# *Lightweight 3D Learning Environment, under Gnu/Linux, For Instructing Students In Microcontroller Programming*

Gabriel Muresan
S.C. IPA S.A., Cluj Subsidiary
Cluj-Napoca, Romania
ipacluj@automation.ro

***Abstract*— *Learning to program microcontrollers fast and well, almost always requires laboratory practice. Traditional laboratory instruction is hindered by limiting factors like high laboratory maintenance costs and limited access to material resources. This paper describes a design model of a lightweight and modular application representing a 3D computer-simulated environment for instruction in microcontroller programming.***

***Keywords-lightweight, plugin style architecture, 3D, experimental electronic circuit (EEC).***

## I. INTRODUCTION

Generally, 3D computer-simulated learning environments have their value in the learning process because *"It is possible in a virtual world to build instructional design into aspects or objects with which the learner must interact"[1]*. The application described in this paper, by means of simplification of user interaction with the 3D objects through a lightweight and intuitive user interface, may accelerate the learning process by allowing the students to concentrate on the experiments rather than the ways the software can be used.

The 3D virtual environment must solve the majority of the problems associated with traditional laboratory instruction, and can provide students with a good level of understanding of the experiments as a real laboratory environment.

Last but not least, in this paper it is given a high level of significance to the operating system and all the tools necessary to implement and run the software application. In respect for the *values promoted by the Free Software Foundation* [1], only free (libre) tools and operating systems will be used.

## II. FREE TOOLS AND LIBRARIES

The majority of Gnu/Linux based operating systems are ideal for programming. Their repositories contain a multitude of tools, software libraries, compilers and integrated development environments; all easy to install and configure. The "Free software/Open Source" community provides excellent support and documentation for components necessary to develop and deploy free software.

For the development of a 3D environment, the following components have to be taken into consideration:

- 3D rendering library
- languages to describe 3D objects
- libraries to implement user interfaces
- a programming language to support all the libraries involved in the development of the application

### *A. The software library for 3D rendering.*

In choosing a 3D library, the factors that are to be taken into account are: performance, the existence and quality of documentation, the platforms that supports the 3d graphic library, the existence of a standardized format for describing graphic objects and object interactions, the licensing etc. Based on above mention factors the choice is Coin3D library.

*"Coin is an OpenGL-based, 3D graphics library that has its roots in the Open Inventor 2.1 API, which Coin still is compatible with. Open Inventor, it is a scene-graph based, retain-mode, rendering and model manipulation, C++ class library, originally designed by SGI. It quickly became the de facto standard graphics library for 3D visualization and visual simulation software in the scientific and engineering community after its release. It also became the basis for the VRML1 file format standard." [3]*

*"Coin is based on the API of the Open Inventor library (see fig 1) , but was developed from scratch independently before SGI Open Inventor became open source (free software) ." [3]*

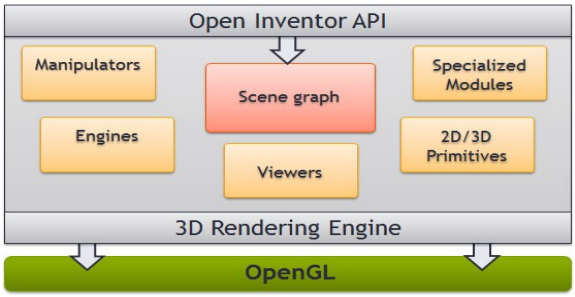


**Fig 1.** Coin 3D architecture.

1 Note: the essence of the values promoted by FSF is:
" "Free software" means software that respects users' freedom and community. Roughly, it means that **the users have the freedom to run, copy, distribute, study, change and improve the software**. Thus, "free software" is a matter of liberty, not price. "

*"Coin reached the goal of Open Inventor 2.1 compatibility in the autumn of the year 2000, and has since then been extended with a huge set of additional features, ranging from 3D sound support to GLSL shader support, additional file formats like VRML97, and a large number of internal changes for keeping up with the newer, more optimized OpenGL rendering techniques that were not available in the earlier days. Another term associated with Coin is "Coin3D", which is the term used on the larger group of libraries that all fall under the same license as Coin. Coin is the core of Coin3D. The development of Coin was in the beginning primarily done on Linux and IRIX systems, but is now mostly developed under Linux, Windows with Cygwin, and Mac OS X systems." [3]*

