Linear-Time Dynamics using Lagrange Multipliers

PHILIP HUANG, University of Toronto, Canada
HANNA JIAMEI ZHANG, University of Toronto, Canada

This work presents an implementation of David Baraff's 1996 work on Linear-Time Dynamics using Lagrange Multipliers. The implementation was done in MATLAB and results are shown on 2D serially jointed structures, trees, and closed-loop chain structures. The dynamics of these systems are handled by 3 different methods 1) standard matrix inversion, 2) a sparse $O(n)$ factorization method, and a 3) dense $O(n^3)$ factorization method. Comparisons of the performance of these different solvers are tested on these systems at varying scales (i.e. size of the structure) are made.

ACM Reference Format:
Philip Huang and Hanna Jiamei Zhang. 2021. Linear-Time Dynamics using Lagrange Multipliers. *ACM Trans. Graph.* 1, 1 (December 2021), 6 pages. https://doi.org/1234567.1234567

1 INTRODUCTION

The simulation of dynamic systems with two or more rigid bodies is a core problem in computer graphics and physics engines with a variety of applications in animation, gaming and robotics. For example, a humanoid robot can be simulated as a set of bodies connected by different types of joints. There are two ways to formulate a multibody systems using dynamics principles. Either we can model constraints in *reduced-coordinates* by removing degrees of freedom (DoF) from the system, or introduce additional forces in the *maximal* coordinates to satisfy the constraints. Lagrange multipliers fall in the latter categorization where the system state is expressed using a simpler set of m maximal coordinates and constraints are enforced by *constraint force.*

In this project, we try to re-implement the linear-time sparse dynamics solver first proposed in [Baraff 1996], which is based on the Lagrange multipliers approaches. While previous works have achieved direct, linear-time solutions for a serial chain (a sequence of links) with recursive articulated-body method [Featherstone and Orin 2000], Baraff's solver works for any acyclic set of primary constraints (i.e. trees). For example, for a set of n bodies with $n-1$ constraints, the method takes $O(n)$ time to compute the constraint forces in the dynamics equation. Furthermore, it is also easy to incorporate auxiliary constraints, such as loop closures, collisions, and fixed joints, into the algorithmic framework. Typically, the number of auxiliary constraintd (denoted as k) is small compared to n, so the added cost to formulate and solve the system is simply $O(nk+k^3) = O(nk)$, which remains efficient.

Authors' addresses: Philip Huang, , University of Toronto, Toronto, Canada; Hanna Jiamei Zhang, , University of Toronto, Toronto, Canada.

2 CONSTRAINT-BASED MECHANICS WITH LAGRANGE MULTIPLIERS

2.1 Primary Constraints

The dimension of the bodies used in our implementation is $dim(i) = 3$, i.e. 2 DoFs for translation in 2D and 1 DoF for rotation. The i^{th} body's velocity is expressed as a vector $\dot{\mathbf{q}} \in \mathbb{R}^3$, the force acting on the i^{th} body is $\mathbf{F}_i \in \mathbb{R}^3$, and the acceleration on the i^{th} body is $\ddot{\mathbf{q}} \in \mathbb{R}^3$. The following relationship holds for each body: $\mathbf{M}_i\ddot{\mathbf{q}}_i = \mathbf{F}_i$

Where $\mathbf{M}_i$ is a 3×3 symmetric positive definite matrix which describes the mass properties of body i. In this work only rectangular bodies are studied for which the moment of inertia is $\dfrac{m(w^2+h^2)}{12}$ with m, w, h being the mass, width, and height of the 2D rectangle body. For this work $\mathbf{M}_i$ is not time-varying as the rectangular blocks are considered as rigid and time-invariant structures. The entire n-body system is expressed as follows,

$$\dot{\mathbf{q}} = \begin{bmatrix}\dot{\mathbf{q}}_1 & \dot{\mathbf{q}}_2 & \dots & \dot{\mathbf{q}}_n\end{bmatrix}, \mathbf{F} = \begin{bmatrix}\mathbf{F}_1 & \mathbf{F}_2 & \dots & \mathbf{F}_n\end{bmatrix},$$
$$\mathbf{M} = \text{diag}(\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_n)$$

