# The Sub-stepping of Havok's Physics Solver

A Description and a Preliminary Analysis

Jad Nohra

## I Introduction

The main purpose of this report is to fill a gap in the internal documentation of ***Havok Physics***. As it stands, there is no formal reference for the mathematics used throughout the constraint solver. One is limited to constantly reverse engineering the code, a practice that is uneconomical, tedious[1] and error-prone for the seasoned engineer while bordering on the impossible for the newcomer. This makes it hard for the interested to classify the implementation in terms of publicly known methods, to compare its details to new research results, or to confidently extend the code in experiments. On a more practical level, due to source code terminology and variable naming having evolved organically with time, making mistakes is around the corner during code modification and this report also helps with that.

## II Notation and Reference

We shall be lax with definition and notation to keep things concise. For example, we shall use matrices and vectors without specifying their spaces or the dimensions thereof. We shall also pull formulas out of thin air and assume the '$\mathrm{J}\mathrm{M}^{-1}\mathrm{J}^{\mathrm{T}}$ approach'[2], which we shall call in short the $\Gamma$ approach, throughout the report. We refer the reader to Shabana's ***Computational Dynamics*** and to Erleben's PhD thesis for formula derivations and clean notation. Having said that, we shall be overly pedantic with notation relating to properties that are commonly overloaded and abused. The motivation to do so is the desired disambiguating nature of this report, which dictates that no two quantities that are not identical should look the same, even within different algorithms. This may make this section feel long-winded but the benefits will become clear as we constantly straddle the line between theory and implementation without falling into ambiguity.

### II.1 Generalities

We shall not decorate vectors to distinguish them from scalars as this is always clear from the context and we shall use lower case letters for both, keeping upper case letters for matrices and calligraphic letters for quite special, usually abstract structures. We shall use both parentheses and brackets mostly for grouping and almost never to enclose parameters of functions because we will have little use for that. We shall multiply linear algebraic quantities without using any multiplication symbol in between, reserving the dot symbol for the vectorial dot product, e.g, $a = b \cdot c$. We shall use the dagger$^{\dagger}$ superscript as shorthand for 'depending on the context' and the double dagger$^{\ddagger}$ to indicate that something is beyond the scope of this document, but potentially the topic of a future report. Finally, we shall strive to keep all variables used in an algorithm immutable so that one does not have to track how they change across the lines of it. Change in time is not considered as mutation and the upper right symbol corner (see below) is used to denote it.

### II.2 Scope $(\grave{x})$ and Dependence $(\underline{x}_{(y)})$

Sometimes, for the sake of clarity, we shall mark a symbol explicitly as being local to its scope within an algorithm. For this, we shall decorate it with a grave accent, e.g, $\grave{x}$, with the exception of trivial loop counters which we leave undecorated. Similarly, when we will find it important to highlight the dependence – in whatever context dependent manner – of a symbol on another, we shall write $\underline{x}_{(y)}$ to say $x$ depends on $y$.

[1] At least due to the high level of low-level optimization.

[2] Baraff, "Linear-Time Dynamics using Lagrange Multipliers."

## II.3 Fixed Symbols

For convenience, we shall fix some symbols to have the meaning assigned to them below, and never anything else.

***Simulation***

$t$ : The time.
$h$ : The time step.
$n$ : The number of sub-steps viz. subdivisions of $h$.

***Constrained Rigid Body System***

$q$ : The coordinate vector[3].
$v$ : The coordinate velocity vector.
$J$ : The constraint Jacobian matrix.
$\beta$ : The constraint right-hand-side (bias) vector.
$\tau, \kappa$ : Two global ad-hok stabilization factors.
$\mathcal{C}$ : The MLCP problem used to solve the system's Lagrange multiplier vector $\lambda$. $\mathcal{C}$ is:
$\underbrace{(JM^{-1}J^T)}_{\Gamma}\lambda = \beta - Jv : \complement^{4}\underset{\smile}{\lambda}^{5}$
or[6]: $J(v + M^{-1}J^T\lambda) = \beta : \complement\underset{\smile}{\lambda}$.