*B. The 3D modeling language.*

*"VRML (the Virtual Reality Modeling Language) has emerged as the de facto standard for describing 3-D shapes and scenery on the World Wide Web. VRML's technology has very broad applicability, including web-based entertainment, distributed visualization, 3-D user interfaces to remote web resources, 3-D collaborative environments, interactive simulations for education, virtual museums, virtual retail spaces, and more. VRML is a key technology shaping the future of the web. "[2]*

Various elements needed to describe a virtual world: basic shapes (cubes, spheres, cones , cylinders), advanced shapes (generic shapes, elevation grids, extrusions), structures describing transformations (translation, rotation, scaling) and structures describing animation and interaction (time sensors, interpolators, drag sensors), are all stored into a scene-graph (see Figure 2).

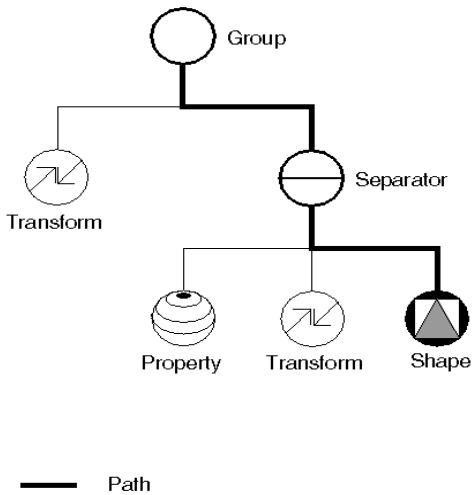


**Figure 2.** VRML simple scene-graph (1 object; 2 transformations and one property)

The resemblance in structure with the JavaScript Object Notation (JSON) and the use of curly brackets to enclose structures makes the Virtual Reality Modeling Language (VRML) very easy to read, understand and use. The VRML files starts with a header containing informations about the language name, language standard version and the variable-width encoding. After the header, the file contains descriptions of nodes (objects), prototypes, animation routes and comments preceded by a hash-tag (#).

VRML file example:

```
#VRML V2.0 utf8

# A directional light source, shining down
DirectionalLight {
       direction 0 -1 0
}
# yellow cone rotated about 57 degrees on Z axis
Transform {
  # axis   X Y Z  Degrees Radian
  rotation 0 0 -1 1
  children [
    Shape {
      appearance Appearance {
        #                  color        R G B
        material Material {diffuseColor 1 1 0}
      }
      geometry Cone {
        bottomRadius 2
        height 8
     }
    }
  ]
}
```

The text above can be interpreted with Coin3D and the result rendered on the screen. (see Figure 3)

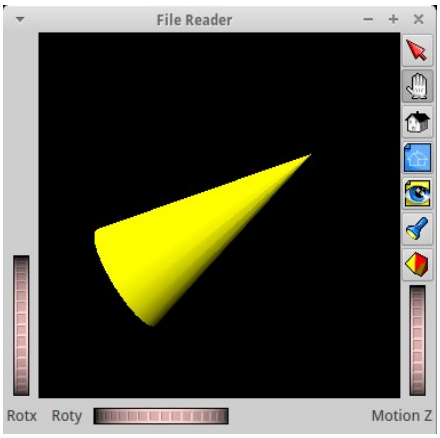


**Figure 3.** VRML file content rendering with a simple Coin3D based viewer.

The various 3D objects (microcontroller representations, circuit boards, electronic components etc) used in the learning environment described in this paper, can be loaded from VRML files or can be generated programmatically.

*C. Programming language and user interface library.*

A multitude of libraries useful for this application are implemented in C++. The excellent speed of C++ encoded applications and the possibility to dynamically load C libraries at runtime are the reasons for choosing C++ language to encode the application.

For the graphical user interfaces, Qt 4.x libraries will be used. Qt is a cross-platform application and UI framework for developers using C++ or QML, a CSS & JavaScript like language.

## III. DESIGN CHOICE

To ensure the lightweightness of the learning environment, while providing a greater number of simulated microcontrollers, a plugin architecture is adopted . A plugin is a software component that can be loaded by a host application at run-time or during startup. The plugin provides additional features to the application, in this case a greater number of microcontrollers and components to experiment with in the 3D learning environment.