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{F} \tag{1}$$

where diag$(\cdot)$ places inputs along the diagonals of a zero matrix, forming a block diagonal matrix. The dimension of a constraint is the number of DoF's the constraint removes from the system. Consider a serial chain of blocks with n bodies. Each constraint between adjacent blocks removes 2 DoFs from the system (i.e. the x,y position of the joint connecting the next block is determined by the pose of the previous block, leaving only orientation of the joint as a remaining free DoF). Suppose we can describe the i^{th} constraint with an equation $C_i(\mathbf{q}) = 0$. If we differentiate this with respect to time, we arrive at the velocity constraint $\dot{C}_i(\mathbf{q},\mathbf{v}) = \mathbf{j}_{i1}\mathbf{v}_1 + \cdots + \mathbf{j}_{in}\mathbf{v}_n = 0$. To solve for the constraint forces, we need one more time derivative to arrive at the acceleration constraint $\ddot{C}_i(\mathbf{q},\mathbf{v},\dot{\mathbf{v}}) = \mathbf{j}_{i1}\dot{\mathbf{v}}_1 + \partial_t(\dot{C}_i(\mathbf{q}_{i1}))\mathbf{v}_1 + \cdots + \mathbf{j}_{in}\dot{\mathbf{v}}_n + \partial_t(\dot{C}_i(\mathbf{q}_{in}))\mathbf{v}_n$. Merging the known terms together leaves the form: $\mathbf{j}_{i1}\dot{\mathbf{v}}_1 + \cdots + \mathbf{j}_{ik}\dot{\mathbf{v}}_k + \cdots + \mathbf{j}_{in}\dot{\mathbf{v}}_n + \hat{\mathbf{c}}_i = \mathbf{0}$. In our 2-D case, $\mathbf{j}_{ik}$ is a matrix in $\mathbb{R}^{2\times 3}$ and $\hat{\mathbf{c}}_i$ is a column vector in $\mathbb{R}^{2\times 1}$.

However, since the constraint $\ddot{C}_i(\mathbf{q},\mathbf{v},\dot{\mathbf{v}})$ is applied in the acceleration space, numerical errors in integration could lead to drifts in the velocity constraint. $\dot{C}_i(\mathbf{q},\mathbf{v})$ and the position constraint $C_i(\mathbf{q})$. One way to rectify this is to apply a spring-like feedback force $-k_qC_i(\mathbf{q}) - k_v\dot{C}_i(\mathbf{q},\mathbf{v})$ and solve for $\ddot{C}_i(\mathbf{q},\mathbf{v},\dot{\mathbf{v}}) = -k_qC_i(\mathbf{q}) - k_v\dot{C}_i(\mathbf{q},\mathbf{v})$ [Witkin 1997]. This is straight forward to implement since we can combine the feedback with the constant term $\hat{\mathbf{c}}_i$ and solve for:

$$\mathbf{j}_{i1}\dot{\mathbf{v}}_1 + \cdots + \mathbf{j}_{in}\dot{\mathbf{v}}_n + \mathbf{c}_i = \mathbf{0}$$

In this formulation primary constraints are expressed through the body accelerations via a linear condition and affect only a *pair* of bodies. To enforce acceleration conditions on the constraints, the workless constraint force is applied in the orthogonal direction of the velocity and takes the form:

$$\mathbf{F}_c^i = \begin{bmatrix} \mathbf{j}_{i1}^t & \dots & \mathbf{j}_{i1}^t \end{bmatrix} \lambda_i \tag{2}$$

where $\lambda_i \in \mathbb{R}^2$ is the lagrange multiplier of the i^{th} constraint. To express r constraints the following acceleration conditions must hold:

$$\mathbf{J}\dot{\mathbf{v}} + \mathbf{c} = \mathbf{0} \tag{3}$$

where

$$\mathbf{J} = \begin{bmatrix} \mathbf{j}_{11} & \mathbf{j}_{12} & \cdots & \mathbf{j}_{1n} \\ \vdots & \vdots & & \vdots \\ \mathbf{j}_{r1} & \mathbf{j}_{r2} & \cdots & \mathbf{j}_{rn} \end{bmatrix}, \mathbf{c} = \begin{bmatrix} \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_r \end{bmatrix}, \lambda = \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \end{bmatrix}$$

It follows that the constraint forces have the form $\mathbf{J}^T\lambda$ which can be determined by finding λ such that the constraint forces and external forces together produce motions that satisfy the constraints (i.e. Equation 3).

