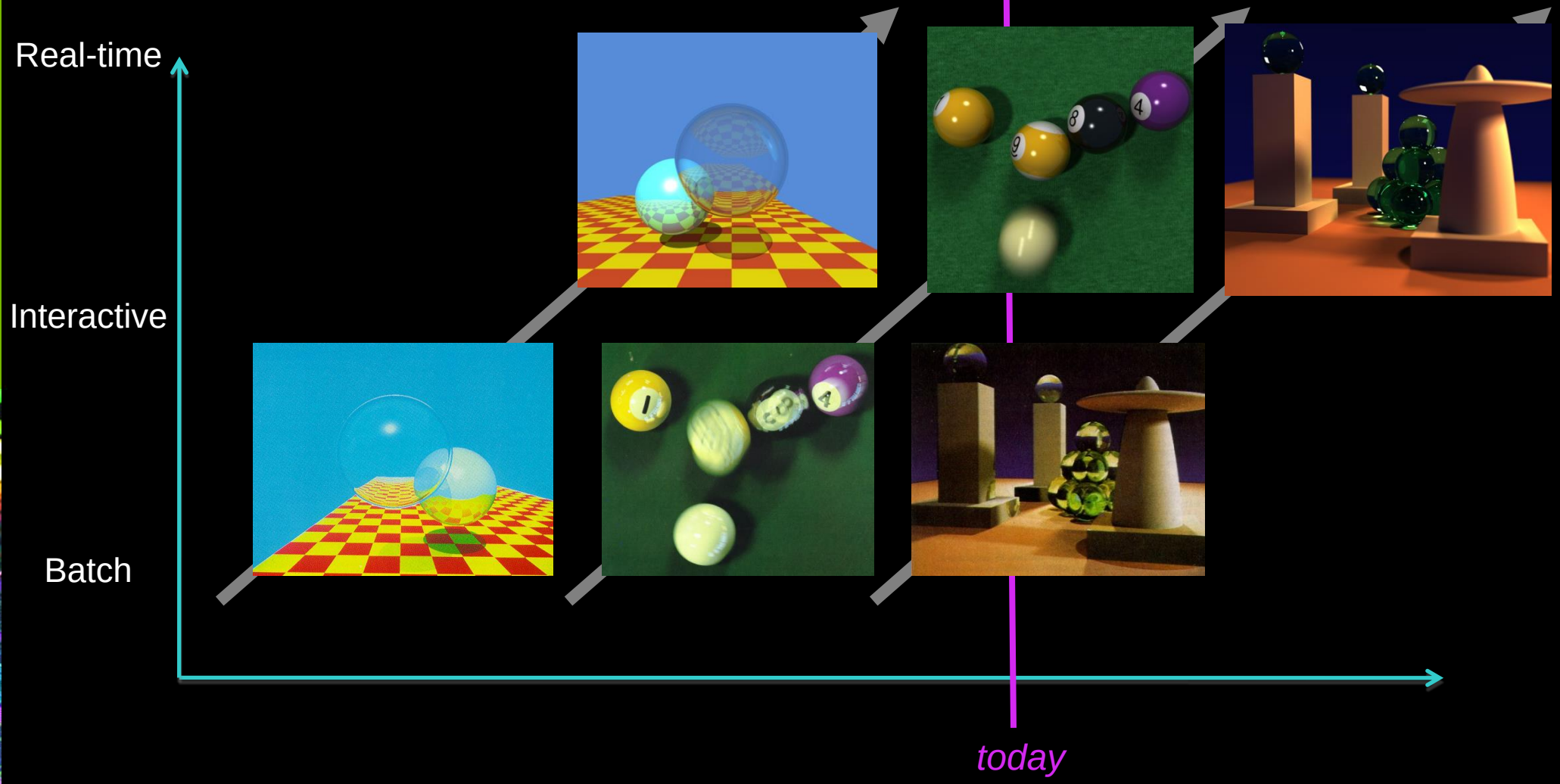
GPU TECHNOLOGY CONFERENCE

Optimizing OptiX Applications

David McAllister
and James Bigler

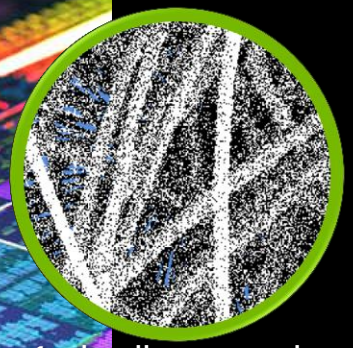
Ray Tracing Regimes



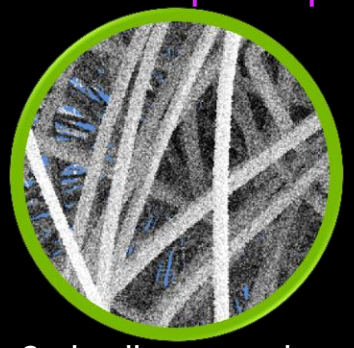
Real Time Path Tracing

- What would it take?
  - 4 rays / sample
  - 50 samples / pixel
  - 2M pixels / frame
  - 30 frames / second
  - 12B rays / second
- GeForce GTX 680: 350M rays / second
  - Need 35X speedup

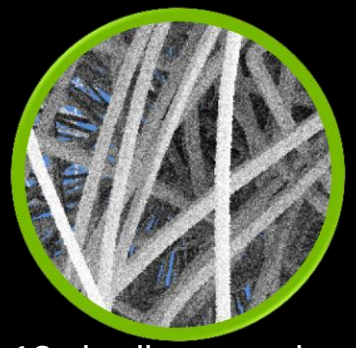
Good enough for games



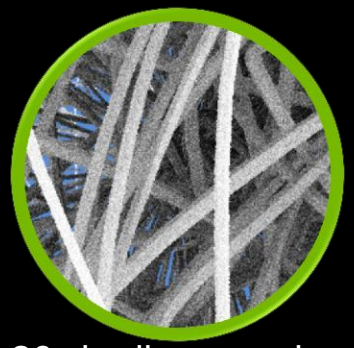
1 shading sample
1 AA sample



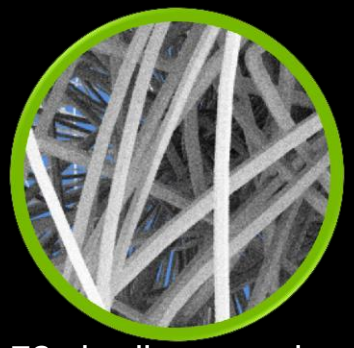
9 shading samples
1 AA sample



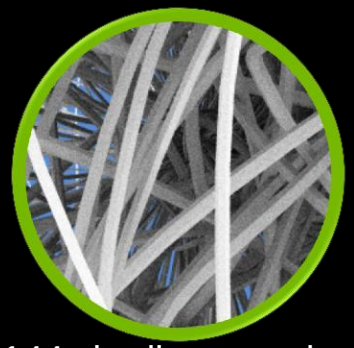
18 shading samples
2 AA samples



36 shading samples
4 AA samples



72 shading samples
8 AA samples



144 shading samples
16 AA samples

3.0

Up to 8X faster SBVH builds

CUDA printf()

