

Installation Guide and User Manual

Multiple Live Video Environment Map Sampling

Tomáš Nikodým

Czech Technical University in Prague, Czech Republic

February 7, 2014

1 Setting up development environment

In order to compile and deploy code for Nokia N900, it is necessary to install and configure the development environment. The steps are described in detail at [FCa] and reviewed in the appendix *Installation Guide* of [Nik12]. Below, we give a summary of the development environment and FCam set-up. To make things clearer, the steps that are to be done on the development machine are marked *[Desktop]* and those to be done on the Nokia N900 are marked *[N900]*.

1. Download and install Nokia Qt SDK. *[Desktop]*
 - The installation includes QtCreator IDE, N900 development libraries, and USB Driver.
2. Update the N900 firmware *[N900]*
 - Required version: 1.3 (20.2010.36-2) or later.
3. Install Mad Developer on N900 *[N900]*
 - This enables remote debugging and deployment on N900 from QtCreator.
4. Set up the connection between the development machine and the N900
 - Connect the N900 via a USB cable.
 - Set 'Windows Networking' mode in the Mad Developer. *[N900]*
 - A new LAN connection should be detected on your development machine. Set a fixed IP address '192.168.2.14'. *[Desktop]*
5. Set up the QtCreator *[Desktop]*
 - Go to Tools → Options and select the tab Linux Devices (depicted in Figure 1). Set the IP address of the N900 (the default when connected via a USB cable is 192.168.2.15). To connect for the first time, use the Mad Developer on your N900 to generate a temporary password. Then deploy a public key and set the authentication type to *key*, so that you no longer need to use a password.
 - A detailed description with screenshots is given in [Nok].
6. Install the FCam drivers on N900 *[N900]*

- The easiest way is to install the FCamera application using the Application Manager, as it comes with all the drivers we need.

7. Download the FCam development library [Desktop]

- It can be downloaded from [Mae].
- Create an environment variable FCAM_PATH that points to the extracted folder, so that our project can find the libraries.

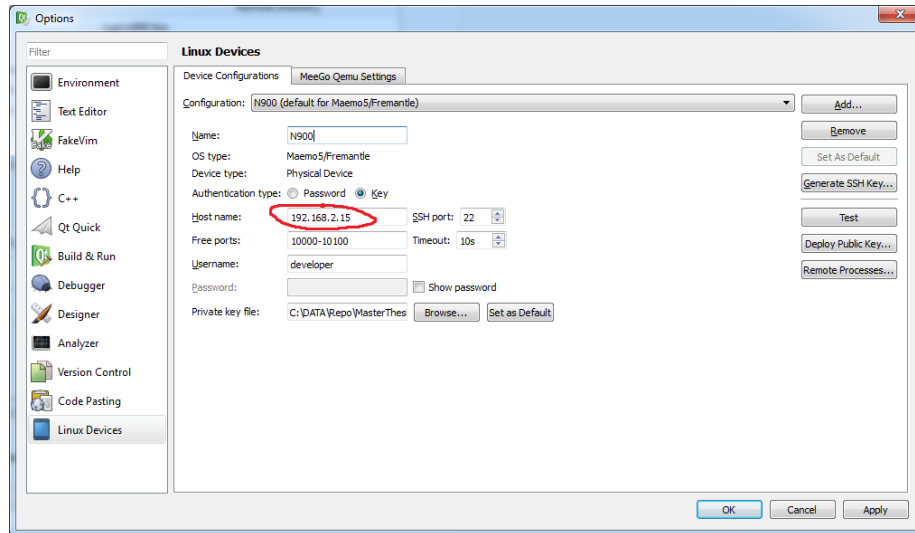


Figure 1: Setting up the QtCreator - adding a remote device.

2 Compiling the server side

The project file for the environment map acquisition and processing application is located in `src/N900/N900.pro`.

The project uses FCam to control the programmable camera of the Nokia N900, so in order to compile the project, you need to download and link the FCam development library. If you have not done so already, set the FCAM_PATH environment variable to the path where the FCam is located, as described in Section 1.

Open the N900.pro project in QtCreator, connect the N900 via a USB cable, set 'Windows Networking' mode in the Mad Developer and hit the *Run* button in the QtCreator. If you successfully completed all the steps described in Section 1, the application should compile, deploy and run without any problems. Open the 'Compile Output' panel to see the compilation progress. First, the application should compile and link. Then, QtCreator connects to the N900 and deploys the application. Finally, the application is started on the N900. You should see a lot of messages popping up in the 'Application Output' panel. To suppress the debug messages, set the verbose level of the logger to any of the six predefined levels ¹.

¹In code, this can be done in the constructor of the Logger class. Refer to refman.pdf for details.

3 Compiling the client side

There are two sample applications that connect to the N900 and read the processed data. The first of them, referred to as *Visualization*², displays position of samples (e.i. directional light sources) and optionally displays a tonemapped HDR environment map on the background. The second sample application, referred to as *Renderer*³, renders a bunny illuminated by a set of directional light sources computed on the N900.

