Simbody/SimTK Core 1.0 Release Plan

Last update: 11/14/2007

First Draft, Michael Sherman

1 Introduction

1.1 Purpose of this document

This document has two purposes:

(1) to set out the near-term schedule for SimTK Core 1.0 development, release, and training, and

(2) to define the specific functionality to be included in the 1.0 release.

Please note that Simbody 1.0 is included in SimTK Core 1.0.

1.2 Intended users of SimTK Core 1.0: "multibody biology" application programmers

SimTK Core 1.0 is a C++ Application Programming Interface (API), that is, a library for use by programmers working in C++. (Some of the lower level toolsets are accessible from C also.) It is intended to provide the specialized computational tools needed by programmers to develop or extend domain-specific applications in the broad area of "multibody biology." Multibody biology refers to the modeling of biological systems composed of identifiable, near-rigid parts moving with respect to one another via identifiable connections, such as joints or bonds. Skeletal mechanics and coarse-grained simulation of biopolymers and molecular machines are examples of such systems. The SimTK Core provides the computational framework in which such domain-specific models may be expressed and efficiently simulated.

We are using the word "application" here in a broad sense, not necessarily just the traditional GUI. Any program or higher-level API which is built to address problems in multibody biology may be considered an application – from a simple "console" main program, to an elaborate text file-driven tool set, to a full blown graphical interface.

1.3 Application areas for SimTK Core 1.0

Although much of SimTK Core is general-purpose, we hope to see users in two primary biosimulation fields, at two very different levels of maturity:

- neuromuscular biomechanical simulation using multibody dynamics, and
- internal coordinate and coarse-grained biomolecular simulation using multibody dynamics.

Use of multibody techniques is well-established in neuromuscular simulation, with many research groups having considerable experience. For those users, the SimTK Core provides open source, high performance capability which can replace existing methods which may require commercial components or be of limited capability. There are some new capabilities, such as custom mobilizers (biologically-inspired joints), but for this application area SimTK Core 1.0 is primarily a new way to access proven capabilities.

Coarse-grained molecular dynamics using multibody methods (MBMD), on the other hand, is a relatively new application area which has not been widely exploited, except in some narrow areas such as NMR structure refinement. Considerable research remains to be done on when and how to use multibody techniques to effectively simulate large molecules. There are many practical difficulties involved in getting a MBMD simulation to work, including building appropriate mechanical models, applying force fields, solvation, correct calculation of thermodynamic quantities, etc. Access to usable software is the primary obstacle to research at this point. For this application area the purpose of SimTK Core 1.0 is to allow researchers to begin exploration in this area, and to

contribute to the advancement of MBMD. Basic modeling tools are provided, serving as examples and "proof of concept" but as yet falling short in both functionality and performance of a complete, physically-accurate MBMD facility.

1.4 Logical structure of the SimTK Core

"SimTK Core" is the name we give to a collection of software tool libraries (APIs) designed for use by application programmers to build applications which make effective use of physics-based simulation of biological structures. These tools are logically organized in a six-layer hierarchy as shown in Figure 1 with higher-level tools dependent on lower ones but not vice versa. Depending on needs and level of sophistication, application programmers can access the tools at any level, ignoring those above.

