

An Algorithm with Logarithmic Time Complexity for Interactive Simulation of Hierarchically Articulated Bodies Using a Distributed-Memory Architecture

Thomas Jung

Abstract—The dynamic simulation of large systems of hierarchically articulated bodies is very time-consuming and cannot be done in real time using current one-processor workstations. This paper discusses the parallelization of dynamic simulation algorithms for such systems. First, the methods used in computer animation for solving initial-value problems are compared. Using test suites based on articulated bodies, the Gear method for non-stiff problems is identified as the solver with the least right-hand-side evaluations of the differential equations for the executed tests. On the basis of this method and the articulated-body method, a "simulation engine" for distributed-memory architectures is presented. By reimplementing the Gear method and rearranging its parts, logarithmic time complexity is achieved for hierarchically articulated bodies. By using the simulation engine, interactive, physically based animation of large articulated figures is possible.

Keywords—Articulated figures; dynamic simulation; interaction; distributed-memory architecture; logarithmic time complexity

I. BACKGROUND

A. Interactive Simulation of Articulated Figures

Interaction has proved to be a powerful technique for generating animations or exploring virtual worlds. To interactively manipulate physically based scenes, real-time simulation algorithms are needed. Various investigations have been carried out to simulate the motions of articulated figures using rigid-body dynamics[1], [2], [3], [4], [5], [6]. Most of them are unsuitable for interactive control. Efforts have been made to speed up simulation so as to allow interactive manipulation of small or medium-sized figures[3], [7], [6], [8].

The simulation of large systems of hierarchically articulated bodies (HABs), e.g. human models with more than 30 degrees of freedom, enables more sophisticated models of human beings or creatures to be obtained. Real-time simulations for these figures are not feasible using current one-processor workstations. There are two possible ways of speeding up the solution of the initial value problem, which is required for interactive manipulation of physically based HABs:

- reducing the number of sequentially computed solutions of the forward-dynamics problem (right-hand-side evaluations of the differential equations) per second, or
- reducing the time complexity of the forward-dynamics algorithm

T. Jung works for the GMD – the German National Research Center for Information Technology, Research Institute for Computer Architecture and Software Technology, Berlin, Germany. E-mail: tj@first.gmd.de .

B. Solving the Forward-Dynamics Problem

There are basically two approaches for solving the forward-dynamics problem:

- calculating recursion coefficients with propagation of motion and force constraints along the mechanism, allowing the problem to be solved directly, or
- obtaining and then solving a set of simultaneous equations in the unknown joint accelerations.

The second approach (e.g. the composite-rigid-body method[9]) allows more flexibility of control by specifying constraint equations[10], but it can only achieve a $O(n^3)$ computational complexity. In systems based on this approach, simulations are not executed in real time [1], [4], [5].

For interactive control of dynamic simulations, the first approach (e.g. [11], [12], [13]) is preferred [3], [6], because it has only linear computational complexity. Featherstone shows[14, p. 151] that the articulated-body method[12] (definition given in appendix) is faster for articulated figures with more than 9 degrees of freedom than the version of the composite-rigid-body method given in[14].

The articulated-body method can be extended to handle loops (e.g. resulting from contacts with the environment)[13]. The computational complexity is not worse than $O(n) + O(l^2)$ for l internal loops.

C. Solving the Initial Value Problem

There is no agreement on the choice of a solver for animating articulated figures. Some authors prefer Runge-Kutta methods [10] because of their ease of use. Information from the previous steps is not needed, so adaptive variation of step size and discontinuous force functions (e.g. in collision situations) can be easily supported.

Multistep methods use information from previous steps to reduce the number of right-hand-side evaluations. Using the Nordsieck-history representation [15], the starting of the solver can be avoided, and adaptive step-size controlling can be carried out.

The Gear method changes not only the step size, but also the order of the interpolation polynomial adaptively. Implementations of this method are available in FORTRAN[15], [16].

Other methods are available, but they are not suited to the animation of articulated figures (e.g. too many right-hand-side evaluations in extrapolation methods) or they are currently under investigation (e.g. cyclic multistep methods[17]). Special methods should be used for stiff situations (e.g. BDF formula [15]).

Green proposes a test suite allowing the experimental comparison of some solvers [18]. He uses a small set of differential equations to simulate linear and non-linear problems with discontinuities and stiffness. The Adams method was the most efficient in the tests[18].

D. Parallelization of Simulation Algorithms

There are two ways of parallelizing simulation algorithms

- across time, and
- across space

Small-scale across-time approaches are available for speeding up Runge-Kutta methods[19]. Large-scale across-time approaches would need knowledge about the future that is not available in interactive dynamic simulations.

Large-scale parallelization can only be carried out across space for large systems of HABs. Some attempts are made to parallelize parts of the composite-rigid-body method [20], [21]. A parallel version of the composite-rigid-body method with time complexity $O(n^2)$, and a parallel version of the conjugent-gradient method [9] with time complexity $O(n * \log(n))$ are given in [22]. A small-scale parallelization of the Gear method for a shared-memory architecture is given in [23].

Parallel simulation algorithms are also used in [3], [24], but no detailed description is given.

Following[25], Lee and Chang show[22] that the time complexity of the articulated-body method cannot be reduced for any linearly chained rigid bodies by parallelization. However, parallelizing the method for HABs will reduce time complexity.

II. EXPERIMENTAL COMPARISON OF SOLVERS

The most efficient methods should be considered for parallelization. To solve the forward-dynamics problem, the articulated-body method is the method that should be chosen. However, there is no agreement on which method should be chosen to solve the initial-value problem. For this reason, a test suite for comparing solvers based on HABs has been developed.

A. Dynamic Model

Different shaped bodies merely result in different constants in the same differential equations. Because of this the shape of a body ought to be less important than the structure of the articulated figure. Different constants can also be obtained by scaling a single rigid body. Consequently, only one type of body is used for the test suite (see Figure 1). This should facilitate reconstruction of the experiments conducted. The dynamic description of the body is given in Appendix B.

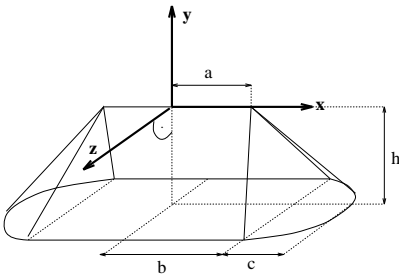


Fig. 1. Wedge-shaped rigid body

Hierarchical mobiles are constructed on the basis of these bodies. A so-called mobile consists of s stages. At the root of the mobile is a single chain forming the first stage. At the end of this, and every other chain, hangs a plateau. O chains hang on each plateau these again ending in plateaus.

TABLE I
PARAMETERS OF FIGURE 2

a	b	c	h	s	l	o	p	q
0.2	0.1	0.2	0.5	3	4	5	2.0	2.0

All chains have l links, including the plateau. All bodies in the tree have the same a -values and the same heights h . All bodies that are not plateaus have the same values for b and c (see Figure 1). Since plateaus have circular bottoms, their b -value is zero. The radius of the bottom depends on the stages hanging on the plateau. The radius of the tree's leaves which are also plateaus, is the sum of the values c and b from links that are not plateaus.

For all other plateaus, the following rules are valid: All chains hang at the brim of the plateau. The outer corner of every body's upper edge cuts the brim. The lines extending the upper edges of the chains hanging on the plateau intersect at the center of its bottom.

