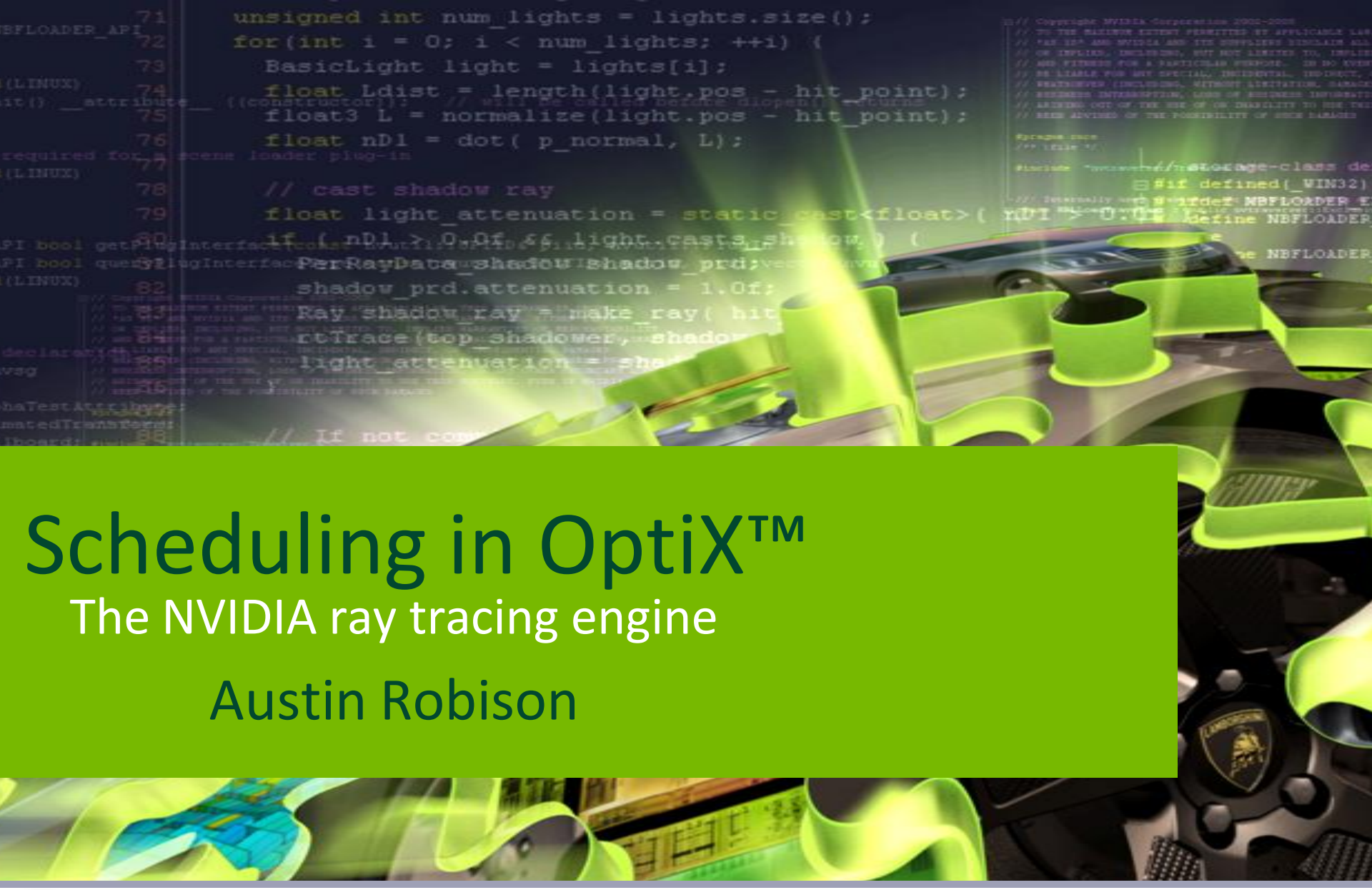


Scheduling in OptiX™

The NVIDIA ray tracing engine

Austin Robison

The OptiX Engine

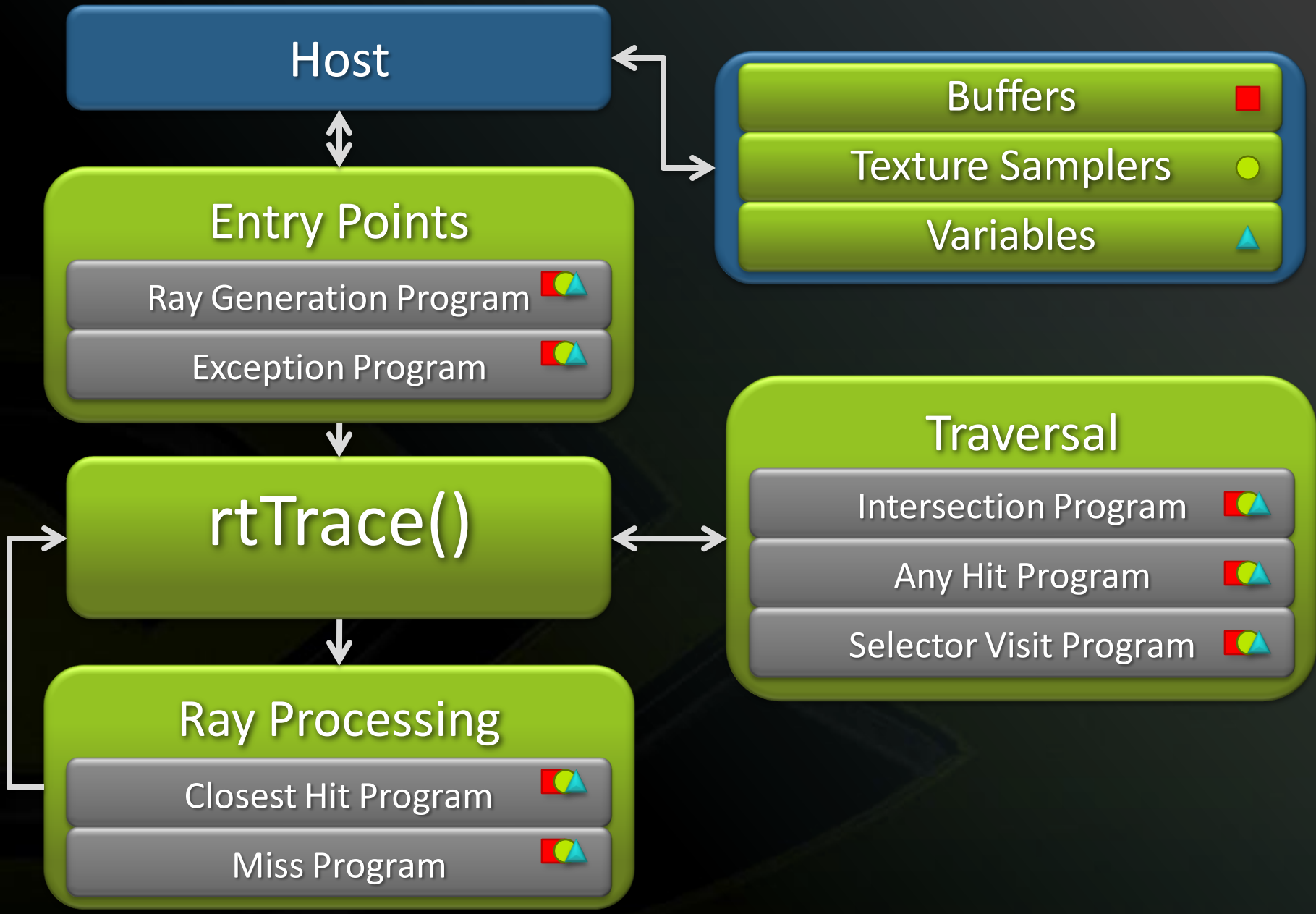
- A General Purpose Ray Tracing API
  - Rendering, baking, collision detection, A.I. queries, etc.
  - Modern shader-centric, stateless and bindless design
  - Is not a renderer but can implement many types of renderers
- Highly Programmable
  - Shading with arbitrary ray payloads
  - Ray generation/framebuffer operations (cameras, data unpacking, etc.)
  - Programmable intersection (triangles, NURBS, implicit surfaces, etc.)
- Easy to Program
  - Write single ray code (no exposed ray packets)
  - No need to rewrite programs to target different hardware
  - Acceleration structures are abstracted by the API

