Reducing Divergence with an in-Shader JIT Virtual Machine for Material Graphs

Presented at the Khronos Vulkanised 2023 Conference

Mateusz Kielan
CEO & CTO
devsh@devsh.eu
What is a Material Graph?

PBRT and Mitsuba are very similar, but we’ll discuss the latter due to its documentation and borrow some diagrams.

We don’t support Hanrahan-Krueger yet.

The “BxDF”s to the left and right are modifiers that can nest other BxDFs.

Missing from the diagram are “twosided”, [opacity] “mask” and the weighted sum blend modifiers.
Our Problem

- Implement Path Tracing with GLSL Compute Shaders before the release of NVX_ray_tracing
- Replace a Mitsuba 0.5 Renderfarm of 56 individual i7 CPU nodes with a single GPU
  - A preferably cheap cloud GPU to scale up and down based on demand
- Raise the default render resolution from 1080p to 4k while shortening render times
- With Materials to be defined as BxDF material graphs from
  - Mitsuba XML, and in the future NvidiaMDL, MaterialX, etc...
  - Must Evaluate and Importance Sample (IS) from this BxDF material graph
- Could not invent custom formats to ensure a smooth transition from existing system
- As a bonus it should run on recent Integrated GPUs to enable local Interactive Previews
Our Solution: The **Nabla Material Compiler**

- **Material Compiler** in Nabla is a Domain Specific Compiler built for efficient evaluation and importance sampling of *All BxDF Graphs in the Scene* in the context of Path Tracing
  - Assumes **Maximally Divergent Subgroup Invocations** due to Low Discrepancy Sequences
- Backends implement a GPU Just-In-Time compiled and optimized Virtual Machine (VM) with a custom Virtual Instruction Set (VIS) and a Register File (RF)
- Our custom Intermediate Representation (IR) is a superset of all Frontends’ capabilities.

**Diagram:**

- Mitsuba XML Frontend
- NVIDIA MDL Frontend
- MaterialX Frontend
- Nabla material IR
- GLSL Virtual Machine Backend
- KHR_raytracing_pipeline callable Backend
- Uberkernel Backend
General architecture of the Nabla Material Compiler

Multiple Frontends

- [DONE] Mitsuba XML - Parses Mitsuba BxDF XML into our IR, performs cached conversions of normal/bumpmaps to derivative maps.
- [PLANNED] MaterialX - Will be done to support the latest glTF, but so far no idea about IR changes needed.
- [IF CLIENTS NEED] Nvidia Material Definition Language - IR was designed with MDL in mind, so could be written with minimal effort.

Intermediate Representation Features

- Bindless Agnostic - Texture Sampling has separate implementations per Backend, IR just references image assets.
- Lossless and Expressive - IR canonicalizes the form of mathematically identical BxDFs so that the form is unique.

Multiple Backends

- [DONE] GLSL Virtual Machine - Implements the Virtual Machine as single unified function reading instructions from SSBOs and uses an array as the Register File which is dynamically sized (this is one of the things we JIT specialize before compilation of the shader VM).
- [PLANNED] KHR_raytracing_pipeline VM - Each instruction would be a Callable, this would still need an emulated RF in the Ray’s Payload.
Toolchain with example Frontend and Backend

- Mitsuba XML
- Intermediate Representation
- Instruction Stream
- GLSL

```
switch (opcode) {
  case OP_DIFFUSE:
    instr_execute_DIFFUSE(instr, regs, dUV, scale, bsdf_data);
    break;
  case OP_ROUGHDIFFUSE:
    instr_execute_ROUGHDIFFUSE(instr, regs, dUV, scale, a2, bsdf_data);
    break;
  ...
}
```
Current Production Backend: **GLSL Virtual Machine**

- **Has** virtual instruction sets which are executed inside a single unified GLSL function
  - 64-bit fixed-length instruction encoding and virtual registers
  - Most of the BxDF parameters are stored in a separate SSBO to allow for the above

- **4 types of instruction set** and **instruction streams** for different purposes:
  - Texture Prefetch to hoist texture fetching out of the BxDF IS and Evaluation functions
  - Normal Precomputation
  - BxDF Importance Sampling Generation
  - BxDF Evaluation and Robust AOV Extraction for denoising

- **Virtual Texturing** with anisotropic filtering, mip tails, multiple views aliased for compat. Formats

- **Can be copy-pasted into ANY Shader Stage** (fragment, compute, raytracing pipeline)
AOV Extraction

- Extract albedo and normals from the scene
- Still evaluate the material graph but estimating an ad-hoc AOV Transport Integral
  - E.g. Normal of the chair is visible behind some transparent glass
- Used as inputs for OptiX AI Denoiser
  - AOVs Transport gives a drastically better and consistent result
Imagine a Rough GGX Dielectric and a Rough Beckmann Conductor, either as specializations or generalizations with conditionals.

Here’s a “not-to-scale” scheduling diagram of two invocations within the same subgroup for a GPU architecture without Independent Program Counters:
### Instruction encoding example

<table>
<thead>
<tr>
<th>opcode</th>
<th>α</th>
<th>ndf</th>
<th>t</th>
<th>pad</th>
<th>BxDF data offset</th>
<th>dest</th>
<th>src1</th>
<th>src2</th>
<th>pad</th>
</tr>
</thead>
<tbody>
<tr>
<td>0011</td>
<td>1</td>
<td>01</td>
<td>1</td>
<td>0000</td>
<td>00000000000000001111</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
<td>00000000</td>
</tr>
</tbody>
</table>

- **Whether first param (α_u) is from texture**
- **NDF_GGX**
- **Whether second param (transmittance) is from texture**
- **Offset of BxDF data in BxDF data buffer**
- **Destination register number is 0**
- **Source 1 register number (used in other ops e.g. OP_BLEND)**
- **Source 2 register number (used in other ops e.g. OP_BLEND)**
BxDF sampling with **Instruction Stream**

- “Stochastically” traverse down the graph to pick leaf BxDF IR node from which we will generate sample. Based on flawed assumption that whole subtree passes the White Furnace Test and conserves energy.
Ongoing Work: **Material Compiler V2**

- **Rewrite in HLSL202x** for ergonomic overrides
- **Common Subexpression Elimination** by Hash Consing with Merkle Tree
  - Client once fed a scene that resulted in a 30MB+ Combined IR, overflowing fixed size allocator
  - Production Scenes have **thousands** of duplicate materials and subexpressions
- **Better BxDF sampling technique**
  - The current BxDF sampling performs poorly when subexpressions are meant to lose energy
  - Will do **Single Reservoir Sampling** with actual final contribution weights of leaf BxDF nodes
- **Incremental Linking of IR DAGs together** to allow for importing multiple scenes after each other
  - Possibly even support “live-edit” using non-JIT VM and optimize with AOT in the background
- **Custom scheduler** to outperform and eliminate the need for Shader Execution Reordering (SER)
  - Distribute Texture Prefetch and Normal Precalculation across entire workgroup
Thank you