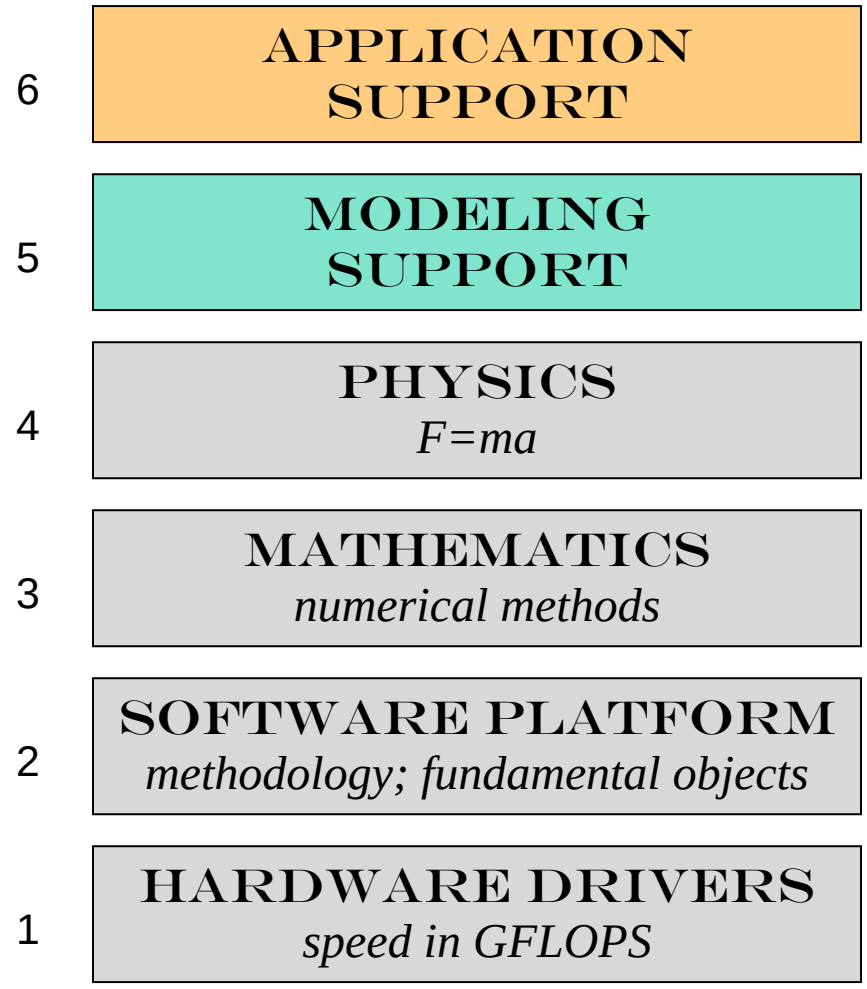

Figure 1: Logical structure of the SimTK Core software stack. Upper level tools can be dependent on everything below, but lower levels never depend on upper ones.

2 Schedule

now – 1/20/2008 (10 weeks less holidays)	general software development
January 21, 2008	Feature freeze; no further functionality development
1/21-3/2/2008 (6 weeks)	packaging, documentation, test, bug fixes, etc.
March 3, 2008	1.0 Release date: binaries available for download on SimTK.org (Windows, Mac, Linux)
3/3 – 3/19/2008 (2 weeks)	Final workshop development
March 20-21, 2008	Simbody/SimTK Core 1.0 workshop (for C++ API users and biosimulation course developers)
3/22-3/31/2008 (1 week)	Final prep for spring course
April 1, 2008	Spring quarter begins

3 Overview of visible features of SimTK Core 1.0

These are the features that are easily visible to the SimTK Core 1.0 API user. Internal, software engineering features of the release are discussed in Section 5.

3.1 Internal coordinate multibody dynamics

- systems of joint-connected rigid bodies in tree- or loop-structured topologies
- dynamic states are arbitrary generalized coordinates and speeds, e.g. torsion angles
- solved in Order(N) times (N is number of generalized coordinates)
- wide assortment of pre-built joints (mobilizers) and constraints
- custom joints (mobilizers) and constraints

3.2 Numerical methods

- continuous time integration with discrete event handling (stiff and non-stiff systems, constraints stabilized with coordinate projection)
- constrained and unconstrained optimization
- numerical differentiation
- numerically stable polynomial root finding (real & complex)

- random number generation (uniform & Gaussian distributions)

3.3 Matrix arithmetic and linear algebra

- zero-overhead facility for working with small, fixed-size, real and complex vectors and matrices
- automatically parallelized, hardware-specific high performance linear algebra for large vectors and matrices using LAPACK and BLAS 3.1.1 with ATLAS-generated kernels.
- object-oriented interface to LAPACK capabilities including linear equation solving, linear least squares, and eigenvalues
- const-correct C and C++ header for direct access to low-level LAPACK and BLAS routines

3.4 Simulation framework

- systems and subsystems
- state
- event handlers and event-driven reporting
- studies

3.5 Basic tools

- extensive support for 3d rotations and transforms, using rotation matrices, quaternions and other orientation representations
- calculation and representation of rigid body inertias
- high precision physical constants from NIST 2002, and mathematical constants like pi
- support for precision-independent programming