2.2 Solving dynamics system with $JM^{-1}J^T$

This is a simple method to solve for the λ's used to get the constraint forces. Given an unknown constraint force $\mathbf{J}^T\lambda$ and known net external force $\mathbf{F}_{\text{ext}}$ the following holds, $\mathbf{M}\dot{\mathbf{v}} = \mathbf{J}^T\lambda + \mathbf{F}_{\text{ext}}$ and can be rearranged to get

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}\mathbf{J}^T\lambda + \mathbf{M}^{-1}\mathbf{F}_{\text{ext}} \tag{4}$$

Substituting Eq. 4 into the constraints (Ex. 3) yields the following for which λ can be solved for using matrix inversion:

$$\begin{aligned} &\mathbf{J}(\mathbf{M}^{-1}\mathbf{J}^T\lambda + \mathbf{M}^{-1}\mathbf{F}_{\text{ext}}) + \mathbf{c} = \mathbf{0} \\ &\mathbf{A}\lambda = \mathbf{b} \\ &\mathbf{A} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T \quad \mathbf{b} = -(\mathbf{J}\mathbf{M}^{-1}\mathbf{F}_{\text{ext}} + \mathbf{c}) \end{aligned}$$

3 ALTERNATIVE SPARSE FORMULATION

The matrix $\mathbf{A}$ above is not sparse if the system has a branching instead of serial structure, without sparsity it is costly to invert $\mathbf{A}$ to solve. An alternative formulation considers the following,

$$\begin{pmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ -\mathbf{c} \end{pmatrix}$$

$$\mathbf{H} = \begin{pmatrix} \mathbf{M} & -\mathbf{J}^T \\ \mathbf{J} & \mathbf{0} \end{pmatrix}$$

Solving this would yield the same constraint forces as the method in Sec. 2.2 above. The benefit of this formulation is that $\mathbf{H}$ is always sparse. We can factor $\mathbf{H}$ as $\mathbf{H} = \mathbf{LDL}^T$, where $\mathbf{L}$ is a lower-triangular block matrix with all ones on the diagonal, and $\mathbf{D}$ is a block-diagonal matrix. To do this, the first step is to re-order $\mathbf{H}$ so that $\mathbf{L}$ will be *just as sparse as* $\mathbf{H}$. A simple way to do this is to use depth first search.

3.1 Ordering H

First, we describe the data structure used to store the tree-like multi-body system. We use a node to store the relevant data for every body/constraint. Each body node has an id, the mass matrix, a parent, and a list of children. The parent of a body node is either empty if the body is the root node, otherwise is the id of the parent constraint. The children of a body node is the list of the ids of all the child constraints. Each constraint node also has an id, the Jacobian matrix, a parent and a single child. The parent and child of a constraint node is always the id of the parent/child body connected by the constraint.

Starting from the root node, we use a depth-first traversal to sort every node in the tree into a list. The procedure is shown the appendix of [Baraff 1996]. Once we have a forward traversal order (ending with the root node last), we can reorder the non-zeros blocks $\mathbf{H}_{ij}$ according to that order. For every new row i in $\mathbf{H}$, we fill the block $\mathbf{H}_{ij}$ with the corresponding Jacobian matrix if j is a child or parent node of i. For every row corresponding to a body node, we also place the corresponding mass matrix on a diagonal block $\mathbf{H}_{ii}$.

3.2 An $O(n^3)$ Factorization Method

We can treat $\mathbf{H}$ as a dense matrix and try to solve the system $\mathbf{LDL}^T\mathbf{x} = \mathbf{b}$ as follow. First, we overwrite the upper triangular portion of $\mathbf{H}$ with $\mathbf{L}^T$, then save the entries of $\mathbf{D}$ onto the diagonal of $\mathbf{H}$. This part requires $O(n^3)$ time. Then, we can solve $\mathbf{Lx}^{(1)} = \mathbf{b}$, followed by $\mathbf{Dx}^{(2)} = \mathbf{x}^{(1)}$, and finally $\mathbf{L}^T\mathbf{x} = \mathbf{x}^{(2)}$. All of these can be done in $O(n^2)$ time. The detailed pseudocode is shown in Section 7.2 of [Baraff 1996].

