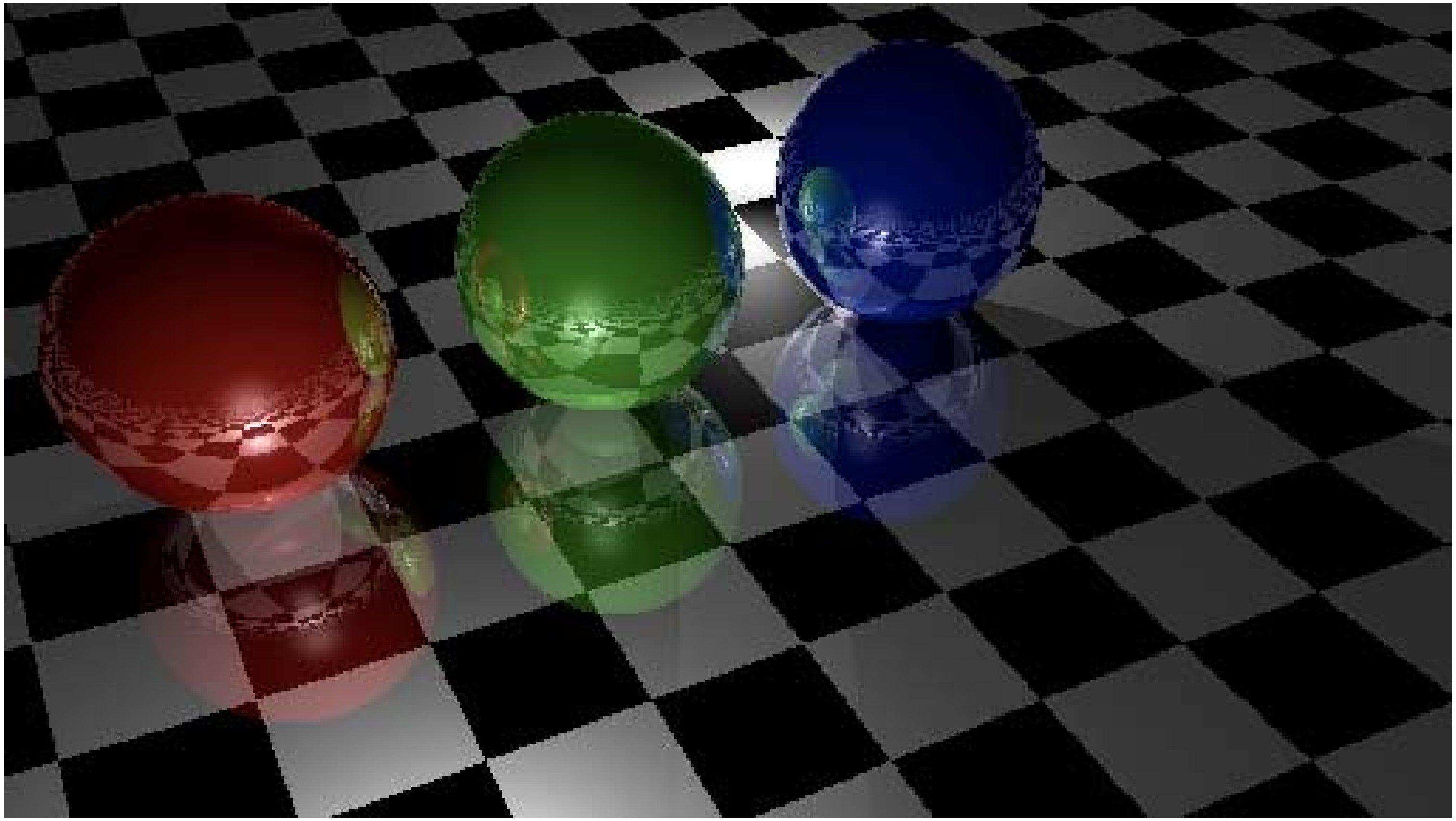