RT_WRAP_CLAMP_TO_BORDER

BVH refit on all builders

RT_WRAP_MIRROR

CUDA Interoperability

Kepler optimization

GPU Direct for GL interop buffers

LRU page replacement policy optimized

rtContextSetTimeoutCallback()

CPU Fallback

Multiple Importance Sampling sample

BVH Refinement

rtContextGetAttribute(USED_HOST_MEMORY)

rtDeviceGetAttribute(CUDA_DEVICE_ORDINAL)

Ocean cuFFT sample

Large node graphs optimized

rtDeviceGetAttribute(TCC_DRIVER)

getGPUPagingForcedOff()

Maximus support

CUDA 5.0 support

Isosurface water sample

getGPUPagingActive()

Better multi-GPU load balancing

Faster compiles in many cases

Callable Programs

Texture IDs

GPU Direct for RT_BUFFER_OUTPUT

setUint(uint4)

Displacement mapping sample

C++ API getters are const

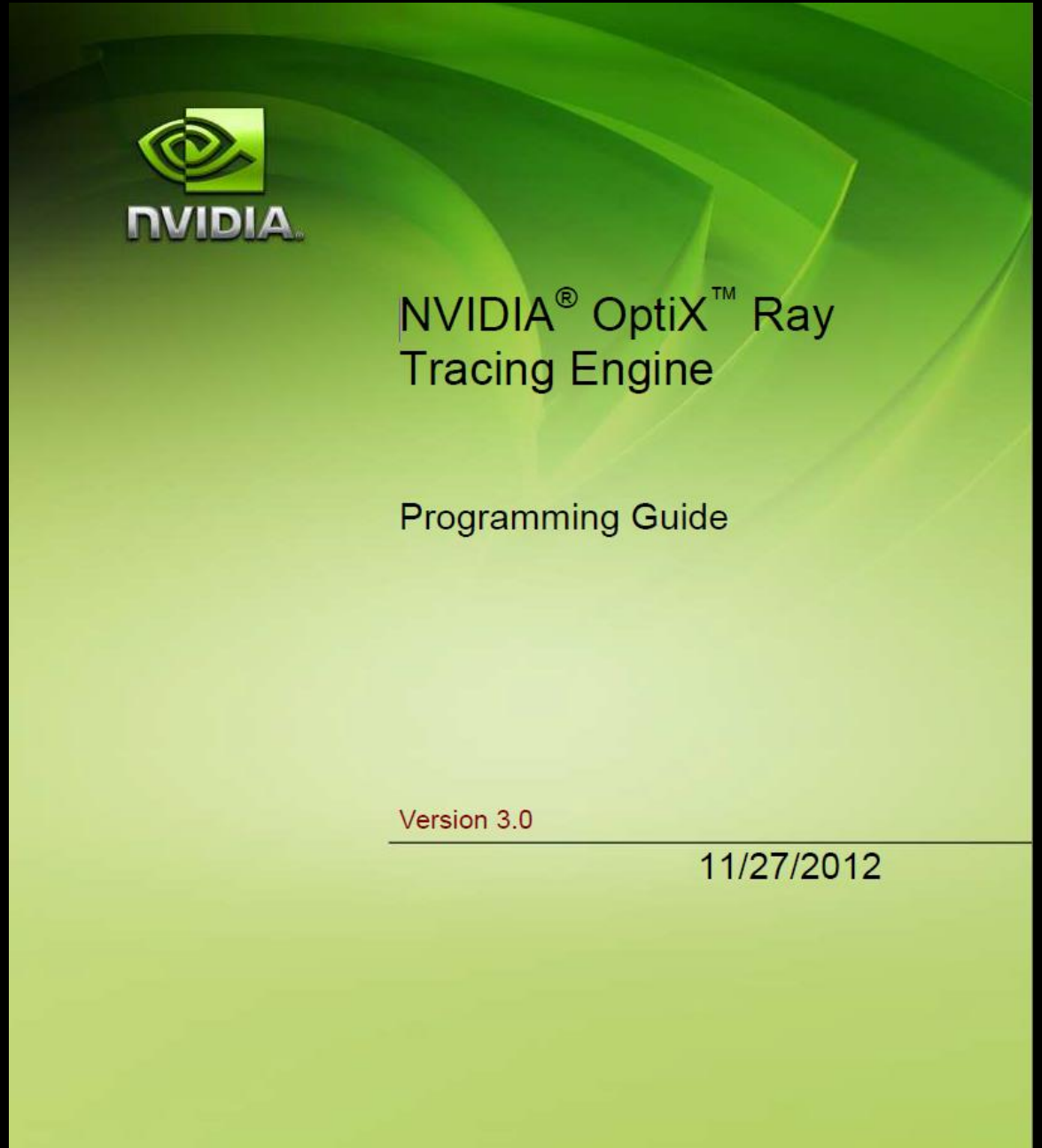
CUDA NVVM optimization

PTX 3 support

Avoid recompiles in many cases

GPU TECHNOLOGY CONFERENCE

- Chapter 9



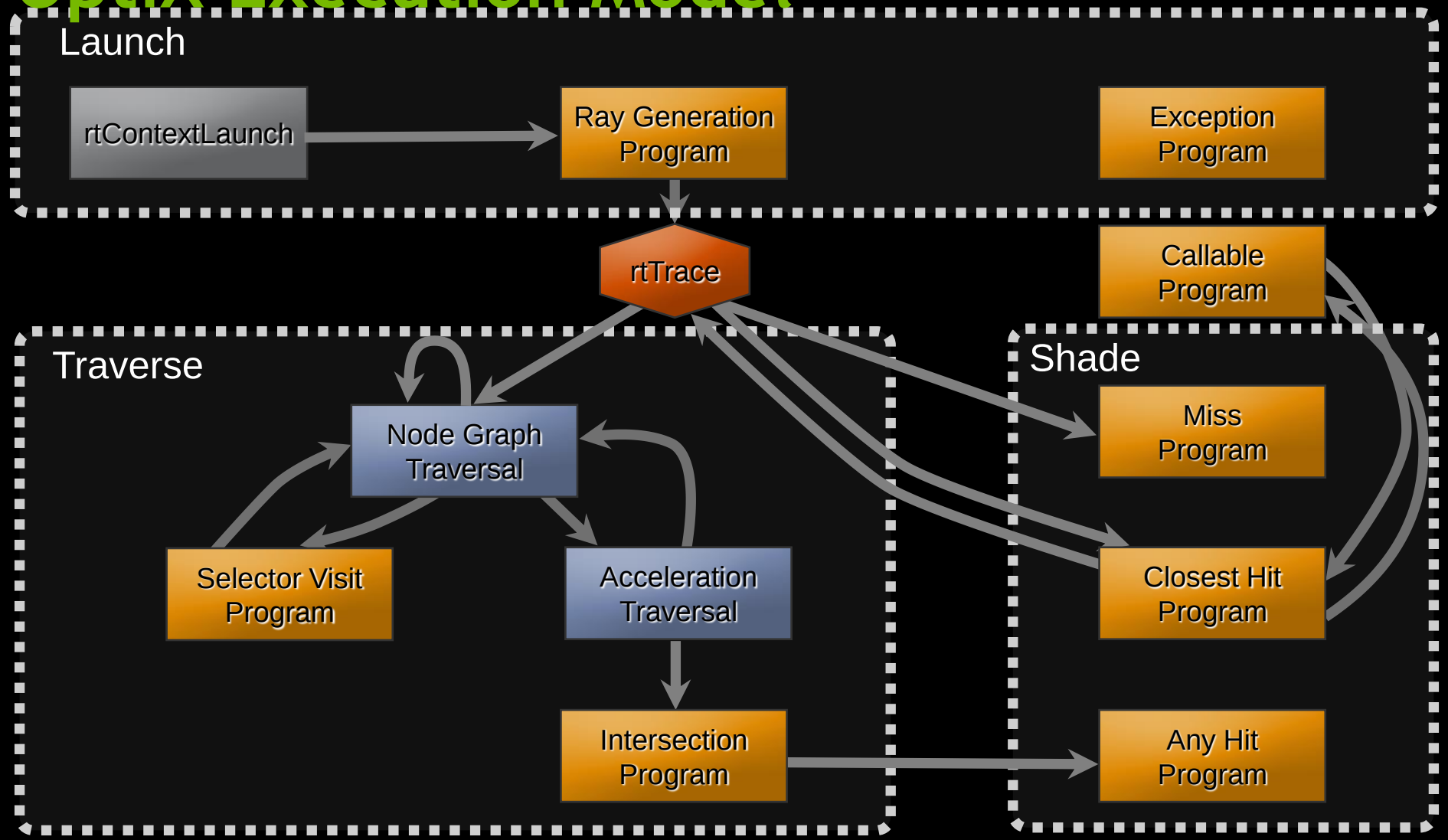
Glass Sample

- Whitted-style recursive
- Reflection and refraction per hit
- Beer's Law attenuation
- Depth cut-off
- Importance cut-off

Optimizing OptiX Device Code

- Maximize rays/second
  - Avoid gratuitous divergence
  - Avoid memory traffic
- Minimize rays needed for given result
  - Improved ray tracing algorithms
    - MIS, QMC, MCMC, BDPT, MLT

OptiX Execution Model



Minimize Continuation State

- OptiX rewrites rtTrace, rtReportIntersection, etc. as:
  - Find all registers needed after rtFunction (continuation state)
  - Save continuation state to local memory
  - Execute function body
  - Restore continuation state from local memory
- Minimizing continuation state can have a large impact

Minimize Continuation State

```
float3 light_pos, light_color, light_scale;
sampleLight(light_pos, light_color, light_scale); // Fill in values

optix::Ray ray = ...;                             // create ray given light_pos

PerRayData shadow_prd;
rtTrace(top_object, ray, shadow_prd);             // Trace shadow ray

return light_color*light_pos*shadow_prd.attenuation;
```

light_pos and light_color saved to local stack

Minimize Continuation State

```
float3 light_pos, light_color, light_scale;
sampleLight(light_pos, light_color, light_scale); // Fill in values
float3 scaled_light_color = light_color*light_scale;
optix::Ray ray = ...;                    // create ray given light_pos

