2018
IJS-DP: 12882

Research reactor simulator

DEVELOPERS MANUAL

JAN MALEC, DAN TOŠKAN, LUKA SNOJ

Table of contents

1. Introduction

The Research Reactor Simulator, developed by Jozef Stefan Institute (IJS) is a real-time simulation code of a Triga-type reactor in real time. The simulator is documented in three manuals:

- **User manual** – Describing the operation of the reactor simulator from the user's perspective
- **Physics manual** – Describing the physics models implemented in the simulator
- **Developers manual** (not yet released) – Description of the software architecture with instructions for compiling and running the code

This document is the Developers manual for the Research Reactor Simulator.

2. Obtaining and building

The code is distributed from a git repository or as a compressed archive on request from the developers.

2.1. Linux

The project source code is located in the "nanogui" folder. Use "cmake" to create build configuration in any folder.

On Ubuntu Linux, the following set of commands should build Research Reactor simulator:

```
apt update && apt update && apt -y install make git cmake xorg-dev libglu1-mesa-dev
mkdir nanogui/build && cd nanogui/build
cmake ..
make
```

2.2. Windows

CMake [1] can be used to generate the build files. While on Linux the product of CMake is a makefile, on Windows CMake generates project files for Visual studio 2017 or Visual Studio 2019.

The first step is to download the latest version of CMake from https://cmake.org/download. When CMake is installed, open the GUI, choose source directory, target build directory and press "Configure". When asked, pick the target version of Visual Studio. If the configure process has been completed without any errors, press "Generate".

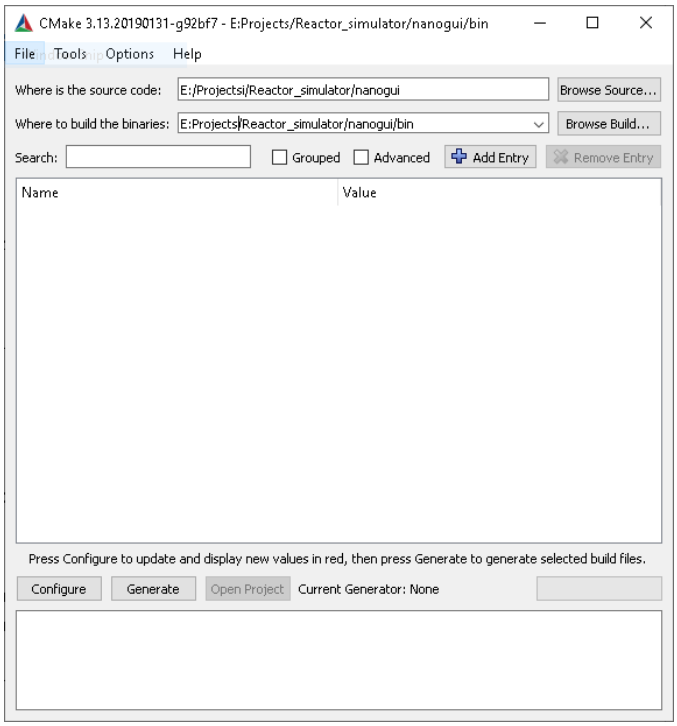

Figure 1 CMake Windows GUI

When successful, CMake When successful, CMake When successful, CMake will generate multiple project files. Locate a file named "NanoGUI.sln" and open it in Visual Studio.

In the Visual Studio interface select "SimulatorGUI" as a start-up project and build the project like any other Visual Studio project. "Debug" and "Release" configurations are both available.

3. Program flow

The simulator uses a feedback loop to synchronize the simulation time with the computer system time. The main application loop updates the graphical widgets that display the simulation results and propagates the simulation. A flowchart depicting the operation of simulation is depicted on Figure 2. Before each frame is drawn, the time t_f needed to execute the previous simulation step is calculated and used to propagate the simulation so it matches the real-time.