## II.4 Symbol Corners $\left({}^{d}_{c}x^{a}_{b}\right)$

We shall often use up all four corners around symbols, each corner carrying a fixed interpretation. Illustrating using ${}^{d}_{c}A^{a}_{b}$, we have

- The corner '$a$' holds time be it continuous or discrete. for example, $A^{t+h}$ is $A$ at time $(t+h)$. When this corner is not specified, $A$ is either† constant or assumed to be $A^t$. Additionally, $A'$ is shorthand for $A^{t+h}$, where $h$ is the simulation's time step. Finally, we shall use another shorthand for discrete time fractions: when it is not ambiguous, instead of writing $A^{t+i\frac{h}{n}}$, we shall simply use $A^i$.
- The corner '$b$' holds the choice of components. for example $x_i$ is the ith component of $x$. When $x$ is not a vector, the meaning should be clear from the context. $A_{i,j}$ for a matrix $A$ refers to row $i$ and column $j$. Only one component may be specified when it is clear whether it refers to a row or a column. We shall not be strict about components. for example, $v$ being velocity, $v_x$ when $x$ is a constraint obviously means, by indirection, the velocities of the bodies associated with $x$.
- The corner '$c$' holds additional partial specification. We shall use it mainly along with tanslational ($t$) and rotational ($r$) components. for example, $v$ being velocity, ${}_{r}v$ is angular (rotational) velocity.
- The corner '$d$' holds the frame in which coordinates are represented. for example, $x$ being a vector, ${}^{i}x$ is $x$ in the frame $i$†. We shall use $\mathcal{E}$ to denote the conventional absolute frame of reference, and when the corner of $x$ is left empty, then the quantity is either† not spatial, or it is assumed that $x$ means ${}^{\mathcal{E}}x$. For a matrix (e.g $A$) we shall write ${}^{y\curlywedge z}A$ meaning that the domain is represented in $y$ and the codomain in $z$. Vector and matrix multiplication can then be notationally checked for consistency in the following manner: ${}^{z}v = ({}^{y\curlywedge z}B)({}^{x\curlywedge y}A)[{}^{x}u]$.

## II.5 Rotation $(R)$

We shall use matrix, rotation (unit) quaternion, and rotation vector parametrizations of rotation. For rotation matrices, we shall reserve the symbol $R$. We shall adopt a shorthand for the profuse usage of the rotational part of body frames such that having a body $i$ we write $R_i$ meaning ${}^{i\curlywedge\mathcal{E}}R$, which is the conventional rotational part of the affine 'body transform'. For the other parameterizations, it will be clear from the context which is which; quaternions will mostly be used for body coordinates and their manipulation, while rotation vectors will stem from angular velocities during integration. For additional clarity and to emphasize that the multiplication order of rotation quaternions, unlike for matrices, should be 'read' from left to right[7], we reserve the circle symbol for quaternion multiplication, e.g, $r = p \circ q$.

[3] Including both translation, and rotation parametrized as a unit quaternion.

[4] Symbolizing linear complementarity.

[5] We denote by $\underset{\smile}{x}$ a lower limit for $x$ and by $\overset{\frown}{x}$ an upper one.

[6] We shall often use this non-canonical and implicit version to stress the physical interpretation of $\mathcal{C}$.

[7] Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors."

#### II.5.1 Reparametrization $(\mathcal{R})$

More often than not, we shall reparametrize a rotation without being interested in expressing the details[8] of the process. For this we shall use $\mathcal{R}()$, meant as a function. The type of the reparametrization's target is indicated in the subscript using $q$, $M$, $aa$, and $v$ standing for quaternion, matrix, axis-angle and rotation vector respectively. The type of the source will be clear from the parameter. To give a practical example, $\mathcal{R}_q(h[_r v_i])$ is the quaternion equivalent to the rotation vector obtained by multiplying the time step with the angular velocity of body $i$, while $\mathcal{R}_M(h[_r v_i])$ is its matrix equivalent.