PerRayData shadow_prd;
rtTrace(top_object, ray, shadow_prd);            // Trace shadow ray

return scaled_light_color*shadow_prd.attenuation;
```

Minimize Continuation State

```
RT_PROGRAM void closestHit() {
    float3 N  = rtTransformNormal( normal );
    float3 P  = ray.origin + t_hit * ray.direction;
    float3 wo = -ray.direction;

    // Compute direct lighting
    float3 on_light  = lightSample();
    float  dist = length(on_light-P)
    float3 wi = (on_light - P) / length;
    float3 bsdf = bsdfVal(wi, N, wo, bsdf_params);
    bool is_occluded = traceShadowRay(P, wi, dist);
    if( !is_occluded ) prd.result = light_col * bsdf;

    // Fill in values for next path trace iteration
    bsdfSample( wo, N, bsdf_params,
                prd.next_wi, prd.next_bsdf_weight );
}
```

Pulled above trace to reduce stack state

```
RT_PROGRAM void closestHit() {
    float3 N  = rtTransformNormal( normal );
    float3 P  = ray.origin + t_hit * ray.direction;
    float3 wo = -ray.direction;

    // Fill in values for next path trace iteration
    bsdfSample( wo, N, bsdf_params,
                prd.next_wi, prd.next_bsdf_weight );

    // Compute direct lighting
    float3 on_light = lightSample();
    float dist = length(on_light - P)
    float3 wi = (on_light - P) / length;
    float3 bsdf = bsdfVal(wi, N, wo, bsdf_params);
    bool is_occluded = traceShadowRay(P, wi, dist);
    if( !is_occluded ) prd.result = light_col * bsdf;
}
```

DesignGarage: iterative path tracer

- Closest hit programs do:
  - Direct lighting (next event estimation with shadow query ray)
  - Compute next ray (sample BSDF for reflected/refracted ray info)
  - Return direct light and next ray info to ray gen program
- Ray gen program iterates

DesignGarage: iterative path tracer

```
RT_PROGRAM void rayGeneration(){
    float3 ray_dir = cameraGetRayDir();
    float3 result = tracePathRay( camera.pos, ray_dir, 1 );
    output_buffer[ launch_index ] = result;
}
```

```
RT_PROGRAM void closestHit() {
    // Calculate BSDF sample for next path ray
    float3 ray_direction, ray_weight;
    sampleBSDF( wo, N, ray_direction, ray_weight );

    // Recurse
    float3 indirect_light = tracePathRay(P, ray_direction,
        ray_weight);
    // Perform direct lighting
    ...
    prd.result = indirect_light + direct_light;
}
```