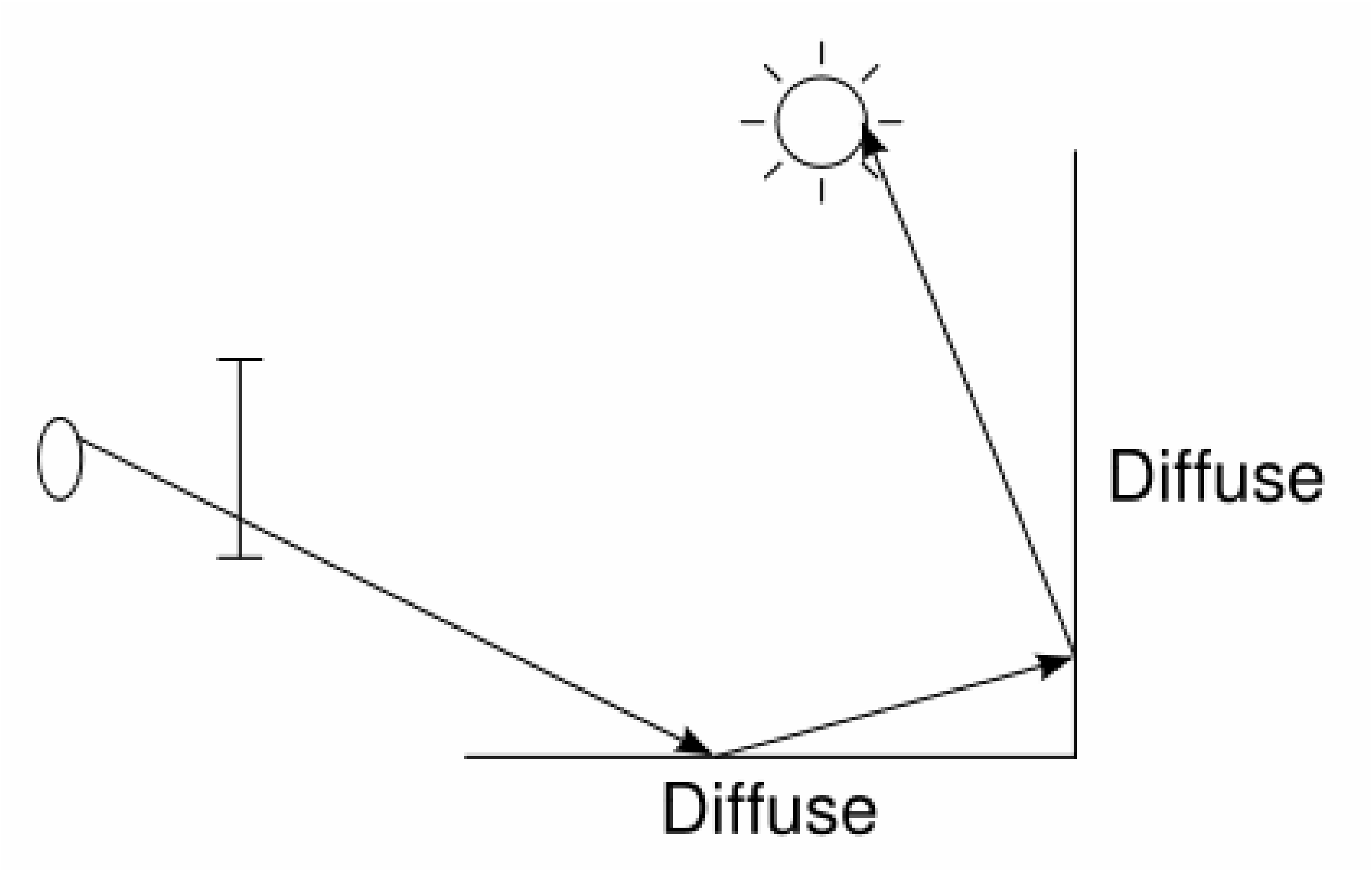
Real-Time Raytracing