### II.6 Local Counterparts $(\breve{x})$

The most common frames of reference that we shall be using are world space (the frame $\mathcal{E}$) and body space, whose frame has as basis the body's principal axes of inertia. Given a body $i$, it is easy to denote a vector $x$ in any of the two frames as ${}^{\mathcal{E}}x$ and ${}^{i}x$ respectively. For a system of bodies however, even if we consider the rotational coordinates alone, we are faced with a system of frames. For this purpose, we shall use a special decoration. For any quantity (e.g, $x$) that is usually represented in $\mathcal{E}$, we shall denote its 'local counterpart' expressed in the aforementioned system of frames by $\breve{x}$. This means that while $x$ is simply ${}^{\mathcal{E}}x$ by our definition, $\breve{x}$ is related to $x$ depending on the nature of the quantity and has to be worked out. The most trivial (and at the same time motivating) example is angular velocity ${}_r v$, where the local counterpart ${}_r\breve{v}$ is exactly what is conventionally meant by 'local angular velocity' (even for a system of multiple bodies).

### II.7 Iteration $(\widehat{x})$

Following our pledge of immutability, we shall use a specific notation for iterative algorithms so that there is no ambiguity between a variable's value at different iterations. Iteration does not generally happen in time, so using the symbol corner designated for time can be ambiguous. Instead, we shall use a wide hat on top of the variable subject to iteration with the sequence number of the iteration next to it. for example, $\widehat{x}^{i+1} = \frac{1}{2}\widehat{x}^i$ is an update rule that halves $x$ at each iteration. Sometimes, the exact value of the last sequence number of a variable is either not known in pseudocode or too long to write. For this, we use the diamond symbol. for example, $y = \widehat{x}^{\diamond}$ assigns to $y$ the final iterate of $x$.

[8] Ibid.

## III Local Angular Velocity Adaptations

When working analytically, it is common to represent all coordinates in the absolute frame ($\mathcal{E}$), viz. in world space. While writing rigid body code, it is often advantageous to use local angular velocities, that is, to have each angular velocity represented in its body's frame. Within the code of ***Havok Physics***, these local velocities are used almost everywhere, with the body's frame having as basis the body's principal axes of inertia. As explained in the notation section, we disambiguate between $v$ and its local counterpart by denoting the latter by $\breve{v}$. In this section we derive the adaptations of relevant expressions as necessitated by the replacement of $v$ with $\breve{v}$.

### III.1 Constraint Formulation

'Jacobian Constraints'[9] are determined by

$$\dot{C}(q) = \frac{\partial C}{\partial q} S v = J v,$$

We seek a counterpart for $J$ that we denote by $\breve{J}$, such that the two satisfy

$$J v = \beta = \breve{J}\breve{v}.$$

For this purpose we split $J$ into a translational part ${}_t J$ that interacts exclusively[10] with ${}_t v$, and a rotational part ${}_r J$. We do this while ignoring the needed rearrangement of subvectors within $v$ since it does not affect our result. Noting that obviously,

$${}_t J = {}_t\breve{J},$$

[9] Erleben, "Stable, Robust, and Versatile Multibody Dynamics Animation," 60–68.

[10] This is possible because $J$ is a matrix.

we turn our attention to the rotational part. It is easy to see that looking at an individual constraint ($i$) and a single body ($j$) involved in it does not cause any loss of generality, so we carry the analysis within this context to simplify notation. The rotational contribution of that body to the number $(Jv)_i$ is

$$({}_rJ_{i,j})({}^{\mathcal{E}}_{r}v_j) = ({}_rJ_{i,j})({}^{j\curlywedge\mathcal{E}}R[{}^{j}_{r}v_j]),$$

where ${}_rJ_{i,j}$ is the transpose of some vector $k$ and by definition ${}^{j}_{r}v_j = {}^{j}_{r}\breve{v}_j$. We therefore have

$$\begin{aligned}(k^T)({}^{\mathcal{E}}_{r}v_j) &= (k^T)({}^{j\curlywedge\mathcal{E}}R)[{}^{j}_{r}\breve{v}_j]\\ &= ({}^{j\curlywedge\mathcal{E}}R^T\ k)^T[{}^{j}_{r}\breve{v}_j]\\ &= ({}^{\mathcal{E}\curlywedge j}R\ k)^T[{}^{j}_{r}\breve{v}_j].\end{aligned}$$