DesignGarage: iterative path tracer

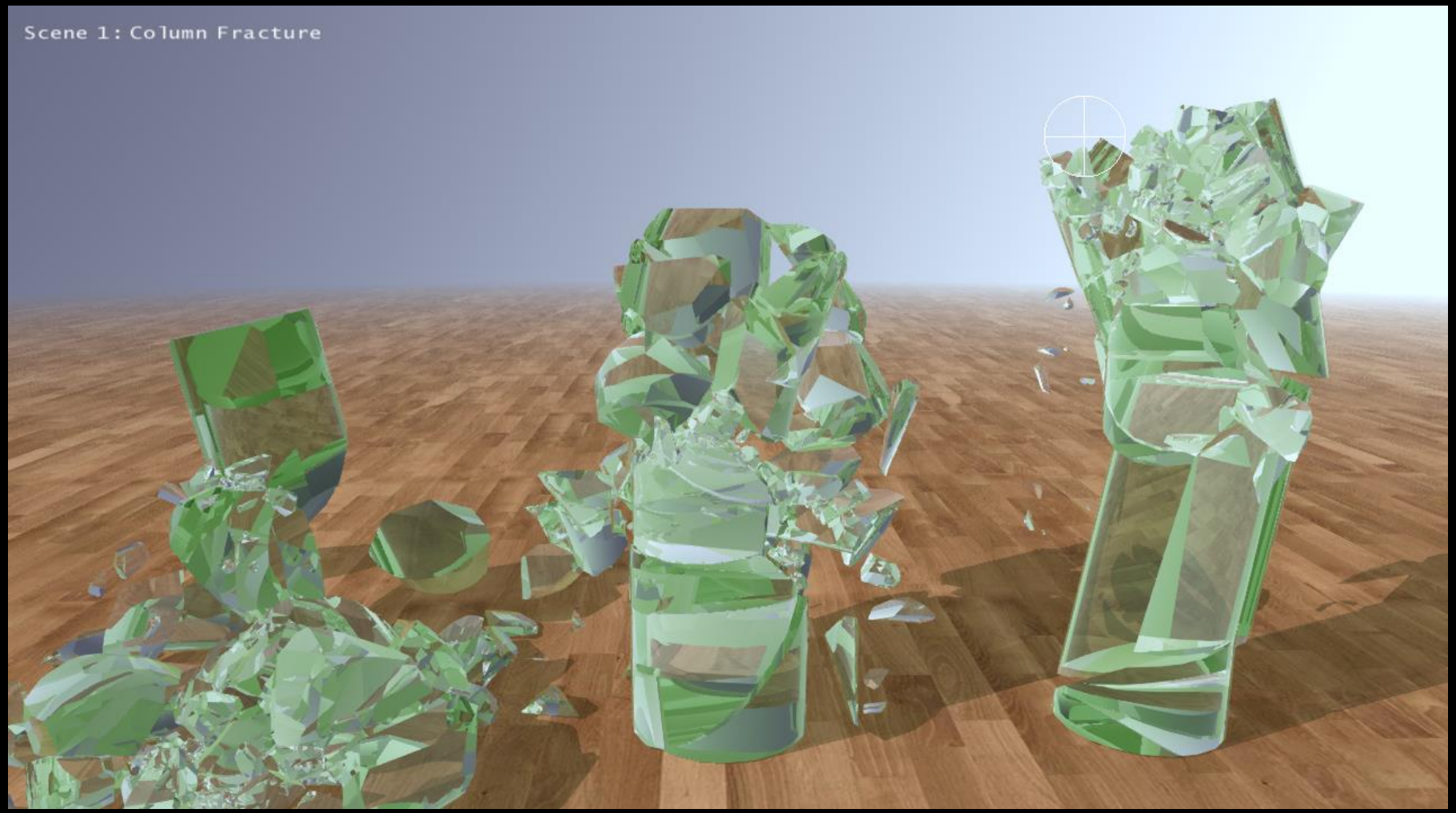
```
RT_PROGRAM void rayGeneration() {
    PerRayData prd;
    prd.ray_dir = cameraGetRayDir();
    prd.ray_origin = camera.position;
    float3 weight = make_float3( 1.0f );
    float3 result = make_float3( 0.0f );

    for( i = 0; i < MAX_DEPTH; ++i ) {
        traceRay( prd.ray_origin, prd.ray_dir, prd );
        result += prd.direct*weight;
        weight *= prd.ray_weight;
    }
    output_buffer[ launch_index ] = result;
}
```

```
RT_PROGRAM void closestHit() {
    // Calculate BSDF sample for next path ray
    float3 ray_direction, ray_weight;
    sampleBSDF( wo, N, ray_direction, ray_weight );

    // Return sampled ray info and let ray_gen iterate
    prd.ray_dir = ray_direction;
    prd.ray_origin = P;
    prd.ray_weight = ray_weight;
    // Perform direct lighting
    ...
    prd.direct = direct_light;
}
```

Registers

- Accessing stack allocated arrays through pointers uses local memory not registers
  - Change float v[3] to float v0, v1, v2
  - Avoid accessing variables via pointer
- Register spilling
  - When working set of registers is too large
  - Registers are stored to local memory
  - Keep working set small
  - Collapse terms when possible

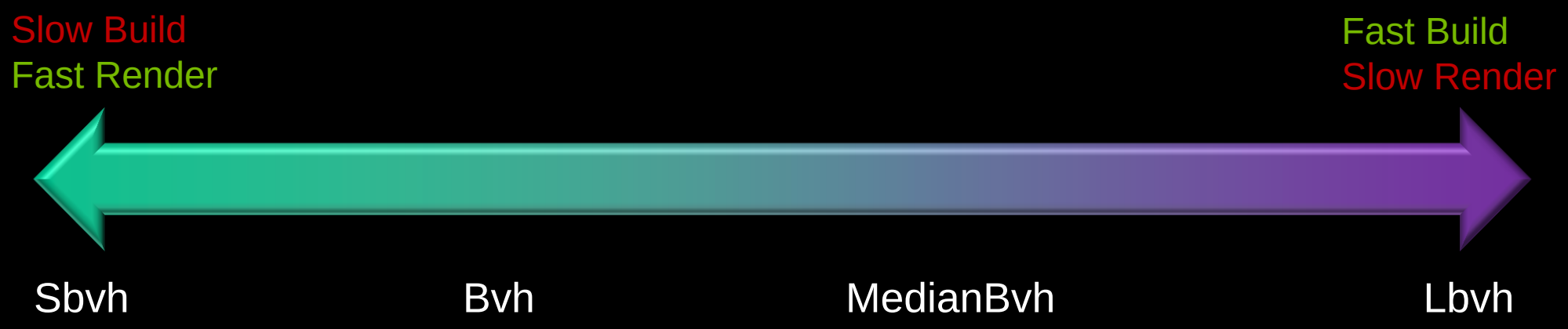
Fracture



GPU Rigid Bodies, Max ray depth = 12, ~350,000 triangles

Acceleration Builder Options

- Sbvh has world class ray tracing performance
- Lbvh is extremely fast and works on very large datasets



Avoiding Acceleration Builds

- Cache acceleration structure alongside model
  - rtAccelerationGetData / rtAccelerationSetData
- Segregate dynamic and static geometry
  - Use lightweight global Acceleration
  - Only rebuild dynamic geometry and global Acceleration
- Add padding to account for special primitive movement
  - Trade off framerate vs. rebuild/refit lag
  - Use sparingly

