Interactive Ray Tracing on the GPU and NVIRT Overview

Presented at I3D'09

Austin Robison

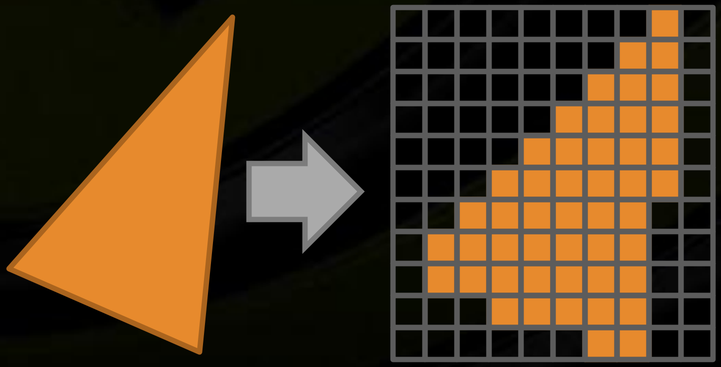
Rasterization & Ray Tracing

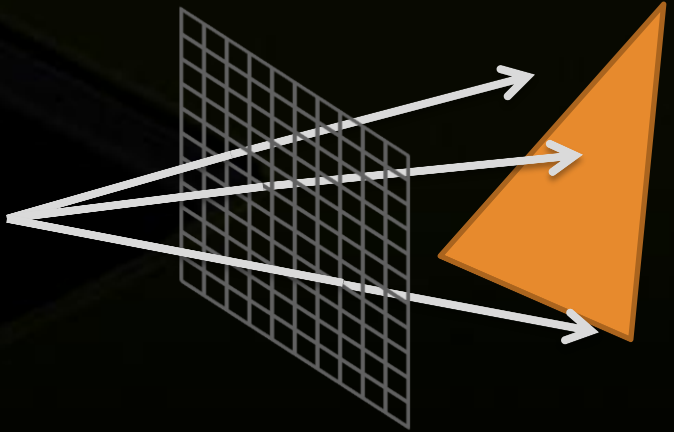
Rasterization

- For each triangle
  - Find the pixels it covers
  - For each pixel: compare to closest triangle so far

Classical Ray Tracing

- For each pixel
  - Find the triangles that might be closest
  - For each triangle: compute distance to pixel





Common Myths

Rasterization is linear in primitives

Ray Tracing is sublinear in primitives

- Rasterization uses LODs and occlusion query

Rasterization is sublinear in pixels

Ray Tracing is linear in pixels

- Ray Tracing uses packets and frustum culling

Rasterization is ugly

Ray Tracing is clean

- They’re both ugly

Rasterization vs. Ray Tracing

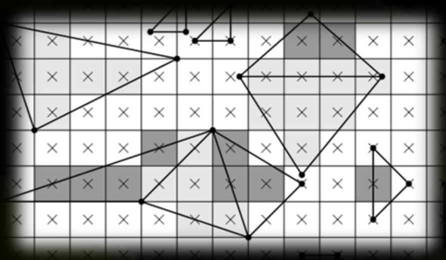
Rasterization

- \+ Fast
- – Needs cleverness to support complex visual effects

Ray Tracing

- \+ Robustly supports complex visual effects
- – Needs cleverness to be fast

Interactive Hybrid Rendering



100% Rasterization

100% Ray Traced

Sweet Spots

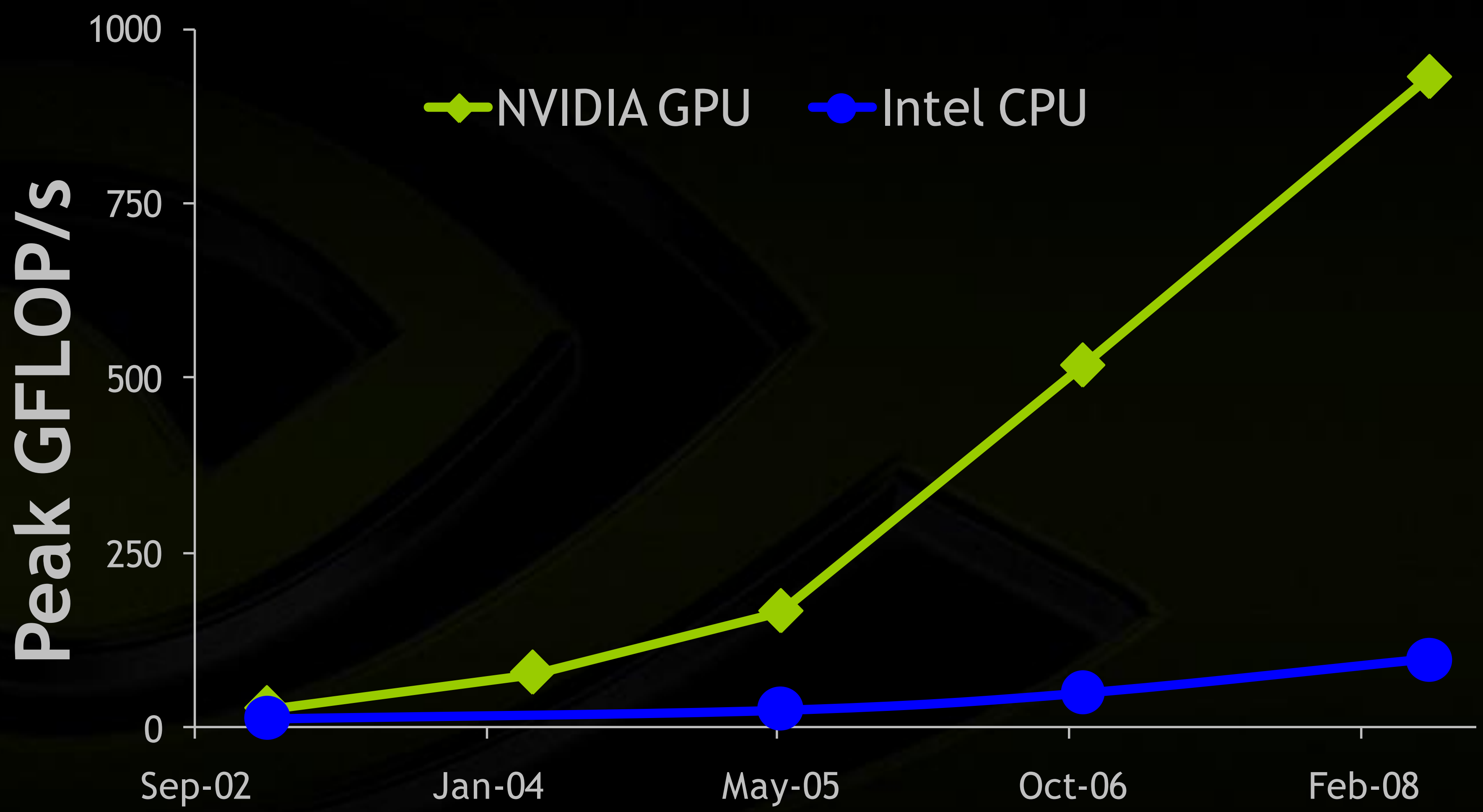
Industrial Strength Ray Tracing

- mental images is market leader for physically correct ray tracing software
- Applicable in numerous markets: automotive, design, architecture, film

Why GPU Ray Tracing?

- Abundant parallelism, massive computational power
- GPUs excel at shading
- Opportunity for hybrid algorithms

GPUs are fast and are getting faster



NVIDIA SIGGRAPH 2008 Demo

- **NVSG-driven animation and interaction**
- **Programmable Shading**
- **Modeled in Maya, imported via COLLADA**
- **Fully Ray Traced**

2 million polygons
Bump-mapping
Movable light source
5 bounce reflection/refraction
Adaptive antialiasing