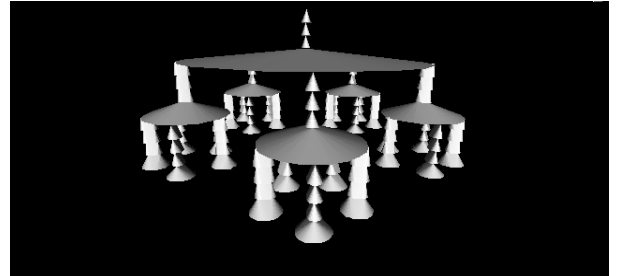


Fig. 2. Mobile

The radius c of the plateau is $(\frac{p*o}{\pi})^k$, where k is the number of stages of the "submobiles" hanging on the bottom of the plateau. The length of the arc between two neighboring chains at the stage containing the leaves of the tree is p .

The axis of the rotational joint linking two bodies is the upper edge of the first body and lies in the plane of the bottom of the second one crossing its center. The joint axes of the links of a chain are parallel.

Using these definitions, many mobiles can be generated by varying only nine parameters (including the density q , which is equal for all bodies). For example, the mobile in Figure 2 can be described by the parameters given in Table I.

Given zero joint values, velocities and forces, the mobiles do not move in a gravitational field. The amount of motion can be controlled by varying the start position of the mobile.

In all the experiments described in this paper, the forward-dynamics problem is solved by using the

TABLE II
THE USED SOLVERS WITH THEIR ABBREVIATIONS

Abbr.	Method	order	step size control
IEC	Improved Euler-Cauchy	2	SD
RKI	Runge-Kutta I	3	EF
RKG	Runge-Kutta-Gill	4	SD
RKF	Runge-Kutta-Fehlberg	5	EF
Gear	Gear method for non-stiff equations	*	*
BDF	BDF method for stiff equ.	*	*

articulated-body method.

B. Selection of Solvers

Since there are many different versions of solving methods, an appropriate selection of solvers must be made. For the following experiments, four different Runge-Kutta methods, an adaptive-order Adams method (the Gear method for non-stiff problems with the stepsize control from Hindmarsh [16]) and a method suited to stiff differential equations ("lsode", based on Gear's BDF-method [16]) were chosen (see Table II). All methods use adaptive step-size control. For the Runge-Kutta methods, step size is controlled by step doubling (SD) [15, p.81] or by using Runge-Kutta methods with different orders (embedding formulas, EF). Details are given in [26].

To verify the C^{++} implementations of the solvers, simulations using the test suite of Green[18] are executed. The number of steps used by the described implementations are similar to those of Green[26].

C. Test Conditions

Since the different step-size control mechanisms are based on different error-estimation strategies, the chosen error tolerance may influence the results of the comparison. For example, a solver with a pessimistic error estimation could achieve better results with larger tolerances, whereas a smaller error tolerance should be given to a solver with an optimistic estimation. For this reason, all tests are executed with five different error tolerances (relative and absolute) from 10^{-2} to 10^{-6} . Only the best result for each solver is considered.

To assess the performance of a solver, the number of right-hand-side evaluations per second is used. In general, the numerical effort required by the solver can be neglected compared to that, required by the forward-dynamics algorithm (e.g. a Runge-Kutta method with f function evaluations per step needs only $f * n$ multiplications per right-hand-side evaluation, whereas the articulated-body method needs $300 * n$ multiplications, n being the number of degrees of freedom).

To ensure the interactivity of a simulation over the whole simulation interval, the maximum number of right-hand-side evaluations per second can be considered. If the average number of evaluations allows real-time simulation, but

TABLE III
PARAMETERS OF MOBILES

Session	h	s	l	o	p
A)	10.0 / l	1	2 ... 12	-	-
B)	1.0	2	3	3 ... 11	3.0
C), D)	1.0	1	4	-	-

many more evaluations per second are needed in some intervals, the simulation will temporarily slow down. If the length of the interval is shorter than the length of the interval between two frames, no visible effect will be noticeable. If its duration is greater, a waiting period must be accepted. To avoid this effect, a minimal step size can be given in many solvers, but this might result in stability problems. The use of the maximum number of right-hand-side evaluations per second is discussed in [26]. In the following tests, the average number is given as a suitable measure for assessing the time required by the solver.

If simulation is used to predict the behavior of a real system, the error of the simulation has to be considered. Fortunately, large errors are acceptable in computer animation because, in general, only the visual impression is of major interest. Thus, only instable solutions (resulting from too large error tolerances) are removed from the following tests. In each test, ten seconds of real time are simulated using a SUN4/75C workstation.

D. Results

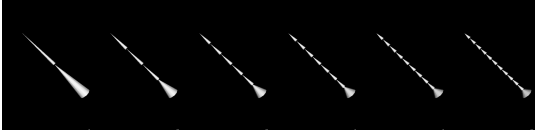
The following four test sessions represent several tests carried out to identify the fastest solver. An overview of the varying mobile parameter is given in Table III (the values $a = 0.2$, $b = 0.3$, $c = 0.2$ and $\varrho = 2.0$ are fixed for all tests). Animation is generated by giving the mobile a start elongation of 0.8 radians in the gravitational field of the Earth. The start velocities are zero; external forces are not specified.

In the first two sessions (Tables IV and V), the depth and breadth of the mobile are varied. The larger the mobile, the more right-hand-side evaluations are needed. In every case, the Gear method is more than twice as fast as the best Runge-Kutta method (RKG). To simulate wide trees, the Gear method does not need more evaluations than for long chains (e.g. 11 chains with 36 joints vs. 12 linearly jointed links). Thus, if large articulated figures are to be simulated, the Gear method should be the one chosen.

In general, there is a dependency between the step size of the solver and local changes in the solution function. Since the change of the position as a function of time is expressed as velocity, fast-moving mobiles should be considered.


In session C), the velocity is varied by giving the mobile start elongations from 0.1 to 1.7 radians (in steps of 0.4 radians). The different potential energy at the beginning of the simulation transforms to different kinetic energy (and hence different velocities). The number of evaluations does not increase by more than a factor of two when using the Gear method in this session (see Table VI). Thus, it is also

TABLE IV
SESSION A) VARIATION OF THE NUMBER OF LINKS



links	number of evaluations					
	IEC	RKI	RKG	RKF	Gear	BDF
2	63.4	90.0	85.6	58.8	15.4	27.1
4	108.2	301.2	109.6	120.6	34.1	79.2
6	135.0	498.6	155.6	165.0	46.4	108.6
8	179.8	680.4	168.4	210.0	64.7	161.6
10	215.0	903.6	187.6	252.6	81.1	229.2
12	234.4	1046.1	200.0	296.4	91.6	203.3

TABLE V
SESSION B) VARIATION OF THE NUMBER OF CHAINS



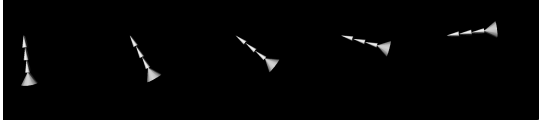
chains	number of evaluations					
	IEC	RKI	RKG	RKF	Gear	BDF
3	186.8	784.8	197.2	211.2	59.6	395.4
5	189.6	836.7	195.2	218.4	61.3	460.7
7	197.2	820.5	187.6	217.2	54.5	437.5
9	245.6	1137.6	202.8	280.2	65.0	1062.0
11	310.4	1692.9	271.2	388.2	85.2	1584.1

the preferred method for fast-moving HABs.