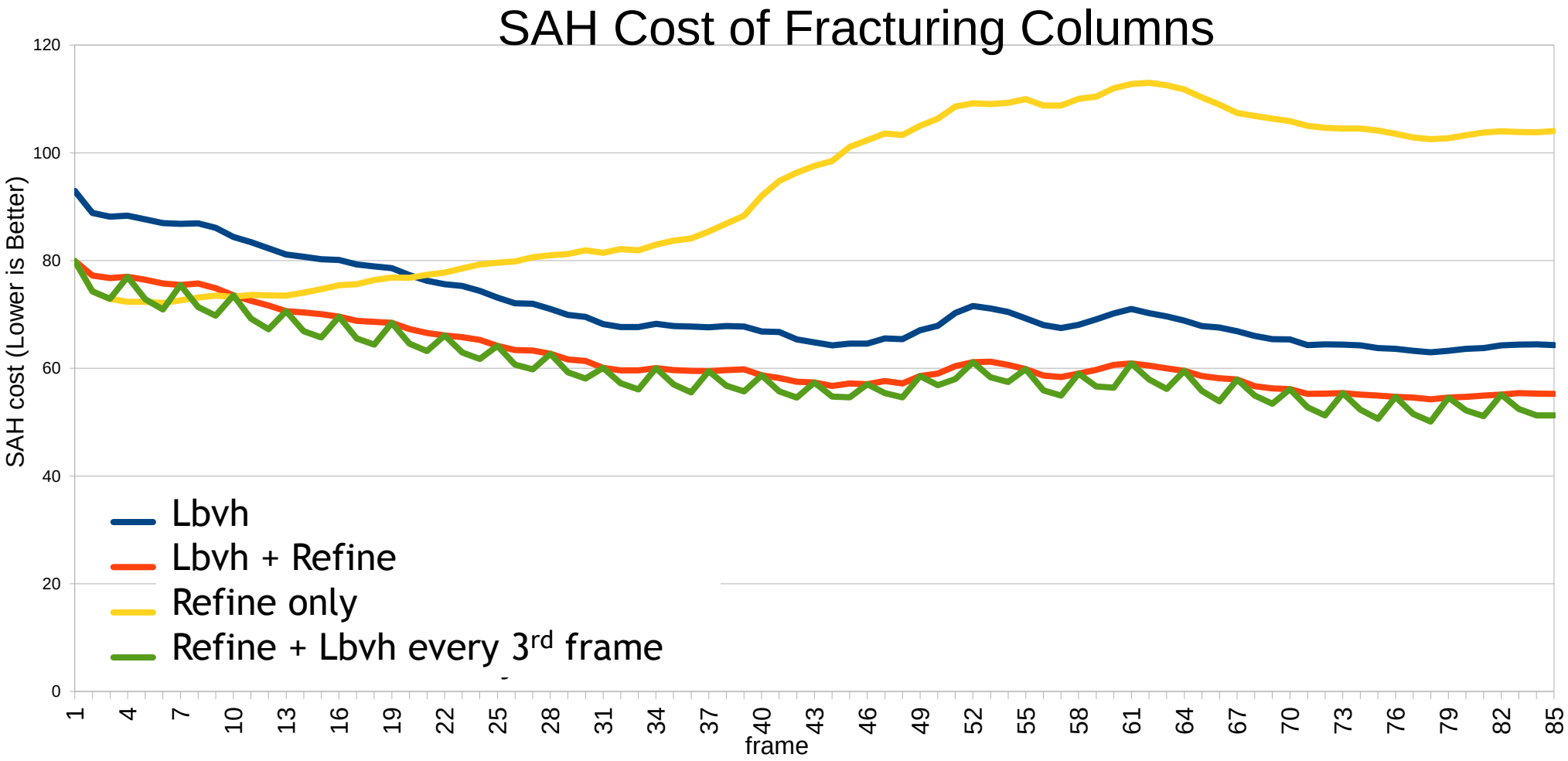
Acceleration Structure Refinement

- BVH Refinement optimizes the quality of a BVH
  - Allows use of fast builder (e.g., MedianBvh) which you can refine between frames
  - Smoother scene editing
  - Smoother animation
- rtAccelerationSetProperty(accel, “refine”, “8”)
  - 0 → rebuild or refit; never refine
  - 1+ → refit and refine N times per frame
    - Rebuild if prim count changes
    - Rebuild if marked dirty

Acceleration Refitting

- Refit keeps existing BVH node structure
- Adjusts bounding box extents to fit moved primitives
- ☺ Very fast update
- ☹ Slower rendering if primitives move too much
- rtAccelerationSetProperty(accel, “refit”, “1”)
  - 0 → rebuild whenever dirty
  - 1 → refit every frame
    - Rebuild if prim count changes
  - 2+ → refit every frame; refine every N^{th} frame
    - Rebuild if prim count changes

Refinement for Animation



API Interoperability

- Share data between OptiX and
  - CUDA
  - OpenGL
  - Direct3D
- Saves host-device copies

Sharing Pointers with CUDA

- rtBufferSetDevicePointer() – CUDA owns the buffer

```
{
  const float* d_out_probe_buf;
  cudaSetDevice(0);
  cudaMalloc(&d_output_probe_buffer, moving_obj_count * sizeof(float));
  rtBufferSetDevicePointer(buf, optixDevice0, d_out_probe_buf);
  rtContextLaunch1D(..., moving_obj_count);

  LOS_reduction<<<moving_obj_count, 1>>> (d_moving_objs, d_out_probe_buf);
}
```