Introducing…

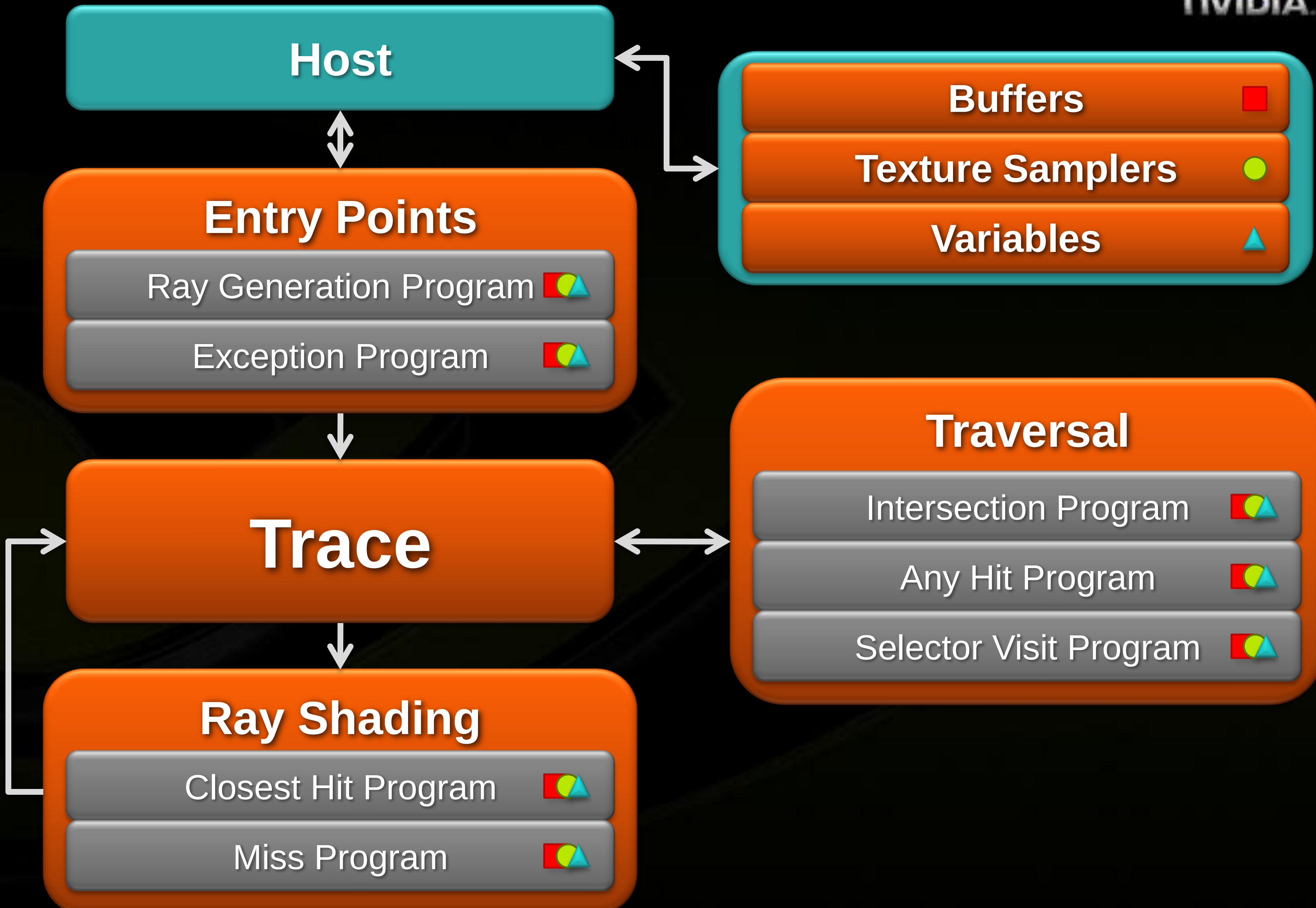
NVIRT

The NVIDIA Interactive Ray Tracing API

NVIRT Design Goals

- Low Level, High Performance API
  - NVIRT is *not* a renderer
  - Can be used for rendering, baking, collision detection, AI queries, etc.
- Programmability
  - In addition to programmable surface shading, provide programmable ray generation, intersection, etc.
  - Program as if it were single ray code (no packets)
- Abstract traversal implementation
  - The best way to write a ray tracer may change on different generations of hardware
  - Automated parallelization