The application of actuator forces can result in discontinuities (e.g. in the case of collisions) and stiff situations (limited time intervals, where the equations are stiff). In the case of discontinuities, the only thing to do is restart the solver. However, stiffness is a more serious problem in the real-time simulation of large systems of HABs. To simulate the impact of stiffness on the solver, viscous damping is introduced. A joint force ($Q_i = -k_{visc} * \dot{q}_i$) is applied to each joint. The viscous coefficient is varied from 0 to 10 in steps of 2.5 in test session D). The chain has four links; the start elongation is the same as that in the sessions A) and B).

Methods for non-stiff problems should not be used to solve stiff differential equations. Table VII shows that for stiff problems the BDF method is superior to the other tested solvers. In general, however, the BDF method cannot be chosen. In the case of large systems of HABs, its numerical cost increases rapidly with the number of degrees

TABLE VI
SESSION C) VARIATION OF START ELONGATION



start pos.	number of evaluations					
	IEC	RKI	RKG	RKF	Gear	BDF
0.1	125.8	263.4	108.8	133.2	44.5	62.4
0.5	128.6	438.0	127.2	160.8	46.5	79.1
0.9	171.4	591.6	174.8	185.4	50.6	156.2
1.3	209.4	963.0	233.2	237.0	71.8	266.8
1.7	214.2	1189.2	325.6	298.2	82.7	361.7

TABLE VII
SESSION D) VARIATION OF THE VISCOUS DAMPING FACTOR

viscous damp.	number of evaluations					
	IEC	RKI	RKG	RKF	Gear	BDF
0.0	158.0	505.5	157.2	174.6	48.1	120.2
2.5	310.4	318.0	416.4	380.4	371.6	27.3
5.0	582.2	640.8	985.2	702.6	766.5	30.5
7.5	954.4	946.8	1557.2	1026.6	1227.7	30.4
10.0	1110.0	1260.9	2040.0	1369.2	1609.0	26.5

of freedom (see e.g. Table V), because the method needs to compute the jacobian of the system, which has at least an $O(n^2)$ complexity. If there is no method for solving stiff differential equations with linear complexity, real-time simulation of large articulated figures will not be possible in stiff situations.

If one of the solvers that are unsuitable for stiff equations is used, stiff situations should be avoided in interactive simulations. Actuator models should be analyzed with respect to their impact on stiffness.

If stiffness is avoided, the Gear method is the fastest method in all tests. Thus, a simulation engine for interactive manipulation of large systems of HABs should combine this method and the articulated-body method, if a large-scale parallelization is possible.

III. DISTRIBUTED SIMULATION ENGINE

This section describes the distributed simulation engine. First, parallelization of the articulated-body method is carried out over the breadth of the tree. Consideration of the exchange of data in a distributed-memory environment and the linear computational complexity of the solver result in parallelization of the Gear method as well. To distribute the method, parts must be rearranged. Finally, algorithms are presented that control the automatic distribution of the new versions of the algorithms.

A. Parallelization of the Articulated-Body Method

The only way to obtain a large-scale parallelization of the articulated-body method is to parallelize across the breadth of the tree structure of the HABs, because linear chains cannot be adequately parallelized. If different chains in the tree can be executed in parallel, the time complexity should be proportional to the length of the longest chain, which is in fact the depth of the tree. Since there is a logarithmic dependency between the number of nodes and the depth of a tree, a time complexity of $\log(n)$ can be achieved after parallelization (n being the number of degrees of freedom).

For parallelization of the articulated-body method, four equations (given in Appendix A) must be considered. In equations 1 and 4, the index λ_i on its right-hand side indicates that information is transferred in the direction from the root to the leaves. In equations 2 and 3 information is transferred in the opposite direction (indicated through μ_i). Since velocity-based information (\hat{p}_i^v) is needed in equation 3, the joint velocities must be computed prior to the values of \hat{p}_i and \hat{I}_i^A . On the other hand, to compute the accelerations (equations 4 and 5), the articulated-body inertias must be known, because they are used to compute the values of \hat{h}_i . To obtain the information when all joint accelerations are computed, a final pass toward the root is needed for synchronization. The data flow of the computation is shown in Figure 3.

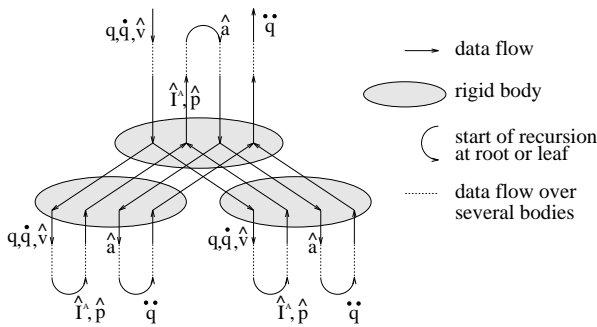


Fig. 3. Data flow of the articulated-body method

B. Considering Data Exchange in a Distributed-Memory Environment

When the joint accelerations are known at the root, a central solver can compute the joint values and velocities for the next iteration of the forward-dynamics algorithm.

However, two further considerations must be taken into account. In Section II-C, the computational cost of the solver is compared with the cost of the forward-dynamics algorithm. To identify the fastest solver, the required solver operations were neglected. However, if the time complexity of the parallelized forward-dynamics algorithm is only logarithmic, the mostly linear computational complexity of the solver can no longer be neglected.

If a distributed-memory architecture is used, the exchange of data must be considered, too. Since the parallelization of the articulated-body method has a fine granu-

TABLE VIII

DATA TRANSMISSION BETWEEN NEIGHBORING LINKS

pass	toward	transmitted information	number of floats
1	leaves	q, \dot{q}, \ddot{v}	$2n+6$
2	root	\hat{p}, \hat{I}^A	33
3	leaves	\hat{a}	6
4	root	\ddot{q}	n

lization of the articulated-body method has a fine granularity (e.g. while computing the spatial velocities, less than fifty floating-point operations per joint are carried out), the cost for communication can be greater than those for computation. The amount of data that must be exchanged between two neighboring but distributed links grows with the number of nodes in the tree (see Table VIII). Since there is a dependency between the execution time of the data transmission and the amount of data to be transmitted, the time complexity per simulation step is, strictly speaking, more than linear. Nevertheless, for medium-sized articulated figures this effect can be neglected. Since we are concerned with the simulation of large systems of HABs, we attempt to reduce communication cost.

C. Distributing the Gear method

Computation of the new joint values and velocities (in the case of the Gear method: prediction and correction) can be performed per joint. Information from other joints is not needed. Thus, parts of the method can be distributed. However, to determine step size and order, information from all joints must be considered.

All implementations of the Gear method known to the author [15], [16] are given in FORTRAN. Since the monolithic structure of the programs is not particularly conducive for parallelization (compare e.g. [23]), an object-oriented reimplement in C^{++} is performed.

The flow chart of the method is shown in Figure 4. Rectangles with a black bar on their left side stand for operations over the whole state space of the method that can be performed independently for each component. Rectangles with triangles on their left side symbolize operations over the state space with additive connections of the components.