Adam Lerer, Sam Gross

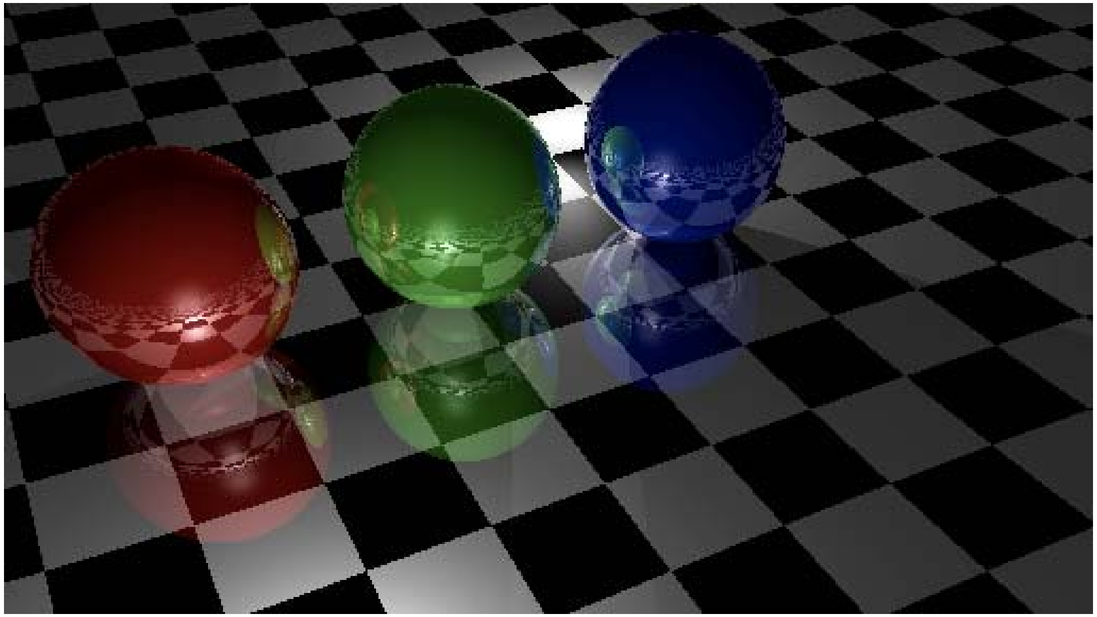


Raytracing Algorithm

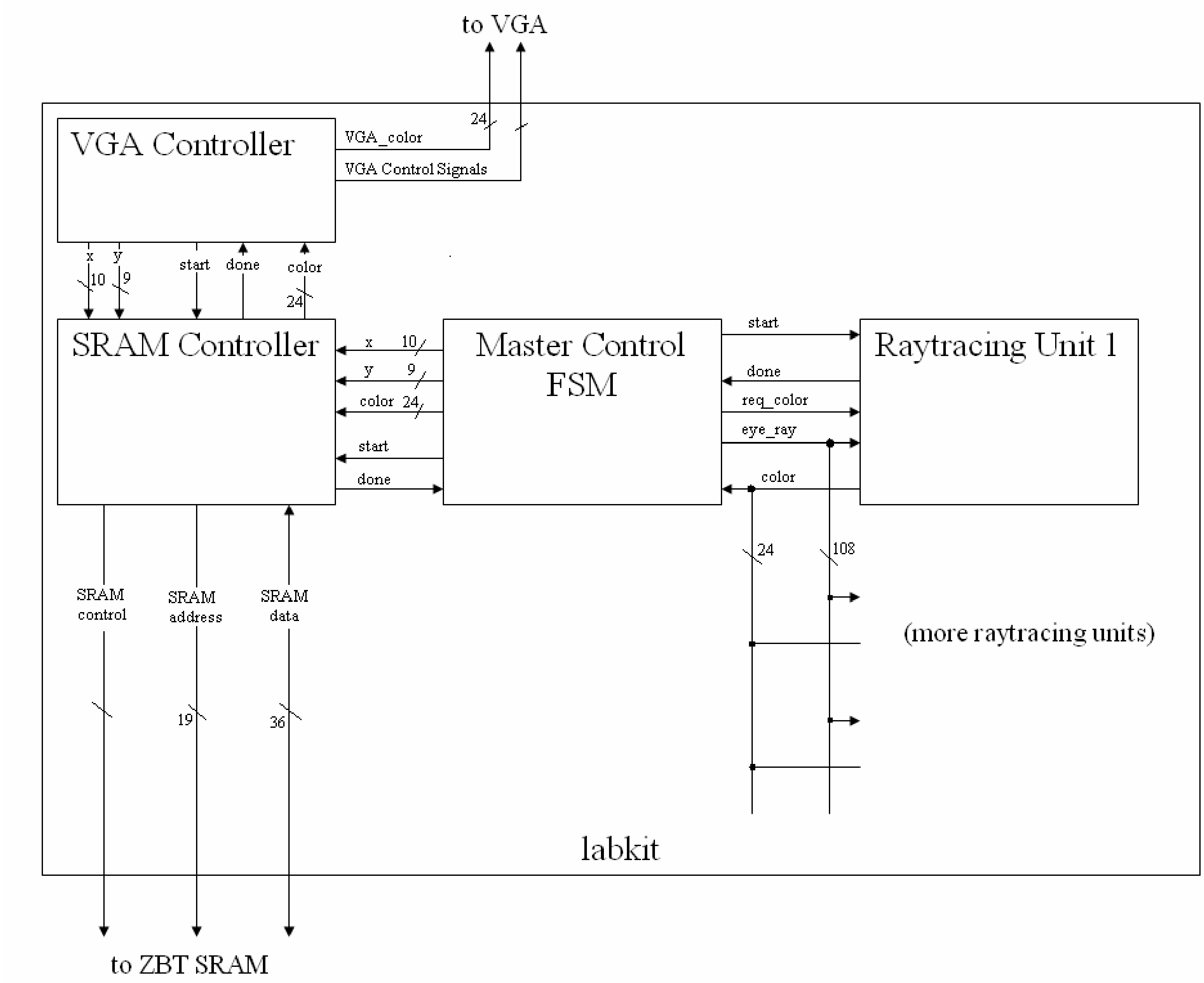