3.6 Application support

- basic mechanical force subsystems (springs & dampers, contact, gravity, etc.)
- basic visualization and animation of multibody systems (using VTK)
- molecular modeling (AMBER99, GBSA, construct internal coordinate coarse-grained protein and RNA models, read/write simple pdb file)
- solve for model coordinates that provide a weighted best-fit to an observed set of point locations (e.g., markers, atom positions in a pdb file)
- solve for model coordinates that minimize potential energy

3.7 Performance expectations for 1.0 (low!)

SimTK Core 1.0 has been architected so that high performance (that is, fast execution time) is achievable; however, 1.0 is a basic functionality release and almost no performance measurement or tuning has been done. The sole exception is the low-level linear algebra routines in LAPACK and BLAS.

You can expect extremely high performance from direct calls to LAPACK and BLAS, and modest performance from everything else. Later releases will substantially improve the performance of the higher-level toolsets such as Simmatrix and Simbody.

Performance of the molecular force field we are providing with the 1.0 release will be poor and should be limited to relatively small, "proof of concept" models rather than used for production work. We hope that our initial users will incorporate more sophisticated methods into their simulations, and that those methods can be migrated back into later releases of the SimTK Core.

3.8 Features missing *from 1.0*

These are features which are not in SimTK Core 1.0, but which some readers might expect would be there. These are planned for inclusion in a subsequent release as soon as possible. There are workarounds for most of these in 1.0.

- high-performance object-oriented interface to BLAS (e.g. matrix-matrix multiply). Currently the high level matrix objects do not exploit BLAS for low-level operations.
- built-in Monte Carlo methods
- explicit solvation for molecules
- automatic translation of sd/fast models to Simbody
- inverse dynamics and reaction loads
- scaling of generalized coordinates and constraint errors
- muscle models and controls
- weld joints
- state serialization
- parameter setting in State for Simbody (e.g., mass props & geometry)
- analytic computation of the dynamic Jacobian (energy Hessian)
- No general nonlinear root finding package (1.0 does have polynomial root finding and constrained optimization)
- No global optimizer. Only local optimizers are provided with SimTK Core 1.0. Global optimizers such as simulated annealing methods can be built on the existing components but none is provided directly.
- No bindings for languages other than C++
- calculation of operational space inertias
- No support for GPUs; no machine-specific handcoding of molecular mechanics inner loops

4 Structure of the release package

SimTK Core 1.0 will be available from the downloads page of the SimTKcore project at https://simtk.org/home/simtkcore.

4.1 Package contents

- SimTK Core is a C++ API, consisting of a set of header files, a set of platform-specific binary libraries, and documentation. Installation and downloads are centralized at SimTK.org project SimTKcore.
- Binary releases for 32 bit Windows, Mac, Linux as static and shared (DLL) libraries. Multiple versions of the SimTKatlas driver-level library are available for specific hardware configurations.
- Open source is available in the form of a set of unrestricted Subversion branches in the projects that make up the SimTK Core. No formal packaging, developer support, manuals, build-from-source-instructions, etc. will be provided in this release.
- Support is available through the SimTKcore project on SimTK.org: mailing lists, Wiki, bug reporting & feature request tracking. Telephone support is not offered although can probably be arranged on request if necessary.
- User's guides for Simbody and other Core modules.
- Doxygen-generated API reference material.

4.2 Supported platforms

4.2.1 Hardware-independent layer

32-bit Windows XP and Vista, Red Hat Linux, and Intel Mac.

4.2.2 Driver layer

TBD: need detail here

4.3 Physical structure of libraries and projects

Physically, the SimTK Core toolkit is organized as a set of libraries and corresponding header files. The libraries are generally confined to a single logical layer, but there may be several independent libraries in any given layer. Some of the libraries are third-party tools; that is, they are written and distributed by others, are unchanged by us, but a particular version and build has been made available on SimTK.org and qualified to work well with the SimTK Core. Others are original software produced by the SimTK Core team, or third party open source software that has been adapted by the SimTK Core team.