The next step to be performed is the reconfiguration of the individual parts of the method for parallelization. The parts are assigned to three phases:

- preparation,
- follow-up and
- control

In the control phase, the new step size and order are computed. Vectorial information is not used in order to avoid linear time complexity of the parallel implementation.

In the Gear method, weighted L_2 -norms of different vectors are used for determining corrector convergence, plausibility of the corrected values and for computing step size and order [15]:

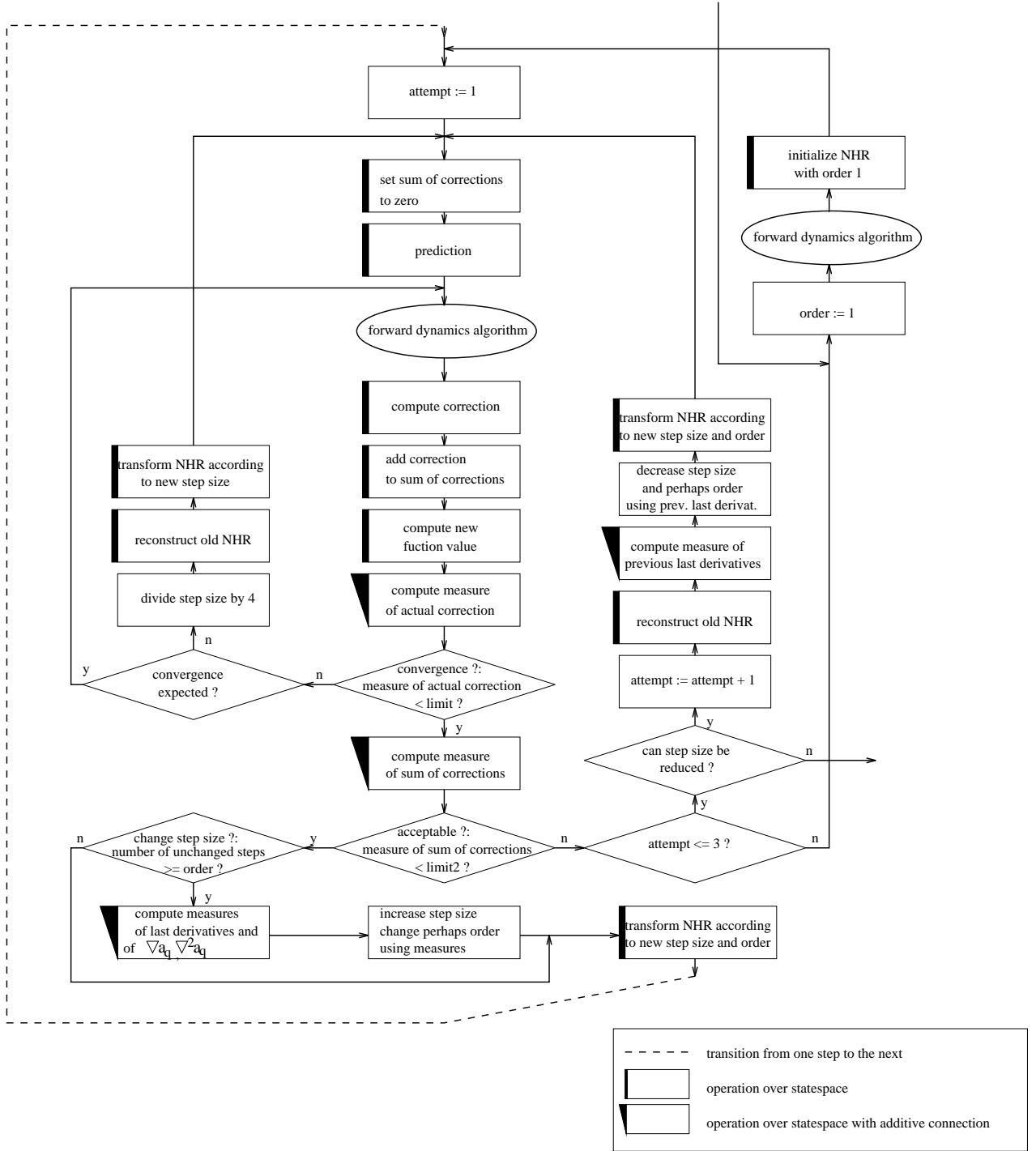


Fig. 4. Data flow of the Gear method

- actual correction $c * G(Y_{n,(m)})$,
- last row of Nordsieck-history representation a_q ,
- ∇a_q , the change of a_q , that is de facto the sum of corrections,
- $\nabla^2 a_q$, the change of ∇a_q and
- an old value of a_q for steps that have to be repeated

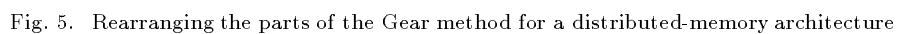
$$\left\| \frac{v}{\omega} \right\|_2 = \sqrt{\frac{1}{n} \sum_{i=1}^n \left(\frac{v_i}{\omega_i} \right)^2},$$

where

$$\omega_i = |a_{1,i}| * Tol_{rel} + Tol_{abs}$$

The weighted L_2 -norm of a vector v is defined by

Tol_{rel} and Tol_{abs} are the relative and absolute error to-



drates of its children and of its own joint.

These and all other operations that are carried out over the state space are assigned to the preparation phase if they have to be performed after the control phase and before calling the forward-dynamics algorithm, and to the follow-up phase in the other cases.

The new structure of the Gear method suitable for distribution is shown in Figure 5. The dashed line indicates that the transition from one step to the next is now hidden in the preparation phase. However, this is not a real problem, because after a successful step only the joint values (and velocities) are used, and not the whole Nordsieck-history representation, which is updated in the next preparation phase.

Since the weighted L_2 -norms of a_q , ∇a_q , and $\nabla^2 a_q$ are only used for determining the new step size and order, in the original version of the Gear method these operations have to be performed relatively seldom compared with the number of function evaluations. Often only the weighted L_2 -norm of the actual correction is computed per function evaluation. To have the measures available if needed in control phase, *all of them* have to be computed *per function evaluation* after rearrangement. Fortunately, the additional operations needed for computation of the measures can be neglected compared with the operations needed for solving the forward-dynamics problem.

To avoid further passes per function evaluation, the preparation and follow-up phases are integrated into the first and fourth pass of the distributed articulated body, respectively. The combination of the parts of the articulated-body and the Gear method is shown in Figure 6. Apart from the starting of the integrator (pass 0 in Table IX), where the starting joint values and velocities are fed to the distributed tree, only a constant number of values is transmitted from one node to another.

