A Computational Model

- An software implementation of the solution to a (scientific) complex problem
- It usually requires a mathematical model or mathematical representation that has been formulated for the problem.
- The software implementation often requires high-performance computing (HPC) to run.

Abstraction and Decomposition

- Abstraction is recognized as a fundamental and essential principle in problem solving and software development.
- Abstraction and decomposition are extremely important in dealing with large and complex systems.
- Abstraction is the activity of hiding the details and exposing only the essential features of a particular system.
- In modeling, one of the critical tasks is representing the various aspects of a system at different levels of abstraction.
- A good abstraction captures the essential elements of a system, and purposely leaving out the rest.

Developing a Computational Model

Levels of Abstraction in the Process

The process of developing a computational model can be divided in three levels:
- Prototyping, using the Matlab programming language
- Performance implementation, using C or Fortran programming languages.
- Including methods, principles, and techniques for high performance computing (HPC).

Improved Performance

- For improved performance, an efficient implementation of the computational model is necessary.
- Two relevant programming languages that provide very efficient implementations are:
  - C
  - Fortran.
- These have been used for very efficient numerical and scientific programming.

IDE tools for Developing C and Fortran Programs

The selection of an appropriate IDE depends on: open source and multi-platform (Windows and Linux).

Popular tools/environments are:
- Eclipse with CDT
- Eclipse (Juno) - Photran
- CodeBlocks
- CodeLite
- Terminal window on Linux (Command window on MS Windows), a text line interface used with an appropriate editor (gedit on Linux)
Introducing The SCL Language

- The SCL language supports the conceptual framework of the scientific style of computation.
- SCL can facilitate the teaching, learning, and application of modeling and developing computational models of large and complex systems.
- The language syntax is defined to be at a higher level of abstraction than C and C++ (and Java) and it is an enhancement to the widely-used pseudo-code syntax used in program and algorithm design.
- It includes language statements for improving program readability, debugging, maintenance, and (not yet) correctness.

The SCL Language for Parallel Programming

- Parallel models are currently implemented in C/C++ and Fortran.
- Language features will facilitate the specification and modeling of family of applications on:
  - Directives-based parallel programming language (using OpenMP)
  - Message Passing (using MPI)

Directives-Based Parallel Programming

- OpenMP (most widely used)
- High Performance Fortran (HPF)
- Directives tell processor how to distribute data and work across the processors
- Directives appear as comments in the serial code
- Implemented on shared memory architectures

Message-Passing Parallel Programming

- Message Passing (Using MPI)
- Pass messages to send/receive data between processes
- Each process has its own local variables
- Can be used on either shared or distributed memory architectures
- Outgrowth of Parallel Virtual Machine (PVM) software and used in distributed computing

Additional Features of The SCL Language

- The SCL language has retained the semantics of C.
- A feature of practical importance of this language is that it includes some similarity to the Matlab language in two ways:
  - several expressions in the assignment statement are very similar to that in Matlab.
  - a number of the functions in its library resemble the ones in Matlab; they present the same or very similar interfaces.
- These features facilitate the transition from Matlab to SCL.
- The SCL language translator generates C code.
The SCL Translator

- The SCL language translator is implemented as a one-pass language processor that parses an SCL program and generates C source code.
- The translator flags general syntax errors.
- The generated code can be integrated conveniently with any C or C++ library.
- The basic run-time support of this language consists of the Gnu Scientific Library (GSL) and a set of additional C functions that are used with to facilitate the development of computational models.

Modules

- A program is usually partitioned into several modules. This is known as modular design.
- A function is the most basic decomposition unit in a C program and it carries out a specific task in a program.
- A module in C is a related collection of C functions. These functions will normally be stored in a single C source file.
- In object oriented programming languages, such as C++ and Java, the fundamental module is a class.
- Every module has a specific purpose and may be implemented in a specific programming language.

