MULTIBODY DYNAMICS 2007, ECCOMAS Thematic Conference
C.L. Bottasso, P. Masarati, L. Trainelli (eds.)
Milano, Italy, 25–28 June 2007

ARCHITECTURE OF THE CHRONO::ENGINE PHYSICS SIMULATION MIDDLEWARE

Alessandro Tasora[⋆]**, Marco Silvestri**[⋆]**, and Paolo Righettini**[†]

[⋆]Dipartimento di Ingegneria Industriale
Universitá degli Studi di Parma, Parco Area delle Scienze 181/A, 43100 Parma, Italy
e-mails: `tasora@ied.unipr.it`, `silvestri@ied.unipr.it` web page:
`http://ied.unipr.it/~tasora`

[†]Dipartimento di Ingegneria Elettrotecnica
Politecnico di Milano, Piazza Leonardo da Vinci, 20100 Milano, Italy
e-mail: `righettini@mech.polimi.it`

Keywords: Multibody, Software architecture, middleware, simulation.

Abstract. *This article outlines design and implementation issues of the Chrono::Engine software, a library of C++ objects and functions which can be used by third-party developers to implement applications featuring complex physical simulations.*

Programmers can exploit the functionalities of Chrono::Engine to address typical problems encountered in the multibody field, such as dynamical, kinematic and static simulations.

Special care has been paid in developing a robust, modular, portable, expandable and fast software architecture. An application based on this library, featuring graphical user interface and interactive 3D modelling, has been already implemented, demonstrating the reliability and the efficiency of the Chrono::Engine middleware.

1 INTRODUCTION

Developing software for general-purpose multibody simulation is a challenging task, involving interdisciplinary knowledge borrowed from fields like theoretical mechanics, physics, numerical analysis, computer science, graph theory and computational geometry, to name a few.

Software houses interested in embedding physical features in their applications may prefer to overcome the difficulties of implementing their own multibody simulation methods, by relying on ready-to-use libraries. These libraries, possibly along with asset management tools, are hence called *middleware*.

Recently, software houses like Ageia, Havok and Pixelux made massive investments and advancements in the field of middleware for multibody simulation because of the interest coming from the multi-million videogame industry [1].

This approach has been extensively used in the CAD field, where complex computational geometry issues are resolved by robust libraries like Parasolid^{TM} or ACIS^{TM}, but only recently took momentum in the simulation field.

As most middleware, the Chrono::Engine project is based on a application program interface (API) for developing in C++ language, that is C++ header files and static and/or dynamic libraries (.dll) containing the pre-compiled code. In order to allow third-party developers to take advantage of Chrono::Engine simulation features, special attention has been paid in providing a set of robust, tested, coherent and well documented functions. Given the complexity of the project, approaching one thousand of source files, the software is organized in *classes* as recommended by the Object Oriented Programming paradigm, targeting modularity, encapsulation, reusability and polymorphism [2].

2 ARCHITECTURE

In Fig. 1 one can see how the Chrono::Engine library represents a software layer between OS-level functionalities and end-user applications. For example, the picture shows that custom applications can rely on this API, for instance an embedded software for flight simulation, a virtual reality program, a CAD, and so on. The library itself is based on a Javascript parser, to allow scripting and easy access to inner functions also by means of scripting.

Thank to compliance to pure ANSI C++ and an OS-abstraction layer, the library is platform independent and is available on different operating systems: this requires a a documented support of various build systems (Jam, Nmake, Cmake) and a complex mainenance of IDE-based configurations.

The optional *Chrono Plug-in* module is an application which provides a full-featured GUI (Graphical User Interface) for the Chrono::Engine software, on which it is based upon. Given the complexity of keeping up-to date an application with 3D viewing and animation capabilities, *Chrono Plug-in* relies on a 3-rd party modeler [1] for visualization and interaction with the user. Thank to the ROOPS[2] layer, also the *Chrono Plug-in* is platform-independent.

Another optional module, *Revolution4D Plug-in*, has been developed in sake of a method to import 3D models from third-party CAD software. This module uses the open-source Open-CASCADE library to parse and convert complex models from STEP, IGES or other formats.

