Project Initiator: GPU Ray Tracing Using OptiX

There are several ways to accelerate ray tracing. An indispensable acceleration technique is the use of spatial data structures to find the triangles that are most likely to be intersected first by a ray. There are several such data structures available. A complementary acceleration technique is parallelization either using clusters, multiple CPU cores, or massively parallel hardware such as the graphics processing unit (GPU). This set of exercises is about acceleration using OptiX, which is a GPU ray tracing API distributed by NVIDIA.[1] Solving these exercises is not mandatory. Their purpose is to be a service toward your final project.

Learning Objectives

- Accelerate ray tracing using massively parallel hardware.
- Port your ray tracing skills to another ray tracer (NVIDIA OptiX).
- Write programs (kernels) that run on the GPU.
- Use ray tracing for real-time applications.

GPU Accelerated Ray Tracing

The purpose is to do something similar to what you did with ray tracing previously in this course, only this time using OptiX. This ray tracing is faster than on the CPU as the GPU has many cores. Use the separate `render_OptiX` framework for the following exercises. Note that the easiest way to generate a Visual Studio solution for an OptiX program is to use CMake.[2]

- Use CMake to generate a C++ solution for building the `render_OptiX` framework on your platform. A step-by-step procedure for Windows and Visual Studio is in Appendix A.
- Take some time to understand what you can do with the keyboard and command line. Information about command line options is printed in the prompt if you compile and run the program with the command line option `-h`. Load the Stanford bunny (`bunny.obj`) into the OptiX framework.
- Implement a shader that uses a directional light source. Do this by implementing the program `__closest-hit__directional` in `shaders.cu`. Render the bunny and save the resulting image both with and without shadows.
- Now load the Cornell box (`CornellBox.obj`) and the blocks inside it (`CornellBlocks.obj`). The Cornell box has an area light source. Implement an area light sampler using the `sample` (or the `sample_center`) function in `AreaLight.h` and the closest hit program `__closesthit__arealight` in `shaders.cu`.
- Load the Cornell box (`CornellBox.obj`) and two spheres (`CornellLeftSphere.obj` and `Cornell-RightSphere.obj`) into the OptiX framework. The materials of the two spheres are mirror and glass respectively, but they will appear black as long as shaders for mirror objects and transparent objects have not been implemented.
- Implement a shader for mirror objects and one for transparent objects. Do this by implementing the corresponding closest hit programs in `shaders.cu`. Use the number of reflections (the trace depth) to stop the recursive ray tracing.
- The ray generation program progressively updates the rendered result as in regular path tracing. Include indirect illumination by implementing `sample_cosine_weighted` in `sampler.h` and using this in the `arealight` program. updating the arealight program.

[1] http://developer.nvidia.com/optix/
[2] http://www.cmake.org/

- As an additional test, load the Cornell box (`CornellBox.obj`), the elephant model (`justElephant-.obj`), and the Stanford bunny (`bunny.obj`). Perhaps give the bunny mirror material (illum 3) and the elephant glass material (use illum 4 for all three wavefront OBJ materials).

Reading Material

Relevant reading material

- Sections 1–5 and 8–11. Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics (SIGGRAPH 2010) 29*(4), Article 66, July 2010.
- NVIDIA. NVIDIA OptiX 7.2 - Programming Guide, Version 1.9. October 2020.

A Using CMake to Build an OptiX Solution

How to use CMake for building an OptiX framework on Windows and Visual Studio:

1. Run the program “cmake-gui” (the current make files assume version 3.1 or later).
2. Click “Browse Source. . . ” and select the folder where you unzipped the framework.
3. Copy the path that now appears under “Where is the source code” to the textbox called “Where to build the binaries” and add `/build` to the path.
4. Click “Configure” and choose “Visual Studio 15 2017” (for example) in the drop-down list specifying the generator. In addition, choose “x64” in the drop-down list called “Optional platform for generator” and click “Finish”.
5. Click “OK” when CMake posts an error and select the path which the variable called `OptiX_INSTALL-_DIR` is set to in the list of variables.
6. Click the “...” button that appears to the right and browse to the install directory of OptiX on the local hard drive (usually something like ”C:/Program Files/NVIDIA Corporation/OptiX SDK 7.2.0”)
7. Click “Configure”. Now some variables are most likely on red background.
8. Click “Configure” again. Now no variables should be on red background.
9. Click “Generate”. The Visual Studio project is now in the “/build” subfolder of where the framework was unzipped.
10. Click “Open Project” or go to the “/build” subfolder of the framework and open the solution file `render_OptiX.sln` in Visual Studio.
11. Right click the “render” project in the “Solution Explorer” window and choose “Set as StartUp project”.
12. Press F5 to compile and run the program. If CMake asks you whether it may update the solution, click “Reload All” and press F5 again when it is done.
13. Right click the “render” project and choose “Properties” and go to “Configuration Properties→Debugging”. Here you can set command line arguments in the textbox called “Command Arguments”. If you run the program using with `-h` as command line argument, you will get a window with help for how to use the program.