Fundamental Principles of Modular Design

- **Coupling** — a high-level description of the dependency of a module on other modules. Description of the interfaces. A module should exhibit loose coupling.
- **Cohesion** — the level of dependency within the module. A module should exhibit strong cohesion.
- **Separation of Concerns**. Each module should address a separate concern (one aspect of functionality).

Modular Design with Graphics

- A program that implements a computational model is partitioned into several modules at various levels of detail.
- At the highest level of abstraction, only the top modules are defined.
- This level is adequate when a program includes:
  - a GUI module for user interaction (front–end module)
  - the main logic or backend module (the computational model implementation)
  - the module for the data visualization of the output data produced by the main logic module.

High-level Modular Structure

- Graphics
  - Data Visualization
- Main Logic
  - Backend
- GUI I/O
  - Frontend

Software Tools for Implementing GUIs

- There are several good GUI toolkits or packages and most are implemented in C or C++.
- Several of these packages are based on the OPENGL standard. Some of the most widely-used toolkits for GUI implementation are:
  - GLUI – A very small and simple package that is implemented on top of GLUT/OPENGL.
  - FLTK – A slightly larger package that is also based on OPENGL.
  - Other similar packages are GLFW, GLOW, FOX, and GLV.
- More complete, larger, and more sophisticated packages are: WxWidgets (Wxwindows), GTK+, and Qt.
- Other packages are more geared toward game development, such as SDL.

Software Tools for Plotting

- There are several packages for data plotting and most are implemented in C or C++.
- Some of the most widely-used toolkits for plotting are:
  - GnuPlot – A very versatile and simple package that is widely known and well-documented.
  - Spreadsheet programs – A general-purpose business type of software packages.
  - PiPlot – A library of C functions with bindings for other programming languages.
  - DiLin – A library of C functions with bindings for other programming languages.
  - GluPlot – A C++ library and utility for data plotting
  - GlPlot – A C++ library and utility for data plotting
- Other packages are more geared toward plotting and data analysis, such as Ggobi.
Gnuplot is a portable command-line driven graphing utility for Linux, MS Windows, MacOS X, and other platforms.

The source code is copyrighted but freely distributed.

It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting.

It is also used as a plotting engine by third-party applications like Octave.

Gnuplot has been supported and under active development since 1986.

These examples store the problem data on a file *.gpl and the Gnuplot commands in a file *.cgp

- The free-falling object, the files are: ffallobj_ode.c and ffallobj_ode.sh For plotting: ffallobjode.gpl, ffallobjode.cgp, ffallobjode.png, ffallobjodev.gpl, and *.png.
- Falling object with a drag force, with a mathematical model given by two first-order ODEs. Files: ffallobj_drag.c and ffallobj_drag.sh. For plotting: ffallobj_drag.gpl, ffallobj_drag.cgp, *.png.
- The Lorenz equations program, the files are: elorenz.c, and elorenz.sh. For plotting: elorenz.gpl, elorenz.cgp, elorenz_zx.cgp, elorenz_png, elorenz_zx.png, elorenz_xyz.cgp, and elorenz_xyz.png.

Programs with Gnuplot

Free-Falling Object

- Applying Newton’s law of force that relates mass, acceleration and force, gives the following expression:
  
  \[-g = \frac{d^2x}{dt^2}\]

- the velocity \(v\) is given by \(\frac{dx}{dt}\)

- The resulting set of first-order ODEs is:
  
  \[
  \frac{dx}{dt} = 0x + v + 0 \\
  \frac{dv}{dt} = 0x + 0v - g
  \]

Vertical Position of Free-Falling Object

Vertical Velocity of Free-Falling Object

Falling Object with Drag Force

- There are two forces applied on the object: one due to gravity with acceleration \(g\) and the second is the drag force against the direction of movement.
- The drag force is \(cv^2\), in which \(c\) is the drag constant.
- Applying Newton’s law of force that relates mass, acceleration and force, gives the following expression:
  
  \[-mg + cv^2 = \frac{m^2x}{dt^2}\]

- The second-order differential equation is reduced to two first-order differential equations:
  
  \[
  v = \frac{dx}{dt} \\
  -mg + cv^2 = \frac{m^2v}{dt^2}
  \]

Vertical Displacement of Falling Object with Drag Force

Vertical Velocity of Falling Object with Drag Force
Edward N. Lorenz was a pioneer of Chaos Theory, he proposed the Lorenz Equations.