From this we see that the vectors $k_i$ that are transposed to become the rows of ${}_rJ_i$ can be simply adapted into

$$\breve{k}_i = (R_j^{-1})k_i,$$

which then are again transposed into the rows of ${}_r\breve{J}_i$. In summary we have

$${}_r\breve{J}^T{}_{i,j} = (R_j^{-1})_r J^T{}_{i,j}$$

and this concludes the derivation of $\breve{J}$.

### III.2 MLCP Formulation

Working along similar lines, we adapt the formulation of $\mathcal{C}$:

$$J(\underbrace{v + M^{-1}J^T\lambda}_{v'}) = \beta : \mathbb{C}\widehat{\lambda},$$

which simply implies the satisfaction of the velocity constraints at the next time step. From the previous adaptation we know that

$$J(v') = \breve{J}(\breve{v}'),$$

and since we wish to replace $J$ by its counterpart, it remains to examine $\breve{v}'$, specifically, its relation to $\breve{v}$. As in the previous adaptation, we drop the translational part and consider constraint and body $i$ and $j$. Another similarity as we shall see, is that like $\beta$, $\lambda$ can be left unadapted. Given that, we seek a counterpart for $M^{-1}$ such that

$${}_r\breve{v}' = {}_r\breve{v} + {}_r\breve{M}^{-1}{}_r\breve{J}^T{}_r\lambda.$$

Since ${}_r\breve{v}'$ is nothing but ${}_rv'$ in body frame space, we have

$$\begin{aligned}{}_r\breve{v}'_j &= {}^{\mathcal{E}\curlywedge j}R({}_rv_j + {}_rM_j^{-1}{}_rJ_{i,j}^T[{}_r\lambda_i)]\\ &= {}_r\breve{v}_j + {}^{\mathcal{E}\curlywedge j}R({}_rM_j^{-1}{}_rJ_{i,j}^T)_r\lambda_i,\\ &= {}_r\breve{v}_j + {}^{\mathcal{E}\curlywedge j}R({}_rM_j^{-1}[{}^{j\curlywedge\mathcal{E}}R({}_r\breve{J}_{i,j}^T)])_r\lambda_i,\\ &= {}_r\breve{v}_j + [\underbrace{{}^{\mathcal{E}\curlywedge j}R({}_rM_j^{-1})^{j\curlywedge\mathcal{E}}R}_{{}_r\breve{M}^{-1}{}_j}]({}_r\breve{J}_{i,j}^T)_r\lambda_i,\end{aligned}$$

Therefore, the counterpart of ${}_rM^{-1}$ is simply obtained by the change of basis

$${}^{j\curlywedge j}{}_r\breve{M}^{-1} = {}^{\mathcal{E}\curlywedge j}R({}^{\mathcal{E}\curlywedge\mathcal{E}}{}_rM^{-1})^{j\curlywedge\mathcal{E}}R.$$

## IV Comparison of Stepping Algorithms

In this section, we shall visit a chain of algorithms, starting from a (quite pure) conceptual one and ending at the actual algorithm that ***Havok Physics*** implements. Along the way, we shall overload no symbols at all, so that the reader can compare any two algorithms quite precisely. for example, we find the symbol $\beta$ in all first four algorithms and this means that all of them use exactly the same constraint bias[11]. On the other hand, $J$ is only used in the conceptual algorithms and replaced by $\breve{J}$ in all others. To avoid clutter, the listing of the algorithms is deferred to the end of the document.