Programmable Operations

Rasterization	Ray Tracing
• Fragment	• Closest Hit • Any Hit
• Vertex • Geometry	• Intersection • Selector
	• Ray Generation • Miss • Exception

The ensemble of programs defines the rendering algorithm

(or collision detection algorithm, or sound propagation algorithm, etc.)


Host
Buffers
Texture Samplers
Variables
Entry Points
Ray Generation Program
Exception Program
rtTrace()
Traversal
Intersection Program
Any Hit Program
Selector Visit Program
Ray Processing
Closest Hit Program
Miss Program

- Closest Hit Programs: called once after traversal has found the closest intersection
  - Used for traditional surface shading
  - Deferred shading
- Any Hit Programs: called during traversal for each potentially closest intersection
  - Transparency without traversal restart (can read textures): rtIgnoreIntersection()
  - Terminate shadow rays that encounter opaque objects: rtTerminateRay()
- Both can be used for shading by modifying per ray state

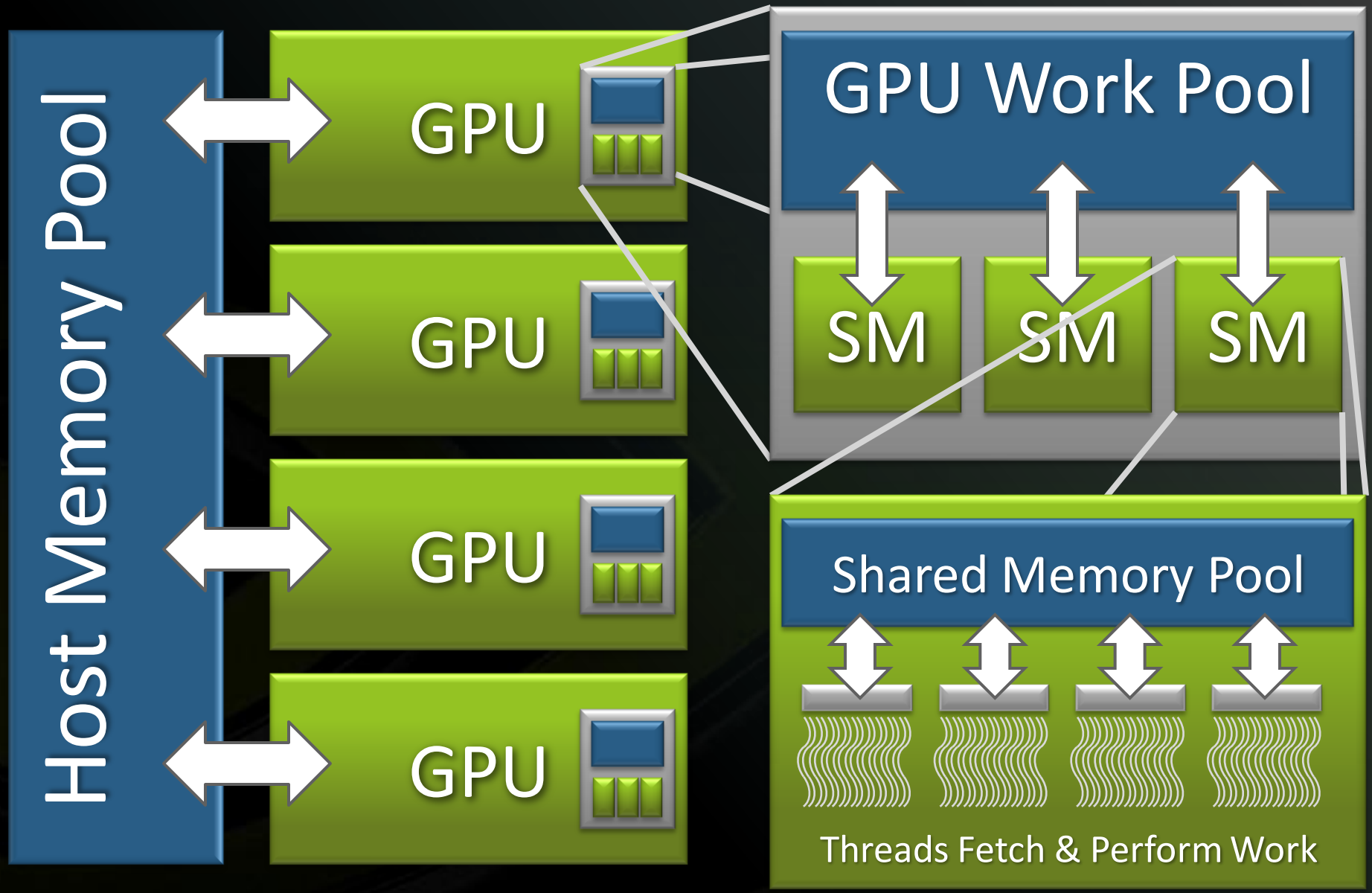
```
struct PerRayData_radiance
{
  float3 result;
  float importance;
  int depth;
};
```