Both of the sample applications use the Qt library and have no other dependences. No extra steps are required to compile and link these two applications.

4 Usage

If you successfully completed the steps described in sections 2 and 3, you should now be able to run the acquisition and processing application (referred to as *server side*) on the Nokia N900, as well as the two testing applications (referred to as *client side*) on the desktop. In this section, we describe the interface of these applications.

4.1 Server side (N900)

The communication protocol and the configuration of sampling algorithm are described in the appendix *User manual* of [Nik12].

When started, the application reads the intrinsic parameters of the camera model from the file *intrinsic.txt*. See Section 5 for details on camera calibration. The application then listens for incoming connections on port 8080. The communication protocol is compliant with the HTTP protocol. To test whether the application is running, connect to it using a web browser. The N900 should send a simple HTML page as a reply, depicted in Figure 2. The page contains links to the configuration of the sampling algorithm.

The basic requests are as follows.

IP:port/data

Reads sampling data in plain text format.

IP:port/hdr

Reads an HDR environment map in .hdr format.

IP:port/change?[param]=[value]

Changes parameters of the sampling algorithm.

The protocol for sending sampling data in plain text format (*data* request) changed slightly since the writing of [Nik12]. The format is given in Algorithm 1⁴. Refer to the source code of `SamplingData::Deserialize()` as ultimate reference.

4.2 Client side

The interface is the same for all testing applications. The GUI is self-explanatory; below, we describe the most important dialogs, accessed from the menu.

[Remote→Connect to host] Connects to the specified host and reads processed results until stopped by the user.

²Project file located at *src/Visualization/Visualization.pro*

³Project file located at *src/Renderer/Renderer.pro*

⁴Current protocol version is 2.

Algorithm 1 Format of the response to *data* requests.

```
<protocol version>
<algorithm>
<id>
<sample count>
<resX resY>
<x y luminance r g b>
<x y luminance r g b>
...
<x y luminance r g b>
```

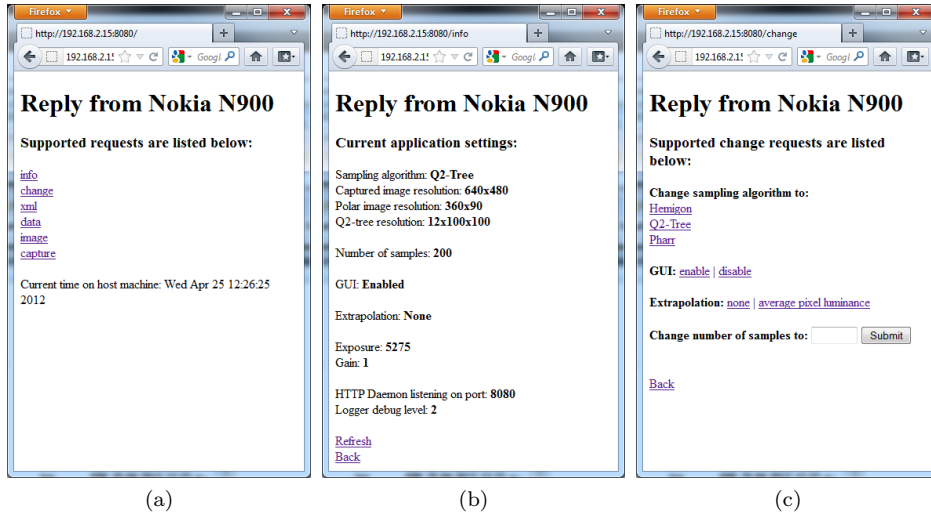


Figure 2: A set of screenshots from the web based configuration of the sampling algorithm.

[Remote→Connect to multiple hosts] Connects to multiple hosts and merges sampling data ⁵. The list of host addresses is specified as IP:port pairs, separated by whitespace. If the checkbox *Use advanced Q2-Tree merging* is checked, the merging algorithm described in [Nik13] is used. Otherwise, a naive merging algorithm is used (applies only to Q2-Tree sampling). The *samples* field specifies the required number of samples (e.i. light sources).

[Remote→Connect to environment map provider] Connects to the specified host and reads the environment maps in .hdr format.

Note that when the application is started without any command line arguments, the multicam feature is not available. Run the application with the '*-m*' option to enable this feature.

⁵Available only when the application is started with the '*-m*' command line argument

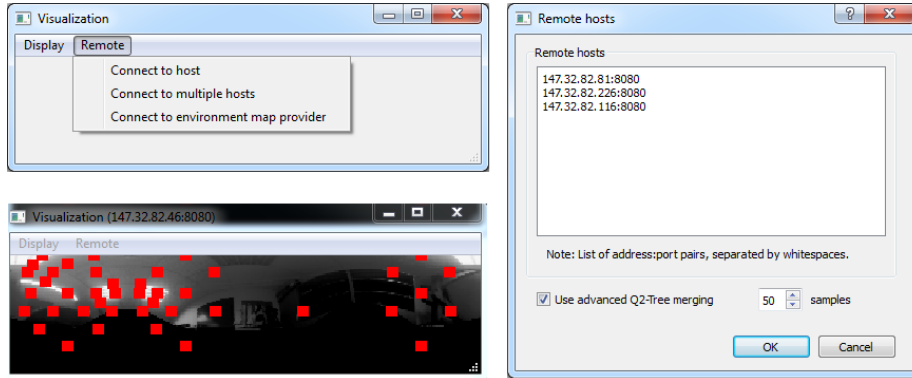


Figure 3: A set of screenshots from the *Visualization* application.