3.3 An $O(n)$ Factorization Method

In our setup, $\mathbf{H}$ is sparse and we can simplify the calculations. Note that a block $\mathbf{H}_{ij}$ is nonzero only if $i \in \text{child}(j)$ or $j \in \text{child}(i)$. After ordering the matrix, we note that in each row, only one nonzero block occurs to the right of the diagonal. This is because every node has at most one parent. This makes it much easier to run the factoring and solving procedures in the previous section. In fact, both the factorization and the solving procedure reduce to $O(n)$ time. The pseudocode is also shown in Section 7.3 of [Baraff 1996].

After computing $\mathbf{x}$, we extract the appropriate elements that form λ, and then compute $\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{J}^T\lambda + \mathbf{F}^{ext})$.

4 AUXILIARY CONSTRAINTS

Sec. 2.2 and 3 describe how to compute λ (and thus the primary constraint force in response to an external force) for primary constraints, i.e. between two bodies. Constraints such as loop closures, contacts, and constraints to fixed points in space cannot be formulated as primary constraints. They fall under something called *auxiliary constraints.*

4.1 Constraint Anticipation

When computing the Lagrangian multipliers for the auxiliary constraints, we will also anticipate the response of the primary constraints due to the auxiliary constraint forces. Once we have computed the auxiliary constraint forces, we go back and compute the primary constraints without worrying about the effect of auxiliary constraints since we have already anticipated the effects.

First, let the following equation represent the auxiliary constraint forces (in acceleration space).

$$\mathbf{a} = \begin{pmatrix} a_i \\ \vdots \\ a_k \end{pmatrix} = \mathbf{J}^a\dot{\mathbf{v}} + \mathbf{c}^a \tag{5}$$

Let the constraint force due to the i^{th} constraint to have the form

$$\mathbf{k}_i \mu$$

where $\mathbf{k}_i$ is a column vector as the same dimension as $\mathbf{v}$. We define $\mathbf{K}$ as the k-column matrix

$$\mathbf{K} = \begin{pmatrix} \mathbf{k}_1 & \mathbf{k}_2 & \dots & \mathbf{k}_k \end{pmatrix}$$

For loop closures and fixed points, the matrix $\mathbf{K}$ is simply $(\mathbf{J}^a)^T$. The external force seen by the auxiliary constraint is the sum of the primary response and the original external force. We define that as

$$\hat{\mathbf{F}}_{\text{ext}} = \mathbf{J}^T \lambda + \mathbf{F}_{\text{ext}} \tag{6}$$

This allows us to calculate the acceleration without the auxiliary constraint $\dot{\mathbf{v}}_{\text{aux}} = \mathbf{M}^{-1}\hat{\mathbf{F}}_{\text{ext}}$. Now we can calculate the auxiliary constraint forces. We make use of the anticipated response matrix $\hat{\mathbf{M}}^{-1}$, which allows us to compute the system's acceleration

$$\dot{\mathbf{v}} = \hat{\mathbf{M}}^{-1}\mathbf{K}\mu + \dot{\mathbf{v}}_{\text{aux}} \tag{7}$$

Substituting this to the auxiliary constraint equation, we can solve for the multipliers μ

$$\mathbf{a} = \mathbf{J}^a\dot{\mathbf{v}} - \mathbf{c}^a = \mathbf{J}^a\hat{\mathbf{M}}^{-1}\mathbf{K}\mu + (\mathbf{J}^a\dot{\mathbf{v}}_{\text{aux}} + \mathbf{c}^a) \tag{8}$$

It is hard to compute $\hat{\mathbf{M}}^{-1}$ directly, but we can compute $\hat{\mathbf{M}}^{-1}\mathbf{K}$ column by column. Given the direction of the i^{th} auxiliary constraint force $\mathbf{k}_i$, we know the anticipated response is $\hat{\mathbf{M}}^{-1}\mathbf{k}_i$. This consists of two parts - the system's response to a force in the direction $\mathbf{k}_i$, and the primary constraints' response to $\mathbf{k}_i$. Given a primary response $\mathbf{F}_{\text{resp}} = \mathbf{J}^T\lambda$ where $\mathbf{A}\lambda = -\mathbf{J}\mathbf{M}\mathbf{k}_i$, we can write $\hat{\mathbf{M}}^{-1}\mathbf{k}_i$ as

$$\hat{\mathbf{M}}^{-1}\mathbf{k}_i = \mathbf{M}^{-1}(\mathbf{F}_{\text{resp}} + \mathbf{k}_i) \tag{9}$$