Through the graphical user interface, a plugin manager is instructed to load the functionality and 3D representation of a microcontroller and an experimental circuit to test the microcontroller. (see Figure 4) The application also runs a simulation/animation loop-cycle.

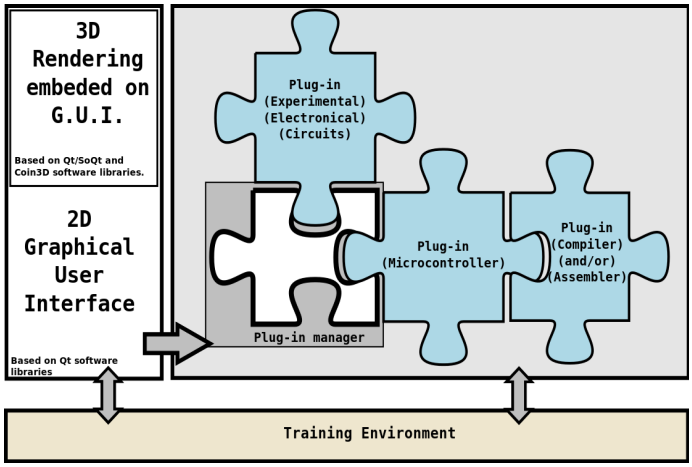


**Figure 4.** Plugin-GUI-Application ensemble.

The C++ library used to dynamically load shared libraries is the libdl library:

```
void *lib_handle;
lib_handle = dlopen("somelib.so", RTLD_LAZY);
```

The dlopen() function gains access to executable object file. Special arguments are needed to compile programs capable of loading libraries dynamically.

```
g++ -rdynamic -o <prog> <prog>.cpp -ldl
```

## IV. *PLUGINS*

### *A. Plugin Interfaces.*

*1) For microcontroller:*

As a parameter, the controller plugins should receive the address to a list of floating point values. (see fig 5) The changes of value (representing a voltage) for each available pin, is operated inside the plugin. These value changes are related to the execution of the machine code provided to the microcontroller object.

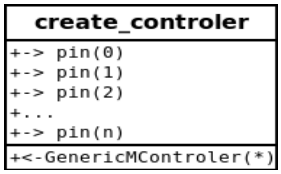


**Figure 4.** Plugin I/O

*C++ function prototype:*

```
GenericMControler* create_controller (
        std::vector<float> * pins,
);
```

The plugin returns a pointer to a generic microcontroller which is defined as an abstract class that serves as base for all specific microcontrollers implemented by the plugin programmer. (see Figure 4)

*C++ generic microcontroller class prototype:*

```
class GenericMControler {

 protected:

        std::vector<float> * _pins;

 public:

        GenericMControler(std::vector<float> * pins);

        virtual SoSeparator* get3Dobject() = 0;

        virtual void update() = 0;

        virtual void loadProgram(std::string program) = 0;

        virtual void fetch() = 0;

        virtual void decode() = 0;

        virtual void execute() = 0; };
```

*2) For compiler:*

The compiler plugin receives the program source as a string parameters and returns a program in machine language. At the choice of the programmer, the actual compiling can be executed inside the plugin or can be delegated to an external compiler.

C++ function prototype:

```
std::string  compiler(std::string source_code);
```

*3) For experimental electronic circuits (EEC's):*

As a first parameter, the plugins should receive the address for the list of floating point values (see fig 5). The changes of value (representing a voltage) for each available pin in the experiment boards socket, will be interpreted inside the plugin. A change of state in the 3D graphics representing the experimental electronic circuit (EEC) may also occur.

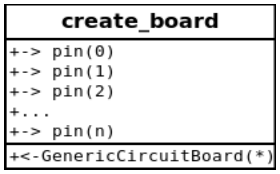


**Figure 5.** Plugin I/O

The plugin returns a pointer to a generic electronic circuit board which is defined as an abstract class, that serves as base for all specific experimental electronic circuits implemented by the plugin programmer.