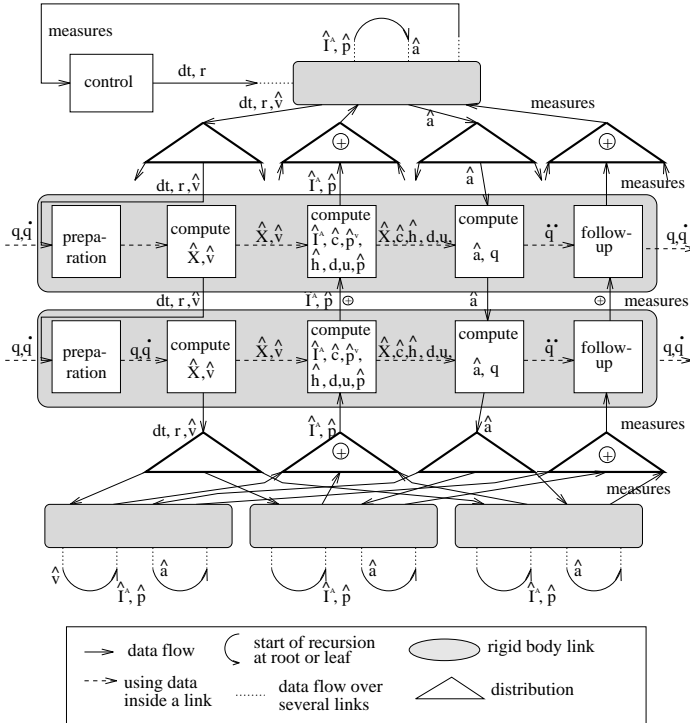


Fig. 6. Combination of the Gear method and the articulated-body method

Following the described steps, the *simulation* of hierar-

chical articulated figures can be executed with logarithmic time complexity.

Two further points must be considered: the visualization of the figures, and the application of forces. For rendering the figure, all joint values are needed. The transmission of the values to a central rendering unit requires at least linear cost because of the operations needed for transmitting data. Since the number of function evaluations per second (e.g. 100 to 10,000) is usually much bigger than the frame rate (e.g. 30 frames per second), this cost can be often neglected. If this is not the case, the combination of rendering and simulation software will have to be considered.

TABLE IX

DATA TRANSMISSION BETWEEN NEIGHBOURED LINKS, DISTRIBUTING THE GEAR METHOD

pass	toward	transmitted information	num. of floats
0	leaves	q, \dot{q}, \hat{v}, t , step size, Tol_{rel}, Tol_{abs}	$2n+10$
1a	leaves	\hat{v}	6
1b	leaves	\hat{v} , new step size, new order	8
2	root	\hat{p}, \hat{I}^A	33
3	leaves	\hat{a}	6
4	root	$\sum (\frac{c * G(Y_{n_i}(m))}{\omega})^2, \sum (\frac{a_{q,old}}{\omega})^2, \sum (\frac{a_{q,new}}{\omega})^2, \sum (\frac{\nabla a_q}{\omega})^2, \sum (\frac{\nabla^2 a_q}{\omega})^2$	5

Analogous to this, the actuator model should be combined with the simulation software to avoid the feeding of forces into and the retrieval of joint values from the tree. To simulate cyclic articulated structures, the computation of the loop-closure forces should also be combined with the four passes of the articulated-body method (the loop-closure forces can be seen as an actuator model).

D. Controlling the Distribution

Since linear chains cannot be parallelized, chains are computed by one processor sequentially. The easiest way to distribute data at branches is to start up all branches sequentially. To reduce start-up times, the starting up of branches should be performed in parallel, too. To obtain the minimum start-up time for all branches, each branch must start other branches until all branches have been started. On the other hand, not all the branches that are to be started have the same size and hence the same time. Branches should be started in the order of their costs (the largest branch first) and not at the same time in order to minimize the time needed for execution.

Hence, a compromise between the two strategies is chosen. To get a simple algorithm, every branch starts at most two sibling branches before it begins computing its own chain (Figure 7). The branch with the longer execution time is started first.

What is needed, then, is a method for estimating the duration of the computation of a branch depending on a given distribution. The algorithm that estimates the time needed for execution of a branch depending on the estimated durations

- of the computation of a single body and (CMPTM)
- of the communication between two distributed bodies (ROITM)

per function evaluation can be sketched as follows:

```

estimated_duration(tree w)
{
  if (w is empty)
    return 0
  else
  {
    dl := 0, dr := 0, dc := 0

    if (left sibling node has processor other than w)
      dr := ROITM,
      dl := dr + estimated_duration(left siblings)
    else
      dr := estimated_duration(right siblings)

    if (right sibling has processor other than w)
      dc := dr + ROITM,
      dr := dr + ROITM
      + estimated_duration(right siblings)
    else
      dc := dr + estimated_duration(right siblings),
      dr := 0

    dc := dc + length of local chain * CMPTM
      + estimated_duration(children)

    return max(dl, dr, dc)
  }
}

```

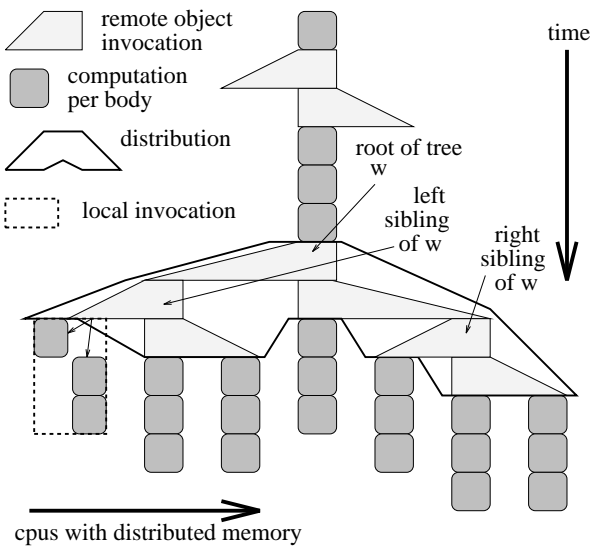


Fig. 7. Distribution on a fork

In the case of there being more branches than processors, there must be a strategy for assigning branches to

processors. Moreover, it may be inefficient to assign a small branch to a separate processor if the communication cost exceeds the cost for computing the branch (e.g. the small branch with one chain on the left-hand side of Figure 7 is computed locally because distribution would not reduce the time needed for execution).

The idea behind the algorithm (given in detail in [26]) is to use a further processor in the branch, that currently needs the longest execution time. The term "is time-reducible" means that this time can be reduced. It cannot be reduced, for example, if every sibling in the branch is assigned to a different processor. Then the minimum execution time is reached. The algorithm can be sketched as follows:

```

distribute n processors over tree w
{
  assign processor 1 to all elements of tree w

  for (processor p := 2
      p ≤ n and tree w is time reducible
      p := p + 1)
  {
    use processor p in tree w
  }
}

```

```

use processor p in tree w
{
  if (left sibling of w is not empty and
      (left sibling has the same processor as w or
       left sibling is time-reducible))
    compute estimated_duration(left siblings)

  if (right sibling of w is not empty and
      (right sibling has the same processor as w or
       right sibling is time-reducible))
  {
    compute estimated_duration(right siblings)
    add ROITM if left sibling is called per ROI
  }

  if (children of w are not empty)
    compute estimated_duration(children)
    add length of local chain * CMPTM
    add one or two times ROITM
    if sibling nodes are called per ROI

  if (this duration is the longest)
    use processor p in the subtree of children
  else
  {
    assign processor p to the subtree with the
    longest duration

    if (left sibling has same processor as w and

```

```

    right sibling does not)
    /* Do the ROI before local computations !!! */
    change siblings
  }
}

assign processor p to subtree w
{
  if (previously assigned processor of w equals
      processor of caller)
    assign processor p to all elements of subtree w
  else
    use processor p in tree w
}

w can be either a subtree or the root of the subtree de-
pending on the context of the operation. The two subtrees
that are called from a subtree before it starts its compu-
tation are called left and right sibling (see Figure 7). The
left sibling is the sibling called first.