The mathematical model:

\[
\begin{align*}
\frac{dx}{dt} &= \sigma(y - x) \\
\frac{dy}{dt} &= x(\rho - z) - y \\
\frac{dz}{dt} &= xy - \beta z
\end{align*}
\]

Constants: \(\sigma = 10\), \(\beta = 8/3\), and \(\rho = 28\). Initial conditions: \(x(0) = 8\), \(y(0) = 8\), and \(z(0) = 27\) and a timespan from \(t = 0\) to \(t = 20\).

### Lorenz Variables

![Lorenz Variables Plot](image)

### Lorenz Variables with Time

![Lorenz Variables X, Y and Z versus Time](image)

### Example of a Generic C Program with a GUI

- A simplified example of complete C code of the main module and C++ of the GUI module is stored in files genmod.c and modgui.cpp.
- These two files are compiled separately with the C compiler and with the C++ compiler respectively.
- The linkage must be done with the C++ compiler.
- The commands for compiling and linking the two modules are:

  $ gcc -c -Wall genmod.c  
  $ g++ -c -Wall modgui.cpp -I /usr/include/GL  
  $ g++ genmod.o modgui.o -lglui -lglut -lm

### The GLUI Library for Implementing GUIs

- GLUI is a GLUT-based C++ user interface library which provides controls such as buttons, checkboxes, radio buttons, spinners, and listboxes to OpenGL applications.
- It is window-system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management.
- The GLUI User Interface Library provides standard user interface elements such as buttons and checkboxes.
- The GLUI library is written entirely over GLUT, and contains no system-dependent code.
- GLUI has been designed for programming simplicity, allowing user interface elements to be added with one line of code each.

### GUIs Using GLUI

![GUIs Using GLUI](image)

### Results

- Output Data
  - Throughput: 192
  - Rejected: 115
  - Avg wait: 0.0
  - CPU util: 0.825
  - Disk util: 0.65
The same general procedure used previously applies when implementing GUIs using the FLTK package.

In the following example, the main difference is the list of names of the functions to get the input values from the GUI module and the names of the functions to set the output values of the results.

The commands for compiling and linking the two modules using FLTK are:

```
$ gcc -c -Wall fgenmod.c
$ g++ -c -Wall fmodgui.cpp
$ g++ fgenmod.o fmodgui.o -lfltk -lm
```

The compressed archive `genmod.tar.gz` includes files:
- `genmod.c`, `fmodgui.cpp`, `genmod.mak`, `fgenmod.o`, `fmodgui.o`, and `fgenmod.c`.

The compressed archive, `ffallobj3g.tar.gz`, includes the following files:
- `ffallobj3g.c`, `compgui.cpp`, `ffallobj3g.mak`, `ffallobj3g.o`, and `ffallobj3g3g.c`.

These files make up the project of the Free-falling object computational model, which uses the GLUI package.

To extract the files from the archive `ffallobj3.tar.gz`, type the following command on an appropriate directory:

```
$ tar -zxvf ffallobj3g.tar.gz
```

The Cartoons library is based on FLTK graphical toolkit and is used to display various plots within real X and Y coordinates.

The main usage is visualization of scientific data but the library is extensible.

Automatic modes can generate tics with proper “density” so that when the drawing is rescaled, the number of tics can automatically change to keep drawings without overlapping labels.

Each drawing can have any number of vertical and/or horizontal axes with custom position relative to the plotting area (left, right, top or bottom).

The Lorenz Equations consists of a formulation of three first-order differential equations.

The program uses the GSL to solve numerically the ODEs.

The program consists of two modules:
- The main module: `elorenz3g.c`
- The module that plots the data: `cartlorenz.cpp`

The make file is: `elorenz3g.mak`