Overview

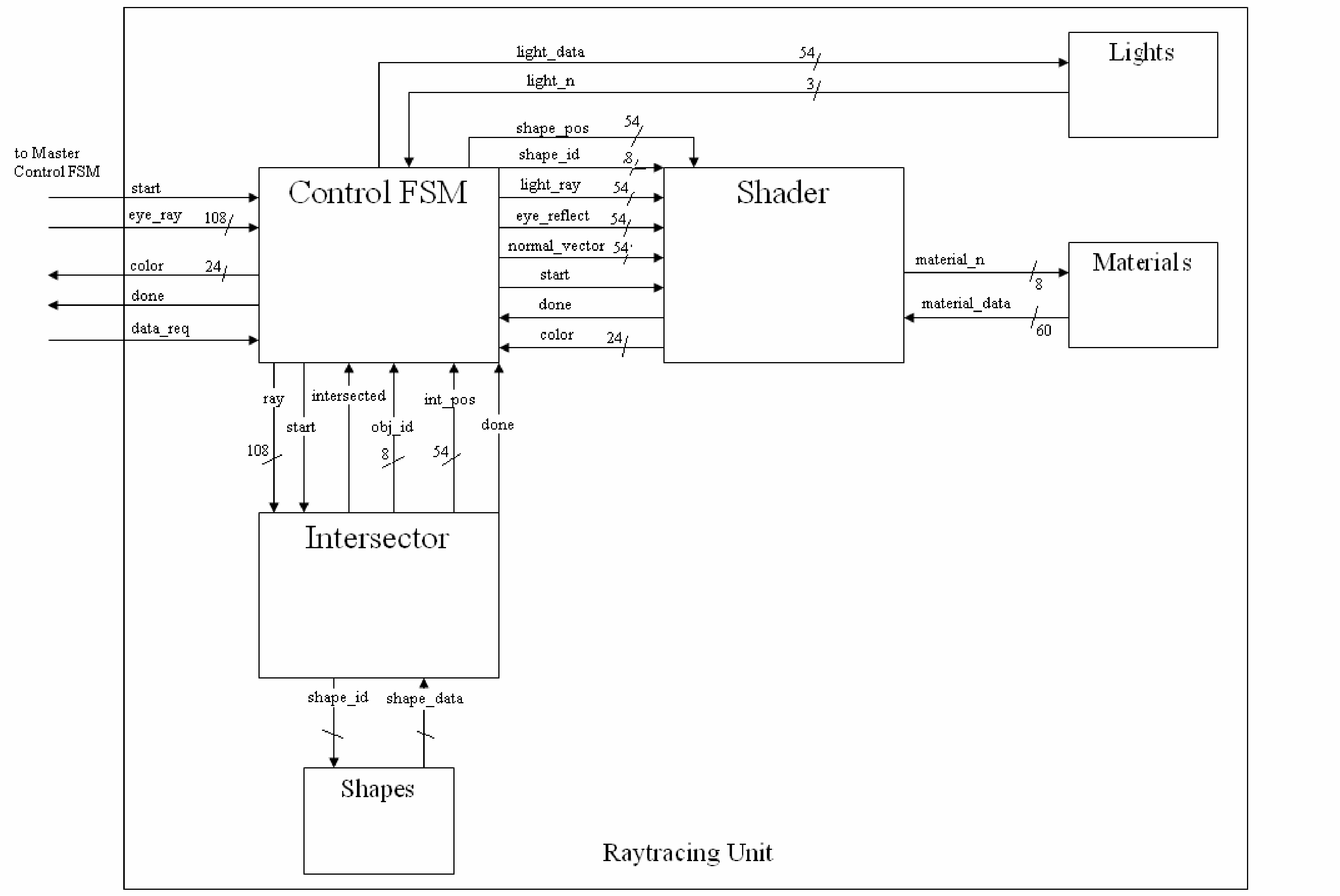
- Features of our ray tracer:
  - Shading
    - Ambient
    - Diffuse
    - Specular
  - Reflections
  - Shadowing
  - Shapes
    - Planes
    - Spheres
    - More?



