Advanced Computer Graphics
Exercise 1 - Basics and Depth Rendering

Handout date: 21.09.2015

Submission deadline: Monday, 28.09.2015, 11:00pm

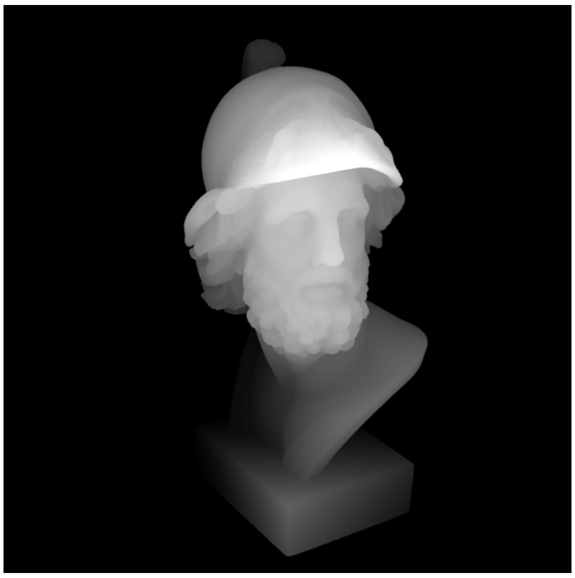


Figure 1: Example of depth rendering.

Note

The first exercise will not be graded and you do not have to submit anything. Its purpose is to get you started and acustomed to the framework.

1.1 Setup of Nori (Ungraded)

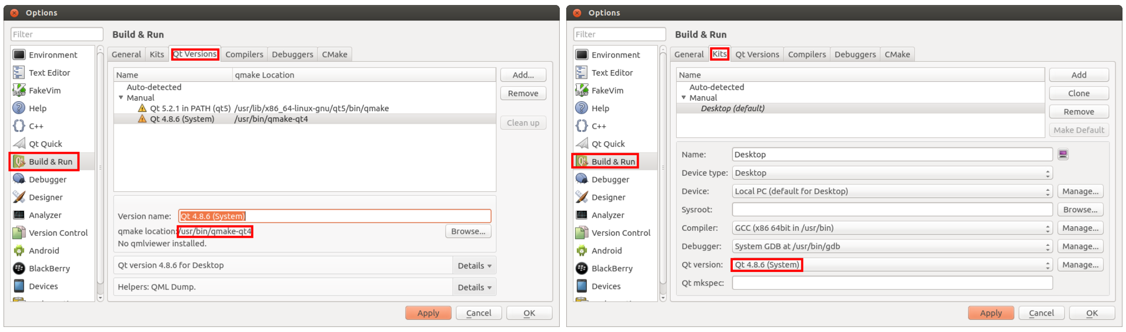
Your first goal is to dive into the rendering framework and try to understand how it works and what are the tools which you are given to work with. The following is a step-by-step tutorial on how to set up the framework.

1. Download the `Exercise1.zip` file from moodle and unzip it into your home directory.

   WARNING: If you're using the lab machines, every file in your home directory is going to be removed once you log out from the computer. So make sure to save your work onto a USB stick or move it into the `myfiles` folder. NOTE however, that you are not able to compile and run nori as long as the framework is stored in `myfiles`. We recommend that you use version control and checkout your project anew each time you work in the lab.

2. Start ”qtcreator” as your development environment.

3. Make sure that QtCreator is using Qt4. Go to “Tools → Options → Build & Run → Qt Versions”. If Qt4 is not listed, add it (it is in `/usr/bin/qmake-qt4`) and press “Apply” (see Fig. 2 on the left). Under the “Kits” tab, change the default kit’s Qt version to Qt4 and press “OK” (see Fig. 2 on the right).



Name: Qt 5.2.1 in PATH (qt5) — qmake Location: /usr/lib/x86_64-linux-gnu/qt5/bin/qmake
Name: Qt 4.8.6 (System) — qmake Location: /usr/bin/qmake-qt4
Version name: Qt 4.8.6 (System)
qmake location: /usr/bin/qmake-qt4
No qmlviewer installed.
Qt version 4.8.6 for Desktop
Helpers: QML Dump.

Name: Desktop
Device type: Desktop
Device: Local PC (default for Desktop)
Sysroot:
Compiler: GCC (x86 64bit in /usr/bin)
Debugger: System GDB at /usr/bin/gdb
Qt version: Qt 4.8.6 (System)
Qt mkspec:

Figure 2: Setup of QtCreator: (left) adding Qt4, (right) changing the default kit to Qt4.