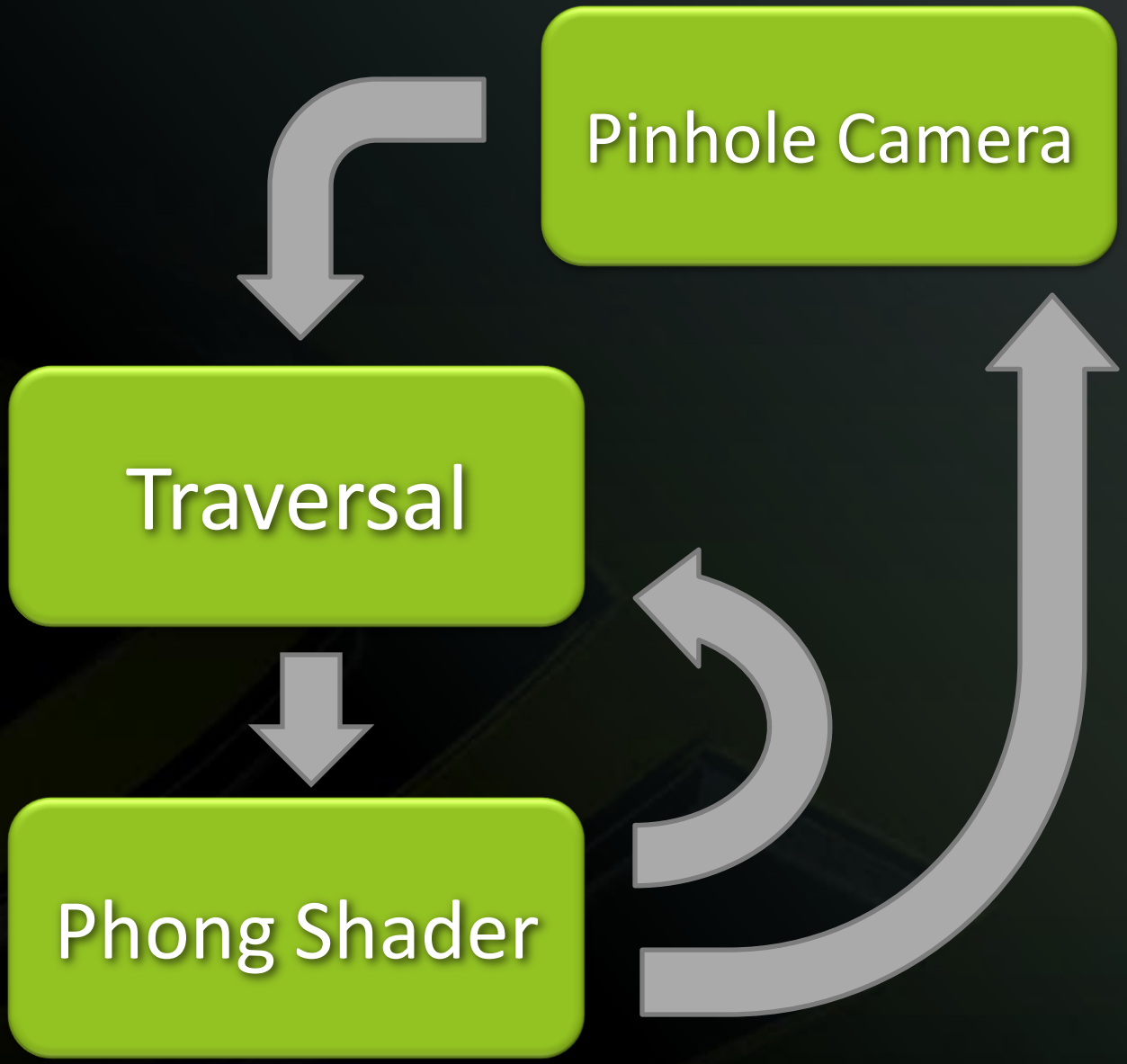
```
rtDeclareVariable(float3, eye);
rtDeclareVariable(float3, U);
rtDeclareVariable(float3, V);
rtDeclareVariable(float3, W);
rtBuffer<float4, 2> output_buffer;
rtDeclareVariable(
 rtNode, top_object);
rtDeclareVariable(
 unsigned int, radiance_ray_type);

rtDeclareSemanticVariable(
 rtRayIndex, rayIndex);
```

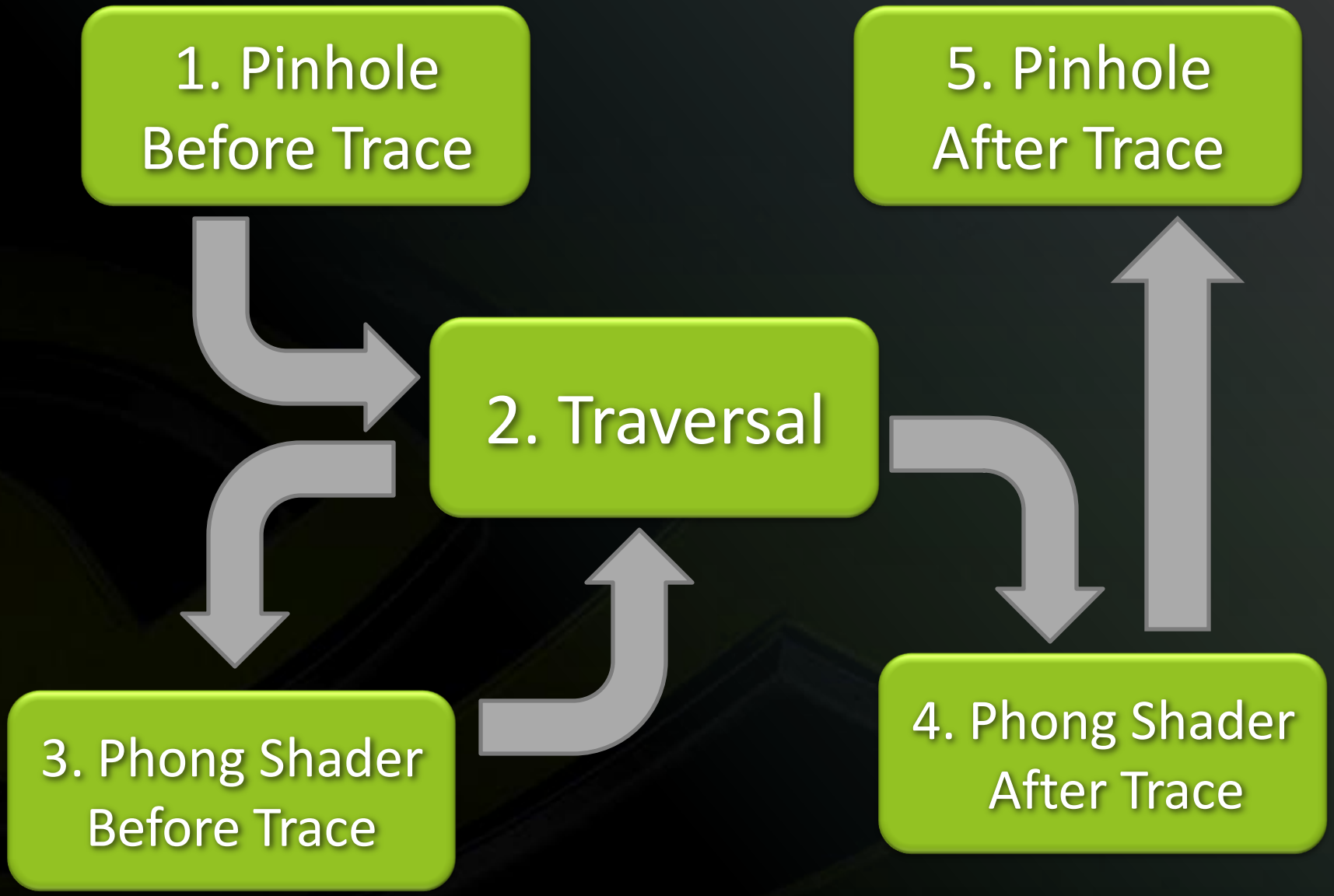
```
RT_PROGRAM void pinhole_camera()
{
  …
  float2 d = make_float2(index) /
   make_float2(screen) * 2.f - 1.f;
  float3 ray_origin = eye;
  float3 ray_direction =
   normalize(d.x*U + d.y*V + W);

  Ray ray = make_ray(ray_origin,
   ray_direction,
   radiance_ray_type,
   scene_epsilon, RT_DEFAULT_MAX);

  PerRayData_radiance prd;
  prd.importance = 1.f;
  prd.depth = 0;

  rtTrace(top_object, ray, prd);
  output_buffer[index] =
   prd.result;
}
```