```

E. Results

The algorithms discussed in the previous sections are implemented on MANNA¹ [27], developed at GMD FIRST, which is a MIMD architecture with distributed memory using a crossbar for information transmission. At present, a version with twenty application processors (Intel i860XP) is available, running the operating system PEACE² [28] which was also developed at the GMD FIRST.

PEACE offers an object model supporting the structuring of parallel programs called "dual objects" [29]. The call of a method of an object located in a different memory area is called Remote Object Invocation (ROI).

The time per ROI for parameters smaller than 49 Byte is about 160 μ s [30, page 154]. For larger amounts of memory, the time per ROI increases with the amount of memory. The cost for communication between two distributed neighbors per function evaluation (ROITM) is about 1.5 milliseconds (for four ROIs including packing, unpacking, etc., measured on MANNA). The numerical cost per link and evaluation (CMPTM) is about 0.4 milliseconds (using a C++-to-C-translator and a C-compiler without optimization like pipelining, measured on MANNA).

To check the logarithmic time complexity, a set of different-sized articulated figures is needed that can be represented by a single logarithmic function (expressing the dependency between the depth of the figure and the number of joints). The parameters of such a set of mobiles are given in Figure 8. The interpolation line shows that the mobiles of the set lie near to the given logarithmic function (an exact match is not possible because of the few mobile parameters that can be varied).

Figure 9 gives the measured evaluation times of the mobiles on MANNA. It shows that the results of the algorithm used for estimation of the time expenditure of a gi-

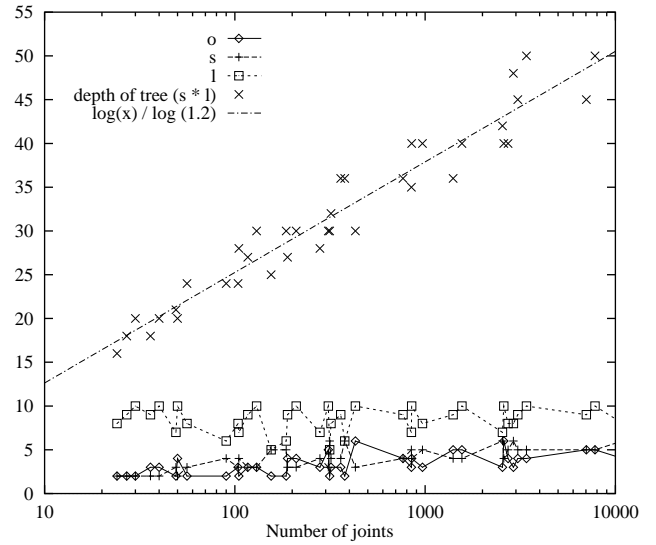


Fig. 8. Mobile parameters

ven distribution are comparable with the measured values for twenty processors. Assuming that the duration of the function evaluation on a MANNA with several thousand processors can also be estimated with the given algorithm, it can be gathered from Figure 9 that the distributed simulation engine has only logarithmic time complexity.

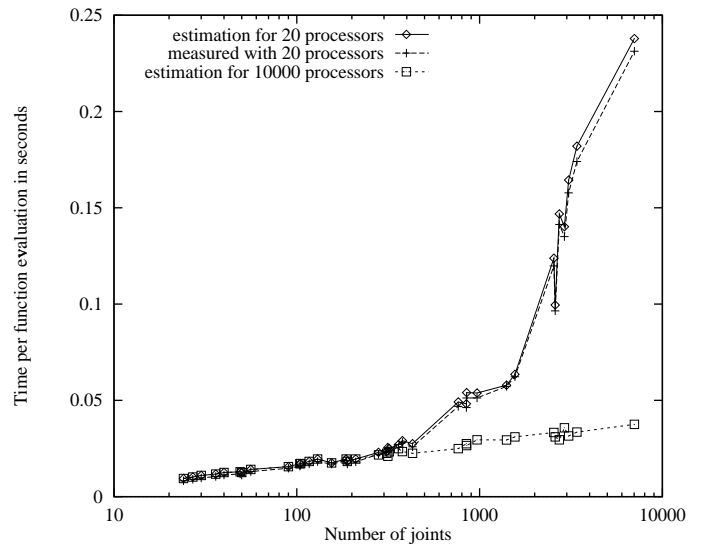


Fig. 9. Evaluation time

To test the distribution algorithm, the evaluation time as a function of the given number of processors is estimated for two different mobiles. Problems that are ideally scalable should have speed-up that is linearly dependent on the number of processors. However, the speed-up of the presented simulation engine is limited by the logarithmic dependency between the number of joints in the tree and the depth of the tree (number of joints / depth of tree). Thus, the parallel function evaluation can only be about 30

¹Massively Parallel Architecture for Numerical and Non-numerical Applications

²Process Execution and Communication Environment

times faster than the sequential one for the smaller mobile with 455 joints, and about 200 times faster for the bigger mobile. Looking at Figure 10, however, it is obvious, that the speed-up of the small mobile is not higher than 10 (the maximum speed-up is attained with 81 processors at the latest, because this is the breadth of the tree).

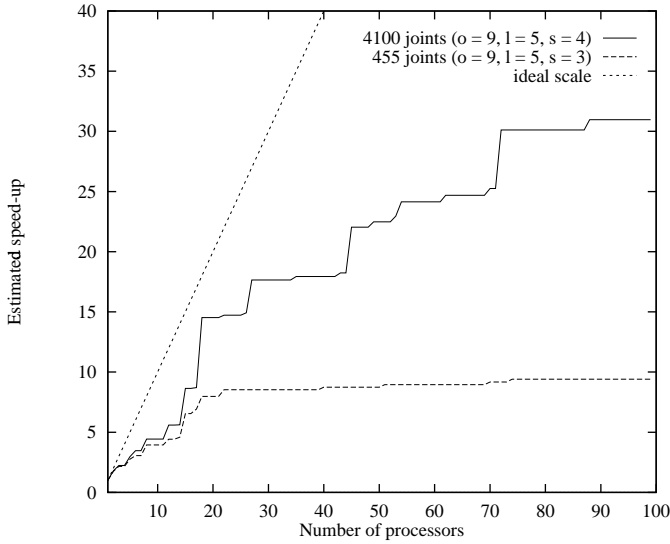


Fig. 10. Speed-up as a function of the number of processors

The "missing" speed-up can be interpreted as the communication cost needed per function evaluation. Since this cost is very high in relation to the cost needed for computation, a reduction in communication time could result in a reasonable reduction of the time required per function evaluation. On MANNA, each application processor has an associated communication processor that is not currently used. The use of this processor for communication in parallel should give a reasonable speed-up.

IV. CONCLUSION

The most efficient algorithms should be used to implement a real-time simulation engine that allows interaction. Since there is no agreement as to which method should be used for the physically based animation of articulated figures, a test environment based on HABs is implemented to compare different solvers. Several tests show that the Gear method for non-stiff problems needs the fewest right-hand-side evaluations. Hence, this method should be used in combination with the articulated-body method in an one-processor environment.

Because of the high numerical cost, real-time simulation of large systems of HABs is not possible with the currently available one-processor workstations. In this paper, parallel variants of well-known algorithms are presented which exploit the benefits of massively parallel distributed-memory architectures. The presented variants have only logarithmic time complexity, and are therefore suitable for simulating very large systems of HABs on a large-scale architecture.

