LEVERAGING NVRTC FOR BUILDING OPTIX SHADERS FROM MDL MATERIALS

Detlef Röttger, NVIDIA
Andreas Mank, ESI Group
2017-05-08
VISUALIZE REAL-WORLD LIGHTS AND MATERIALS
Introduction

Evolution of the Renderer Architecture

Introduction to NVRTC

Shader Generation from MDL Materials

Integration into ESI IC.IDO

Live Demonstration
INTRODUCTION
MOTIVATION

Scene exchange among applications often loses material information

Use the NVIDIA Material Definition Language (MDL) to exchange materials

Take advantage of existing MDL material libraries

Create new MDL materials with available third-party tools

Renderer goal:
Handle MDL materials at runtime on end-user target system
**MDL**

**NVIDIA Material Definition Language**

A domain-specific language for abstract declarative material description

Independent of a specific rendering system

Procedural programming language to customize texture image lookups or procedural textures

**MDL Handbook and Specifications:**
[http://www.mdlhandbook.com](http://www.mdlhandbook.com)

**More Information:**
OPTIX
NVIDIA GPU Ray Casting API

High-level GPU accelerated ray-casting API

C-API to setup scene and data

Multiple program domains and per ray payload under developer’s control

Flexible single ray programming model

Supports multi-GPU and progressive rendering on remote NVIDIA VCA cluster

Develop "to the algorithm"
EVOLUTION OF THE RENDERER ARCHITECTURE
RENDERER IMPLEMENTATION
Goal: Handle MDL Materials at Runtime

Represent complex layered material hierarchies as CUDA C++ code

Templated classes for layered material construction from "fixed-function" building blocks (BSDFs, EDFs, VDFs, Layers, Mixers, Modifiers, Conditionals)

Connect user defined parameter calculations with building blocks

Derive from generated "getter" classes which fill building block input parameters.
All functions inlined into material hierarchy traverser function.

Generate high-level CUDA C++ code

Easy prototyping and debugging before writing the code generator.
Benefit from all optimizations inside the CUDA compiler.
* rectangles are fixed-function code
* round rectangles are bindless callable programs
INTRODUCTION TO NVRTC
NVRTC vs. NVCC
CUDA Compilation

NVRTC standalone library

- Translates CUDA C++ source to PTX device code
- End-users do not need a full development environment (e.g. MSVS)
- Three times faster compile times compared to NVCC

NVCC CUDA Compiler

- Supports host and device code
- Works in combination with a host compiler of a full development environment
NVRTC
Advantages of Runtime Compilation

OptiX shaders can be compiled on-demand

- Applications do not have to provide a large number of individual shaders upfront
- Materials can be created and changed at runtime

Specialized shaders improve performance

- No large „uber-shaders“ necessary
- Shader code can be kept compact
nvrtcProgram prog;
nvrtcCreateProgram(&prog, src, NULL, 0, NULL, NULL);
nvrtcCompileProgram(prog, numOptions, options);

nvrtcGetProgramLogSize(prog, &logSize);
if (1 < logSize)
{
    nvrtcGetProgramLog(prog, log);
}

nvrtcGetPTXSize(prog, &ptxSize);
if (1 < ptxSize)
{
    nvrtcGetPTX(prog, ptx);
}

nvrtcDestroyProgram(&prog);
const std::string cudaIncludes = std::string("-I") + m_cudaIncludePath;
const std::string optixIncludes = std::string("-I") + m_optixIncludePath;
const std::string rendererIncludes = std::string("-I") + m_rendererIncludePath;

const char* options[] = {
    "--gpu-architecture=compute_30",
    "--use_fast_math",
    "--device-as-default-execution-space",
    "--relocatable-device-code=true",
    "-D__x86_64",
    cudaIncludes.c_str(),
    optixIncludes.c_str(),
    rendererIncludes.c_str()};
SHADER GENERATION FROM MDL MATERIALS
OPTIX SHADER GENERATION
Using the MDL SDK and NVRTC

- `<name>.mdl`
- MDL SDK
- Compiled Material
- Builder Class
- DAG Nodes

Texture References
Parameter Interface
Header
Hierarchy Typedefs
Parameter Macros
Traverser Function
Getter Classes
Material Constructor

Description
Traverser `<hash>.cu`
NVRTC Compiler
Traverser `<hash>.ptx`
INTEGRATION INTO ESI IC.IDO
“ESI Rendering Innovations with NVIDIA DesignWorks™”
Andreas Mank, Team Leader Visualization, ESI Group
Markus Tavenrath, Senior Developer Technology Engineer, NVIDIA
— Source: GTC 2016, s6306

“MDL Materials to GLSL Shaders – Theory and Practice”
Andreas Süßenbach, Senior Developer Technology Engineer, NVIDIA
Andreas Mank, Team Leader Visualization, ESI Group
— Source: GTC 2016, s6311

“Implementing MDL Materials with Support for IES Lights and AxF Appearance Representations”
Detlef Roettger, Senior Developer Technology Engineer, NVIDIA
Andreas Dietrich, Senior Software Developer Visualization, ESI Group
— Source: GTCEU 2016, s6135
Helios Rendering Architecture
Overview