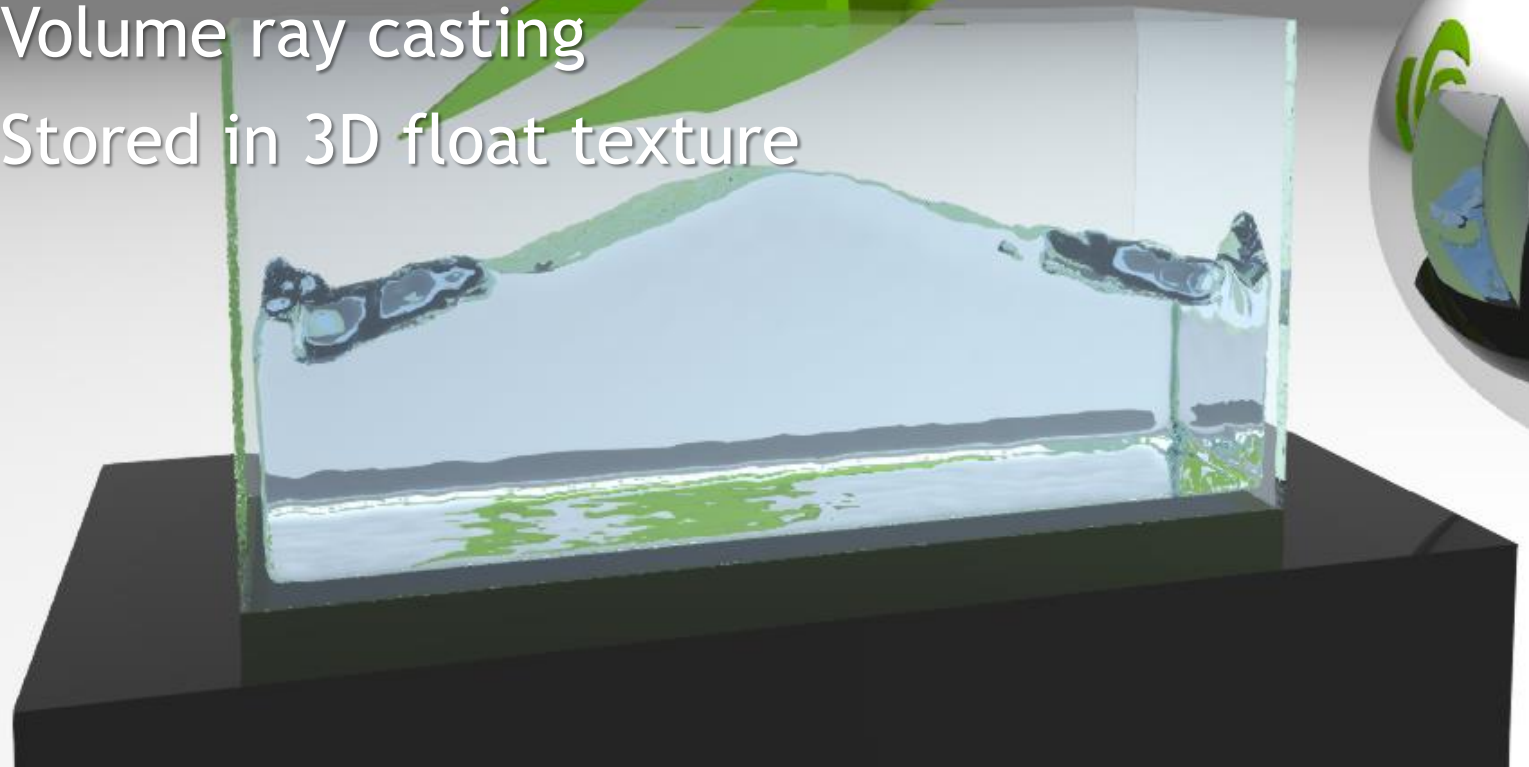
Sharing Pointers with CUDA

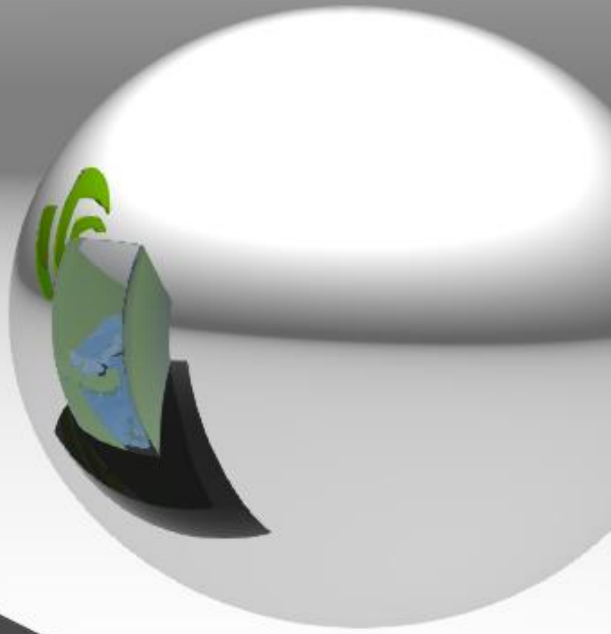
- rtBufferGetDevicePointer() – OptiX owns the buffer

```
{
  rtContextLaunch1D(..., moving_obj_count);
  const float* d_out_probe_buf;
  rtBufferGetDevicePointer(buf, optixDevice0, &d_out_probe_buf);

  cudaSetDevice(0);
  LOS_reduction<<<moving_obj_count, 1>>> (d_moving_objs, d_out_probe_buf)
}
```

Water

- 128x64x64 water volume
- No isosurfaces
- Volume ray casting
- Stored in 3D float texture





Reuse RTprograms

- Use variables or control flow to reuse programs
  - ☹ singleSidedDiffuse closest hit and doubleSidedDiffuse closest hit
  - ☺ diffuse closest hit and RTvariable do_double_sided
- Use in moderation: uber shaders cause longer compilation

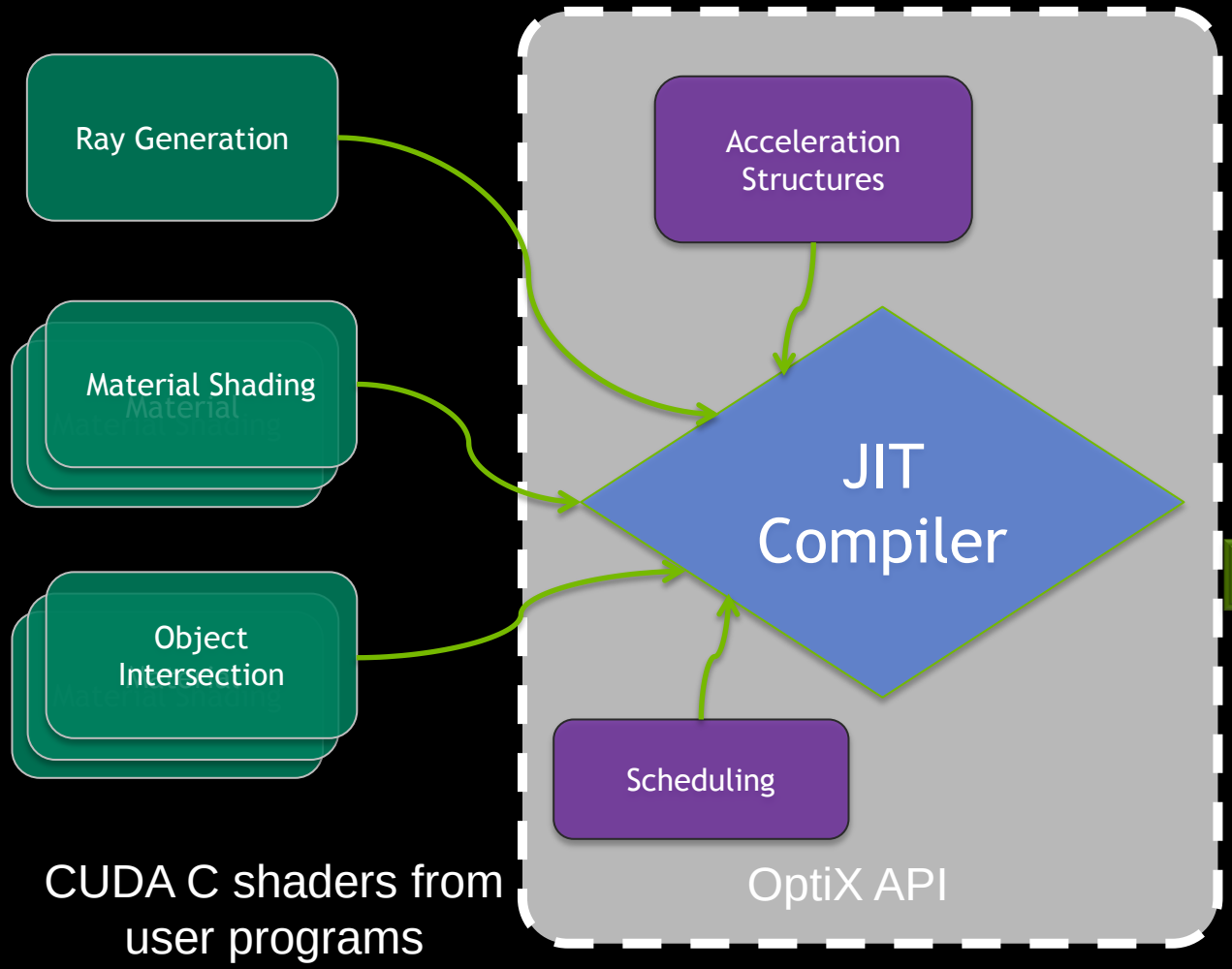
Careful arithmetic

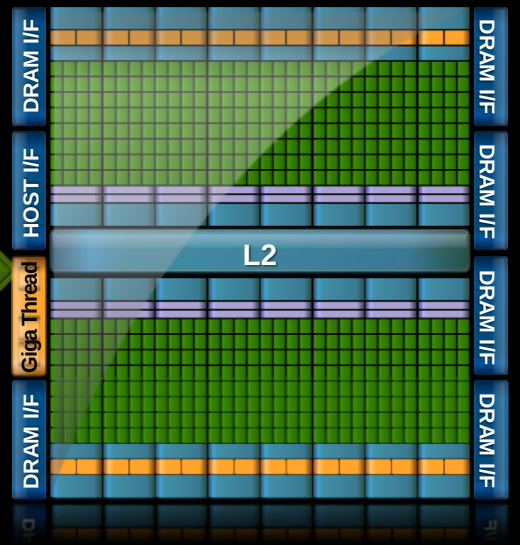
- nvcc --use-fast-math
- Do not unintentionally use double precision math
  - 1.0 != 1.0f
  - cos() != cosf()
- Search for ".f64" in your PTX files