C++ generic circuit board class prototype:

```
class GenericCircuitBoard
{
 protected:
        std::vector<float> * _socket;
 public:
        GenericCircuitBoard(std::vector<float> * socket);
        virtual SoSeparator* get3Dobject() = 0;
        virtual void update() = 0;
};
```

*B. Plugin development and deployment walktrough.*

*1) Files involved*

The programmer must then include the *'plugin.h'* file in their '.cpp' project then define and implement a class for a specific microcontroller/experimental electronic circuit (EEC), implement the compiler function, the microcontroller/EEC object instantiator functions considering the following guidelines :

- The programmer must create derived classes from *GenericMControler* and *GenericCircuitBoard*. These new classes will describe specific microcontrollers and specific EEC's.
- The get3Dobject() function of the implemented microcontroller and the EEC has to return valid 3D VRML scene-graphs for both microcontroller and EEC's.
- The programmer is free to choose the modality in wich the microcontrollers internal processes are simulated.
- The programmer must create a new class, for the specific microcontroller he chooses to simulate, as a derived class from GenericMControler; and call the non-default constructor of the base class with a reference to a vector of floating point numbers.
- The compiler function and the microcontroller instatniator function must be defined and implemented inside a **extern "C"{ .. }** construction.

The programmer will receive valid pin arrays from the main applications in the form of a pointer to a C++ standard vector of floats. It is the programmer's obligation to use this vector to 'publish' the voltages at the microcontroller's pins at any change of state in the simulated microcontroller.

*2) Shell script for library creation under GNU/Linux.*

*#!/bin/bash*

```
#
#  Coin3D and SoQt related
#
CPPFLAGS_COIN=`soqt-config --cppflags`
CXXFLAGS_COIN=`soqt-config --cxxflags`
LDFLAGS_COIN=`soqt-config --ldflags`
LIBS_COIN=`soqt-config --libs`

#
#  [ -Wall ] display all warnings
#  [ -fPIC ] generate Position Independent Code
#
g++ -Wall -fPIC -c $1 -o $2.o

#
#  [ -Wl,-snoname,<name> ] instuct the linker the
#  linker to generate a Dynamic Shared Objects
#  instead of an application
#
g++     $CPPFLAGS_COIN $CXXFLAGS_COIN
        -shared -Wl,-soname,$3.so.1 -o
        $3.so.1.0  $2.o
        $LDFLAGS_COIN $LIBS_COIN

# create links
ln -sf $3.so.1.0 $3.so.1
ln -sf $3.so.1.0 $3.so
```

Usage:

```
~$: ./make2.sh <plugin_file>.cpp
        <name_oject_file_no_extension>
        <name_dynamic_lib_no_extension>
```

Example:

```
~$: ./make2.sh ctest.cpp ctest libctest
```

This creates the dynamic library and symbolic links to it.

## V. Design Details

*A. UI concept*

In order to construct the most intuitive and user friendly interface, the following guiding principles *[5]* are taken into consideration:

- The structure principle: the user interface should be organized in a meaningfull way by grouping related elements.
- The visibility principle: no unnecessary information should be added to the user interface.
- The simplicity principle: only language and descriptions easy to understand are to be used.

It is easier to construct a simpler, cleaner and more intuitive interface on a single 'page', without menus and hidden forms. The proposed design contains on the same frame all the buttons, editing fields and the 3D rendering area (fig 6).

UI Groups:

- (1) Rendering area
- (2) Experiment selection area
- (3) Programming area (code window and generated machine code)
- (4) Compile and Load Program buttons
- (5) Start/Pause simulation buttons
- (6) Help/Exit buttons
- (7) Step selector

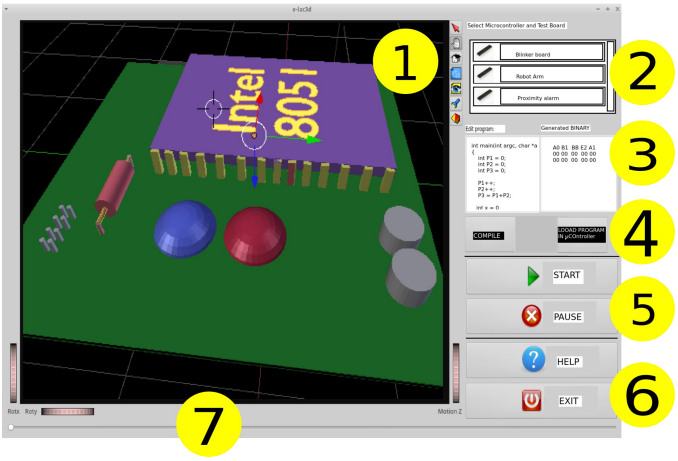


**Fig 6.** UI concept

*B. Applications execution diagram*

The simulation and animation of the microcontroller - experiment board ensemble is accomplished in a loop containing calls to certain methods of currently loaded microcontroller and electronic board objects in a specific order:

1. call to microcontroller's method: 'fetch'
2. call to microcontroller's method: 'decode'
3. call to microcontroller's method: 'execute'
4. call to board's method: 'update'

This cycle replicates the fetch-decode-execute cycle of the microcontroller and triggers a change in the electronic board graphics at the end of one loop cycle. (see Fig 7)

The conditions for entering and exiting the loop are the successful load of all plugins in the first case and the complete execution of the program in the microcontroller in the second case.

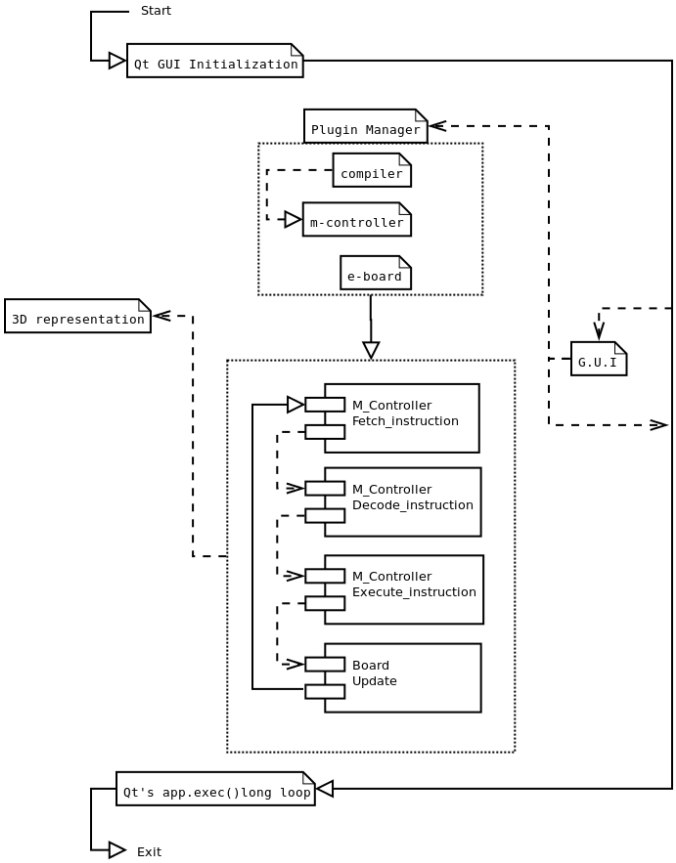


**Fig 7.** Fetch-decode-execute-update graphics cycle

## VI. Conclusion

All the necessary free tools and software libraries needed to develop a lightweight learning microcontroller 3d laboratory are well documented and easily accessible under Gnu/Linux. The mechanisms for implementing a modular plugin-style architecture proposed is simple and allows the programmers to develop small plugins to simulate microcontroller and EEC behavior and display the results on screen in a tridimensional form.

## References

[1] Traub, D. (1994). The promise of virtual reality for learning. In Loeffler, C. E. & Anderson, T. (eds), *The Virtual Reality Casebook*. Van Nostrand Reinhold. New York.

[2] Introduction to VRML 97 - David R. Nadeau, nadeau@sdsc.edu, http://www.sdsc.edu/~nadeau, San Diego Supercomputer Center

[3] Web source https://bitbucket.org/Coin3D/coin/wiki/IntroductionToCoin3D [1.02.2014]

[4] How To Write Shared Libraries – Ulrich Drepper, drepper@gmail.com, December 10, 2011

[5] A practical guide to the Models and Methods of Usage-centered Design – Larry L. Constantine, Lucy A.D. Lockwood, ACM Press New York, April