5 Camera calibration

The file containing the intrinsic parameters of the camera model (*src/intrinsics.txt*) is automatically copied from the development machine to the N900 during deployment. In order to recompute the camera calibration, use the stand-alone application based on OpenCV. The project ⁶ can be compiled and linked using the QtCreator. The application has a simple command line interface and requires a set of checkerboard photographs taken from various viewpoints. A script that semi-automizes the process of taking multiple photographs and downloads the photographs from the N900 via *wget* is available ⁷. The application outputs the intrinsics file. It is also possible to modify the file directly on the N900 without the use of the SDK.

6 Developer's manual

We provide an API for communication with the server based on the Qt library [Dig]. All of the API classes inherit from the *IClient* abstract class, as illustrated in Figure 4. The *GenericHttpClient* abstract class implements routines for initializing and maintaining the connection to the server over HTTP, as well as error handling. The classes *HdrDownloader*, *HttpDataClient*, and *HttpQ2TreeClient* implement routines for parsing of received data, which are specific for any particular type of data request (HDR image, list of directional light sources, Q2-Tree). The *MultiCamClient* class allows for simultaneous connection to multiple servers and implements merging of received data. The remaining classes that implement the *IClient* interface but are not described here serve only for testing or debugging purposes.

The *IClient* abstract class declares several public members, of which the following two are essential:

- **Signal 'DataReady':** this signal is emitted every time the client receives updated data from the server. At the time when the signal gets emitted, the data are already parsed and prepared into a form independent of the underlying transfer protocol. In the slot handling this signal, the programmer should use the *ReadData()* method to read the updated data.

⁶Project file located at *src/CameraCalibration/CameraCalibration.pro*

⁷Script located at *scripts/calibration/takeShots.bat*; press the camera button on your N900 to take the photo, followed by any key in the terminal to download the photo. The default USB IP is hardcoded in the script so change it if the device is connected over WiFi.

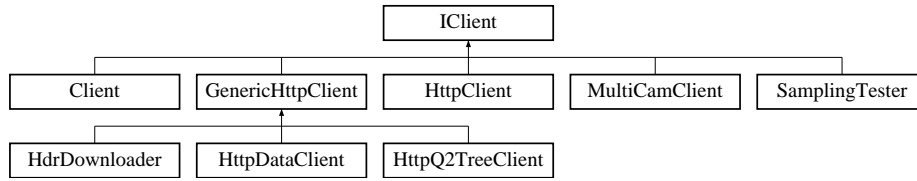


Figure 4: Inheritance diagram

- **Method 'ReadData()'**: this method reads the latest data.

The workflow of a typical usage is as follows.

1. Register a handler for the *DataReady* signal.
2. Connect to the host (or multiple hosts ⁸) using the *ConnectToHost()* method.
3. On *DataReady*, read the updated data using the *ReadData()* method.

The client maintains the connection to the server until stopped by a call to the *Stop()* method or by the destructor. The *DataReady* signal is emitted once per frame processed on the server device. The use of the data read by the *ReadData()* method depends on the particular application. Typically, the positions and intensities of directional light sources are transferred to the GPU for illumination computation.

See the attached file 'API documentation.pdf' for a complete documentation of the API, and 'refman.pdf' for a doxygen generated documentation of the entire project.

References

- [Dig] Digia. Qt Project. [Online; accessed 8-January-2013], URL: <http://qt-project.org/>.
- [FCa] FCam. FCam API - Getting started. [Online; accessed 3-January-2013], URL: <http://fcam.garage.maemo.org/gettingStarted.html>.
- [Mae] Maemo. FCam Camera Control API: Project Filelist. [Online; accessed 3-January-2013], URL: https://garage.maemo.org/frs/?group_id=1693.
- [Nik12] Tomáš Nikodým. Global Illumination Computation for Augmented Reality. Master's thesis, Czech Technical University in Prague, May 2012.
- [Nik13] Tomáš Nikodým. Multiple Live Video Environment Map Sampling, 2013. [Not yet published].
- [Nok] Nokia. Setting Up Development Environment for Maemo - Configuring Connections in Qt Creator. [Online; accessed 3-January-2013], URL: <http://doc.qt.digia.com/nokia-qtsdk-1.0.1/creator-developing-maemo.html#configuring-connections-in-qt-creator>.

⁸The capability to connect to multiple hosts simultaneously is implemented in the MultiCamClient class via method *ConnectToHosts()*.