We can then express $\hat{\mathbf{M}}^{-1}\mathbf{K}$ as $\left[\hat{\mathbf{M}}^{-1}\mathbf{k}_1\ \hat{\mathbf{M}}^{-1}\mathbf{k}_2 \dots \hat{\mathbf{M}}^{-1}\mathbf{k}_k\right]$. The cost to compute each column $\hat{\mathbf{M}}^{-1}\mathbf{k}_i$ is mainly from solving $\mathbf{F}_{\text{resp}}$, which is $O(n)$. Since $\mathbf{K}$ has k columns, the total cost is $O(kn)$.

4.2 Computing the Net Constraint Force

Once we can express $\hat{\mathbf{M}}^{-1}\mathbf{K}$, we can compute multipliers μ by solving equation 8 with a standard linear system solver. With the auxiliary constraint force $\mathbf{K}\mu$, we can compute the primary constraint force to the combined external and auxiliary constraint force $\mathbf{F}_{\text{ext}} + \mathbf{K}\mu$. That is just solving $\mathbf{A}\lambda = -(\mathbf{J}\mathbf{M}^{-1}(\mathbf{K}\mu + \mathbf{F}_{\text{ext}}) + \mathbf{b})$

The final constraint force due to both the primary constraint and auxiliary constraints is just $\mathbf{K}\mu + \mathbf{J}^T\lambda$.

5 RESULTS

5.1 Test Environment

The three different Lagrange multiplier-based solvers for system dynamics described above are implemented in MATLAB and executed using an Intel Core i7-7500U processor with 2 cores at 2.90 GHz. The test environment consists of uniform rectangular bodies with width 1 unit, height 5 units with joints located 2 units from the COM in the length-wise direction and uniformly distributed mass of 1 unit. The local body frame is chosen to be centered at the COM with x-axis aligned width-wise and y-axis aligned length-wise. The joint in the positive y direction of the local frame will be referred to as the anterior joint and the other the posterior joint. These bodies were arranged into serial arms and binary trees. The three different solvers were used on these two cases to evaluate and compare performance quantitatively with computation time. Simulations results presented below used a 0.01 second timestep for a 5 second time period.

The primary constraints for serial arm structures are expressed using constraints between successive joints, i.e. anterior joint connected with posterior joint of the succeeding structure. The primary constraints for the binary tree structure are expressed using BFS tree traversal and indexing of parent and children in trees to connect appropriate anterior and posterior joints of the bodies that make up the tree.

An auxiliary constraint is used to fix the posterior joint of the root body to a fixed point in space for serial arm and binary tree structures. The location of the anterior joint of the last body is constrained to be the same as the location of the posterior joint of the root body in order to create a free floating closed-loop chain.

5.2 Implementation

The naive approach detailed in Sec. 2.2 was implemented using MATLAB's native *mldivide* function. We implement the alternative sparse factorization methods as described in 3 with MATLAB functions and data structures following the pseudo-code from Baraff's paper [Baraff 1996]. The naive approach from Sec. 2.2 will be referred to as (1) A/b, and the factorization approaches from Sec. 3 will be referred to as (2) Sparse, and (3) Dense in the following sections.

The qualitative metric used to evaluate the performance of the algorithms was computation time. This was measured using the MATLAB *tic toc* function. The values recorded were average time to compute the λ's for only primary constraints. Only the lines of code relevant to system solving were included in this measurements of complexity. When considering auxiliary constraints, the first step is still computing the primary constraints λ's for constraint anticipation. Following that, there is simply more λ's to compute according to Section 4. Thus, time complexity results for the three solving methods are thus shown for primary constraints to compare the performance of the algorithms alone.

