An Object-Oriented Architecture for the Simulation of Rigid-Body Kinematics

Geoff George and Brian K. Guenter

An Experience Paper

Contact information:

Geoff George
662 Fern Street
Marietta, GA 30067

(404) 612-8212
geoff@cc.gatech.edu

Brian K. Guenter
Georgia Tech College of Computing
801 Atlantic Drive
Atlanta, GA 30332-0280

(404) 853-9387
brian.guenter@cc.gatech.edu

Abstract

Kinematic simulation involves finding simultaneous zeros for a set of functions representing mechanical linkages which constrain the positions and orientations of a set of bodies. Evaluation of these functions, and of their Jacobians with respect to the positions and orientations of the bodies, is sufficient to allow the use of a multidimensional Newton-Raphson procedure to solve the kinematic constraint equations. An abstraction comprising such a function and its Jacobian may be implemented as a virtual class in an object-oriented language, yielding a software architecture embodying benefits of object-oriented design and implementation: abstraction, encapsulation, extensibility and reusability. An implementation in Eiffel is described.

Introduction

Kinematics is the study of time-independent qualities of motion, i.e. the study of motion without regard to velocities, forces or accelerations (Haug (1989)). Kinematics is the basis of the study of dynamics, which has become central to most computer animation. This paper describes an object-oriented architecture for the simulation of the kinematics of mechanical systems.

How kinematic simulation works

From the viewpoint of kinematics, a mechanical system is a set of rigid bodies whose motions are constrained by the linkages among them. The state of the system is the concatenation of the states of all the bodies; the state of a body is its position and orientation. In two dimensions the state of a body is represented by three quantities, two (x and y) for its position and one (θ) for its orientation. The state of the system of n bodies is therefore given by 3n quantities.

Each linkage in the system is represented by a set of constraint functions, typically one or two, each of which maps the system state to a scalar value. A collection of such constraint functions, considered as a vector, is denoted by a boldface capital phi, Φ. These functions need have no meaning other than that they are simultaneously zero iff the constraints are satisfied, i.e. iff the bodies whose motions are constrained by a linkage are in positions and orientations allowed by that linkage, although usually the functions do have some geometric significance. Each kind of linkage constrains some fixed number of degrees of freedom of the system; the number of unconstrained degrees of freedom of the system is the number of free variables, i.e. state components, whose values may be independently specified.

For example, in two dimensions a revolute joint constrains a point on one body to coincide with some point on another body. Mechanically, this corresponds to a pin through both bodies, so that they are

constrained to rotate about a common axis having no fixed position. Given this constraint, the six state variables of the two bodies may no longer each be specified independently; once any four of these variables have been specified, the values of the other two are determined by the constraint, assuming the constraint can be satisfied. This constrains two degrees of freedom, and is represented by two scalar constraint equations. If the two points are expressed in global coordinates as (x_1, y_1) and (x_2, y_2), these equations are:

$$x_1 - x_2 = 0 \quad \text{and}$$
$$y_1 - y_2 = 0.$$

If the global coordinates of these points are given by the vectors $\mathbf{r}_1$ and $\mathbf{r}_2$, these constraint equations may be written as the single vector equation:

$$\Phi = \mathbf{r}_1 - \mathbf{r}_2 = \mathbf{0}.$$

If these points, each in its own body-local coordinate system, are given by the vectors $\mathbf{p}_1$ and $\mathbf{p}_2$, and the bodies' local-to-global rotation matrices are R_1 and R_2, and their origins in global coordinates are $\mathbf{O}_1$ and $\mathbf{O}_2$, these equations may also be written as:

$$\Phi = \mathbf{O}_1 + R_1\,\mathbf{p}_1 - \mathbf{O}_2 + R_2\,\mathbf{p}_2 = \mathbf{0}.$$

The Eiffel code which implements this Φ function is:

```
phi: VECTOR is
   -- A revolute joint's phi vector.
   -- Contains one element for each constraint equation of the joint.
do
```

```
        Result := b1.local_to_global (fp1) - b2.local_to_global (fp2)
    end; -- phi
```

where b1 and b2 are the bodies constrained by this linkage, and fp1 and fp2 are the fixed points on those bodies, in local coordinates, which are constrained to coincide in global coordinates.

