911     cam_x_r <= '0; cam_y_r <= '0; cam_z_r <= '0;
3912     right_x_r <= '0; right_z_r <= '0;
3913     up_x_r <= '0; up_y_r <= '0; up_z_r <= '0;
3914     fwd_x_r <= '0; fwd_y_r <= '0; fwd_z_r <= '0;
3915     in_K_const <= '0;
3916 end else if (start) begin
3917     pix_base <= pixel_x;
3918     pix_y_r <= pixel_y;
3919     in_light_x <= light_x; in_light_y <= light_y; in_light_z <= light_z;
3920     in_scene <= scene;
3921     max_depth_r <= max_depth;
3922     cam_x_r <= cam_x; cam_y_r <= cam_y; cam_z_r <= cam_z;
3923     right_x_r <= right_x; right_z_r <= right_z;
3924     up_x_r <= up_x; up_y_r <= up_y; up_z_r <= up_z;
3925     fwd_x_r <= fwd_x; fwd_y_r <= fwd_y; fwd_z_r <= fwd_z;
3926     in_K_const <= K_const;
3927 end
3928 end
3929
3930 // ----- Batch lifecycle counters -----
3931 // busy asserts on start, deasserts when complete_count reaches BATCH.
3932 // issue_count = primaries issued (spawns reuse parent's tag and must
3933 // not increment). complete_count = chain-terminal emits (any_emit).
3934 // done = 1-cycle pulse on the BATCH-th completion.
3935 always_ff @(posedge clk or negedge rst_n) begin
3936     if (!rst_n) begin
3937         issue_count <= '0;
3938         complete_count <= '0;
3939         busy <= 1'b0;
3940         done <= 1'b0;
3941     end else begin
3942         done <= 1'b0;
3943         if (start) begin
3944             issue_count <= '0;
3945             complete_count <= '0;
3946             busy <= 1'b1;
3947         end else if (busy) begin
3948             if (issue_fire && !issue_is_spawn) issue_count <= issue_count + 1;
3949             if (any_emit) complete_count <= complete_count + 1;
3950             if (any_emit && (complete_count == ($clog2(BATCH+1))'(BATCH - 1))) begin
3951                 done <= 1'b1;
3952                 busy <= 1'b0;
3953             end
3954         end
3955     end
3956 end
3957
3958 always_ff @(posedge clk or negedge rst_n) begin
3959     if (!rst_n) begin
3960         for (int i = 0; i < K; i++) begin
3961             in_light_x_p[i] <= '0;
3962             in_light_y_p[i] <= '0;
3963             in_light_z_p[i] <= '0;
3964             in_K_const_p[i] <= '0;
3965             in_scene_p[i] <= '0;
3966         end
3967     end else begin
3968         for (int i = 0; i < K; i++) begin
3969             in_light_x_p[i] <= in_light_x;
3970             in_light_y_p[i] <= in_light_y;
3971             in_light_z_p[i] <= in_light_z;
3972             in_K_const_p[i] <= in_K_const;
3973             in_scene_p[i] <= in_scene;
3974         end
3975     end
3976 end
3977
3978 // BRAM write on terminal chain emit. acc_emit_w is one-hot per cycle.
3979 always_comb begin
3980     wr_en = acc_emit_w;
3981     wr_addr = acc_emit_tag_w;
3982     wr_data = {sat_byte(acc_emit_r_w),
3983               sat_byte(acc_emit_g_w),

```



```

3984         sat_byte(acc_emit_b_w));
3985     end
3986
3987 `ifdef SIMULATION
3988     // Guard the one-drop-per-cycle invariant the BRAM write port assumes.
3989     int drop_count;
3990     always_comb begin
3991         drop_count = 0;
3992         for (int p = 0; p < K; p++) begin
3993             if (pipe_drop[p]) drop_count = drop_count + 1;
3994         end
3995     end
3996     always_ff @(posedge clk) begin
3997         if (rst_n && drop_count > 1)
3998             $error("rtl_batch_pipe: multiple simultaneous drops (single-port BRAM assumption violated)");
3999     end
4000 `endif
4001
4002 endmodule

```

raytracer_hw.tcl

```

1  # TCL File Generated by Component Editor 21.1
2  # Tue Apr 14 17:37:35 EDT 2026
3  # DO NOT MODIFY
4
5
6  #
7  # raytracer "Raytracer" v1.0
8  # 2026.04.14.17:37:35
9  #
10 #
11
12 #
13 # request TCL package from ACDS 16.1
14 #
15 package require -exact qsys 16.1
16
17
18 #
19 # module raytracer
20 #
21 set_module_property DESCRIPTION ""
22 set_module_property NAME raytracer
23 set_module_property VERSION 1.0
24 set_module_property INTERNAL false
25 set_module_property OPAQUE_ADDRESS_MAP true
26 set_module_property AUTHOR ""
27 set_module_property DISPLAY_NAME Raytracer
28 set_module_property INSTITUTE_IN_SYSTEM_MODULE true
29 set_module_property EDITABLE true
30 set_module_property REPORT_TO_TALKBACK false
31 set_module_property ALLOW_GREYBOX_GENERATION false
32 set_module_property REPORT_HIERARCHY false
33
34
35 #
36 # file sets
37 #
38 add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
39 set_fileset_property QUARTUS_SYNTH TOP_LEVEL raytracer_batch_mm
40 set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
41 set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
42 add_fileset_file raytracer.sv SYSTEM_VERILOG PATH raytracer.sv TOP_LEVEL_FILE
43
44
45 #
46 # parameters
47 #
48 add_parameter N INTEGER 480 ""
49 set_parameter_property N DEFAULT_VALUE 480
50 set_parameter_property N DISPLAY_NAME N

```

```

51 set_parameter_property N WIDTH ""
52 set_parameter_property N TYPE INTEGER
53 set_parameter_property N UNITS None
54 set_parameter_property N ALLOWED_RANGES -2147483648:2147483647
55 set_parameter_property N DESCRIPTION ""
56 set_parameter_property N HDL_PARAMETER true
57 add_parameter FRAC_BITS INTEGER 13 ""
58 set_parameter_property FRAC_BITS DEFAULT_VALUE 13
59 set_parameter_property FRAC_BITS DISPLAY_NAME FRAC_BITS
60 set_parameter_property FRAC_BITS WIDTH ""
61 set_parameter_property FRAC_BITS TYPE INTEGER
62 set_parameter_property FRAC_BITS UNITS None
63 set_parameter_property FRAC_BITS ALLOWED_RANGES -2147483648:2147483647
64 set_parameter_property FRAC_BITS DESCRIPTION ""
65 set_parameter_property FRAC_BITS HDL_PARAMETER true
66 add_parameter WORD INTEGER 27 ""
67 set_parameter_property WORD DEFAULT_VALUE 27
68 set_parameter_property WORD DISPLAY_NAME WORD
69 set_parameter_property WORD WIDTH ""
70 set_parameter_property WORD TYPE INTEGER
71 set_parameter_property WORD UNITS None
72 set_parameter_property WORD ALLOWED_RANGES -2147483648:2147483647
73 set_parameter_property WORD DESCRIPTION ""
74 set_parameter_property WORD HDL_PARAMETER true
75 add_parameter WIDTH INTEGER 480 ""
76 set_parameter_property WIDTH DEFAULT_VALUE 480
77 set_parameter_property WIDTH DISPLAY_NAME WIDTH
78 set_parameter_property WIDTH WIDTH ""
79 set_parameter_property WIDTH TYPE INTEGER
80 set_parameter_property WIDTH UNITS None
81 set_parameter_property WIDTH ALLOWED_RANGES -2147483648:2147483647
82 set_parameter_property WIDTH DESCRIPTION ""
83 set_parameter_property WIDTH HDL_PARAMETER true
84 add_parameter HEIGHT INTEGER 360 ""
85 set_parameter_property HEIGHT DEFAULT_VALUE 360
86 set_parameter_property HEIGHT DISPLAY_NAME HEIGHT
87 set_parameter_property HEIGHT WIDTH ""
88 set_parameter_property HEIGHT TYPE INTEGER
89 set_parameter_property HEIGHT UNITS None
90 set_parameter_property HEIGHT ALLOWED_RANGES -2147483648:2147483647
91 set_parameter_property HEIGHT DESCRIPTION ""
92 set_parameter_property HEIGHT HDL_PARAMETER true
93
94
95 #
96 # module assignments
97 #
98 set_module_assignment embeddedsw.dts.group raytracer
99 set_module_assignment embeddedsw.dts.name raytracer
100 set_module_assignment embeddedsw.dts.vendor csee4840
101
102
103 #
104 # display items
105 #
106
107
108 #
109 # connection point clock
110 #
111 add_interface clock clock end
112 set_interface_property clock clockRate 0
113 set_interface_property clock ENABLED true
114 set_interface_property clock EXPORT_OF ""
115 set_interface_property clock PORT_NAME_MAP ""
116 set_interface_property clock CMSIS_SVD_VARIABLES ""
117 set_interface_property clock SVD_ADDRESS_GROUP ""
118
119 add_interface_port clock clk clk Input 1
120
121
122 #
123 # connection point reset
124 #

```

```

125 add_interface reset reset end
126 set_interface_property reset associatedClock clock
127 set_interface_property reset synchronousEdges DEASSERT
128 set_interface_property reset ENABLED true
129 set_interface_property reset EXPORT_OF ""
130 set_interface_property reset PORT_NAME_MAP ""
131 set_interface_property reset CMSIS_SVD_VARIABLES ""
132 set_interface_property reset SVD_ADDRESS_GROUP ""
133
134 add_interface_port reset reset reset Input 1
135
136
137 #
138 # connection point avalon_slave_0
139 #
140 add_interface avalon_slave_0 avalon end
141 set_interface_property avalon_slave_0 addressUnits WORDS
142 set_interface_property avalon_slave_0 associatedClock clock
143 set_interface_property avalon_slave_0 associatedReset reset
144 set_interface_property avalon_slave_0 bitsPerSymbol 8
145 set_interface_property avalon_slave_0 explicitAddressSpan 0
146 set_interface_property avalon_slave_0 holdTime 0
147 set_interface_property avalon_slave_0 maximumPendingReadTransactions 0
148 set_interface_property avalon_slave_0 maximumPendingWriteTransactions 0
149 set_interface_property avalon_slave_0 readLatency 1
150 set_interface_property avalon_slave_0 setupTime 0
151 set_interface_property avalon_slave_0 timingUnits Cycles
152 set_interface_property avalon_slave_0 writeWaitTime 0
153 set_interface_property avalon_slave_0 ENABLED true
154 set_interface_property avalon_slave_0 EXPORT_OF ""
155 set_interface_property avalon_slave_0 PORT_NAME_MAP ""
156 set_interface_property avalon_slave_0 CMSIS_SVD_VARIABLES ""
157 set_interface_property avalon_slave_0 SVD_ADDRESS_GROUP ""
158
159 add_interface_port avalon_slave_0 chipselect chipselect Input 1
160 add_interface_port avalon_slave_0 read read Input 1
161 add_interface_port avalon_slave_0 write write Input 1
162 add_interface_port avalon_slave_0 address address Input 10
163 add_interface_port avalon_slave_0 writedata writedata Input 32
164 add_interface_port avalon_slave_0 readdata readdata Output 32
165 set_interface_assignment avalon_slave_0 embeddedsw.configuration.isFlash 0
166 set_interface_assignment avalon_slave_0 embeddedsw.configuration.isMemoryDevice 0
167 set_interface_assignment avalon_slave_0 embeddedsw.configuration.isNonVolatileStorage 0
168 set_interface_assignment avalon_slave_0 embeddedsw.configuration.isPrintableDevice 0
169
170
171 #
172 # connection point avalon_master_0
173 #
174 # Avalon-MM master for DMA-out. Pushes color results to a SW-programmed
175 # HPS DDR3 phys address via hps_0.f2h_axi_slave. qsys auto-bridges the
176 # Avalon master to the AXI3 slave; the bridge generates AWUSER/AWPROT/
177 # AWCACHE for f2h_axi_slave automatically. Pattern matches `master_secure`
178 # in the DE1-SoC DE1_SOC_Linux_FB reference design.
179 #
180 # Single 240-beat burst per batch (1920 B at 64-bit). The bridge splits
181 # this into AXI3 16-beat bursts internally to satisfy the AXI3 burst
182 # length limit.
183 add_interface avalon_master_0 avalon start
184 set_interface_property avalon_master_0 addressUnits SYMBOLS
185 set_interface_property avalon_master_0 associatedClock clock
186 set_interface_property avalon_master_0 associatedReset reset
187 set_interface_property avalon_master_0 bitsPerSymbol 8
188 set_interface_property avalon_master_0 burstOnBurstBoundariesOnly false
189 set_interface_property avalon_master_0 burstcountUnits WORDS
190 set_interface_property avalon_master_0 doStreamReads false
191 set_interface_property avalon_master_0 doStreamWrites false
192 set_interface_property avalon_master_0 holdTime 0
193 set_interface_property avalon_master_0 linewidthBursts false
194 set_interface_property avalon_master_0 maximumPendingReadTransactions 0
195 # Set to 0 instead of 1: a non-zero pending-write count requires the
196 # master to expose `response` + `writeresponsevalid` signals (Quartus
197 # 21.1 Platform Designer enforces this). With 0, the slave's
198 # waitrequest is the sole write back-pressure mechanism - same as the

```

```

199 # DE1-SoC DE1_SOC_Linux_FB demo's master_secure.
200 set_interface_property avalon_master_0 maximumPendingWriteTransactions 0
201 set_interface_property avalon_master_0 readLatency 0
202 set_interface_property avalon_master_0 readWaitTime 0
203 set_interface_property avalon_master_0 setupTime 0
204 set_interface_property avalon_master_0 timingUnits Cycles
205 set_interface_property avalon_master_0 writeWaitTime 0
206 set_interface_property avalon_master_0 ENABLED true
207 set_interface_property avalon_master_0 EXPORT_OF ""
208 set_interface_property avalon_master_0 PORT_NAME_MAP ""
209 set_interface_property avalon_master_0 CMSIS_SVD_VARIABLES ""
210 set_interface_property avalon_master_0 SVD_ADDRESS_GROUP ""
211
212 add_interface_port avalon_master_0 m_avm_address address Output 32
213 add_interface_port avalon_master_0 m_avm_byteenable byteenable Output 8
214 add_interface_port avalon_master_0 m_avm_write write Output 1
215 add_interface_port avalon_master_0 m_avm_writedata writedata Output 64
216 add_interface_port avalon_master_0 m_avm_burstcount burstcount Output 8
217 add_interface_port avalon_master_0 m_avm_waitrequest waitrequest Input 1

```

soc_system_assignment_defaults.qdf

```

1 set_global_assignment -name OPTIMIZATION_MODE "AGGRESSIVE PERFORMANCE"
2 set_global_assignment -name ADV_NETLIST_OPT_SYNTH_WYSIWYG_REMAP ON
3 set_global_assignment -name PHYSICAL_SYNTHESIS_REGISTER_DUPLICATION ON
4 set_global_assignment -name PHYSICAL_SYNTHESIS_ASYNCHRONOUS_SIGNAL_PIPELINING ON
5 set_global_assignment -name SAVE_DISK_SPACE OFF
6 set_global_assignment -name TIMING_ANALYZER_MULTICORNER_ANALYSIS ON
7 set_global_assignment -name NUM_PARALLEL_PROCESSORS ALL
8 set_global_assignment -name MUX_RESTRUCTURE ON
9 set_global_assignment -name REMOVE_REDUNDANT_LOGIC_CELLS ON
10 set_global_assignment -name AUTO_RESOURCE_SHARING ON
11 set_global_assignment -name OPTIMIZE_POWER_DURING_SYNTHESIS OFF
12 set_global_assignment -name SYNTH_PROTECT_SDC_CONSTRAINT ON
13 set_global_assignment -name PRE_MAPPING_RESYNTHESIS ON
14 set_global_assignment -name ROUTER_CLOCKING_TOPOLOGY_ANALYSIS ON
15 set_global_assignment -name OPTIMIZE_MULTI_CORNER_TIMING OFF
16 set_global_assignment -name OPTIMIZE_POWER_DURING_FITTING OFF
17 set_global_assignment -name PERIPHERY_TO_CORE_PLACEMENT_AND_ROUTING_OPTIMIZATION ON
18 set_global_assignment -name PHYSICAL_SYNTHESIS_COMBO_LOGIC_FOR_AREA ON
19 set_global_assignment -name PHYSICAL_SYNTHESIS_EFFORT EXTRA
20 set_global_assignment -name ROUTER_LCELL_INSERTION_AND_LOGIC_DUPLICATION ON
21 set_global_assignment -name ALM_REGISTER_PACKING_EFFORT HIGH
22
23 # Beneficial skew optimization globally (currently only enabled for SDRAM
24 # in soc_system.qsf). Quartus inserts deliberate clock-tree skew on
25 # destination FFs to borrow setup time from less-critical paths. The
26 # lx_r → light_x_local path already shows -0.478 ns clock skew helping;
27 # this option may push that further.
28 set_global_assignment -name ENABLE_BENEFICIAL_SKEW_OPTIMIZATION ON
29 set_global_assignment -name ENABLE_BENEFICIAL_SKEW_OPTIMIZATION_FOR_NON_GLOBAL_CLOCKS ON
30
31 set_global_assignment -name SEED 2
32
33 # Promote rtl_batch_pipe broadcast source FFs to global routing networks.
34 # Cyclone V has 16 dedicated global networks (CLKCTRL_G*) that can carry
35 # data with very low IC delay across the chip. The default routing for
36 # our 7.13 ns →FFFF broadcast IC may benefit from the dedicated tree.
37 # Caveat: global networks add ~1 ns clock-buffer overhead at the source,
38 # so net gain depends on Quartus's tradeoff. Failed pin assignments
39 # above are removed (rejected by Lite Edition).
40 set_instance_assignment -name GLOBAL_SIGNAL ON -to soc_system0|raytracer_0|core_inst|lx_r
41 set_instance_assignment -name GLOBAL_SIGNAL ON -to soc_system0|raytracer_0|core_inst|ly_r
42 set_instance_assignment -name GLOBAL_SIGNAL ON -to soc_system0|raytracer_0|core_inst|lz_r

```

soc_system.qsys

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <system name="{{$FILENAME}}">
3   <component

```

```

4     name="${FILENAME}"
5     displayName="${FILENAME}"
6     version="1.0"
7     description=""
8     tags=""
9     categories="System" />
10    <parameter name="bonusData"><![CDATA[bonusData
11    {
12        element clk_0
13        {
14            datum _sortIndex
15            {
16                value = "0";
17                type = "int";
18            }
19            datum sopceditor_expanded
20            {
21                value = "1";
22                type = "boolean";
23            }
24        }
25        element hps_0
26        {
27            datum _sortIndex
28            {
29                value = "1";
30                type = "int";
31            }
32        }
33        element pll_0
34        {
35            datum _sortIndex
36            {
37                value = "3";
38                type = "int";
39            }
40        }
41        element raytracer_0
42        {
43            datum _sortIndex
44            {
45                value = "2";
46                type = "int";
47            }
48        }
49        element soc_system
50        {
51            datum _originalDeviceFamily
52            {
53                value = "Cyclone V";
54                type = "String";
55            }
56        }
57        element soc_system
58        {
59            datum _originalDeviceFamily
60            {
61                value = "Cyclone V";
62                type = "String";
63            }
64        }
65    }
66    ]></parameter>
67    <parameter name="clockCrossingAdapter" value="HANDSHAKE" />
68    <parameter name="device" value="5CSEMA5F31C6" />
69    <parameter name="deviceFamily" value="Cyclone V" />
70    <parameter name="deviceSpeedGrade" value="6" />
71    <parameter name="fabricMode" value="QSYS" />
72    <parameter name="generateLegacySim" value="false" />
73    <parameter name="generationId" value="0" />
74    <parameter name="globalResetBus" value="false" />
75    <parameter name="hdlLanguage" value="VERILOG" />
76    <parameter name="hideFromIPCatalog" value="false" />
77    <parameter name="lockedInterfaceDefinition" value="" />

```

```

78 <parameter name="maxAdditionalLatency" value="2" />
79 <parameter name="projectName" value="soc_system.qpf" />
80 <parameter name="sopcBorderPoints" value="false" />
81 <parameter name="systemHash" value="0" />
82 <parameter name="testBenchDutName" value="" />
83 <parameter name="timeStamp" value="0" />
84 <parameter name="useTestBenchNamingPattern" value="false" />
85 <instanceScript></instanceScript>
86 <interface name="clk" internal="clk_0.clk_in" type="clock" dir="end" />
87 <interface name="hps" internal="hps_0.hps_io" type="conduit" dir="end" />
88 <interface name="hps_ddr3" internal="hps_0.memory" type="conduit" dir="end" />
89 <interface name="pll_0_locked" internal="pll_0.locked" type="conduit" dir="end" />
90 <interface name="reset" internal="clk_0.clk_in_reset" type="reset" dir="end" />
91 <module name="clk_0" kind="clock_source" version="21.1" enabled="1">
92   <parameter name="clockFrequency" value="100000000" />
93   <parameter name="clockFrequencyKnown" value="true" />
94   <parameter name="inputClockFrequency" value="0" />
95   <parameter name="resetSynchronousEdges" value="NONE" />
96 </module>
97 <module name="hps_0" kind="altera_hps" version="21.1" enabled="1">
98   <parameter name="ABSTRACT_REAL_COMPARE_TEST" value="false" />
99   <parameter name="ABS_RAM_MEM_INIT_FILENAME" value="meminit" />
100   <parameter name="ACV_PHY_CLK_ADD_FR_PHASE" value="0.0" />
101   <parameter name="AC_PACKAGE_DESKEW" value="false" />
102   <parameter name="AC_ROM_USER_ADD_0" value="0_0000_0000_0000" />
103   <parameter name="AC_ROM_USER_ADD_1" value="0_0000_0000_1000" />
104   <parameter name="ADDR_ORDER" value="0" />
105   <parameter name="ADD_EFFICIENCY_MONITOR" value="false" />
106   <parameter name="ADD_EXTERNAL_SEQ_DEBUG_NIOS" value="false" />
107   <parameter name="ADVANCED_CK_PHASES" value="false" />
108   <parameter name="ADVERTISE_SEQUENCER_SW_BUILD_FILES" value="false" />
109   <parameter name="AFI_DEBUG_INFO_WIDTH" value="32" />
110   <parameter name="ALTMEMPHY_COMPATIBLE_MODE" value="false" />
111   <parameter name="AP_MODE" value="false" />
112   <parameter name="AP_MODE_EN" value="0" />
113   <parameter name="AUTO_DEVICE_SPEEDGRADE" value="6" />
114   <parameter name="AUTO_PD_CYCLES" value="0" />
115   <parameter name="AUTO_POWERDN_EN" value="false" />
116   <parameter name="AVL_DATA_WIDTH_PORT" value="32,32,32,32,32,32" />
117   <parameter name="AVL_MAX_SIZE" value="4" />
118   <parameter name="BONDING_OUT_ENABLED" value="false" />
119   <parameter name="BOOTFROMFPGA_Enable" value="false" />
120   <parameter name="BSEL" value="1" />
121   <parameter name="BSEL_EN" value="false" />
122   <parameter name="BYTE_ENABLE" value="true" />
123   <parameter name="C2P_WRITE_CLOCK_ADD_PHASE" value="0.0" />
124   <parameter name="CALIBRATION_MODE" value="Skip" />
125   <parameter name="CALIB_REG_WIDTH" value="8" />
126   <parameter name="CANO_Mode" value="N/A" />
127   <parameter name="CANO_PinMuxing" value="Unused" />
128   <parameter name="CAN1_Mode" value="N/A" />
129   <parameter name="CAN1_PinMuxing" value="Unused" />
130   <parameter name="CFG_DATA_REORDERING_TYPE" value="INTER_BANK" />
131   <parameter name="CFG_REORDER_DATA" value="true" />
132   <parameter name="CFG_TCCD_NS" value="2.5" />
133   <parameter name="COMMAND_PHASE" value="0.0" />
134   <parameter name="CONTROLLER_LATENCY" value="5" />
135   <parameter name="CORE_DEBUG_CONNECTION" value="EXPORT" />
136   <parameter name="CPORT_TYPE_PORT">Bidirectional,Bidirectional,Bidirectional,Bidirectional,
Bidirectional</parameter>
137   <parameter name="CSEL" value="0" />
138   <parameter name="CSEL_EN" value="false" />
139   <parameter name="CTI_Enable" value="false" />
140   <parameter name="CTL_AUTOPCH_EN" value="false" />
141   <parameter name="CTL_CMD_QUEUE_DEPTH" value="8" />
142   <parameter name="CTL_CSR_CONNECTION" value="INTERNAL_JTAG" />
143   <parameter name="CTL_CSR_ENABLED" value="false" />
144   <parameter name="CTL_CSR_READ_ONLY" value="1" />
145   <parameter name="CTL_DEEP_POWERDN_EN" value="false" />
146   <parameter name="CTL_DYNAMIC_BANK_ALLOCATION" value="false" />
147   <parameter name="CTL_DYNAMIC_BANK_NUM" value="4" />
148   <parameter name="CTL_ECC_AUTO_CORRECTION_ENABLED" value="false" />
149   <parameter name="CTL_ECC_ENABLED" value="false" />
150   <parameter name="CTL_ENABLE_BURST_INTERRUPT" value="false" />

```

```

151 <parameter name="CTL_ENABLE_BURST_TERMINATE" value="false" />
152 <parameter name="CTL_HRB_ENABLED" value="false" />
153 <parameter name="CTL_LOOK_AHEAD_DEPTH" value="4" />
154 <parameter name="CTL_SELF_REFRESH_EN" value="false" />
155 <parameter name="CTL_USR_REFRESH_EN" value="false" />
156 <parameter name="CTL_ZQCAL_EN" value="false" />
157 <parameter name="CUT_NEW_FAMILY_TIMING" value="true" />
158 <parameter name="DAT_DATA_WIDTH" value="32" />
159 <parameter name="DEBUGAPB_Enable" value="false" />
160 <parameter name="DEBUG_MODE" value="false" />
161 <parameter name="DEVICE_DEPTH" value="1" />
162 <parameter name="DEVICE_FAMILY_PARAM" value="" />
163 <parameter name="DISABLE_CHILD_MESSAGING" value="false" />
164 <parameter name="DISCRETE_FLY_BY" value="true" />
165 <parameter name="DLL_SHARING_MODE" value="None" />
166 <parameter name="DMA_Enable">No,No,No,No,No,No,No,No</parameter>
167 <parameter name="DQS_DQSN_MODE" value="DIFFERENTIAL" />
168 <parameter name="DQ_INPUT_REG_USE_CLKN" value="false" />
169 <parameter name="DUPLICATE_AC" value="false" />
170 <parameter name="ED_EXPORT_SEQ_DEBUG" value="false" />
171 <parameter name="EMACO_Mode" value="N/A" />
172 <parameter name="EMACO_PTP" value="false" />
173 <parameter name="EMACO_PinMuxing" value="Unused" />
174 <parameter name="EMAC1_Mode" value="RGMII" />
175 <parameter name="EMAC1_PTP" value="false" />
176 <parameter name="EMAC1_PinMuxing" value="HPS I/O Set 0" />
177 <parameter name="ENABLE_ABS_RAM_MEM_INIT" value="false" />
178 <parameter name="ENABLE_BONDING" value="false" />
179 <parameter name="ENABLE_BURST_MERGE" value="false" />
180 <parameter name="ENABLE_CTRL_AVALON_INTERFACE" value="true" />
181 <parameter name="ENABLE_DELAY_CHAIN_WRITE" value="false" />
182 <parameter name="ENABLE_EMIT_BFM_MASTER" value="false" />
183 <parameter name="ENABLE_EXPORT_SEQ_DEBUG_BRIDGE" value="false" />
184 <parameter name="ENABLE_EXTRA_REPORTING" value="false" />
185 <parameter name="ENABLE_ISS_PROBES" value="false" />
186 <parameter name="ENABLE_NON_DESTRUCTIVE_CALIB" value="false" />
187 <parameter name="ENABLE_NON_DES_CAL" value="false" />
188 <parameter name="ENABLE_NON_DES_CAL_TEST" value="false" />
189 <parameter name="ENABLE_SEQUENCER_MARGINING_ON_BY_DEFAULT" value="false" />
190 <parameter name="ENABLE_USER_ECC" value="false" />
191 <parameter name="EXPORT_AFI_HALF_CLK" value="false" />
192 <parameter name="EXTRA_SETTINGS" value="" />
193 <parameter name="F2H_AXI_CLOCK_FREQ" value="100000000" />
194 <parameter name="F2H_SDRAM0_CLOCK_FREQ" value="100" />
195 <parameter name="F2H_SDRAM1_CLOCK_FREQ" value="100" />
196 <parameter name="F2H_SDRAM2_CLOCK_FREQ" value="100" />
197 <parameter name="F2H_SDRAM3_CLOCK_FREQ" value="100" />
198 <parameter name="F2H_SDRAM4_CLOCK_FREQ" value="100" />
199 <parameter name="F2H_SDRAM5_CLOCK_FREQ" value="100" />
200 <parameter name="F2SCLK_COLD_RST_Enable" value="false" />
201 <parameter name="F2SCLK_DBG_RST_Enable" value="false" />
202 <parameter name="F2SCLK_PERIPHCLK_Enable" value="false" />
203 <parameter name="F2SCLK_PERIPHCLK_FREQ" value="0" />
204 <parameter name="F2SCLK_SDRAMCLK_Enable" value="false" />
205 <parameter name="F2SCLK_SDRAMCLK_FREQ" value="0" />
206 <parameter name="F2SCLK_WARM_RST_Enable" value="false" />
207 <parameter name="F2SDRAM_Type" value="" />
208 <parameter name="F2SDRAM_Width" value="" />
209 <parameter name="F2SINTERRUPT_Enable" value="false" />
210 <parameter name="F2S_Width" value="2" />
211 <parameter name="FIX_READ_LATENCY" value="8" />
212 <parameter name="FORCED_NON_LDC_ADDR_CMD_MEM_CK_INVERT" value="false" />
213 <parameter name="FORCED_NUM_WRITE_FR_CYCLE_SHIFTS" value="0" />
214 <parameter name="FORCE_DQS_TRACKING" value="AUTO" />
215 <parameter name="FORCE_MAX_LATENCY_COUNT_WIDTH" value="0" />
216 <parameter name="FORCE_SEQUENCER_TCL_DEBUG_MODE" value="false" />
217 <parameter name="FORCE_SHADOW_REGS" value="AUTO" />
218 <parameter name="FORCE_SYNTHESIS_LANGUAGE" value="" />
219 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMACO_RX_CLK_IN" value="100" />
220 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMACO_TX_CLK_IN" value="100" />
221 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_RX_CLK_IN" value="100" />
222 <parameter name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC1_TX_CLK_IN" value="100" />
223 <parameter
224     name="FPGA_PERIPHERAL_INPUT_CLOCK_FREQ_EMAC_PTP_REF_CLOCK"

```



```

293 <parameter name="MEM_DQ_PER_DQS" value="8" />
294 <parameter name="MEM_DQ_WIDTH" value="32" />
295 <parameter name="MEM_DRV_STR" value="RZQ/7" />
296 <parameter name="MEM_FORMAT" value="DISCRETE" />
297 <parameter name="MEM_GUARANTEED_WRITE_INIT" value="false" />
298 <parameter name="MEM_IF_BOARD_BASE_DELAY" value="10" />
299 <parameter name="MEM_IF_DM_PINS_EN" value="true" />
300 <parameter name="MEM_IF_DQSN_EN" value="true" />
301 <parameter name="MEM_IF_SIM_VALID_WINDOW" value="0" />
302 <parameter name="MEM_INIT_EN" value="false" />
303 <parameter name="MEM_INIT_FILE" value="" />
304 <parameter name="MEM_MIRROR_ADDRESSING" value="0" />
305 <parameter name="MEM_NUMBER_OF_DIMMS" value="1" />
306 <parameter name="MEM_NUMBER_OF_RANKS_PER_DEVICE" value="1" />
307 <parameter name="MEM_NUMBER_OF_RANKS_PER_DIMM" value="1" />
308 <parameter name="MEM_PD" value="DLL off" />
309 <parameter name="MEM_RANK_MULTIPLICATION_FACTOR" value="1" />
310 <parameter name="MEM_ROW_ADDR_WIDTH" value="15" />
311 <parameter name="MEM_RTT_NOM" value="RZQ/4" />
312 <parameter name="MEM_RTT_WR" value="RZQ/4" />
313 <parameter name="MEM_SRT" value="Normal" />
314 <parameter name="MEM_TCL" value="11" />
315 <parameter name="MEM_TFAW_NS" value="30.0" />
316 <parameter name="MEM_TINIT_US" value="500" />
317 <parameter name="MEM_TMRD_CK" value="4" />
318 <parameter name="MEM_TRAS_NS" value="35.0" />
319 <parameter name="MEM_TRCD_NS" value="13.75" />
320 <parameter name="MEM_TREFFI_US" value="7.8" />
321 <parameter name="MEM_TRFC_NS" value="260.0" />
322 <parameter name="MEM_TRP_NS" value="13.75" />
323 <parameter name="MEM_TRRD_NS" value="7.7" />
324 <parameter name="MEM_TRTP_NS" value="7.5" />
325 <parameter name="MEM_TWR_NS" value="15.0" />
326 <parameter name="MEM_TWTR" value="4" />
327 <parameter name="MEM_USER_LEVELING_MODE" value="Leveling" />
328 <parameter name="MEM_VENDOR" value="JEDEC" />
329 <parameter name="MEM_VERBOSE" value="true" />
330 <parameter name="MEM_VOLTAGE" value="1.5V DDR3" />
331 <parameter name="MEM_WTCL" value="8" />
332 <parameter name="MPU_EVENTS_Enable" value="false" />
333 <parameter name="MRS_MIRROR_PING_PONG_ATSO" value="false" />
334 <parameter name="MULTICAST_EN" value="false" />
335 <parameter name="NAND_Mode" value="N/A" />
336 <parameter name="NAND_PinMuxing" value="Unused" />
337 <parameter name="NEXTGEN" value="true" />
338 <parameter name="NIOS_ROM_DATA_WIDTH" value="32" />
339 <parameter name="NUM_DLL_SHARING_INTERFACES" value="1" />
340 <parameter name="NUM_EXTRA_REPORT_PATH" value="10" />
341 <parameter name="NUM_OCT_SHARING_INTERFACES" value="1" />
342 <parameter name="NUM_OF_PORTS" value="1" />
343 <parameter name="NUM_PLL_SHARING_INTERFACES" value="1" />
344 <parameter name="OCT_SHARING_MODE" value="None" />
345 <parameter name="P2C_READ_CLOCK_ADD_PHASE" value="0.0" />
346 <parameter name="PACKAGE_DESKEW" value="false" />
347 <parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM" value="" />
348 <parameter name="PARSE_FRIENDLY_DEVICE_FAMILY_PARAM_VALID" value="false" />
349 <parameter name="PHY_CSR_CONNECTION" value="INTERNAL_JTAG" />
350 <parameter name="PHY_CSR_ENABLED" value="false" />
351 <parameter name="PHY_ONLY" value="false" />
352 <parameter name="PINGPONGPHY_EN" value="false" />
353 <parameter name="PLL_ADDR_CMD_CLK_DIV_PARAM" value="0" />
354 <parameter name="PLL_ADDR_CMD_CLK_FREQ_PARAM" value="0.0" />
355 <parameter name="PLL_ADDR_CMD_CLK_FREQ_SIM_STR_PARAM" value="" />
356 <parameter name="PLL_ADDR_CMD_CLK_MULT_PARAM" value="0" />
357 <parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_PARAM" value="0" />
358 <parameter name="PLL_ADDR_CMD_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
359 <parameter name="PLL_AFI_CLK_DIV_PARAM" value="0" />
360 <parameter name="PLL_AFI_CLK_FREQ_PARAM" value="0.0" />
361 <parameter name="PLL_AFI_CLK_FREQ_SIM_STR_PARAM" value="" />
362 <parameter name="PLL_AFI_CLK_MULT_PARAM" value="0" />
363 <parameter name="PLL_AFI_CLK_PHASE_PS_PARAM" value="0" />
364 <parameter name="PLL_AFI_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
365 <parameter name="PLL_AFI_HALF_CLK_DIV_PARAM" value="0" />
366 <parameter name="PLL_AFI_HALF_CLK_FREQ_PARAM" value="0.0" />

```

```

367 <parameter name="PLL_AFI_HALF_CLK_FREQ_SIM_STR_PARAM" value="" />
368 <parameter name="PLL_AFI_HALF_CLK_MULT_PARAM" value="0" />
369 <parameter name="PLL_AFI_HALF_CLK_PHASE_PS_PARAM" value="0" />
370 <parameter name="PLL_AFI_HALF_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
371 <parameter name="PLL_AFI_PHY_CLK_DIV_PARAM" value="0" />
372 <parameter name="PLL_AFI_PHY_CLK_FREQ_PARAM" value="0.0" />
373 <parameter name="PLL_AFI_PHY_CLK_FREQ_SIM_STR_PARAM" value="" />
374 <parameter name="PLL_AFI_PHY_CLK_MULT_PARAM" value="0" />
375 <parameter name="PLL_AFI_PHY_CLK_PHASE_PS_PARAM" value="0" />
376 <parameter name="PLL_AFI_PHY_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
377 <parameter name="PLL_C2P_WRITE_CLK_DIV_PARAM" value="0" />
378 <parameter name="PLL_C2P_WRITE_CLK_FREQ_PARAM" value="0.0" />
379 <parameter name="PLL_C2P_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
380 <parameter name="PLL_C2P_WRITE_CLK_MULT_PARAM" value="0" />
381 <parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_PARAM" value="0" />
382 <parameter name="PLL_C2P_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
383 <parameter name="PLL_CLK_PARAM_VALID" value="false" />
384 <parameter name="PLL_CONFIG_CLK_DIV_PARAM" value="0" />
385 <parameter name="PLL_CONFIG_CLK_FREQ_PARAM" value="0.0" />
386 <parameter name="PLL_CONFIG_CLK_FREQ_SIM_STR_PARAM" value="" />
387 <parameter name="PLL_CONFIG_CLK_MULT_PARAM" value="0" />
388 <parameter name="PLL_CONFIG_CLK_PHASE_PS_PARAM" value="0" />
389 <parameter name="PLL_CONFIG_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
390 <parameter name="PLL_DR_CLK_DIV_PARAM" value="0" />
391 <parameter name="PLL_DR_CLK_FREQ_PARAM" value="0.0" />
392 <parameter name="PLL_DR_CLK_FREQ_SIM_STR_PARAM" value="" />
393 <parameter name="PLL_DR_CLK_MULT_PARAM" value="0" />
394 <parameter name="PLL_DR_CLK_PHASE_PS_PARAM" value="0" />
395 <parameter name="PLL_DR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
396 <parameter name="PLL_HR_CLK_DIV_PARAM" value="0" />
397 <parameter name="PLL_HR_CLK_FREQ_PARAM" value="0.0" />
398 <parameter name="PLL_HR_CLK_FREQ_SIM_STR_PARAM" value="" />
399 <parameter name="PLL_HR_CLK_MULT_PARAM" value="0" />
400 <parameter name="PLL_HR_CLK_PHASE_PS_PARAM" value="0" />
401 <parameter name="PLL_HR_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
402 <parameter name="PLL_LOCATION" value="Top_Bottom" />
403 <parameter name="PLL_MEM_CLK_DIV_PARAM" value="0" />
404 <parameter name="PLL_MEM_CLK_FREQ_PARAM" value="0.0" />
405 <parameter name="PLL_MEM_CLK_FREQ_SIM_STR_PARAM" value="" />
406 <parameter name="PLL_MEM_CLK_MULT_PARAM" value="0" />
407 <parameter name="PLL_MEM_CLK_PHASE_PS_PARAM" value="0" />
408 <parameter name="PLL_MEM_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
409 <parameter name="PLL_NIOS_CLK_DIV_PARAM" value="0" />
410 <parameter name="PLL_NIOS_CLK_FREQ_PARAM" value="0.0" />
411 <parameter name="PLL_NIOS_CLK_FREQ_SIM_STR_PARAM" value="" />
412 <parameter name="PLL_NIOS_CLK_MULT_PARAM" value="0" />
413 <parameter name="PLL_NIOS_CLK_PHASE_PS_PARAM" value="0" />
414 <parameter name="PLL_NIOS_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
415 <parameter name="PLL_P2C_READ_CLK_DIV_PARAM" value="0" />
416 <parameter name="PLL_P2C_READ_CLK_FREQ_PARAM" value="0.0" />
417 <parameter name="PLL_P2C_READ_CLK_FREQ_SIM_STR_PARAM" value="" />
418 <parameter name="PLL_P2C_READ_CLK_MULT_PARAM" value="0" />
419 <parameter name="PLL_P2C_READ_CLK_PHASE_PS_PARAM" value="0" />
420 <parameter name="PLL_P2C_READ_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
421 <parameter name="PLL_SHARING_MODE" value="None" />
422 <parameter name="PLL_WRITE_CLK_DIV_PARAM" value="0" />
423 <parameter name="PLL_WRITE_CLK_FREQ_PARAM" value="0.0" />
424 <parameter name="PLL_WRITE_CLK_FREQ_SIM_STR_PARAM" value="" />
425 <parameter name="PLL_WRITE_CLK_MULT_PARAM" value="0" />
426 <parameter name="PLL_WRITE_CLK_PHASE_PS_PARAM" value="0" />
427 <parameter name="PLL_WRITE_CLK_PHASE_PS_SIM_STR_PARAM" value="" />
428 <parameter name="POWER_OF_TWO_BUS" value="false" />
429 <parameter name="PRIORITY_PORT" value="1,1,1,1,1" />
430 <parameter name="QSPI_Mode" value="N/A" />
431 <parameter name="QSPI_PinMuxing" value="Unused" />
432 <parameter name="RATE" value="Full" />
433 <parameter name="RDIMM_CONFIG" value="0000000000000000" />
434 <parameter name="READ_DQ_DQS_CLOCK_SOURCE" value="INVERTED_DQS_BUS" />
435 <parameter name="READ_FIFO_SIZE" value="8" />
436 <parameter name="REFRESH_BURST_VALIDATION" value="false" />
437 <parameter name="REFRESH_INTERVAL" value="15000" />
438 <parameter name="REF_CLK_FREQ" value="25.0" />
439 <parameter name="REF_CLK_FREQ_MAX_PARAM" value="0.0" />
440 <parameter name="REF_CLK_FREQ_MIN_PARAM" value="0.0" />