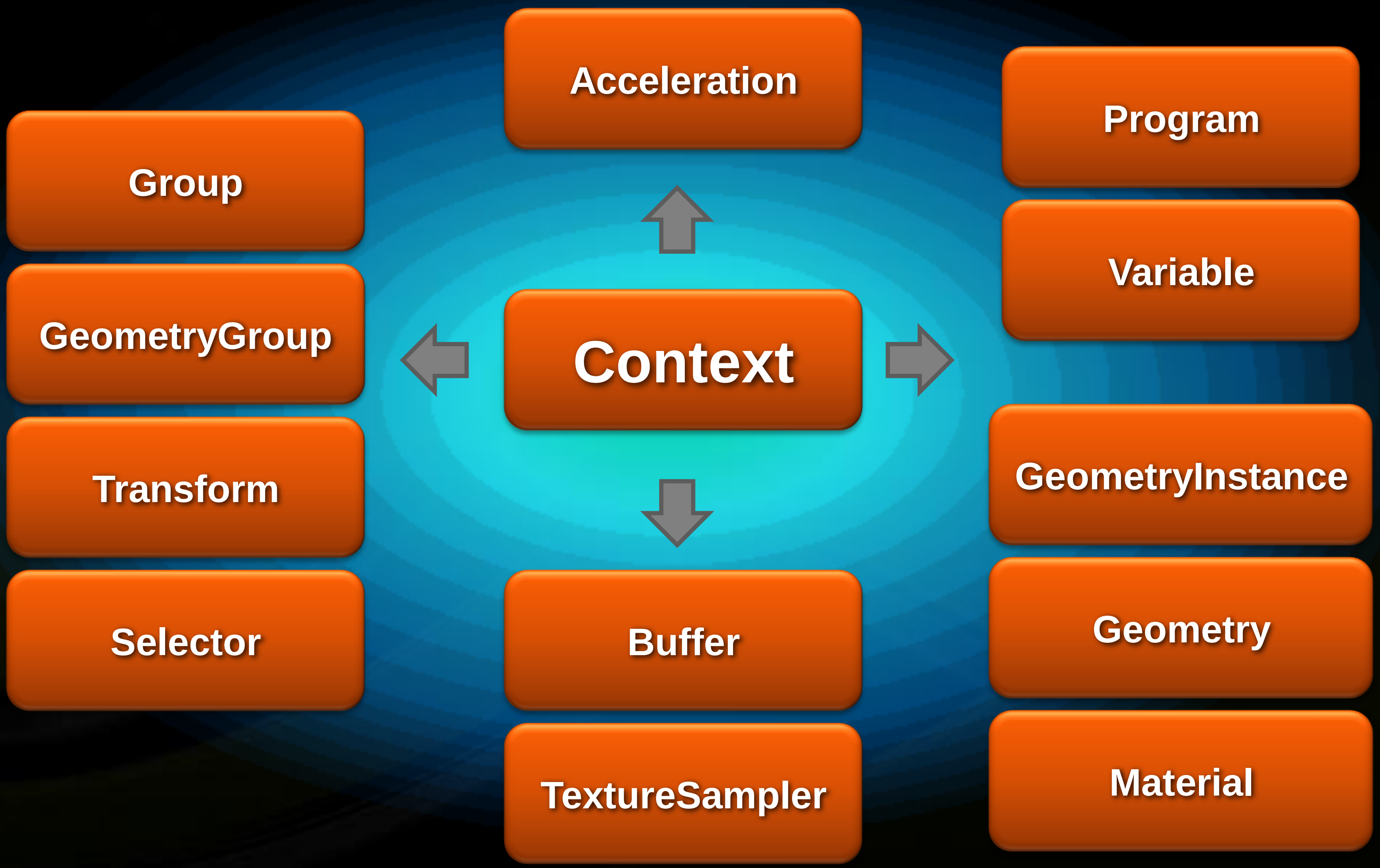
The Ray Tracing Pipeline



Closest Hit and Any Hit Programs

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

Overview – API Objects



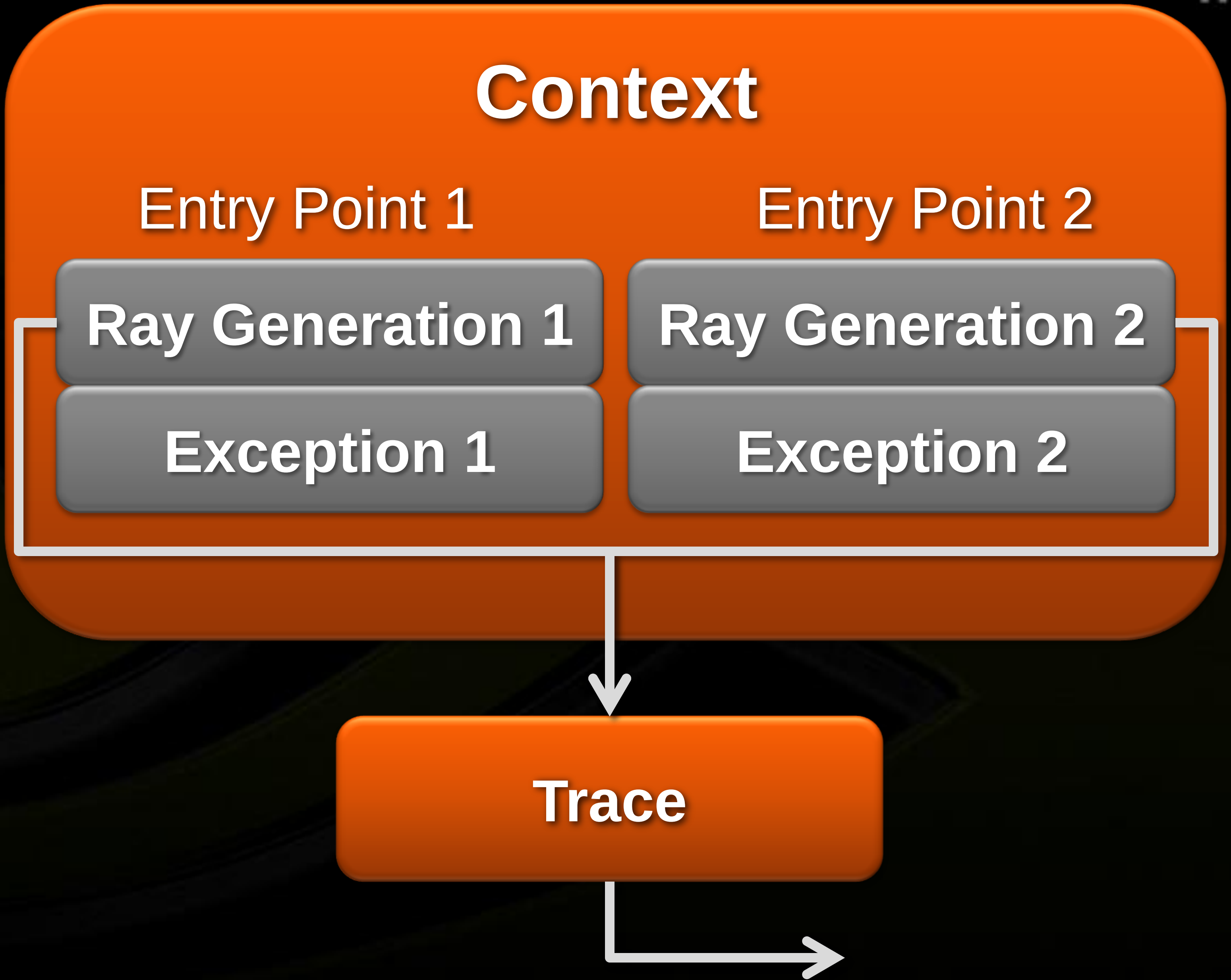
API Objects – Context

- Manages API Object State
  - Program Loading
  - Validation and Compilation
- Manages Acceleration Structures
  - Building and Updating
- Provides Entry Points into the system
  - rtContextTrace1D()
  - rtContextTrace2D()
  - rtContextTrace3D()

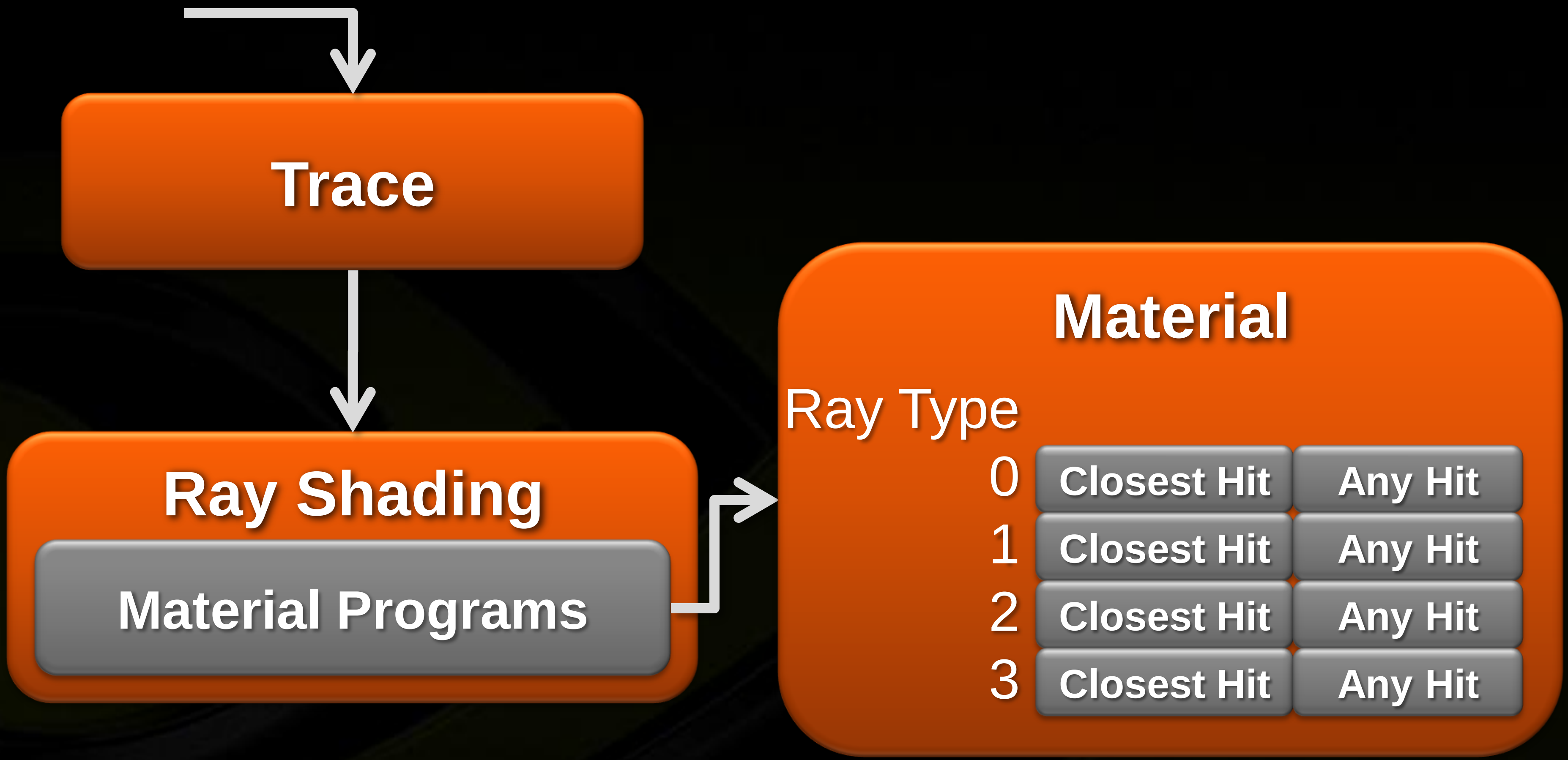
Context

Ray Gen Programs
Exception Programs
Miss Programs
User Variables

Entry Points and Ray Types



Entry Points and Ray Types Cont'd



API Objects – Nodes

- Nodes contain children
  - Other nodes
  - Geometry instances
- Transforms hold matrices
  - Applied to all children
- Selectors have Visit programs
  - Provide programmable selection of children
  - Similar to “switch nodes”
  - Can implement LOD systems
- Acceleration Structures
  - Builds over children of attached node