Execution Model on GT200 Class HW

- Continuations
  - Execution is a state machine, presented as recursion
  - Software managed local stack
  - Accomplished through PTX recompilation
- Persistent Warps
  - Launch just enough threads to fill the machine
  - Each warp, upon termination of its rays, gets new work
- Two level load balancing
  - Balance work between SMs and their persistent warps
  - Balance work between GPUs

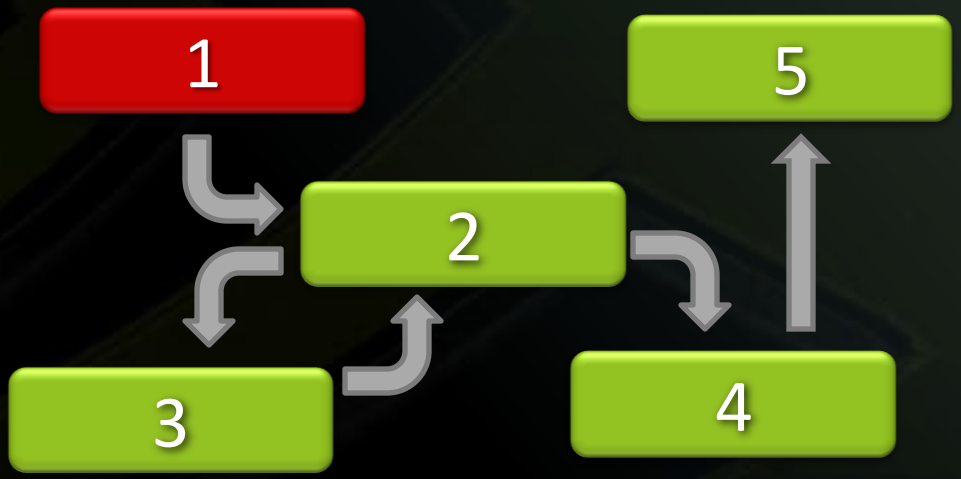

Host Memory Pool
GPU
GPU
GPU
GPU
GPU Work Pool
SM
SM
SM
Shared Memory Pool
Threads Fetch & Perform Work


Pinhole Camera
Traversal
Phong Shader


1. Pinhole Before Trace
2. Traversal
3. Phong Shader Before Trace
4. Phong Shader After Trace
5. Pinhole After Trace

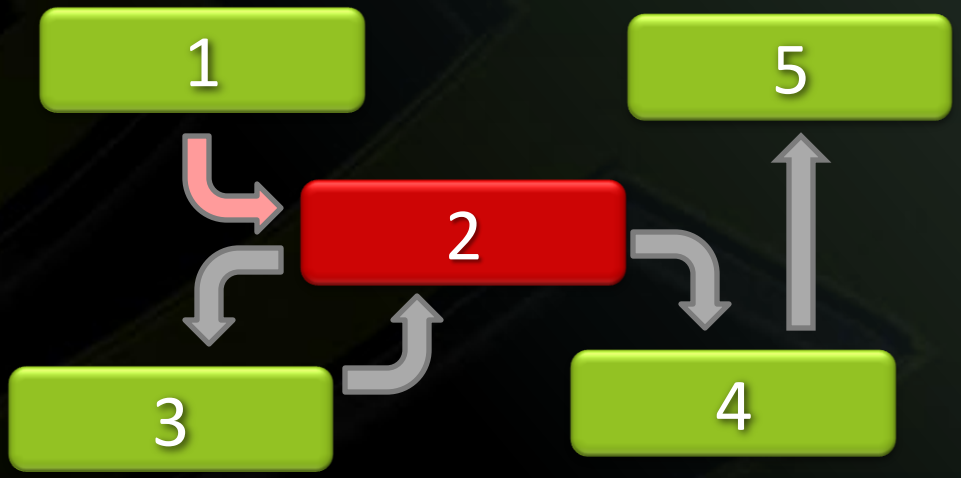
```
while(true):
   switch(state):
      case 0:
         …code for state0…
      case 1:
         …code for state1…
      case 2:
         …code for state2…
      …
```

Call to rtTrace()



2	2	2	2	2	2	2	2

All threads enter traversal, hit the Phong material

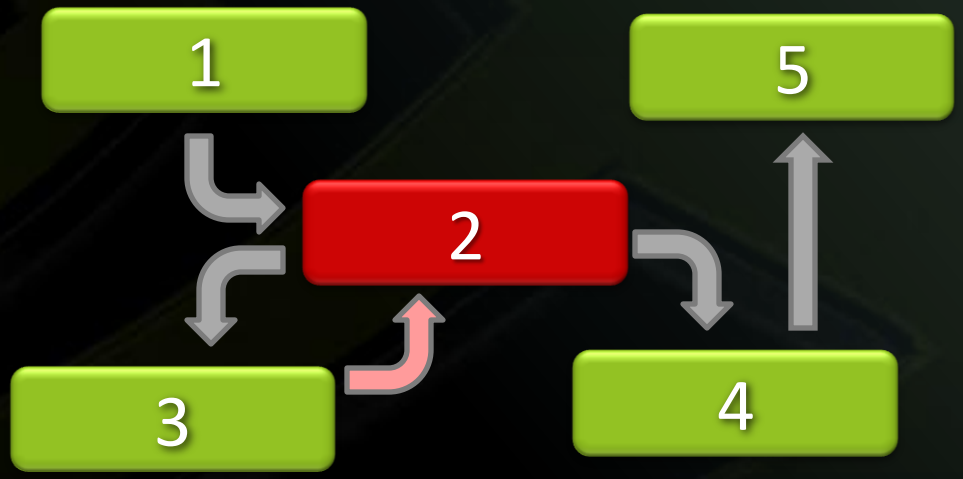


3	3	3	3	3	3	3	3

All cast secondary rays via rtTrace()

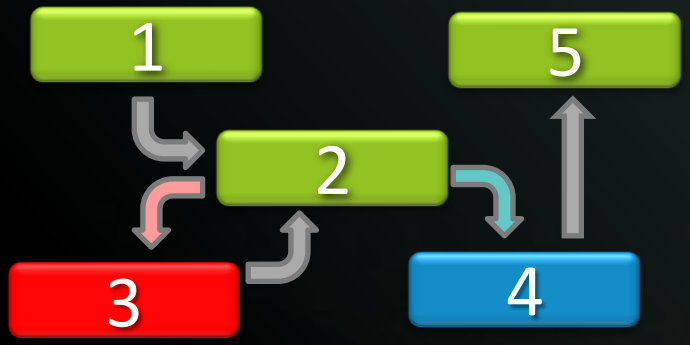
2	2	2	2	2	2	2	2

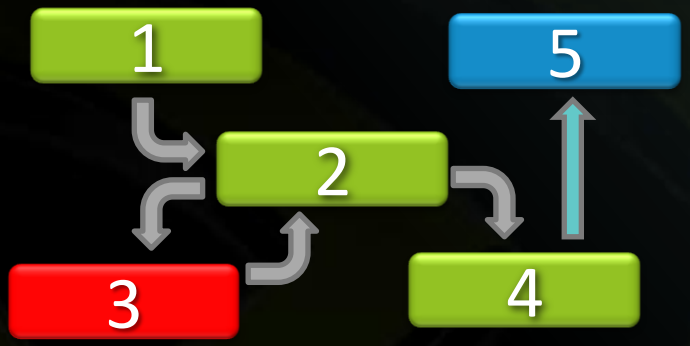
Back to traversal,
some rays hit again and some miss