Shallow Node Hierarchies

- Flatten node hierarchy
  - Collapse nested RTtransforms
  - Pre-transform vertices
  - Use RTselectors judiciously
- Combine multiple meshes into single mesh
- ☺ A single BVH over all geometry
- ☹ Per-mesh BVHes

How OptiX Links Your Code

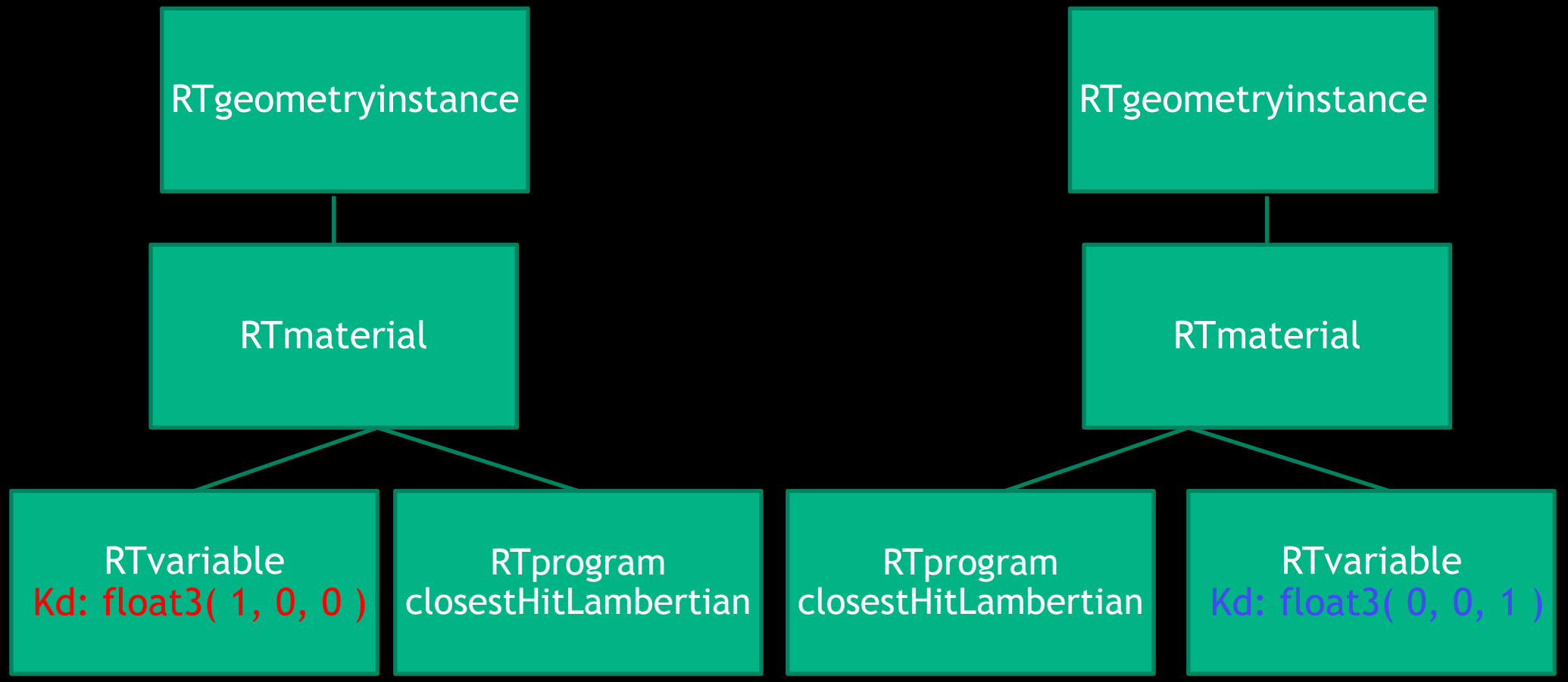




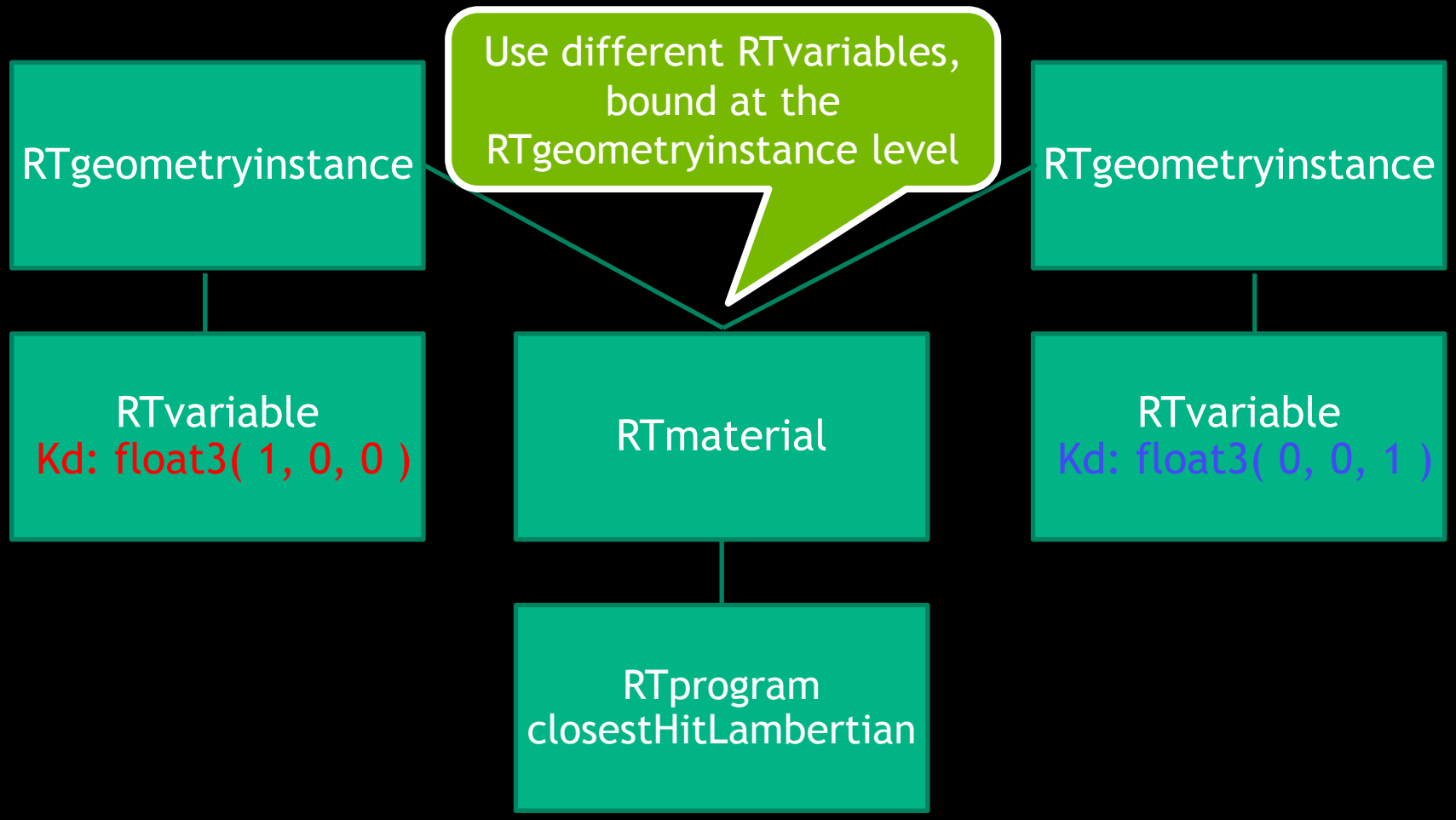
Minimize graph changes between launches

- Some changes cause kernel recompile
  - Adding an RTprogram for PTX function not used before
  - Adding a non-bindless texture to the scene
- OptiX at least traverses graph to check validity
- Safe changes:
  - RTbuffer contents
  - Rtvariable values
- Bindless textures avoid recompiles

Share Graph Nodes



Share Graph Nodes



Leverage CUDA compute cache

- On disk cache of PTX codes compiled to machine code
- Force compile when convenient
  - rtContextLaunch2D(entry_point, 0)

Callable Programs Speed Up Compilation

- OptiX inlines all CUDA functions
- ☺ Fast execution
- ☹ Large kernel to compile
- Use callable programs

Callable Programs Speed Up Compilation

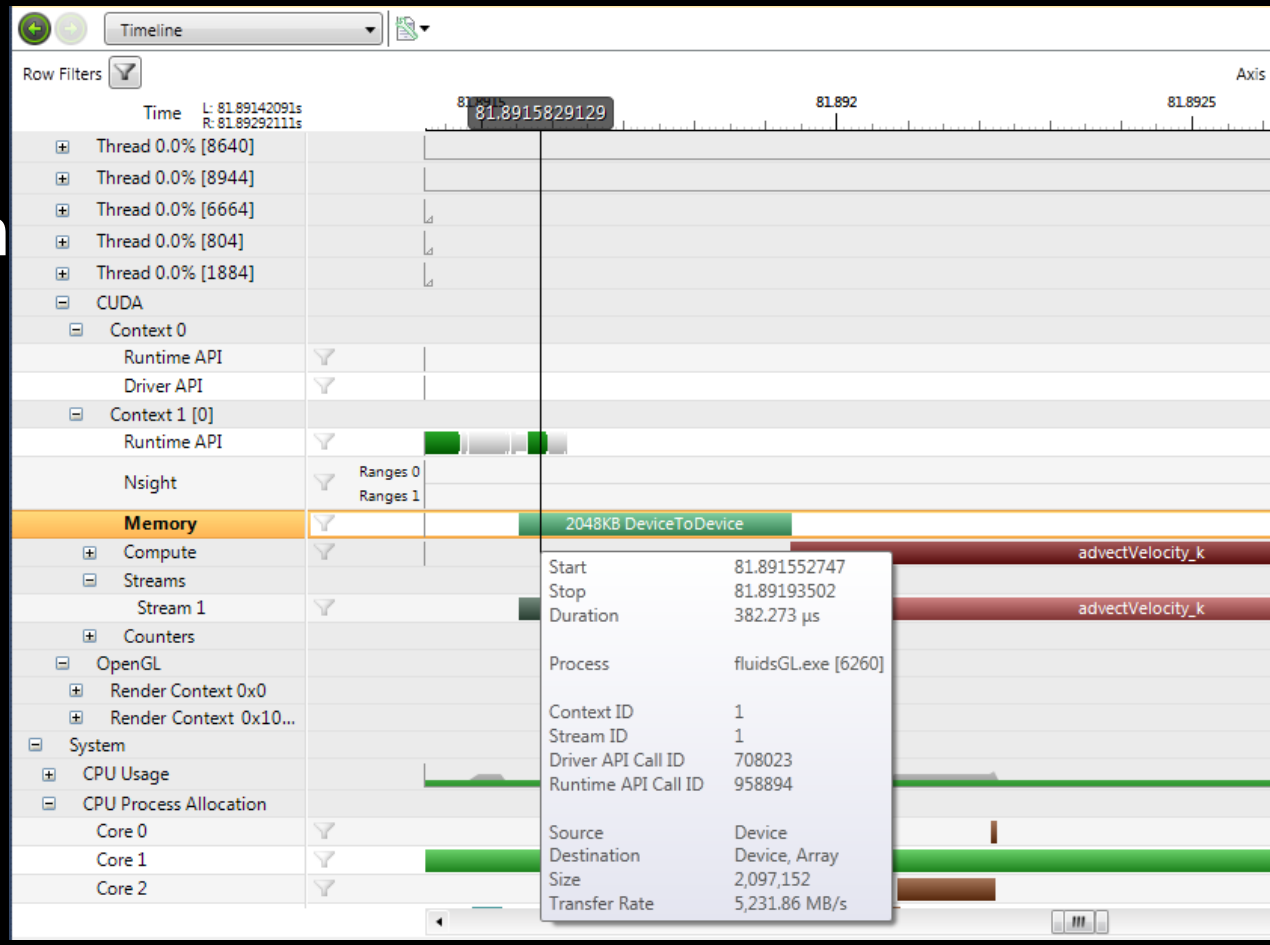
```
RT_CALLABLE_PROGRAM float3 checker_color(float3 input_color, float scale)
{
  uint2 tile_size = make_uint2(launch_dim.x / N, launch_dim.y / N);