```

```

441 <parameter name="REF_CLK_FREQ_PARAM_VALID" value="false" />
442 <parameter name="S2FCLK_COLD_RST_Enable" value="false" />
443 <parameter name="S2FCLK_PENDING_RST_Enable" value="false" />
444 <parameter name="S2FCLK_USER0CLK_Enable" value="false" />
445 <parameter name="S2FCLK_USER1CLK_Enable" value="true" />
446 <parameter name="S2FCLK_USER1CLK_FREQ" value="100.0" />
447 <parameter name="S2FCLK_USER2CLK" value="5" />
448 <parameter name="S2FCLK_USER2CLK_Enable" value="false" />
449 <parameter name="S2FCLK_USER2CLK_FREQ" value="100.0" />
450 <parameter name="S2FINTERRUPT_CAN_Enable" value="false" />
451 <parameter name="S2FINTERRUPT_CLOCKPERIPHERAL_Enable" value="false" />
452 <parameter name="S2FINTERRUPT_CTI_Enable" value="false" />
453 <parameter name="S2FINTERRUPT_DMA_Enable" value="false" />
454 <parameter name="S2FINTERRUPT_EMAC_Enable" value="false" />
455 <parameter name="S2FINTERRUPT_FPGAMANAGER_Enable" value="false" />
456 <parameter name="S2FINTERRUPT_GPIO_Enable" value="false" />
457 <parameter name="S2FINTERRUPT_I2CEMAC_Enable" value="false" />
458 <parameter name="S2FINTERRUPT_I2CPERIPHERAL_Enable" value="false" />
459 <parameter name="S2FINTERRUPT_L4TIMER_Enable" value="false" />
460 <parameter name="S2FINTERRUPT_NAND_Enable" value="false" />
461 <parameter name="S2FINTERRUPT_OSCTIMER_Enable" value="false" />
462 <parameter name="S2FINTERRUPT_QSPI_Enable" value="false" />
463 <parameter name="S2FINTERRUPT_SDMMC_Enable" value="false" />
464 <parameter name="S2FINTERRUPT_SPIMASTER_Enable" value="false" />
465 <parameter name="S2FINTERRUPT_SPISLAVE_Enable" value="false" />
466 <parameter name="S2FINTERRUPT_UART_Enable" value="false" />
467 <parameter name="S2FINTERRUPT_USB_Enable" value="false" />
468 <parameter name="S2FINTERRUPT_WATCHDOG_Enable" value="false" />
469 <parameter name="S2F_Width" value="2" />
470 <parameter name="SDIO_Mode" value="4-bit Data" />
471 <parameter name="SDIO_PinMuxing" value="HPS I/O Set 0" />
472 <parameter name="SEQUENCER_TYPE" value="NIOS" />
473 <parameter name="SEQ_MODE" value="0" />
474 <parameter name="SKIP_MEM_INIT" value="true" />
475 <parameter name="SOPC_COMPAT_RESET" value="false" />
476 <parameter name="SPEED_GRADE" value="7" />
477 <parameter name="SPIMO_Mode" value="N/A" />
478 <parameter name="SPIMO_PinMuxing" value="Unused" />
479 <parameter name="SPIM1_Mode" value="Single Slave Select" />
480 <parameter name="SPIM1_PinMuxing" value="HPS I/O Set 0" />
481 <parameter name="SPISO_Mode" value="N/A" />
482 <parameter name="SPISO_PinMuxing" value="Unused" />
483 <parameter name="SPIS1_Mode" value="N/A" />
484 <parameter name="SPIS1_PinMuxing" value="Unused" />
485 <parameter name="STARVE_LIMIT" value="10" />
486 <parameter name="STM_Enable" value="false" />
487 <parameter name="SYS_INFO_DEVICE_FAMILY" value="Cyclone V" />
488 <parameter name="TEST_Enable" value="false" />
489 <parameter name="TIMING_BOARD_AC_EYE_REDUCTION_H" value="0.0" />
490 <parameter name="TIMING_BOARD_AC_EYE_REDUCTION_SU" value="0.0" />
491 <parameter name="TIMING_BOARD_AC_SKEW" value="0.03" />
492 <parameter name="TIMING_BOARD_AC_SLEW_RATE" value="1.0" />
493 <parameter name="TIMING_BOARD_AC_TO_CK_SKEW" value="0.0" />
494 <parameter name="TIMING_BOARD_CK_CKN_SLEW_RATE" value="2.0" />
495 <parameter name="TIMING_BOARD_DELTA_DQS_ARRIVAL_TIME" value="0.0" />
496 <parameter name="TIMING_BOARD_DELTA_READ_DQS_ARRIVAL_TIME" value="0.0" />
497 <parameter name="TIMING_BOARD_DERATE_METHOD" value="AUTO" />
498 <parameter name="TIMING_BOARD_DQS_DQSN_SLEW_RATE" value="2.0" />
499 <parameter name="TIMING_BOARD_DQ_EYE_REDUCTION" value="0.0" />
500 <parameter name="TIMING_BOARD_DQ_SLEW_RATE" value="1.0" />
501 <parameter name="TIMING_BOARD_DQ_TO_DQS_SKEW" value="0.0" />
502 <parameter name="TIMING_BOARD_ISI_METHOD" value="AUTO" />
503 <parameter name="TIMING_BOARD_MAX_CK_DELAY" value="0.03" />
504 <parameter name="TIMING_BOARD_MAX_DQS_DELAY" value="0.02" />
505 <parameter name="TIMING_BOARD_READ_DQ_EYE_REDUCTION" value="0.0" />
506 <parameter name="TIMING_BOARD_SKEW_BETWEEN_DIMMS" value="0.05" />
507 <parameter name="TIMING_BOARD_SKEW_BETWEEN_DQS" value="0.08" />
508 <parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MAX" value="0.16" />
509 <parameter name="TIMING_BOARD_SKEW_CKDQS_DIMM_MIN" value="0.09" />
510 <parameter name="TIMING_BOARD_SKEW_WITHIN_DQS" value="0.01" />
511 <parameter name="TIMING_BOARD_TDH" value="0.0" />
512 <parameter name="TIMING_BOARD_TDS" value="0.0" />
513 <parameter name="TIMING_BOARD_TIH" value="0.0" />
514 <parameter name="TIMING_BOARD_TIS" value="0.0" />

```

```

515 <parameter name="TIMING_TDH" value="65" />
516 <parameter name="TIMING_TDQCK" value="255" />
517 <parameter name="TIMING_TDQCKDL" value="1200" />
518 <parameter name="TIMING_TDQCKDM" value="900" />
519 <parameter name="TIMING_TDQCKDS" value="450" />
520 <parameter name="TIMING_TDQSH" value="0.35" />
521 <parameter name="TIMING_TDQSQ" value="125" />
522 <parameter name="TIMING_TDQSS" value="0.25" />
523 <parameter name="TIMING_TDS" value="30" />
524 <parameter name="TIMING_TDSH" value="0.2" />
525 <parameter name="TIMING_TDSS" value="0.2" />
526 <parameter name="TIMING_TIH" value="140" />
527 <parameter name="TIMING_TIS" value="180" />
528 <parameter name="TIMING_TQH" value="0.38" />
529 <parameter name="TIMING_TQHS" value="300" />
530 <parameter name="TIMING_TQSH" value="0.4" />
531 <parameter name="TPIUFPGA_Enable" value="false" />
532 <parameter name="TPIUFPGA_alt" value="false" />
533 <parameter name="TRACE_Mode" value="N/A" />
534 <parameter name="TRACE_PinMuxing" value="Unused" />
535 <parameter name="TRACKING_ERROR_TEST" value="false" />
536 <parameter name="TRACKING_WATCH_TEST" value="false" />
537 <parameter name="TREFI" value="35100" />
538 <parameter name="TRFC" value="350" />
539 <parameter name="UART0_Mode" value="No Flow Control" />
540 <parameter name="UART0_PinMuxing" value="HPS I/O Set 0" />
541 <parameter name="UART1_Mode" value="N/A" />
542 <parameter name="UART1_PinMuxing" value="Unused" />
543 <parameter name="USB0_Mode" value="N/A" />
544 <parameter name="USB0_PinMuxing" value="Unused" />
545 <parameter name="USB1_Mode" value="SDR" />
546 <parameter name="USB1_PinMuxing" value="HPS I/O Set 0" />
547 <parameter name="USER_DEBUG_LEVEL" value="1" />
548 <parameter name="USE_AXI_ADAPTOR" value="false" />
549 <parameter name="USE_FAKE_PHY" value="false" />
550 <parameter name="USE_MEM_CLK_FREQ" value="false" />
551 <parameter name="USE_MM_ADAPTOR" value="true" />
552 <parameter name="USE_SEQUENCER_BFM" value="false" />
553 <parameter name="WEIGHT_PORT" value="0,0,0,0,0" />
554 <parameter name="WRBUFFER_ADDR_WIDTH" value="6" />
555 <parameter name="can0_clk_div" value="1" />
556 <parameter name="can1_clk_div" value="1" />
557 <parameter name="configure_advanced_parameters" value="false" />
558 <parameter name="customize_device_pll_info" value="false" />
559 <parameter name="dbctrl_stayosc1" value="true" />
560 <parameter name="dbg_at_clk_div" value="0" />
561 <parameter name="dbg_clk_div" value="1" />
562 <parameter name="dbg_trace_clk_div" value="0" />
563 <parameter name="desired_can0_clk_mhz" value="100.0" />
564 <parameter name="desired_can1_clk_mhz" value="100.0" />
565 <parameter name="desired_cfg_clk_mhz" value="97.368421" />
566 <parameter name="desired_emac0_clk_mhz" value="250.0" />
567 <parameter name="desired_emac1_clk_mhz" value="250.0" />
568 <parameter name="desired_gpio_db_clk_hz" value="32000" />
569 <parameter name="desired_l4_mp_clk_mhz" value="100.0" />
570 <parameter name="desired_l4_sp_clk_mhz" value="100.0" />
571 <parameter name="desired_mpu_clk_mhz" value="800.0" />
572 <parameter name="desired_nand_clk_mhz" value="12.5" />
573 <parameter name="desired_qspi_clk_mhz" value="400.0" />
574 <parameter name="desired_sdmnc_clk_mhz" value="200.0" />
575 <parameter name="desired_spi_m_clk_mhz" value="200.0" />
576 <parameter name="desired_usb_mp_clk_mhz" value="200.0" />
577 <parameter name="device_name" value="5CSEMA5F31C6" />
578 <parameter name="device_pll_info_manual">{320000000 1600000000} {320000000 1000000000} {800000000 400000000
400000000}</parameter>
579 <parameter name="eosc1_clk_mhz" value="25.0" />
580 <parameter name="eosc2_clk_mhz" value="25.0" />
581 <parameter name="gpio_db_clk_div" value="6249" />
582 <parameter name="l3_mp_clk_div" value="1" />
583 <parameter name="l3_sp_clk_div" value="1" />
584 <parameter name="l4_mp_clk_div" value="1" />
585 <parameter name="l4_mp_clk_source" value="1" />
586 <parameter name="l4_sp_clk_div" value="1" />
587 <parameter name="l4_sp_clk_source" value="1" />

```

```

588 <parameter name="main_pll_c3" value="3" />
589 <parameter name="main_pll_c4" value="3" />
590 <parameter name="main_pll_c5" value="15" />
591 <parameter name="main_pll_m" value="63" />
592 <parameter name="main_pll_n" value="0" />
593 <parameter name="nand_clk_source" value="2" />
594 <parameter name="periph_pll_c0" value="3" />
595 <parameter name="periph_pll_c1" value="3" />
596 <parameter name="periph_pll_c2" value="1" />
597 <parameter name="periph_pll_c3" value="19" />
598 <parameter name="periph_pll_c4" value="4" />
599 <parameter name="periph_pll_c5" value="9" />
600 <parameter name="periph_pll_m" value="79" />
601 <parameter name="periph_pll_n" value="1" />
602 <parameter name="periph_pll_source" value="0" />
603 <parameter name="qspi_clk_source" value="1" />
604 <parameter name="quartus_ini_hps_emif_pll" value="false" />
605 <parameter
606     name="quartus_ini_hps_ip_enable_all_peripheral_fpga_interfaces"
607     value="false" />
608 <parameter name="quartus_ini_hps_ip_enable_bsel_csel" value="false" />
609 <parameter
610     name="quartus_ini_hps_ip_enable_emac0_peripheral_fpga_interface"
611     value="false" />
612 <parameter
613     name="quartus_ini_hps_ip_enable_low_speed_serial_fpga_interfaces"
614     value="false" />
615 <parameter name="quartus_ini_hps_ip_enable_test_interface" value="false" />
616 <parameter name="quartus_ini_hps_ip_f2sdr_am_bonding_out" value="false" />
617 <parameter name="quartus_ini_hps_ip_fast_f2sdr_am_sim_model" value="false" />
618 <parameter name="quartus_ini_hps_ip_suppress_sdr_am_synth" value="false" />
619 <parameter name="sdmmc_clk_source" value="2" />
620 <parameter name="show_advanced_parameters" value="false" />
621 <parameter name="show_debug_info_as_warning_msg" value="false" />
622 <parameter name="show_warning_as_error_msg" value="false" />
623 <parameter name="spi_m_clk_div" value="0" />
624 <parameter name="usb_mp_clk_div" value="0" />
625 <parameter name="use_default_mpu_clk" value="true" />
626 </module>
627 <module name="pll_0" kind="altera_pll" version="21.1" enabled="1">
628 <parameter name="debug_print_output" value="false" />
629 <parameter name="debug_use_rbc_taf_method" value="false" />
630 <parameter name="device" value="5CSEMA5F31C6" />
631 <parameter name="device_family" value="Cyclone V" />
632 <parameter name="gui_active_clk" value="false" />
633 <parameter name="gui_actual_output_clock_frequency0" value="0 MHz" />
634 <parameter name="gui_actual_output_clock_frequency1" value="0 MHz" />
635 <parameter name="gui_actual_output_clock_frequency10" value="0 MHz" />
636 <parameter name="gui_actual_output_clock_frequency11" value="0 MHz" />
637 <parameter name="gui_actual_output_clock_frequency12" value="0 MHz" />
638 <parameter name="gui_actual_output_clock_frequency13" value="0 MHz" />
639 <parameter name="gui_actual_output_clock_frequency14" value="0 MHz" />
640 <parameter name="gui_actual_output_clock_frequency15" value="0 MHz" />
641 <parameter name="gui_actual_output_clock_frequency16" value="0 MHz" />
642 <parameter name="gui_actual_output_clock_frequency17" value="0 MHz" />
643 <parameter name="gui_actual_output_clock_frequency2" value="0 MHz" />
644 <parameter name="gui_actual_output_clock_frequency3" value="0 MHz" />
645 <parameter name="gui_actual_output_clock_frequency4" value="0 MHz" />
646 <parameter name="gui_actual_output_clock_frequency5" value="0 MHz" />
647 <parameter name="gui_actual_output_clock_frequency6" value="0 MHz" />
648 <parameter name="gui_actual_output_clock_frequency7" value="0 MHz" />
649 <parameter name="gui_actual_output_clock_frequency8" value="0 MHz" />
650 <parameter name="gui_actual_output_clock_frequency9" value="0 MHz" />
651 <parameter name="gui_actual_phase_shift0" value="0" />
652 <parameter name="gui_actual_phase_shift1" value="0" />
653 <parameter name="gui_actual_phase_shift10" value="0" />
654 <parameter name="gui_actual_phase_shift11" value="0" />
655 <parameter name="gui_actual_phase_shift12" value="0" />
656 <parameter name="gui_actual_phase_shift13" value="0" />
657 <parameter name="gui_actual_phase_shift14" value="0" />
658 <parameter name="gui_actual_phase_shift15" value="0" />
659 <parameter name="gui_actual_phase_shift16" value="0" />
660 <parameter name="gui_actual_phase_shift17" value="0" />
661 <parameter name="gui_actual_phase_shift2" value="0" />

```

```

662 <parameter name="gui_actual_phase_shift3" value="0" />
663 <parameter name="gui_actual_phase_shift4" value="0" />
664 <parameter name="gui_actual_phase_shift5" value="0" />
665 <parameter name="gui_actual_phase_shift6" value="0" />
666 <parameter name="gui_actual_phase_shift7" value="0" />
667 <parameter name="gui_actual_phase_shift8" value="0" />
668 <parameter name="gui_actual_phase_shift9" value="0" />
669 <parameter name="gui_cascade_counter0" value="false" />
670 <parameter name="gui_cascade_counter1" value="false" />
671 <parameter name="gui_cascade_counter10" value="false" />
672 <parameter name="gui_cascade_counter11" value="false" />
673 <parameter name="gui_cascade_counter12" value="false" />
674 <parameter name="gui_cascade_counter13" value="false" />
675 <parameter name="gui_cascade_counter14" value="false" />
676 <parameter name="gui_cascade_counter15" value="false" />
677 <parameter name="gui_cascade_counter16" value="false" />
678 <parameter name="gui_cascade_counter17" value="false" />
679 <parameter name="gui_cascade_counter2" value="false" />
680 <parameter name="gui_cascade_counter3" value="false" />
681 <parameter name="gui_cascade_counter4" value="false" />
682 <parameter name="gui_cascade_counter5" value="false" />
683 <parameter name="gui_cascade_counter6" value="false" />
684 <parameter name="gui_cascade_counter7" value="false" />
685 <parameter name="gui_cascade_counter8" value="false" />
686 <parameter name="gui_cascade_counter9" value="false" />
687 <parameter name="gui_cascade_outclk_index" value="0" />
688 <parameter name="gui_channel_spacing" value="0.0" />
689 <parameter name="gui_clk_bad" value="false" />
690 <parameter name="gui_device_speed_grade" value="1" />
691 <parameter name="gui_divide_factor_c0" value="1" />
692 <parameter name="gui_divide_factor_c1" value="1" />
693 <parameter name="gui_divide_factor_c10" value="1" />
694 <parameter name="gui_divide_factor_c11" value="1" />
695 <parameter name="gui_divide_factor_c12" value="1" />
696 <parameter name="gui_divide_factor_c13" value="1" />
697 <parameter name="gui_divide_factor_c14" value="1" />
698 <parameter name="gui_divide_factor_c15" value="1" />
699 <parameter name="gui_divide_factor_c16" value="1" />
700 <parameter name="gui_divide_factor_c17" value="1" />
701 <parameter name="gui_divide_factor_c2" value="1" />
702 <parameter name="gui_divide_factor_c3" value="1" />
703 <parameter name="gui_divide_factor_c4" value="1" />
704 <parameter name="gui_divide_factor_c5" value="1" />
705 <parameter name="gui_divide_factor_c6" value="1" />
706 <parameter name="gui_divide_factor_c7" value="1" />
707 <parameter name="gui_divide_factor_c8" value="1" />
708 <parameter name="gui_divide_factor_c9" value="1" />
709 <parameter name="gui_divide_factor_n" value="1" />
710 <parameter name="gui_dps_cntr" value="C0" />
711 <parameter name="gui_dps_dir" value="Positive" />
712 <parameter name="gui_dps_num" value="1" />
713 <parameter name="gui_dsm_out_sel" value="1st_order" />
714 <parameter name="gui_duty_cycle0" value="50" />
715 <parameter name="gui_duty_cycle1" value="50" />
716 <parameter name="gui_duty_cycle10" value="50" />
717 <parameter name="gui_duty_cycle11" value="50" />
718 <parameter name="gui_duty_cycle12" value="50" />
719 <parameter name="gui_duty_cycle13" value="50" />
720 <parameter name="gui_duty_cycle14" value="50" />
721 <parameter name="gui_duty_cycle15" value="50" />
722 <parameter name="gui_duty_cycle16" value="50" />
723 <parameter name="gui_duty_cycle17" value="50" />
724 <parameter name="gui_duty_cycle2" value="50" />
725 <parameter name="gui_duty_cycle3" value="50" />
726 <parameter name="gui_duty_cycle4" value="50" />
727 <parameter name="gui_duty_cycle5" value="50" />
728 <parameter name="gui_duty_cycle6" value="50" />
729 <parameter name="gui_duty_cycle7" value="50" />
730 <parameter name="gui_duty_cycle8" value="50" />
731 <parameter name="gui_duty_cycle9" value="50" />
732 <parameter name="gui_en_adv_params" value="false" />
733 <parameter name="gui_en_dps_ports" value="false" />
734 <parameter name="gui_en_phout_ports" value="false" />
735 <parameter name="gui_en_reconf" value="false" />

```

```

736 <parameter name="gui_enable_cascade_in" value="false" />
737 <parameter name="gui_enable_cascade_out" value="false" />
738 <parameter name="gui_enable_mif_dps" value="false" />
739 <parameter name="gui_feedback_clock" value="Global Clock" />
740 <parameter name="gui_frac_multiply_factor" value="1" />
741 <parameter name="gui_fractional_cout" value="32" />
742 <parameter name="gui_mif_generate" value="false" />
743 <parameter name="gui_multiply_factor" value="1" />
744 <parameter name="gui_number_of_clocks" value="1" />
745 <parameter name="gui_operation_mode" value="direct" />
746 <parameter name="gui_output_clock_frequency0" value="125.0" />
747 <parameter name="gui_output_clock_frequency1" value="100.0" />
748 <parameter name="gui_output_clock_frequency10" value="100.0" />
749 <parameter name="gui_output_clock_frequency11" value="100.0" />
750 <parameter name="gui_output_clock_frequency12" value="100.0" />
751 <parameter name="gui_output_clock_frequency13" value="100.0" />
752 <parameter name="gui_output_clock_frequency14" value="100.0" />
753 <parameter name="gui_output_clock_frequency15" value="100.0" />
754 <parameter name="gui_output_clock_frequency16" value="100.0" />
755 <parameter name="gui_output_clock_frequency17" value="100.0" />
756 <parameter name="gui_output_clock_frequency2" value="100.0" />
757 <parameter name="gui_output_clock_frequency3" value="100.0" />
758 <parameter name="gui_output_clock_frequency4" value="100.0" />
759 <parameter name="gui_output_clock_frequency5" value="100.0" />
760 <parameter name="gui_output_clock_frequency6" value="100.0" />
761 <parameter name="gui_output_clock_frequency7" value="100.0" />
762 <parameter name="gui_output_clock_frequency8" value="100.0" />
763 <parameter name="gui_output_clock_frequency9" value="100.0" />
764 <parameter name="gui_phase_shift0" value="0" />
765 <parameter name="gui_phase_shift1" value="0" />
766 <parameter name="gui_phase_shift10" value="0" />
767 <parameter name="gui_phase_shift11" value="0" />
768 <parameter name="gui_phase_shift12" value="0" />
769 <parameter name="gui_phase_shift13" value="0" />
770 <parameter name="gui_phase_shift14" value="0" />
771 <parameter name="gui_phase_shift15" value="0" />
772 <parameter name="gui_phase_shift16" value="0" />
773 <parameter name="gui_phase_shift17" value="0" />
774 <parameter name="gui_phase_shift2" value="0" />
775 <parameter name="gui_phase_shift3" value="0" />
776 <parameter name="gui_phase_shift4" value="0" />
777 <parameter name="gui_phase_shift5" value="0" />
778 <parameter name="gui_phase_shift6" value="0" />
779 <parameter name="gui_phase_shift7" value="0" />
780 <parameter name="gui_phase_shift8" value="0" />
781 <parameter name="gui_phase_shift9" value="0" />
782 <parameter name="gui_phase_shift_deg0" value="0.0" />
783 <parameter name="gui_phase_shift_deg1" value="0.0" />
784 <parameter name="gui_phase_shift_deg10" value="0.0" />
785 <parameter name="gui_phase_shift_deg11" value="0.0" />
786 <parameter name="gui_phase_shift_deg12" value="0.0" />
787 <parameter name="gui_phase_shift_deg13" value="0.0" />
788 <parameter name="gui_phase_shift_deg14" value="0.0" />
789 <parameter name="gui_phase_shift_deg15" value="0.0" />
790 <parameter name="gui_phase_shift_deg16" value="0.0" />
791 <parameter name="gui_phase_shift_deg17" value="0.0" />
792 <parameter name="gui_phase_shift_deg2" value="0.0" />
793 <parameter name="gui_phase_shift_deg3" value="0.0" />
794 <parameter name="gui_phase_shift_deg4" value="0.0" />
795 <parameter name="gui_phase_shift_deg5" value="0.0" />
796 <parameter name="gui_phase_shift_deg6" value="0.0" />
797 <parameter name="gui_phase_shift_deg7" value="0.0" />
798 <parameter name="gui_phase_shift_deg8" value="0.0" />
799 <parameter name="gui_phase_shift_deg9" value="0.0" />
800 <parameter name="gui_phout_division" value="1" />
801 <parameter name="gui_pll_auto_reset" value="Off" />
802 <parameter name="gui_pll_bandwidth_preset" value="Auto" />
803 <parameter name="gui_pll_cascading_mode">Create an adjpll in signal to connect with an upstream PLL</parameter>
804 <parameter name="gui_pll_mode" value="Integer-N PLL" />
805 <parameter name="gui_ps_units0" value="ps" />
806 <parameter name="gui_ps_units1" value="ps" />
807 <parameter name="gui_ps_units10" value="ps" />
808 <parameter name="gui_ps_units11" value="ps" />
809 <parameter name="gui_ps_units12" value="ps" />

```

```

810 <parameter name="gui_ps_units13" value="ps" />
811 <parameter name="gui_ps_units14" value="ps" />
812 <parameter name="gui_ps_units15" value="ps" />
813 <parameter name="gui_ps_units16" value="ps" />
814 <parameter name="gui_ps_units17" value="ps" />
815 <parameter name="gui_ps_units2" value="ps" />
816 <parameter name="gui_ps_units3" value="ps" />
817 <parameter name="gui_ps_units4" value="ps" />
818 <parameter name="gui_ps_units5" value="ps" />
819 <parameter name="gui_ps_units6" value="ps" />
820 <parameter name="gui_ps_units7" value="ps" />
821 <parameter name="gui_ps_units8" value="ps" />
822 <parameter name="gui_ps_units9" value="ps" />
823 <parameter name="gui_refclk1_frequency" value="100.0" />
824 <parameter name="gui_refclk_switch" value="false" />
825 <parameter name="gui_reference_clock_frequency" value="50.0" />
826 <parameter name="gui_switchover_delay" value="0" />
827 <parameter name="gui_switchover_mode">Automatic Switchover</parameter>
828 <parameter name="gui_use_locked" value="true" />
829 </module>
830 <module name="raytracer_0" kind="raytracer" version="1.0" enabled="1">
831 <parameter name="FRAC_BITS" value="13" />
832 <parameter name="HEIGHT" value="360" />
833 <parameter name="N" value="480" />
834 <parameter name="WIDTH" value="480" />
835 <parameter name="WORD" value="27" />
836 </module>
837 <connection
838 kind="avalon"
839 version="21.1"
840 start="hps_0.h2f_axi_master"
841 end="raytracer_0.avalon_slave_0">
842 <parameter name="arbitrationPriority" value="1" />
843 <parameter name="baseAddress" value="0x0000" />
844 <parameter name="defaultConnection" value="false" />
845 </connection>
846 <connection
847 kind="avalon"
848 version="21.1"
849 start="raytracer_0.avalon_master_0"
850 end="hps_0.f2h_axi_slave">
851 <parameter name="arbitrationPriority" value="1" />
852 <parameter name="baseAddress" value="0x0000" />
853 <parameter name="defaultConnection" value="false" />
854 </connection>
855 <connection
856 kind="clock"
857 version="21.1"
858 start="clk_0.clk"
859 end="hps_0.f2h_axi_clock" />
860 <connection
861 kind="clock"
862 version="21.1"
863 start="clk_0.clk"
864 end="hps_0.h2f_axi_clock" />
865 <connection
866 kind="clock"
867 version="21.1"
868 start="clk_0.clk"
869 end="hps_0.h2f_lw_axi_clock" />
870 <connection kind="clock" version="21.1" start="clk_0.clk" end="pll_0.refclk" />
871 <connection
872 kind="clock"
873 version="21.1"
874 start="pll_0.outclk0"
875 end="raytracer_0.clock" />
876 <connection
877 kind="reset"
878 version="21.1"
879 start="clk_0.clk_reset"
880 end="raytracer_0.reset" />
881 <connection kind="reset" version="21.1" start="clk_0.clk_reset" end="pll_0.reset" />
882 <interconnectRequirement for="$system" name="qsys_mm.clockCrossingAdapter" value="HANDSHAKE" />
883 <interconnectRequirement for="$system" name="qsys_mm.enableEccProtection" value="FALSE" />

```



```

884 <interconnectRequirement for="$system" name="qsys_mm.insertDefaultSlave" value="FALSE" />
885 <interconnectRequirement for="$system" name="qsys_mm.maxAdditionalLatency" value="2" />
886 </system>

```

soc_system.tcl

```

1  # Generate Quartus project files for the DE1-SoC board
2  #
3  # Stephen A. Edwards, Columbia University
4
5  # Invoke as
6  #
7  # quartus_sh -t soc_system.tcl
8
9  set project "soc_system"
10
11 # Top-level SystemVerilog file should be <project>_top.sv, with Verilog module
12 # <project>_top in it
13
14 set systemVerilogSource "${project}_top.sv"
15 set qip "${project}/synthesis/${project}.qip"
16
17 project_new $project -overwrite
18
19 foreach {name value} {
20     FAMILY "Cyclone V"
21     DEVICE 5CSEMA5F31C6
22
23     PROJECT_OUTPUT_DIRECTORY output_files
24
25     CYCLONEII_RESERVE_NCEO_AFTER_CONFIGURATION "USE AS REGULAR IO"
26
27     NUM_PARALLEL_PROCESSORS 4
28 } { set_global_assignment -name $name $value }
29
30 set_global_assignment -name TOP_LEVEL_ENTITY "${project}_top"
31
32 foreach filename $systemVerilogSource {
33     set_global_assignment -name SYSTEMVERILOG_FILE $filename
34 }
35
36 foreach filename $qip {
37     set_global_assignment -name QIP_FILE $filename
38 }
39
40
41 # FPGA pin assignments
42
43 foreach {pin port} {
44     PIN_AJ4 ADC_CS_N
45     PIN_AK4 ADC_DIN
46     PIN_AK3 ADC_DOUT
47     PIN_AK2 ADC_SCLK
48
49     PIN_K7 AUD_ADCCDAT
50     PIN_K8 AUD_ADCLRCK
51     PIN_H7 AUD_BCLK
52     PIN_J7 AUD_DACDAT
53     PIN_H8 AUD_DACLCK
54     PIN_G7 AUD_XCK
55
56     PIN_AA16 CLOCK2_50
57     PIN_Y26 CLOCK3_50
58     PIN_K14 CLOCK4_50
59     PIN_AF14 CLOCK_50
60
61     PIN_AK14 DRAM_ADDR[0]
62     PIN_AH14 DRAM_ADDR[1]
63     PIN_AG15 DRAM_ADDR[2]
64     PIN_AE14 DRAM_ADDR[3]
65     PIN_AB15 DRAM_ADDR[4]
66     PIN_AC14 DRAM_ADDR[5]

```

```

67 PIN_AD14 DRAM_ADDR[6]
68 PIN_AF15 DRAM_ADDR[7]
69 PIN_AH15 DRAM_ADDR[8]
70 PIN_AG13 DRAM_ADDR[9]
71 PIN_AG12 DRAM_ADDR[10]
72 PIN_AH13 DRAM_ADDR[11]
73 PIN_AJ14 DRAM_ADDR[12]
74 PIN_AF13 DRAM_BA[0]
75 PIN_AJ12 DRAM_BA[1]
76 PIN_AF11 DRAM_CAS_N
77 PIN_AK13 DRAM_CKE
78 PIN_AH12 DRAM_CLK
79 PIN_AG11 DRAM_CS_N
80 PIN_AK6 DRAM_DQ[0]
81 PIN_AJ7 DRAM_DQ[1]
82 PIN_AK7 DRAM_DQ[2]
83 PIN_AK8 DRAM_DQ[3]
84 PIN_AK9 DRAM_DQ[4]
85 PIN_AG10 DRAM_DQ[5]
86 PIN_AK11 DRAM_DQ[6]
87 PIN_AJ11 DRAM_DQ[7]
88 PIN_AH10 DRAM_DQ[8]
89 PIN_AJ10 DRAM_DQ[9]
90 PIN_AJ9 DRAM_DQ[10]
91 PIN_AH9 DRAM_DQ[11]
92 PIN_AH8 DRAM_DQ[12]
93 PIN_AH7 DRAM_DQ[13]
94 PIN_AJ6 DRAM_DQ[14]
95 PIN_AJ5 DRAM_DQ[15]
96 PIN_AB13 DRAM_LDQM
97 PIN_AE13 DRAM_RAS_N
98 PIN_AK12 DRAM_UDQM
99 PIN_AA13 DRAM_WE_N
100
101 PIN_AA12 FAN_CTRL
102
103 PIN_J12 FPGA_I2C_SCLK
104 PIN_K12 FPGA_I2C_SDAT
105
106 PIN_AC18 GPIO_0[0]
107 PIN_Y17 GPIO_0[1]
108 PIN_AD17 GPIO_0[2]
109 PIN_Y18 GPIO_0[3]
110 PIN_AK16 GPIO_0[4]
111 PIN_AK18 GPIO_0[5]
112 PIN_AK19 GPIO_0[6]
113 PIN_AJ19 GPIO_0[7]
114 PIN_AJ17 GPIO_0[8]
115 PIN_AJ16 GPIO_0[9]
116 PIN_AH18 GPIO_0[10]
117 PIN_AH17 GPIO_0[11]
118 PIN_AG16 GPIO_0[12]
119 PIN_AE16 GPIO_0[13]
120 PIN_AF16 GPIO_0[14]
121 PIN_AG17 GPIO_0[15]
122 PIN_AA18 GPIO_0[16]
123 PIN_AA19 GPIO_0[17]
124 PIN_AE17 GPIO_0[18]
125 PIN_AC20 GPIO_0[19]
126 PIN_AH19 GPIO_0[20]
127 PIN_AJ20 GPIO_0[21]
128 PIN_AH20 GPIO_0[22]
129 PIN_AK21 GPIO_0[23]
130 PIN_AD19 GPIO_0[24]
131 PIN_AD20 GPIO_0[25]
132 PIN_AE18 GPIO_0[26]
133 PIN_AE19 GPIO_0[27]
134 PIN_AF20 GPIO_0[28]
135 PIN_AF21 GPIO_0[29]
136 PIN_AF19 GPIO_0[30]
137 PIN_AG21 GPIO_0[31]
138 PIN_AF18 GPIO_0[32]
139 PIN_AG20 GPIO_0[33]
140 PIN_AG18 GPIO_0[34]

```

141	PIN_AJ21	GPIO_0[35]
142		
143	PIN_AB17	GPIO_1[0]
144	PIN_AA21	GPIO_1[1]
145	PIN_AB21	GPIO_1[2]
146	PIN_AC23	GPIO_1[3]
147	PIN_AD24	GPIO_1[4]
148	PIN_AE23	GPIO_1[5]
149	PIN_AE24	GPIO_1[6]
150	PIN_AF25	GPIO_1[7]
151	PIN_AF26	GPIO_1[8]
152	PIN_AG25	GPIO_1[9]
153	PIN_AG26	GPIO_1[10]
154	PIN_AH24	GPIO_1[11]
155	PIN_AH27	GPIO_1[12]
156	PIN_AJ27	GPIO_1[13]
157	PIN_AK29	GPIO_1[14]
158	PIN_AK28	GPIO_1[15]
159	PIN_AK27	GPIO_1[16]
160	PIN_AJ26	GPIO_1[17]
161	PIN_AK26	GPIO_1[18]
162	PIN_AH25	GPIO_1[19]
163	PIN_AJ25	GPIO_1[20]
164	PIN_AJ24	GPIO_1[21]
165	PIN_AK24	GPIO_1[22]
166	PIN_AG23	GPIO_1[23]
167	PIN_AK23	GPIO_1[24]
168	PIN_AH23	GPIO_1[25]
169	PIN_AK22	GPIO_1[26]
170	PIN_AJ22	GPIO_1[27]
171	PIN_AH22	GPIO_1[28]
172	PIN_AG22	GPIO_1[29]
173	PIN_AF24	GPIO_1[30]
174	PIN_AF23	GPIO_1[31]
175	PIN_AE22	GPIO_1[32]
176	PIN_AD21	GPIO_1[33]
177	PIN_AA20	GPIO_1[34]
178	PIN_AC22	GPIO_1[35]
179		
180	PIN_AE26	HEX0[0]
181	PIN_AE27	HEX0[1]
182	PIN_AE28	HEX0[2]
183	PIN_AG27	HEX0[3]
184	PIN_AF28	HEX0[4]
185	PIN_AG28	HEX0[5]
186	PIN_AH28	HEX0[6]
187		
188	PIN_AJ29	HEX1[0]
189	PIN_AH29	HEX1[1]
190	PIN_AH30	HEX1[2]
191	PIN_AG30	HEX1[3]
192	PIN_AF29	HEX1[4]
193	PIN_AF30	HEX1[5]
194	PIN_AD27	HEX1[6]
195		
196	PIN_AB23	HEX2[0]
197	PIN_AE29	HEX2[1]
198	PIN_AD29	HEX2[2]
199	PIN_AC28	HEX2[3]
200	PIN_AD30	HEX2[4]
201	PIN_AC29	HEX2[5]
202	PIN_AC30	HEX2[6]
203		
204	PIN_AD26	HEX3[0]
205	PIN_AC27	HEX3[1]
206	PIN_AD25	HEX3[2]
207	PIN_AC25	HEX3[3]
208	PIN_AB28	HEX3[4]
209	PIN_AB25	HEX3[5]
210	PIN_AB22	HEX3[6]
211		
212	PIN_AA24	HEX4[0]
213	PIN_Y23	HEX4[1]
214	PIN_Y24	HEX4[2]