The modular approach, based on a middleware surrounded by optional modules with user

[1]Realsoft3D, from Realsoft OY. This software provides modeling with CSG or B-rep geometrical representation, a material system, ray-tracing shading for quality rendering and other typical features often available in 3D modeling applications.

[2]Realsoft Object Oriented Programming System

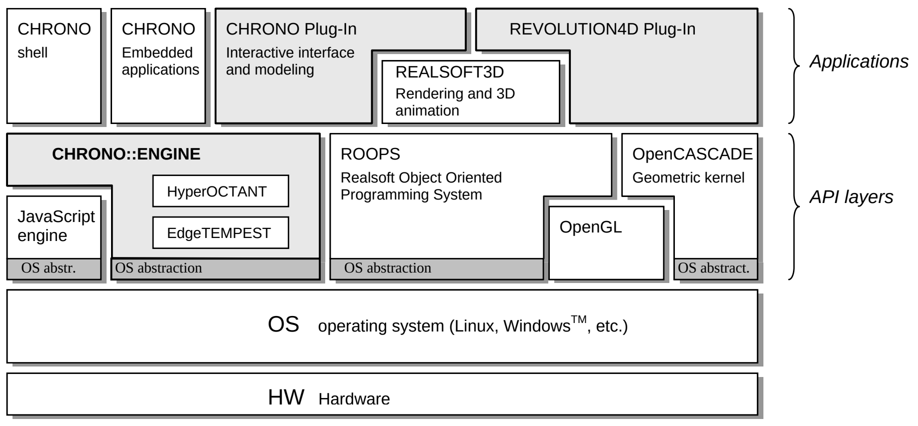


Figure 1: Dependency of the Chrono::Engine middleware.

interfaces, allowed us to concentrate efforts on physical issues without worrying too much about GUI and OS details.

Moreover, the Chrono::Engine library itself is divided in further components. For instance there is a module for basic linear algebra, a module for numerical methods, and so on, each module belonging to a separate C++ namespace. Among these, the most noticeable components are HyperOCTANT and EdgeTEMPEST.

2.1 The HyperOCTANT library

To address the dynamical analysis of large systems we developed the custom *HyperOCTANT* solver, which can handle the critical cases of friction, collision and stacking. This component embeds either a direct simplex solver and an iterative fixed-point method.

Since algorithmic robustness is an imperative requirement for real-time applications, this solver can also handle the cases of redundant or ill-posed constraints.

The *HyperOCTANT* library relies on a recent numerical method which avoids the temporary allocation of matrices [9].

2.2 The EdgeTEMPEST library

Sometime it is necessary to simulate contacts and collisions between sliding or rolling shapes, as in the case of conveyor belts, packaging machines and so on. If so, collision detection features are needed. To this end, the *EdgeTEMPEST* component has been developed.

The *EdgeTEMPEST* library embeds a three stage collision detection pipeline. At each simulation step, an optimized *broad phase* algorithm can detect the potential colliding pairs using the sweep-and-prune approach. This avoids a combinatory complexity as in brute force approaches, and allows scenarios with thousands of colliding bodies.

After the narrow-phase, a further refinement is done by a stackless binary-traversal method, which explores the AABB trees of the two potential colliding meshes in case these are made of many triangles or compound shapes.

Finally, the *narrow phase* stage can detect the precise position for the colliding points.

The *EdgeTEMPEST* module features a persistent contact manifold and robust fallbacks for degenerate cases: in this way it is possible to simulate singular situations like the case of a cylinder over a flat surface, and so on.

The case of contact between freeform surfaces is handled without the need of polygonal

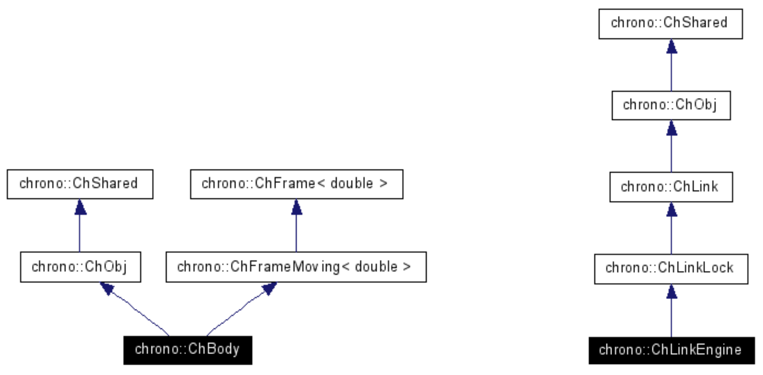


Figure 2: Example of class inheritance diagrams for rigid body and link classes.

approximations [8], although at a cost of high computational requirements.