  if (launch_index.x/tile_size.x ^ launch_index.y/tile_size.y)
    return input_color * scale;
  else
    return input_color;
}

rtCallableProgram(float3, get_color, (float3, float));

RT_PROGRAM camera()
{
  float3 initial_color;
  // … trace a ray, get the initial color …
  float3 final_color = get_color( initial_color, 0.5f );
  // … write new final color to output buffer …
}
```

Callable Programs Speed Up Compilation

- Callable programs reduce OCG compile times
- Bigger wins in future with separate compilation and linking
- Small rendering performance overhead
- Enables shade trees and plugin rendering architectures

Visualize application using nSight

- activity over time
- CPU & GPU interaction



nvToolsExt

```
#include “nvToolsExt.h” // Found inside the Nsight installation
#pragma comment( lib, “nvToolsExt64_1.lib”)

nvtxMarkEx(“Something interesting happening”);

nvtxRangePushEx(“Doing process A”);
A();
nvtxRangePop();
```

GPU TECHNOLOGY CONFERENCE

- Download OptiX
  - Available free: Windows, Linux, Mac
  - http://developer.nvidia.com
- OptiX-Help@nvidia.com
- Ray tracing talks:
  - http://www.gputechconf.com
  - OptiX
  - Iray
  - Material Description Language