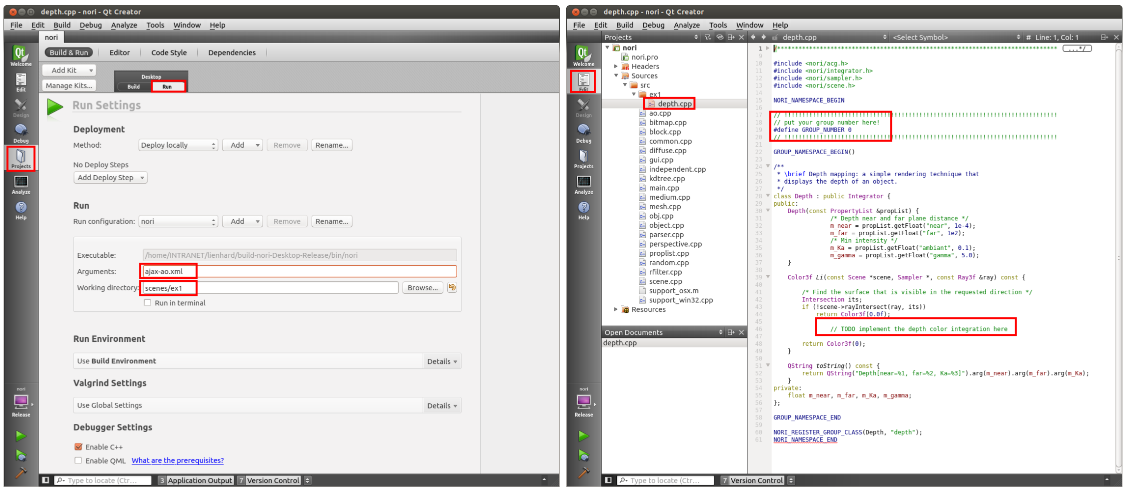
4. Load the `nori.pro` file (this is the project file that defines the source and header files and the required libraries similarly to a Makefile). You can chose the build directory by clicking “Details”. Once you are happy with the setup click “Configure Project”.

5. Compile the project (Build → Build Project)

6. Change the working directory of the project (left tool bar “projects” and “run settings”, see Fig. 3) to `scenes/ex1` folder within nori.

7. On the same page add the name of the scene you want to load as command line arguments (e.g., `ajax-ao.xml`, see Fig. 3).

8. Now you can run nori and create your first image (see Fig. 4).



Method: Deploy locally
Run configuration: nori
Executable: /home/INTRANET/lienhard/build-nori-Desktop-Release/bin/nori
Arguments: ajax-ao.xml
Working directory: scenes/ex1

```cpp
/*******************************************************************...*/

#include <nori/acg.h>
#include <nori/integrator.h>
#include <nori/sampler.h>
#include <nori/scene.h>

NORI_NAMESPACE_BEGIN

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// put your group number here!
#define GROUP_NUMBER 0
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

GROUP_NAMESPACE_BEGIN()

/**
 * \brief Depth mapping: a simple rendering technique that
 * displays the depth of an object.
 */
class Depth : public Integrator {
public:
    Depth(const PropertyList &propList) {
                /* Depth near and far plane distance */
                m_near = propList.getFloat("near", 1e-4);
                m_far = propList.getFloat("far", 1e2);
                /* Min intensity */
                m_Ka = propList.getFloat("ambiant", 0.1);
                m_gamma = propList.getFloat("gamma", 5.0);
    }

    Color3f Li(const Scene *scene, Sampler *, const Ray3f &ray) const {

        /* Find the surface that is visible in the requested direction */
        Intersection its;
        if (!scene->rayIntersect(ray, its))
            return Color3f(0.0f);

                // TODO implement the depth color integration here

        return Color3f(0);
    }

    QString toString() const {
        return QString("Depth[near=%1, far=%2, Ka=%3]").arg(m_near).arg(m_far).arg(m_Ka);
    }
private:
    float m_near, m_far, m_Ka, m_gamma;
};

GROUP_NAMESPACE_END

NORI_REGISTER_GROUP_CLASS(Depth, "depth");
NORI_NAMESPACE_END
```

Figure 3: Screenshot of QtCreator: (left) settings to change the working directory and arguments, (right) exercise 1.2.