3 IMPLEMENTATION

The Chrono::Engine middleware is written in C++ language: all items are organized in classes and namespaces, according to the OOP guidelines. Classes and objects have been tested and profiled for fast execution, in order to achieve real-time performance even for complex scenarios.

Modern programming techniques have been adopted, like metaprogramming, class templating, shared pointers, class factories, memory leak trackers, cross platform and cross compiler archiving, persistent-transient data mapping [5]. The C++ feature of operator overloading have been used to provide a compact algebra to manage quaternions, static and moving coordinate systems.

3.1 Class design

Using software engineering design techniques, such as UML and OMT, a broad set of classes has been outlined and implemented.

A full documentation of all classes and their members is reported in [10], along with examples. The documentation of the API is completely automated by means of the *Doxygen* tool, which extracts class member description from formatted comments in the source code.

Some classes, as those depicted in Fig. 2, belong to deep inheritance trees. A noticeable example is the `ChBody` class: rigid body objects, featuring attributes such as mass and inertia, also inherit features such as speed and acceleration from the parent class `ChMovingFrame` (while the more general `ChFrame` class just defines position and alignment). Thank to multiple inheritance, the `ChBody` also owns the features of `ChObj` and `ChShared` classes, for instance the name attribute or the capability of being serialized from transient to persistent data.

A complex class hierarchy is related to mechanical constraints, since Chrono::Engine features dozens of different joints: revolute joints, spherical bearings, prismatic guides, gears, screws, imposed trajectories, surface contact, glyphs and so on.

Aiming at the fast simulation of systems with thousands of bodies, crucial issues have been addressed. In detail, most algorithm had to be reduced to the O(n) linear-time linear-space complexity order, by adopting sparse matrices [7] or innovative iterative methods [9]. Also, Chrono::Engine can handle non-smooth simulations as those arising from colliding and contacting bodies, using the differential inclusion approach of [6].

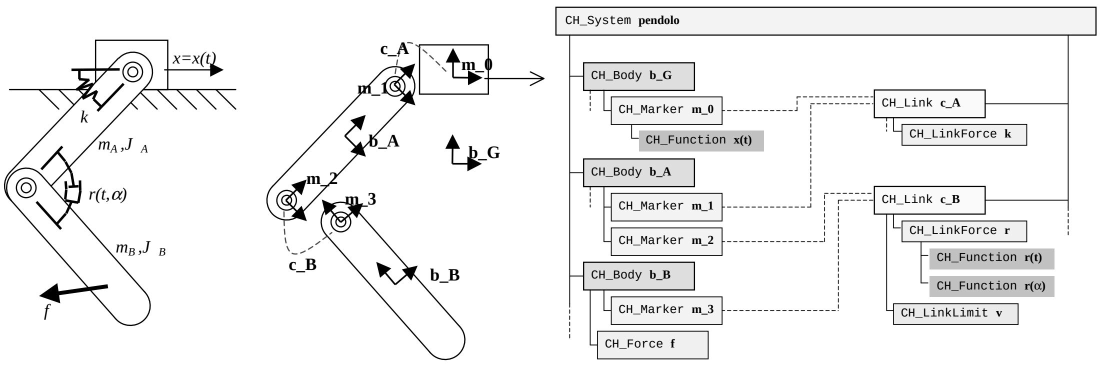


Figure 3: Object diagram showing pointer relationships between physical items.

Component		**Source files**	**Lines of code**	**Classes**
Chrono::Engine	EdgeTempest	206	37’029	121
	HyperOctant	37	5’305	19
	Physics	103	35’418	53
	Core functions	39	14’462	26
	Other	106	33’274	46
Chrono plugin		300	89’171	180
Revolution4D		119	23’063	65
TOTAL		**910**	**237’722**	**510**

Table 1: Software metrics, at 26-2-2007.