### IV.1 The Two 'Essential' Algorithms

***Algorithm 1*** can be thought of as the conceptual essence of an implementation of the $\Gamma$ approach and has therefore the shortest pseudocode. Assuming the use of a direct solver

[11] Without forgetting its dependence on the time step as indicated in the algorithms.

for $\mathcal{C}$, it is the one that would produce the highest fidelity of simulation compared to the other versions. However, such an implementation has not turned out performant enough for adoption in ***Havok Physics***. Furthermore, a direct MLCP solver is technically more complicated to implement than an iterative one. For this reason, ***Algorithm 2*** is the first step towards the actual implementation; it is still quite conceptual, but iterative. The pseudocode of these two algorithms should require no comment except that we have chosen the coordinate stepping scheme that ***Havok Physics*** implements[12].

## IV.2 Havok Stepping, Conceptual

***Algorithm 3*** introduces a concept quite unique to the ***Havok Physics*** implementation, namely sub-stepping. The idea behind it is to achieve the higher quality associated with the reduction of step size during numerical integration[13] without suffering the computational cost of running the full simulation at a reduced time-step.

### IV.2.1 Sub-stepping

Before we describe sub-stepping, let us abuse the symbol $\Gamma$ and introduce some convenient pseudo-formal terms:

- $\Gamma(l, x)$ : A step of a simulation at time $l$ with a time-step of $x$, using a direct solver for $\mathcal{C}$.
- $\widehat{\Gamma}(l, x)$ : Ditto, but using an iterative solver. When the number of iterations is fixed, it can be made explicit by superscript e.g, $\widehat{\Gamma}^{\mathrm{m}}(l, x)$.
- $\Gamma(l, x, y)$ : A step of a simulation using sub-stepping to emulate $\Gamma(t, \frac{x}{y})$, $y$ being a natural number. Obviously, we expect $\Gamma(l, x, 1)$ to compute the same result as $\Gamma(l, x)$, numerical issues aside.
- $\widehat{\Gamma}(l, x, y)$ : Ditto, but using an iterative solver.
- $\Gamma(l, x, \widetilde{y})$ : Here, the emulation of the simulation with time-step $\frac{x}{y}$ is approximative, and ditto for $\widehat{\Gamma}(l, x, \widetilde{y})$.

Let us furthermore divide a conceptual rigid body simulation into four steps: geometry processing (s1), formulation of velocity constraints for contact points and joints (s2), velocity stepping including solving the velocity constraints (s3), and coordinate stepping (s4). With the above we are finally in a position to faithfully explain how sub-stepping achieves its goal by describing what happens during $\Gamma(t, h, n)$:

- Step s1 is run identically to ${}_{\mathrm{s1}}\Gamma(t, h)$.
- Step s2 is run identically to ${}_{\mathrm{s2}}\Gamma(t, \frac{h}{n})$. Here, the time-step enters into the calculation of coordinate drift[14‡] correcting constraint biases.
- Step s3 loops $n$ times, called sub-steps. Sub-step $j$ (one based loop counter) being identical to step ${}_{\mathrm{s3}}\Gamma(t + (j-1)\frac{h}{n}, \frac{h}{n})$ followed by an emulation of the effect of step s4 (of the same) on $\mathcal{C}^{t+j\frac{h}{n}}$ by updating a part of it.
- Step s4 steps the coordinates to $(t + h)$.

With this terminology A3[15] can be tersely describe by saying that it implements ${}_{\mathrm{s3}}\widehat{\Gamma}(t, x, y)$ while A2 implements ${}_{\mathrm{s3}}\widehat{\Gamma}(t, x)$. When comparing A3 to A2 there are two useful cases to consider depending on the time-step used for A2. While for A3 we are mainly interested in

$$ {}_{\mathrm{s3}}\widehat{\Gamma}(t, x = h, y = n), $$

for A2 we should consider both

$$ {}_{\mathrm{s3}}\widehat{\Gamma}(t, x = h) $$

and

$$ {}_{\mathrm{s3}}\widehat{\Gamma}(t, x = \frac{h}{n}), $$

such that the goal of sub-stepping can be restated as: the usage of A3 to achieve the higher quality of the latter combined with the lower computational cost of the former.