Given a system of n degrees of freedom, the requirement that all the constraint functions be zero and that all the free variables have specified values (driver constraints), may be considered a system of n equations in n unknowns (the system state variables):

$$\Phi (\mathbf{q}) = \mathbf{0}$$

where Φ is a vector which is the aggregate of all the constraint functions, including the driver constraints, and **q** is the system state vector. If all the constraint functions are continuously differentiable, a Newton-Raphson procedure can be used to find values for all non-free variables which make the constraint functions zero. This set of values is a system state which satisfies all the constraints, i.e. in which all bodies are in positions and orientations given by the free variables' values and allowed by the linkages.

An object-oriented architecture

For the purposes of kinematic simulation, any given mechanical linkage is sufficiently characterized by its constraint equations. Routines which evaluate the corresponding constraint functions and their Jacobian are sufficient to allow the constraint equations to be solved using a multidimensional Newton-Raphson procedure. The principle of information hiding through abstraction may be satisfied by defining a class (named e.g. JOINT) which exports two deferred (virtual) routines, intended to evaluate the constraint functions, and their Jacobians, which define some mechanical linkage. A particular kind of mechanical

linkage, e.g. a revolute joint, is implemented by defining a descendant class of JOINT with routines to evaluate that linkage's constraint function and its Jacobian.

Given a list of objects whose types are descendants of the class JOINT, representing a set of mechanical linkages, it is straightforward to implement the aggregate constraint function Φ for the entire mechanical system comprising those linkages, and to solve the corresponding constraint equations, with no further knowledge of the actual types of those objects.

Implementation in Eiffel

Three classes are central to the kinematic simulation software; they are MECHANISM, BODY and JOINT.

A MECHANISM is simply a list of BODYs and a list of JOINTs. It implements bookkeeping routines for constructing the matrices and vectors to which the Newton-Raphson procedure is applied, from values of the constraint functions and their Jacobian returned by the JOINTs it contains. Its central routine, update_state, is one which does this construction and then calls the Newton-Raphson procedure. A MECHANISM is initialized by adding to it some number of BODYs, and sufficient JOINTs (including drivers) to constrain the corresponding number of degrees of freedom.

A BODY is simply a placeholder for the state vector of a particular body, i.e. for its position and orientation. This class also implements utility routines, e.g. for transforming positions and vectors between global and body-local coordinate systems. Any graphical entity may be attached to a BODY; a MECHANISM is draws itself by drawing each of its component bodies' graphics in that body's current position and orientation. A BODY is initialized by specifying an initial state (position and orientation) and a graphical entity to be used to draw the BODY.

The class JOINT is a deferred (virtual) class which declares routines which calculate the constraint functions and their derivatives. A mechanical linkage is represented by some descendant of JOINT, which implements the actual constraint functions corresponding to that kind of linkage, and their Jacobian. The partial derivatives may be calculated numerically or analytically. The latter is preferred as being computationally less expensive and more numerically stable, although the analytical derivation is a laborious and tedious, if conceptually simple, process. A JOINT is initialized by specifying the particular BODY or BODYs whose states it constrains, plus other information specific to kind of linkage being created.

```
deferred class JOINT
export
   num_constraints, num_bodies, bodies, phi, phi_q
inherit
   MECHANISM_CONSTANTS
feature
   num_constraints: INTEGER is deferred end;
         -- Number of degrees of freedom constrained.
   bodies: ARRAY [BODY];
         -- Bodies this joint constrains, 1 .. `num_bodies'.

num_bodies: INTEGER is
   -- Number of bodies in `bodies'.
do
Result := bodies.upper;
end; -- num_bodies
```

```
phi: VECTOR is
   -- This joint's phi vector.
   -- Contains one element for each constraint equation of the joint.
deferred
end; -- phi

phi_q: MATRIX is
   -- This joint's Jacobian matrix.
   -- Contains one row for each constraint equation, and one column for
   -- each component of each constrained body.
deferred
end -- phi_q

end -- class JOINT
```

Examples of descendants of JOINT are the classes REVOLUTE, ABSOLUTE, SPHERICAL, TRANSLATIONAL and GEAR, which evaluate the constraint functions for the suggested linkages. The class DRIVER, another descendant of JOINT, is crucial; it allows a value to be specified at any time, for any one state component of one BODY. A DRIVER corresponds to a free variable, i.e. a state component whose value is not uniquely determined by the mechanical linkages, but is specified independently. The current values of all the DRIVERs determine the values of the non-free state components.