For serial arm structures a valid initialization is given to the structure such that all primary constraints begin fulfilled. For the case of the binary tree structure, no such initialization was given. All blocks began in the same initial zero position, so primary constraints started out unfilled. Given an appropriately small time scaling (0.01s) the solvers was able to compute appropriate feedback forces to arrange the blocks from initial zero configurations into the binary tree structure with a few hundred iterations. Similarly, for the closed loop structure, primary and auxiliary constraints were not fulfilled at startup. The initial block positions were set at position (0,0) with relative orientation between successive blocks set at $-\frac{3\pi}{4}$. This was done so that the system was close to fulfilling primary and auxiliary constraints at initialization to avoid very large constraint forces in the first few iterations of the simulation.

The time performance of the three solving methods is evaluated with only the primary constraint calculation, but can be measured

n	(1) A\b	(2) Sparse	(3) Dense
2	2.23E-05	3.89E-04	3.94E-04
5	1.28E-04	6.52E-04	1.20E-03
10	3.06E-04	1.00E-03	4.00E-03
20	9.34E-04	1.90E-03	1.97E-02
50	2.30E-03	4.10E-03	2.38E-01
100	8.50E-03	8.00E-03	1.67E+00
200	6.47E-02	1.90E-02	1.35E+01

Table 1. Algorithm performance on serial bar structures. Quantities presented are in units of seconds per timestep (i.e. the average time to solve for λ). Table quantities were obtained using a value of $dt = 0.01$ for the timestep over 5 second time period.

n	(1) A\b	(2) Sparse	(3) Dense
3	1.35E-05	3.98E-04	5.27E-04
7	1.35E-05	9.15E-04	2.60E-03
15	5.13E-04	2.00E-03	1.05E-02
31	1.40E-03	4.20E-03	5.55E-02
63	4.60E-03	5.40E-03	3.99E-01
127	2.27E-02	1.05E-02	3.67E+00
255	2.44E-01	3.82E-02	3.97E+01

Table 2. Algorithm performance on binary tree structures. Quantities presented are in units of seconds per timestep (i.e. the average time to solve for λ). Table quantities were obtained using a value of $dt = 0.01$ for the timestep over 5 second time period.

in a similar manner for auxiliary constraints (see code implementation).

The Jacobians of all systems are implemented using MATLAB's symbolic math tool box and symbolic derivative functions. To update these variables with numerical quantities these functions are converted with the *matlabFunction* function. Forward Euler is used to step the systems forward in time by dt each time step forward.

5.3 Rigid Body Systems: Primary Constraints Only

The runtime of the computation of the λ's for the primary contraint forces was evaluated for serial bar linkages and binary trees of varying sizes. Results are detailed in the below subsections.

5.3.1 Serial Bar Linkages. The left image in Figure 4 shows a visualization of the serial bar linkages considering only primary constraints. Table 1 summarizes the runtime data of each solving technique on the serial bar linkages structure. Figure 1 is a graph of the performance of each solver and relevant fitting data.

5.3.2 Binary Tree Structure. The top image in Figure 5 shows a visualization of the binary tree structure considering only primary constraints. Table 2 summarizes the runtime data of each solving technique on the binary tree structure. Figure 2 is a graph of the performance of each solver and relevant fitting data.

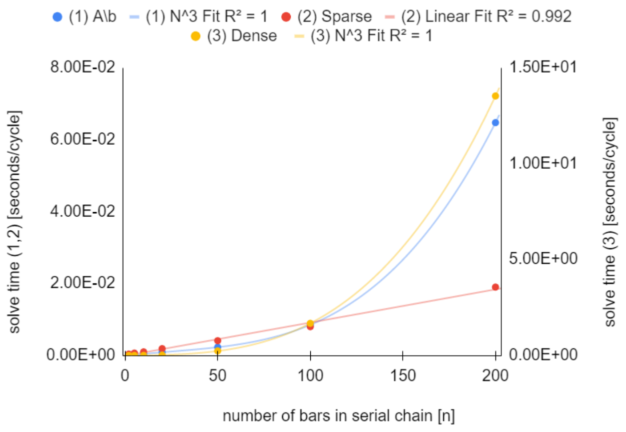

Fig. 1. Plot of data from Table 1 with 3rd order polynomial trendlines fitted to the data obtained from (1) A\b(blue) and (3) Dense (yellow) solving methods and a linear trendline fitted to the data obtained from the (2) Sparse (red) solving methods. R^2 values for each of these curve fits are noted in the legend at the top. Note that the data for (3) Dense is plotted on the right vertical axis as the time complexity is way higher than the other two methods which are plotted on the left vertical axis.