Physical systems are described in terms of rigid bodies connected by constraints, in unlimited number. This means that a transient database of physical items must be kept updated using run-time allocated containers: as shown in Fig.3, the physical system object owns either a list of rigid bodies and a list of constraints. Meanwhile, also rigid bodies have their own containers: these can be used to add forces and auxiliary references to rigid bodies.

Thank to polimorphism and abstraction, a large set of specialized constraint types can be either used or modified by inheritance, without requiring any modification to the `CHsystem` class or to the solver. These constraints (joints, screws, clutches, brakes, springs, etc.) are organized in a large class hierarchy. Specific constraints have been added to allow the simulation of drives and linear motors, with special attention to mechatronic and robotic applications.

Items can be added or removed from containers even while the simulator is running: this may be a requirement for some kind of real-time user interaction, such as in VR applications.

In Tab.1 there is a summary showing the SLOC (Source Lines Of Code) for all the C++ components of the Chrono::Engine project, along with the number of classes and source files. These software metrics show that only a fraction of the coding efforts is about about physical issues while, on the other hand, components for collision detection require a complex and meticulous design.

4 EXAMPLES

The Chrono::Engine library has been extensively used in many engineering projects. In this paragraph we present some basic examples.

5 A basic test: a pendulum

Following is a basic example on how to create and simulate a pendulum inside a C++ program.

```
ChSystem my_system;                        // create the physical system

ChSharedBodyPtr  my_body_A(new ChBody);    // create the truss
ChSharedBodyPtr  my_body_B(new ChBody);    // create the pendulum

my_body_A->SetBodyFixed(true);             // truss does not move
my_body_B->SetPos(ChVector<>(0,-2,0));     // ex. set initial position for pendulum

                                           // create a revolute joint
ChSharedPtr<ChLinkLockRevolute>  my_link_AB(new ChLinkLockRevolute);
my_link_AB->Initialize(  my_body_A,
                         my_body_B,
                         ChCoordsys<>(ChVector<>(1,2,0)));

my_system.AddBody(my_body_A);              // add the truss to the system
my_system.AddBody(my_body_B);              // add the pendulum to the system
my_system.AddLink(my_link_AB);             // add the joint to the system

while(my_system.GetTime()<10)              // 10s of default dynamic simulation
{
     my_system.StepDynamics( 0.01);        // advance the simulation

     // ..do something, ex. plot results, etc.
}
```

In the example above, a remarkable issue is the fact that dynamically allocated instances, such as rigid bodies and constraints, are managed by shared pointers [3]. This means that the programmer does not need to care about the deletion of instances, because deallocation happens automatically. This has positive side effects on the ease of coding and minimizes the risk of memory leaks.

The simulation is advanced in the `while(){...}` loop, thank to the `StepDynamics()` function. In this loop, one could perform additional tasks such as update the position of 3D shapes on the screen, plot data, acquire interactive user input for man-in-the-loop simulations, share data with acquisition devices for real-time applications, and so on. Using specific settings, the `StepDynamics()` function may perform multiple sub-steps, if integration with steps of variable size is required for stiff systems.

6 The Granit robot

The Granit robot is a parallel kinematics manipulator which features very high precision and stiffness [11]. This design has been tested, developed and optimized using the Chrono plug-in. By means of multibody simulations, repeated over a set of benchmark trajectories, the optimal compromise between motor, reducer and arm dimensions has been obtained.

Fig.4 shows a frame from one of the many simulations, along with a plot of the torque requested by one of the four motors in order to move the end-effector along the benchmark trajectory.

Thank to the optimal design, this robot is able of accelerations in excess of $40m/s^2$ without sacrificing precision and repeatability, both about $0.01mm$. The robot is currently being succesfully used in an industrial environment.

[3] In Chrono::Engine there are classes for handling pointers as a special type of *smart pointers*, either of intrusive or non-intrusive type.