### IV.2.2 Comparison to *Algorithm 2*

General Differences

[12] Nohra, ***The Mathematics of Havok's Solver***, 1, 4, 5.
[13] Colloquially called 'stiffness' within ***Havok***
[14] Baraff, "Linear-Time Dynamics using Lagrange Multipliers," 138.
[15] A$i$ is shorthand for ***Algorithm*** $i$.

1. What is called iteration both traditionally and in A2 is called micro-iteration in A3; the iterative refinement of $\lambda$ at a certain time.
2. In A2, the iteration processes $\lambda$ row by row. In A3, some rows are solved simultaneously that is to say that some constraints are block-solved[16]. We do not analyze the details of block-solving here, but we note that only constraints with infinite lower and upper limits for $\lambda$ are solved this way. When this is the case, the block is effectively a linear system and not a MLCP. This system is then solved using inversion of a small (up to $3 \times 3$) sub-matrix of $\Gamma$. Such constraints are called 'stable' in ***Havok Physics*** because they additionally have the feature that their Jacobian and bias are calculated using a next frame approximation‡.
3. In A2, PSOR[17] is employed as indicated by the use of the parameter $w$. This is of course not the only way to solve the MLCP iteratively, but it was chosen to highlight the fact that the parameters $\kappa$ and $\tau$ used in A3 (mirroring their use in the actual implementation)[18] are not the ones used in PSOR, at least not exactly‡.
4. In A2, iteration is stopped once some convergence criterium or an iteration cap is reached. This ensures that the estimate for $\lambda$ is good enough. For performance reasons, this is not done in A3 and a low (usually equal to one) iteration count ($m$) is used unconditionally.

Sub-stepping Related Differences

1. For A3, ${}_{\text{s3}}\widehat{\Gamma}(t,h,n)$ must emulate the computation of Jacobians and biases for all sub-steps beyond the first one as seen at ***proc3:15***[19]. This is not necessary in any of the A2's ${}_{\text{s3}}\widehat{\Gamma}(t+(j-1)\frac{h}{n},\frac{h}{n})$ steps that A3 is emulating since this calculation usually happens in step s2. In the case of A3, s2 only calculates the quantities involved used in the sub-step $(t+0\frac{h}{n})$.
2. In A3 and the algorithms that follow it, local angular velocity is used along with all related local counterparts. This has computational performance advantages, but it also has the well known property of making the mass matrix time independent, and it does not have to be updated during sub-stepping. This can be observed by noticing the absence of $\breve{M}^{-1}$ at ***proc3:15***. Also, the order of quaternion multiplication has to be adapted (***proc3:19***).
3. So that in practice, the performance cost of A3 is close to that of ${}_{\text{s3}}\widehat{\Gamma}(t,h)$ the number of micro-iterations has to be adjusted. for example, if the latter is run with four micro-iterations, and the former with four sub-steps and four micro-iterations, it will be roughly four times costlier. Instead, A3 is usually run with only one micro-iteration. Even though the total number of iterations on $\lambda$ is then the same, A3 provides the higher quality it promises because it emulates a reduced integration step. In short, the quality of $\widehat{\Gamma}^1(t,h,n)$ is equal to that of $\widehat{\Gamma}^1(t,\frac{h}{n})$, which is better[20]‡ than that of $\widehat{\Gamma}^{\text{n}}(t,h)$. Yet simpler, computation cycles are better invested in sub-steps than in micro-iterations.

## IV.3 Havok Stepping, Approximative

***Algorithm 4*** differs from its predecessor in that it resorts to a number of approximations in order to improve performance. In terms of pseudocode, the differences occur at the lines 15,16, and 20 of ***proc4***. We shall examine them one by one and observe that as a whole, A4 viz. $\widehat{\Gamma}(t,h,\widetilde{n})$ is at best a first order approximation of A3.