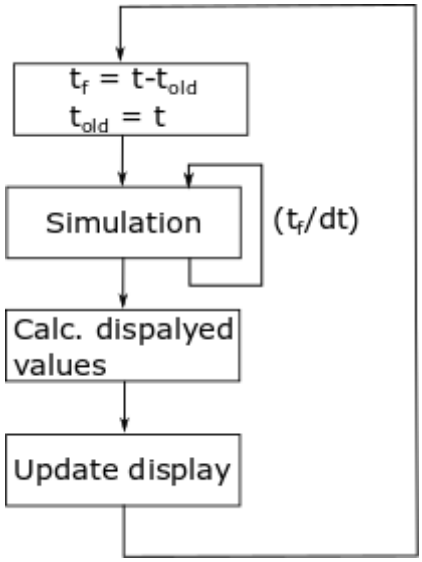

Figure 2 Flowchart depicting operation of the simulator. The current time is marked with t, the time expired from the previous frame is marked with t_f and the time step used to solve the kinetic equations is dt.

The simulation step consists of the following sub steps needed to propagate the reactor power, reactivity, fuel temperature and temperature in the water tank.

The number of steps needed to propagate the simulation is $t_f\ S/dt$, where dt is the a pre-defined, fixed time step and S is a multiplier that can be used to speed up the simulation. On a personal computer with Intel i7-7700K CPU and Nvidia GeForce 1050 Ti graphics card, the time required to draw the simulation results is longer than the time required to propagate the physics simulation with the time by at least two orders of magnitude when running at normal simulation speed, so the effect of the physics simulation on graphics refresh rate is neglible. The simulation results are added to a circular buffer that is implemented as a fixed, allocated block of memory in the heap. An index with periodic boundary conditions is used to keep the current position in the buffer.

Each simulation step consists of the following operations

1. Pulse handling routine: Check if the reactor is currently pulsing and if so, advance the pulse timer. When the pulse timer expires, scram the reactor and calculate pulse peak power, peak temperature, full width at half maximum (FWHM) and the amount of energy that was released.
2. Re-calculate the fuel temperature based on power from the previous iteration.
3. Move control rods if the current rod position does not match the set position.
4. Propagate the Xenon and Iodine concentrations.
5. Re-calculate the reactivity.
6. Numerically evaluate the pint kinetic equations (1)
7. Propagate the temperature of the water in the fuel tank.

8. Check operational limits and initiate emergency shutdown if the operational limits have been violated.
9. Advance the simulation timer and push new results to the buffer.

After each simulation step it is necessary to re-calculate all the simulation results that are not saved in the simulation buffer, but only displayed using the graphical widgets. The average reactor period is calculated is calculated using a running average of the last 700 reactivity points. Calculating the average is necessary to avoid quick jumps that cannot be observed at JSI Triga instrumentation due to the nature of the analogue meter. Just before the next frame is drawn, an algorithm iterates through the elements and updates an array with a new maximum power value is such value has been found and pushes any maximum values that correspond to simulation results no longer in memory out of the array. The array of maximum values is used to scale the reactor power plot.

4. File and folder structure

The Research Reactor Simulator is using a modified version of the NanoGUI [2] library. Since the library was so heavily modified that practically no classes are used in the original form, the library is frozen in time and not up to date with the mainstream version. While there are future plans to mainstream the NanoVG library, no such plans are currently foreseen for the NanoGUI widgets.

All source files are in the <root>/src directory. No distinction is being made between the original NanoGUI classes and the classes that were developed only for use in the Research Reactor Simulator. The main source files are "Simulator.cpp", which holds the physics model and "SimulatorGUI.cpp" where the main graphical user interface is constructed.

The header files are located in the <root>/include directory. The header files for the graphical widgets are separated other header files and placed in the <root>/include/nanogui directory.

5. REFERENCES

1. CMake https://cmake.org
2. NanoGUI https://github.com/wjakob/nanogui