Group

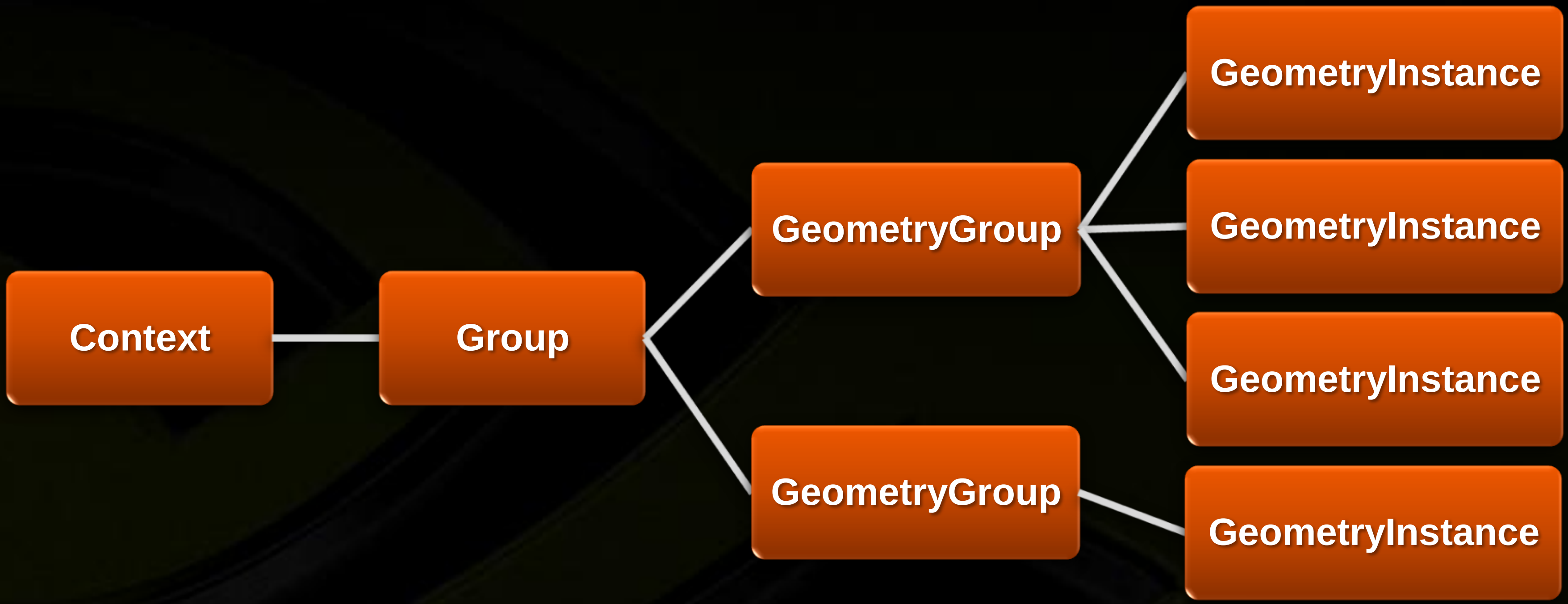
GeometryGroup

Transform

Selector

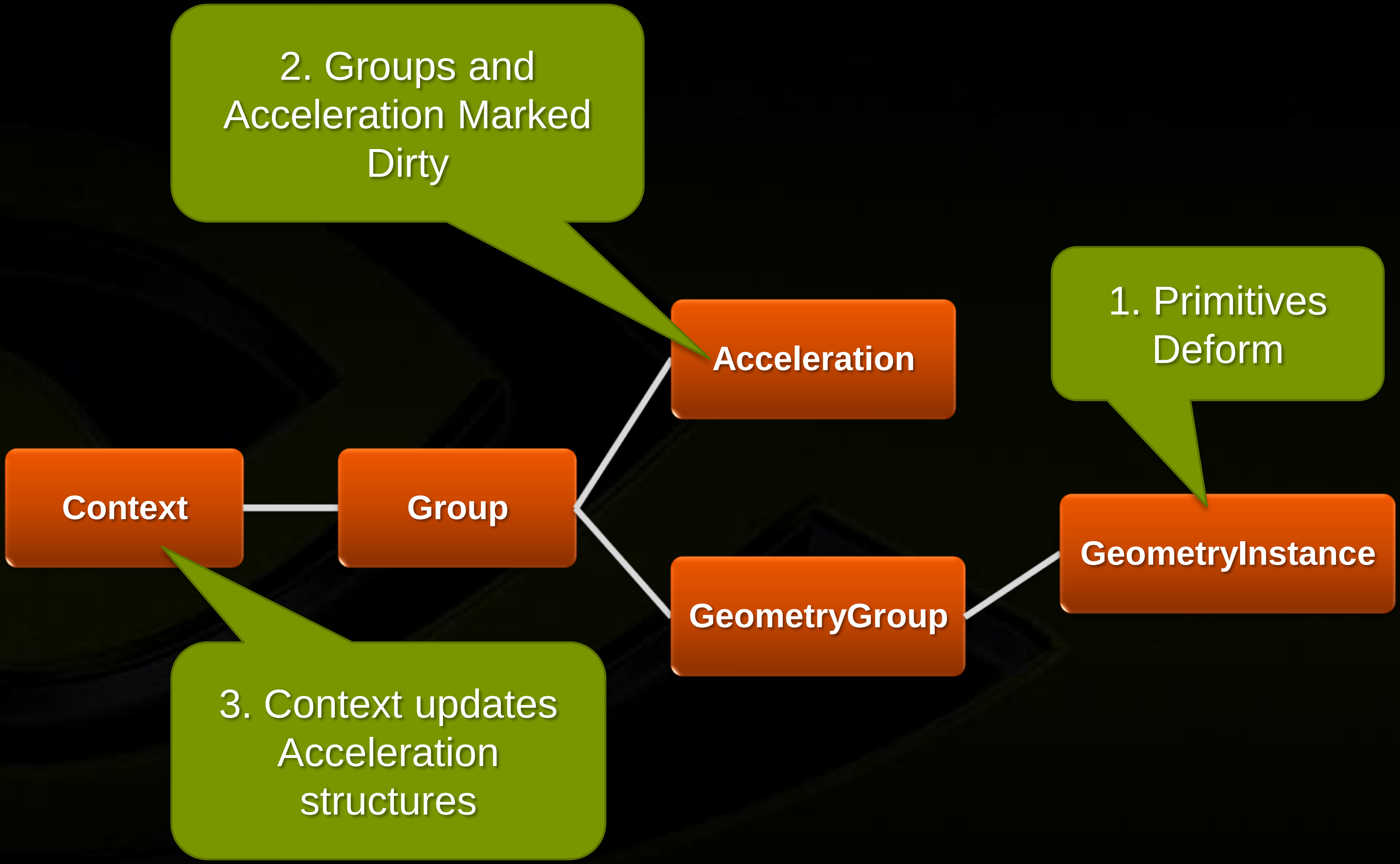
Acceleration

The Object Hierarchy



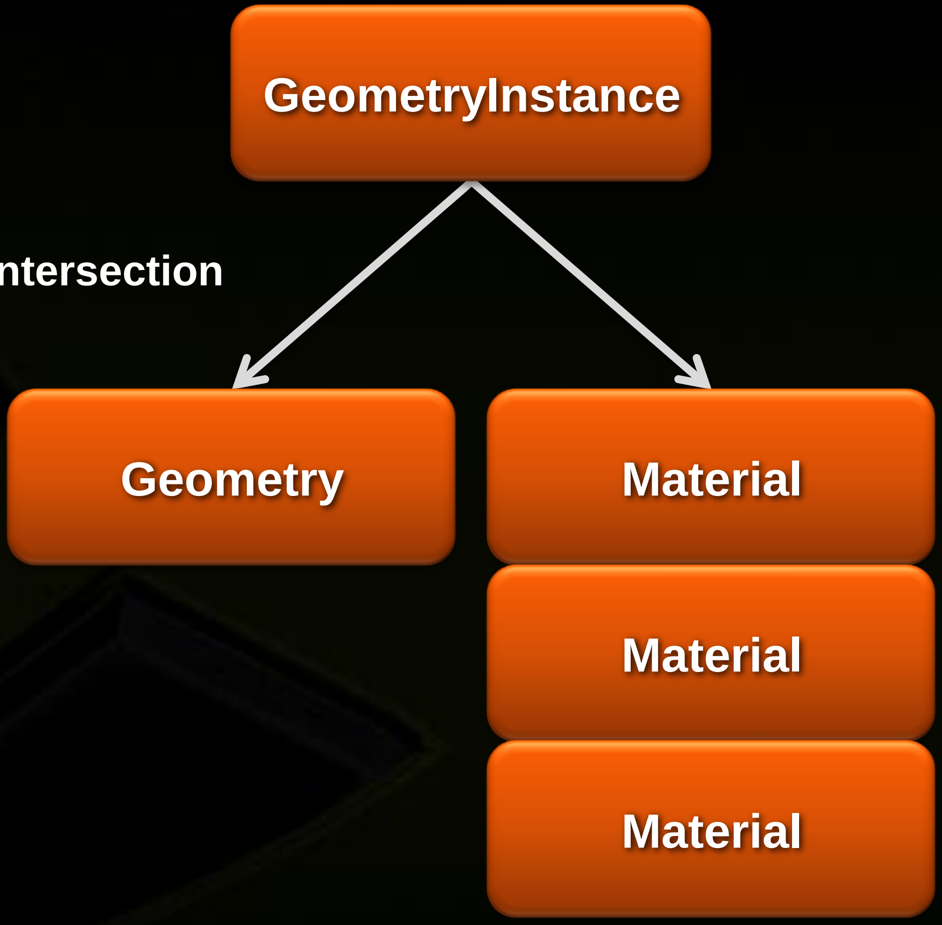
Not a scene graph!