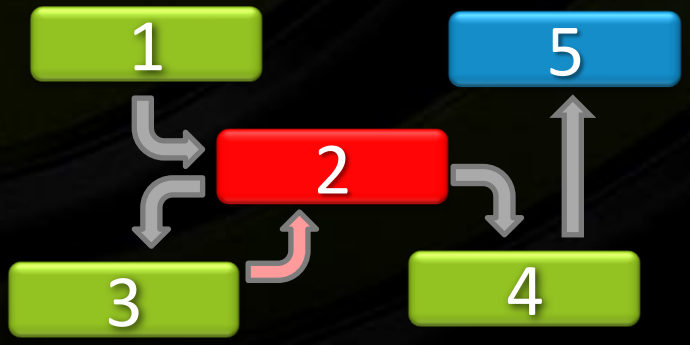


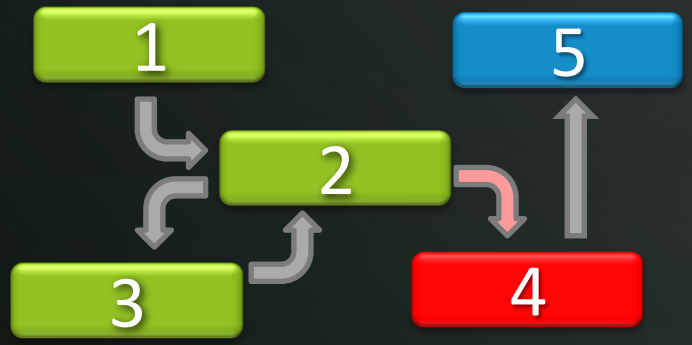
We now have divergence in the warp's execution.
What is the minimum number of steps to state 5?

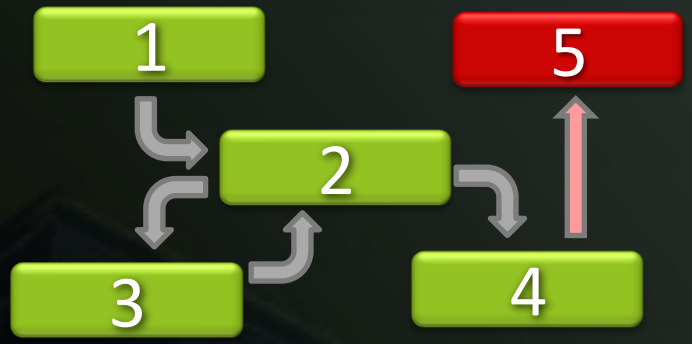
```
while(true):
   schedule = schedule_state()
   if(state == schedule):
      switch(state):
         case 0:
            …code for state0…
         case 1:
            …code for state1…
         case 2:
            …code for state2…
         …
```





