Procedural Physically based BRDF for Real-Time Rendering of Glints

The OpenGL 4.5 implementation of the paper: Procedural Physically based BRDF for Real-Time Rendering of Glints.

Xavier Chermain (ICUBE), Basile Sauvage (ICUBE), Jean-Michel Dishler (ICUBE) and Carsten Dachsbacher (KIT).

Accepted for Pacific Graphic 2020 and for CGF special issue.

The implementation reproduces real-time editing of the BRDF parameters, as shown in the video of the paper, from 0:42 to 1:40.

GitHub repository: https://github.com/ASTex-ICube/real_time_glint See https://github.com/ASTex-ICube/real_time_glint_dictgenerator to know how to generate the dictionary used by the renderer.

A high level speudo code is available in the appendix of the paper.

The files are organized as follows:

- `real_time_glint`: the folder containing the paper code (CPU and GPU side),
  - `real_time_glint/shader`: folder of the vertex and fragment shaders (GPU),
  - `real_time_glint/sceneglint.*`: the API / CPU part, with loading of the dictionary in an array texture
- `media`: data
  - `media/dictionary`: the dictionary used in the paper,
  - `media/sphere`: the mesh of the sphere,
- `opengl`: files of the OpenGL framework.

After the build, launch the code with the command parameter `glint`.

The OpenGL framework is based on OpenGL 4 Cookbook and learnopengl.com.

The libraries used in this project are:

- GLM Mathematics Library
- GLFW
- ASSIMP
- tinyexr
- stb
- Dear ImGui
- GLAD

If you modify the `glsl` code, don't forget to re-run cmake.

Requirements. Based on readme.md of OpenGL 4 Cookbook.

To compile this example, you'll need the following:

- The GLM Mathematics Library version 0.9.6 or later. Note that versions prior to 0.9.6 may not work properly because of a switch from degrees to radians. GLM 0.9.5 will work, but you'll need to add `#define GLM_FORCE_RADIANS` prior to including the glm header files.
- GLFW version 3.0 or later.
- ASSIMP version 5.0 or later.

Compiling the example

The example code builds with CMake.

1. Install GLFW by following the instructions on their web site.
2. Install the latest version of GLM. Note that for CMake to find GLM correctly, you need to run the install "build" (e.g. `make install`) or install GLM from your favorite package manager. Otherwise, the CMake config files will not be created/available.
3. Install the latest version of ASSIMP.
4. Download this example code, or clone using git.
5. Run cmake. If cmake has difficulties finding the GLFW or GLM installations, set the variable `CMAKE_PREFIX_PATH` to help cmake find them.
   It can be tricky to get CMake to find the GLM libraries, unfortunately. See below for tips.
6. Compile by running `make`.

Tips for getting CMake to find GLM

When searching for GLM, CMake looks for the files `glmConfig.cmake` and `glmConfigVersion.cmake`.

If you install GLM using a package manager such as Homebrew on macOS, or a Linux package manager the cmake files should already be included.

Otherwise, if you're using the GLM source distribution, you'll have to run GLM through CMake to get it to generate the `glmConfig.cmake` and `glmConfigVersion.cmake` files.

1. Download GLM and extract it to some location: `GLM_SRC`
2. `cd $GLM_SRC`
3. `mkdir build`
4. `cd build`
5. `cmake -D GLM_TEST_ENABLE=OFF -D CMAKE_INSTALL_PREFIX=MY_GLM_LOCATION ..`
6. `cmake --build . --target install`

Replace `GLM_SRC` above with the place where you extracted the GLM zip file, and replace `MY_GLM_LOCATION` with the location where you want to install GLM. This should generate the needed cmake files and install all of GLM to `MY_GLM_LOCATION`.

Tips for compiling for Windows with Visual Studio

- Use the Visual Studio target in CMake: `-G "Visual Studio..."`, open the Visual Studio solution.
- The example code requires a command line argument to choose a scene. When running in VS, be sure to set the 'Command Argument' under 'Properties' for the appropriate scene.

Best solution for Windows + Visual Studio User. From Allen.

Windows doesn't own a unified folder (/usr/local/lib) to search library. Microsoft provides vcpkg for package management to speed up the development process.

1. Install [vcpkg][https://github.com/microsoft/vcpkg] Follow the vcpkg installation instructions. [vcpkg document][https://vcpkg.readthedocs.io/en/latest/] Instance a new powershell console.

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg
PS> .\bootstrap-vcpkg.bat
```

2. Install GLM, GLFW and ASSIMP via vcpkg.

```
x64 version:
vcpkg.exe install glfw3:x64-windows
vcpkg.exe install glm:x64-windows
vcpkg.exe install assimp:x64-windows

x86 version:
vcpkg.exe install glfw3
vcpkg.exe install glm
vcpkg.exe install assimp
```

2. Add VCPKG Toolchain -DCMAKE_TOOLCHAIN_FILE=C:/vcpkg/scripts/buildsystems/vcpkg.cmake Replace `C:/vcpkg/scripts/buildsystems/vcpkg.cmake` with your VCPKG root. After this, CMake could find GLM, GLFW3 and ASSIMP automatically.

OpenGL Function Loading

An OpenGL header file and a function loader for a 4.5 core profile are included with this project. They were generated using GLAD. This loader should also work on MacOS under a 4.1 core profile, but of course not all functions will load. The code has been tested with OpenGL 4.5 on Windows.