Deformable Objects



API Objects – Geometry

- GeometryInstance references:
  - Geometry object
  - A collection of Materials
    - Indexed by argument from intersection
- Geometry
  - A collection of primitives
  - Intersection Program
  - Bounding Box Program
- Material
  - Any Hit Program
  - Closest Hit Program

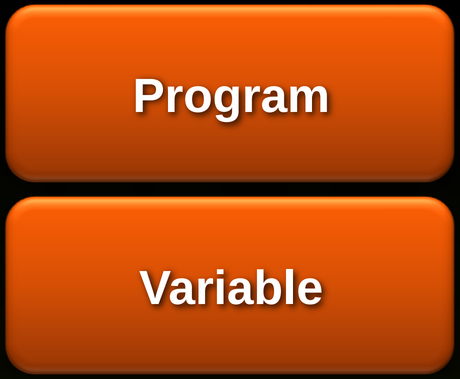


API Objects – Data Management

- Supports 1D, 2D and 3D buffers
- Buffer formats
  - RT_FORMAT_FLOAT3
  - RT_FORMAT_UNSIGNED_BYTE4
  - RT_FORMAT_USER
  - etc.

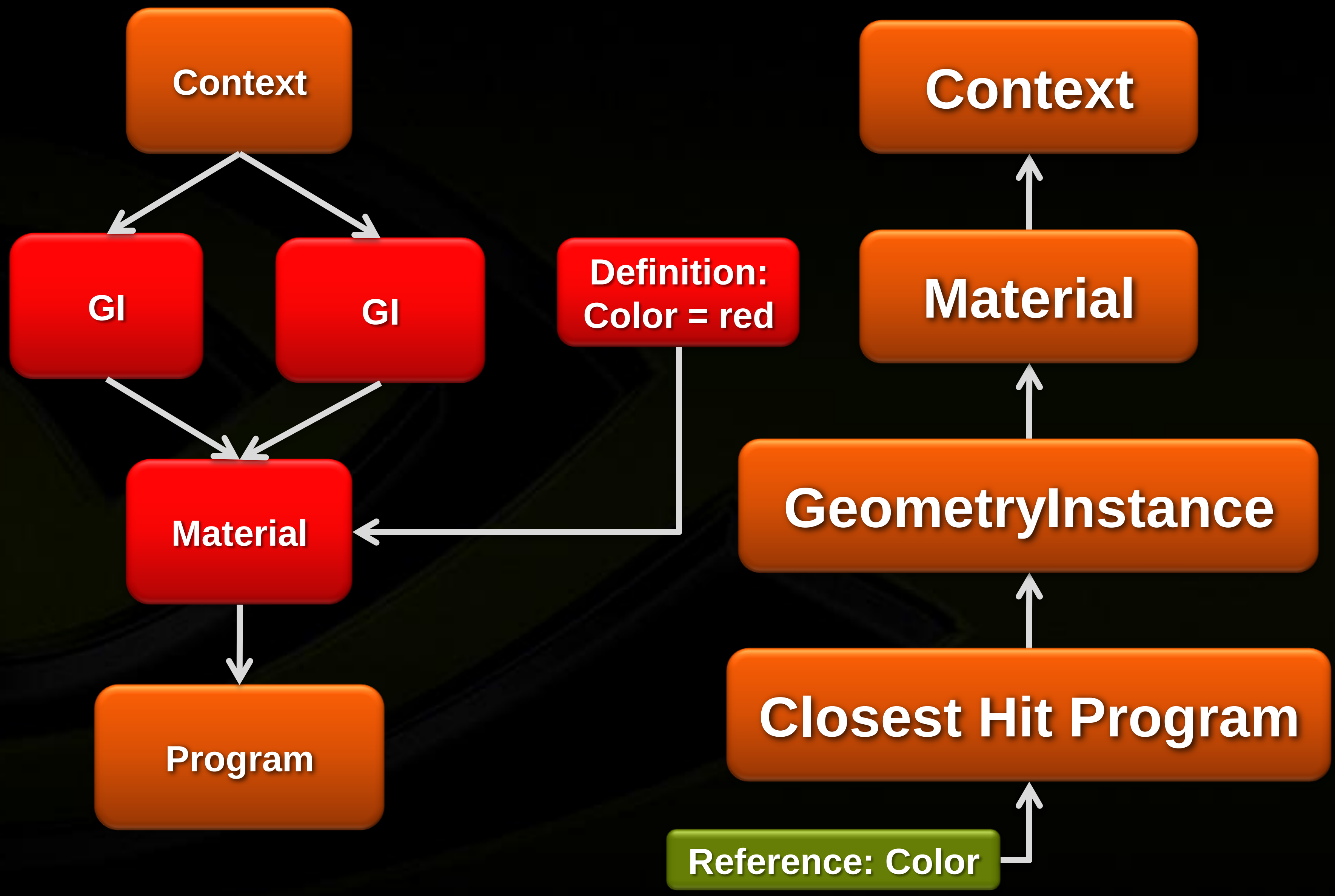
- 3D API Interoperability
  - e.g. create buffers from OpenGL buffer objects
- TextureSamplers reference Buffers
  - Attach buffers to MIP levels, array slices, etc.