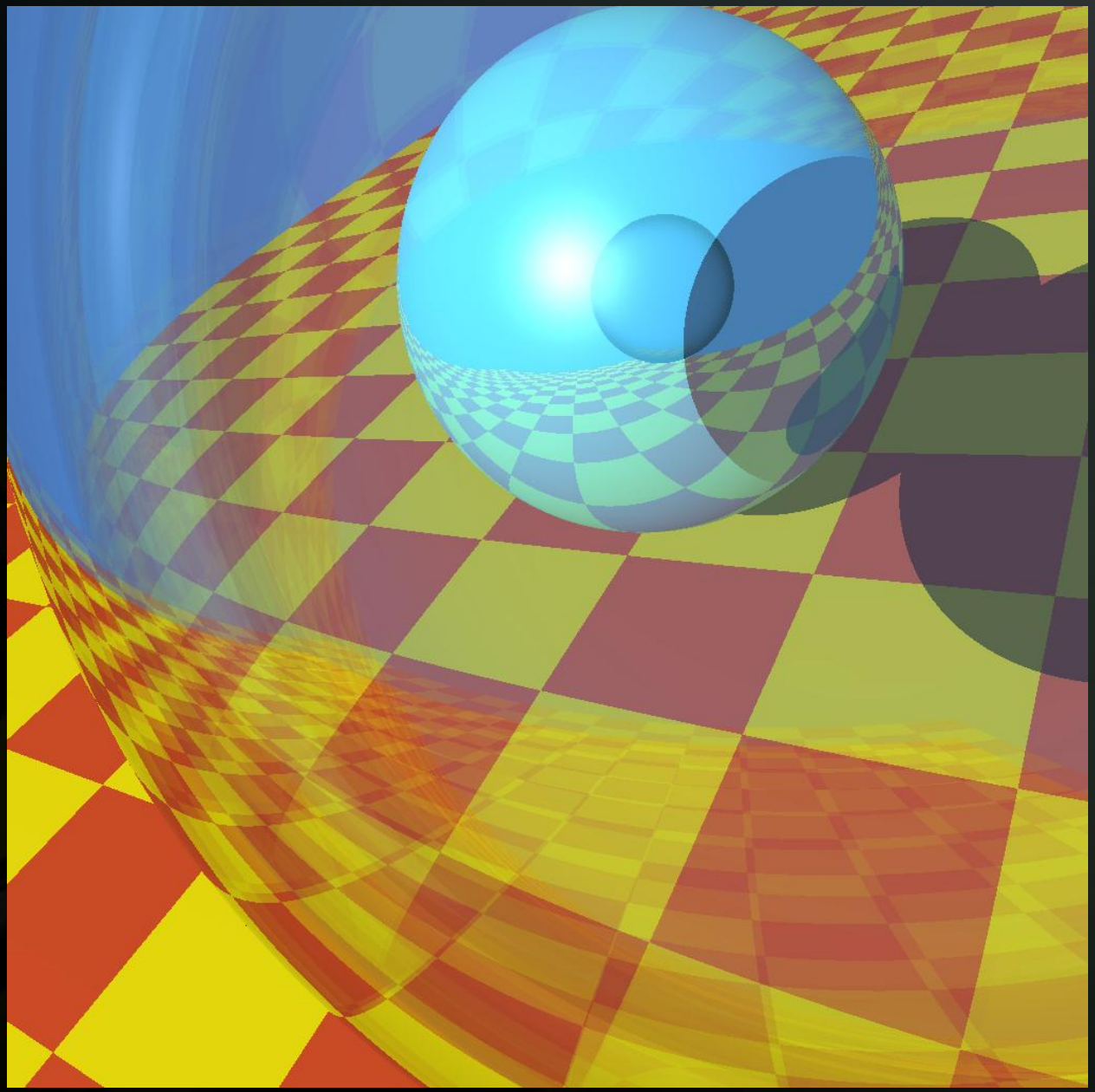






4 State Transitions to Finish

3 State Transitions to Finish

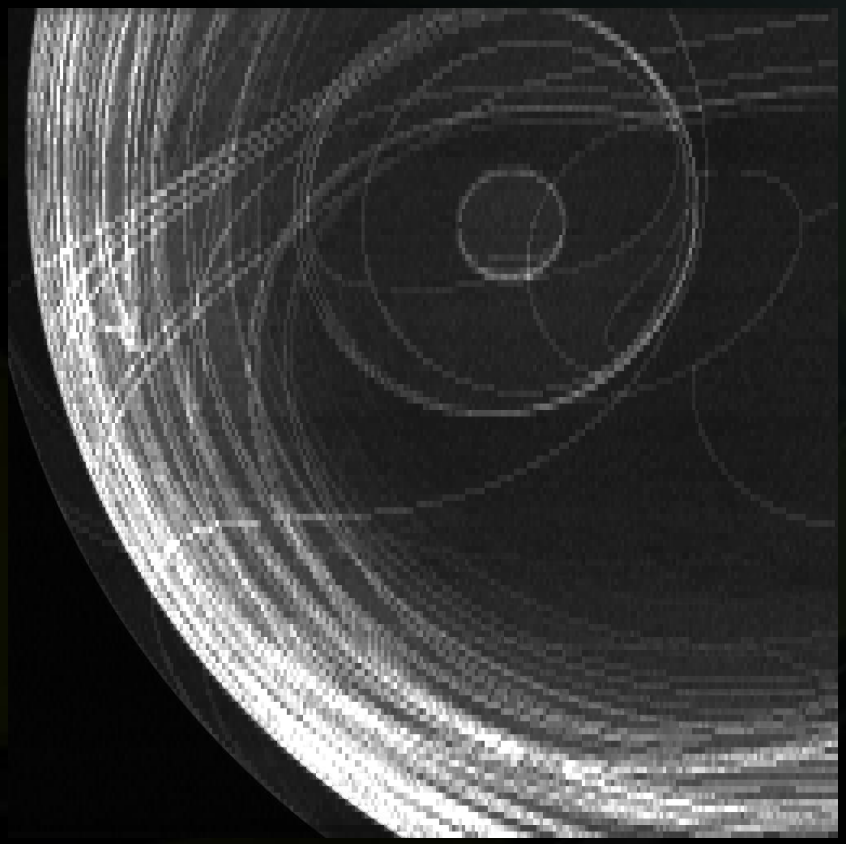


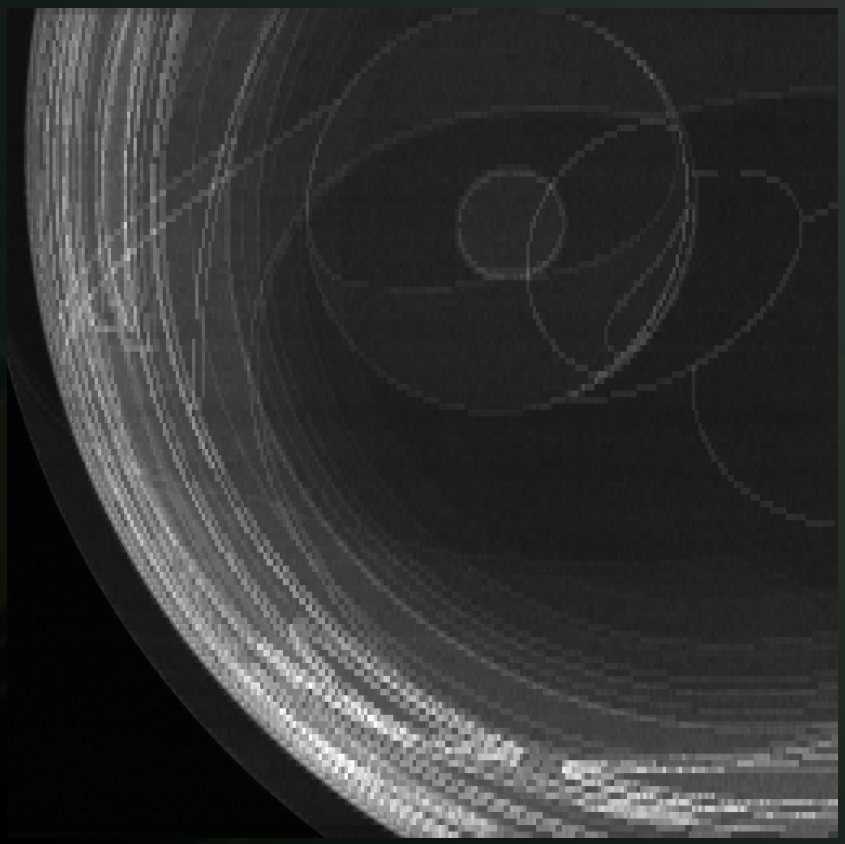
Per-pixel Render Time

Warp Synchronous

Prioritized

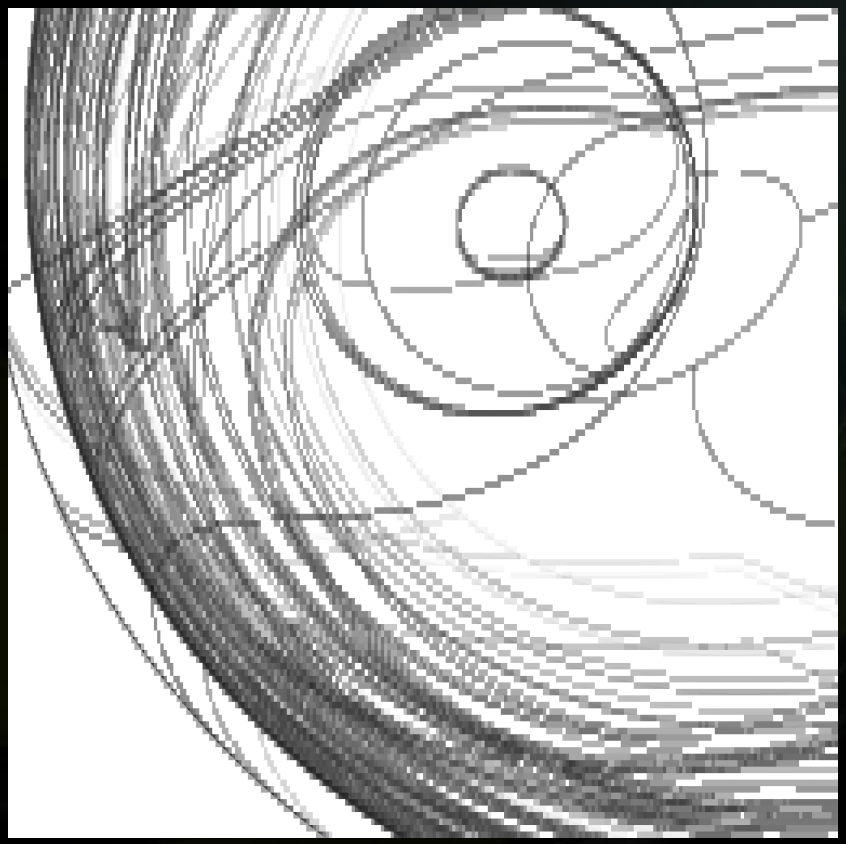
Frame rate: 1.25x





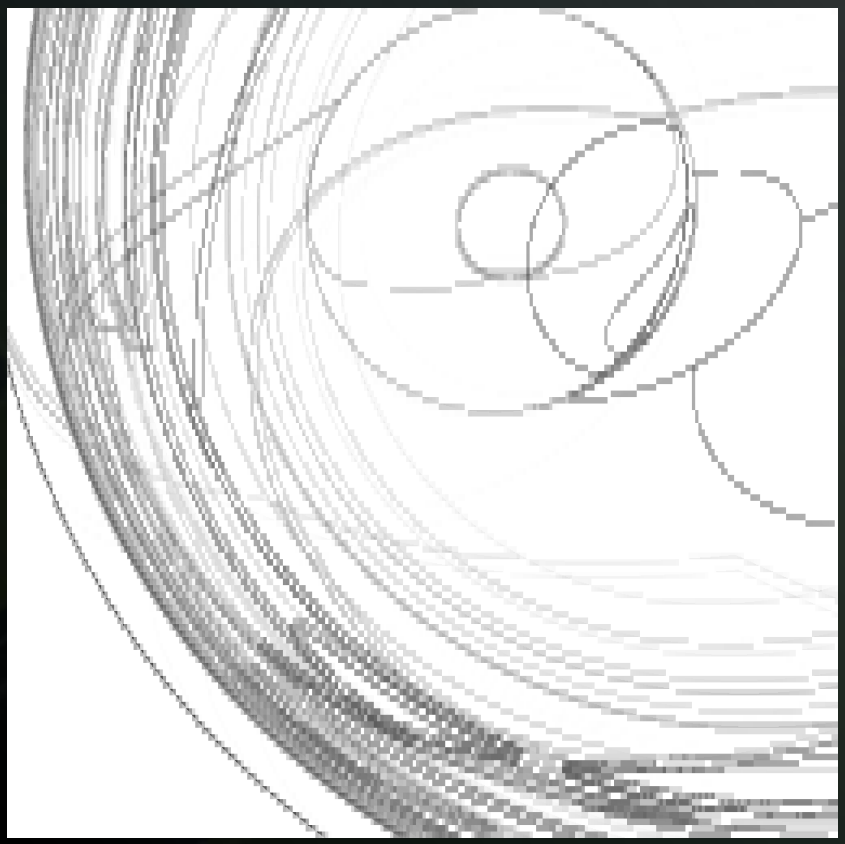
Warp Divergence

Warp Synchronous

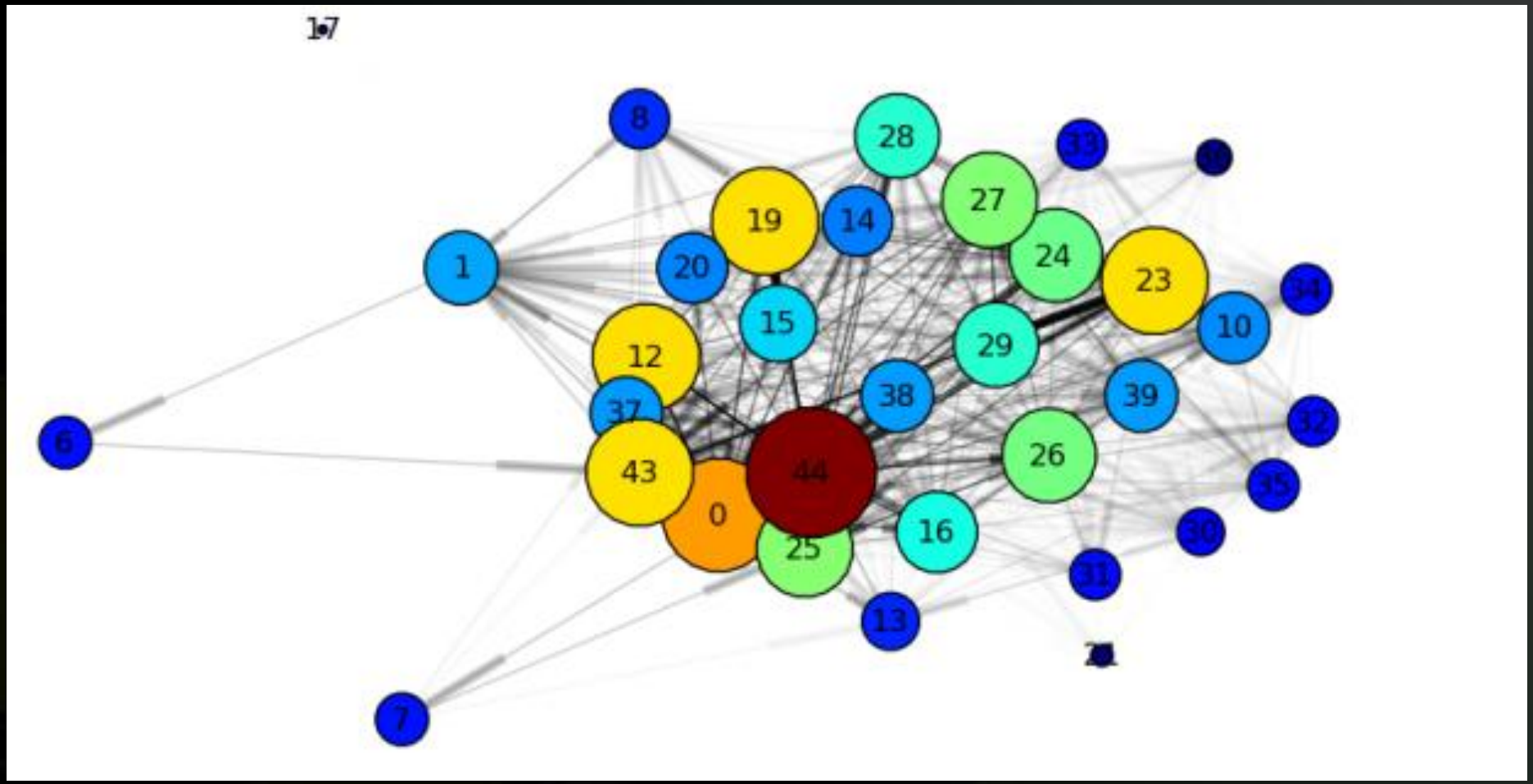


Prioritized

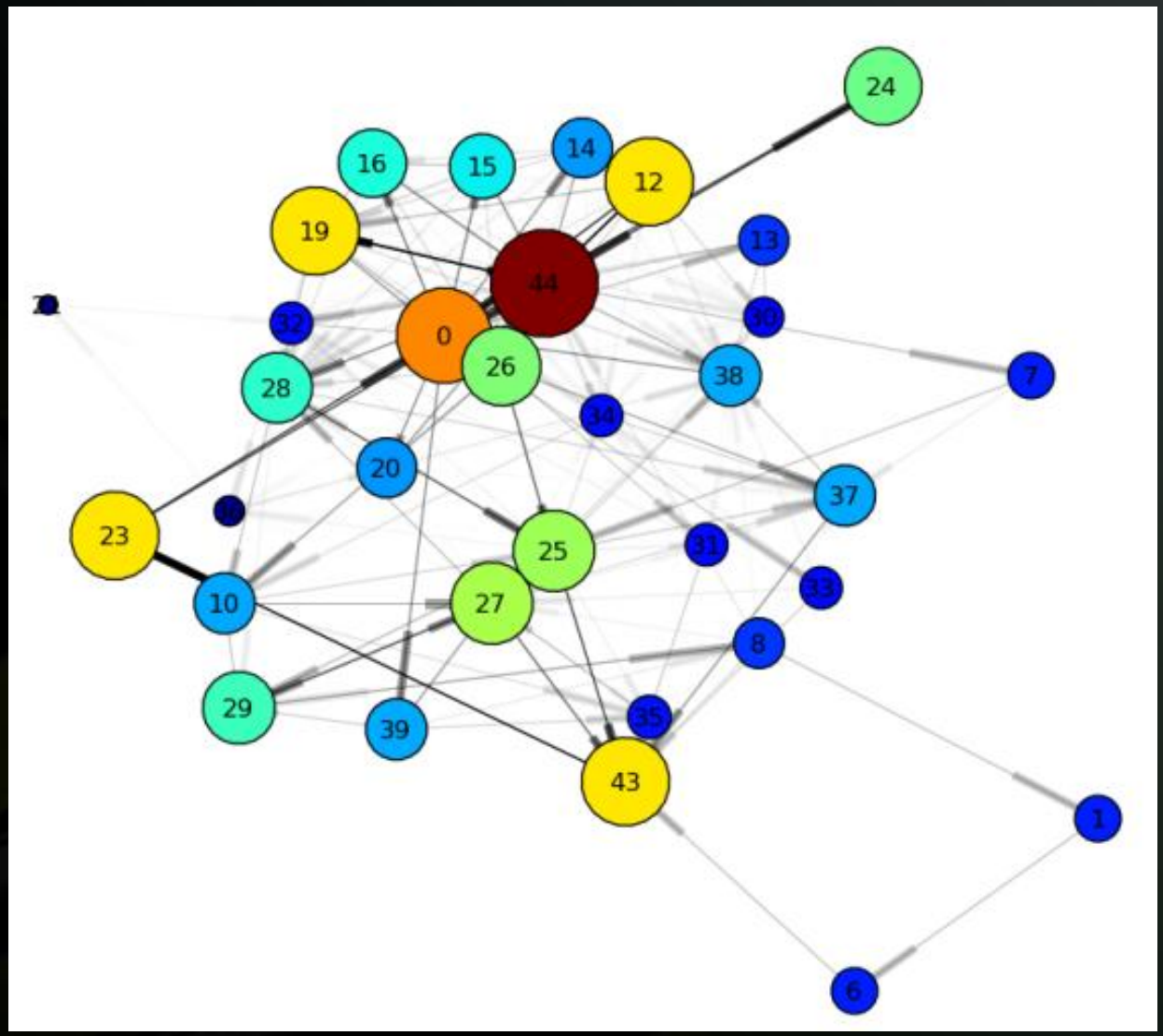
States executed: 0.74x



Warp Synchronous State History



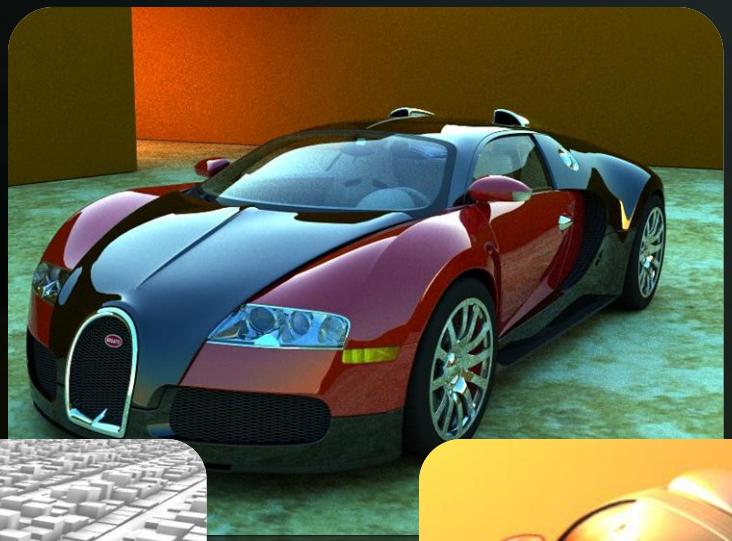