215	PIN_W22	HEX4[3]
216	PIN_W24	HEX4[4]
217	PIN_V23	HEX4[5]
218	PIN_W25	HEX4[6]
219		
220	PIN_V25	HEX5[0]
221	PIN_AA28	HEX5[1]
222	PIN_Y27	HEX5[2]
223	PIN_AB27	HEX5[3]
224	PIN_AB26	HEX5[4]
225	PIN_AA26	HEX5[5]
226	PIN_AA25	HEX5[6]
227		
228	PIN_AA30	IRDA_RXD
229	PIN_AB30	IRDA_TXD
230		
231	PIN_AA14	KEY[0]
232	PIN_AA15	KEY[1]
233	PIN_W15	KEY[2]
234	PIN_Y16	KEY[3]
235		
236	PIN_V16	LEDR[0]
237	PIN_W16	LEDR[1]
238	PIN_V17	LEDR[2]
239	PIN_V18	LEDR[3]
240	PIN_W17	LEDR[4]
241	PIN_W19	LEDR[5]
242	PIN_Y19	LEDR[6]
243	PIN_W20	LEDR[7]
244	PIN_W21	LEDR[8]
245	PIN_Y21	LEDR[9]
246		
247	PIN_AD7	PS2_CLK
248	PIN_AD9	PS2_CLK2
249	PIN_AE7	PS2_DAT
250	PIN_AE9	PS2_DAT2
251		
252	PIN_AB12	SW[0]
253	PIN_AC12	SW[1]
254	PIN_AF9	SW[2]
255	PIN_AF10	SW[3]
256	PIN_AD11	SW[4]
257	PIN_AD12	SW[5]
258	PIN_AE11	SW[6]
259	PIN_AC9	SW[7]
260	PIN_AD10	SW[8]
261	PIN_AE12	SW[9]
262		
263	PIN_H15	TD_CLK27
264	PIN_D2	TD_DATA[0]
265	PIN_B1	TD_DATA[1]
266	PIN_E2	TD_DATA[2]
267	PIN_B2	TD_DATA[3]
268	PIN_D1	TD_DATA[4]
269	PIN_E1	TD_DATA[5]
270	PIN_C2	TD_DATA[6]
271	PIN_B3	TD_DATA[7]
272	PIN_A5	TD_HS
273	PIN_F6	TD_RESET_N
274	PIN_A3	TD_VS
275		
276	PIN_A13	VGA_R[0]
277	PIN_C13	VGA_R[1]
278	PIN_E13	VGA_R[2]
279	PIN_B12	VGA_R[3]
280	PIN_C12	VGA_R[4]
281	PIN_D12	VGA_R[5]
282	PIN_E12	VGA_R[6]
283	PIN_F13	VGA_R[7]
284		
285	PIN_J9	VGA_G[0]
286	PIN_J10	VGA_G[1]
287	PIN_H12	VGA_G[2]
288	PIN_G10	VGA_G[3]

```

289     PIN_G11 VGA_G[4]
290     PIN_G12 VGA_G[5]
291     PIN_F11 VGA_G[6]
292     PIN_E11 VGA_G[7]
293
294     PIN_B13 VGA_B[0]
295     PIN_G13 VGA_B[1]
296     PIN_H13 VGA_B[2]
297     PIN_F14 VGA_B[3]
298     PIN_H14 VGA_B[4]
299     PIN_F15 VGA_B[5]
300     PIN_G15 VGA_B[6]
301     PIN_J14 VGA_B[7]
302
303     PIN_A11 VGA_CLK
304     PIN_B11 VGA_HS
305     PIN_D11 VGA_VS
306     PIN_F10 VGA_BLANK_N
307     PIN_C10 VGA_SYNC_N
308 } {
309     set_location_assignment $pin -to $port
310     set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
311 }
312
313 # HPS assignments
314
315 # 3.3-V LVTTL pins
316 foreach port {
317     HPS_CONV_USB_N
318     HPS_ENET_GTX_CLK
319     HPS_ENET_INT_N
320     HPS_ENET_MDC
321     HPS_ENET_MDIO
322     HPS_ENET_RX_CLK
323     HPS_ENET_RX_DATA[0]
324     HPS_ENET_RX_DATA[1]
325     HPS_ENET_RX_DATA[2]
326     HPS_ENET_RX_DATA[3]
327     HPS_ENET_RX_DV
328     HPS_ENET_TX_DATA[0]
329     HPS_ENET_TX_DATA[1]
330     HPS_ENET_TX_DATA[2]
331     HPS_ENET_TX_DATA[3]
332     HPS_ENET_TX_EN
333     HPS_GSENSOR_INT
334     HPS_I2C1_SCLK
335     HPS_I2C1_SDAT
336     HPS_I2C2_SCLK
337     HPS_I2C2_SDAT
338     HPS_I2C_CONTROL
339     HPS_KEY
340     HPS_LED
341     HPS_LTC_GPIO
342     HPS_SD_CLK
343     HPS_SD_CMD
344     HPS_SD_DATA[0]
345     HPS_SD_DATA[1]
346     HPS_SD_DATA[2]
347     HPS_SD_DATA[3]
348     HPS_SPIM_CLK
349     HPS_SPIM_MISO
350     HPS_SPIM_MOSI
351     HPS_SPIM_SS
352     HPS_UART_RX
353     HPS_UART_TX
354     HPS_USB_CLKOUT
355     HPS_USB_DATA[0]
356     HPS_USB_DATA[1]
357     HPS_USB_DATA[2]
358     HPS_USB_DATA[3]
359     HPS_USB_DATA[4]
360     HPS_USB_DATA[5]
361     HPS_USB_DATA[6]
362     HPS_USB_DATA[7]

```

```

363     HPS_USB_DIR
364     HPS_USB_NXT
365     HPS_USB_STP
366 } {
367     set_instance_assignment -name IO_STANDARD "3.3-V LVTTL" -to $port
368 }
369
370 # There are a lot of settings for the HPS_DDR3 interface not listed here.
371 # Instead, the
372 #
373 # soc_system/synthesis/submodules/hps_sdram_p0_pin_assignments.tcl
374 #
375 # script generated by qsys adds that information. However, quartus_map
376 # must be run before this .tcl script may run because the script
377 # relies on being able to look at the (HPS) netlist to determine which
378 # pins to constrain
379
380 set sdcFilename "${project}.sdc"
381
382 set_global_assignment -name SDC_FILE $sdcFilename
383
384 set sdcf [open $sdcFilename "w"]
385 puts $sdcf {
386     foreach {clock port} {
387         clock_50_1 CLOCK_50
388         clock_50_2 CLOCK2_50
389         clock_50_3 CLOCK3_50
390         clock_50_4 CLOCK4_50
391     } {
392         create_clock -name $clock -period 20ns [get_ports $port]
393     }
394
395     create_clock -name clock_27_1 -period 37 [get_ports TD_CLK27]
396
397     derive_pll_clocks -create_base_clocks
398     derive_clock_uncertainty
399 }
400 close $sdcf
401
402 project_close

```

2 Software (HPS Driver + Userspace Server)

SW/raytracer_batch.h

```

1  /* raytracer_batch.h */
2  #ifndef RAYTRACER_BATCH_H
3  #define RAYTRACER_BATCH_H
4
5  #include <linux/types.h>
6  #include <linux/ioctl.h>
7
8  /* Batch size (lanes per FPGA compute kick). Must equal the FPGA's
9   * raytracer_0.N parameter - these are not validated against each other,
10  * a mismatch silently corrupts data.
11  *
12  * To raise above 480, all of the following must be updated together:
13  *
14  *   THIS FILE:
15  *     RTB_N (this define)
16  *
17  *   FPGA, parameter values:
18  *     soc_system.qsys          <parameter name="N" value="...">
19  *     raytracer_hw.tcl         add_parameter N INTEGER ... (default)
20  *     raytracer.sv             parameter int N = ... (default)
21  *
22  *   FPGA, things that scale with N (currently sized for N 510):
23  *     raytracer.sv             `8'd240` literal in DMA_IDLE init of
24  *                             beats_remaining; should be (N/2)
25  *     raytracer.sv             `8'd240` literal driving m_avm_burstcount;
26  *                             same fix

```

```

27 * raytracer.sv          `beats_remaining` is 8 bits - widen to
28 *                      9+ bits if N/2 > 255 (i.e. N 512)
29 * raytracer_hw.tcl      m_avm_burstcount port width is 8 - widen
30 *                      to 9+ if N 512
31 *
32 * FPGA, address span (currently sized for N+11 1024):
33 *   The Avalon-MM slave's `address` is 10 bits = 1024 word-aperture.
34 *   Control regs use 11 words (0..10), color mirror uses N words
35 *   (11..N+10). At N=1013 this hits the limit.
36 * raytracer_hw.tcl      avalon_slave_0 address port width
37 * raytracer.sv          address [9:0] in module port list
38 *
39 * AXI3 4 KB burst-boundary rule (per ARM IHI 0022):
40 *   At N=1024 the 240-beat -> 4096-byte burst spans the full page.
41 *   At N>1024 (>4096 B) the qsys bridge can't split a single Avalon
42 *   burst across a 4 KB boundary; the master must issue >1 Avalon
43 *   burst per batch. alloc_pages_exact always returns 4 KB-aligned,
44 *   so the buffer base is fine - it's the in-burst byte count that
45 *   matters.
46 *
47 * render_frame.c:
48 *   BATCHES = WIDTH / RTB_N. With WIDTH=480, BATCHES=1 at N=480 and
49 *   0 at N>480 - the rendering loop wouldn't run. Going above 480
50 *   means batches now span multiple rows; rework the pixel-address
51 *   stride and the row-write logic.
52 *
53 * Compute-side capacity:
54 *   rtl_batch_pipe latency scales linearly with BATCH (~37*N/K
55 *   cycles, K=3). At N=480 ~60 us; at N=960 ~120 us. Already faster
56 *   than the DMA-out + commit at any N up through ~1000 lanes; if
57 *   pushing further, may want K=4 to keep compute off the critical
58 *   path.
59 */
60 #define RTB_N 480
61
62 /* Block until the in-flight FRAME_DONE asserts, clear the sticky bit, and
63 * invalidate the just-finished slot's CPU caches so userspace can read
64 * cached. Returns the index (0 or 1) of the slot whose render just
65 * committed; userspace reads at `frame + slot * (WIDTH*HEIGHT)`.
66 *
67 * Replaces the userspace `while (csr->status & FRAME_DONE) ;` poll +
68 * `csr->control = CLEAR_FRAME_DONE` write that the mmap-based path
69 * used to do inline. The reason this is a kernel ioctl (rather than
70 * pure userspace polling) is the cache invalidate: the frame buffer
71 * is now allocated cached for ~6x faster CPU reads, and on Cortex-A9
72 * + PL310 the L2 invalidate after DMA-write is privileged. The poll
73 * + clear are folded in to keep userspace at one syscall per frame. */
74 #define RTB_IOCTL_WAIT_FRAME_IOR('r', 3, __u32)
75
76 /* =====
77 * mmap interface (Stage 4 - HW per-frame auto-kick + frame-level double
78 * buffering with pipelined render).
79 * =====
80 *
81 * Userspace mmaps two regions, each one page-aligned offset:
82 *
83 *   mmap(fd, off=RTB_MMAP_CSR_OFFSET)  → CSR page, RW, uncached.
84 *                                       Cast to volatile rtb_csrs *.
85 *   mmap(fd, off=RTB_MMAP_FRAME_OFFSET) → 2-slot frame ring, R, cached
86 *                                       (write-back). Each slot holds
87 *                                       one full WIDTH*HEIGHT*4 frame
88 *                                       in packed {00, R, G, B} per
89 *                                       pixel, indexed (y*WIDTH+x).
90 *                                       Total mapping is 2 * frame size.
91 *                                       Coherence with FPGA writes is
92 *                                       maintained by RTB_IOCTL_WAIT_FRAME
93 *                                       which invalidates the just-
94 *                                       finished slot before returning.
95 *
96 * Stage 4 double-buffer slot indexing:
97 *   - HW maintains an internal write_slot bit (1 = slot 1, 0 = slot 0)
98 *     that toggles after each FRAME_DONE_PULSE. Frame N lands in slot
99 *     N%2.
100 *   - SW receives the just-finished slot index from RTB_IOCTL_WAIT_FRAME

```

```

101 * and reads `frame + slot * (WIDTH * HEIGHT)` (or, equivalently,
102 * byte offset = slot * RTB_FRAME_BYTES_PER_SLOT).
103 *
104 * Pipelined wire protocol (server.c <-> client.py): pipeline-depth-2.
105 * - Client primes: send light 0 + light 1 immediately on connect.
106 * - Client steady: after recv frame N, send light N+2.
107 * - Server kicks frame N+1 BEFORE reading slot N, so HW renders
108 * N+1 in parallel with SW reading + sending N. Total per-frame
109 * wall clock = max(HW_render, SW_read_send) HW_render.
110 *
111 * Driver programs FRAME_BUF_BASE (= base of slot 0) and DMA_CTRL[0]
112 * at probe; userspace must NOT write those CSR fields. The struct
113 * exposes frame_buf_base for completeness/debug only.
114 */
115
116 #define RTB_MMAP_PAGE_SIZE 4096u
117 #define RTB_MMAP_CSR_OFFSET (0u * RTB_MMAP_PAGE_SIZE)
118 #define RTB_MMAP_FRAME_OFFSET (1u * RTB_MMAP_PAGE_SIZE)
119
120 /* Stage 4: bytes per slot (one full frame). The mmap'd frame region is
121 * 2 * RTB_FRAME_BYTES_PER_SLOT total; SW indexes into the active slot. */
122 #define RTB_FRAME_BYTES_PER_SLOT (480u * 360u * 4u)
123
124 /* CSR layout - byte-exact mirror of the Avalon-MM slave register map
125 * declared in raytracer.sv (localparam ADDR_* block). Cast a CSR mmap
126 * to `volatile struct rtb_csrs *` and access fields directly.
127 *
128 * Per-frame loop: WAIT_FRAME → write all scene CSRs (light, spheres,
129 * camera basis, visual params) → FRAME_START. SW must finish all writes
130 * before kicking and never write CSRs while a frame is in flight; HW does
131 * not double-buffer scene state.
132 *
133 * frame_buf_base is the DDR3 physical address of the 2-slot frame ring
134 * (Stage 4): a 2*WIDTH*HEIGHT*4 contiguous DDR3 region the FPGA scatters
135 * batch DMAs into. HW alternates which slot it writes on each
136 * FRAME_DONE_PULSE. Driver programs it at probe; SW must NOT write
137 * this field.
138 *
139 * Camera basis (right.y hardcoded 0 - no roll). SW computes from yaw +
140 * pitch and ships these once per frame. HW applies
141 * raw_dir = right.px + up.py + fwd in stage A. cam_y locked at 1.5.
142 *
143 * max_depth caps recursion (0..7; 0 disables reflection). reflect_shift_{0,1}
144 * set per-sphere power-of-2 reflectivity (each bounce off sphere X
145 * attenuates by >> reflect_shift_X). Defaults: depth=3, blue=25% (RS=2),
146 * red=50% (RS=1). Cold-start defaults reproduce the historical hardcoded
147 * scene exactly: cam at (0, 1.5, -8), identity basis (right=x, up=y, fwd=z).
148 *
149 * Q-format note: all signed-Q fields use FRAC_BITS = 13 (Q14.13). To
150 * convert a float to wire format: int32_t v = (int32_t)(f * (1 << 13));
151 * Negative values use two's-complement; HW takes writedata[WORD-1:0] so
152 * masking the upper bits is unnecessary.
153 */
154 struct rtb_csrs {
155     __u32 control; /* 0x00 W: bit2 frame_start, bit3 clear_frame_done */
156     __u32 status; /* 0x04 R: bit2 frame_done_sticky,
157                  * bit3 frame_running, bit4 write_slot */
158     __u32_rsvd_pixel_x; /* 0x08 reserved (HW frame-mode generator owns the
159                  * internal pixel_x_reg; SW write path removed) */
160     __u32_rsvd_pixel_y; /* 0x0c reserved (same as 0x08) */
161     __u32 light_x; /* 0x10 W: signed Q12, low 26 bits */
162     __u32 light_y; /* 0x14 W */
163     __u32 light_z; /* 0x18 W */
164     __u32 r0_sq; /* 0x1c W: signed Q FRAC_BITS sphere-0 radius^2
165                  * (SW pre-squares r before writing). */
166     __u32 r1_sq; /* 0x20 W: signed Q FRAC_BITS sphere-1 radius^2 */
167     __u32 dma_ctrl; /* 0x24 W: bit0 enable - DRIVER-OWNED */
168     __u32 dma_status; /* 0x28 R: bit0 busy, bit1 err, bit2 done */
169     __u32_rsvd_2c; /* 0x2c reserved (formerly LIGHT_COMMIT - collapsed
170                  * with NEXT/ACTIVE shadow regs) */
171     __u32_rsvd_30_38[3]; /* 0x30/0x34/0x38 reserved (former LIGHT_*_ACTIVE
172                  * debug readback) */
173     __u32 frame_buf_base; /* 0x3c W: DDR3 phys addr - DRIVER-OWNED */
174     __u32 scl_x; /* 0x40 W: signed Q12 sphere-1 center X */

```



```

175 __u32 sc1_y;          /* 0x44 W                      Y */
176 __u32 sc1_z;          /* 0x48 W                      Z */
177 __u32 max_depth;      /* 0x4c W: 3-bit max recursion depth (default 3) */
178 __u32 reflect_shift_0; /* 0x50 W: 5-bit pow2 reflectivity, sphere 0 (default 2 = 25%) */
179 __u32 reflect_shift_1; /* 0x54 W: 5-bit pow2 reflectivity, sphere 1 (default 1 = 50%) */
180 /* Camera position (cam_y is locked in HW at 1.5 - no CSR). */
181 __u32 cam_x;          /* 0x58 W: signed Q12 camera origin X (default 0.0) */
182 __u32 cam_z;          /* 0x5c W: signed Q12 camera origin Z (default -8.0) */
183 /* Camera basis (right.y hardcoded 0 - no roll). */
184 __u32 right_x;        /* 0x60 W: signed Q12 right.x (default -1.0) */
185 __u32 right_z;        /* 0x64 W: signed Q12 right.z (default 0.0) */
186 __u32 up_x;           /* 0x68 W: signed Q12 up.x (default 0.0) */
187 __u32 up_y;           /* 0x6c W: signed Q12 up.y (default 1.0) */
188 __u32 up_z;           /* 0x70 W: signed Q12 up.z (default 0.0) */
189 __u32 fwd_x;          /* 0x74 W: signed Q12 fwd.x (default 0.0) */
190 __u32 fwd_y;          /* 0x78 W: signed Q12 fwd.y (default 0.0) */
191 __u32 fwd_z;          /* 0x7c W: signed Q12 fwd.z (default 1.0) */
192 /* Visual-upgrades CSRs. Defaults reproduce the legacy hardcoded scene
193  * byte-equal so SW that doesn't write any of these still gets the old image. */
194 __u32 sc0_x;          /* 0x80 W: signed Q12 sphere-0 center X (default 0.0) */
195 __u32 sc0_y;          /* 0x84 W                      Y (default 1.0) */
196 __u32 sc0_z;          /* 0x88 W                      Z (default 0.0) */
197 __u32 pci_r;          /* 0x8c W: signed Q12 floor color #1 R (default 0.9) */
198 __u32 pci_g;          /* 0x90 W                      G (default 0.9) */
199 __u32 pci_b;          /* 0x94 W                      B (default 0.9) */
200 __u32 pc2_r;          /* 0x98 W: signed Q12 floor color #2 R (default 0.4) */
201 __u32 pc2_g;          /* 0x9c W                      G (default 0.4) */
202 __u32 pc2_b;          /* 0xa0 W                      B (default 0.4) */
203 __u32 scol0_r;        /* 0xa4 W: signed Q12 sphere-0 color R (default 0.2) */
204 __u32 scol0_g;        /* 0xa8 W                      G (default 0.3) */
205 __u32 scol0_b;        /* 0xac W                      B (default 0.8) */
206 __u32 scol1_r;        /* 0xb0 W: signed Q12 sphere-1 color R (default 0.8) */
207 __u32 scol1_g;        /* 0xb4 W                      G (default 0.3) */
208 __u32 scol1_b;        /* 0xb8 W                      B (default 0.2) */
209 __u32 floor_mode;     /* 0xbc W: 2-bit floor pattern selector
210  * 00 = checker (default), 01 = Sierpinski,
211  * 10 = stripes, 11 = fine checker */
212 __u32 floor_lim_x;    /* 0xc0 W: signed Q12 floor X half-extent
213  * (default = max-positive infinite) */
214 __u32 floor_lim_z;    /* 0xc4 W: signed Q12 floor Z half-extent */
215 /* Cone primitive (Y-axis cone, apex on top, opens downward over
216  * y [apex.y - height, apex.y]). Surface
217  *  $(x - Ax)^2 + (z - Az)^2 = k\_sq \cdot (y - Ay)^2$ 
218  * Default height=0 disables the cone (slab is empty). reflect_shift
219  * mirrors the per-sphere REFLECT_SHIFT_* CSRs. Sun-billboard uses
220  * light/light| automatically - no separate sun CSRs. */
221 __u32 cone_x;         /* 0xc8 W: signed Q12 cone apex X (default 0) */
222 __u32 cone_y;         /* 0xcc W: signed Q12 cone apex Y (default 0) */
223 __u32 cone_z;         /* 0xd0 W: signed Q12 cone apex Z (default 0) */
224 __u32 cone_k_sq;      /* 0xd4 W: signed Q12  $\tan^2(\text{half-angle})$  (default 1.0 = 45°) */
225 __u32 cone_height;    /* 0xd8 W: signed Q12 cone height (default 0 = disabled) */
226 __u32 cone_col_r;     /* 0xdc W: signed Q12 cone color R (default 0.5) */
227 __u32 cone_col_g;     /* 0xe0 W                      G (default 0.5) */
228 __u32 cone_col_b;     /* 0xe4 W                      B (default 0.5) */
229 __u32 reflect_shift_cone; /* 0xe8 W: 5-bit pow2 reflectivity, cone (default 0 = matte) */
230 __u32 cone_norm_factor; /* 0xec W: signed Q12,  $\sqrt{k^2 \cdot (1+k^2)}$ . SW must
231  * pre-compute this from cone_k_sq -
232  * lets HW recover the cone surface
233  * |normal| with one mul instead of
234  * three squarings + a sqrt. Default
235  * 5793  $\sqrt{2 \cdot 2^{12}}$  (= sqrt(2) for  $k^2=1$ ). */
236 __u32_rsvd_f0_fc[4]; /* 0xf0..0xfc reserved (padding to align color
237  * mirror to word 64 / byte 0x100) */
238 __u32 color[];       /* 0x100+ R: BRAM mirror, debug only. */
239 };
240
241 /* Bit defines for the polled fields. */
242 #define RTB_CONTROL_FRAME_START 0x00000004u
243 #define RTB_CONTROL_CLEAR_FRAME_DONE 0x00000008u
244 #define RTB_STATUS_FRAME_DONE 0x00000004u
245 #define RTB_STATUS_FRAME_RUNNING 0x00000008u
246 #define RTB_STATUS_WRITE_SLOT 0x00000010u /* Stage 4: current write slot */
247
248 /* mmap layout returned by RTB_IOCTL_GET_LAYOUT. One-time call at

```

```

249  * userspace setup; all values are stable for the device's lifetime. */
250  struct rtb_layout {
251      __u32 csr_size;          /* size of CSR mmap (currently 4 KB) */
252      __u32 frame_buf_bytes;   /* size of the frame buffer mmap (WIDTH*HEIGHT*4) */
253      __u32 csr_offset;        /* RTB_MMAP_CSR_OFFSET (echoed for sanity) */
254      __u32 frame_buf_offset; /* RTB_MMAP_FRAME_OFFSET */
255  };
256
257  #define RTB_IOCTL_GET_LAYOUT _IOR('r', 2, struct rtb_layout)
258
259  #endif

```

SW/raytrace_batch_drv.c

```

1  /* raytracer_batch_drv.c */
2
3  #include <linux/module.h>
4  #include <linux/init.h>
5  #include <linux/errno.h>
6  #include <linux/kernel.h>
7  #include <linux/platform_device.h>
8  #include <linux/miscdevice.h>
9  #include <linux/io.h>
10 #include <linux/of.h>
11 #include <linux/of_address.h>
12 #include <linux/fs.h>
13 #include <linux/uaccess.h>
14 #include <linux/delay.h>
15 #include <linux/jiffies.h>
16 #include <linux/ktime.h>
17 #include <linux/slab.h>
18 #include <linux/dma-mapping.h>
19
20 #include "raytracer_batch.h"
21
22 #define DRIVER_NAME "raytracer"
23
24 /* CSR offsets the driver itself accesses. The full register layout lives in
25  * `struct rtb_csrs` (raytracer_batch.h); userspace pokes everything else via
26  * the CSR mmap. */
27 #define REG_CONTROL          0x00
28 #define REG_STATUS           0x04
29 #define REG_DMA_CTRL         0x24 /* bit0 ENABLE, bit1 CLEAR_ERR, bit2 CLEAR_DONE */
30 #define REG_FRAME_BUF_BASE  0x3c /* W: DDR3 phys addr of contiguous frame buffer */
31
32 #define CONTROL_FRAME_START   0x00000004u
33 #define CONTROL_CLEAR_FRAME_DONE 0x00000008u
34
35 #define STATUS_FRAME_DONE     0x00000004u
36 #define STATUS_FRAME_RUNNING 0x00000008u
37
38 #define DMA_CTRL_ENABLE       0x00000001u
39 #define DMA_CTRL_CLEAR_ERR    0x00000002u
40 #define DMA_CTRL_CLEAR_DONE   0x00000004u
41
42 #define RTB_WIDTH             480u
43 #define RTB_HEIGHT            360u
44 #define RTB_TIMEOUT_MS        1000u
45
46 #define RTB_FRAME_BYTES_PER_SLOT (RTB_WIDTH * RTB_HEIGHT * 4u)
47 #define RTB_FRAME_BYTES        (2u * RTB_FRAME_BYTES_PER_SLOT)
48
49 struct raytracer_batch_dev {
50     struct resource res;
51     void __iomem *virtbase;
52     struct device *dma_dev; /* parent device for streaming-DMA APIs */
53     void *cpu_frame; /* cached kernel VA (alloc_pages_exact) */
54     dma_addr_t dma_frame; /* bus addr sent to FPGA via FRAME_BUF_BASE */
55 } device_state;
56
57
58 /* RTB_IOCTL_WAIT_FRAME - block until FRAME_DONE, clear the sticky bit,

```

```

59  * invalidate the just-finished slot's CPU caches, and return its slot
60  * index (0 or 1). The L2 invalidate on Cortex-A9 + PL310 is privileged,
61  * which is why poll + clear + invalidate live in one ioctl instead of
62  * in userspace. */
63  static long raytracer_batch_ioctl_wait_frame(unsigned long arg)
64  {
65      unsigned long deadline = jiffies + msecs_to_jiffies(RTB_TIMEOUT_MS);
66      u32 status;
67      __u32 done_slot;
68
69      while (1) {
70          status = ioread32(device_state.virtbase + REG_STATUS);
71          if (status & STATUS_FRAME_DONE)
72              break;
73          if (time_after_eq(jiffies, deadline))
74              return -ETIMEDOUT;
75      }
76
77      done_slot = (status & RTB_STATUS_WRITE_SLOT) ? 0u : 1u;
78
79      iowrite32(CONTROL_CLEAR_FRAME_DONE,
80               device_state.virtbase + REG_CONTROL);
81
82      dma_sync_single_for_cpu(device_state.dma_dev,
83                              device_state.dma_frame
84                              + done_slot * RTB_FRAME_BYTES_PER_SLOT,
85                              RTB_FRAME_BYTES_PER_SLOT,
86                              DMA_FROM_DEVICE);
87
88      if (copy_to_user((void __user *)arg, &done_slot, sizeof(done_slot)))
89          return -EFAULT;
90      return 0;
91  }
92
93  /* RTB_IOCTL_GET_LAYOUT - one-time mmap-region descriptor. */
94  static long raytracer_batch_ioctl_get_layout(unsigned long arg)
95  {
96      struct rtb_layout layout;
97
98      memset(&layout, 0, sizeof(layout));
99      layout.csr_size      = RTB_MMAP_PAGE_SIZE;
100     layout.frame_buf_bytes = RTB_FRAME_BYTES;
101     layout.csr_offset      = RTB_MMAP_CSR_OFFSET;
102     layout.frame_buf_offset = RTB_MMAP_FRAME_OFFSET;
103
104     if (copy_to_user((void __user *)arg, &layout, sizeof(layout)))
105         return -EFAULT;
106     return 0;
107  }
108
109  static long raytracer_batch_ioctl(struct file *file, unsigned int cmd,
110                                  unsigned long arg)
111  {
112     if (cmd == RTB_IOCTL_GET_LAYOUT)
113         return raytracer_batch_ioctl_get_layout(arg);
114
115     if (cmd == RTB_IOCTL_WAIT_FRAME)
116         return raytracer_batch_ioctl_wait_frame(arg);
117
118     return -EINVAL;
119  }
120
121  /* mmap handler - vm_pgoff selects which region:
122   * pgoff 0 → CSR window (uncached, RW). CSR side effects require no caching.
123   * pgoff 1 → frame buffer (cached, R). Coherence with FPGA DMA writes is
124   * maintained by RTB_IOCTL_WAIT_FRAME's per-slot invalidate. */
125  static int raytracer_batch_mmap(struct file *file, struct vm_area_struct *vma)
126  {
127     unsigned long pgoff = vma->vm_pgoff;
128     size_t size = vma->vm_end - vma->vm_start;
129     int ret;
130
131     switch (pgoff) {
132     case 0: {

```

```

133     size_t csr_phys_size = resource_size(&device_state.res);
134
135     if (size > csr_phys_size)
136         return -EINVAL;
137
138     vma->vm_pgoff = 0;
139     vma->vm_page_prot = pgprot_noncached(vma->vm_page_prot);
140     ret = io_remap_pfn_range(vma, vma->vm_start,
141                             device_state.res.start >> PAGE_SHIFT,
142                             size, vma->vm_page_prot);
143     return ret;
144 }
145 case 1: {
146     unsigned long pfn;
147
148     if (size > PAGE_ALIGN(RTB_FRAME_BYTES))
149         return -EINVAL;
150
151     pfn = virt_to_phys(device_state.cpu_frame) >> PAGE_SHIFT;
152     ret = remap_pfn_range(vma, vma->vm_start, pfn, size,
153                          vma->vm_page_prot);
154     return ret;
155 }
156 default:
157     return -EINVAL;
158 }
159 }
160
161 static const struct file_operations raytracer_batch_fops = {
162     .owner = THIS_MODULE,
163     .unlocked_ioctl = raytracer_batch_ioctl,
164     .mmap = raytracer_batch_mmap,
165 };
166
167 static struct miscdevice raytracer_batch_misc_device = {
168     .minor = MISC_DYNAMIC_MINOR,
169     .name = DRIVER_NAME,
170     .fops = &raytracer_batch_fops,
171 };
172
173 static int __init raytracer_batch_probe(struct platform_device *pdev)
174 {
175     int ret_value;
176
177     ret_value = misc_register(&raytracer_batch_misc_device);
178     if (ret_value)
179         return ret_value;
180
181     ret_value = of_address_to_resource(pdev->dev.of_node, 0, &device_state.res);
182     if (ret_value) {
183         ret_value = -ENOENT;
184         goto deregister_out;
185     }
186
187     if (!request_mem_region(device_state.res.start,
188                            resource_size(&device_state.res),
189                            DRIVER_NAME)) {
190         ret_value = -EBUSY;
191         goto deregister_out;
192     }
193
194     device_state.virtbase = of_iomap(pdev->dev.of_node, 0);
195     if (!device_state.virtbase) {
196         ret_value = -ENOMEM;
197         goto release_out;
198     }
199
200     device_state.dma_dev = &pdev->dev;
201     ret_value = dma_set_mask_and_coherent(device_state.dma_dev,
202                                           DMA_BIT_MASK(32));
203     if (ret_value)
204         goto unmap_out;
205
206     device_state.cpu_frame = alloc_pages_exact(RTB_FRAME_BYTES, GFP_KERNEL);

```

```

207     if (!device_state.cpu_frame) {
208         ret_value = -ENOMEM;
209         goto unmap_out;
210     }
211
212     device_state.dma_frame = dma_map_single(device_state.dma_dev,
213         device_state.cpu_frame,
214         RTB_FRAME_BYTES,
215         DMA_FROM_DEVICE);
216     if (dma_mapping_error(device_state.dma_dev, device_state.dma_frame)) {
217         ret_value = -ENOMEM;
218         goto free_frame_out;
219     }
220
221     if ((u64)device_state.dma_frame >= 0x80000000ull) {
222         pr_err(DRIVER_NAME ": DMA addr unsuitable frame=%pad\n",
223             &device_state.dma_frame);
224         ret_value = -EINVAL;
225         goto unmap_frame_out;
226     }
227
228     iowrite32((u32)device_state.dma_frame,
229         device_state.virtbase + REG_FRAME_BUF_BASE);
230     iowrite32(DMA_CTRL_CLEAR_DONE | DMA_CTRL_CLEAR_ERR,
231         device_state.virtbase + REG_DMA_CTRL);
232     iowrite32(DMA_CTRL_ENABLE,
233         device_state.virtbase + REG_DMA_CTRL);
234
235     pr_info(DRIVER_NAME ": frame buffer %pad (%u B)\n",
236         &device_state.dma_frame, RTB_FRAME_BYTES);
237
238     return 0;
239
240 unmap_frame_out:
241     dma_unmap_single(device_state.dma_dev, device_state.dma_frame,
242         RTB_FRAME_BYTES, DMA_FROM_DEVICE);
243 free_frame_out:
244     free_pages_exact(device_state.cpu_frame, RTB_FRAME_BYTES);
245 unmap_out:
246     iounmap(device_state.virtbase);
247 release_out:
248     release_mem_region(device_state.res.start,
249         resource_size(&device_state.res));
250 deregister_out:
251     misc_deregister(&raytracer_batch_misc_device);
252     return ret_value;
253 }
254
255 static int raytracer_batch_remove(struct platform_device *pdev)
256 {
257     iowrite32(0u, device_state.virtbase + REG_DMA_CTRL);
258     dma_unmap_single(device_state.dma_dev, device_state.dma_frame,
259         RTB_FRAME_BYTES, DMA_FROM_DEVICE);
260     free_pages_exact(device_state.cpu_frame, RTB_FRAME_BYTES);
261     iounmap(device_state.virtbase);
262     release_mem_region(device_state.res.start,
263         resource_size(&device_state.res));
264     misc_deregister(&raytracer_batch_misc_device);
265     return 0;
266 }
267
268 #ifdef CONFIG_OF
269 static const struct of_device_id raytracer_batch_of_match[] = {
270     { .compatible = "csee4840,raytracer-1.0" },
271     { },
272 };
273 MODULE_DEVICE_TABLE(of, raytracer_batch_of_match);
274 #endif
275
276 static struct platform_driver raytracer_batch_driver = {
277     .driver = {
278         .name = DRIVER_NAME,
279         .owner = THIS_MODULE,
280         .of_match_table = of_match_ptr(raytracer_batch_of_match),

```

```

281     },
282     .remove = __exit_p(raytracer_batch_remove),
283 };
284
285 static int __init raytracer_batch_init(void)
286 {
287     pr_info(DRIVER_NAME ": init\n");
288     return platform_driver_probe(&raytracer_batch_driver,
289                                raytracer_batch_probe);
290 }
291
292 static void __exit raytracer_batch_exit(void)
293 {
294     platform_driver_unregister(&raytracer_batch_driver);
295     pr_info(DRIVER_NAME ": exit\n");
296 }
297
298 module_init(raytracer_batch_init);
299 module_exit(raytracer_batch_exit);
300
301 MODULE_LICENSE("GPL");
302 MODULE_DESCRIPTION("raytracer batch driver");

```

SW/fpga/server.c

```

1  /*
2   * server.c - TCP bridge for the FPGA raytracer batch device.
3   *
4   * Wire protocol (pipeline-depth-2):
5   *   Client → server on connect: inputs 0, inputs 1 (priming).
6   *   Steady state per frame:
7   *     Server → client: full frame N (691,200 B BGRX, raw HW layout).
8   *     Client → server: inputs N+2 (49 × i32 network-order; see
9   *                               client.pack_inputs for layout).
10  *
11  * Per frame: kick frame N+1 BEFORE sending N, so HW renders N+1 in
12  * parallel with the SW send of N. Total wall time max(HW, SW).
13  *
14  * On client disconnect, drain the in-flight frame before tearing
15  * down the mmap so DMA doesn't write to freed user memory.
16  *
17  * Usage: ./server [-v] [port]      (default port 12345; -v = per-frame log)
18  */
19
20 #include <stdio.h>
21 #include <stdlib.h>
22 #include <string.h>
23 #include <stdint.h>
24 #include <signal.h>
25 #include <errno.h>
26 #include <time.h>
27 #include <unistd.h>
28 #include <fcntl.h>
29 #include <sys/ioctl.h>
30 #include <sys/mman.h>
31 #include <sys/socket.h>
32 #include <netinet/in.h>
33 #include <netinet/tcp.h>
34 #include <arpa/inet.h>
35
36 #include "../raytracer_batch.h"
37
38 #define WIDTH      480
39 #define HEIGHT     360
40 #define FRAME_BYTES (WIDTH * HEIGHT * 4) /* 691,200 - BGRX wire size */
41 #define SLOT_WORDS  (WIDTH * HEIGHT)    /* uint32_t per slot */
42 #define SLOT_BYTES  (SLOT_WORDS * 4u)   /* 691,200 - DDR slot stride */
43
44 #define DEFAULT_PORT 12345
45
46 static volatile sig_atomic_t stop;
47 static int verbose = 0;

```

```

48 static void sigint_handler(int sig)
49 {
50     (void)sig;
51     stop = 1;
52 }
53
54 static int recv_exact(int fd, void *buf, size_t n)
55 {
56     size_t got = 0;
57     while (got < n) {
58         ssize_t r = recv(fd, (char *)buf + got, n - got, 0);
59         if (r <= 0)
60             return -1;
61         got += (size_t)r;
62     }
63     return 0;
64 }
65
66 static int send_all(int fd, const void *buf, size_t n)
67 {
68     size_t sent = 0;
69     while (sent < n) {
70         ssize_t w = send(fd, (const char *)buf + sent, n - sent, 0);
71         if (w < 0)
72             return -1;
73         sent += (size_t)w;
74     }
75     return 0;
76 }
77
78
79 /* Per-frame inputs received from the client. cam_y is locked at 1.5 in HW
80 * and right.y is hardcoded 0 (no roll), so neither has a field here.
81 * cone_norm_factor = sqrt(k2·(1+k2)) is pre-computed client-side so HW
82 * can recover the cone surface |normal| with one mul. Sun is derived from
83 * (lx, ly, lz) - no separate sun fields. */
84 struct frame_inputs {
85     int32_t lx, ly, lz;
86     int32_t sx, sy, sz;
87     uint32_t max_depth;
88     uint32_t shift_0;
89     uint32_t shift_1;
90     int32_t cam_x, cam_z;
91     int32_t right_x, right_z;
92     int32_t up_x, up_y, up_z;
93     int32_t fwd_x, fwd_y, fwd_z;
94     int32_t r0_sq, r1_sq;
95     int32_t sc0_x, sc0_y, sc0_z;
96     int32_t pc1_r, pc1_g, pc1_b;
97     int32_t pc2_r, pc2_g, pc2_b;
98     int32_t scol0_r, scol0_g, scol0_b;
99     int32_t scol1_r, scol1_g, scol1_b;
100     uint32_t floor_mode;
101     int32_t floor_lim_x, floor_lim_z;
102     int32_t cone_x, cone_y, cone_z;
103     int32_t cone_k_sq;
104     int32_t cone_height;
105     int32_t cone_col_r, cone_col_g, cone_col_b;
106     uint32_t reflect_shift_cone;
107     int32_t cone_norm_factor;
108 };
109
110 static inline void kick_frame(volatile struct rtb_csrs *csr,
111                             const struct frame_inputs *in)
112 {
113     csr->light_x      = (uint32_t)in->lx;
114     csr->light_y      = (uint32_t)in->ly;
115     csr->light_z      = (uint32_t)in->lz;
116     csr->sc1_x        = (uint32_t)in->sx;
117     csr->sc1_y        = (uint32_t)in->sy;
118     csr->sc1_z        = (uint32_t)in->sz;
119     csr->max_depth     = in->max_depth & 0x7u;
120     csr->reflect_shift_0 = in->shift_0 & 0x1fu;
121     csr->reflect_shift_1 = in->shift_1 & 0x1fu;