For the remainder of this document we will be talking only about projects that need to be done in layers 1-4. These layers currently consist of the libraries shown in the table below:

Library name	Level	SimTK.org project name	Description
`SimTKsimbody`	4	`simbody`	Internal coordinate rigid-body mechanics toolset for use in "multibody biology" applications from human gait simulation to coarse-grained molecular simulations.

SimTKmath	3	simmath	Numerical mathematics algorithms including optimization, numerical integration and discrete time stepping, root finding, linear and nonlinear algebra.
SimTKcpodes	3	cpodes	SimTK adaptation of the CPODES integrator developed in collaboration with Radu Serban at LLNL. This is derived from the DOE Sundials suite of numerical software tools.
SimTKcommon	2	simtkcommon	Classes supporting the basic SimTK architecture. Also includes the Simmatrix vector- and matrix-handling toolset, physical constants, random number generation and polynomial root finding.
SimTKlapack	2	lapack	Hardware-independent portion of the LAPACK/BLAS toolset for high-speed linear algebra.
SimTKatlas	1	lapack	Hardware-dependent portion ("driver") of the LAPACK/BLAS high performance linear algebra library.

5 Internal features of SimTK Core 1.0

This section discusses important software engineering aspects of the SimTK Core which are critical to its long term success but will be largely invisible to API users.

5.1 Licensing, registration

- Anyone can download documentation; download of binaries requires a (freely available) SimTK.org account so we can track usage and report to our sponsors.
- All software licensed under non-viral terms allowing unlimited modification and redistribution, for academic, commercial, nonprofit, or government purposes.

5.2 API stability, binary compatibility, and other promises

The 1.0 API is partitioned into pieces designated "Stable," "Candidate," and "Experimental" to indicate to the user what degree of stability can be expected from release to release of the SimTK Core. (There are also categories "Deprecated" and "Obsolete" which will see more use in later releases.)

A particular class may be Stable, while some of its methods are Candidate or Experimental. In other cases an entire class may be Candidate or Experimental in which case none of its methods can be Stable.

- Binary compatibility is guaranteed for the *Stable* API through "point" releases (1.1, 1.2 … until release 2.0) , subject to reasonable limitations. When possible stability will be maintained across major releases. Very little of the 1.0 release will provide a binary compatibility guarantee. An increasing fraction of the API will provide such a guarantee as our release numbers increase.

- The Candidate API represents code which we hope to move to the Stable API in the near future. However, we do not yet have enough confidence in it (generally due to a lack of user testing at time of release) to guarantee stability at release 1.0. We encourage use of the Candidate API and welcome user feedback as to whether it should be promoted, modified, or abandoned.
- The Experimental API will almost certainly change at the next point release. We encourage and appreciate brave users who try using these features and let us know their opinions. Some degree of change will be required in any user code which uses this part of the API.

5.2.1 Namespace conflicts

- No potentially conflicting symbols are introduced in the global namespace. All symbols are in the SimTK:: name space when possible, otherwise they begin with "SimTK_". Future releases will always follow this policy so users who refrain from defining symbols in SimTK:: or beginning with "SimTK_" they will not have conflicts in the future.

5.2.2 Thread safety

- SimTK Core code is thread safe in the following sense: any SimTK Core object allocated in a thread can be safely accessed from that thread regardless of the presence of other threads. This does *not* imply that multiple threads can access the *same* object! This guarantee means that the SimTK Core classes are written so that they either (a) do not depend on static or global information that could become inconsistent in the presence of multiple threads, or (b) synchronize such data properly so that they automatically behave correctly when there are multiple threads.
- *Sharing* objects across threads in SimTK Core 1.0 requires explicit coordination at a higher level; it is feasible but not yet supported directly by SimTK.
- In addition, some low-level SimTK Core code is *multithreaded*, meaning that it will use multiple threads when called if multiple CPUs are available. Currently the only multithreaded code supported is the low-level linear algebra BLAS routines at level 3 (that is, matrix-matrix multiplies, factoring, etc.). This code is itself thread safe, in the sense that if it is invoked from multiple threads, each thread will see correct behavior, although the overall performance may be suboptimal.

5.2.3 Static initialization

C++ itself does not make guarantees about the order in which static initializers in separate compilation units are executed, so that in general use of static initialized constants in static initializers may result in a reference being made prior to construction. However, such references to SimTK Core constants are always safe; users will never find one that is uninitialized.