On the basis of the simulation engine, an interactive demonstration program is implemented allowing the swinging of mobiles up to about twenty links. Larger mobiles can also be manipulated interactively if the user avoids abrupt movements with the three-dimensional input device (with six degrees of freedom) which would result in discontinuous force functions and stiff situations.

Acknowledgements

My thanks go to Stefan Jähnichen and Dietmar Jackèl for supervising my Ph.D. Thesis; to Stefan Neunreither for the many fruitful discussions we had, and to Birgitt Schmidt and Phil Bacon for polishing up the English text.

APPENDIX

I. ARTICULATED-BODY METHOD

The articulated-body method [12] for hierarchically articulated figures is defined using "spatial algebra" [14] by the equations:

$$\hat{c}_i = \hat{v}_i \times \hat{s}_i \hat{q}_i$$

$$\hat{h}_i = \hat{I}_i^A \hat{s}_i$$

$$d_i = \hat{s}_i^S \hat{h}_i$$

$$u_i = Q_i - \hat{h}_i^S \hat{c}_i - \hat{s}_i^S \hat{p}_i$$

$$\hat{v}_i = {}_i\hat{X}_{\lambda_i} \hat{v}_{\lambda_i} + \hat{s}_i \hat{q}_i, \quad (\hat{v}_0 = \hat{0}) \quad (1)$$

$$\hat{p}_i^v = \hat{v}_i \times \hat{I}_i \hat{v}_i$$

$$\hat{I}_i^A = \hat{I}_i + \sum_{j \in \mu_i} {}_i\hat{X}_j \left(\hat{I}_j^A - \frac{\hat{I}_j^A \hat{s}_j \hat{s}_j \hat{I}_j^A}{\hat{s}_j^S \hat{I}_j^A \hat{s}_j} \right) {}_j\hat{X}_i \quad (2)$$

$$\hat{p}_i = \hat{p}_i^v + \sum_{j \in \mu_i} {}_i\hat{X}_j \left(\hat{p}_j + \hat{I}_j^A \hat{c}_j + \frac{u_j}{d_j} \hat{h}_j \right) \quad (3)$$

$$\hat{a}_i = {}_i\hat{X}_{\lambda_i} \hat{a}_{\lambda_i} + \hat{c}_i + \hat{s}_i \hat{q}_i, \quad (\hat{a}_0 = \hat{0}) \quad (4)$$

$$\hat{q}_i = \frac{u_i - \hat{h}_i^S {}_i\hat{X}_{\lambda_i} \hat{a}_{\lambda_i}}{d_i} \quad (5)$$

where i is the body index, λ_i the index of the predecessor of i , and μ_i the set of indices referring to the children of i .

The joint force is called Q_i , the joint value q_i . Its time derivatives are called \dot{q}_i (joint velocity) and \ddot{q}_i (joint acceleration).

\hat{p}_i refers to the spatial bias force of the body i (\hat{p}_i^v to velocity-product forces) and \hat{I}_i to the rigid-body inertia. The "inertia" of dynamic assemblies of rigid bodies is called articulated-body inertia (\hat{I}_i^A).

The spatial velocity is called \hat{v}_i , the spatial acceleration \hat{a}_i and the joint axis \hat{s}_i . All values are given in local body coordinates.

The transformation from the coordinate system of body i to that of body j can be expressed using the spatial transformation ${}_j\hat{X}_i$. The other values are partial results without any physical meaning.

Spatial force combines linear force f and torque r as

$$\hat{f} = \begin{pmatrix} f \\ r \end{pmatrix},$$

spatial accelerations linear acceleration a and rotational acceleration α as

$$\hat{a} = \begin{pmatrix} \alpha \\ a \end{pmatrix},$$

spatial velocities analogous to spatial accelerations and joints s with the direction of the rotation axis r or the direction of translation t , respectively, as

$$\hat{s} = \begin{pmatrix} t \\ s \end{pmatrix}.$$

Spatial transformations are expressed as

$$\hat{X} = \begin{pmatrix} E & 0 \\ r \times E & E \end{pmatrix},$$

where E is the rotation taking a body from one frame to another, and r is the corresponding translation. The rigid-body inertia is expressed as

$$\hat{I} = \begin{pmatrix} H^T & M \\ I & H \end{pmatrix}, \quad (6)$$

with $H = m * s \times$ (s center of gravity, m mass), $I = I^* + s \times m(-s) \times$ (I^* inertia tensor) and $M = m * \bar{I}$ (\bar{I} unit matrix).

The spatial version of the cross operator

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \times = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}$$

is

$$\begin{pmatrix} a \\ a_0 \end{pmatrix} \hat{\times} = \begin{pmatrix} a \times & 0 \\ a_0 \times & a \times \end{pmatrix}$$

Using the spatial transposition for vectors

$$\hat{a}^S = \begin{pmatrix} a \\ a_O \end{pmatrix}^S = (a_O^T \ a^T)$$

the spatial vector scalar product is defined as $\hat{a}^S \hat{b}^S$.

II. DYNAMIC DESCRIPTION OF A SINGLE BODY

The shape of the body from Figure 1 can be described through the four parameters a , b , c and h . The integral:

$$\begin{aligned} \int f(x, y, z) &= \int_{y=-h}^0 \int_{z=z_1(y)}^{z_2(y)} \int_{x=x_1(y,z)}^{x_2(y,z)} f(x, y, z) \\ -x_1(y, z) &= x_2(y, z) = a + \frac{y}{h}(b-a) + \sqrt{\frac{y^2 c^2}{h^2} - z^2} \\ -z_1(y) &= z_2(y) = \frac{yc}{h} \end{aligned}$$

extend over the volume of the body. For $f(x, y, z) = \varrho(y^2 + z^2)$ the component I_{xx} of the inertia tensor is obtained, and for $f(x, y, z) = \varrho(x^2 + z^2)$ and $f(x, y, z) = \varrho(x^2 + y^2)$, the components I_{yy} and I_{zz} , respectively. The other components of the matrix I are zero.

For $f(x, y, z) = \varrho$ the mass is obtained, and for $f(x, y, z) = \varrho * r$ the center of mass scaled by the mass (r is the vector from the origin).

The evaluation of the integrals leads to:

$$I_{xx} = \frac{\varrho ch}{60} (4ac^2 + 16bc^2 + 12ah^2 + 48bh^2 + 3c^3\pi + 12ch^2\pi)$$

$$I_{yy} = \frac{\varrho ch}{30} \begin{pmatrix} 2a^3 + 4a^2b + 6ab^2 + 8b^3 + 6ac^2 + 24bc^2 + \\ a^2c\pi + 3abc\pi + 6b^2c\pi + 3c^3\pi \end{pmatrix}$$

$$I_{zz} = \frac{\varrho ch}{60} \begin{pmatrix} 4a^3 + 8a^2b + 12ab^2 + 16b^3 + 8ac^2 + \\ 32bc^2 + 12ah^2 + 48bh^2 + 2a^2c\pi + 6abc\pi + \\ 12b^2c\pi + 3c^3\pi + 12ch^2\pi \end{pmatrix}$$

$$m * s = \begin{pmatrix} 0 & \frac{-\varrho ch^2(4a+12b+3c\pi)}{12} & 0 \end{pmatrix}$$

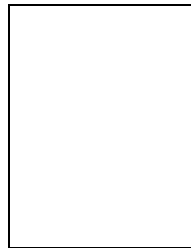
$$m = \frac{\varrho ch(2a + 4b + c\pi)}{3}$$

With $M = m * \bar{I}$ (\bar{I} unit matrix) and $H = m * s \times$, the rigid-body inertia can be formed (according to equation 6).