```

```

122     csr->cam_x      = (uint32_t)in->cam_x;
123     csr->cam_z      = (uint32_t)in->cam_z;
124     csr->right_x    = (uint32_t)in->right_x;
125     csr->right_z    = (uint32_t)in->right_z;
126     csr->up_x       = (uint32_t)in->up_x;
127     csr->up_y       = (uint32_t)in->up_y;
128     csr->up_z       = (uint32_t)in->up_z;
129     csr->fwd_x      = (uint32_t)in->fwd_x;
130     csr->fwd_y      = (uint32_t)in->fwd_y;
131     csr->fwd_z      = (uint32_t)in->fwd_z;
132     csr->r0_sq       = (uint32_t)in->r0_sq;
133     csr->r1_sq       = (uint32_t)in->r1_sq;
134     /* Visual-upgrades CSRs -(#0#5). Same atomic commit as the rest. */
135     csr->sc0_x       = (uint32_t)in->sc0_x;
136     csr->sc0_y       = (uint32_t)in->sc0_y;
137     csr->sc0_z       = (uint32_t)in->sc0_z;
138     csr->pc1_r       = (uint32_t)in->pc1_r;
139     csr->pc1_g       = (uint32_t)in->pc1_g;
140     csr->pc1_b       = (uint32_t)in->pc1_b;
141     csr->pc2_r       = (uint32_t)in->pc2_r;
142     csr->pc2_g       = (uint32_t)in->pc2_g;
143     csr->pc2_b       = (uint32_t)in->pc2_b;
144     csr->scol0_r      = (uint32_t)in->scol0_r;
145     csr->scol0_g      = (uint32_t)in->scol0_g;
146     csr->scol0_b      = (uint32_t)in->scol0_b;
147     csr->scol1_r      = (uint32_t)in->scol1_r;
148     csr->scol1_g      = (uint32_t)in->scol1_g;
149     csr->scol1_b      = (uint32_t)in->scol1_b;
150     csr->floor_mode   = in->floor_mode & 0x3u;
151     csr->floor_lim_x  = (uint32_t)in->floor_lim_x;
152     csr->floor_lim_z  = (uint32_t)in->floor_lim_z;
153     /* Cone primitive */
154     csr->cone_x       = (uint32_t)in->cone_x;
155     csr->cone_y       = (uint32_t)in->cone_y;
156     csr->cone_z       = (uint32_t)in->cone_z;
157     csr->cone_k_sq     = (uint32_t)in->cone_k_sq;
158     csr->cone_height  = (uint32_t)in->cone_height;
159     csr->cone_col_r    = (uint32_t)in->cone_col_r;
160     csr->cone_col_g    = (uint32_t)in->cone_col_g;
161     csr->cone_col_b    = (uint32_t)in->cone_col_b;
162     csr->reflect_shift_cone = in->reflect_shift_cone & 0x1fu;
163     csr->cone_norm_factor = (uint32_t)in->cone_norm_factor;
164     csr->control       = RTB_CONTROL_FRAME_START;
165 }
166
167 static int recv_frame_inputs(int fd, struct frame_inputs *in)
168 {
169     uint32_t hdr[49];
170     if (recv_exact(fd, hdr, sizeof(hdr)) < 0)
171         return -1;
172     in->lx      = (int32_t)ntohl(hdr[0]);
173     in->ly      = (int32_t)ntohl(hdr[1]);
174     in->lz      = (int32_t)ntohl(hdr[2]);
175     in->sx      = (int32_t)ntohl(hdr[3]);
176     in->sy      = (int32_t)ntohl(hdr[4]);
177     in->sz      = (int32_t)ntohl(hdr[5]);
178     in->max_depth = ntohl(hdr[6]);
179     in->shift_0  = ntohl(hdr[7]);
180     in->shift_1  = ntohl(hdr[8]);
181     in->cam_x    = (int32_t)ntohl(hdr[9]);
182     in->cam_z    = (int32_t)ntohl(hdr[10]);
183     in->right_x  = (int32_t)ntohl(hdr[11]);
184     in->right_z  = (int32_t)ntohl(hdr[12]);
185     in->up_x     = (int32_t)ntohl(hdr[13]);
186     in->up_y     = (int32_t)ntohl(hdr[14]);
187     in->up_z     = (int32_t)ntohl(hdr[15]);
188     in->fwd_x    = (int32_t)ntohl(hdr[16]);
189     in->fwd_y    = (int32_t)ntohl(hdr[17]);
190     in->fwd_z    = (int32_t)ntohl(hdr[18]);
191     in->r0_sq     = (int32_t)ntohl(hdr[19]);
192     in->r1_sq     = (int32_t)ntohl(hdr[20]);
193     in->sc0_x     = (int32_t)ntohl(hdr[21]);
194     in->sc0_y     = (int32_t)ntohl(hdr[22]);
195     in->sc0_z     = (int32_t)ntohl(hdr[23]);

```



```

196     in->pc1_r      = (int32_t)ntohl(hdr[24]);
197     in->pc1_g      = (int32_t)ntohl(hdr[25]);
198     in->pc1_b      = (int32_t)ntohl(hdr[26]);
199     in->pc2_r      = (int32_t)ntohl(hdr[27]);
200     in->pc2_g      = (int32_t)ntohl(hdr[28]);
201     in->pc2_b      = (int32_t)ntohl(hdr[29]);
202     in->scol0_r     = (int32_t)ntohl(hdr[30]);
203     in->scol0_g     = (int32_t)ntohl(hdr[31]);
204     in->scol0_b     = (int32_t)ntohl(hdr[32]);
205     in->scol1_r     = (int32_t)ntohl(hdr[33]);
206     in->scol1_g     = (int32_t)ntohl(hdr[34]);
207     in->scol1_b     = (int32_t)ntohl(hdr[35]);
208     in->floor_mode  = ntohl(hdr[36]);
209     in->floor_lim_x = (int32_t)ntohl(hdr[37]);
210     in->floor_lim_z = (int32_t)ntohl(hdr[38]);
211     in->cone_x      = (int32_t)ntohl(hdr[39]);
212     in->cone_y      = (int32_t)ntohl(hdr[40]);
213     in->cone_z      = (int32_t)ntohl(hdr[41]);
214     in->cone_k_sq   = (int32_t)ntohl(hdr[42]);
215     in->cone_height = (int32_t)ntohl(hdr[43]);
216     in->cone_col_r  = (int32_t)ntohl(hdr[44]);
217     in->cone_col_g  = (int32_t)ntohl(hdr[45]);
218     in->cone_col_b  = (int32_t)ntohl(hdr[46]);
219     in->reflect_shift_cone = ntohl(hdr[47]);
220     in->cone_norm_factor = (int32_t)ntohl(hdr[48]);
221     return 0;
222 }
223
224 /* Drain the in-flight HW frame so DMA doesn't write to the buffer after
225  * the userspace mmap is torn down. Called on client disconnect. */
226 static void drain_frame(volatile struct rtb_csrs *csr)
227 {
228     while (!(csr->status & RTB_STATUS_FRAME_DONE))
229         ;
230     csr->control = RTB_CONTROL_CLEAR_FRAME_DONE;
231 }
232
233 static void handle_client(int dev_fd,
234                          volatile struct rtb_csrs *csr,
235                          const uint32_t *frame,
236                          int cli_fd,
237                          struct sockaddr_in *addr)
238 {
239     char ip[INET_ADDRSTRLEN];
240     inet_ntop(AF_INET, &addr->sin_addr, ip, sizeof(ip));
241     printf("client connected: %s:%d\n", ip, ntohs(addr->sin_port));
242
243     struct frame_inputs in0;
244     struct frame_inputs in_pending;
245
246     if (recv_frame_inputs(cli_fd, &in0) < 0 ||
247         recv_frame_inputs(cli_fd, &in_pending) < 0) {
248         printf("client disconnected during prime\n");
249         close(cli_fd);
250         return;
251     }
252
253     kick_frame(csr, &in0);
254
255     unsigned frames = 0;
256
257     for (;;) {
258         struct timespec t0, t_wait, t_kick, t_send;
259         if (verbose) clock_gettime(CLOCK_MONOTONIC, &t0);
260
261         uint32_t done_slot;
262         if (ioctl(dev_fd, RTB_IOCTL_WAIT_FRAME, &done_slot) < 0) {
263             perror("ioctl(RTB_IOCTL_WAIT_FRAME)");
264             drain_frame(csr);
265             break;
266         }
267         if (verbose) clock_gettime(CLOCK_MONOTONIC, &t_wait);
268
269         /* Kick N+1 before sending N so HW renders in parallel with TCP. */

```

```

270     kick_frame(csr, &in_pending);
271     if (verbose) clock_gettime(CLOCK_MONOTONIC, &t_kick);
272
273     int send_failed = 0;
274     if (send_all(cli_fd, frame + done_slot * SLOT_WORDS,
275                 SLOT_BYTES) < 0) {
276         perror("send");
277         send_failed = 1;
278     }
279     if (verbose) clock_gettime(CLOCK_MONOTONIC, &t_send);
280
281     if (send_failed) {
282         drain_frame(csr);
283         break;
284     }
285
286     if (verbose) {
287         #define DELTA_MS(a, b) (((b).tv_sec - (a).tv_sec) * 1000.0 + \
288                               ((b).tv_nsec - (a).tv_nsec) / 1.0e6)
289         double total_ms = DELTA_MS(t0, t_send);
290         double wait_ms  = DELTA_MS(t0, t_wait);
291         double kick_ms  = DELTA_MS(t_wait, t_kick);
292         double send_ms  = DELTA_MS(t_kick, t_send);
293         #undef DELTA_MS
294
295         fprintf(stderr,
296                 "frame %u: %.2f ms (%.1f fps) - wait=%.2f kick=%.3f send=%.2f\n",
297                 frames, total_ms, 1000.0 / total_ms,
298                 wait_ms, kick_ms, send_ms);
299     }
300
301     frames++;
302
303     if (recv_frame_inputs(cli_fd, &in_pending) < 0) {
304         drain_frame(csr);
305         break;
306     }
307 }
308
309 printf("client disconnected (%u frames served)\n", frames);
310 close(cli_fd);
311 }
312
313 static void usage(const char *prog)
314 {
315     fprintf(stderr, "usage: %s [-v] [port]\n -v per-frame timing log\n", prog);
316 }
317
318 int main(int argc, char *argv[])
319 {
320     uint16_t port = DEFAULT_PORT;
321     int opt_c;
322     while ((opt_c = getopt(argc, argv, "vh")) != -1) {
323         switch (opt_c) {
324             case 'v': verbose = 1; break;
325             case 'h':
326                 usage(argv[0]); return opt_c == 'h' ? 0 : 1;
327         }
328     }
329     if (optind < argc)
330         port = (uint16_t)atoi(argv[optind]);
331
332     signal(SIGPIPE, SIG_IGN);
333     signal(SIGINT, sigint_handler);
334
335     int dev_fd = open("/dev/raytracer", O_RDWR);
336     if (dev_fd < 0) {
337         perror("open /dev/raytracer");
338         return 1;
339     }
340
341     struct rtb_layout layout;
342     if (ioctl(dev_fd, RTB_IOCTL_GET_LAYOUT, &layout) < 0) {
343         perror("ioctl(RTB_IOCTL_GET_LAYOUT)");

```

```

344     close(dev_fd);
345     return 1;
346 }
347
348 volatile struct rtb_csrs *csr = mmap(NULL, layout.csr_size,
349                                     PROT_READ | PROT_WRITE,
350                                     MAP_SHARED, dev_fd,
351                                     layout.csr_offset);
352
353 if (csr == MAP_FAILED) {
354     perror("mmap CSR");
355     close(dev_fd);
356     return 1;
357 }
358
359 /* Covers both slots; slot N starts at frame[N * SLOT_WORDS]. */
360 const uint32_t *frame = mmap(NULL, layout.frame_buf_bytes,
361                               PROT_READ, MAP_SHARED, dev_fd,
362                               layout.frame_buf_offset);
363
364 if (frame == MAP_FAILED) {
365     perror("mmap frame");
366     close(dev_fd);
367     return 1;
368 }
369
370 int listen_fd = socket(AF_INET, SOCK_STREAM, 0);
371 if (listen_fd < 0) {
372     perror("socket");
373     close(dev_fd);
374     return 1;
375 }
376
377 int opt = 1;
378 setsockopt(listen_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
379
380 struct sockaddr_in saddr;
381 memset(&saddr, 0, sizeof(saddr));
382 saddr.sin_family = AF_INET;
383 saddr.sin_addr.s_addr = htonl(INADDR_ANY);
384 saddr.sin_port = htons(port);
385
386 if (bind(listen_fd, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
387     perror("bind");
388     close(listen_fd);
389     close(dev_fd);
390     return 1;
391 }
392
393 if (listen(listen_fd, 1) < 0) {
394     perror("listen");
395     close(listen_fd);
396     close(dev_fd);
397     return 1;
398 }
399
400 printf("listening on :%d (%dx%d, %d bytes/frame, pipeline-depth-2)\n",
401        port, WIDTH, HEIGHT, FRAME_BYTES);
402
403 while (!stop) {
404     struct sockaddr_in caddr;
405     socklen_t clen = sizeof(caddr);
406     int cli_fd = accept(listen_fd, (struct sockaddr *)&caddr, &clen);
407     if (cli_fd < 0) {
408         if (errno == EINTR)
409             break;
410         perror("accept");
411         continue;
412     }
413
414     opt = 1;
415     setsockopt(cli_fd, IPPROTO_TCP, TCP_NODELAY, &opt, sizeof(opt));
416
417     /* SO_SNDBUF frame so send() returns once the kernel has copied
418      * from the cached mmap, allowing wire-out to overlap HW render. */
419     int sndbuf = 1 << 20;

```

```

418         setsockopt(cli_fd, SOL_SOCKET, SO_SNDBUF, &sndbuf, sizeof(sndbuf));
419
420         handle_client(dev_fd, csr, frame, cli_fd, &caddr);
421     }
422
423     printf("\nshutting down\n");
424     close(listen_fd);
425     munmap((void *)frame, layout.frame_buf_bytes);
426     munmap((void *)csr, layout.csr_size);
427     close(dev_fd);
428     return 0;
429 }

```

SW/fpga/render_frame.c

```

1  /*
2  * render_frame.c - one-shot tool: render one full 480×360 frame via the
3  * HW per-frame auto-kick (CONTROL[2] = FRAME_START), then write it as a
4  * raw PPM image to stdout.
5  *
6  * Usage:
7  * ./render_frame LX LY LZ [MAX_DEPTH SHIFT0 SHIFT1] > frame.ppm
8  *
9  * Light position (3 floats, e.g. 0.0 4.0 4.0) is required. Reflection
10 * params are optional; when omitted the HW boots into its default
11 * scene (max_depth=3, shift0=2, shift1=1 → blue sphere reflects 25%,
12 * red sphere 50%, 3 bounces deep). Pass `MAX_DEPTH=0` to disable
13 * reflection entirely.
14 *
15 * Output is a P6 PPM file (viewable with feh, eog, GIMP, etc.).
16 *
17 * The frame buffer is a 2-slot ring (1.4 MB total). RTB_IOCTL_WAIT_FRAME
18 * blocks until FRAME_DONE, invalidates the just-finished slot's caches,
19 * and returns its index - we read directly from the cached mmap. No
20 * pipelining (single frame, doesn't pay).
21 */
22
23 #include <stdio.h>
24 #include <stdlib.h>
25 #include <string.h>
26 #include <stdint.h>
27 #include <fcntl.h>
28 #include <unistd.h>
29 #include <sys/ioctl.h>
30 #include <sys/mman.h>
31
32 #include "../raytracer_batch.h"
33
34 #define WIDTH      480
35 #define HEIGHT     360
36 #define FP_SCALE 8192 /* 1 << FRAC_BITS at Q14.13 (must match raytracer.sv FRAC_BITS) */
37
38 /* HW reset defaults (must match raytracer.sv DEFAULT_MAX_DEPTH /
39 * DEFAULT_REFLECT_SHIFT_*). */
40 #define DEFAULT_MAX_DEPTH      3u
41 #define DEFAULT_REFLECT_SHIFT_0 2u
42 #define DEFAULT_REFLECT_SHIFT_1 1u
43
44 int main(int argc, char *argv[])
45 {
46     if (argc != 4 && argc != 7) {
47         fprintf(stderr,
48             "usage: %s LX LY LZ [MAX_DEPTH SHIFT0 SHIFT1]\n"
49             "      light position required (3 floats);\n"
50             "      reflection params optional, defaulting to\n"
51             "      depth=%u shift0=%u shift1=%u (HW reset values).\n",
52             argv[0], DEFAULT_MAX_DEPTH,
53             DEFAULT_REFLECT_SHIFT_0, DEFAULT_REFLECT_SHIFT_1);
54         return 1;
55     }
56
57     int32_t lx = (int32_t)(atof(argv[1]) * FP_SCALE);

```

```

58 int32_t ly = (int32_t)(atof(argv[2]) * FP_SCALE);
59 int32_t lz = (int32_t)(atof(argv[3]) * FP_SCALE);
60
61 uint32_t max_depth = DEFAULT_MAX_DEPTH;
62 uint32_t shift_0 = DEFAULT_REFLECT_SHIFT_0;
63 uint32_t shift_1 = DEFAULT_REFLECT_SHIFT_1;
64 if (argc == 7) {
65     max_depth = (uint32_t)strtoul(argv[4], NULL, 0);
66     shift_0 = (uint32_t)strtoul(argv[5], NULL, 0);
67     shift_1 = (uint32_t)strtoul(argv[6], NULL, 0);
68 }
69
70 int fd = open("/dev/raytracer", O_RDWR);
71 if (fd < 0) {
72     perror("open /dev/raytracer");
73     return 1;
74 }
75
76 struct rtb_layout layout;
77 if (ioctl(fd, RTB_IOCTL_GET_LAYOUT, &layout) < 0) {
78     perror("ioctl(RTB_IOCTL_GET_LAYOUT)");
79     close(fd);
80     return 1;
81 }
82
83 volatile struct rtb_csrs *csr = mmap(NULL, layout.csr_size,
84                                     PROT_READ | PROT_WRITE,
85                                     MAP_SHARED, fd,
86                                     layout.csr_offset);
87
88 if (csr == MAP_FAILED) {
89     perror("mmap CSR");
90     close(fd);
91     return 1;
92 }
93
94 const uint32_t *frame = mmap(NULL, layout.frame_buf_bytes,
95                             PROT_READ, MAP_SHARED, fd,
96                             layout.frame_buf_offset);
97
98 if (frame == MAP_FAILED) {
99     perror("mmap frame");
100     close(fd);
101     return 1;
102 }
103
104 /* Stage light + reflection params + kick. sc1_* is left untouched
105  * here (keeps its current value, which is the HW default at first
106  * kick). HW picks the write_slot internally; we get the just-finished
107  * slot back from RTB_IOCTL_WAIT_FRAME. */
108 csr->light_x = (uint32_t)lx;
109 csr->light_y = (uint32_t)ly;
110 csr->light_z = (uint32_t)lz;
111 csr->max_depth = max_depth & 0x7u;
112 csr->reflect_shift_0 = shift_0 & 0x1fu;
113 csr->reflect_shift_1 = shift_1 & 0x1fu;
114 csr->control = RTB_CONTROL_FRAME_START;
115
116 uint32_t slot;
117 if (ioctl(fd, RTB_IOCTL_WAIT_FRAME, &slot) < 0) {
118     perror("ioctl(RTB_IOCTL_WAIT_FRAME)");
119     close(fd);
120     return 1;
121 }
122
123 /* Unpack →BGRXRGB directly from the cached mmap. The WAIT_FRAME
124  * ioctl already invalidated this slot's cache lines, so the loop
125  * fills L1/L2 from DDR in one streaming pass. */
126 const uint32_t *slot_src = frame + slot * (WIDTH * HEIGHT);
127
128 /* PPM header */
129 printf("P6\n%d %d\n255\n", WIDTH, HEIGHT);
130
131 uint8_t row[WIDTH * 3];
132 for (unsigned y = 0; y < HEIGHT; y++) {
133     const uint32_t *row_src = slot_src + y * WIDTH;

```

```

132     uint8_t *p = row;
133     for (unsigned x = 0; x < WIDTH; x++) {
134         uint32_t pk = row_src[x];
135         *p++ = (uint8_t)((pk >> 16) & 0xff);
136         *p++ = (uint8_t)((pk >> 8) & 0xff);
137         *p++ = (uint8_t)(pk & 0xff);
138     }
139     fwrite(row, 1, sizeof(row), stdout);
140 }
141
142 munmap((void *)frame, layout.frame_buf_bytes);
143 munmap((void *)csr, layout.csr_size);
144 close(fd);
145
146 fprintf(stderr,
147     "rendered %dx%d frame (light %.1f, %.1f, %.1f, "
148     "max_depth=%u shift0=%u shift1=%u)\n",
149     WIDTH, HEIGHT,
150     lx / (double)FP_SCALE,
151     ly / (double)FP_SCALE,
152     lz / (double)FP_SCALE,
153     max_depth, shift_0, shift_1);
154 return 0;
155 }

```

SW/fpga/Makefile

```

1 CROSS_COMPILE ?=
2 CC := $(CROSS_COMPILE)gcc
3 CFLAGS := -O2 -Wall -Wextra -I..
4 LDFLAGS :=
5
6 all: server render_frame
7
8 server: server.c ../raytracer_batch.h
9     $(CC) $(CFLAGS) -o $@ server.c $(LDFLAGS)
10
11 render_frame: render_frame.c ../raytracer_batch.h
12     $(CC) $(CFLAGS) -o $@ render_frame.c $(LDFLAGS)
13
14 clean:
15     rm -f server render_frame
16
17 .PHONY: clean

```

3 Desktop Client + Demos

SW/desktop/client.py

```

1 #!/usr/bin/env python3
2 """Desktop client for the FPGA raytracer.
3
4 Single frame:
5     python client.py HOST PORT --light LX LY LZ -o out.png
6
7 Animation (orbiting light, saved to GIF):
8     python client.py HOST PORT --animate N -o out.gif [--radius 4.0] [--height 4.0] [--duration 80]
9
10 Live stream (orbiting light, displayed in a window):
11     python client.py HOST PORT --stream [--radius 4.0] [--height 4.0] [--period 3.0]
12 """
13
14 import argparse
15 import math
16 import socket
17 import struct
18 import time
19
20 from PIL import Image

```

```

21 # Frame geometry - must match the FPGA hardware / driver constants.
22 WIDTH = 480
23 HEIGHT = 360
24 FP_SHIFT = 13
25 FP_SCALE = 1 << FP_SHIFT
26 # Wire format is the raw HW DDR layout: 4 bytes per pixel as BGRX.
27 # FPGA writes {00,R,G,B} per word; little-endian ARM HPS stores it as
28 # B,G,R,00 in memory and ships those bytes straight to the desktop,
29 # which decodes →BGRXRGB with PIL (cheaper than unpacking on the
30 # Cortex-A9).
31 FRAME_BYTES = WIDTH * HEIGHT * 4 # 691,200
32
33
34 # HW defaults - must mirror raytracer.sv's DEFAULT_* localparams so a
35 # client that doesn't override anything sends the byte-equivalent scene.
36 SPHERE0_CENTER = (0.0, 1.0, 0.0)
37 SPHERE1_DEFAULT_CENTER = (2.5, 1.0, 0.0)
38
39 DEFAULT_MAX_DEPTH = 3 # 0..7; 0 disables reflection
40 DEFAULT_REFLECT_SHIFT_0 = 2 # blue sphere reflection scale = >> 2 (25%)
41 DEFAULT_REFLECT_SHIFT_1 = 1 # red sphere = >> 1 (50%)
42
43 # SW pre-squares radii before shipping; the CSR holds  $r^2$  directly so HW
44 # avoids a multiply per pixel.
45 DEFAULT_RADIUS_0 = 1.0
46 DEFAULT_RADIUS_1 = 1.0
47
48 # Camera defaults. cam_y is locked at 1.5 in HW (no Y-translation), so
49 # only X / Z are CSR-driven. yaw=0, pitch=0 produces the identity basis
50 # (right=x, up=y, fwd=z) - the legacy hardcoded scene byte-equal.
51 DEFAULT_CAM_X = 0.0
52 DEFAULT_CAM_Z = -8.0
53 DEFAULT_YAW = 0.0
54 DEFAULT_PITCH = 0.0
55
56 DEFAULT_SC0 = (0.0, 1.0, 0.0)
57 DEFAULT_PC1_RGB = (0.9, 0.9, 0.9)
58 DEFAULT_PC2_RGB = (0.4, 0.4, 0.4)
59 DEFAULT_SCOLO_RGB = (0.2, 0.3, 0.8)
60 DEFAULT_SCOL1_RGB = (0.8, 0.3, 0.2)
61 DEFAULT_FLOOR_MODE = 0
62 DEFAULT_FLOOR_LIM = 1000.0 # ~infinite half-extent
63
64 # Y-axis cone, apex on top, opens downward over y [apex.y - height,
65 # apex.y]. height=0 disables the cone - matches the HW reset state.
66 DEFAULT_CONE_APEX = (0.0, 2.5, 0.0)
67 DEFAULT_CONE_K_SQ = 1.0 #  $\tan^2(\text{half-angle})$ ; 1.0 = 45°
68 DEFAULT_CONE_HEIGHT = 0.0
69 DEFAULT_CONE_COL_RGB = (0.5, 0.5, 0.5)
70 DEFAULT_REFLECT_SHIFT_CONE = 0
71
72
73 def to_fp(f: float) -> int:
74     """Convert a float to Q14.13 signed fixed-point (int32)."""
75     return int(round(f * FP_SCALE))
76
77
78 def camera_basis_from_euler(yaw: float, pitch: float
79                             ) -> tuple[tuple[float, float],
80                                         tuple[float, float, float],
81                                         tuple[float, float, float]]:
82     """Compute the right / up / fwd basis vectors from yaw + pitch
83     (radians). right.y is always 0 (we don't support roll), so right
84     is returned as a 2-tuple (x, z); up and fwd are full 3-tuples.
85
86     Convention (matches raytracer_fp.py:camera_basis):
87     yaw=0, pitch=0 → right=(-1, 0), up=(0, 1, 0), fwd=(0, 0, 1)
88     yaw>0 → camera turns right (forward picks up +x)
89     pitch>0 → camera tilts up (forward picks up +y)
90     """
91     cy, sy = math.cos(yaw), math.sin(yaw)
92     cp, sp = math.cos(pitch), math.sin(pitch)
93     right = (-cy, sy)
94     up = (-sy * sp, cp, -cy * sp)

```

```

95     forward = ( sy * cp,  sp,          cy * cp)
96     return right, up, forward
97
98
99 def pack_inputs(lx: float, ly: float, lz: float,
100               sx: float, sy: float, sz: float,
101               max_depth: int = DEFAULT_MAX_DEPTH,
102               shift_0: int = DEFAULT_REFLECT_SHIFT_0,
103               shift_1: int = DEFAULT_REFLECT_SHIFT_1,
104               cam_x: float = DEFAULT_CAM_X,
105               cam_z: float = DEFAULT_CAM_Z,
106               yaw: float = DEFAULT_YAW,
107               pitch: float = DEFAULT_PITCH,
108               r0: float = DEFAULT_RADIUS_0,
109               r1: float = DEFAULT_RADIUS_1,
110               sc0: tuple[float, float, float] = DEFAULT_SCO,
111               pc1_rgb: tuple[float, float, float] = DEFAULT_PC1_RGB,
112               pc2_rgb: tuple[float, float, float] = DEFAULT_PC2_RGB,
113               sco10_rgb: tuple[float, float, float] = DEFAULT_SCOLO_RGB,
114               sco11_rgb: tuple[float, float, float] = DEFAULT_SCOL1_RGB,
115               floor_mode: int = DEFAULT_FLOOR_MODE,
116               floor_lim_x: float = DEFAULT_FLOOR_LIM,
117               floor_lim_z: float = DEFAULT_FLOOR_LIM,
118               cone_apex: tuple[float, float, float] = DEFAULT_CONE_APEX,
119               cone_k_sq: float = DEFAULT_CONE_K_SQ,
120               cone_height: float = DEFAULT_CONE_HEIGHT,
121               cone_col_rgb: tuple[float, float, float] = DEFAULT_CONE_COL_RGB,
122               reflect_shift_cone: int = DEFAULT_REFLECT_SHIFT_CONE,
123               ) -> bytes:
124     """Pack a per-frame inputs blob: 49 × int32 (network-order, 196 bytes).
125     Layout matches server.c's recv_frame_inputs():
126     [0..2]   light XYZ                (Q14.13 signed)
127     [3..5]   sphere-1 center XYZ
128     [6]      max_depth                (0..7; 0 disables reflection)
129     [7]      reflect_shift_0          (0..31)
130     [8]      reflect_shift_1          (0..31)
131     [9..10]  camera origin X / Z      (cam_y locked in HW)
132     [11..12] basis right.x / right.z  (right.y = 0)
133     [13..15] basis up.x / .y / .z
134     [16..18] basis fwd.x / .y / .z
135     [19..20] sphere radii2 (r02, r12) - SW pre-squares r
136     [21..23] sphere-0 center XYZ
137     [24..26] floor color #1 RGB
138     [27..29] floor color #2 RGB
139     [30..32] sphere-0 RGB
140     [33..35] sphere-1 RGB
141     [36]     floor_mode                (0..3)
142     [37..38] floor_lim X / Z
143     [39..41] cone apex XYZ
144     [42]     cone_k_sq                 (tan2(half-angle))
145     [43]     cone_height               (0 disables)
146     [44..46] cone color RGB
147     [47]     reflect_shift_cone        (0..31)
148     [48]     cone_norm_factor          (SW-computed  $\sqrt{k^2 \cdot (1+k^2)}$ ) so HW
149                                     recovers |normal| with one mul)
150     HW masks integer fields to their register widths on commit, so
151     out-of-range values are silently clamped."""
152     right, up, forward = camera_basis_from_euler(yaw, pitch)
153     cone_norm_factor = math.sqrt(cone_k_sq * (1.0 + cone_k_sq))
154     return struct.pack(
155         "!iiiiiiIIiiiiiiiiiiii"      # legacy 21
156         "iiiiiiiiiiiiiiiiIii"        # extension 18 (1 unsigned: floor_mode)
157         "iiiiiiiIii",                 # cone      10 (1 unsigned: reflect_shift_cone)
158         to_fp(lx), to_fp(ly), to_fp(lz),
159         to_fp(sx), to_fp(sy), to_fp(sz),
160         max_depth & 0x7,
161         shift_0 & 0x1f,
162         shift_1 & 0x1f,
163         to_fp(cam_x), to_fp(cam_z),
164         to_fp(right[0]), to_fp(right[1]),
165         to_fp(up[0]), to_fp(up[1]), to_fp(up[2]),
166         to_fp(forward[0]), to_fp(forward[1]), to_fp(forward[2]),
167         to_fp(r0 * r0), to_fp(r1 * r1),
168         to_fp(sc0[0]), to_fp(sc0[1]), to_fp(sc0[2]),

```



```

169         to_fp(pc1_rgb[0]), to_fp(pc1_rgb[1]), to_fp(pc1_rgb[2]),
170         to_fp(pc2_rgb[0]), to_fp(pc2_rgb[1]), to_fp(pc2_rgb[2]),
171         to_fp(scol0_rgb[0]), to_fp(scol0_rgb[1]), to_fp(scol0_rgb[2]),
172         to_fp(scol1_rgb[0]), to_fp(scol1_rgb[1]), to_fp(scol1_rgb[2]),
173         floor_mode & 0x3,
174         to_fp(floor_lim_x), to_fp(floor_lim_z),
175         to_fp(cone_apex[0]), to_fp(cone_apex[1]), to_fp(cone_apex[2]),
176         to_fp(cone_k_sq), to_fp(cone_height),
177         to_fp(cone_col_rgb[0]), to_fp(cone_col_rgb[1]), to_fp(cone_col_rgb[2]),
178         reflect_shift_cone & 0x1f,
179         to_fp(cone_norm_factor),
180     )
181
182
183 def recv_frame(sock: socket.socket) -> bytes:
184     """Read a full frame of RGB888 data off the socket."""
185     buf = bytearray(FRAME_BYTES)
186     view = memoryview(buf)
187     got = 0
188     while got < FRAME_BYTES:
189         n = sock.recv_into(view[got:])
190         if n == 0:
191             raise ConnectionError(
192                 f"server closed after {got}/{FRAME_BYTES} bytes"
193             )
194         got += n
195     return bytes(buf)
196
197
198 # Wire protocol - pipeline-depth-2.
199 # The server kicks frame N+1 to HW before reading frame N's slot, so HW
200 # renders in parallel with SW reading + sending. The client must therefore
201 # send 2 inputs blobs up front ("prime") and then 1 blob per received
202 # frame. For one-shot single-frame rendering: prime with 2 copies of the
203 # same blob, receive 1 frame, and close - the server drains the extra
204 # in-flight frame on disconnect.
205
206
207 def request_frame(sock: socket.socket,
208                  lx: float, ly: float, lz: float,
209                  sphere: tuple[float, float, float] = SPHERE1_DEFAULT_CENTER,
210                  max_depth: int = DEFAULT_MAX_DEPTH,
211                  shift_0: int = DEFAULT_REFLECT_SHIFT_0,
212                  shift_1: int = DEFAULT_REFLECT_SHIFT_1) -> bytes:
213     """One-shot: prime with 2 identical inputs blobs, recv 1 frame. The
214     extra in-flight HW frame is drained on close. For continuous use see
215     render_animation / render_stream, which keep the pipe full."""
216     payload = pack_inputs(lx, ly, lz, *sphere,
217                          max_depth=max_depth,
218                          shift_0=shift_0, shift_1=shift_1)
219     sock.sendall(payload)
220     sock.sendall(payload)
221     return recv_frame(sock)
222
223
224 def decode(data: bytes) -> Image.Image:
225     """Decode raw BGRX (4 B/pixel) bytes into an RGB PIL Image - PIL's
226     raw decoder does the →BGRXRGB swap in one C-level pass."""
227     return Image.frombuffer("RGB", (WIDTH, HEIGHT), data, "raw", "BGRX", 0, 1)
228
229
230 def render_single(sock, lx, ly, lz, output,
231                  max_depth=DEFAULT_MAX_DEPTH,
232                  shift_0=DEFAULT_REFLECT_SHIFT_0,
233                  shift_1=DEFAULT_REFLECT_SHIFT_1):
234     """Render one frame and save it."""
235     data = request_frame(sock, lx, ly, lz,
236                        max_depth=max_depth,
237                        shift_0=shift_0, shift_1=shift_1)
238     img = decode(data)
239     img.save(output)
240     print(f"saved {output} ({WIDTH}x{HEIGHT}), "
241           f"max_depth={max_depth} shifts=({shift_0},{shift_1})")
242

```

```

243
244 class LightController:
245     """Interface for driving the light position over time."""
246
247     def light_at(self, t: float) -> tuple[float, float, float]:
248         raise NotImplementedError
249
250
251 class AutoOrbitController(LightController):
252     """Orbits the light in a circle at constant height."""
253
254     def __init__(self, radius: float = 4.0, height: float = 4.0, period: float = 3.0):
255         self.radius = radius
256         self.height = height
257         self.period = period
258
259     def light_at(self, t: float) -> tuple[float, float, float]:
260         angle = 2.0 * math.pi * (t / self.period)
261         return (
262             self.radius * math.cos(angle),
263             self.height,
264             self.radius * math.sin(angle),
265         )
266
267
268 class SphereController:
269     """Interface for driving sphere 1 (red) center position over time."""
270
271     def sphere_at(self, t: float) -> tuple[float, float, float]:
272         raise NotImplementedError
273
274
275 class StaticSphereController(SphereController):
276     """Holds sphere 1 at a fixed position (matches the HW DEFAULT_SC1_*)."""
277
278     def __init__(self, position: tuple[float, float, float] = SPHERE1_DEFAULT_CENTER):
279         self.position = position
280
281     def sphere_at(self, t: float) -> tuple[float, float, float]:
282         return self.position
283
284
285 class OrbitSphereController(SphereController):
286     """Orbits sphere 1 (red) horizontally around a fixed center (default:
287     sphere 0's position) at constant height."""
288
289     def __init__(self,
290         center: tuple[float, float, float] = SPHERE0_CENTER,
291         radius: float = 2.5,
292         height: float = 1.0,
293         period: float = 40.0):
294         self.center = center
295         self.radius = radius
296         self.height = height
297         self.period = period
298
299     def sphere_at(self, t: float) -> tuple[float, float, float]:
300         angle = 2.0 * math.pi * (t / self.period)
301         cx, _, cz = self.center
302         return (
303             cx + self.radius * math.cos(angle),
304             self.height,
305             cz + self.radius * math.sin(angle),
306         )
307
308
309 def render_stream(sock: socket.socket,
310     light_controller: LightController,
311     sphere_controller: SphereController,
312     max_depth: int = DEFAULT_MAX_DEPTH,
313     shift_0: int = DEFAULT_REFLECT_SHIFT_0,
314     shift_1: int = DEFAULT_REFLECT_SHIFT_1) -> None:
315     """Continuously request frames and blit them to a pygame window.
316

```

```

317 Uses the Stage 4 pipeline-depth-2 protocol: prime with inputs 0 and 1,
318 then send inputs N+2 after receiving frame N. Light + sphere positions
319 are sampled at fixed wall-clock intervals so both orbits advance
320 smoothly even when the pipeline is N=2 frames ahead of the displayed
321 frame. Reflection params are constant across the stream.""
322 try:
323     import pygame
324 except ImportError:
325     raise SystemExit(
326         "pygame is required for --stream mode. Install it with: pip install pygame"
327     )
328
329 pygame.init()
330 screen = pygame.display.set_mode((WIDTH, HEIGHT), pygame.FULLSCREEN | pygame.SCALED)
331 pygame.display.set_caption("FPGA Raytracer (live)")
332 clock = pygame.time.Clock()
333
334 pygame.joystick.init()
335 if pygame.joystick.get_count() > 0:
336     joy = pygame.joystick.Joystick(0)
337     joy.init()
338     print(f"Connected: {joy.get_name()}")
339 else:
340     print("ERROR: No joysticks found!")
341     return
342
343 t0 = time.monotonic()
344
345 # Light position is driven by joystick axes 0/1 (X/Z). The
346 # light_controller arg is kept in the signature for API compatibility
347 # with the main()->render_stream path but the joystick takes priority.
348 light_pos = [0.0, 4.0, 0.0]
349 LIGHT_MOVE_SCALE = 5.0
350
351 def user_inputs() -> bytes:
352     t = time.monotonic() - t0
353     return pack_inputs(
354         *light_pos,
355         *sphere_controller.sphere_at(t),
356         max_depth=max_depth,
357         shift_0=shift_0, shift_1=shift_1)
358
359 def update_light_pos():
360     light_pos[0] = joy.get_axis(0) * LIGHT_MOVE_SCALE
361     light_pos[2] = -joy.get_axis(1) * LIGHT_MOVE_SCALE
362
363 sock.sendall(user_inputs())
364 sock.sendall(user_inputs())
365
366 frames = 0
367 running = True
368 try:
369     while running:
370         for ev in pygame.event.get():
371             if ev.type == pygame.QUIT:
372                 running = False
373                 break
374             if ev.type == pygame.KEYDOWN and ev.key == pygame.K_ESCAPE:
375                 running = False
376                 break
377         if not running:
378             break
379
380         data = recv_frame(sock)
381         update_light_pos()
382         # Pipeline next inputs immediately so they land on the wire
383         # while the server is still sending the rest of this frame.
384         sock.sendall(user_inputs())
385
386         img = decode(data)
387         surf = pygame.image.fromstring(img.tobytes(), img.size, img.mode)
388         screen.blit(surf, (0, 0))
389         pygame.display.flip()
390