API Objects – Programmability

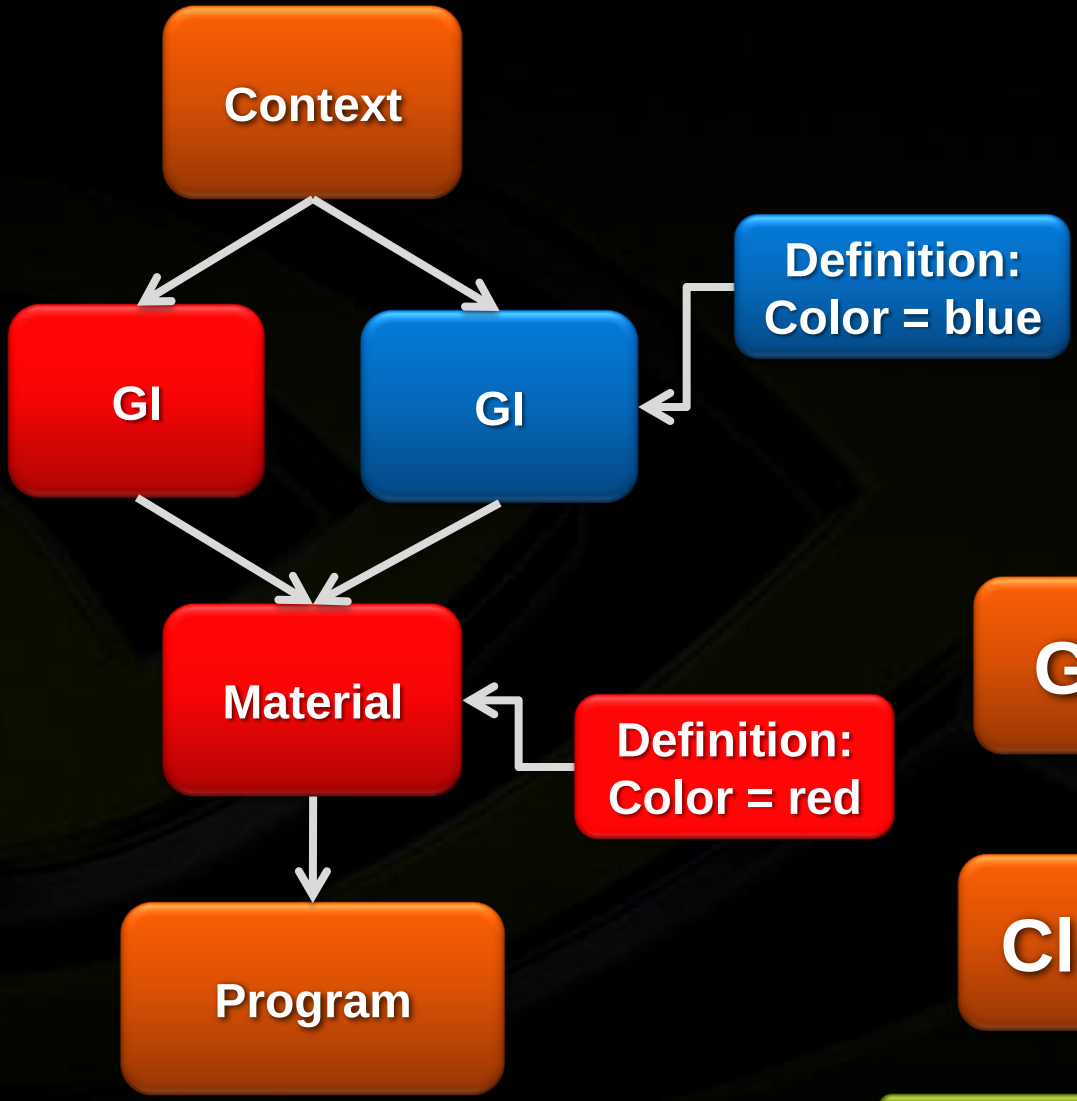


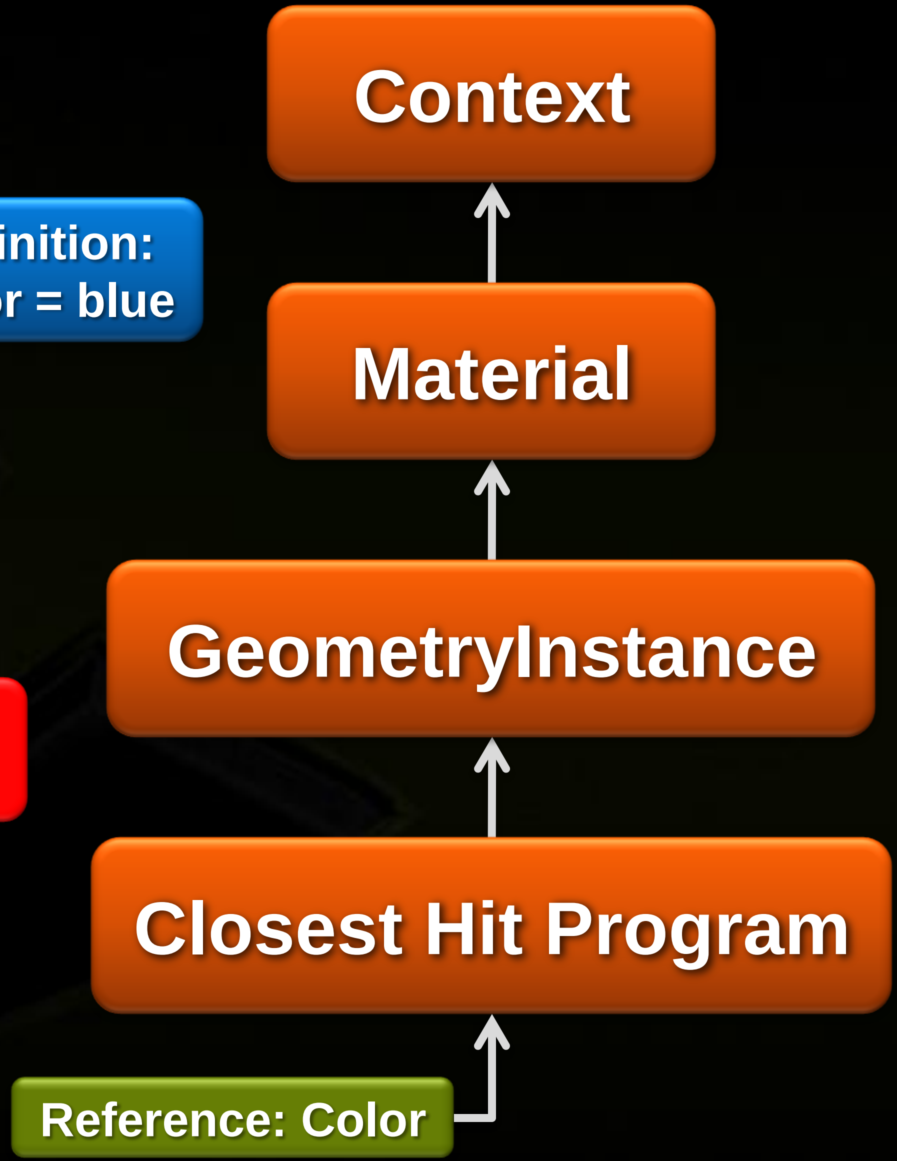
- Runs on CUDA
  - Cg-like vectors plus pointers
  - Uses CUDA virtual assembly language
  - C wrapper for use with NVCC compiler
- Implements recursion and dynamic dispatch
  - Intrinsic functions: rtTrace(), rtReportIntersection(), etc.
- Programs reference variables by name
- Variables are defined by
  - Static initializers
  - Binding to API Objects in the hierarchy

Variable Scoping Rules



Variable Scoping Rules Cont'd





Per Ray Data and Attributes

- Per Ray Data
  - User-defined struct attached to rays
  - Can be used to pass data up and down the ray tree
  - Varies per Ray Type
- Arbitrary Attributes
  - Produced by Intersection Programs
  - Consumed by Any Hit and Closest Hit Programs

Program Example – Pinhole Camera

```
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()
{
  uint2 screen = output_buffer.size();
  uint2 index =
   make_uint2(rayIndex.get());

  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;
}
```

Program Example - Attributes

Sphere Intersection

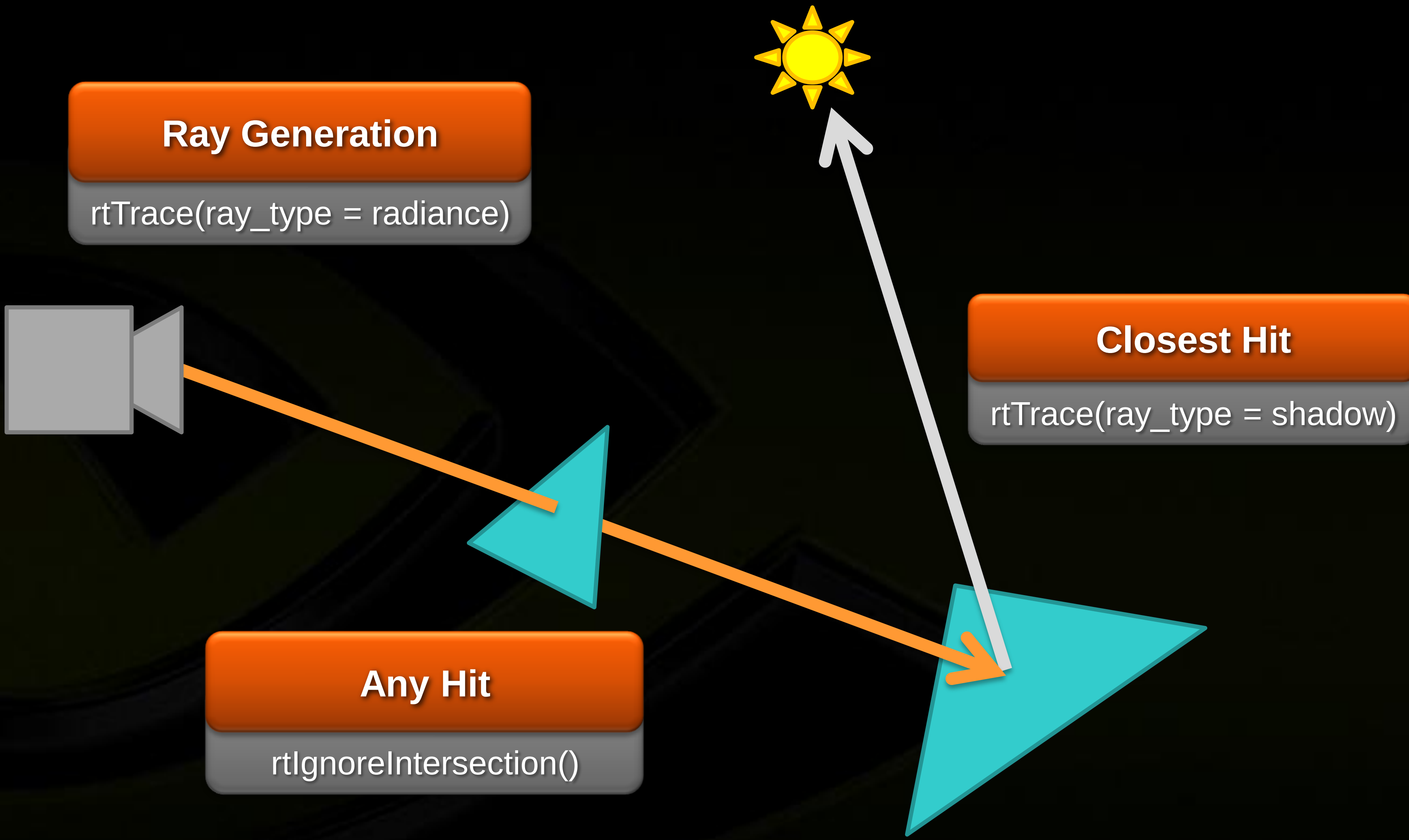
```
rtDeclareAttribute(float3, normal);
RT_PROGRAM void intersect(int primIdx)
{
  …
  if(rtPotentialIntersection( root1 ) )
   {
      normal = (O + root1*D)/radius;
      if(rtReportIntersection(0))
   }
  …
}
```