- All SimTK defined constants are created in a manner which makes it safe for them to be employed in user code static initializers.

5.2.4 Enumeration values are fixed

All SimTK enumeration values are assigned specific integers rather than compiler-generated ones so that these values can be maintained unchanged in future releases even if more enumerations are added.

5.2.5 Adherence to SimTK Core coding standards

The source code for the SimTK Core is written in adherence to the SimTK Coding Guidelines document. This can be found on SimTK.org in the "resources" project, that is, https://simtk.org/home/resources. Follow the "Documents" link there.

All new code in the SimTK Core is written in C++. Code which is derived from earlier third party efforts may be in other languages. Currently C code is in use in various places and at level 1 (LAPACK) there is FORTRAN and even some assembler.

5.3 Build/test/release procedures

- Subproject builds are done through the automated CMake/CTest build system and posted via DART to SimTK.org.
- For this release, the full SimTK Core package is assembled from the subprojects with some manual intervention.

6 Things-to-do list for 1.0 release

This is the list of tasks that must be performed by the March 3 release date. It is divided into "functionality," which must be complete by January 21, and "packaging" which can be done in the Jan 21-Mar 3 period.

TBD: this list is not complete: what else?

6.1 Functionality completion tasks

Build system

- (Peter) All SimTK Core projects should build reliably on Mac, Windows, Linux and post (green) dashboards.
- (Peter & Jack) Separate "driver" layer install from the rest of the core. People should be able to install just the driver to pick up LAPACK & BLAS, and then OpenMM and other hardware-specific, GPU-exploiting goodies.
- (Jack) Make sure that project libraries and build procedures are workable for putting together the overall Core installation downloads.

Simbody

- (Sherm) Implement generalized constraint module
- (Sherm) Implement custom mobilizer interface
- (Sherm) Weld joints if possible
- (Sherm) Add state-variable-using force example (i.e., a "z" state variable)

Molecule modeler

- (Chris) Nucleotides for RNA (and an extra one for DNA?)
- (Chris) End caps for peptides.
- (Chris) Add biotypes for protein and RNA.
- (Chris) Debug internal coordinate partitioning for protein & RNA.
- (Mark) Re-engineer GBSA interface to make it separable from Gromacs.
- (Mark & Chris) Add GBSA to existing Simbody force field.
- (Chris) Add some partitioning strategies. Most important: selective rigidizing. Also, rigid vs. torsion omega; protonation of histidine; etc. mostly just to test out the interface and provide examples.
- (Chris) Allow some way to override charges so we can neutralize RNAs for demo purposes.
- (Chris & Peter) Output trajectories as concatenated PDB files.
- (Chris & Peter) Read sequence and point locations from PDB; create internal coordinate model and best-fit to point locations
- (Peter) Implement steric-clash removing solver.
- (Peter) Implement some form of temperature maintenance; no need for statistical purity. Need to be able to use this in a simulated annealing loop as a global minimizer.

Simmath

- (Jack) FactorLU, FactorQTZ, Eigenvalue classes
- (Jack) Optimizer cleanup.
- (Peter) Cpodes event handling/restart changes; bug fixes

Simulation Framework (SimTKcommon)

- (Peter) Randomize the Random() class unless explicitly seeded.
- (All) Switch to using std:: classes, or extensions of std:: classes for basic containers used in the API
- (All) Ensure that PIMPL design pattern is in use wherever appropriate
- (All) Ensure enums all have specific integer values for future compatibility.
- (?) Refactor the scalar, small matrix, rotation and other templatized, exposed-implementation classes to permit distribution of just those capabilities as standalone headers.

6.2 Packaging tasks

6.2.1 Documentation

- (All) Add doxygen comments wherever useful.
- (Jack?) Figure out how to package and release doxygen-generated HTML.
- (All) Annotate the API to indicate the level of stability guarantee for each class and/or method.

6.2.2 Examples

- (Chris & Peter) Example/tutorial showing how to model and simulate a molecule.

6.2.3 Testing

6.2.4 Release packages

6.3 Course development

TBD: develop the workshop