```

```

391         frames += 1
392         clock.tick()
393         if frames % 30 == 0:
394             print(f"{frames} frames  ({clock.get_fps():.1f} fps)")
395     finally:
396         pygame.quit()
397         print(f"streamed {frames} frames")
398
399
400 def render_animation(sock, num_frames, output, radius, height, duration,
401                     sphere_controller: SphereController | None = None,
402                     max_depth: int = DEFAULT_MAX_DEPTH,
403                     shift_0: int = DEFAULT_REFLECT_SHIFT_0,
404                     shift_1: int = DEFAULT_REFLECT_SHIFT_1):
405     """Render an orbiting-light animation and save as GIF. Optional
406     sphere_controller orbits sphere 1; defaults to a static position.
407     Same pipeline-depth-2 wire protocol as render_stream."""
408     if sphere_controller is None:
409         sphere_controller = StaticSphereController()
410
411     def light_for(i):
412         angle = 2.0 * math.pi * i / num_frames
413         return (radius * math.cos(angle), height, radius * math.sin(angle))
414
415     # Sample sphere at the same per-frame phase as light. Picking the
416     # period from duration*num_frames makes a full sphere orbit span
417     # the whole GIF.
418     def inputs_for(i):
419         return pack_inputs(
420             *light_for(i),
421             *sphere_controller.sphere_at(i * duration / 1000.0),
422             max_depth=max_depth,
423             shift_0=shift_0, shift_1=shift_1,
424         )
425
426     sock.sendall(inputs_for(0))
427     if num_frames >= 2:
428         sock.sendall(inputs_for(1))
429     else:
430         sock.sendall(inputs_for(0))    # 1-frame anim: pad the prime
431
432     # Server's loop ships frame K then blocks on inputs K+2 before iter
433     # K+1. To ship the final real frame we send N+1 blobs total: the 2
434     # primes, N-2 in-loop sends, and one phantom tail (duplicate of
435     # inputs N-1) so iter N-2's tail recv unblocks. The server then
436     # kicks a phantom frame N which is drained on close.
437     frames = []
438     for i in range(num_frames):
439         data = recv_frame(sock)
440         next_idx = i + 2
441         if next_idx < num_frames:
442             sock.sendall(inputs_for(next_idx))
443         elif next_idx == num_frames:
444             sock.sendall(inputs_for(num_frames - 1))    # phantom tail
445         frames.append(decode(data))
446         print(f"frame {i + 1}/{num_frames}")
447
448     frames[0].save(
449         output,
450         save_all=True,
451         append_images=frames[1:],
452         duration=duration,
453         loop=0,
454     )
455     print(f"saved {output} ({num_frames} frames)")
456
457
458 def main():
459     parser = argparse.ArgumentParser(description="FPGA raytracer client")
460     parser.add_argument("host", help="FPGA server IP or hostname")
461     parser.add_argument("port", type=int, help="FPGA server port")
462     parser.add_argument("-o", "--output", default="out.png", help="output file")
463
464     group = parser.add_mutually_exclusive_group(required=True)

```

```

465 group.add_argument(
466     "--light",
467     nargs=3,
468     type=float,
469     metavar=("LX", "LY", "LZ"),
470     help="light position (floats); saves a single PNG",
471 )
472 group.add_argument(
473     "--animate",
474     type=int,
475     metavar="N",
476     help="render N frames with orbiting light; saves a GIF",
477 )
478 group.add_argument(
479     "--stream",
480     action="store_true",
481     help="continuously display orbiting-light frames in a pygame window",
482 )
483
484 parser.add_argument("--radius", type=float, default=4.0, help="orbit radius")
485 parser.add_argument("--height", type=float, default=4.0, help="light height")
486 parser.add_argument(
487     "--duration", type=int, default=80, help="GIF frame duration (ms)"
488 )
489 parser.add_argument(
490     "--period", type=float, default=3.0,
491     help="stream mode: seconds per full light orbit",
492 )
493
494 parser.add_argument(
495     "--max-depth", type=int, default=DEFAULT_MAX_DEPTH,
496     help=f"reflection recursion cap (0..7; default {DEFAULT_MAX_DEPTH}; 0 disables)",
497 )
498 parser.add_argument(
499     "--shift0", type=int, default=DEFAULT_REFLECT_SHIFT_0,
500     help=f"sphere 0 reflectivity = >> shift (0..31; default {DEFAULT_REFLECT_SHIFT_0})",
501 )
502 parser.add_argument(
503     "--shift1", type=int, default=DEFAULT_REFLECT_SHIFT_1,
504     help=f"sphere 1 reflectivity = >> shift (0..31; default {DEFAULT_REFLECT_SHIFT_1})",
505 )
506
507 args = parser.parse_args()
508
509 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
510 sock.connect((args.host, args.port))
511
512 try:
513     if args.light is not None:
514         render_single(sock, *args.light, args.output,
515                       max_depth=args.max_depth,
516                       shift_0=args.shift0, shift_1=args.shift1)
517     elif args.stream:
518         light_controller = AutoOrbitController(
519             radius=args.radius, height=args.height, period=args.period,
520         )
521         sphere_controller = OrbitSphereController()
522         render_stream(sock, light_controller, sphere_controller,
523                      max_depth=args.max_depth,
524                      shift_0=args.shift0, shift_1=args.shift1)
525     else:
526         render_animation(
527             sock, args.animate, args.output,
528             args.radius, args.height, args.duration,
529             sphere_controller=OrbitSphereController(),
530             max_depth=args.max_depth,
531             shift_0=args.shift0, shift_1=args.shift1,
532         )
533 finally:
534     sock.close()
535
536
537 if __name__ == "__main__":
538     main()

```

SW/desktop/camera.py

```

1  #!/usr/bin/env python3
2  """Desktop client for the FPGA raytracer with joystick CAMERA control.
3
4  Like client.py's --stream mode but the joystick drives the camera
5  instead of the light:
6      Axis 0 → strafe (camera-relative ±right)
7      Axis 1 → walk   (camera-relative ±forward)
8      Axis 2 → yaw    (clockwise from above)      - if 4-axis device
9      Axis 3 → pitch  (clamped ±80° to dodge basis singularity)
10
11  Axes 2/3 use a wide cliff (default 30%% of full range) anchored at the
12  absolute axis zero, NOT a runtime snapshot. On a flight stick (where
13  axis 3 is a throttle slider that doesn't self-center) this means: park
14  the slider near its mechanical center to disable pitch; leaving it
15  parked off-center will integrate pitch continuously. Move the throttle
16  slider into the cliff to stop pitching.
17
18  The camera moves in its own XZ frame so "forward" always means whichever
19  direction the camera is looking at, not world +Z. cam_y is locked at 1.5
20  in HW (no Y-translation), matching the camera plan.
21
22  Light stays at a static position (--light, default (0, 4, 0)) so camera
23  motion is the only thing changing in the scene - easier to tell whether
24  the basis math is firing correctly.
25
26  Usage:
27      python camera.py HOST PORT
28      python camera.py HOST PORT --start-x 2.0 --start-yaw 30
29      python camera.py HOST PORT --light 4 4 4 --max-depth 0 # no reflection
30  """
31
32  import argparse
33  import math
34  import socket
35  import struct
36  import time
37
38  from PIL import Image
39
40  import client
41
42  # Frame geometry - must match the FPGA hardware / driver constants.
43  WIDTH = 480
44  HEIGHT = 360
45  FP_SHIFT = 13
46  FP_SCALE = 1 << FP_SHIFT
47  FRAME_BYTES = WIDTH * HEIGHT * 4
48
49  SPHEREO_CENTER = (0.0, 1.0, 0.0)
50  SPHERE1_DEFAULT_CENTER = (2.5, 1.0, 0.0)
51
52  DEFAULT_MAX_DEPTH = 3
53  DEFAULT_REFLECT_SHIFT_0 = 2
54  DEFAULT_REFLECT_SHIFT_1 = 1
55
56  DEFAULT_CAM_X = 0.0
57  DEFAULT_CAM_Z = -8.0
58  DEFAULT_YAW = 0.0
59  DEFAULT_PITCH = 0.0
60
61  # Pitch limit: at ±/2 the basis collapses (fwd → ŷ±, up degenerates).
62  # 80° is well clear of the singularity and still gives near-vertical look.
63  PITCH_LIMIT = math.radians(80.0)
64
65
66  def camera_basis_from_euler(yaw: float, pitch: float):
67      """Same convention as client.py / raytracer_fp.camera_basis. yaw=0,
68      pitch=0 yields right=(-1,0), up=(0,1,0), fwd=(0,0,1). Kept local for
69      CameraController's strafing physics; the wire-format basis goes
70      through client.pack_inputs."""
71      cy, sy = math.cos(yaw), math.sin(yaw)
72      cp, sp = math.cos(pitch), math.sin(pitch)

```

```

73     right = (-cy,          sy)
74     up    = (-sy * sp, cp, -cy * sp)
75     forward = ( sy * cp, sp,  cy * cp)
76     return right, up, forward
77
78
79 def pack_inputs(lx: float, ly: float, lz: float,
80                sx: float, sy: float, sz: float,
81                max_depth: int, shift_0: int, shift_1: int,
82                cam_x: float, cam_z: float,
83                yaw: float, pitch: float,
84                r0: float = 1.0, r1: float = 1.0,
85                ) -> bytes:
86     return client.pack_inputs(
87         lx, ly, lz, sx, sy, sz,
88         max_depth=max_depth, shift_0=shift_0, shift_1=shift_1,
89         cam_x=cam_x, cam_z=cam_z, yaw=yaw, pitch=pitch,
90         r0=r0, r1=r1,
91     )
92
93
94 def recv_frame(sock: socket.socket) -> bytes:
95     buf = bytearray(FRAME_BYTES)
96     view = memoryview(buf)
97     got = 0
98     while got < FRAME_BYTES:
99         n = sock.recv_into(view[got:])
100         if n == 0:
101             raise ConnectionError(f"server closed after {got}/{FRAME_BYTES} bytes")
102         got += n
103     return bytes(buf)
104
105
106 def decode(data: bytes) -> Image.Image:
107     return Image.frombuffer("RGB", (WIDTH, HEIGHT), data, "raw", "BGRX", 0, 1)
108
109
110 class CameraController:
111     """Joystick-driven camera state, integrated each frame against dt.
112
113     Translation is camera-relative on the XZ ground plane (pitch is not
114     applied to walk - the camera doesn't fly). Strafe direction is the
115     player's right when looking +fwd, derived from yaw alone - this is
116     the player-strafe basis, NOT the ray-direction basis from
117     camera_basis_from_euler (those differ by a world-x flip from the
118     screenworld mapping).
119     """
120
121     def __init__(self, joy,
122                  x: float = DEFAULT_CAM_X, z: float = DEFAULT_CAM_Z,
123                  yaw: float = DEFAULT_YAW, pitch: float = DEFAULT_PITCH,
124                  move_scale: float = 3.0,
125                  rot_scale_deg: float = 90.0,
126                  deadzone: float = 0.15,
127                  rot_deadzone: float = 0.30,
128                  invert_yaw: bool = True,
129                  invert_pitch: bool = False,
130                  invert_strafe: bool = True):
131         self.joy = joy
132         self.x = x
133         self.z = z
134         self.yaw = yaw
135         self.pitch = pitch
136         self.move_scale = move_scale
137         self.rot_scale = math.radians(rot_scale_deg)
138         # Wider rotation cliff because flight-stick twist/throttle don't
139         # self-center the way the spring-loaded translation axes do.
140         self.deadzone = deadzone
141         self.rot_deadzone = rot_deadzone
142         # Default sign flips picked to match Logitech Extreme 3D Pro feel:
143         # twist CW → camera right, stick right → walk toward screen-right.
144         self.yaw_sign = -1.0 if invert_yaw else 1.0
145         self.pitch_sign = -1.0 if invert_pitch else 1.0
146         self.strafe_sign = -1.0 if invert_strafe else 1.0

```

```

147         self.has_right_stick = joy.get_numaxes() >= 4
148
149     def _dz(self, v: float, dz: float = None) -> float:
150         """Deadzone with linear remap so partial deflection is still useful."""
151         if dz is None:
152             dz = self.deadzone
153         if abs(v) < dz:
154             return 0.0
155         sign = 1.0 if v > 0.0 else -1.0
156         return sign * (abs(v) - dz) / (1.0 - dz)
157
158     def update(self, dt: float) -> None:
159         lx = self._dz(self.joy.get_axis(0))
160         ly = self._dz(self.joy.get_axis(1))
161         if self.has_right_stick:
162             rx = self._dz(self.joy.get_axis(2), self.rot_deadzone)
163             ry = self._dz(self.joy.get_axis(3), self.rot_deadzone)
164         else:
165             rx = ry = 0.0
166
167         self.yaw += rx * self.rot_scale * dt * self.yaw_sign
168         self.pitch += -ry * self.rot_scale * dt * self.pitch_sign
169         if self.pitch > PITCH_LIMIT: self.pitch = PITCH_LIMIT
170         if self.pitch < -PITCH_LIMIT: self.pitch = -PITCH_LIMIT
171
172         sy, cy = math.sin(self.yaw), math.cos(self.yaw)
173         walk = -ly
174         strafe = lx * self.strafe_sign
175
176         self.x += (sy * walk + cy * strafe) * self.move_scale * dt
177         self.z += (cy * walk - sy * strafe) * self.move_scale * dt
178
179
180     def render_stream(sock: socket.socket,
181                     light: tuple[float, float, float],
182                     sphere: tuple[float, float, float],
183                     start: tuple[float, float, float, float],
184                     move_scale: float,
185                     rot_scale_deg: float,
186                     rot_deadzone: float,
187                     invert_yaw: bool,
188                     invert_pitch: bool,
189                     invert_strafe: bool,
190                     max_depth: int,
191                     shift_0: int,
192                     shift_1: int,
193                     r0_start: float,
194                     r1_start: float,
195                     radius_rate: float,
196                     radius_min: float,
197                     radius_max: float,
198                     btn_r0_grow: int,
199                     btn_r0_shrink: int,
200                     btn_r1_grow: int,
201                     btn_r1_shrink: int) -> None:
202         """Pipeline-depth-2 stream loop with joystick camera. Same prime +
203         steady-state recv/send pattern as client.render_stream."""
204         try:
205             import pygame
206         except ImportError:
207             raise SystemExit(
208                 "pygame is required for camera.py. Install it with: pip install pygame"
209             )
210
211         pygame.init()
212         screen = pygame.display.set_mode((WIDTH, HEIGHT), pygame.FULLSCREEN | pygame.SCALED)
213         pygame.display.set_caption("FPGA Raytracer (camera)")
214         clock = pygame.time.Clock()
215
216         pygame.joystick.init()
217         if pygame.joystick.get_count() == 0:
218             print("ERROR: No joysticks found!")
219             return
220         joy = pygame.joystick.Joystick(0)

```



```

221 joy.init()
222 print(f"Connected: {joy.get_name()} ({joy.get_numaxes()} axes)")
223
224 sx, sz, syaw, spitch = start
225 camera = CameraController(joy, x=sx, z=sz, yaw=syaw, pitch=spitch,
226                             move_scale=move_scale,
227                             rot_scale_deg=rot_scale_deg,
228                             rot_deadzone=rot_deadzone,
229                             invert_yaw=invert_yaw,
230                             invert_pitch=invert_pitch,
231                             invert_strafe=invert_strafe)
232
233 if camera.has_right_stick:
234     print(f"axis2/3 cliff: ±{camera.rot_deadzone:.2f} around 0 "
235           f"(park throttle/twist at center for no rotation)")
236
237 nbuttons = joy.get_numbuttons()
238 radius_buttons_ok = (
239     max(btn_r0_grow, btn_r0_shrink, btn_r1_grow, btn_r1_shrink) < nbuttons
240 )
241 if not radius_buttons_ok:
242     print(f"WARN: joystick has {nbuttons} buttons; radius buttons "
243           f"({btn_r0_grow}/{btn_r0_shrink}/{btn_r1_grow}/{btn_r1_shrink}) "
244           f"out of range - radii will stay at start values.")
245 else:
246     print(f"radii: hold btn{btn_r0_grow}/{btn_r0_shrink} for r0 grow/shrink, "
247           f"btn{btn_r1_grow}/{btn_r1_shrink} for r1 ({radius_rate:.2f} u/s, "
248           f"clamped [{radius_min:.2f}, {radius_max:.2f}])")
249
250 r0 = max(radius_min, min(radius_max, r0_start))
251 r1 = max(radius_min, min(radius_max, r1_start))
252
253 def make_inputs() -> bytes:
254     return pack_inputs(*light, *sphere,
255                         max_depth, shift_0, shift_1,
256                         camera.x, camera.z, camera.yaw, camera.pitch,
257                         r0, r1)
258
259 sock.sendall(make_inputs())
260 sock.sendall(make_inputs())
261
262 frames = 0
263 last_t = time.monotonic()
264 running = True
265 try:
266     while running:
267         for ev in pygame.event.get():
268             if ev.type == pygame.QUIT:
269                 running = False
270                 break
271             if ev.type == pygame.KEYDOWN and ev.key == pygame.K_ESCAPE:
272                 running = False
273                 break
274         if not running:
275             break
276
277         data = recv_frame(sock)
278
279         now = time.monotonic()
280         dt = now - last_t
281         last_t = now
282         camera.update(dt)
283
284         # Hold-to-grow/shrink radii at radius_rate while a button is held.
285         if radius_buttons_ok:
286             if joy.get_button(btn_r0_grow):
287                 r0 = min(radius_max, r0 + radius_rate * dt)
288             if joy.get_button(btn_r0_shrink):
289                 r0 = max(radius_min, r0 - radius_rate * dt)
290             if joy.get_button(btn_r1_grow):
291                 r1 = min(radius_max, r1 + radius_rate * dt)
292             if joy.get_button(btn_r1_shrink):
293                 r1 = max(radius_min, r1 - radius_rate * dt)
294
295         sock.sendall(make_inputs())

```

```

295         img = decode(data)
296         surf = pygame.image.fromstring(img.tobytes(), img.size, img.mode)
297         screen.blit(surf, (0, 0))
298         pygame.display.flip()
299
300     frames += 1
301     clock.tick()
302     if frames % 30 == 0:
303         print(f"{frames} frames ({clock.get_fps():.1f} fps) "
304               f"cam=({camera.x:+.2f}, 1.50, {camera.z:+.2f}) "
305               f"yaw={math.degrees(camera.yaw):+6.1f}° "
306               f"pitch={math.degrees(camera.pitch):+5.1f}° "
307               f"r0={r0:.2f} r1={r1:.2f}")
308
309     finally:
310         pygame.quit()
311         print(f"streamed {frames} frames")
312
313
314 def main():
315     parser = argparse.ArgumentParser(
316         description="FPGA raytracer client - joystick drives camera"
317     )
318     parser.add_argument("host", help="FPGA server IP or hostname")
319     parser.add_argument("port", type=int, help="FPGA server port")
320
321     parser.add_argument("--light", nargs=3, type=float,
322                         default=[0.0, 4.0, 0.0],
323                         metavar=("LX", "LY", "LZ"),
324                         help="static light position (default: 0 4 0)")
325     parser.add_argument("--sphere1", nargs=3, type=float,
326                         default=list(SPHERE1_DEFAULT_CENTER),
327                         metavar=("SX", "SY", "SZ"),
328                         help=f"static sphere-1 center (default: {SPHERE1_DEFAULT_CENTER})")
329
330     parser.add_argument("--start-x", type=float, default=DEFAULT_CAM_X)
331     parser.add_argument("--start-z", type=float, default=DEFAULT_CAM_Z)
332     parser.add_argument("--start-yaw", type=float, default=0.0,
333                         help="starting yaw in degrees (default 0)")
334     parser.add_argument("--start-pitch", type=float, default=0.0,
335                         help="starting pitch in degrees (default 0)")
336
337     parser.add_argument("--move-scale", type=float, default=3.0,
338                         help="translation speed at full deflection (units/sec)")
339     parser.add_argument("--rot-scale", type=float, default=90.0,
340                         help="rotation speed at full deflection (deg/sec)")
341     parser.add_argument("--rot-deadzone", type=float, default=0.30,
342                         help="cliff/deadzone for axes 2/3 (yaw/pitch). Larger "
343                             "= more dead area around the launch zero before "
344                             "rotation starts integrating. Default 0.30 = 30%% "
345                             "of full range either side.")
346     parser.add_argument("--no-invert-yaw", action="store_true",
347                         help="don't flip yaw sign. Default IS to invert "
348                             "(matches Logitech Extreme 3D Pro twist convention).")
349     parser.add_argument("--invert-pitch", action="store_true",
350                         help="flip pitch sign (default: not inverted).")
351     parser.add_argument("--no-invert-strafe", action="store_true",
352                         help="don't flip strafe sign. Default IS to invert "
353                             "(HW basis right=(-cos(yaw),sin(yaw)) mirrors "
354                             "world-x to screen-x; flipping makes stick-right "
355                             "move the camera toward screen-right).")
356
357     parser.add_argument("--max-depth", type=int, default=DEFAULT_MAX_DEPTH,
358                         help=f"reflection recursion cap (0..7; default {DEFAULT_MAX_DEPTH})")
359     parser.add_argument("--shift0", type=int, default=DEFAULT_REFLECT_SHIFT_0)
360     parser.add_argument("--shift1", type=int, default=DEFAULT_REFLECT_SHIFT_1)
361
362     parser.add_argument("--r0", type=float, default=1.0,
363                         help="starting radius for sphere 0 (blue). Default 1.0.")
364     parser.add_argument("--r1", type=float, default=1.0,
365                         help="starting radius for sphere 1 (red). Default 1.0.")
366     parser.add_argument("--radius-rate", type=float, default=1.0,
367                         help="grow/shrink rate while a radius button is held "
368                             "(units/sec). Default 1.0.")

```

```

369 parser.add_argument("--radius-min", type=float, default=0.2,
370                     help="lower clamp on radii (default 0.2).")
371 parser.add_argument("--radius-max", type=float, default=4.0,
372                     help="upper clamp on radii (default 4.0).")
373 parser.add_argument("--btn-r0-grow", type=int, default=5,
374                     help="joystick button to grow sphere-0 radius (default 0)")
375 parser.add_argument("--btn-r0-shrink", type=int, default=3,
376                     help="joystick button to shrink sphere-0 radius (default 1)")
377 parser.add_argument("--btn-r1-grow", type=int, default=4,
378                     help="joystick button to grow sphere-1 radius (default 2)")
379 parser.add_argument("--btn-r1-shrink", type=int, default=2,
380                     help="joystick button to shrink sphere-1 radius (default 3)")
381
382 args = parser.parse_args()
383
384 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
385 sock.connect((args.host, args.port))
386 try:
387     render_stream(
388         sock,
389         light=tuple(args.light),
390         sphere=tuple(args.sphere1),
391         start=(args.start_x, args.start_z,
392              math.radians(args.start_yaw),
393              math.radians(args.start_pitch)),
394         move_scale=args.move_scale,
395         rot_scale_deg=args.rot_scale,
396         rot_deadzone=args.rot_deadzone,
397         invert_yaw=not args.no_invert_yaw,
398         invert_pitch=args.invert_pitch,
399         invert_strafe=not args.no_invert_strafe,
400         max_depth=args.max_depth,
401         shift_0=args.shift0,
402         shift_1=args.shift1,
403         r0_start=args.r0,
404         r1_start=args.r1,
405         radius_rate=args.radius_rate,
406         radius_min=args.radius_min,
407         radius_max=args.radius_max,
408         btn_r0_grow=args.btn_r0_grow,
409         btn_r0_shrink=args.btn_r0_shrink,
410         btn_r1_grow=args.btn_r1_grow,
411         btn_r1_shrink=args.btn_r1_shrink,
412     )
413 finally:
414     sock.close()
415
416
417 if __name__ == "__main__":
418     main()

```

SW/desktop/pong.py

```

1  #!/usr/bin/env python3
2  """Pong, rendered by the FPGA raytracer.
3
4  Two spheres are paddles and the cone is the ball, played in the XY
5  plane in front of the default camera:
6
7      Sphere 0 (blue) -> LEFT paddle (player)
8      Sphere 1 (red)  -> RIGHT paddle (AI)
9      Cone    (white) -> ball
10
11  Controls (joystick, like camera_cone.py):
12      Axis 1    -> player paddle up/down (stick up = paddle up)
13      Button 0  -> serve / next round / restart after game over
14      ESC      -> quit
15
16  If no joystick is connected the demo falls back to AI-vs-AI: the
17  "player" paddle also tracks the ball, and serves / restarts trigger
18  automatically after a short pause so the screen never sits idle.
19

```

```

20 The wire / prime-the-pipeline pattern matches camera_cone.py: send 2
21 input blobs up front, then 1 per received frame. Game state is
22 integrated against wall-clock dt between recv_frame calls.
23
24 Usage:
25     python pong.py HOST PORT
26     """
27
28 import argparse
29 import math
30 import random
31 import socket
32 import time
33
34 import client
35 import camera
36
37
38 # Court geometry (world units). Paddles sit at x = ±PADDLE_X; ball is
39 # the cone primitive so BALL_R is the collision radius, separate from
40 # the rendered cone height.
41 PADDLE_X    = 3.0
42 PADDLE_R    = 0.6
43 BALL_R      = 0.35
44 COURT_Y_MIN = 0.5
45 COURT_Y_MAX = 4.0
46 COURT_X_OUT = 4.5
47
48 BALL_CONE_HEIGHT = 0.7
49 BALL_CONE_K_SQ   = 1.0          # 45° half-angle → base radius = height
50 BALL_COLOR       = (0.95, 0.95, 0.95)
51
52 PLAYER_SPEED     = 4.5
53 AI_SPEED         = 3.0          # < PLAYER_SPEED so the AI is beatable
54 BALL_SPEED_INIT  = 3.5
55 BALL_SPEED_MAX   = 7.5
56
57 WIN_SCORE = 5
58
59 LIGHT_POS = (1.5, 4.5, -2.0)
60 CAM_X     = 0.0
61 CAM_Z     = -8.0
62 CAM_YAW   = 0.0
63 CAM_PITCH = math.radians(8.0)   # tilt up so ceiling isn't off-screen
64
65 STATE_SERVING = "serving"
66 STATE_PLAYING = "playing"
67 STATE_GAMEOVER = "gameover"
68
69 PADDLE_Y_MIN = COURT_Y_MIN + PADDLE_R
70 PADDLE_Y_MAX = COURT_Y_MAX - PADDLE_R
71 BALL_Y_MIN   = COURT_Y_MIN + BALL_R
72 BALL_Y_MAX   = COURT_Y_MAX - BALL_R
73
74
75 class PongGame:
76     def __init__(self):
77         mid = 0.5 * (COURT_Y_MIN + COURT_Y_MAX)
78         self.player_y = mid
79         self.ai_y     = mid
80         self.ball_x   = 0.0
81         self.ball_y   = mid
82         self.vx       = 0.0
83         self.vy       = 0.0
84         self.score_p  = 0
85         self.score_a  = 0
86         self.state    = STATE_SERVING
87         self._set_serve_velocity(direction=-1)
88
89     def _set_serve_velocity(self, direction: int):
90         angle = random.uniform(-0.30, 0.30) * math.pi
91         self.vx = direction * BALL_SPEED_INIT * math.cos(angle)
92         self.vy = BALL_SPEED_INIT * math.sin(angle)
93

```

```

94 def _reset_ball(self, direction: int):
95     self.ball_x = 0.0
96     self.ball_y = 0.5 * (COURT_Y_MIN + COURT_Y_MAX)
97     self._set_serve_velocity(direction)
98
99 def serve(self):
100     if self.state == STATE_SERVING:
101         self.state = STATE_PLAYING
102     elif self.state == STATE_GAMEOVER:
103         self.score_p = 0
104         self.score_a = 0
105         self._reset_ball(direction=-1)
106         self.state = STATE_PLAYING
107
108 def update_player(self, axis: float, dt: float):
109     # joystick up reports negative; negate so stick-up moves the paddle up.
110     self.player_y -= axis * PLAYER_SPEED * dt
111     if self.player_y < PADDLE_Y_MIN: self.player_y = PADDLE_Y_MIN
112     if self.player_y > PADDLE_Y_MAX: self.player_y = PADDLE_Y_MAX
113
114 def _track_ball(self, paddle_y: float, dt: float) -> float:
115     delta = self.ball_y - paddle_y
116     step = AI_SPEED * dt
117     if abs(delta) <= step:
118         paddle_y = self.ball_y
119     else:
120         paddle_y += step if delta > 0 else -step
121     if paddle_y < PADDLE_Y_MIN: paddle_y = PADDLE_Y_MIN
122     if paddle_y > PADDLE_Y_MAX: paddle_y = PADDLE_Y_MAX
123     return paddle_y
124
125 def _update_ai(self, dt: float):
126     self.ai_y = self._track_ball(self.ai_y, dt)
127
128 def update_player_ai(self, dt: float):
129     self.player_y = self._track_ball(self.player_y, dt)
130
131 def _paddle_hit(self, paddle_y: float, dir_x: int):
132     # Reflect x, add english from where the ball hit on the paddle,
133     # bump speed slightly each rally (capped at BALL_SPEED_MAX).
134     rel = (self.ball_y - paddle_y) / PADDLE_R
135     rel = max(-1.0, min(1.0, rel))
136     speed = math.hypot(self.vx, self.vy) * 1.05
137     if speed > BALL_SPEED_MAX:
138         speed = BALL_SPEED_MAX
139     angle = rel * math.radians(55.0)
140     self.vx = dir_x * speed * math.cos(angle)
141     self.vy = speed * math.sin(angle)
142
143 def step(self, dt: float):
144     if self.state != STATE_PLAYING:
145         return
146
147     self._update_ai(dt)
148
149     self.ball_x += self.vx * dt
150     self.ball_y += self.vy * dt
151
152     if self.ball_y < BALL_Y_MIN:
153         self.ball_y = BALL_Y_MIN
154         self.vy = abs(self.vy)
155     elif self.ball_y > BALL_Y_MAX:
156         self.ball_y = BALL_Y_MAX
157         self.vy = -abs(self.vy)
158
159     # Gate on vx sign so a ball that already passed the paddle plane
160     # can't re-collide on the way out.
161     rsum2 = (PADDLE_R + BALL_R) ** 2
162     if self.vx < 0:
163         dx = self.ball_x - (-PADDLE_X)
164         dy = self.ball_y - self.player_y
165         if dx > 0 and dx * dx + dy * dy < rsum2:
166             self._paddle_hit(self.player_y, +1)
167     elif self.vx > 0:

```

```

168         dx = self.ball_x - PADDLE_X
169         dy = self.ball_y - self.ai_y
170         if dx < 0 and dx * dx + dy * dy < rsum2:
171             self._paddle_hit(self.ai_y, -1)
172
173         if self.ball_x < -COURT_X_OUT:
174             self.score_a += 1
175             self._after_score(direction=+1)
176         elif self.ball_x > COURT_X_OUT:
177             self.score_p += 1
178             self._after_score(direction=-1)
179
180     def _after_score(self, direction: int):
181         if self.score_p >= WIN_SCORE or self.score_a >= WIN_SCORE:
182             self.state = STATE_GAMEOVER
183         else:
184             self.state = STATE_SERVING
185             self._reset_ball(direction)
186
187
188 def run(sock):
189     try:
190         import pygame
191     except ImportError:
192         raise SystemExit("pygame required: pip install pygame")
193
194     pygame.init()
195     screen = pygame.display.set_mode((camera.WIDTH, camera.HEIGHT), pygame.SCALED)
196     pygame.display.set_caption("FPGA Raytracer Pong")
197     clock = pygame.time.Clock()
198     pygame.font.init()
199     big_font = pygame.font.SysFont("monospace", 36, bold=True)
200     sm_font = pygame.font.SysFont("monospace", 14, bold=True)
201
202     pygame.joystick.init()
203     joy = None
204     if pygame.joystick.get_count() > 0:
205         joy = pygame.joystick.Joystick(0)
206         joy.init()
207         print(f"Connected: {joy.get_name()} ({joy.get_numaxes()} axes)")
208     else:
209         print("No joystick - running AI-vs-AI demo mode")
210
211     # No-joystick mode auto-serves on these delays so the screen never
212     # idles on the "press button" prompt.
213     AUTO_SERVE_DELAY = 1.0
214     AUTO_RESTART_DELAY = 3.0
215     auto_serve_at = None
216
217     game = PongGame()
218
219     def make_inputs() -> bytes:
220         sc0 = (-PADDLE_X, game.player_y, 0.0)
221         sc1 = ( PADDLE_X, game.ai_y, 0.0)
222         # Apex sits half a cone-height above the logical ball position so
223         # the rendered cone body straddles the collider symmetrically.
224         cone_apex = (game.ball_x, game.ball_y + 0.5 * BALL_CONE_HEIGHT, 0.0)
225         return client.pack_inputs(
226             *LIGHT_POS, *sc1,
227             max_depth=2, shift_0=2, shift_1=2,
228             cam_x=CAM_X, cam_z=CAM_Z,
229             yaw=CAM_YAW, pitch=CAM_PITCH,
230             r0=PADDLE_R, r1=PADDLE_R,
231             sc0=sc0,
232             cone_apex=cone_apex,
233             cone_k_sq=BALL_CONE_K_SQ,
234             cone_height=BALL_CONE_HEIGHT,
235             cone_col_rgb=BALL_COLOR,
236             reflect_shift_cone=0,
237         )
238
239     sock.sendall(make_inputs())
240     sock.sendall(make_inputs())
241

```

```

242 last_t = time.monotonic()
243 running = True
244 try:
245     while running:
246         for ev in pygame.event.get():
247             if ev.type == pygame.QUIT:
248                 running = False
249                 break
250             if ev.type == pygame.KEYDOWN and ev.key == pygame.K_ESCAPE:
251                 running = False
252                 break
253             if ev.type == pygame.JOYBUTTONDOWN and ev.button == 0:
254                 game.serve()
255             if ev.type == pygame.KEYDOWN and ev.key == pygame.K_SPACE:
256                 game.serve()
257         if not running:
258             break
259
260         data = camera.recv_frame(sock)
261
262         now = time.monotonic()
263         dt = now - last_t
264         if dt > 0.05:          # clamp so a network stall doesn't teleport the ball
265             dt = 0.05
266         last_t = now
267
268         if joy is not None:
269             axis = joy.get_axis(1)
270             if abs(axis) < 0.15:
271                 axis = 0.0
272             game.update_player(axis, dt)
273         else:
274             game.update_player_ai(dt)
275             if game.state == STATE_PLAYING:
276                 auto_serve_at = None
277             else:
278                 delay = AUTO_RESTART_DELAY if game.state == STATE_GAMEOVER \
279                     else AUTO_SERVE_DELAY
280                 if auto_serve_at is None:
281                     auto_serve_at = now + delay
282                 elif now >= auto_serve_at:
283                     game.serve()
284                     auto_serve_at = None
285             game.step(dt)
286
287         sock.sendall(make_inputs())
288
289         img = camera.decode(data)
290         surf = pygame.image.fromstring(img.tobytes(), img.size, img.mode)
291         screen.blit(surf, (0, 0))
292
293         score_txt = f"{game.score_p} {game.score_a}"
294         score_surf = big_font.render(score_txt, True, (255, 255, 255))
295         score_shadow = big_font.render(score_txt, True, (0, 0, 0))
296         sx = camera.WIDTH // 2 - score_surf.get_width() // 2
297         screen.blit(score_shadow, (sx + 2, 10))
298         screen.blit(score_surf, (sx, 8))
299
300         if game.state == STATE_SERVING:
301             msg = "serving..." if joy is None else "press button 0 to serve"
302         elif game.state == STATE_GAMEOVER:
303             winner = "PLAYER" if game.score_p > game.score_a else "AI"
304             tail = "restarting..." if joy is None else "press button 0"
305             msg = f"{winner} WINS - {tail}"
306         else:
307             msg = None
308         if msg is not None:
309             m_surf = sm_font.render(msg, True, (255, 255, 255))
310             m_shadow = sm_font.render(msg, True, (0, 0, 0))
311             mx = camera.WIDTH // 2 - m_surf.get_width() // 2
312             my = camera.HEIGHT - 28
313             screen.blit(m_shadow, (mx + 1, my + 1))
314             screen.blit(m_surf, (mx, my))
315

```

```

316         fps_txt = f"{clock.get_fps():5.1f} fps"
317         f_surf = sm_font.render(fps_txt, True, (255, 255, 255))
318         f_shadow = sm_font.render(fps_txt, True, (0, 0, 0))
319         screen.blit(f_shadow, (7, 7))
320         screen.blit(f_surf, (6, 6))
321
322         pygame.display.flip()
323         clock.tick()
324     finally:
325         pygame.quit()
326
327
328 def main():
329     ap = argparse.ArgumentParser(
330         description="Pong, rendered by the FPGA raytracer"
331     )
332     ap.add_argument("host")
333     ap.add_argument("port", type=int)
334     args = ap.parse_args()
335
336     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
337     sock.connect((args.host, args.port))
338     try:
339         run(sock)
340     finally:
341         sock.close()
342
343
344 if __name__ == "__main__":
345     main()

```

SW/desktop/sandbox.py

```

1  #!/usr/bin/env python3
2  """Sandbox - two-flightstick puppet show that exposes every CSR knob.
3
4  Per-stick controls (stick 0 drives ball 0, stick 1 drives ball 1):
5
6      Main stick (axes 0/1)
7          default          ball X/Z velocity (this stick's ball)
8          +trigger (btn 0) held camera X/Z velocity (twist = yaw)
9          +side   (btn 10) held camera pitch (stick fwd/back = look up/down)
10         +thumb  (btn 1) held stick 0: cone k^2 (X) + height (Y)
11                                stick 1: cone apex X / apex Z
12      Throttle (axis 3)      stick 0: cone reflectivity (up=bright, down=matte)
13                                stick 1: ball radius velocity (additive w/ btns)
14      POV hat                stick 0: light X / Z
15                                stick 1: light Y
16
17      Top cluster
18          btn 5 / 3          ball radius grow / shrink
19          btn 4 / 2          ball Y up / down
20
21      Base
22          stick 0:
23              btn 6          cycle floor mode (0..3)
24              btn 7          cycle ball 0 color
25              btn 8          toggle cone on/off
26              btn 9          cycle max reflection depth (0..4)
27          stick 1:
28              btn 6          cycle ball 1 color
29              btn 7          cycle cone color
30              btn 8          cycle cone shape preset (k^2 + height)
31
32      Trigger and thumb are per-stick modifiers. Holding either stick's
33      trigger puts that stick into camera mode; the OTHER stick keeps doing
34      its default thing. The cone apex's y coordinate tracks the cone height
35      so the cone base stays on the floor.
36
37      Usage:
38          python sandbox.py HOST PORT
39
40      """
41
42      import argparse

```



```

40 import math
41 import socket
42 import time
43
44 import client
45 import camera
46
47
48 WIDTH = camera.WIDTH
49 HEIGHT = camera.HEIGHT
50 CAM_Y = 1.5
51
52 # Rates / scales - world units (or radians) per second.
53 BALL_MOVE_SCALE = 3.0
54 BALL_Y_RATE = 1.5
55 RADIUS_RATE = 1.0
56 CAM_MOVE_SCALE = 4.0
57 YAW_RATE = math.radians(90.0)
58 PITCH_RATE = math.radians(70.0)
59 PITCH_LIMIT = math.radians(80.0)
60 LIGHT_MOVE_SCALE = 3.0
61 LIGHT_Y_RATE = 1.5
62 CONE_HEIGHT_RATE = 1.5
63 CONE_K_RATE = 0.6
64 CONE_APEX_RATE = 2.0
65
66 BALL_Y_MIN, BALL_Y_MAX = 0.3, 5.0
67 LIGHT_Y_MIN, LIGHT_Y_MAX = 0.5, 8.0
68 R_MIN, R_MAX = 0.2, 3.0
69 CONE_H_MIN, CONE_H_MAX = 0.3, 5.0
70 CONE_K_MIN, CONE_K_MAX = 0.05, 2.0
71 CAM_X_LIM, CAM_Z_LIM = 15.0, 15.0
72
73 DZ_STICK = 0.15
74 DZ_TWIST = 0.30
75 DZ_THROTTLE = 0.15
76
77 BALL_COLORS = [
78     (0.20, 0.30, 0.80), (0.80, 0.30, 0.20),
79     (0.20, 0.80, 0.30), (0.95, 0.85, 0.20),
80     (0.70, 0.30, 0.90), (0.95, 0.55, 0.20),
81     (0.20, 0.80, 0.80), (0.95, 0.95, 0.95),
82 ]
83 CONE_COLORS = [
84     (0.95, 0.55, 0.20),
85     (0.50, 0.50, 0.50),
86     (0.20, 0.70, 0.30),
87     (0.80, 0.20, 0.80),
88     (0.30, 0.30, 0.90),
89 ]
90
91 # (k_sq, height) presets. The pack step pins apex.y = height so the
92 # base always sits on the floor.
93 CONE_SHAPES = [
94     (0.25, 3.0), # narrow, tall
95     (0.10, 4.0), # spire
96     (0.50, 2.5), # standard
97     (1.00, 1.5), # 45° wide & short
98     (2.00, 1.2), # very wide cap
99 ]
100
101 # reflect_shift_cone is a >> shift on the reflection contribution:
102 # 0 = brightest mirror; 8 = HW won't spawn a reflection ray (matte).
103 REFLECT_SHIFT_MIN = 0
104 REFLECT_SHIFT_MAX = 8
105
106 MAX_DEPTH_LEVELS = [0, 1, 2, 3, 4]
107
108
109 def deadzone(v: float, dz: float = DZ_STICK) -> float:
110     if abs(v) < dz:
111         return 0.0
112     s = 1.0 if v > 0.0 else -1.0
113     return s * (abs(v) - dz) / (1.0 - dz)