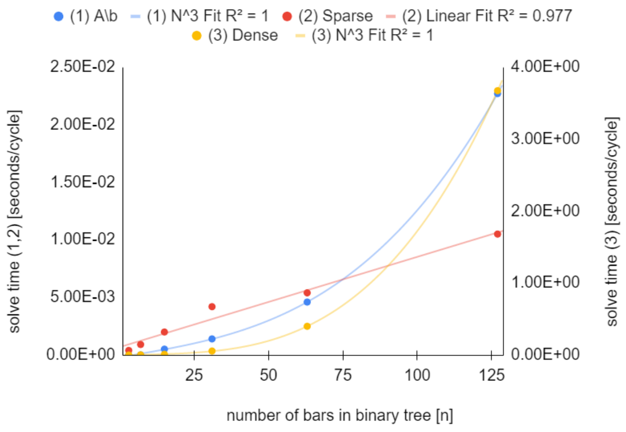

Fig. 2. Plot of data from Table 2 with 3rd order polynomial trendlines fitted to the data obtained from (1) A\b(blue) and (3) Dense (yellow) solving methods and a linear trendline fitted to the data obtained from the (2) Sparse (red) solving methods. R^2 values for each of these curve fits are noted in the legend at the top. Note that the data for (3) Dense is plotted on the right vertical axis as the time complexity is way higher than the other two methods which are plotted on the left vertical axis.

5.4 Rigid Body Systems: Primary and Secondary (Auxiliary) Constraints

To evaluate the performance of our implementation of auxiliary constraints the closed-loop four-bar linkage, fixed base serial bar linkages, and fixed base binary tree. Results for this section are shown through qualitative results.

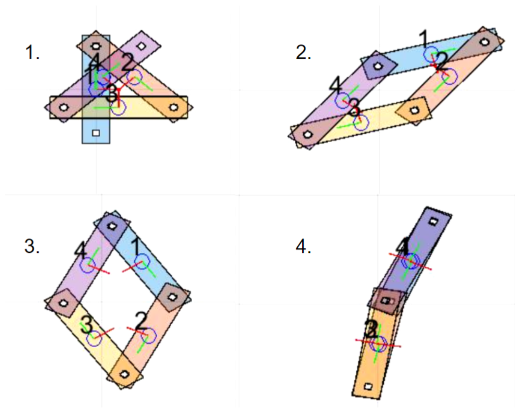

Fig. 3. Visualization of the $n = 4$ four-bar linkage structure using $dt = 0.01$ over 5 second time period. Structure states are shown overlaid over one another for equally spaced timepoints over the total simulation time period. The figure on the left is without any auxiliary constraints, that on the right is with one auxiliary constraint fixing the base body's posterior joint.

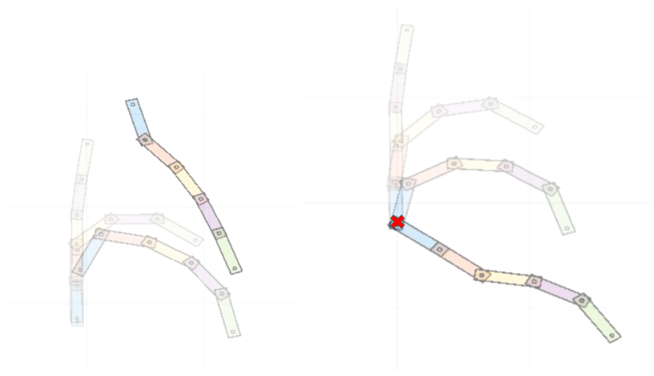

Fig. 4. Visualization of the $n = 5$ serial-arm structure using $dt = 0.01$ over 5 second time period. Structure states are shown overlaid over one another for equally spaced time points over the total simulation time period. The figure on the left is without any auxiliary constraints, that on the right is with one auxiliary constraint fixing the base body's posterior joint.

5.4.1 Closed-Loop Four-bar Linkage. Figure 3 shows the performance of method (1) A/b on a closed-loop four-bar linkage. Any solving method could have been used to solve for primary and auxiliary constraints.