Prioritized State History

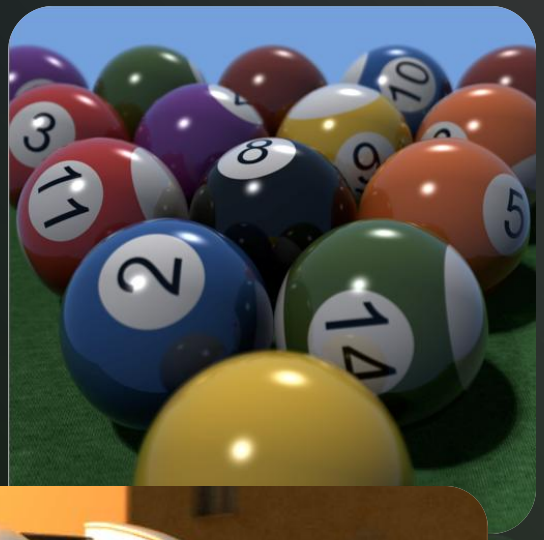


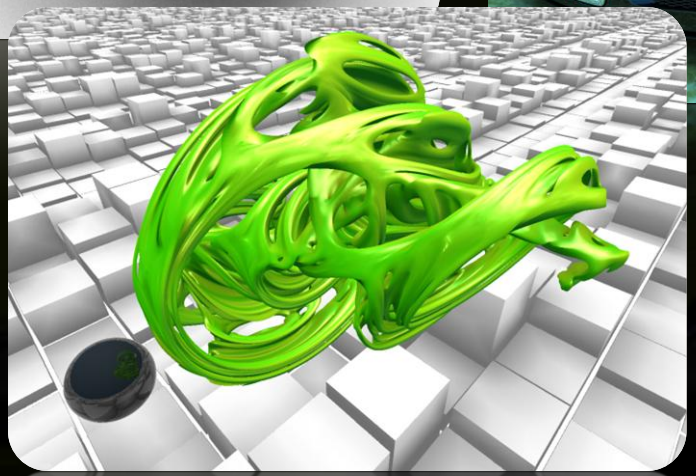

Warp Executions (Lower is Better)
Warp Synchronous
Prioritized
3.5E+06
3.0E+06
2.5E+06
2.0E+06
1.5E+06
1.0E+06
5.0E+05
0.0E+00
builtin function return_0
builtin exit_1
pinhole_camera_6
pinhole_camera_7
pinhole_camera_8
miss_10
intersect_12
intersect_13
intersect_14
intersect_15
intersect_16
intersect_17
intersect_19
intersect_20
intersect_21
intersect_23
intersect_24
closest_hit_radiance_25
closest_hit_radiance_26
closest_hit_radiance_27
closest_hit_radiance_28
closest_hit_radiance_29
any_hit_shadow_30
closest_hit_radiance_31
closest_hit_radiance_32
closest_hit_radiance_33
closest_hit_radiance_34
closest_hit_radiance_35
any_hit_shadow_36
closest_hit_radiance_37
closest_hit_radiance_38
closest_hit_radiance_39
traverse_geometry_43
traverse_geometry_44

OptiX SDK Release

Available to registered developers in early fall from http://www.nvidia.com







Go See Steve Parker's talk

Wednesday at **2:45**

SIGGRAPH Room 294

for more API information

and a short tutorial

Questions?

arobison@nvidia.com

http://www.nvidia.com