Normal Visualization Shader

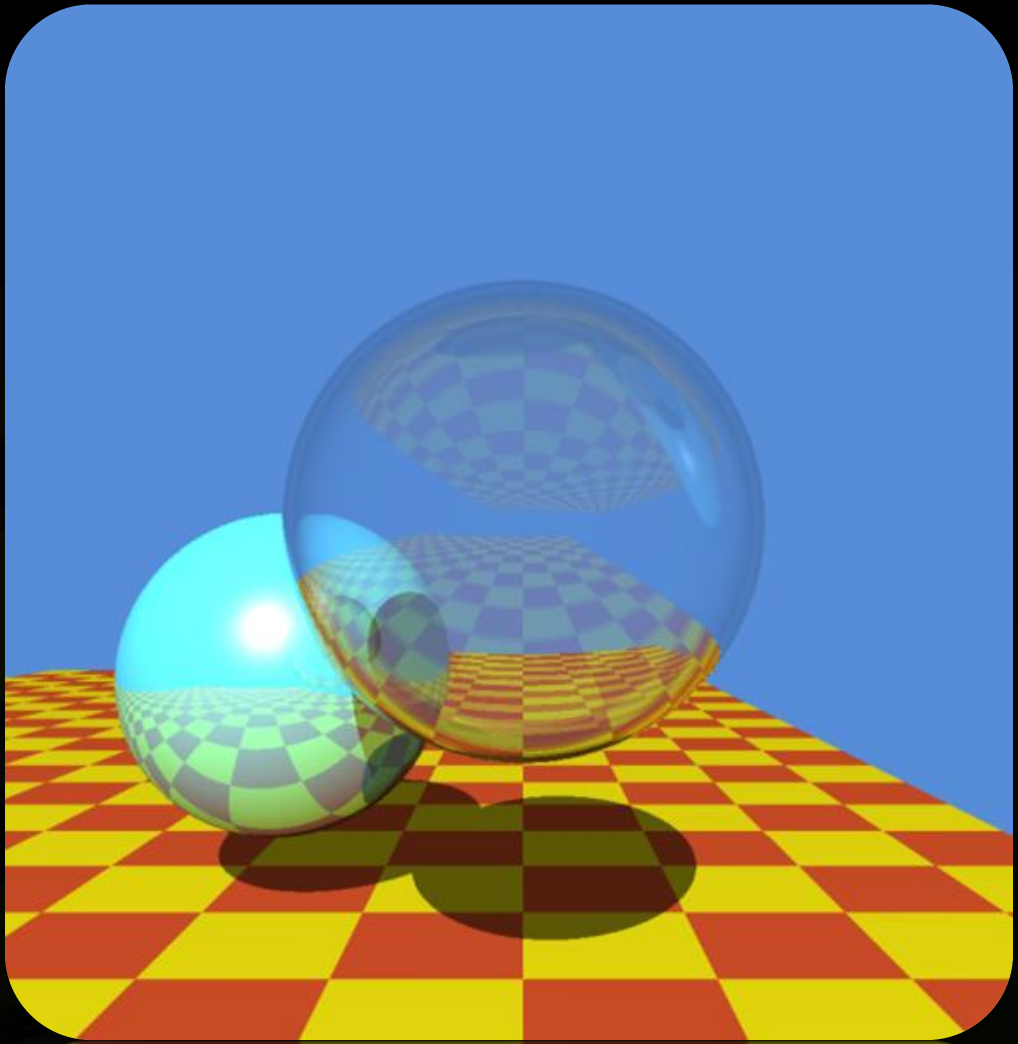
```
rtDeclareAttribute(float3, normal);
rtDeclareRayData(PerRayData_radiance,
    prd_radiance);

RT_PROGRAM void closest_hit_radiance()
{
  PerRayData_radiance& prd =
    prd_radiance.reference();
  prd.result = normal*0.5f + 0.5f;
}
```

Execution Flow



An Example – Whitted's Scene



Whitted's Scene – Context Setup

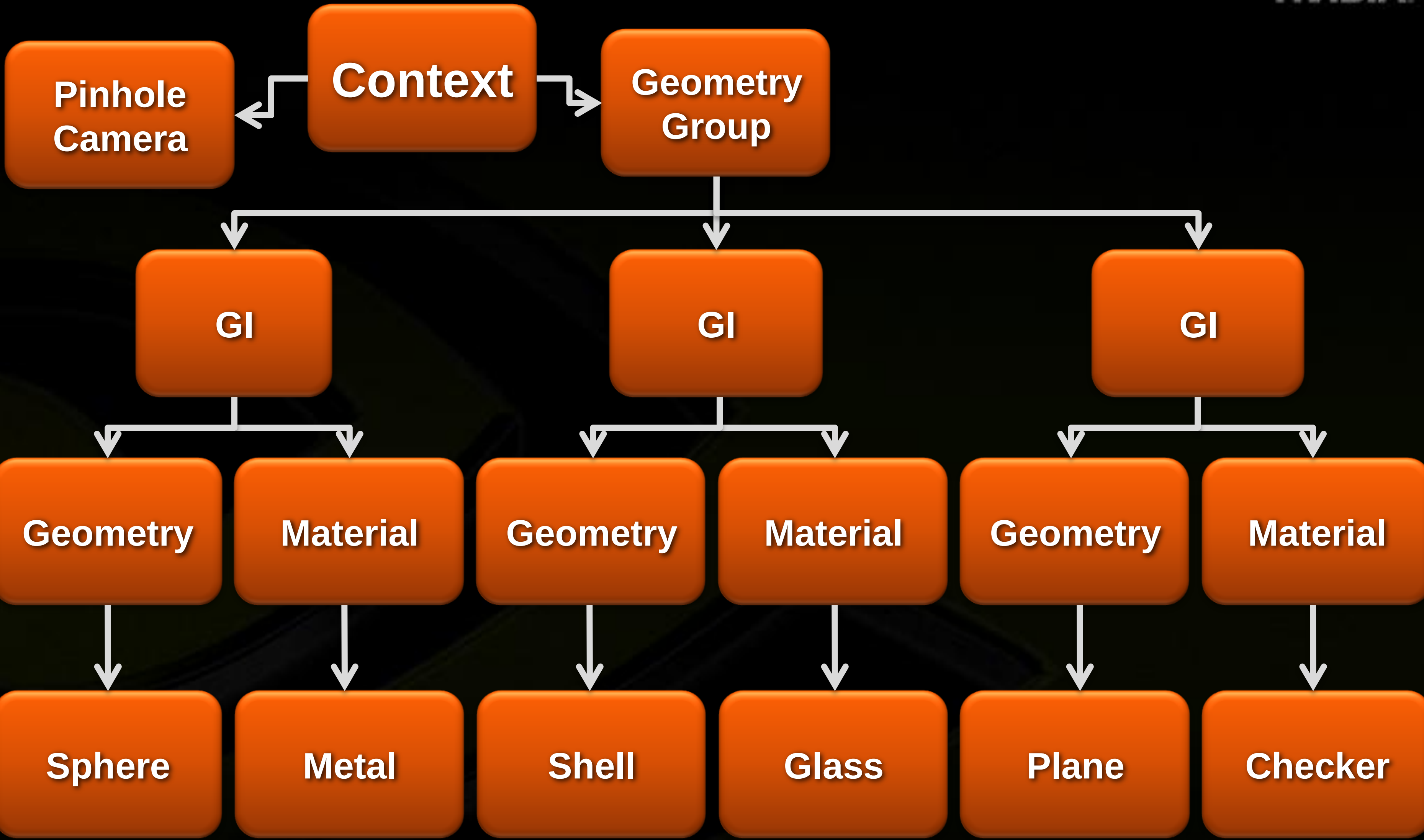
```
struct PerRayData_radiance
{
  float3 result;
  float  importance;
  int    depth;
};

struct PerRayData_shadow
{
  float  attenuation;
};
```