```

```

114
115
116 def clamp(x, lo, hi):
117     return lo if x < lo else (hi if x > hi else x)
118
119
120 class State:
121     def __init__(self):
122         self.b0      = [-1.5, 1.0, 0.0]
123         self.r0      = 1.0
124         self.color0   = 0
125
126         self.b1      = [ 1.5, 1.0, 0.0]
127         self.r1      = 1.0
128         self.color1   = 1
129
130         self.light    = [3.0, 4.0, 2.0]
131
132         self.cam_x    = 0.0
133         self.cam_z    = -8.0
134         self.yaw      = 0.0
135         self.pitch    = math.radians(-3.0)
136
137         self.cone_on   = False
138         self.cone_apex_xz = [0.0, 1.0]
139         self.cone_height = 3.0
140         self.cone_k_sq  = 0.25
141         self.cone_color = 0
142         self.cone_shape = 0
143         self.reflect_shift_cone = 2
144         self.max_depth_idx = 3
145
146         self.floor_mode = 0
147
148     def pack(self) -> bytes:
149         # apex.y = height keeps the cone base on y = 0 (the floor plane).
150         h = self.cone_height if self.cone_on else 0.0
151         apex = (self.cone_apex_xz[0], h, self.cone_apex_xz[1])
152         return client.pack_inputs(
153             *self.light, *self.b1,
154             max_depth=MAX_DEPTH_LEVELS[self.max_depth_idx],
155             shift_0=1, shift_1=1,
156             cam_x=self.cam_x, cam_z=self.cam_z,
157             yaw=self.yaw, pitch=self.pitch,
158             r0=self.r0, r1=self.r1,
159             sc0=tuple(self.b0),
160             scol0_rgb=BALL_COLORS[self.color0],
161             scol1_rgb=BALL_COLORS[self.color1],
162             floor_mode=self.floor_mode,
163             cone_apex=apex,
164             cone_k_sq=self.cone_k_sq,
165             cone_height=h,
166             cone_col_rgb=CONE_COLORS[self.cone_color],
167             reflect_shift_cone=self.reflect_shift_cone,
168         )
169
170
171     def _btn(joy, idx) -> bool:
172         return idx < joy.get_numbuttons() and joy.get_button(idx) != 0
173
174
175     def _axis(joy, idx, dz=DZ_STICK) -> float:
176         if idx >= joy.get_numaxes():
177             return 0.0
178         return deadzone(joy.get_axis(idx), dz)
179
180
181     def _hat(joy):
182         if joy.get_numhats() == 0:
183             return (0, 0)
184         return joy.get_hat(0)
185
186
187     def update_stick(state: State, joy, joy_idx: int, dt: float):

```

```

188     """Read one joystick's inputs and mutate state accordingly."""
189     if joy is None:
190         return
191
192     mx = _axis(joy, 0)
193     my = _axis(joy, 1)
194     trigger = _btn(joy, 0)
195     thumb = _btn(joy, 1)
196     side = _btn(joy, 10)
197
198     # Main stick: per-stick mode-switch on trigger / side / thumb.
199     if trigger:
200         sy, cy = math.sin(state.yaw), math.cos(state.yaw)
201         walk = -my
202         strafe = -mx
203         state.cam_x += (sy * walk + cy * strafe) * CAM_MOVE_SCALE * dt
204         state.cam_z += (cy * walk - sy * strafe) * CAM_MOVE_SCALE * dt
205         state.cam_x = clamp(state.cam_x, -CAM_X_LIM, CAM_X_LIM)
206         state.cam_z = clamp(state.cam_z, -CAM_Z_LIM, CAM_Z_LIM)
207         twist = _axis(joy, 2, DZ_TWIST)
208         state.yaw += -twist * YAW_RATE * dt
209     elif side:
210         state.pitch += -my * PITCH_RATE * dt
211         state.pitch = clamp(state.pitch, -PITCH_LIMIT, PITCH_LIMIT)
212     elif thumb:
213         if joy_idx == 0:
214             state.cone_k_sq += mx * CONE_K_RATE * dt
215             state.cone_height += -my * CONE_HEIGHT_RATE * dt
216             state.cone_k_sq = clamp(state.cone_k_sq, CONE_K_MIN, CONE_K_MAX)
217             state.cone_height = clamp(state.cone_height, CONE_H_MIN, CONE_H_MAX)
218         else:
219             state.cone_apex_xz[0] += mx * CONE_APEX_RATE * dt
220             state.cone_apex_xz[1] += -my * CONE_APEX_RATE * dt
221     else:
222         ball = state.b0 if joy_idx == 0 else state.b1
223         ball[0] += -mx * BALL_MOVE_SCALE * dt
224         ball[2] += -my * BALL_MOVE_SCALE * dt
225
226     hx, hy = _hat(joy)
227     if joy_idx == 0:
228         state.light[0] += hx * LIGHT_MOVE_SCALE * dt
229         state.light[2] += hy * LIGHT_MOVE_SCALE * dt
230     else:
231         state.light[1] += hy * LIGHT_Y_RATE * dt
232         state.light[1] = clamp(state.light[1], LIGHT_Y_MIN, LIGHT_Y_MAX)
233
234     # Logitech flightstick throttle: axis 3 = -1 fully up, +1 fully down.
235     throttle = _axis(joy, 3, DZ_THROTTLE)
236     if joy_idx == 0:
237         # Stick 0 throttle → cone reflect_shift (up = brightest mirror).
238         norm = (throttle + 1.0) * 0.5
239         shift = round(norm * (REFLECT_SHIFT_MAX - REFLECT_SHIFT_MIN)) + REFLECT_SHIFT_MIN
240         if shift < REFLECT_SHIFT_MIN: shift = REFLECT_SHIFT_MIN
241         if shift > REFLECT_SHIFT_MAX: shift = REFLECT_SHIFT_MAX
242         state.reflect_shift_cone = shift
243         dr_throttle = 0.0
244     else:
245         dr_throttle = -throttle * RADIUS_RATE * dt
246
247     dr_btn = (1.0 if _btn(joy, 5) else 0.0) - (1.0 if _btn(joy, 3) else 0.0)
248     dy_btn = (1.0 if _btn(joy, 4) else 0.0) - (1.0 if _btn(joy, 2) else 0.0)
249
250     r_attr = "r0" if joy_idx == 0 else "r1"
251     new_r = getattr(state, r_attr) + dr_throttle + dr_btn * RADIUS_RATE * dt
252     setattr(state, r_attr, clamp(new_r, R_MIN, R_MAX))
253
254     ball = state.b0 if joy_idx == 0 else state.b1
255     ball[1] += dy_btn * BALL_Y_RATE * dt
256     ball[1] = clamp(ball[1], BALL_Y_MIN, BALL_Y_MAX)
257
258
259 def handle_button_edge(ev, state: State):
260     """Edge-triggered toggles / cycles on JOYBUTTONDOWN."""
261     if ev.joy == 0:

```

```

262         if ev.button == 6: state.floor_mode = (state.floor_mode + 1) % 4
263         elif ev.button == 7: state.color0 = (state.color0 + 1) % len(BALL_COLORS)
264         elif ev.button == 8: state.cone_on = not state.cone_on
265         elif ev.button == 9:
266             state.max_depth_idx = (state.max_depth_idx + 1) % len(MAX_DEPTH_LEVELS)
267     elif ev.joy == 1:
268         if ev.button == 6: state.color1 = (state.color1 + 1) % len(BALL_COLORS)
269         elif ev.button == 7: state.cone_color = (state.cone_color + 1) % len(CONE_COLORS)
270         elif ev.button == 8:
271             state.cone_shape = (state.cone_shape + 1) % len(CONE_SHAPES)
272             state.cone_k_sq, state.cone_height = CONE_SHAPES[state.cone_shape]
273
274
275 def run(sock):
276     try:
277         import pygame
278     except ImportError:
279         raise SystemExit("pygame required: pip install pygame")
280
281     pygame.init()
282     screen = pygame.display.set_mode((WIDTH, HEIGHT), pygame.SCALED)
283     pygame.display.set_caption("FPGA Raytracer - Sandbox")
284     clock = pygame.time.Clock()
285     pygame.font.init()
286     sm = pygame.font.SysFont("monospace", 12, bold=True)
287
288     pygame.joystick.init()
289     joys = []
290     for i in range(pygame.joystick.get_count()):
291         j = pygame.joystick.Joystick(i)
292         j.init()
293         joys.append(j)
294         print(f"Joystick {i}: {j.get_name()} ({j.get_numaxes()} axes, "
295               f"{j.get_numbuttons()} buttons, {j.get_numhats()} hats)")
296     if not joys:
297         print("ERROR: no joysticks found - this demo requires at least one.")
298         return
299
300     state = State()
301     sock.sendall(state.pack())
302     sock.sendall(state.pack())
303
304     last_t = time.monotonic()
305     running = True
306     try:
307         while running:
308             for ev in pygame.event.get():
309                 if ev.type == pygame.QUIT:
310                     running = False
311                     break
312                 if ev.type == pygame.KEYDOWN and ev.key == pygame.K_ESCAPE:
313                     running = False
314                     break
315                 if ev.type == pygame.JOYBUTTONDOWN:
316                     print(f" joy{ev.joy} btn{ev.button}")
317                     handle_button_edge(ev, state)
318             if not running:
319                 break
320
321             data = camera.recv_frame(sock)
322
323             now = time.monotonic()
324             dt = now - last_t
325             if dt > 0.05:
326                 dt = 0.05
327             last_t = now
328
329             for i, j in enumerate(joys):
330                 update_stick(state, j, i, dt)
331
332             sock.sendall(state.pack())
333
334             img = camera.decode(data)
335             surf = pygame.image.fromstring(img.tobytes(), img.size, img.mode)

```

```

336         screen.blit(surf, (0, 0))
337
338     for i, j in enumerate(joys):
339         if _btn(j, 0): mode = "CAM"
340         elif _btn(j, 10): mode = "TILT"
341         elif _btn(j, 1): mode = "CONE"
342         else: mode = f"BALL{i}"
343         held = [str(b) for b in range(j.get_numbuttons()) if j.get_button(b)]
344         held_str = ",".join(held) if held else "-"
345         msg = f"S{i}:{mode} held:{held_str}"
346         surf_m = sm.render(msg, True, (255, 255, 255))
347         shadow = sm.render(msg, True, (0, 0, 0))
348         x = 6 + i * 180
349         screen.blit(shadow, (x + 1, 7))
350         screen.blit(surf_m, (x, 6))
351
352     lines = [
353         f"b0 ({state.b0[0]:+.1f},{state.b0[1]:.1f},{state.b0[2]:+.1f}) r={state.r0:.2f}",
354         f"b1 ({state.b1[0]:+.1f},{state.b1[1]:.1f},{state.b1[2]:+.1f}) r={state.r1:.2f}",
355         f"lt ({state.light[0]:+.1f},{state.light[1]:.1f},{state.light[2]:+.1f})",
356         f"cm ({state.cam_x:+.1f},1.5,{state.cam_z:+.1f}) yaw={math.degrees(state.yaw):+5.0f}",
357         f"cone {'ON' if state.cone_on else 'off'} h={state.cone_height:.1f} k2={state.cone_k_sq:.2f}",
358         f"refl shift={state.reflect_shift_cone} depth={MAX_DEPTH_LEVELS[state.max_depth_idx]}",
359         f"floor {state.floor_mode}",
360         f"{clock.get_fps():5.1f} fps",
361     ]
362     for k, line in enumerate(lines):
363         s_surf = sm.render(line, True, (255, 255, 255))
364         s_sh = sm.render(line, True, (0, 0, 0))
365         x = WIDTH - s_surf.get_width() - 6
366         y = 6 + k * 14
367         screen.blit(s_sh, (x + 1, y + 1))
368         screen.blit(s_surf, (x, y))
369
370     pygame.display.flip()
371     clock.tick()
372 finally:
373     pygame.quit()
374
375
376 def main():
377     ap = argparse.ArgumentParser(
378         description="FPGA raytracer sandbox - two-flightstick scene puppet"
379     )
380     ap.add_argument("host")
381     ap.add_argument("port", type=int)
382     args = ap.parse_args()
383
384     sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
385     sock.connect((args.host, args.port))
386     try:
387         run(sock)
388     finally:
389         sock.close()
390
391
392 if __name__ == "__main__":
393     main()

```

4 Python Reference + Cocotb Tests

raytracer_fp.py

```

1  """
2  Basic ray tracer: sphere on a plane with an orbiting light source.
3  Fixed-point version for embedded systems.
4  Uses Q14.13 format: 14-bit integer + 13-bit fractional (signed, total 27 bits).
5  """
6
7  from PIL import Image
8  import math

```

```

9
10 class FixedPoint:
11     SHIFT = 13
12     SCALE = 1 << SHIFT
13
14     @staticmethod
15     def from_float(f):
16         return int(round(f * FixedPoint.SCALE))
17
18     @staticmethod
19     def to_float(fx):
20         return fx / FixedPoint.SCALE
21
22     @staticmethod
23     def mul(a, b):
24         return (a * b) >> FixedPoint.SHIFT
25
26     @staticmethod
27     def div(a, b):
28         return (a << FixedPoint.SHIFT) // b if b != 0 else 0
29
30     @staticmethod
31     def sqrt(fx):
32         if fx <= 0:
33             return 0
34         # Newton's method for fixed-point sqrt
35         x = fx
36         for _ in range(8):
37             x = (x + FixedPoint.div(fx, x)) >> 1
38         return x
39
40     @staticmethod
41     def sin_approx(fx):
42         # Map to [-pi, pi] range
43         two_pi = FixedPoint.from_float(2 * math.pi)
44         fx = fx % two_pi
45         if fx > FixedPoint.from_float(math.pi):
46             fx = fx - two_pi
47
48         # Polynomial approximation for sin(x)
49         x2 = FixedPoint.mul(fx, fx)
50         c1 = FixedPoint.from_float(1.0)
51         c3 = FixedPoint.from_float(-1.0/6.0)
52         c5 = FixedPoint.from_float(1.0/120.0)
53         c7 = FixedPoint.from_float(-1.0/5040.0)
54
55         term1 = FixedPoint.mul(x2, c3)
56         term2 = FixedPoint.mul(FixedPoint.mul(x2, x2), c5)
57         term3 = FixedPoint.mul(FixedPoint.mul(FixedPoint.mul(x2, x2), x2), c7)
58
59         return fx + FixedPoint.mul(fx, term1 + term2 + term3)
60
61     @staticmethod
62     def cos_approx(fx):
63         # cos(x) = sin(x + pi/2)
64         pi_2 = FixedPoint.from_float(math.pi / 2)
65         return FixedPoint.sin_approx(fx + pi_2)
66
67 # Fixed-point 3D vector
68 class Vec3FX:
69     def __init__(self, x, y, z):
70         self.x = x if isinstance(x, int) else FixedPoint.from_float(x)
71         self.y = y if isinstance(y, int) else FixedPoint.from_float(y)
72         self.z = z if isinstance(z, int) else FixedPoint.from_float(z)
73
74     def __add__(self, other):
75         return Vec3FX(self.x + other.x, self.y + other.y, self.z + other.z)
76
77     def __sub__(self, other):
78         return Vec3FX(self.x - other.x, self.y - other.y, self.z - other.z)
79
80     def __mul__(self, scalar):
81         if isinstance(scalar, Vec3FX):
82             return Vec3FX(

```

```

83         FixedPoint.mul(self.x, scalar.x),
84         FixedPoint.mul(self.y, scalar.y),
85         FixedPoint.mul(self.z, scalar.z)
86     )
87     return Vec3FX(
88         FixedPoint.mul(self.x, scalar),
89         FixedPoint.mul(self.y, scalar),
90         FixedPoint.mul(self.z, scalar)
91     )
92
93     def __rmul__(self, scalar):
94         return self * scalar
95
96     def dot(self, other):
97         return (FixedPoint.mul(self.x, other.x) +
98                 FixedPoint.mul(self.y, other.y) +
99                 FixedPoint.mul(self.z, other.z))
100
101     def cross(self, other):
102         return Vec3FX(
103             FixedPoint.mul(self.y, other.z) - FixedPoint.mul(self.z, other.y),
104             FixedPoint.mul(self.z, other.x) - FixedPoint.mul(self.x, other.z),
105             FixedPoint.mul(self.x, other.y) - FixedPoint.mul(self.y, other.x)
106         )
107
108     def norm_sq(self):
109         return self.dot(self)
110
111     def norm(self):
112         return FixedPoint.sqrt(self.norm_sq())
113
114     def normalize(self):
115         n = self.norm()
116         if n > 0:
117             inv_n = FixedPoint.div(FixedPoint.SCALE, n)
118             return self * inv_n
119         return self
120
121     def to_float_array(self):
122         return [FixedPoint.to_float(self.x),
123                 FixedPoint.to_float(self.y),
124                 FixedPoint.to_float(self.z)]
125
126     def copy(self):
127         return Vec3FX(self.x, self.y, self.z)
128
129
130 # --- Ray-object intersection ---
131 def intersect_sphere(origin, direction, center, radius):
132     # Mirrors HW (raytracer.sv stage C/D): half-discriminant form. Python's
133     # earlier full-b form was algebraically equivalent but rounded its
134     # extra multiplies differently, so at higher precision (Q14.13+) some
135     # sphere-tangent pixels disagreed with HW on hit/miss.
136     oc = origin - center
137     a = direction.dot(direction)
138     half_b = direction.dot(oc)
139     c = oc.dot(oc) - FixedPoint.mul(radius, radius)
140     disc = FixedPoint.mul(half_b, half_b) - FixedPoint.mul(a, c)
141     if disc < 0:
142         return None
143     sqrt_disc = FixedPoint.sqrt(disc)
144     t0 = FixedPoint.div(-half_b - sqrt_disc, a)
145     t1 = FixedPoint.div(-half_b + sqrt_disc, a)
146     t = t0 if t0 > 0 else t1
147     return t if t > 0 else None
148
149 def intersect_plane(origin, direction, point, normal):
150     denom = normal.dot(direction)
151     if abs(denom) < FixedPoint.from_float(1e-6):
152         return None
153     t = FixedPoint.div((point - origin).dot(normal), denom)
154     return t if t > 0 else None
155
156 def intersect_cone(origin, direction, apex, k_sq, height):

```

```

157 # Y-axis cone, apex on top (at apex), opens downward over y
158 # [apex.y - height, apex.y]. Surface  $(-xAx)^2 + (-zAz)^2 = k^2 \cdot (-yAy)^2$ .
159 # Mirrors the HW (raytracer.sv stage C/D): quadratic  $at^2 + 2bt + c = 0$  in
160 # the same half-discriminant form as sphere, plus a flat disc cap at
161 #  $y = \text{apex.y} - \text{height}$  with  $\text{radius}^2 = k^2 \cdot \text{height}^2$ . Returns (t, normal)
162 # for the closest forward hit on either side or cap, or (None, None).
163 delta = origin - apex
164 a = (FixedPoint.mul(direction.x, direction.x) +
165      FixedPoint.mul(direction.z, direction.z) -
166      FixedPoint.mul(k_sq, FixedPoint.mul(direction.y, direction.y)))
167 half_b = (FixedPoint.mul(delta.x, direction.x) +
168          FixedPoint.mul(delta.z, direction.z) -
169          FixedPoint.mul(k_sq, FixedPoint.mul(delta.y, direction.y)))
170 c = (FixedPoint.mul(delta.x, delta.x) +
171      FixedPoint.mul(delta.z, delta.z) -
172      FixedPoint.mul(k_sq, FixedPoint.mul(delta.y, delta.y)))
173
174 t_side = None
175 if a != 0:
176     disc = FixedPoint.mul(half_b, half_b) - FixedPoint.mul(a, c)
177     if disc >= 0:
178         sqrt_disc = FixedPoint.sqrt(disc)
179         t0 = FixedPoint.div(-half_b - sqrt_disc, a)
180         t1 = FixedPoint.div(-half_b + sqrt_disc, a)
181         # try both roots; accept the smallest forward-hit whose hit_y
182         # falls within the single-half slab
183         y_top = apex.y
184         y_bot = apex.y - height
185         for cand in sorted([t0, t1]):
186             if cand <= 0:
187                 continue
188             hit_y = origin.y + FixedPoint.mul(cand, direction.y)
189             if y_bot <= hit_y <= y_top:
190                 t_side = cand
191                 break
192
193 # Bottom disc cap: plane at  $y = \text{apex.y} - \text{height}$  with normal (0,-1,0).
194 t_cap = None
195 if direction.y != 0:
196     cand = FixedPoint.div(apex.y - height - origin.y, direction.y)
197     if cand > 0:
198         hit_x = origin.x + FixedPoint.mul(cand, direction.x)
199         hit_z = origin.z + FixedPoint.mul(cand, direction.z)
200         dx = hit_x - apex.x
201         dz = hit_z - apex.z
202         r_sq_hit = FixedPoint.mul(dx, dx) + FixedPoint.mul(dz, dz)
203         r_base_sq = FixedPoint.mul(k_sq, FixedPoint.mul(height, height))
204         if r_sq_hit <= r_base_sq:
205             t_cap = cand
206
207 candidates = [(t, kind) for t, kind in [(t_side, 'side'), (t_cap, 'cap')]
208         if t is not None]
209 if not candidates:
210     return None, None
211 t, kind = min(candidates, key=lambda x: x[0])
212
213 if kind == 'side':
214     # Outward gradient of  $(-xAx)^2 + (-zAz)^2 - k^2(-yAy)^2 = 0$ .
215     hit = origin + direction * t
216     n = Vec3FX(hit.x - apex.x,
217               -FixedPoint.mul(k_sq, hit.y - apex.y),
218               hit.z - apex.z)
219     return t, n.normalize()
220 else:
221     return t, Vec3FX(0.0, -1.0, 0.0)
222
223 # --- Scene definition ---
224 # Two-sphere scene matching the HW pipeline (raytracer_batch_mm defaults
225 # and the cocotb test). Sphere 0 = blue at original position; sphere 1 =
226 # red, 2.5 units to the +x of sphere 0.
227 SPHERE0_CENTER = Vec3FX(0.0, 1.0, 0.0)
228 SPHERE0_RADIUS = FixedPoint.from_float(1.0)
229 SPHERE0_COLOR = Vec3FX(0.2, 0.3, 0.8)
230

```



```

231 SPHERE1_CENTER = Vec3FX(2.5, 1.0, 0.0)
232 SPHERE1_RADIUS = FixedPoint.from_float(1.0)
233 SPHERE1_COLOR = Vec3FX(0.8, 0.3, 0.2)
234
235 # Back-compat aliases - kept so test_raytracer_batch.py's `from raytracer_fp
236 # import ...` still works for code that hasn't migrated to the two-sphere
237 # tuple form.
238 SPHERE_CENTER = SPHERE0_CENTER
239 SPHERE_RADIUS = SPHERE0_RADIUS
240 SPHERE_COLOR = SPHERE0_COLOR
241
242 PLANE_POINT = Vec3FX(0.0, 0.0, 0.0)
243 PLANE_NORMAL = Vec3FX(0.0, 1.0, 0.0)
244
245 PLANE_COLOR1 = Vec3FX(0.9, 0.9, 0.9)
246 PLANE_COLOR2 = Vec3FX(0.4, 0.4, 0.4)
247
248 # Y-axis cone, apex on top, opens downward. Defaults match the HW
249 # defaults in raytracer_batch_mm: apex at origin, k2=1.0 (45° half-angle),
250 # height=2.0m, mid-grey, matte (reflect_shift=0). CONE_ENABLED gates the
251 # cone out of the scene when False - set to False for sphere-only tests.
252 CONE_ENABLED = False
253 CONE_APEX = Vec3FX(0.0, 2.0, 4.0)
254 CONE_K_SQ = FixedPoint.from_float(1.0)
255 CONE_HEIGHT = FixedPoint.from_float(2.0)
256 CONE_COLOR = Vec3FX(0.5, 0.5, 0.5)
257 CONE_REFLECT_SHIFT = 0 # 0 = matte; 1 means (color * bright) >> shift, like spheres
258
259 AMBIENT = FixedPoint.from_float(0.15)
260 SPECULAR_EXP = 64
261 SPECULAR_STRENGTH = FixedPoint.from_float(0.6)
262
263 WIDTH, HEIGHT = 480, 360
264 ASPECT = WIDTH / HEIGHT
265
266 # Camera with a vertical image plane: same y for origin/target makes
267 # forward = (0, 0, 1), right = (-1, 0, 0), up = (0, 1, 0).
268 CAM_ORIGIN = Vec3FX(0.0, 1.5, -8.0)
269 CAM_TARGET = Vec3FX(0.0, 1.5, 0.0)
270
271 # Image-plane half-extents at distance 1 from the camera. Chosen directly
272 # (no tan/FOV), so both the Python reference and the SV constants are exact.
273 # HALF_H = HALF_W / ASPECT = 0.5 * 3/4.
274 HALF_W = 0.5
275 HALF_H = 0.375
276
277 # --- Camera basis ---
278 def camera_basis(origin, target):
279     forward = (target - origin).normalize()
280     y_axis = Vec3FX(0.0, 1.0, 0.0)
281     right = forward.cross(y_axis).normalize()
282     up = right.cross(forward)
283     return forward, right, up
284
285 # --- Checkerboard pattern for the plane ---
286 def plane_color(hit):
287     scale = 1
288     cx = int(FixedPoint.to_float(hit.x) * scale)
289     cz = int(FixedPoint.to_float(hit.z) * scale)
290     if (cx + cz) % 2 == 0:
291         return PLANE_COLOR1
292     return PLANE_COLOR2
293
294 # --- Trace a single ray ---
295 def trace(origin, direction, light_pos):
296     """Closest-t intersection across {sphere0, sphere1, plane, cone} then
297     shade. Mirrors the HW pipeline: raw (un-normalized) direction,
298     hit_type-driven self-skip on shadow rays, HW's >>FRAC_BITS shadow
299     offset. Cone is included only if CONE_ENABLED is True."""
300     t_s0 = intersect_sphere(origin, direction, SPHERE0_CENTER, SPHERE0_RADIUS)
301     t_s1 = intersect_sphere(origin, direction, SPHERE1_CENTER, SPHERE1_RADIUS)
302     t_pl = intersect_plane(origin, direction, PLANE_POINT, PLANE_NORMAL)
303     if CONE_ENABLED:
304         t_cn, cone_normal = intersect_cone(origin, direction,

```

```

305                                     CONE_APEX, CONE_K_SQ, CONE_HEIGHT)
306     else:
307         t_cn, cone_normal = None, None
308
309     hit_t = None
310     hit_type = None
311     if t_s0 is not None and (hit_t is None or t_s0 < hit_t):
312         hit_t = t_s0; hit_type = 'sphere0'
313     if t_s1 is not None and (hit_t is None or t_s1 < hit_t):
314         hit_t = t_s1; hit_type = 'sphere1'
315     if t_pl is not None and (hit_t is None or t_pl < hit_t):
316         hit_t = t_pl; hit_type = 'plane'
317     if t_cn is not None and (hit_t is None or t_cn < hit_t):
318         hit_t = t_cn; hit_type = 'cone'
319
320     if hit_type is None:
321         # Sky gradient
322         t_sky = FixedPoint.div(direction.y + FixedPoint.SCALE, FixedPoint.from_float(2))
323         inv_t = FixedPoint.SCALE - t_sky
324         sky1 = Vec3FX(1.0, 1.0, 1.0) * inv_t
325         sky2 = Vec3FX(0.5, 0.7, 1.0) * t_sky
326         return sky1 + sky2
327
328     hit = origin + direction * hit_t
329
330     if hit_type == 'sphere0':
331         normal = (hit - SPHERE0_CENTER).normalize()
332         color = SPHERE0_COLOR
333     elif hit_type == 'sphere1':
334         normal = (hit - SPHERE1_CENTER).normalize()
335         color = SPHERE1_COLOR
336     elif hit_type == 'cone':
337         normal = cone_normal
338         color = CONE_COLOR
339     else:
340         normal = PLANE_NORMAL.copy()
341         color = plane_color(hit)
342
343     # Lighting
344     to_light = light_pos - hit
345     light_dist = to_light.norm()
346     inv_light_dist = FixedPoint.div(FixedPoint.SCALE, light_dist) if light_dist > 0 else 0
347     light_dir = to_light * inv_light_dist
348
349     # Shadow check against spheres + cone, with self-skip via hit_type.
350     # Shadow origin uses HW's >>FRAC_BITS offset (= norm/1024 in Q10).
351     in_shadow = False
352     sh = Vec3FX(hit.x + (normal.x >> FixedPoint.SHIFT),
353                hit.y + (normal.y >> FixedPoint.SHIFT),
354                hit.z + (normal.z >> FixedPoint.SHIFT))
355     if hit_type != 'sphere0':
356         ts = intersect_sphere(sh, light_dir, SPHERE0_CENTER, SPHERE0_RADIUS)
357         if ts is not None and ts < light_dist:
358             in_shadow = True
359     if not in_shadow and hit_type != 'sphere1':
360         ts = intersect_sphere(sh, light_dir, SPHERE1_CENTER, SPHERE1_RADIUS)
361         if ts is not None and ts < light_dist:
362             in_shadow = True
363     if not in_shadow and CONE_ENABLED and hit_type != 'cone':
364         ts, _ = intersect_cone(sh, light_dir, CONE_APEX, CONE_K_SQ, CONE_HEIGHT)
365         if ts is not None and ts < light_dist:
366             in_shadow = True
367
368     if in_shadow:
369         return color * AMBIENT
370
371     # Diffuse
372     dot_prod = normal.dot(light_dir)
373     diff = dot_prod if dot_prod > 0 else 0
374
375     ambient_term = color * AMBIENT
376     diffuse_term = color * diff
377     result = ambient_term + diffuse_term
378

```

```

379     # Clamp to [0, SCALE]
380     result.x = max(0, min(FixedPoint.SCALE, result.x))
381     result.y = max(0, min(FixedPoint.SCALE, result.y))
382     result.z = max(0, min(FixedPoint.SCALE, result.z))
383
384     return result
385
386 # --- Render one frame ---
387 def render_frame(light_pos):
388     forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)
389
390     img = []
391
392     for y in range(HEIGHT):
393         row = []
394         py_flt = (1.0 - 2.0 * y / HEIGHT) * HALF_H
395         py = FixedPoint.from_float(py_flt)
396         for x in range(WIDTH):
397             px_flt = (2.0 * x / WIDTH - 1.0) * HALF_W
398             px = FixedPoint.from_float(px_flt)
399
400             direction = (forward +
401                          right * px +
402                          up * py)
403
404             result = trace(CAM_ORIGIN, direction, light_pos)
405
406             # Convert fixed-point to 8-bit RGB
407             r = int(max(0, min(255, FixedPoint.to_float(result.x) * 255)))
408             g = int(max(0, min(255, FixedPoint.to_float(result.y) * 255)))
409             b = int(max(0, min(255, FixedPoint.to_float(result.z) * 255)))
410             row.append((r, g, b))
411         img.append(row)
412
413     return img
414
415 # --- Animate ---
416 def main():
417     num_frames = 10
418     light_radius_float = 4.0
419     light_height = FixedPoint.from_float(4.0)
420     frames = []
421
422     for i in range(num_frames):
423         angle = 2 * math.pi * i / num_frames
424         cos_angle = FixedPoint.cos_approx(FixedPoint.from_float(angle))
425         sin_angle = FixedPoint.sin_approx(FixedPoint.from_float(angle))
426         light_radius_fx = FixedPoint.from_float(light_radius_float)
427
428         light_pos = Vec3FX(
429             FixedPoint.mul(light_radius_fx, cos_angle),
430             FixedPoint.to_float(light_height),
431             FixedPoint.mul(light_radius_fx, sin_angle),
432         )
433
434         print(f"Rendering frame {i + 1}/{num_frames}...")
435         pixels = render_frame(light_pos)
436
437         # Convert pixel list to PIL Image
438         img_array = [[pixels[y][x] for x in range(WIDTH)] for y in range(HEIGHT)]
439         pil_img = Image.new('RGB', (WIDTH, HEIGHT))
440         pil_img.putdata([pixel for row in img_array for pixel in row])
441         frames.append(pil_img)
442
443     out_path = "raytracer.gif"
444     frames[0].save(
445         out_path,
446         save_all=True,
447         append_images=frames[1:],
448         duration=80,
449         loop=0,
450     )
451     print(f"Saved animation to {out_path}")
452

```

```

453 if __name__ == "__main__":
454     main()

```

raytracer_reflect_ref.py

```

1  """HW-matching Python reference for the reflection-enabled raytracer.
2
3  Replicates `raytracer.sv`'s rtl_batch_pipe + spawn router + accumulator at
4  the bit level so the output should be byte-identical to the FPGA build,
5  modulo the same ~12-LSB tolerance the cocotb suite already accepts on
6  sphere-edge pixels (HW Q12 vs Python-fp drift through the sqrt/div
7  chain).
8
9  Reflection model = true geometric reflection:  $r = d - 2*(d \cdot n)$ .  $d$  is
10 the parent's raw (un-normalized) ray direction;  $n$  is the unit surface
11 normal.  $|r| = |d|$ , so the next bounce's stage C absorbs  $|r|^2$  into
12  $a = |raw|^2$  exactly like a primary ray. Origin = parent's hit point,
13 depth incremented, excluded = parent's hit_type so stage D's no_hit mask
14 blocks the matching sphere on re-entry.
15
16 Color accumulation =  $\sum_{i \text{ in chain}} (\text{to\_byte}(\text{stage\_K\_color\_i}) \gg$ 
17  $\text{total\_shift\_i})$ , saturated at 0xff per channel - matching the HW
18 accumulator's  $(\text{acc} + (\text{drop\_byte\_d1} \gg \text{meta.total\_shift})) \text{ RMW} + \text{sat\_byte}$ .
19
20 Run as a script: renders one frame at the cocotb test's default scene/
21 light and saves to ./raytracer_reflect.png. Tweak DEFAULTS at the
22 bottom to match the HW CSR programming you want to compare against.
23 """
24
25 import math
26 import sys
27 from PIL import Image
28
29 from raytracer_fp import (
30     FixedPoint, Vec3FX, camera_basis,
31     intersect_sphere, intersect_plane,
32     PLANE_POINT, PLANE_NORMAL,
33     PLANE_COLOR1, PLANE_COLOR2, AMBIENT,
34     CAM_ORIGIN, CAM_TARGET, WIDTH, HEIGHT, HALF_W, HALF_H,
35 )
36
37 # ---- Scene constants (must match HW localparams in raytracer.sv) ----
38 # Sphere positions/colors are the cocotb test's DEFAULT_SCENE - same as
39 # raytracer_batch_mm's defaults so a fresh boot of the FPGA renders the
40 # same scene this script does. Sphere 1's position is mutable (matches
41 # the HW SC1_*_NEXT CSRs); set_sphere1_position() can be called before
42 # rendering to relocate it for, e.g., the demo.py close-pair test.
43 SPHERE0_CENTER = Vec3FX(0.0, 1.0, 0.0)
44 SPHERE0_RADIUS = FixedPoint.from_float(1.0)
45 SPHERE0_COLOR = Vec3FX(0.2, 0.3, 0.8)
46
47 SPHERE1_CENTER = Vec3FX(2.5, 1.0, 0.0)
48 SPHERE1_RADIUS = FixedPoint.from_float(1.0)
49 SPHERE1_COLOR = Vec3FX(0.8, 0.3, 0.2)
50
51
52 def set_sphere1_position(x: float, y: float, z: float):
53     """Move sphere 1's center. Mirrors the HW SC1_X_NEXT/Y/Z CSRs which
54     are CSR-driven; sphere 0 stays hardcoded at (0, 1, 0)."""
55     global SPHERE1_CENTER
56     SPHERE1_CENTER = Vec3FX(x, y, z)
57
58 ONE = FixedPoint.SCALE # 4096 in Q12
59
60 # Hit-type encoding mirrors stage D's 2-bit field.
61 HIT_SKY = 0b00
62 HIT_SPHERE0 = 0b01
63 HIT_SPHERE1 = 0b10
64 HIT_PLANE = 0b11
65
66 # Sun-billboard color: rtl_hit_resolve overrides the sky gradient with
67 # this when the ray direction aligns with the light to within  $\cos^2$ 

```

```

68 # 63/64 ( 7.2° half-angle). Constants match raytracer.sv's localparams
69 # bit-for-bit (Verilog integer division truncates, so (95*ONE)/100 gives
70 # 7782 in Q14.13, not the rounded 7782.4).
71 SUN_COL = Vec3FX(ONE, (95 * ONE) // 100, (55 * ONE) // 100)
72
73
74 def _sun_active(direction, light_pos):
75     """Mirror rtl_hit_resolve's sun-billboard test (capture at t=15):
76     sun_active = (sun_dot > 0) & (sun_dot > K_const * a)
77     where sun_dot = direction · light_pos, a = |direction|², and
78     K_const = |light_pos|² · 63/64. HW treats light_pos as a direction-
79     from-origin (no parallax correction at the hit point), so this uses
80     light_pos directly rather than (light_pos - origin)."""
81     sun_dot = direction.dot(light_pos)
82     if sun_dot < 0:
83         return False
84     lx_sq = FixedPoint.mul(light_pos.x, light_pos.x)
85     ly_sq = FixedPoint.mul(light_pos.y, light_pos.y)
86     lz_sq = FixedPoint.mul(light_pos.z, light_pos.z)
87     light_sq = lx_sq + ly_sq + lz_sq
88     K_const = light_sq - (light_sq >> 6) # |L|² · 63/64
89     a = (FixedPoint.mul(direction.x, direction.x) +
90         FixedPoint.mul(direction.y, direction.y) +
91         FixedPoint.mul(direction.z, direction.z))
92     sun_dot_sq = FixedPoint.mul(sun_dot, sun_dot)
93     return sun_dot_sq >= FixedPoint.mul(K_const, a)
94
95
96 # ---- HW-matching helpers -----
97
98 def to_byte(v_fp):
99     """Mirror to_byte() in raytracer.sv: saturate v >= ONE to 0xff,
100     otherwise return the 8 bits immediately below the integer bit
101     (= map [0, ONE] to [0, 256)).
102
103     HW guarantees v is clamped to [0, ONE] by stage K's clamp01, so the
104     saturate branch only fires at exactly 1.0."""
105     if v_fp >= ONE:
106         return 0xff
107     if v_fp < 0:
108         return 0
109     return (v_fp >> (FixedPoint.SHIFT - 8)) & 0xff
110
111
112 def sat_byte(v_16):
113     """Mirror sat_byte() in raytracer.sv: 16-bit accumulator value
114     clamped to 8-bit. high-byte non-zero -> 0xff."""
115     if v_16 > 0xff:
116         return 0xff
117     if v_16 < 0:
118         return 0
119     return v_16 & 0xff
120
121
122 def shaded_color(origin, direction, light_pos, excluded):
123     """Mirror stage K's output: closest-t hit -> shaded color in
124     fixed-point [0, ONE], plus the auxiliary data the spawn router
125     needs (hit_type, hit point, surface normal).
126
127     Matches the HW exactly:
128     - direction is raw (un-normalized); stages C/D absorb |raw| in t.
129     - sphere intersect uses the half-discriminant form.
130     - meta.excluded ORs into the matching sphere's no_hit (mirrors
131       stage D's mask in M2c).
132     - plane number = -origin.y (mirrors stage D's variable plane number).
133     - shadow ray origin = hit + (normal >> FRAC_BITS), per-component
134       (mirrors stage I's shadow offset).
135     - shadow self-skip: stage I ORs (hit_type == matching sphere) into
136       shadow_no_hit. Same here.
137     - shading: sky branch passes col through; in_shadow -> col * AMBIENT;
138       else col*AMBIENT + col*max(0, dot(norm, ldir)). All clamped to
139       [0, ONE].
140
141     Returns (color_vec, hit_type, hit_point, surface_normal). hit_type