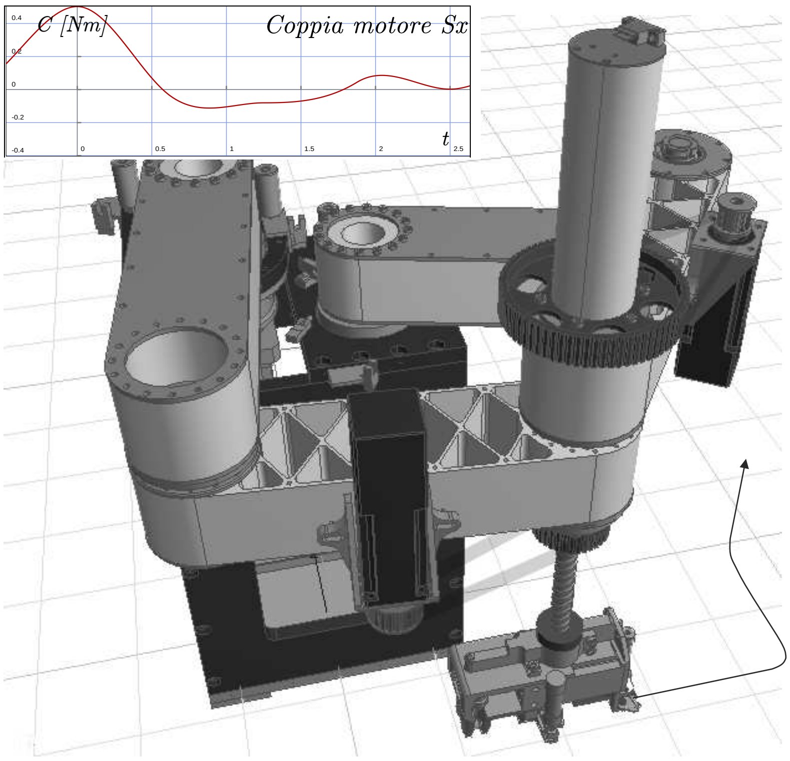
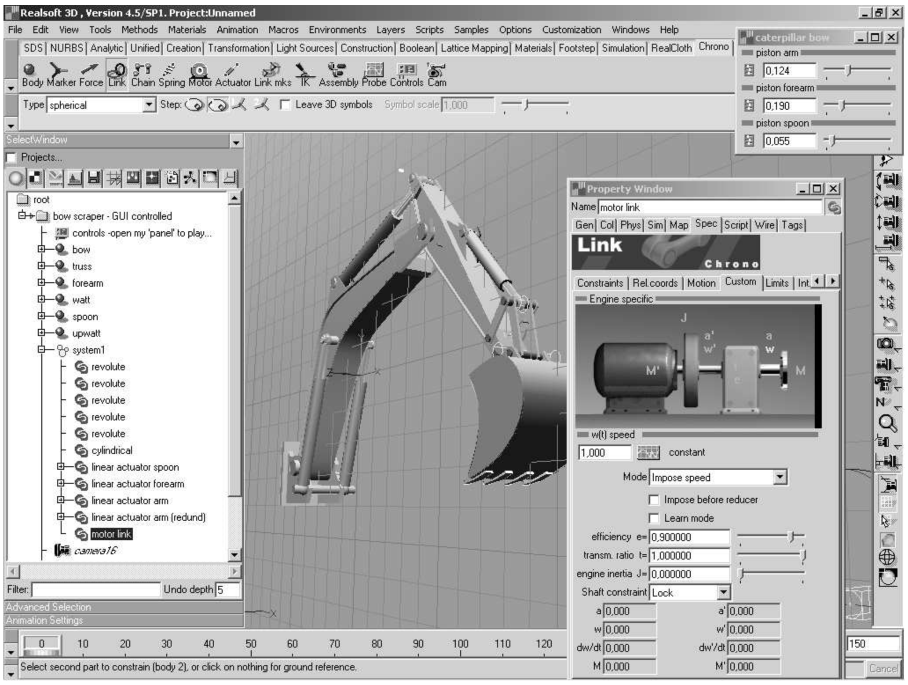


Figure 4: Simulation of the GRANIT robot.