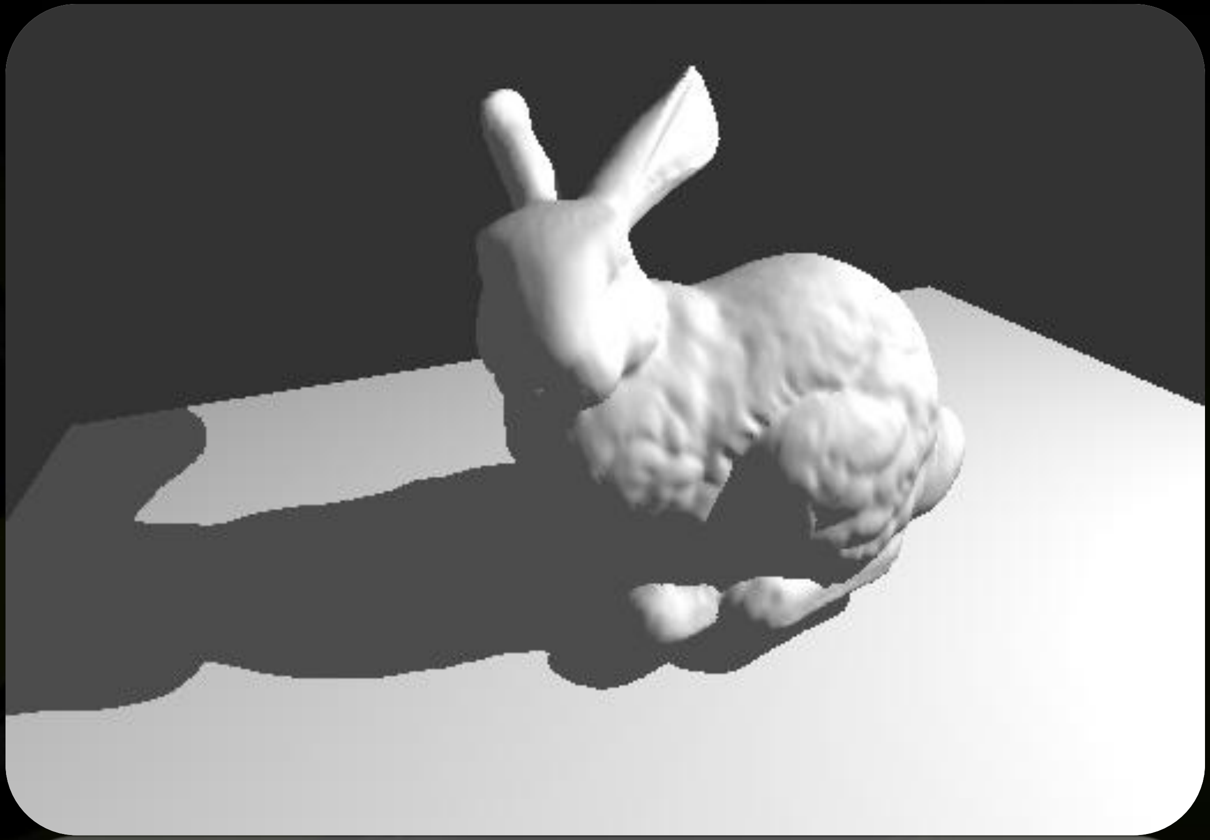
Context

Num. Ray Types = 2
Num. Entry Points = 1

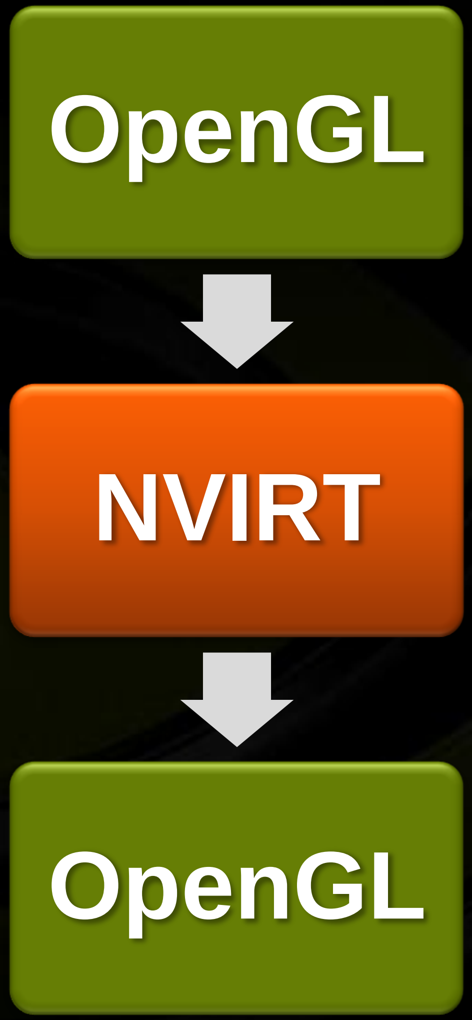
Whitted's Scene – Object Hierarchy



An Example – Hybrid Hard Shadows



Hybrid Hard Shadows - Pipeline



1. Rasterize shadow ray requests with OpenGL
2. Trace shadow rays against scene geometry
3. Use NVIRT output during OpenGL shading

Hybrid Hard Shadows – Ray Generation Program

- Rasterize world space positions to FBO
- Send NVIRT output to texture and render

```
RT_PROGRAM void shadow_request()
{
  uint2 index = make_uint2(ray_index.get());
  float3 ray_origin = request_buffer[index];
  PerRayData_shadow prd;
  prd.intensity = 1;
  if( !isnan(ray_origin.x) ) {
    float3 ray_direction = normalize(light_pos-ray_origin);
    Ray ray = make_ray(ray_origin, ray_direction, shadow_ray_type,
   scene_epsilon, RT_DEFAULT_MAX);
    rtTrace(shadow_casters, ray, prd);
  }
  shadow_buffer[index] = prd.intensity;
}
```

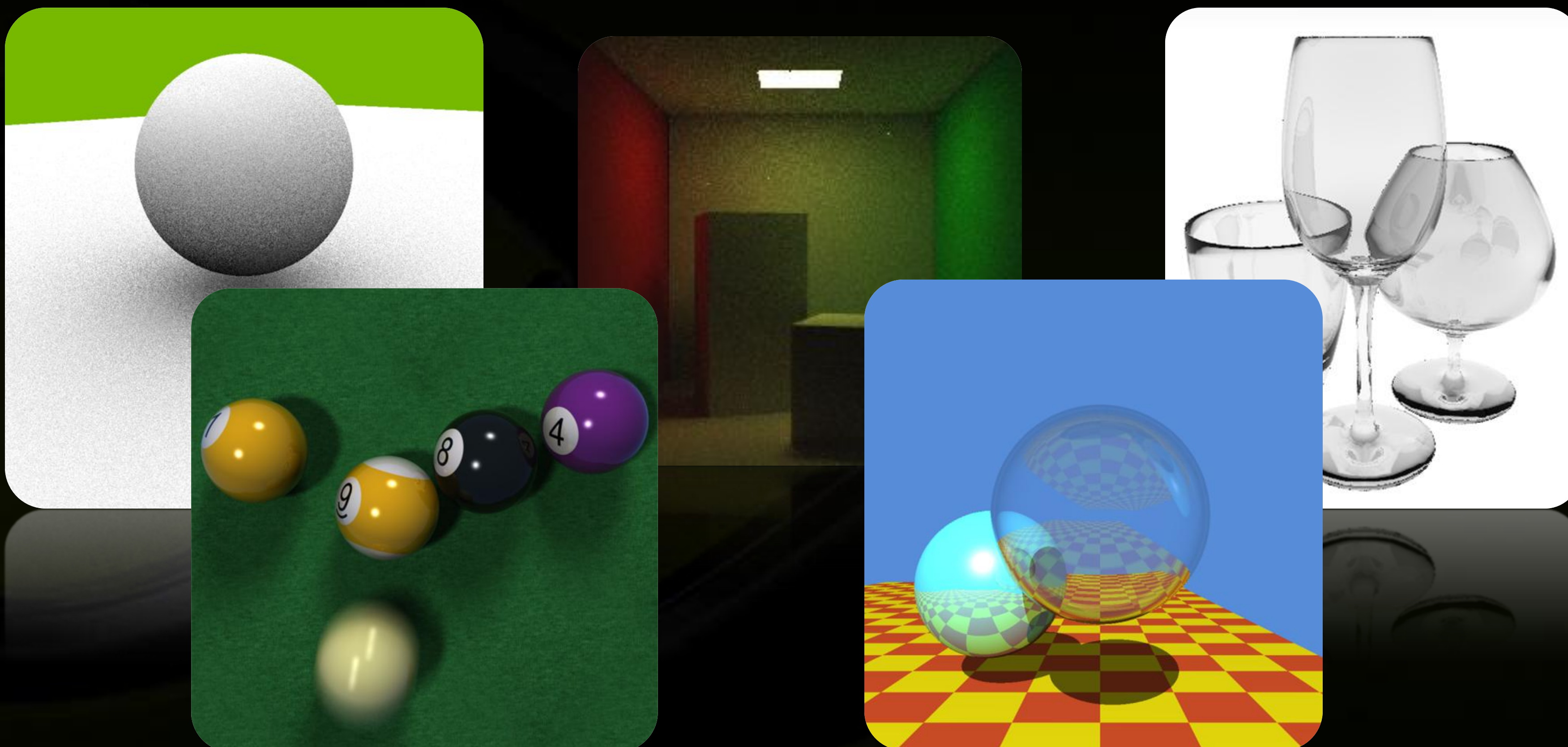
NVIRT Wrap-up

- NVIRT is not a renderer
  - Can but used to implement a renderer, collision detection, baking, etc.
- Programmable Ray Tracing Pipeline
  - Intersection
  - Shading
  - Traversal
- Abstract Tracing mechanism can take advantage of future NVIDIA hardware
  - No need to change your code

NVIRT SDK Public Beta

Available this spring from http://www.nvidia.com

Next NVSG release will include NVIRT based renderer



Questions?

arobison@nvidia.com

http://www.nvidia.com