```

```

142 is HIT_* enum. hit_point/normal are None for sky.
143 """
144 skip_s0 = (excluded == HIT_SPHERE0)
145 skip_s1 = (excluded == HIT_SPHERE1)
146
147 t_s0 = None if skip_s0 else intersect_sphere(
148     origin, direction, SPHERE0_CENTER, SPHERE0_RADIUS)
149 t_s1 = None if skip_s1 else intersect_sphere(
150     origin, direction, SPHERE1_CENTER, SPHERE1_RADIUS)
151 # Plane denom uses direction.y; numer = dot(plane_origin - origin, n)
152 # = -origin.y. intersect_plane in raytracer_fp.py already takes a
153 # plane point + normal, so passing PLANE_POINT and PLANE_NORMAL is
154 # equivalent (the math reduces to t = -origin.y / dir.y).
155 t_pl = intersect_plane(origin, direction, PLANE_POINT, PLANE_NORMAL)
156
157 hit_t = None
158 hit_type = HIT_SKY
159 if t_s0 is not None and (hit_t is None or t_s0 < hit_t):
160     hit_t = t_s0; hit_type = HIT_SPHERE0
161 if t_s1 is not None and (hit_t is None or t_s1 < hit_t):
162     hit_t = t_s1; hit_type = HIT_SPHERE1
163 if t_pl is not None and (hit_t is None or t_pl < hit_t):
164     hit_t = t_pl; hit_type = HIT_PLANE
165
166 # Sky branch: gradient based on direction.y (HW does the same in
167 # stage E's hit_type==00 path). The sun-billboard override at t=16
168 # replaces the gradient with SUN_COL when the ray points within
169 # ~7.2° of the light direction.
170 if hit_type == HIT_SKY:
171     if _sun_active(direction, light_pos):
172         return SUN_COL.copy(), hit_type, None, None
173     t_sky = FixedPoint.div(direction.y + ONE, FixedPoint.from_float(2))
174     inv_t = ONE - t_sky
175     sky1 = Vec3FX(1.0, 1.0, 1.0) * inv_t
176     sky2 = Vec3FX(0.5, 0.7, 1.0) * t_sky
177     col = sky1 + sky2
178     # Sky branch in stage K passes col through clamp01 (no-op since
179     # sky gradient is already in [0, ONE]).
180     col.x = max(0, min(ONE, col.x))
181     col.y = max(0, min(ONE, col.y))
182     col.z = max(0, min(ONE, col.z))
183     return col, hit_type, None, None
184
185 # Hit point + base color + normal.
186 hit = origin + direction * hit_t
187 if hit_type == HIT_SPHERE0:
188     normal = (hit - SPHERE0_CENTER).normalize()
189     color = SPHERE0_COLOR
190 elif hit_type == HIT_SPHERE1:
191     normal = (hit - SPHERE1_CENTER).normalize()
192     color = SPHERE1_COLOR
193 else: # plane
194     normal = PLANE_NORMAL.copy()
195     cx = int(FixedPoint.to_float(hit.x))
196     cz = int(FixedPoint.to_float(hit.z))
197     color = PLANE_COLOR1 if (cx + cz) % 2 == 0 else PLANE_COLOR2
198
199 # Light direction (normalized).
200 to_light = light_pos - hit
201 light_dist = to_light.norm()
202 inv_ld = FixedPoint.div(ONE, light_dist) if light_dist > 0 else 0
203 light_dir = to_light * inv_ld
204
205 # Shadow: HW's hit_type-driven self-skip + per-component norm/1024
206 # offset for the shadow origin.
207 sh = Vec3FX(hit.x + (normal.x >> FixedPoint.SHIFT),
208             hit.y + (normal.y >> FixedPoint.SHIFT),
209             hit.z + (normal.z >> FixedPoint.SHIFT))
210 in_shadow = False
211 if hit_type != HIT_SPHERE0:
212     ts = intersect_sphere(sh, light_dir, SPHERE0_CENTER, SPHERE0_RADIUS)
213     if ts is not None and ts < light_dist:
214         in_shadow = True
215 if not in_shadow and hit_type != HIT_SPHERE1:

```

```

216         ts = intersect_sphere(sh, light_dir, SPHERE1_CENTER, SPHERE1_RADIUS)
217         if ts is not None and ts < light_dist:
218             in_shadow = True
219
220     if in_shadow:
221         r = color * AMBIENT
222     else:
223         dot_prod = normal.dot(light_dir)
224         diff = dot_prod if dot_prod > 0 else 0
225         r = color * AMBIENT + color * diff
226
227     # clamp01 (matches stage K's tick=14 saturate).
228     r.x = max(0, min(ONE, r.x))
229     r.y = max(0, min(ONE, r.y))
230     r.z = max(0, min(ONE, r.z))
231     return r, hit_type, hit, normal
232
233
234 def trace_chain(origin, direction, light_pos,
235               max_depth, reflect_shifts,
236               depth=0, excluded=HIT_SKY, total_shift=0):
237     """Mirror the HW spawn-loop: shade this ray, accumulate weighted
238     contribution, and if it hit a sphere AND depth < max_depth, recurse
239     with origin=hit, direction=r-d-2*(d·n)*n, excluded=hit_type,
240     total_shift+= reflect_shifts[hit_type].
241
242     Returns three 16-bit accumulator-style ints (one per channel) that
243     the caller saturates with sat_byte.
244     """
245     col, hit_type, hit_pos, normal = shaded_color(
246         origin, direction, light_pos, excluded)
247
248     # weighted contribution = to_byte(col) >> total_shift; HW also caps
249     # the shift at 8 (anything past saturates to 0).
250     if total_shift >= 8:
251         wr = wg = wb = 0
252     else:
253         wr = to_byte(col.x) >> total_shift
254         wg = to_byte(col.y) >> total_shift
255         wb = to_byte(col.z) >> total_shift
256
257     # Spawn check: HW spawn_decision = (depth < max_depth) &&
258     # (hit_type == sphere0 || sphere1).
259     if depth < max_depth and hit_type in (HIT_SPHERE0, HIT_SPHERE1):
260         child_shift = reflect_shifts[0] if hit_type == HIT_SPHERE0 \
261             else reflect_shifts[1]
262         # r = d - 2*(d·n)*n. d is non-unit (raw), n is unit; result is
263         # |d|*r - same magnitude as d, so stage C's a=|raw|^2 absorbs it.
264         dot_dn = direction.dot(normal)
265         two_dot = dot_dn << 1
266         refl_dir = direction - normal * two_dot
267         cr, cg, cb = trace_chain(
268             hit_pos, refl_dir, light_pos,
269             max_depth, reflect_shifts,
270             depth=depth + 1,
271             excluded=hit_type,
272             total_shift=total_shift + child_shift,
273         )
274         return wr + cr, wg + cg, wb + cb
275
276     return wr, wg, wb
277
278
279 # ---- Per-pixel render -----
280
281 # Mirror the cocotb test's pixel_to_screen exactly so px/py drop into
282 # the same Q-format coords the HW computes in stage A.
283 _RECIP_SHIFT = 24
284 _HALF_W_Q = (1 << FixedPoint.SHIFT) // 2
285 _HALF_H_Q = (_HALF_W_Q * HEIGHT) // WIDTH
286 _INV_W_Q = ((2 * _HALF_W_Q) << _RECIP_SHIFT) // WIDTH
287 _INV_H_Q = ((2 * _HALF_H_Q) << _RECIP_SHIFT) // HEIGHT
288
289

```

```

290 def pixel_to_screen(x, y):
291     px_c = ((x * _INV_W_Q) >> _RECIP_SHIFT) - _HALF_W_Q
292     py_c = _HALF_H_Q - ((y * _INV_H_Q) >> _RECIP_SHIFT)
293     return px_c, py_c
294
295
296 def camera_basis_from_euler(yaw_rad, pitch_rad):
297     """Float-domain basis from yaw + pitch, matching the Euler convention
298     in SW/desktop/camera.py:camera_basis_from_euler. yaw=0, pitch=0 yields
299     right=(-1,0,0), up=(0,1,0), fwd=(0,0,1) - the legacy hardcoded scene.
300     Returns three Vec3FX (right.y is hardcoded 0 - no roll)."""
301     cy, sy = math.cos(yaw_rad), math.sin(yaw_rad)
302     cp, sp = math.cos(pitch_rad), math.sin(pitch_rad)
303     right = Vec3FX(-cy, 0.0, sy)
304     up = Vec3FX(-sy * sp, cp, -cy * sp)
305     forward = Vec3FX(sy * cp, sp, cy * cp)
306     return forward, right, up
307
308
309 def render_pixel(x, y, light_pos, max_depth, reflect_shifts,
310                 cam_origin, forward, right, up):
311     px_fp, py_fp = pixel_to_screen(x, y)
312     raw = forward + right * px_fp + up * py_fp
313     ar, ag, ab = trace_chain(cam_origin, raw, light_pos,
314                             max_depth, reflect_shifts)
315     return sat_byte(ar), sat_byte(ag), sat_byte(ab)
316
317
318 def render_frame(light_pos, max_depth=1, reflect_shifts=(2, 1),
319                 cam_origin=None, yaw_rad=0.0, pitch_rad=0.0):
320     """Full WIDTH x HEIGHT render. Returns a PIL Image.
321
322     cam_origin defaults to the legacy CAM_ORIGIN = (0, 1.5, -8) when
323     None. yaw/pitch default to 0 - identity basis matching the legacy
324     look-at-target setup. To match a runtime HW frame, pass cam_origin
325     + the same yaw/pitch the HW received via its CSRs."""
326     if cam_origin is None:
327         cam_origin = CAM_ORIGIN
328     forward, right, up = camera_basis_from_euler(yaw_rad, pitch_rad)
329     img = Image.new("RGB", (WIDTH, HEIGHT))
330     pixels = []
331     for y in range(HEIGHT):
332         for x in range(WIDTH):
333             pixels.append(render_pixel(x, y, light_pos, max_depth,
334                                       reflect_shifts, cam_origin,
335                                       forward, right, up))
336
337             if y % 30 == 0:
338                 print(f" row {y}/{HEIGHT}", file=sys.stderr)
339     img.putdata(pixels)
340     return img
341
342 # ---- Defaults match raytracer_batch_mm's reset state -----
343
344 DEFAULT_MAX_DEPTH = 3
345 DEFAULT_REFLECT_SHIFT_0 = 2 # 25% contribution from sphere 0 reflection
346 DEFAULT_REFLECT_SHIFT_1 = 1 # 50% from sphere 1
347 DEFAULT_LIGHT = Vec3FX(4.0, 4.0, 0.0)
348
349
350 def main():
351     """Render one frame at the HW reset defaults + center light."""
352     import argparse
353     p = argparse.ArgumentParser(
354         description="HW-matching reflection raytracer reference."
355     )
356     p.add_argument("--max-depth", type=int, default=DEFAULT_MAX_DEPTH,
357                   help="recursion cap (0=no reflection; default %(default)d)")
358     p.add_argument("--shift0", type=int, default=DEFAULT_REFLECT_SHIFT_0,
359                   help="sphere 0 reflect_shift (default %(default)d)")
360     p.add_argument("--shift1", type=int, default=DEFAULT_REFLECT_SHIFT_1,
361                   help="sphere 1 reflect_shift (default %(default)d)")
362     p.add_argument("--light-x", type=float, default=4.0)
363     p.add_argument("--light-y", type=float, default=4.0)

```



```

364 p.add_argument("--light-z", type=float, default=0.0)
365 p.add_argument("--sphere1", nargs=3, type=float, metavar=("X", "Y", "Z"),
366                 help="sphere 1 (red) center (default 2.5 1.0 0.0)")
367 # Runtime camera (matches the HW CSR-driven camera state). Defaults
368 # reproduce the legacy hardcoded scene exactly. cam_y is locked at 1.5
369 # in HW so it has no CLI knob.
370 p.add_argument("--cam-x", type=float, default=0.0,
371                 help="camera origin x (default 0.0)")
372 p.add_argument("--cam-z", type=float, default=-8.0,
373                 help="camera origin z (default -8.0)")
374 p.add_argument("--yaw", type=float, default=0.0,
375                 help="camera yaw in degrees (default 0)")
376 p.add_argument("--pitch", type=float, default=0.0,
377                 help="camera pitch in degrees (default 0)")
378 p.add_argument("--out", default="raytracer_reflect.png",
379                 help="output path (default %(default)s)")
380 p.add_argument("--gif-frames", type=int, default=0,
381                 help="if >0, render an orbiting-light GIF with N frames")
382 args = p.parse_args()
383
384 if args.sphere1 is not None:
385     set_sphere1_position(*args.sphere1)
386
387 light = Vec3FX(args.light_x, args.light_y, args.light_z)
388 shifts = (args.shift0, args.shift1)
389 # cam_y locked at 1.5 in HW (no Y-translation per camera plan).
390 cam_origin = Vec3FX(args.cam_x, 1.5, args.cam_z)
391 yaw_rad = math.radians(args.yaw)
392 pitch_rad = math.radians(args.pitch)
393
394 if args.gif_frames > 0:
395     light_radius = math.sqrt(args.light_x**2 + args.light_z**2) or 4.0
396     frames = []
397     for i in range(args.gif_frames):
398         angle = 2 * math.pi * i / args.gif_frames
399         lp = Vec3FX(
400             light_radius * math.cos(angle),
401             args.light_y,
402             light_radius * math.sin(angle),
403         )
404         print(f"frame {i+1}/{args.gif_frames} (light angle "
405               f"{math.degrees(angle):.1f}°)", file=sys.stderr)
406         frames.append(render_frame(lp, args.max_depth, shifts,
407                                   cam_origin=cam_origin,
408                                   yaw_rad=yaw_rad,
409                                   pitch_rad=pitch_rad))
410     out_path = args.out if args.out.endswith(".gif") else \
411         args.out.rsplit(".", 1)[0] + ".gif"
412     frames[0].save(out_path, save_all=True,
413                   append_images=frames[1:], duration=80, loop=0)
414     print(f"saved {out_path}", file=sys.stderr)
415 else:
416     print(f"rendering {WIDTH}x{HEIGHT} at max_depth={args.max_depth}, "
417           f"shifts=({args.shift0},{args.shift1}), "
418           f"cam=({args.cam_x:.2f}, 1.50, {args.cam_z:.2f}), "
419           f"yaw={args.yaw:.1f}° pitch={args.pitch:.1f}°",
420           file=sys.stderr)
421     img = render_frame(light, args.max_depth, shifts,
422                       cam_origin=cam_origin,
423                       yaw_rad=yaw_rad,
424                       pitch_rad=pitch_rad)
425     img.save(args.out)
426     print(f"saved {args.out}", file=sys.stderr)
427
428
429 if __name__ == "__main__":
430     main()

```

test_raytracer_batch.py

```

1 import cocotb
2 from cocotb.clock import Clock

```

```

3 from cocotb.triggers import RisingEdge
4 import sys, os, math
5
6 sys.path.insert(0, os.path.dirname(__file__))
7 from raytracer_fp import (FixedPoint, Vec3FX, camera_basis,
8                           intersect_sphere, intersect_plane,
9                           PLANE_POINT, PLANE_NORMAL,
10                          PLANE_COLOR1, PLANE_COLOR2, AMBIENT,
11                          CAM_ORIGIN, CAM_TARGET, WIDTH, HEIGHT,
12                          HALF_W, HALF_H)
13
14 WORD = 27
15 N = 120 # must match the BATCH parameter in rtl_batch_pipe
16 TOLERANCE = 12 # 8-bit channel LSBs; covers py-model vs HW-fp drift
17 # on sphere-edge pixels
18
19 # Two-sphere scene. Sphere 0 keeps the historical center (matches the old
20 # 1-sphere golden); sphere 1 sits 2.5 units to the right. Colors picked to
21 # visually distinguish them. Same defaults as raytracer_batch_mm hardcodes.
22 SPHERE0_CENTER = Vec3FX(0.0, 1.0, 0.0)
23 SPHERE0_RADIUS = FixedPoint.from_float(1.0)
24 SPHERE0_COLOR = Vec3FX(0.2, 0.3, 0.8)
25
26 SPHERE1_CENTER = Vec3FX(2.5, 1.0, 0.0)
27 SPHERE1_RADIUS = FixedPoint.from_float(1.0)
28 SPHERE1_COLOR = Vec3FX(0.8, 0.3, 0.2)
29
30 # Sphere colors are baked into stage E as localparams (SCOLO_*, SCOL1_*) so
31 # they are NOT runtime-configurable. Tests that change sphere positions/radii
32 # via the dut still get the same colors back from HW.
33 def _scene(sc0=SPHERE0_CENTER, r0=SPHERE0_RADIUS,
34           sc1=SPHERE1_CENTER, r1=SPHERE1_RADIUS):
35     return (sc0, r0, sc1, r1)
36
37 DEFAULT_SCENE = _scene()
38
39
40 def trace_py(origin, direction, light_pos, scene=DEFAULT_SCENE):
41     """Python reference trace for the two-sphere pipeline.
42
43     Mirrors HW: closest-t winner across {sphere0, sphere1, plane}; raw
44     (un-normalized) direction; shadow ray with HW's >>FRAC_BITS offset and
45     self-skip on the matching sphere via hit_type."""
46     sc0, r0, sc1, r1 = scene
47     t_s0 = intersect_sphere(origin, direction, sc0, r0)
48     t_s1 = intersect_sphere(origin, direction, sc1, r1)
49     t_pl = intersect_plane(origin, direction, PLANE_POINT, PLANE_NORMAL)
50
51     hit_t = None
52     hit_type = None
53     if t_s0 is not None and (hit_t is None or t_s0 < hit_t):
54         hit_t = t_s0; hit_type = 'sphere0'
55     if t_s1 is not None and (hit_t is None or t_s1 < hit_t):
56         hit_t = t_s1; hit_type = 'sphere1'
57     if t_pl is not None and (hit_t is None or t_pl < hit_t):
58         hit_t = t_pl; hit_type = 'plane'
59
60     if hit_type is None:
61         SCALE = FixedPoint.SCALE
62         t_sky = FixedPoint.div(direction.y + SCALE, FixedPoint.from_float(2))
63         inv_t = SCALE - t_sky
64         sky1 = Vec3FX(1.0, 1.0, 1.0) * inv_t
65         sky2 = Vec3FX(0.5, 0.7, 1.0) * t_sky
66         r = sky1 + sky2
67         return (r.x, r.y, r.z)
68
69     hit = origin + direction * hit_t
70     if hit_type == 'sphere0':
71         normal = (hit - sc0).normalize()
72         color = SPHERE0_COLOR
73     elif hit_type == 'sphere1':
74         normal = (hit - sc1).normalize()
75         color = SPHERE1_COLOR
76     else:

```

```

77     normal = PLANE_NORMAL.copy()
78     cx = int(FixedPoint.to_float(hit.x))
79     cz = int(FixedPoint.to_float(hit.z))
80     color = PLANE_COLOR1 if (cx + cz) % 2 == 0 else PLANE_COLOR2
81
82     to_light = light_pos - hit
83     light_dist = to_light.norm()
84     inv_ld = FixedPoint.div(FixedPoint.SCALE, light_dist) if light_dist > 0 else 0
85     light_dir = to_light * inv_ld
86
87     # Shadow ray. HW computes shad_origin = hit + (norm >>> FRAC_BITS),
88     # which is the per-component arithmetic right shift in fixed point.
89     in_shadow = False
90     if hit_type != 'sky':
91         sh = Vec3FX(hit.x + (normal.x >> FixedPoint.SHIFT),
92                     hit.y + (normal.y >> FixedPoint.SHIFT),
93                     hit.z + (normal.z >> FixedPoint.SHIFT))
94         if hit_type != 'sphere0':
95             ts = intersect_sphere(sh, light_dir, sc0, r0)
96             if ts is not None and ts < light_dist:
97                 in_shadow = True
98         if not in_shadow and hit_type != 'sphere1':
99             ts = intersect_sphere(sh, light_dir, sc1, r1)
100             if ts is not None and ts < light_dist:
101                 in_shadow = True
102
103     if in_shadow:
104         r = color * AMBIENT
105     else:
106         dot_prod = normal.dot(light_dir)
107         diff = dot_prod if dot_prod > 0 else 0
108         r = color * AMBIENT + color * diff
109
110     r.x = max(0, min(FixedPoint.SCALE, r.x))
111     r.y = max(0, min(FixedPoint.SCALE, r.y))
112     r.z = max(0, min(FixedPoint.SCALE, r.z))
113     return (r.x, r.y, r.z)
114
115 def to_unsigned(val):
116     if val < 0:
117         val = val + (1 << WORD)
118     return val
119
120 def pack_rgb(color):
121     """Pack raytracer_pkg::rgb_t {r, g, b} (MSB first) into a 3*WORD-bit int."""
122     return (
123         (to_unsigned(color.x) << (2 * WORD)) |
124         (to_unsigned(color.y) << (1 * WORD)) |
125         (to_unsigned(color.z))
126     )
127
128 # sphere_t = {x, y, z, r_sq, color: rgb_t, reflect_shift[4:0]}.
129 # Width = 4*WORD + 3*WORD + 5 = 7*WORD + 5 = 194 bits at WORD=27.
130 SPHERE_BITS = 7 * WORD + 5
131
132 def pack_sphere(sc, r_sq, color=Vec3FX(0.5, 0.5, 0.5), reflect_shift=0):
133     val = 0
134     val = (val << WORD) | to_unsigned(sc.x)
135     val = (val << WORD) | to_unsigned(sc.y)
136     val = (val << WORD) | to_unsigned(sc.z)
137     val = (val << WORD) | to_unsigned(r_sq)
138     val = (val << WORD) | to_unsigned(color.x)
139     val = (val << WORD) | to_unsigned(color.y)
140     val = (val << WORD) | to_unsigned(color.z)
141     val = (val << 5) | (int(reflect_shift) & 0x1f)
142     return val
143
144 # cone_t = {apex_x, apex_y, apex_z, k_sq, height, norm_factor, color, reflect_shift[4:0]}.
145 # Width = 6*WORD + 3*WORD + 5 = 9*WORD + 5 = 248 bits at WORD=27.
146 CONE_BITS = 9 * WORD + 5
147
148 def pack_cone(apex, k_sq, height, norm_factor,
149               color=Vec3FX(0.5, 0.5, 0.5), reflect_shift=0):
150     val = 0

```

```

151     val = (val << WORD) | to_unsigned(apex.x)
152     val = (val << WORD) | to_unsigned(apex.y)
153     val = (val << WORD) | to_unsigned(apex.z)
154     val = (val << WORD) | to_unsigned(k_sq)
155     val = (val << WORD) | to_unsigned(height)
156     val = (val << WORD) | to_unsigned(norm_factor)
157     val = (val << WORD) | to_unsigned(color.x)
158     val = (val << WORD) | to_unsigned(color.y)
159     val = (val << WORD) | to_unsigned(color.z)
160     val = (val << 5)      | (int(reflect_shift) & 0x1f)
161     return val
162
163 # floor_t = {pc1: rgb_t, pc2: rgb_t, mode[1:0], lim_x, lim_z}.
164 # Width = 6*WORD + 2 + 2*WORD = 8*WORD + 2 = 218 bits at WORD=27.
165 FLOOR_BITS = 8 * WORD + 2
166
167 def pack_floor(pc1, pc2, mode=0, lim_x=None, lim_z=None):
168     if lim_x is None:
169         lim_x = (1 << (WORD - 1)) - 1
170     if lim_z is None:
171         lim_z = (1 << (WORD - 1)) - 1
172     val = 0
173     for v in [pc1.x, pc1.y, pc1.z, pc2.x, pc2.y, pc2.z]:
174         val = (val << WORD) | to_unsigned(v)
175     val = (val << 2)      | (int(mode) & 0x3)
176     val = (val << WORD) | to_unsigned(lim_x)
177     val = (val << WORD) | to_unsigned(lim_z)
178     return val
179
180 # scene_t = {sphere_t s0, sphere_t s1, cone_t cone, floor_t floor}.
181 SCENE_BITS = 2 * SPHERE_BITS + CONE_BITS + FLOOR_BITS
182
183 def pack_scene(s0_pack, s1_pack, cone_pack, floor_pack):
184     val = s0_pack
185     val = (val << SPHERE_BITS) | s1_pack
186     val = (val << CONE_BITS)   | cone_pack
187     val = (val << FLOOR_BITS)  | floor_pack
188     return val
189
190
191 class SceneState:
192     """Mutable scene mirror that knows how to push itself into dut.scene.
193     Defaults match the wrapper's reset block so the cocotb top renders the
194     same picture as a hardware boot."""
195     def __init__(self):
196         # Sphere 0 - blue, 25% reflective.
197         self.s0_center = SPHERE0_CENTER
198         self.s0_r_sq   = FixedPoint.mul(SPHERE0_RADIUS, SPHERE0_RADIUS)
199         self.s0_color   = SPHERE0_COLOR
200         self.s0_reflect_shift = 2
201         # Sphere 1 - red, 50% reflective.
202         self.s1_center = SPHERE1_CENTER
203         self.s1_r_sq   = FixedPoint.mul(SPHERE1_RADIUS, SPHERE1_RADIUS)
204         self.s1_color   = SPHERE1_COLOR
205         self.s1_reflect_shift = 1
206         # Cone - disabled (height=0), matte, mid-grey, k²=1 (45°).
207         self.cone_apex = Vec3FX(0.0, 0.0, 0.0)
208         self.cone_k_sq = FixedPoint.from_float(1.0)
209         self.cone_height = FixedPoint.from_float(0.0)
210         self.cone_norm_factor = FixedPoint.from_float(math.sqrt(2.0))
211         self.cone_color = Vec3FX(0.5, 0.5, 0.5)
212         self.cone_reflect_shift = 0
213         # Floor - legacy checker.
214         self.floor_pc1 = Vec3FX(0.9, 0.9, 0.9)
215         self.floor_pc2 = Vec3FX(0.4, 0.4, 0.4)
216         self.floor_mode = 0
217         self.floor_lim_x = None # → max-positive in pack_floor
218         self.floor_lim_z = None
219
220     def pack(self):
221         s0 = pack_sphere(self.s0_center, self.s0_r_sq, self.s0_color, self.s0_reflect_shift)
222         s1 = pack_sphere(self.s1_center, self.s1_r_sq, self.s1_color, self.s1_reflect_shift)
223         c  = pack_cone(self.cone_apex, self.cone_k_sq, self.cone_height,
224                       self.cone_norm_factor, self.cone_color, self.cone_reflect_shift)

```

```

225         fl = pack_floor(self.floor_pc1, self.floor_pc2, self.floor_mode,
226                         self.floor_lim_x, self.floor_lim_z)
227         return pack_scene(s0, s1, c, fl)
228
229     def drive(self, dut):
230         dut.scene.value = self.pack()
231
232     def from_scene_tuple(self, scene_tuple):
233         """Update sphere geometry from a (sc0, r0, sc1, r1) tuple. Colors and
234         reflect_shift unchanged."""
235         sc0, r0, sc1, r1 = scene_tuple
236         self.s0_center = sc0
237         self.s0_r_sq = FixedPoint.mul(r0, r0)
238         self.s1_center = sc1
239         self.s1_r_sq = FixedPoint.mul(r1, r1)
240         return self
241
242     def rgb8(fp_val):
243         return int(max(0, min(255, FixedPoint.to_float(fp_val) * 255)))
244
245     # Mirror stage A's INV_W/INV_H multiply-shift exactly so the golden's raw
246     # values match HW bit-for-bit. Float-based pixel-to-screen drifts by 1 LSB
247     # at some pixels, which used to be absorbed by stage B's normalize but now
248     # tips disc through zero at the sphere silhouette.
249     _RECIP_SHIFT = 24
250     _HALF_W_Q = (1 << FixedPoint.SHIFT) // 2
251     _HALF_H_Q = (_HALF_W_Q * HEIGHT) // WIDTH
252     _INV_W_Q = ((2 * _HALF_W_Q) << _RECIP_SHIFT) // WIDTH
253     _INV_H_Q = ((2 * _HALF_H_Q) << _RECIP_SHIFT) // HEIGHT
254
255     def pixel_to_screen(x, y):
256         """Integer pixel coords -> Q-format screen-space px, py."""
257         px_c = ((x * _INV_W_Q) >> _RECIP_SHIFT) - _HALF_W_Q
258         py_c = _HALF_H_Q - ((y * _INV_H_Q) >> _RECIP_SHIFT)
259         return px_c, py_c
260
261     def render_pixel_py(px_fp, py_fp, light_pos, scene=DEFAULT_SCENE):
262         """Python reference: compute direction from px/py, call trace_py.
263
264         Stage B (camera-ray normalize) was dropped in HW: stages downstream
265         operate on un-normalized raw directly. The sphere/plane quadratics
266         absorb the |raw| scaling so hit points are invariant, and dropping
267         normalize here keeps the Q10 rounding bit-identical at the
268         silhouette edge."""
269         forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)
270         raw = forward + right * px_fp + up * py_fp
271         return trace_py(CAM_ORIGIN, raw, light_pos, scene)
272
273     async def reset(dut):
274         dut.rst_n.value = 0
275         await RisingEdge(dut.clk)
276         await RisingEdge(dut.clk)
277         dut.rst_n.value = 1
278         await RisingEdge(dut.clk)
279
280
281     def k_const_from_light(light):
282         """Replicate the wrapper's K_const compute for cocotb. The wrapper
283         runs a 3-phase u_l_sq + sum + (x - x>>>6) on light*_reg; in
284         test we drive rtl_batch_pipe directly so the test has to feed the
285         same wire. K_const = (lx^2 + ly^2 + lz^2) · 63/64, all in Q FRAC_BITS."""
286         lx_sq = FixedPoint.mul(light.x, light.x)
287         ly_sq = FixedPoint.mul(light.y, light.y)
288         lz_sq = FixedPoint.mul(light.z, light.z)
289         light_sq = lx_sq + ly_sq + lz_sq
290         return light_sq - (light_sq >> 6)
291
292
293     def drive_default_camera(dut):
294         """Set the camera CSR-driven inputs to the historical hardcoded scene:
295         cam at (0, 1.5, -8), identity basis (right=x, up=y, fwd=z). With these
296         defaults the new basis math reduces to (-px, py, ONE) - bit-equal to the
297         legacy raw_x_c/y_c/z_c, so existing test goldens still pass."""
298         forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)

```

```

299     dut.cam_x.value = to_unsigned(CAM_ORIGIN.x)
300     dut.cam_y.value = to_unsigned(CAM_ORIGIN.y)
301     dut.cam_z.value = to_unsigned(CAM_ORIGIN.z)
302     dut.right_x.value = to_unsigned(right.x)
303     dut.right_z.value = to_unsigned(right.z)
304     dut.up_x.value = to_unsigned(up.x)
305     dut.up_y.value = to_unsigned(up.y)
306     dut.up_z.value = to_unsigned(up.z)
307     dut.fwd_x.value = to_unsigned(forward.x)
308     dut.fwd_y.value = to_unsigned(forward.y)
309     dut.fwd_z.value = to_unsigned(forward.z)
310
311     async def do_batch_count_emits(dut, base_x, y, light, scene=DEFAULT_SCENE,
312                                   scene_state=None):
313         """Same as do_batch but also returns (total_emits, unique_addrs)
314         so reflection tests can detect the failure mode where complete_count
315         reaches BATCH but some lanes never wr_en (or some get hit twice).
316         """
317         if scene_state is None:
318             scene_state = SceneState().from_scene_tuple(scene)
319         dut.pixel_x.value = base_x
320         dut.pixel_y.value = y
321         dut.light_x.value = to_unsigned(light.x)
322         dut.light_y.value = to_unsigned(light.y)
323         dut.light_z.value = to_unsigned(light.z)
324         if hasattr(dut, 'K_const'):
325             dut.K_const.value = to_unsigned(k_const_from_light(light))
326         scene_state.drive(dut)
327         drive_default_camera(dut)
328         dut.start.value = 1
329         await RisingEdge(dut.clk)
330         dut.start.value = 0
331
332         captured = {}
333         total_emits = 0
334         unique_addrs = set()
335         cycles = 0
336         while True:
337             await RisingEdge(dut.clk)
338             if int(dut.wr_en.value) == 1:
339                 addr = int(dut.wr_addr.value)
340                 data = int(dut.wr_data.value)
341                 captured[addr] = ((data >> 16) & 0xff,
342                                  (data >> 8) & 0xff,
343                                  data & 0xff)
344                 total_emits += 1
345                 unique_addrs.add(addr)
346             if int(dut.done.value) == 1:
347                 break
348             cycles += 1
349             if cycles > 5000:
350                 raise TimeoutError("raytracer_batch did not assert done")
351
352         results = [captured.get(i, (0, 0, 0)) for i in range(N)]
353         return results, cycles, total_emits, len(unique_addrs)
354
355     async def do_batch(dut, base_x, y, light, scene=DEFAULT_SCENE,
356                       scene_state=None):
357         """Start a batch at (base_x, y) and capture per-lane RGB by listening on
358         the BRAM write port. If `scene_state` is supplied, it's pushed into
359         dut.scene; otherwise a default SceneState is built from the scene tuple."""
360         if scene_state is None:
361             scene_state = SceneState().from_scene_tuple(scene)
362         dut.pixel_x.value = base_x
363         dut.pixel_y.value = y
364         dut.light_x.value = to_unsigned(light.x)
365         dut.light_y.value = to_unsigned(light.y)
366         dut.light_z.value = to_unsigned(light.z)
367         if hasattr(dut, 'K_const'):
368             dut.K_const.value = to_unsigned(k_const_from_light(light))
369         scene_state.drive(dut)
370         drive_default_camera(dut)
371         dut.start.value = 1

```

```

373     await RisingEdge(dut.clk)
374     dut.start.value = 0
375
376     captured = {}
377     cycles = 0
378     while True:
379         await RisingEdge(dut.clk)
380         if int(dut.wr_en.value) == 1:
381             addr = int(dut.wr_addr.value)
382             data = int(dut.wr_data.value)
383             captured[addr] = ((data >> 16) & 0xff,
384                             (data >> 8) & 0xff,
385                             data & 0xff)
386             if int(dut.done.value) == 1:
387                 break
388             cycles += 1
389             if cycles > 5000:
390                 raise TimeoutError("raytracer_batch did not assert done")
391
392     results = [captured.get(i, (0, 0, 0)) for i in range(N)]
393     return results, cycles
394
395
396 @cocotb.test()
397 async def test_batch_center(dut):
398     """One batch near the screen center; check every lane against Python ref."""
399     clock = Clock(dut.clk, 10, units="ns")
400     cocotb.start_soon(clock.start())
401     await reset(dut)
402
403     light = Vec3FX(4.0, 4.0, 0.0)
404     base_x = WIDTH // 2 - N // 2
405     y = HEIGHT // 2
406
407     results, cycles = await do_batch(dut, base_x, y, light)
408     dut._log.info(f"batch at ({base_x},{y}) took {cycles} cycles")
409
410     fail = []
411     for i, (r, g, b) in enumerate(results):
412         x = base_x + i
413         px_fp, py_fp = pixel_to_screen(x, y)
414         py_r, py_g, py_b = render_pixel_py(px_fp, py_fp, light)
415         py_r8, py_g8, py_b8 = rgb8(py_r), rgb8(py_g), rgb8(py_b)
416         ok = (abs(r - py_r8) <= TOLERANCE and
417              abs(g - py_g8) <= TOLERANCE and
418              abs(b - py_b8) <= TOLERANCE)
419         marker = "OK" if ok else "FAIL"
420         dut._log.info(
421             f" lane {i:2d} px={x:3d}: hw=({r},{g},{b}) "
422             f"py=({py_r8},{py_g8},{py_b8}) [{marker}]"
423         )
424         if not ok:
425             fail.append((i, x, (r, g, b), (py_r8, py_g8, py_b8)))
426
427     assert not fail, f"{len(fail)} lane(s) mismatched: {fail}"
428
429
430 @cocotb.test()
431 async def test_batch_multiple_columns(dut):
432     """Several batches at different (x, y); also exercises the right-edge case."""
433     clock = Clock(dut.clk, 10, units="ns")
434     cocotb.start_soon(clock.start())
435     await reset(dut)
436
437     light = Vec3FX(4.0, 4.0, 0.0)
438     test_points = [
439         (0, HEIGHT // 2), # leftmost
440         (WIDTH // 4, HEIGHT // 4), # upper-left quadrant
441         (WIDTH - N, HEIGHT - 1), # rightmost (just at the edge)
442     ]
443     for base_x, y in test_points:
444         results, cycles = await do_batch(dut, base_x, y, light)
445         for i, (r, g, b) in enumerate(results):
446             x = base_x + i

```

```

447         px_fp, py_fp = pixel_to_screen(x, y)
448         py_r, py_g, py_b = render_pixel_py(px_fp, py_fp, light)
449         py_r8, py_g8, py_b8 = rgb8(py_r), rgb8(py_g), rgb8(py_b)
450         assert (abs(r - py_r8) <= TOLERANCE and
451                 abs(g - py_g8) <= TOLERANCE and
452                 abs(b - py_b8) <= TOLERANCE), (
453             f"batch ({base_x},{y}) lane {i} px={x}: "
454             f"hw=({r},{g},{b}) py=({py_r8},{py_g8},{py_b8})"
455         )
456         dut._log.info(f"batch at ({base_x},{y}): {cycles} cycles, {N} lanes OK")
457
458
459 @cocotb.test()
460 async def test_two_sphere_occlusion(dut):
461     """Sphere 1 placed *in front of* sphere 0 along the camera axis. Pixels
462     near the screen center should hit sphere 1 (red), with sphere 0 (blue)
463     fully occluded behind it. Verifies the closest-t winner mux in stage D
464     actually picks the closer sphere - not just the first one tested.
465
466     Also asserts the OCCLUSION INVARIANT directly: at least one center lane
467     must come back red-dominated (R > B). The vanilla HW vs golden compare
468     only catches *consistency* between the two; this extra check catches
469     the case where both compute "sphere 0" together due to a missing
470     closest-t comparison."""
471     clock = Clock(dut.clk, 10, units="ns")
472     cocotb.start_soon(clock.start())
473     await reset(dut)
474
475     # Sphere 0 stays at (0, 1, 0) r=1 (the historical scene).
476     # Sphere 1 sits 3 units closer to camera at (0, 1, -3) r=1.
477     # Camera at (0, 1.5, -8) looks at +z, so sphere 1 (z=-3) is in front
478     # of sphere 0 (z=0); both project to the same screen region.
479     occ_scene = _scene(
480         sc0=Vec3FX(0.0, 1.0, 0.0), r0=FixedPoint.from_float(1.0),
481         sc1=Vec3FX(0.0, 1.0, -3.0), r1=FixedPoint.from_float(1.0),
482     )
483
484     light = Vec3FX(4.0, 4.0, 0.0)
485     base_x = WIDTH // 2 - N // 2
486     y = HEIGHT // 2
487
488     results, cycles = await do_batch(dut, base_x, y, light, scene=occ_scene)
489     dut._log.info(f"occlusion batch at ({base_x},{y}) took {cycles} cycles")
490
491     # Compare HW vs Python golden across all lanes.
492     fail = []
493     for i, (r, g, b) in enumerate(results):
494         x = base_x + i
495         px_fp, py_fp = pixel_to_screen(x, y)
496         py_r, py_g, py_b = render_pixel_py(px_fp, py_fp, light, scene=occ_scene)
497         py_r8, py_g8, py_b8 = rgb8(py_r), rgb8(py_g), rgb8(py_b)
498         ok = (abs(r - py_r8) <= TOLERANCE and
499               abs(g - py_g8) <= TOLERANCE and
500               abs(b - py_b8) <= TOLERANCE)
501         if not ok:
502             fail.append((i, x, (r, g, b), (py_r8, py_g8, py_b8)))
503     assert not fail, f"{len(fail)} lane(s) mismatched against golden: {fail[:5]}..."
504
505     # Direct occlusion check: at least one center lane is red-dominated
506     # (sphere 1's color), proving the closest-t mux fired.
507     center = N // 2
508     red_dominant = sum(1 for (r, g, b) in results[center-10:center+10]
509                        if r > b + 20)
510     assert red_dominant > 0, (
511         "no red-dominated lane near screen center - sphere 1 is supposed "
512         "to occlude sphere 0 but HW returned blue or non-sphere colors. "
513         f"Center 20 lanes: {results[center-10:center+10]}"
514     )
515     dut._log.info(
516         f"occlusion verified: {red_dominant}/20 center lanes red-dominated"
517     )
518
519
520 @cocotb.test()