Realsoft 3D , Version 4.5/SP1. Project:Unnamed
File Edit View Tools Methods Materials Animation Macros Environments Layers Scripts Samples Options Customization Windows Help
SDS NURBS Analytic Unified Creation Transformation Light Sources Construction Boolean Lattice Mapping Materials Footstep Simulation RealCloth Chrono
Body Marker Force Link Chain Spring Motor Actuator Link mks IK Assembly Probe Controls Cam
SelectWindow
Projects...
Advanced Selection
Animation Settings
0 10 20 30 40 50 60 70 80 90 100 110 120
Property Window
Gen Col Phys Sim Map Spec Script Wire Tags
Link
Chrono
Constraints Rel.coords Motion Custom Limits Int.
Engine specific
J
a'
w'
a
w
M'
M
Cancel

Type: spherical · Step · ☐ Leave 3D symbols · Symbol scale 1,000

- root
  - bow scraper - GUI controlled
    - controls -open my 'panel' to play...
    - bow
    - truss
    - forearm
    - watt
    - spoon
    - upwatt
    - system1
      - revolute
      - revolute
      - revolute
      - revolute
      - revolute
      - cylindrical
      - linear actuator spoon
      - linear actuator forearm
      - linear actuator arm
      - linear actuator arm (redund)
      - motor link
    - camera16

Filter: · Undo depth 5

Select second part to constrain (body 2), or click on nothing for ground reference.

caterpillar bow
piston arm 0,124
piston forearm 0,190
piston spoon 0,055

Name motor link

w(t) speed 1,000 constant

Mode Impose speed

☐ Impose before reducer

☐ Learn mode

efficiency e= 0,900000

transm. ratio t= 1,000000

engine inertia J= 0,000000

Shaft constraint Lock

a	0,000	a'	0,000
w	0,000	w'	0,000
dw/dt	0,000	dw'/dt	0,000
M	0,000	M'	0,000

150

Figure 5: User interface based on the Chrono::Engine API.

7 CONCLUSIONS

Programmers can take advantage of the algorithm contained in the C++ application programming interface of the Chrono::Engine in order to build applications featuring realistic simulation of contacts, collisions, constraints, motors, mechanical devices and so on.

Based on Chrono::Engine middleware, also a full-featured application with user interface has been developed, the *CHRONO plug-in* simulator, whose interface is depicted in Fig. 5.

8 ACKNOWLEDGEMENTS

Thanks to the staff at Realsoft OY, Finland, for supporting the plug-in implementation into the Realsoft3D modeler.

REFERENCES

[1] R.Tonge, L.Zhang, D.Sequeira, AGEIA Technologies Inc. Method and program solving LCPs for rigid body dynamics. *US patent N.7079145 B2*, 2006.

[2] B.Stroustrup. The C++ programming language. Addison-Wesley Longman Publishing Co., USA, 1986.

[3] W.H.Press, S.A.Teukolsky, W.T.Vetterling, B.P.Flannery. Numerical recipes in C++. Cambridge University Press, 2003.

[4] D.R.Musser, A.Saini. The STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library. Addison-Wesley Longman Publishing Co., USA, 1995.

[5] A.Alexandrescu. Modern C++ design: generic programming and design patterns applied. Addison-Wesley Professional, 2001.

[6] M.Anitescu, G.D.Hart. A constraint-stabilized time-stepping approach for rigid multibody dynamics with joints, contact and friction. *International Journal for Numerical Methods in Engineering*, 60(14), 2335-2371, 2004

[7] A.Tasora. An optimized lagrangian-multiplier approach for interactive multibody simulation in kinematic and dynamical digital prototyping. *Proceedings of VIII ISCSB*, F. Casolo (ed.), CLUP, Milano, 312, 2001, .

[8] A.Tasora, P.Righettini. Sliding contact between freeform surfaces. *Multibody System Dynamics* 10: 239–262, October 2003, Volume 10, Issue 3. ISSN 1384-5640, ed.Kluwer Academic Publishers

[9] A.Tasora. An iterative fixed-point method for solving large complementarity problems in multibody systems. *Proceedings of GIMC 2006, XVI Congress of the Italian Group of Computational Mechanics*, Bologna, 26-28 June 2006

[10] A.Tasora. *Chrono::Engine web site:* www.deltaknowledge.com/chronoengine, 2006

[11] A.Tasora, P.Righettini, S.Chatterton. Design of the GRANIT parallel kinematic manipulator. *Proceedings of RAAD05*, Bucharest May 26-28, 2005.