5.4.2 Fixed Base Serial Linkages. Figure 4 shows the performance of method (1) A/b on a closed-loop four-bar linkage with and without an auxiliary constraint constraining the motion of the base joint. Any solving method could have been used to solve for primary and auxiliary constraints.

5.4.3 Fixed Base Binary Tree. Figure 5 shows the performance of method (1) A/b on a closed-loop four-bar linkage with and without an auxiliary constraint constraining the motion of the base joint. Any solving method could have been used to solve for primary and auxiliary constraints.

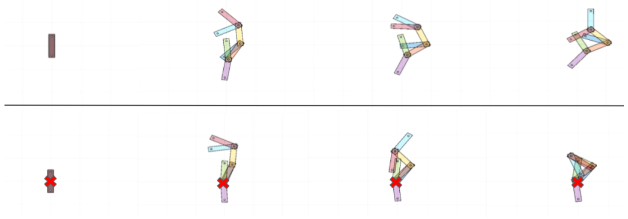

Fig. 5. Visualization of the $h = 3, n = 7$ binary tree structure using $dt = 0.01$ over 5 second time period. Structure states are shown overlaid over one another for equally spaced time points over the total simulation time period. The figure on the top is without any auxiliary constraints, that on the bottom is with one auxiliary constraint fixing the base body's posterior joint.

6 DISCUSSION

The three solving methods performed as expected on both serial bar linkages and binary tree structures. As per Figures 1 and 2, as n, the number of bars to be simulated increases the time complexity in all cases. With respect to our implementation of method (1) A/b the 3rd order polynomial fit to the data in both test cases yields an R^2 value of 1, meaning that this is a perfect fit. This verifies that method (1) A/b is indeed $O(n^3)$. Similarly, it is clear that our implementation of method (3) Dense is also verified to be $O(n^3)$ with R^2 values of 1 for both test cases. For method (2) Sparse, the data is fit to a linear trendline with an R^2 value of 0.992 and 0.977 for serial bar linkages and binary tree structures respectively. This indicates that the data is very reasonably following a linear trend and it follows that this method is indeed a linear algorithm capable of $O(n)$ or linear time performance.

Although method (1) A/b and method (3) Dense are $O(n^3)$ algorithms, method (1) A/b performs multiple orders of magnitude faster than method (3) Dense. It performs so well that it is on the same order of magnitude as the (2) Sparse method $O(n)$ and actually outperforms it for up to approximately 100 bars the serial bar linage tests and up to approximately 75 bars in a binary tree. This is likely due to the optimizations MATLAB does for their *mldivide* function. This is consistent with our understanding of the (2) Sparse method having superior performance compared to the naive (1) A/b method on trees and branching structures.

As to Figure 2, the 4^{th} data point for the binary tree $h = 5, n = 15$ using method (2) Sparse looks quite off the linear trendline. This is likely an outlier due to experimental error during data collection (i.e. processing power being diverted for some other parallel computing task).

It was noted that a major bottleneck of this implementation was not the algorithm itself but the method we chose to build the system Jacobians. The MATLAB symbolic toolbox, symbolic derivatives, and *matlabFunction* function used to compute our system Jacobians doesn't handle the computation of higher-order system derivatives efficiently and thus performed very poorly (on the order of hours) when there were more than 50 bodies in the simulation. Once the system Jacobians were computed, the Euler timestepping took time that scales with the results presented in Tables 1 and 2.

7 CONCLUSIONS AND NEXT STEPS

The main contribution of Baraff's 1996 paper was for a method that enables computation of system dynamics in linear time compared to 3rd order polynomial time of other state of the art methods. In our project we confirm this performance in a simple modular MATLAB implementation of his work and compare it to other less efficient methods for solving the same problem. Further work with this could involve trying different timestepping methods, experimenting with different structures or auxiliary constraints (e.g. friction and contacts), extending to simulate 3D objects, and optimizing the Jacobian computations that were a bottleneck.

REFERENCES

David Baraff. 1996. Linear-time dynamics using lagrange multipliers. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 137–146.

Roy Featherstone and David Orin. 2000. Robot dynamics: equations and algorithms. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, Vol. 1. IEEE, 826–834.

Andrew Witkin. 1997. An introduction to physically based modeling: Constrained dynamics. *Robotics Institute* (1997).