```



```

521 async def test_render_gif(dut):
522     """Render an animated GIF using the parallel raytracer_batch DUT.
523
524     Each row is processed in WIDTH/N batches; lanes within a batch run in
525     parallel, so a row takes ~N× fewer DUT cycles than the single-pixel test.
526     The width is always the DUT's full WIDTH (so the batches stride cleanly);
527     if RENDER_W differs we PIL-resize each finished frame at the end.
528
529     Configure via env vars:
530         RENDER_W      output width    (default WIDTH; resized via PIL if smaller)
531         RENDER_H      row count       (default 360, sampled from HEIGHT with stride)
532         RENDER_FRAMES  frame count     (default 1)
533         RENDER_OUT     output path     (default raytracer_batch.gif)
534     """
535     from PIL import Image
536
537     rw      = int(os.environ.get("RENDER_W", str(WIDTH)))
538     rh      = int(os.environ.get("RENDER_H", "360"))
539     nframes = int(os.environ.get("RENDER_FRAMES", "1"))
540     out_path = os.environ.get("RENDER_OUT", "raytracer_batch.gif")
541     # M2c: optional reflection knobs. Defaults preserve historical
542     # behavior (no reflection) so existing renders match byte-for-byte.
543     max_depth = int(os.environ.get("RENDER_MAX_DEPTH", "0"))
544     reflect_shift_0 = int(os.environ.get("RENDER_REFLECT_SHIFT_0", "1"))
545     reflect_shift_1 = int(os.environ.get("RENDER_REFLECT_SHIFT_1", "1"))
546     # Optional sphere 1 position override (close-pair scenes etc).
547     sc1_x = float(os.environ.get("RENDER_SC1_X", "2.5"))
548     sc1_y = float(os.environ.get("RENDER_SC1_Y", "1.0"))
549     sc1_z = float(os.environ.get("RENDER_SC1_Z", "0.0"))
550     scene = _scene(sc1=Vec3FX(sc1_x, sc1_y, sc1_z))
551
552     # Optional cone primitive. RENDER_CONE_HEIGHT > 0 enables it; default
553     # 0 disables (matches HW reset state).
554     cone_apex = Vec3FX(float(os.environ.get("RENDER_CONE_X", "0.0")),
555                        float(os.environ.get("RENDER_CONE_Y", "0.0")),
556                        float(os.environ.get("RENDER_CONE_Z", "0.0")))
557     cone_k_sq = FixedPoint.from_float(float(os.environ.get("RENDER_CONE_K_SQ", "1.0")))
558     cone_height = FixedPoint.from_float(float(os.environ.get("RENDER_CONE_HEIGHT", "0.0")))
559     cone_color = Vec3FX(float(os.environ.get("RENDER_CONE_R", "0.5")),
560                        float(os.environ.get("RENDER_CONE_G", "0.5")),
561                        float(os.environ.get("RENDER_CONE_B", "0.5")))
562     cone_refl = int(os.environ.get("RENDER_CONE_REFLECT", "0"))
563     # Pre-compute the normal-magnitude factor sqrt(k²·(1+k²)) once per
564     # frame in SW. HW recovers |n| with one mul instead of 3 squarings + sqrt.
565     k_sq_f = float(os.environ.get("RENDER_CONE_K_SQ", "1.0"))
566     cone_nf_v = FixedPoint.from_float(math.sqrt(k_sq_f * (1.0 + k_sq_f)))
567
568     # Build a SceneState carrying the render-frame env overrides; the
569     # geometry-only sphere positions still come from `scene` and are
570     # patched in via from_scene_tuple. SceneState defaults handle the
571     # rest (floor, sphere colors, etc. - all matching the wrapper reset).
572     render_scene = SceneState().from_scene_tuple(scene)
573     render_scene.s0_reflect_shift = reflect_shift_0
574     render_scene.s1_reflect_shift = reflect_shift_1
575     render_scene.cone_apex = cone_apex
576     render_scene.cone_k_sq = cone_k_sq
577     render_scene.cone_height = cone_height
578     render_scene.cone_norm_factor = cone_nf_v
579     render_scene.cone_color = cone_color
580     render_scene.cone_reflect_shift = cone_refl
581
582     assert WIDTH % N == 0, (
583         f"WIDTH={WIDTH} must be a multiple of N={N} so each row has whole batches"
584     )
585     batches_per_row = WIDTH // N
586
587     clock = Clock(dut.clk, 10, units="ns")
588     cocotb.start_soon(clock.start())
589     await reset(dut)
590
591     # Drive reflection recursion cap once after reset; do_batch's start
592     # latches scene at each batch's start edge.
593     dut.max_depth.value = max_depth
594

```

```

595 light_radius = float(os.environ.get("RENDER_LIGHT_RADIUS", "4.0"))
596 # Sun elevation = arctan(light_height / light_radius). 1.0 puts it
597 # in the upper third of the frame; raise via RENDER_LIGHT_HEIGHT for
598 # a higher sun (e.g. 3.0 37°, 4.0 45°).
599 light_height = float(os.environ.get("RENDER_LIGHT_HEIGHT", "1.0"))
600 # Optional explicit light position override (single frame only). When
601 # any of RENDER_LIGHT_{X,Y,Z} is set, all three are required and the
602 # orbit math above is bypassed.
603 light_xyz_override = (
604     os.environ.get("RENDER_LIGHT_X") if os.environ.get("RENDER_LIGHT_X") is not None
605     else os.environ.get("RENDER_LIGHT_Y")
606 )
607
608 frames = []
609 for f in range(nframes):
610     if os.environ.get("RENDER_LIGHT_X") is not None:
611         light = Vec3FX(
612             float(os.environ["RENDER_LIGHT_X"]),
613             float(os.environ["RENDER_LIGHT_Y"]),
614             float(os.environ["RENDER_LIGHT_Z"]),
615         )
616     else:
617         angle = 2 * math.pi * f / nframes
618         light = Vec3FX(
619             light_radius * math.cos(angle),
620             light_height,
621             light_radius * math.sin(angle),
622         )
623
624     dut._log.info(f"Rendering frame {f+1}/{nframes} ({WIDTH}x{rh}) ...")
625
626     pixels = [] # flat list of (r, g, b) tuples in row-major order
627     total_cycles = 0
628     for y in range(rh):
629         dut_y = (y * HEIGHT) // rh
630         row_cycles = 0
631         for batch_idx in range(batches_per_row):
632             base_x = batch_idx * N
633             results, c = await do_batch(
634                 dut, base_x, dut_y, light, scene,
635                 scene_state=render_scene,
636             )
637             row_cycles += c
638             total_cycles += c
639             pixels.extend(results)
640         dut._log.info(
641             f" frame {f+1}/{nframes} row {y+1}/{rh} done "
642             f"({batches_per_row} batches, {row_cycles} cycles)"
643         )
644     dut._log.info(
645         f"Total cycles: {total_cycles}"
646     )
647
648     img = Image.new("RGB", (WIDTH, rh))
649     img.putdata(pixels)
650     if rw != WIDTH:
651         img = img.resize((rw, rh))
652     frames.append(img)
653
654 frames[0].save(
655     out_path,
656     save_all=True,
657     append_images=frames[1:],
658     duration=80,
659     loop=0,
660 )
661 dut._log.info(f"Saved animation to {out_path}")
662
663 # -----
664 # Reflection (M2c) - verifies the spawn router / per-pipe accumulator combo
665 # actually re-issues a child ray and adds the attenuated reflection color.
666 # -----
667
668

```

```

669 @cocotb.test()
670 async def test_reflection_max_depth_1(dut):
671     """One bounce of reflection. max_depth=1, reflect_shift_0=2 (= 25%
672     contribution of the reflected color). The reflection direction in HW
673     is  $r = d - 2 \cdot (d \cdot n) \cdot n$  (true geometric reflection, computed in stage K).
674
675     Strategy: render the same center batch twice - once with max_depth=0
676     (baseline) and once with max_depth=1. For lanes that hit a sphere,
677     the reflection child traverses the pipeline using the parent's hit
678     point as origin and the surface normal as direction; its shaded
679     color (>> reflect_shift) sums into the lane's accumulator slot. So
680     sphere lanes should differ between the two runs (reflection adds
681     color), while sky lanes should be unchanged (sky rays don't spawn
682     children - hit_type=00 fails the spawn gate). The plane case behaves
683     like sky here too (hit_type=11 also fails the gate).
684     """
685     clock = Clock(dut.clk, 10, units="ns")
686     cocotb.start_soon(clock.start())
687     await reset(dut)
688
689     # Light positioned to cast a non-shadowed sphere0 hit at the screen
690     # center (so the primary's color is bright and reflection is visible
691     # in the sum).
692     light = Vec3FX(4.0, 4.0, 0.0)
693     base_x = WIDTH // 2 - N // 2
694     y = HEIGHT // 2
695
696     # Baseline pass: max_depth=0 with all reflect_shifts zeroed.
697     dut.max_depth.value = 0
698     baseline_scene = SceneState()
699     baseline_scene.s0_reflect_shift = 0
700     baseline_scene.s1_reflect_shift = 0
701     base_results, base_cyc = await do_batch(dut, base_x, y, light,
702                                           scene_state=baseline_scene)
703     dut._log.info(f"baseline (max_depth=0) batch took {base_cyc} cycles")
704
705     # Reflection pass: max_depth=1, reflect_shift=2 (= 1/4 contribution).
706     dut.max_depth.value = 1
707     refl_scene = SceneState()
708     refl_scene.s0_reflect_shift = 2
709     refl_scene.s1_reflect_shift = 2
710     refl_results, refl_cyc = await do_batch(dut, base_x, y, light,
711                                           scene_state=refl_scene)
712     dut._log.info(f"reflection (max_depth=1) batch took {refl_cyc} cycles")
713
714     # Categorize lanes by what the python ref says they hit.
715     sphere_hit_lanes = []
716     nonsphere_lanes = []
717     for i in range(N):
718         x = base_x + i
719         px_fp, py_fp = pixel_to_screen(x, y)
720         forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)
721         raw = forward + right * px_fp + up * py_fp
722         sc0, r0, sc1, r1 = DEFAULT_SCENE
723         t_s0 = intersect_sphere(CAM_ORIGIN, raw, sc0, r0)
724         t_s1 = intersect_sphere(CAM_ORIGIN, raw, sc1, r1)
725         if t_s0 is not None or t_s1 is not None:
726             sphere_hit_lanes.append(i)
727         else:
728             nonsphere_lanes.append(i)
729
730     dut._log.info(
731         f"classified: {len(sphere_hit_lanes)} sphere lanes, "
732         f"{len(nonsphere_lanes)} non-sphere lanes"
733     )
734     assert len(sphere_hit_lanes) > 0, (
735         "test setup error: no sphere hits at the chosen pixel coords"
736     )
737
738     # Non-sphere lanes (sky/plane) must NOT change - their hit_type
739     # never triggers the spawn router. Identical RGB byte-for-byte.
740     nonsphere_mismatches = []
741     for i in nonsphere_lanes:
742         if base_results[i] != refl_results[i]:

```

```

743         nonsphere_mismatches.append((i, base_results[i], refl_results[i]))
744     assert not nonsphere_mismatches, (
745         f"non-sphere lanes changed under reflection (should pass through):"
746         f" {nonsphere_mismatches[:5]}"
747     )
748
749     # Sphere lanes must change AT LEAST SOMETIMES. Even when the
750     # reflected ray hits sky directly above the sphere top, the sky
751     # gradient adds a non-zero contribution.
752     sphere_diffs = 0
753     for i in sphere_hit_lanes:
754         if base_results[i] != refl_results[i]:
755             sphere_diffs += 1
756     dut._log.info(
757         f"reflection deltas: {sphere_diffs}/{len(sphere_hit_lanes)} "
758         f"sphere lanes show different color with max_depth=1"
759     )
760     assert sphere_diffs > 0, (
761         "no sphere lane changed under reflection - spawn router or "
762         "accumulator weighting is broken (chain never re-issues, or "
763         "child color always sums to zero)"
764     )
765
766     # Spot-check direction: for a lane that hits sphere0 with normal
767     # pointing roughly "up", the reflection ray hits sky. Sky's blue
768     # channel is high, so reflection adds blue. Check that at least one
769     # such lane has its blue channel rise.
770     blue_rises = 0
771     for i in sphere_hit_lanes:
772         _, _, b_base = base_results[i]
773         _, _, b_refl = refl_results[i]
774         if b_refl > b_base:
775             blue_rises += 1
776     dut._log.info(
777         f"blue-channel rises (sky-reflection signature): "
778         f"{blue_rises}/{len(sphere_hit_lanes)} sphere lanes"
779     )
780     assert blue_rises > 0, (
781         "no sphere lane gained blue under reflection - reflected sky "
782         "color isn't being added to the accumulator"
783     )
784
785
786 @cocotb.test()
787 async def test_reflection_completion(dut):
788     """Sanity: with reflection on, every lane still completes (one
789     wr_en write per lane). Catches the failure mode where the spawn
790     FIFO gets stuck or re-issued rays never drop, which would leave
791     `done` deasserted forever (caught by do_batch's 5000-cycle timeout
792     but worth an explicit check).
793
794     Also asserts the reflection batch takes more cycles than the
795     baseline - chain rays each cost ~11=37c per pipe stage they
796     occupy.
797     """
798     clock = Clock(dut.clk, 10, units="ns")
799     cocotb.start_soon(clock.start())
800     await reset(dut)
801
802     light = Vec3FX(4.0, 4.0, 0.0)
803     base_x = WIDTH // 2 - N // 2
804     y = HEIGHT // 2
805
806     dut.max_depth.value = 0
807     baseline_scene = SceneState()
808     baseline_scene.s0_reflect_shift = 0
809     baseline_scene.s1_reflect_shift = 0
810     _, base_cyc = await do_batch(dut, base_x, y, light,
811                                scene_state=baseline_scene)
812
813     dut.max_depth.value = 1
814     refl_scene = SceneState()
815     refl_scene.s0_reflect_shift = 1
816     refl_scene.s1_reflect_shift = 1

```

```

817 refl_results, refl_cyc, total_emits, unique_addrs = \
818     await do_batch_count_emits(dut, base_x, y, light,
819                               scene_state=refl_scene)
820
821 dut._log.info(
822     f"reflection: cycles={refl_cyc}, total_emits={total_emits}, "
823     f"unique_addrs={unique_addrs}"
824 )
825
826 # `done` fires only when complete_count reaches BATCH = N, so we
827 # expect exactly N total emits and N unique addresses. A mismatch
828 # exposes either dropped emits (FIFO race) or duplicate emits
829 # (pending-bit confusion that fires acc_emit twice for one chain).
830 assert total_emits == N, (
831     f"reflection batch had {total_emits} emits (expected {N}); "
832     f"{'extra' if total_emits > N else 'missing'} emits suggest "
833     f"a pending-bit or chain-termination bug"
834 )
835 assert unique_addrs == N, (
836     f"reflection batch wrote {unique_addrs} unique tags ({N} expected); "
837     f"some addrs got hit twice - pending-bit didn't keep the "
838     f"chain in flight, or a stale ray from a prior batch dropped "
839     f"into the new batch's slot"
840 )
841
842 # Reflection adds work; should take at least as long as baseline.
843 # Allow some slack since not every lane spawns (sky/plane lanes
844 # don't), so the increase scales with how many lanes hit spheres.
845 assert refl_cyc >= base_cyc, (
846     f"reflection batch ({refl_cyc} c) ran faster than baseline "
847     f"({base_cyc} c) - that's geometrically impossible; chain "
848     f"rays can't be free"
849 )
850 dut._log.info(
851     f"completion verified: base={base_cyc}c, refl={refl_cyc}c, "
852     f"emits={total_emits}, unique={unique_addrs}"
853 )
854
855 @cocotb.test()
856 async def test_reflection_python_golden(dut):
857     """Per-lane bit-exact comparison of HW vs raytracer_reflect_ref.py.
858     The Python ref reimplements stage K shading + the spawn router +
859     accumulator weighting to mirror the HW byte for byte. Same TOLERANCE
860     the baseline tests use (~12 LSBs) for sphere-edge drift through
861     the sqrt/div fixed-point chain.
862     """
863     from raytracer_reflect_ref import render_pixel as render_pixel_refl
864
865     clock = Clock(dut.clk, 10, units="ns")
866     cocotb.start_soon(clock.start())
867     await reset(dut)
868
869     light = Vec3FX(4.0, 4.0, 0.0)
870     base_x = WIDTH // 2 - N // 2
871     y = HEIGHT // 2
872
873     max_depth = 1
874     shift_0 = 2
875     shift_1 = 2
876
877     dut.max_depth.value = max_depth
878     golden_scene = SceneState()
879     golden_scene.s0_reflect_shift = shift_0
880     golden_scene.s1_reflect_shift = shift_1
881     results, cycles = await do_batch(dut, base_x, y, light,
882                                     scene_state=golden_scene)
883     dut._log.info(f"reflection golden batch took {cycles} cycles")
884
885     forward, right, up = camera_basis(CAM_ORIGIN, CAM_TARGET)
886     fail = []
887     for i, (r, g, b) in enumerate(results):
888         x = base_x + i
889         py_r, py_g, py_b = render_pixel_refl(

```

```

891         x, y, light, max_depth, (shift_0, shift_1),
892         CAM_ORIGIN, forward, right, up
893     )
894     ok = (abs(r - py_r) <= TOLERANCE and
895           abs(g - py_g) <= TOLERANCE and
896           abs(b - py_b) <= TOLERANCE)
897     if not ok:
898         fail.append((i, x, (r, g, b), (py_r, py_g, py_b)))
899
900     if fail:
901         for item in fail[:5]:
902             dut._log.info(f" mismatch lane={item[0]} x={item[1]} "
903                           f"hw={item[2]} py={item[3]}")
904     assert not fail, f"{len(fail)}/{N} lanes mismatch python ref"
905     dut._log.info(f"all {N} lanes match raytracer_reflect_ref within ±{TOLERANCE}")
906
907
908 @cocotb.test()
909 async def test_reflection_max_depth_2(dut):
910     """Two bounces. max_depth=2 means a primary that hits a sphere
911     spawns one child, and that child (depth=1, depth<max_depth=2) can
912     itself spawn a grandchild if it hits a sphere too. Total chain
913     length up to 3 rays per slot.
914
915     The accumulator must handle 3 RMWs per slot before emitting:
916     - Primary: pending=1 (spawn), color += primary_byte >> 0
917     - Child: pending=1 (spawn), color += child_byte >> reflect_shift
918     - Grandchild: pending=0, color += gc_byte >> 2*reflect_shift, emit
919
920     Stresses the FIFO depth, the per-tag chain coherence, and the
921     is_spawn_d1 / pending logic across multiple loops.
922     """
923     clock = Clock(dut.clk, 10, units="ns")
924     cocotb.start_soon(clock.start())
925     await reset(dut)
926
927     light = Vec3FX(4.0, 4.0, 0.0)
928     base_x = WIDTH // 2 - N // 2
929     y = HEIGHT // 2
930
931     dut.max_depth.value = 2
932     depth2_scene = SceneState()
933     depth2_scene.s0_reflect_shift = 1
934     depth2_scene.s1_reflect_shift = 1
935     refl_results, refl_cyc, total_emits, unique_addrs = \
936         await do_batch_count_emits(dut, base_x, y, light,
937                                   scene_state=depth2_scene)
938
939     dut._log.info(
940         f"max_depth=2: cycles={refl_cyc}, total_emits={total_emits}, "
941         f"unique_addrs={unique_addrs}"
942     )
943     assert total_emits == N, (
944         f"max_depth=2 batch had {total_emits} emits (expected {N})"
945     )
946     assert unique_addrs == N, (
947         f"max_depth=2 batch wrote {unique_addrs} unique tags ({N} expected)"
948     )

```

test_cone.py

```

1  """Unit tests for cone intersection in raytracer_fp.
2
3  Self-contained: runs as `python3 test_cone.py`. Prints one line per case.
4  Exits 0 if all pass, 1 if any fail. No pytest dep.
5
6  Cases cover:
7      side hit + correct outward normal
8      cap hit + (0, -1, 0) normal
9      miss: ray points away
10     miss: ray would hit only the double-cone's upper half
11     miss: ray passes through cap radius but at y above slab

```

```

12     cap radius boundary
13     k2 1 (steeper cone): hit point + normal
14     """
15     import math
16     import sys
17
18     import raytracer_fp as rt
19     from raytracer_fp import FixedPoint as FP, Vec3FX, intersect_cone
20
21
22     # Helpers -----
23     def vfx(x, y, z):
24         return Vec3FX(float(x), float(y), float(z))
25
26
27     def fx(v):
28         return FP.from_float(float(v))
29
30
31     def to_floats(v):
32         return [round(FP.to_float(v.x), 4),
33                 round(FP.to_float(v.y), 4),
34                 round(FP.to_float(v.z), 4)]
35
36
37     PASSES = []
38     FAILS = []
39
40
41     def check(name, cond, detail=""):
42         if cond:
43             PASSES.append(name)
44             print(f" PASS {name}")
45         else:
46             FAILS.append((name, detail))
47             print(f" FAIL {name} {detail}")
48
49
50     def approx(a, b, tol=0.05):
51         """tol is generous - fp_sqrt+fp_div in Q14.12 has limited precision."""
52         return abs(a - b) <= tol
53
54
55     # Standard 45° cone at apex (0,2,4), height 2 → spans y[0,2], r_base=2 ----
56     APEX = vfx(0, 2, 4)
57     KSQ_45 = fx(1.0)      # tan2(45°)
58     HEIGHT_2 = fx(2.0)
59
60
61     # Test cases -----
62     def test_side_hit_front():
63         # Ray straight at the cone from in front: (0,1,-4) → +z. Hits at z=3,
64         # y=1 (mid-cone). Normal is gradient ((0, -1·(1-2), 3-4)) = (0,1,-1)
65         # normalized = (0, 0.707, -0.707).
66         o = vfx(0, 1, -4)
67         d = vfx(0, 0, 1)
68         t, n = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
69         check("side hit: t exists", t is not None)
70         if t is None:
71             return
72         check("side hit: t7", approx(FP.to_float(t), 7.0))
73         nf = to_floats(n)
74         check("side hit: normal y0.707", approx(nf[1], 0.7071))
75         check("side hit: normal z-0.707", approx(nf[2], -0.7071))
76         check("side hit: normal x0", approx(nf[0], 0.0))
77
78
79     def test_cap_hit_below():
80         # Ray from beneath, pointing straight up: hits the cap at y=0 first
81         # (cap is at apex.y - height = 0). Hit_point at apex_xz = (0,0,4).
82         # Cap normal is (0,-1,0).
83         o = vfx(0, -2, 4)
84         d = vfx(0, 1, 0)
85         t, n = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)

```

```

86     check("cap hit below: t exists", t is not None)
87     if t is None:
88         return
89     check("cap hit below: t2", approx(FP.to_float(t), 2.0))
90     nf = to_floats(n)
91     check("cap hit below: normal == (0,-1,0)",
92           approx(nf[0], 0) and approx(nf[1], -1) and approx(nf[2], 0))
93
94
95 def test_miss_pointing_away():
96     # Ray pointing away from cone (negative z, cone is at +z).
97     o = vfx(0, 1, 0)
98     d = vfx(0, 0, -1)
99     t, _ = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
100    check("miss: pointing away", t is None)
101
102
103 def test_miss_above_apex():
104     # Ray entirely above apex (y > 2): horizontal ray at y=5 should miss
105     # the single-half cone (lower half).
106     o = vfx(0, 5, -4)
107     d = vfx(0, 0, 1)
108     t, _ = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
109     check("miss: ray above apex slab", t is None)
110
111
112 def test_cap_boundary():
113     # Ray straight up through the cap edge: at apex_xz=(0,0,4), r_base=2,
114     # so cap edge is at (2,0,4) say. Origin (2, -1, 4), direction (0,1,0)
115     # hits cap exactly at radius 2.
116     o = vfx(2, -1, 4)
117     d = vfx(0, 1, 0)
118     t, n = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
119     check("cap boundary: t exists", t is not None)
120     if t is None:
121         return
122     # Could hit either cap (at y=0, t=1) or side; cap is closer.
123     check("cap boundary: t1", approx(FP.to_float(t), 1.0, tol=0.1))
124
125
126 def test_cap_outside():
127     # Ray straight up at (3, -1, 4) - radius 3 > r_base=2. Should miss the
128     # cap. The side might still be hit if the ray geometry allows, but in
129     # this case the ray is parallel to the cone axis (y-aligned), so it's
130     # the parallel-to-side case. With k=1,  $|D_{xz}|^2 - k^2 \cdot D_y^2 = 0 - 1 = -1$ 
131     # (a < 0), so the quadratic might still have real roots. Let's check.
132     # Actually a is negative, both roots are real. The side intersection
133     # could happen above the cap radius - let's see what the test gives.
134     # If it hits the side, that's still a valid hit. But the cap should
135     # not register since 3 > r_base=2.
136     o = vfx(3, -1, 4)
137     d = vfx(0, 1, 0)
138     t, _ = intersect_cone(o, d, APEX, KSQ_45, HEIGHT_2)
139     # If t is not None, must be a side hit not a cap hit. That's OK either
140     # way - the math is consistent. We just want it not to be the cap.
141     if t is not None:
142         # Verify hit is on the side: at hit_y, the implicit eq holds
143         ho = FP.to_float(o.x) + FP.to_float(t) * FP.to_float(d.x)
144         hy = FP.to_float(o.y) + FP.to_float(t) * FP.to_float(d.y)
145         hz = FP.to_float(o.z) + FP.to_float(t) * FP.to_float(d.z)
146         # cone surface check ( $k^2=1$ ):  $(x-Ax)^2 + (z-Az)^2 = (y-Ay)^2$ 
147         lhs = (ho - 0)**2 + (hz - 4)**2
148         rhs = (hy - 2)**2
149         check("cap-outside: hit lies on side surface (not cap)",
150               approx(lhs, rhs, tol=0.5),
151               detail=f"lhs={lhs:.3f} rhs={rhs:.3f} hit=({ho:.2f},{hy:.2f},{hz:.2f})")
152     else:
153         check("cap-outside: ray missed entirely (acceptable)", True)
154
155
156 def test_steep_cone():
157     #  $k^2=0.25 \rightarrow \tan()=0.5$ , half-angle 26.6°. Steeper, narrower cone.
158     # r_base =  $\sqrt{0.25} \cdot \text{height} = 0.5 \cdot 2 = 1$ . So cap is a 1-radius disc.
159     # Ray from (0, -1, 4) +y-axis hits cap at y=0, x=0, z=4.

```



```

160     ksq_steep = fx(0.25)
161     o = vfx(0, -1, 4)
162     d = vfx(0, 1, 0)
163     t, n = intersect_cone(o, d, APEX, ksq_steep, HEIGHT_2)
164     check("steep cone: cap hit exists", t is not None)
165     if t is None:
166         return
167     check("steep cone: cap t1", approx(FP.to_float(t), 1.0))
168     nf = to_floats(n)
169     check("steep cone: cap normal == (0,-1,0)",
170         approx(nf[1], -1))
171
172
173 def test_apex_origin_cone():
174     # Apex at origin (the SW default scene). Cone covers y[-2,0]. Ray
175     # from (0, -1, -2) +z direction hits side. With k=1, side at  $(y-0)^2$ 
176     # =  $x^2 + z^2$ . At y=-1, that's r=1. Ray goes (0,-1,-2)+t(0,0,1). At
177     # t=1 hit (0,-1,-1):  $(-1)^2 = 0^2 + (-1)^2 \rightarrow 1=1$ . So expected t=1.
178     apex_o = vfx(0, 0, 0)
179     o = vfx(0, -1, -2)
180     d = vfx(0, 0, 1)
181     t, _ = intersect_cone(o, d, apex_o, fx(1.0), fx(2.0))
182     check("apex-at-origin: t exists", t is not None)
183     if t is None:
184         return
185     check("apex-at-origin: t1", approx(FP.to_float(t), 1.0))
186
187
188 # Run -----
189 print("Running cone unit tests:")
190 print()
191 print("[side_hit_front]")
192 test_side_hit_front()
193 print("[cap_hit_below]")
194 test_cap_hit_below()
195 print("[miss_pointing_away]")
196 test_miss_pointing_away()
197 print("[miss_above_apex]")
198 test_miss_above_apex()
199 print("[cap_boundary]")
200 test_cap_boundary()
201 print("[cap_outside]")
202 test_cap_outside()
203 print("[steep_cone]")
204 test_steep_cone()
205 print("[apex_origin_cone]")
206 test_apex_origin_cone()
207 print()
208 print(f"Result: {len(PASSES)} pass, {len(FAILS)} fail")
209 sys.exit(0 if not FAILS else 1)

```

test_fp.py

```

1  import cocotb
2  from cocotb.clock import Clock
3  from cocotb.triggers import RisingEdge, Timer
4  import sys, os
5
6  sys.path.insert(0, os.path.dirname(__file__))
7  from raytracer_fp import FixedPoint
8
9  WORD = 27
10
11 def to_unsigned(val):
12     """Convert signed Python int to unsigned for cocotb assignment."""
13     if val < 0:
14         val = val + (1 << WORD)
15     return val
16
17 def to_signed(val):
18     """Convert unsigned cocotb result back to signed Python int."""
19     val = int(val) & ((1 << WORD) - 1)

```

```

20     if val >= (1 << (WORD - 1)):
21         val -= (1 << WORD)
22     return val
23
24 # ---- fp_mul tests ----
25
26 @cocotb.test()
27 async def test_fp_mul_basic(dut):
28     """Test basic multiplication cases."""
29     cases = [
30         (1.5, 2.0),
31         (-3.0, 0.5),
32         (0.1, 0.1),
33         (0.0, 5.0),
34         (-1.0, -1.0),
35         (3.0, -2.5),
36         (0.001, 1000.0),
37     ]
38     for a_f, b_f in cases:
39         a = FixedPoint.from_float(a_f)
40         b = FixedPoint.from_float(b_f)
41         expected = FixedPoint.mul(a, b)
42
43         dut.a.value = to_unsigned(a)
44         dut.b.value = to_unsigned(b)
45         await Timer(1, units="ns")
46
47         result = to_signed(dut.result.value)
48         assert result == expected, (
49             f"fp_mul({a_f}, {b_f}): got {result}, expected {expected}"
50         )
51     dut._log.info("All fp_mul basic tests passed")

```

test_fp_div.py

```

1  import cocotb
2  from cocotb.clock import Clock
3  from cocotb.triggers import RisingEdge, Timer
4  import sys, os
5
6  sys.path.insert(0, os.path.dirname(__file__))
7  from raytracer_fp import FixedPoint
8
9  WORD = 27
10
11 def to_unsigned(val):
12     if val < 0:
13         val = val + (1 << WORD)
14     return val
15
16 def to_signed(val):
17     val = int(val) & ((1 << WORD) - 1)
18     if val >= (1 << (WORD - 1)):
19         val -= (1 << WORD)
20     return val
21
22 async def do_div(dut, a_fp, b_fp):
23     """Drive one division and wait for done. Returns (result, cycles)."""
24     dut.a.value = to_unsigned(a_fp)
25     dut.b.value = to_unsigned(b_fp)
26     dut.start.value = 1
27     await RisingEdge(dut.clk)
28     dut.start.value = 0
29
30     cycles = 0
31     for _ in range(200):
32         await RisingEdge(dut.clk)
33         cycles += 1
34         if dut.done.value == 1:
35             return to_signed(dut.result.value), cycles
36
37     raise TimeoutError("fp_div did not assert done")

```

```

38
39 @cocotb.test()
40 async def test_fp_div_basic(dut):
41     """Test basic division cases."""
42     clock = Clock(dut.clk, 10, units="ns")
43     cocotb.start_soon(clock.start())
44
45     dut.rst_n.value = 0
46     await RisingEdge(dut.clk)
47     await RisingEdge(dut.clk)
48     dut.rst_n.value = 1
49     await RisingEdge(dut.clk)
50
51     cases = [
52         (3.0, 2.0),
53         (1.0, 4.0),
54         (-6.0, 3.0),
55         (7.5, -2.5),
56         (-1.0, -1.0),
57         (0.0, 1.0),
58         (100.0, 0.1),
59     ]
60     for a_f, b_f in cases:
61         a = FixedPoint.from_float(a_f)
62         b = FixedPoint.from_float(b_f)
63         expected = FixedPoint.div(a, b)
64
65         result, cycles = await do_div(dut, a, b)
66
67         diff = abs(result - expected)
68         assert diff <= 1, (
69             f"fp_div({a_f}, {b_f}): got {result} ({FixedPoint.to_float(result):.6f}), "
70             f"expected {expected} ({FixedPoint.to_float(expected):.6f}), diff={diff}"
71         )
72         dut._log.info(f"fp_div({a_f}, {b_f}) = {FixedPoint.to_float(result):.6f} OK [{cycles} cycles]")
73
74 @cocotb.test()
75 async def test_fp_div_by_zero(dut):
76     """Division by zero should return 0 immediately."""
77     clock = Clock(dut.clk, 10, units="ns")
78     cocotb.start_soon(clock.start())
79
80     dut.rst_n.value = 0
81     await RisingEdge(dut.clk)
82     await RisingEdge(dut.clk)
83     dut.rst_n.value = 1
84     await RisingEdge(dut.clk)
85
86     a = FixedPoint.from_float(5.0)
87     result, cycles = await do_div(dut, a, 0)
88     assert result == 0, f"div by zero: got {result}, expected 0"
89     dut._log.info(f"Division by zero returns 0 OK [{cycles} cycles]")

```

test_fp_sqrt.py

```

1  import cocotb
2  from cocotb.clock import Clock
3  from cocotb.triggers import RisingEdge
4  import sys, os
5
6  sys.path.insert(0, os.path.dirname(__file__))
7  from raytracer_fp import FixedPoint
8
9  WORD = 27
10
11 def to_unsigned(val):
12     if val < 0:
13         val = val + (1 << WORD)
14     return val
15
16 def to_signed(val):
17     val = int(val) & ((1 << WORD) - 1)

```

```

18     if val >= (1 << (WORD - 1)):
19         val -= (1 << WORD)
20     return val
21
22 async def do_sqrt(dut, a_fp):
23     """Drive one sqrt and wait for done. Returns (result, cycles)."""
24     dut.a.value = to_unsigned(a_fp)
25     dut.start.value = 1
26     await RisingEdge(dut.clk)
27     dut.start.value = 0
28
29     cycles = 0
30     for _ in range(200):
31         await RisingEdge(dut.clk)
32         cycles += 1
33         if dut.done.value == 1:
34             return to_signed(dut.result.value), cycles
35
36     raise TimeoutError("fp_sqrt did not assert done")
37
38 @cocotb.test()
39 async def test_fp_sqrt_basic(dut):
40     """Test square root against Python reference."""
41     clock = Clock(dut.clk, 10, units="ns")
42     cocotb.start_soon(clock.start())
43
44     dut.rst_n.value = 0
45     await RisingEdge(dut.clk)
46     await RisingEdge(dut.clk)
47     dut.rst_n.value = 1
48     await RisingEdge(dut.clk)
49
50     cases = [1.0, 2.0, 4.0, 0.25, 0.5, 9.0, 16.0, 0.01, 100.0, 3.14159]
51     for val in cases:
52         a = FixedPoint.from_float(val)
53         expected = FixedPoint.sqrt(a)
54
55         result, cycles = await do_sqrt(dut, a)
56
57         diff = abs(result - expected)
58         tolerance = 2 # allow ±2 LSB since algorithms differ
59         assert diff <= tolerance, (
60             f"fp_sqrt({val}): got {result} ({FixedPoint.to_float(result):.6f}), "
61             f"expected {expected} ({FixedPoint.to_float(expected):.6f}), diff={diff}"
62         )
63         dut._log.info(
64             f"fp_sqrt({val}) = {FixedPoint.to_float(result):.6f} "
65             f"(expected {FixedPoint.to_float(expected):.6f}, diff={diff}) [{cycles} cycles]"
66         )
67
68 @cocotb.test()
69 async def test_fp_sqrt_zero(dut):
70     """sqrt(0) should return 0 immediately."""
71     clock = Clock(dut.clk, 10, units="ns")
72     cocotb.start_soon(clock.start())
73
74     dut.rst_n.value = 0
75     await RisingEdge(dut.clk)
76     await RisingEdge(dut.clk)
77     dut.rst_n.value = 1
78     await RisingEdge(dut.clk)
79
80     result, cycles = await do_sqrt(dut, 0)
81     assert result == 0, f"sqrt(0): got {result}, expected 0"
82     dut._log.info(f"sqrt(0) = 0 OK [{cycles} cycles]")

```

test_fp_inv.py

```

1 import cocotb
2 from cocotb.clock import Clock
3 from cocotb.triggers import RisingEdge
4 import sys, os

```

```

5 sys.path.insert(0, os.path.dirname(__file__))
6 from raytracer_fp import FixedPoint
7
8
9 WORD = 26
10
11
12 def to_unsigned(val):
13     if val < 0:
14         val = val + (1 << WORD)
15     return val
16
17
18 def to_signed(val):
19     val = int(val) & ((1 << WORD) - 1)
20     if val >= (1 << (WORD - 1)):
21         val -= (1 << WORD)
22     return val
23
24
25 async def do_inv(dut, b_fp):
26     dut.b.value = to_unsigned(b_fp)
27     dut.start.value = 1
28     await RisingEdge(dut.clk)
29     dut.start.value = 0
30
31     cycles = 0
32     for _ in range(200):
33         await RisingEdge(dut.clk)
34         cycles += 1
35         if dut.done.value == 1:
36             return to_signed(dut.result.value), cycles
37
38     raise TimeoutError("fp_inv did not assert done")
39
40
41 @cocotb.test()
42 async def test_fp_inv_basic(dut):
43     """1/b matches Python reference (= ONE/b via fp_div)."""
44     clock = Clock(dut.clk, 10, units="ns")
45     cocotb.start_soon(clock.start())
46
47     dut.rst_n.value = 0
48     await RisingEdge(dut.clk)
49     await RisingEdge(dut.clk)
50     dut.rst_n.value = 1
51     await RisingEdge(dut.clk)
52
53     cases = [2.0, 4.0, -3.0, -2.5, -1.0, 1.0, 0.1, 8.0, 0.5, 100.0]
54     one = FixedPoint.from_float(1.0)
55     for b_f in cases:
56         b = FixedPoint.from_float(b_f)
57         expected = FixedPoint.div(one, b)
58
59         result, cycles = await do_inv(dut, b)
60
61         diff = abs(result - expected)
62         assert diff <= 1, (
63             f"fp_inv({b_f}): got {result} ({FixedPoint.to_float(result):.6f}), "
64             f"expected {expected} ({FixedPoint.to_float(expected):.6f}), diff={diff}"
65         )
66         dut._log.info(f"fp_inv({b_f}) = {FixedPoint.to_float(result):.6f} OK  [{cycles} cycles]")
67
68
69 @cocotb.test()
70 async def test_fp_inv_by_zero(dut):
71     """1/0 should return 0 immediately (matches fp_div's b==0 short-circuit)."""
72     clock = Clock(dut.clk, 10, units="ns")
73     cocotb.start_soon(clock.start())
74
75     dut.rst_n.value = 0
76     await RisingEdge(dut.clk)
77     await RisingEdge(dut.clk)
78     dut.rst_n.value = 1

```

```

79     await RisingEdge(dut.clk)
80
81     result, cycles = await do_inv(dut, 0)
82     assert result == 0, f"1/0: got {result}, expected 0"
83     dut._log.info(f"1/0 returns 0 OK  [{cycles} cycles]")

```

Makefile.raytracer_batch_pipe

```

1  SIM = verilator
2  TOPLEVEL_LANG = verilog
3  VERILOG_SOURCES = $(shell pwd)/raytracer.sv
4  TOPLEVEL = rtl_batch_pipe
5  MODULE = test_raytracer_batch
6  SIM_BUILD = sim_build_raytracer_batch_pipe
7  EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC
8  COMPILE_ARGS += -GK=4 -GBATCH=120
9
10 include $(shell cocotb-config --makefiles)/Makefile.sim

```

Makefile.div

```

1  SIM = verilator
2  TOPLEVEL_LANG = verilog
3  VERILOG_SOURCES = $(shell pwd)/raytracer.sv
4  TOPLEVEL = fp_div
5  MODULE = test_fp_div
6  SIM_BUILD = sim_build_div
7  EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC
8
9  include $(shell cocotb-config --makefiles)/Makefile.sim

```

Makefile.mul

```

1  SIM = verilator
2  TOPLEVEL_LANG = verilog
3  VERILOG_SOURCES = $(shell pwd)/raytracer.sv
4  TOPLEVEL = fp_mul
5  MODULE = test_fp
6  EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC
7
8  include $(shell cocotb-config --makefiles)/Makefile.sim

```

Makefile.sqrt

```

1  SIM = verilator
2  TOPLEVEL_LANG = verilog
3  VERILOG_SOURCES = $(shell pwd)/raytracer.sv
4  TOPLEVEL = fp_sqrt
5  MODULE = test_fp_sqrt
6  SIM_BUILD = sim_build_sqrt
7  EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC
8
9  include $(shell cocotb-config --makefiles)/Makefile.sim

```

Makefile.fp_inv

```

1  SIM = verilator
2  TOPLEVEL_LANG = verilog
3  VERILOG_SOURCES = $(shell pwd)/raytracer.sv
4  TOPLEVEL = fp_inv
5  MODULE = test_fp_inv
6  SIM_BUILD = sim_build_inv
7  EXTRA_ARGS += --timing -Wno-WIDTHEXPAND -Wno-WIDTHTRUNC -GFRAC_BITS=12 -GWORD=26
8
9  include $(shell cocotb-config --makefiles)/Makefile.sim

```