Given this machinery, simulations of fairly complex mechanical systems have proven very easy to implement, and can be implemented in a highly stylized manner. The general outline of every program that

has been written using this system so far, is:

```
Create and initialize the BODYs
Create and initialize the JOINTs
Create and initialize the MECHANISM
Initialize the graphics display
loop
        Specify current values for all DRIVERs, i.e. free variables
        Call the Newton-Raphson procedure, to determine values for
                non-free state variables
        Call the draw routine to display the current state of the mechanism
end loop
```

This is typically a hundred lines of straight-line initialization code, and a dozen lines of main loop.

Implementing a new kind of linkage with a new descendant of JOINT has also proved to be straightforward, and highly stylized; it normally involves very little more than plugging expressions for the constraint functions and their analytical partial derivatives into a generic template.

Benefits of the Object-Oriented Approach

This architecture has simplified the construction of graphical animations, due to the benefits of the object-oriented paradigm: abstraction, encapsulation, extensibility and reusability.

The constraint functions which represent a mechanical linkage, and their Jacobians, are the essential characteristics of the linkage, from the point of view of kinematic simulation. The essential characteristics of

a thing constitute an abstraction (Booch (1990)). This abstraction, and its direct implementation in Eiffel as the deferred class JOINT, have simplified the connection between the code which implements the constraint functions and the code which uses
them to calculate the system state.

Conversely, the implementation of a subclass of JOINT representing some linkage requires no knowledge of what code will call it, or in what circumstances; it simply evaluates a linkage's constraint functions, and their Jacobians. The code which calls these routines can make no assumptions about how these routines are implemented, beyond the interfaces specified in the class JOINT. This enforcement of information hiding is encapsulation (Booch (1990), Micallef (1988)). This encapsulation simplifies the implementation of new subclasses of JOINT by allowing them to be implemented in ignorance of the context in which they will be used.

New linkages have been implemented very simply, by plugging expressions for the linkage's constraint functions and the components of its Jacobian into a stock template. This easy addition of new features, allowed by the object-oriented architecture and implementation, constitutes extensibility (Booch (1990), Micallef (1988)).

Several new simulation programs have required the addition of special-purpose linkages. These were added simply, with no modifications to the rest of the simulation libraries, allowing the rest of the libraries to be used unmodified for the simulation of new mechanical systems. This reusability (Micallef (1988)) would not have been possible without the object-oriented architecture and implementation.

Conclusions

The object-oriented architecture described in this paper results in a kinematic simulation system which

embodies the virtues of the object-oriented design and implementation of software: abstraction, encapsulation, extensibility and reusability. The implementation of such a system without an object-oriented approach would have been considerably more complex and error-prone, involving (e.g.) tags and case statements in the place of inheritance. It is expected that the same benefits will accrue to a dynamics simulation system based on the same abstraction of a mechanical linkage.

References

Booch, G. (1990) Object Oriented Design. The Benjamin/Cummings Publishing Company, Redwood City, California.

Haug, E. J. (1989) Computer-Aided Kinematics and Dynamics of Mechanical Systems. Allyn and Bacon, Boston.

Micallef, J. (1988) Encapsulation, Reusability and Extensibility in Object-Oriented Programming Languages. J. Object-Oriented Programming 1 (1), 12-36.

Müller, W. (1983) The Knee - Form, Function and Ligament Reconstruction. Springer-Verlag, Berlin.

Nikravesh, P. E. (1988) Computer-Aided Analysis of Mechanical Systems. Prentice-Hall, Englewood Cliffs, New Jersey.