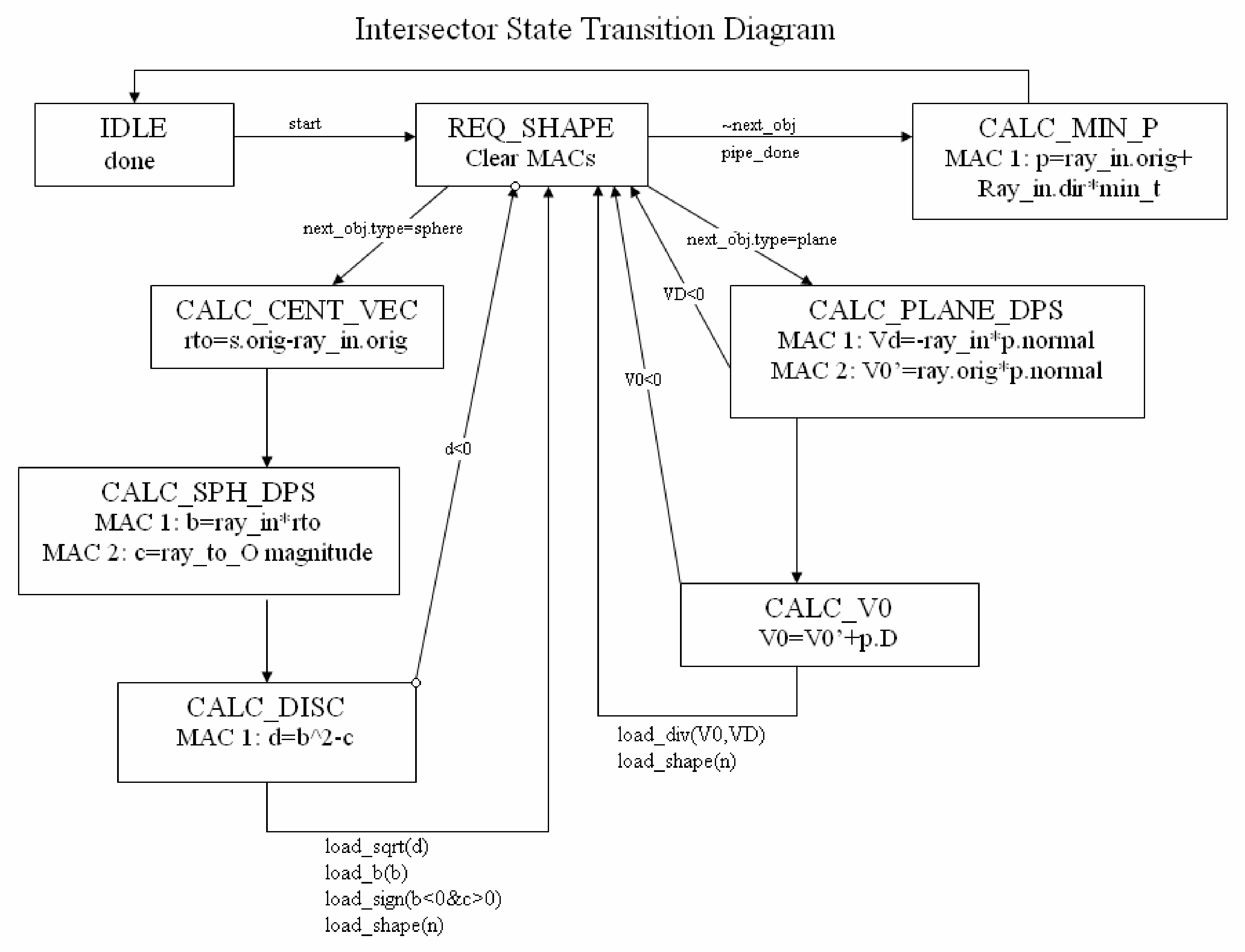
The Raytracer



The Raytracing Unit



The Intersector



Intersector State Transition Diagram

IDLE
done

start

REQ_SHAPE
Clear MACs

~next_obj
pipe_done

CALC_MIN_P
MAC 1: p=ray_in.orig+ Ray_in.dir*min_t

next_obj.type=sphere

next_obj.type=plane

CALC_CENT_VEC
rto=s.orig-ray_in.orig

VD<0

CALC_PLANE_DPS
MAC 1: Vd=-ray_in*p.normal
MAC 2: V0'=ray.orig*p.normal

V0<0

d<0

CALC_SPH_DPS
MAC 1: b=ray_in*rto
MAC 2: c=ray_to_O magnitude

CALC_V0
V0=V0'+p.D

CALC_DISC
MAC 1: d=b^2-c

load_div(V0,VD)
load_shape(n)

load_sqrt(d)
load_b(b)
load_sign(b<0&c>0)
load_shape(n)

Pipelining: Success & Failure

- Ray tracing requires high-latency operations
  - Square root, divide
- IP implementations are 10+-cycle, pipelined
- Can we utilize this pipelining?
- Success: Calculating Intersection Points
- Failure: Normalizing Vectors

Multiple FPGAs

- Raytracing calculates each pixel independently
- If we time-multiplex I/O, we only need one ~150-bit bus shared data bus
- Slave FPGAs containing just RTUs can be utilized for a linear speed increase
- Treat RTU I/O as asynchronous; register inputs and outputs

Progress So Far

- Java Prototype
- VGA
  - 640x480
  - 1024x768
- SRAM
  - Double-buffered ZBT SRAM
  - Ping-pong buffer alternation

Questions?