Application → IC.IDO → Helios → Renderer

- RiXGL Back-End (DLL)
- OptiX Back-End (DLL)
Helios Rendering Architecture

Interfaces

Application API

- Load and unload rendering back-ends (DLLs can be loaded at run-time)
- Switch between back-ends (e.g., ray tracing or rasterization based)
- Render graph control (e.g., hybrid rendering, frame composition)
- Provide original (unoptimized) scene data

Back-end API

- Set scene geometry and transformations (flattened two-level scene graph)
- Set rendering parameters (e.g., materials, lights, whitted ray tracing or GI)
Helios OptiX Back-End

**Helios**

**OptiX Back-End**

**MDL Parser**
- Reads MDL files
- Generates material parameter lists

**OptiX Builder**
- Generates material traversers (CUDA C++)
- Compiles bindless callable programs

**MDL SDK**

**OptiX**

**NVRTC**
MDL IN CUSTOM RENDERERS

DEFINITION

IMPLEMENTATION

NVIDIA IRAY

CUDA (Ray Tracer)

MDL SDK

MATERIAL SHARING (LIBRARY)
LIVE DEMONSTRATION
START SD6
CREATE A NEW MDL MATERIAL
SET MDL MATERIAL PROPERTIES

1. Select Metallic > Metallic Anisotropic
2. Graph Name: MySDMaterial
3. Click OK
1. Load mesh
2. Link 3D Mesh
3. 3D Mesh
ASSIGN NEW MATERIAL
LOAD ADDITIONAL MATERIALS
SHOW ENVIRONMENT
import :df: microfacet_ggx_smith_bsdf;
import :df: scatter_mode;
import :state: normal;
import :state: texture_tangent w;

export material MySDMaterial{
  float anisotropyLevel = 0.0f {[
    :anno: description("Anisotropy Level channel"),
    :anno: hard_range(0.0, 0.9999999983),
    :anno: display_name("Anisotropy Level"),
    :anno: in_group("Anisotropy")
  ]},
  float roughness = 0.5f [
    :anno: display_name("Roughness"),
    :anno: in_group("Roughness")
  ],
  :anno: hard_range(0, 1)
},

color base_color = color(0.212231f, 0.212231f, 0.212231f) [ ]
  :anno: display_name("Base Color"),
  :anno: in_group("Base Color")
],

float anisotropyAngle = 0.0f [
  :anno: description("Anisotropy Angle channel"),
  :anno: soft_range(0, 1),
  :anno: display_name("Anisotropy Angle"),
  :anno: in_group("Anisotropy")
],

float3 normal = :state: normal () [ ]
  :anno: display_name("Normal"),
  :anno: in_group("Normal")
}[
  :anno: author(""")
]
START ICIDO PROOF-OF-CONCEPT
ENABLE GLOBAL ILLUMINATION
IMPORT LIBRARY
ASSIGN MATERIALS FROM LIBRARY
CHANGE COLOR
::anno::display_name("Anisotropy Level"),
::anno::in_group("Anisotropy")
],
float roughness = 0.5f [
  ::anno::display_name("Roughness"),
  ::anno::hard_range(0,1)
],
},
color base_color = color(0.f, 0.401978f, 0.00917f) [
  ::anno::display_name("Base Color"),
  ::anno::in_group("Base Color")
],
float anisotropyAngle = 0.f [
  ::anno::display_name("Anisotropy Angle channel"),
  ::anno::soft_range(0,1),
  ::anno::display_name("Anisotropy Angle"),
  ::anno::in_group("Anisotropy")
],
float3 normal = ::state::normal() [
  ::anno::display_name("Normal"),
  ::anno::in_group("Normal")
]]
);
[[
  ::anno::author(""")
]]

let [
  bool tmp0 = false;
  material_surface tmp1(
    :base::anisotropy_conversion(roughness * roughness, anisotropyLevel, anisotropyAngle, :state::texture_range),
    material_emission(emission: edf(), intensity: color(0.f, 0.f, 0.f), mode: intensity_radiant_existance));
  material_surface tmp2 = material_surface(scattering: bsdf(), emission: material_emission(emission: edf(), intensity: color(0.f, 0.f, 0.f), mode: intensity_radiant_existance));
  material_volume tmp3 = color(1.f, 1.f, 1.f);
  material_volume tmp4 = material_volume(scattering: vdf(), absorption_coefficient: color(0.f, 0.f, 0.f), scattering_coefficient: color(0.f, 0.f, 0.f), absorption_power: 0.f); material_geometry tmp5(
    float3(0.f),
    1.f)
1. Select MDL Module for package 'Unsaved Package'.

2. This PC > Desktop > Organize by Name > File name: MySDMaterial.mdl

3. Save as type: Nvidia Material Definition Language (*.mdl)

EXPORT MDL MATERIAL
THANK YOU

JOIN THE CONVERSATION
#GTC17  

droettger@nvidia.com
andreas.mank@esi-group.com