REFERENCES

- [1] Jane Wilhelms and Brian A. Barsky, "Using dynamic analysis to animate articulated bodies such as humans and robots", *Proceedings of Graphics Interface*, pp. 97-104, 1985.
- [2] William W. Armstrong and Mark W. Green, "The dynamics of articulated rigid bodies for the purposes of animation", *The Visual Computer*, vol. 1, pp. 231-240, 1985.
- [3] William W. Armstrong, Mark Green, and Robert Lake, "Near-real-time control of human figure models", *IEEE Computer Graphics and Applications*, pp. 52-61, 1987.
- [4] Paul M. Isaacs and Michael F. Cohen, "Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics", *SIGGRAPH Proceedings, Computer Graphics*, pp. 215-224, 1987.
- [5] Ronen Barzel and Alan H. Barr, "A modeling system based on dynamic constraints", *SIGGRAPH Proceedings, Computer Graphics*, pp. 179-188, 1988.
- [6] Peter Schröder and David Zeltzer, "The virtual erector set: dynamic simulation with linear recursive constraint propagation", in *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, Snowbird, Utah, March 1990, pp. 23-31.
- [7] C. W. A. M. van Overveld, "An iterative approach to dynamic simulation of 3-d rigid-body motions for real-time interactive computer animation", *The Visual Computer*, vol. 7, pp. 27-38, 1991.
- [8] Andrew Witkin, Michael Gleicher, and William Welch, "Interactive dynamics", in *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, Snowbird, Utah, March 1990, pp. 11-21.
- [9] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms", in *Proceedings of the Joint Automatic Control Conference*, Charlottesville, VA, 1981.
- [10] Jane Wilhelms, "Dynamic experiences", in *Making them move: mechanics, control, and animation of articulated figures*, Norman I. Badler, Brian A. Barsky, and David Zeltzer, Eds. 1991, pp. 265-279, Morgan Kaufmann Publishers, Inc.
- [11] W. W. Armstrong, "Recursive solution to the equations of motion of an n-link manipulator", in *Proceedings of the Fifth World Congress on Theory of Machines and Mechanisms*, 1979, pp. 1343-1346, The American Society of Mechanical Engineers.
- [12] R. Featherstone, "The calculation of robot dynamics using articulated-body inertias", *The International Journal of Robotics Research*, vol. 2, no. 1, pp. 13-30, 1983.

- [13] Richard H. Lathrop, "Constrained (closed-loop) robot simulation by local constraint propagation", in *Robotics and Automation*. IEEE Council on Robotics and Automation, 1986.
- [14] Roy Featherstone, *Robot dynamic algorithms*, Kluwer Academic Publishers, Boston/Dordrecht/Lancaster, 1987.
- [15] C. William Gear, *Numerical initial value problems in ordinary differential equations*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1971.
- [16] Alan C. Hindmarsh, "Odepack, a systemized collection of ode solvers", in *Scientific Computing*, R. Stepleman et al, Ed. 1983, pp. 55–64, IMACS/North-Holland Publishing Company.
- [17] P. E. Tischer and G. K. Gupta, "An evaluation of some new cyclic linear multistep formulas for stiff odes", *ACM Transactions on Mathematical Software*, vol. 11, no. 3, pp. 263–270, 1985.
- [18] Mark Green, "Using dynamics in computer animation: control and solution issues", in *Making them move: mechanics, control, and animation of articulated figures*, Norman I. Badler, Brian A. Barsky, and David Zeltzer, Eds. 1991, pp. 281–314, Morgan Kaufmann Publishers, Inc.
- [19] P. J. Vanderhouven and B. P. Sommeijer, "Analysis of parallel diagonally implicit iteration of runge kutta methods", *Applied Numerical Mathematics*, vol. 11, no. 1–3, pp. 169–188, 1993.
- [20] M. Amin-Javaheri and D. E. Orin, "A systolic architecture for computation of the manipulator inertia matrix", in *Proc. IEEE Int. Conf. Robotics and Automation*, Raleigh, NC, 1987, pp. 647–653.
- [21] A. Fijany and A. K. Bejczy, "A class of parallel algorithms for computation of the manipulator inertia matrix", in *IEEE Int. Conf. Robotics and Automation*, 1989, pp. 1818–1826.
- [22] C. S. G. Lee and P. R. Chang, "Efficient parallel algorithms for robot forward dynamics computation", in *Proc. IEEE Int. Conf. Robotics and Automation*, Raleigh, NC, 1987, pp. 654–659.
- [23] R. E. Strout, J. R. McGraw, and A. C. Hindmarsh, "An examination of the conversion of software to multiprocessors", *Journal of Parallel and Distributed Computing*, , no. 13, pp. 1–16, 1991.
- [24] Rod Deyo, John A. Briggs, and Pete Doenges, "Getting graphics in gear: graphics and dynamics in driving simulation", *SIGGRAPH Proceedings, Computer Graphics*, pp. 317–326, 1988.
- [25] H. T. Kung, "New algorithm and lower bounds for the parallel evaluation of certain rational expressions and recurrence", *J. of Association for Computing Machinery*, vol. 23, no. 2, pp. 251–261, Apr. 1976.
- [26] Thomas Jung, *Entwicklung einer Plattform zur interaktiven physikalisch basierten Animation gelenkig verbundener Systeme*, (GMD-Bericht Nr. 249). R. Oldenbourg, München, Wien, 1995, also: Ph.D. Thesis, Technical University of Berlin.
- [27] W. K. Giloi, *Towards the next generation parallel computers: MANNA and META*, Proc. ZEUS '95, Linköping, Sweden, 1995.
- [28] J. Cordsen and W. Schröder-Preikschat, "Objective peace: A bridge between parallel and distributed operating systems", in *Proceedings of the ERCIM Workshop on Distributed Systems*, Lisboa, Portugal, November 14–15, 1991, pp. 109–112.
- [29] J. Nolte and W. Schröder-Preikschat, "An object-oriented computing surface for distributed memory architectures", in *Software Technology*, Ted Lewis Hesham El-Rewini and Bruce D. Shriver, Eds., Maui, Hawaii, January 5–8 1993, vol. 2 of *Proceedings of the Twenty-Sixth Annual Hawaii International Conference on System Sciences*, pp. 134–143, IEEE Computer Society Press.
- [30] Jörg Nolte, *Duale Objekte – Ein Modell zur objektorientierten Konstruktion von Programmfamilien für massiv parallele Systeme*, (GMD-Bericht Nr. 232). R. Oldenbourg, München, Wien, 1994, also: Ph.D. Thesis, Technical University of Berlin.



Thomas Jung was born in Berlin, Germany, on April 16, 1964. He received his M.S. and Ph.D. degrees in computer science from the Technical University of Berlin in 1989 and 1995, respectively. Since 1987, he has been with the GMD-FIRST. His current research interests include distributed algorithm design, physically based animation, and virtual environments.