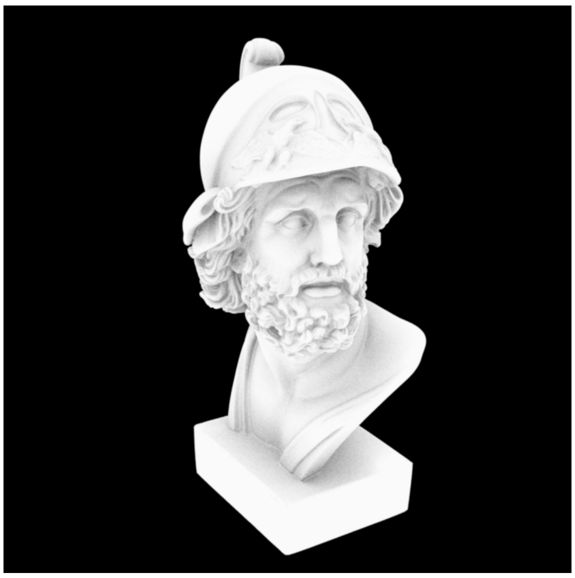


Figure 4: Ambiant Occlusion rendering of Ajax using the `ao` integrator.

To compile the code on your own machine you will most probably need to install Qt 4 and the OpenEXR library. On linux these libraries can be found in the package manager. To install the libraries on Mac OSX you can use homebrew. On Windows you can use Visual Studio and you will have to download precompiled libraries for Qt and OpenEXR (or compile them yourself).

1.2 Implement Your Own Integrator (Ungraded)

Now put `depth.xml` as command line argument (see step 7) to load the scene defined in this file. Instead of the ambient occlusion integrator this scene selects a depth integrator, which you will implement in this exercise. You can find the file under `src/ex1` (see Fig. 3 right screenshot).
The first thing you should do is **insert your group number in line 19** in this files (see Fig. 3 on the right).
In this section, you will need to implement the method `Color3f Li(const Scene *scene, Sampler *sampler, const Ray3f &ray)` in `src/ex1/depth.cpp` which does one step (i.e., for one ray) of picture integration. It should render the depth of the model. The depth should be rescaled linearly such that 0 corresponds to the far plane distance and 1 to the near plane distance. To get a better visual result you can apply a gamma correction to the rescaled depth using $Ka + (1 - Ka) \times Kd^g$, where Ka is the ambient color, Kd is the rescaled depth value and g is the gamma value.
Your implementation starts on line 46. A reference solution for the scene `scenes/ex1/depth.xml` is shown in Figure 1.

Hints

The result will certainly depend on the scene configuration. Try to use the provided parameters in the `xml` scenes as these are appropriately set for the each specific scene. To understand how the framework works and how parameters are passed you can look at the `ao` renderer (for *ambient occlusion*) implemented in `src/ao.cpp` and `scenes/ex1/ajax-ao.xml`.

1.3 Understanding the Nori Framework (Ungraded)

Here we don't ask you to implement any more code, but to show your understanding of the framework and the pipeline within the nori framework.

1.3.1 Understand Central Classes

Take some time to understand the goal of these classes

- the `Integrator` abstract class (`include/nori/integrator.h`)
- the `Sampler` abstract class (`include/nori/sampler.h`)
- the `Scene` class (`include/nori/scene.h` and `src/scene.cpp`)

1.3.2 UML Sequence Diagram

Generate an UML sequence diagram summarizing the interaction of the main components of the rendering framework. The diagram should describe the process of calculating the color for a single pixel in your output image.
Start at block.cpp line 202: the loop iterates over all pixel in one image block and creates a number of samples for one pixel. From there visualize the components within nori involved to compute the final color value.
If you need to refresh your knowledge about UML you can start here: Introduction to sequence diagrams, but there are plenty of other sources to get information about UML.

Hints

You can generate a UML diagram with a tool of your choice. If you don't own any UML editor, you can use one of the many available free online editors such as

- `https://www.draw.io/`
- `http://www.asciiflow.com/`
- `http://www.jrromero.net/tools/jsUML2`
- `http://alexdp.free.fr/violetumleditor/page.php`
- `http://bramp.github.io/js-sequence-diagrams/`

This list is obviously non-exhaustive. You can use any tool which works for you.