### IV.3.1 Approximation of the Bias

A4 approximates the bias at ***proc4:16*** using the update rule

$$\underline{\beta^{j+1}}_{(\frac{h}{n})} = \underline{\beta^{j}}_{(\frac{h}{n})} - \breve{J}^j\breve{v}^j$$

We shall study this approximation by examining its effect on contact point constraints (ignoring friction), which were at the base of its development[21]. To do this, let us first recall the formulation of this kind of constraint, working in $\mathcal{E}$ because our conclusion will be independent of the frame used.

[16] Saad, ***Iterative Methods for Sparse Linear Systems***.

[17] Erleben, "Numerical Methods for Linear Complementarity Problems in Physics-based Animation," 16–17, Saad, ***Iterative Methods for Sparse Linear Systems*** pp.97.

[18] Strunk, ***Solver Introduction***, 3.

[19] We shall refer to lines in the algorithm listings by specifying the procedure name followed by the line number separated by a colon.

[20] The truth of this claim holds in practice for most use cases of ***Havok Physics***

[21] Ibid., 3.

**IV.3.1.1 Contact Constraint Formulation** Consider two points $a$ and $b$ with coordinates $p_a$ and $p_b$ each fixed on rigid bodies $A$ and $B$ respectively. Let a constraint defined by a fixed axis $(n)$ and a limit $(hi)$ relate the two points by the inequality

$$p_b \cdot n \leq (p_a \cdot n) + hi.$$

The biased velocity formulation of this constraint is

$$(\dot{p}_a - \dot{p}_b) \cdot n \geq (\tfrac{-1}{h})[(p_a - p_b) \cdot n + hi].$$

Taking point $a$ as an example and using the symbol $\sigma$ for arm, the relations between the coordinates of a body and a fixed point on it are[22]

$$\begin{aligned} p_a &= {}_t q_A + {}^{\mathcal{E}}\sigma_a, \\ \dot{p}_a &= {}_t \dot{q}_A + {}_r v_A \times ({}^{\mathcal{E}}\sigma_a). \end{aligned}$$

These can be plugged into the constraint's formulations to extract the Jacobian and the bias which is

$$\begin{aligned} J_{(a,b)} &= (n^T, [{}^{\mathcal{E}}\sigma_a \times n]^T, -n^T, -[{}^{\mathcal{E}}\sigma_b \times n]^T), \\ \beta_{(a,b)} &= (\tfrac{-1}{h})[{}_t\beta_{(a,b)} + {}_r\beta_{(a,b)} + hi], \\ {}_t\beta_{(a,b)} &= ({}_t q_A - {}_t q_B) \cdot n, \\ {}_r\beta_{(a,b)} &= (\sigma_a - \sigma_b) \cdot n. \end{aligned}$$

**IV.3.1.2 Exact Bias Update** Let us now consider the bias at $(t + h)$ by plugging in the stepped coordinates into its expression. We obtain

$$\begin{aligned} {}_t\beta'_{(a,b)} &= ([{}_t q_A + h_t v_A] - [{}_t q_B + h_t v_B]) \cdot n', \\ {}_r\beta'_{(a,b)} &= ([\mathcal{R}_M(h[{}_r v_A])\sigma_a] - [\mathcal{R}_M(h[{}_r v_B])\sigma_b]) \cdot n', \end{aligned}$$

**IV.3.1.3 Approximate Bias Update** The first approximation made is to assume that the constraint axis does not change much between two frames, viz.

$$n \approx n'.$$

For our case of contact point constraints, this is a spatio-temporal coherence assumption that is affected by[‡] the way in which the closest points between the two bodies change as they move. An example of an extreme case in which the assumption holds exactly because the axis is not a function of the configuration is that of a rigid body colliding with a flat plane of infinite mass. Heuristically, the assumption also holds reasonably well when the bodies involved are either spherical or rotate quite slowly within the time-step.

In any case, having accepted this approximation we start getting closer to the formula at ***proc4:16*** if we consider the translational part of the bias which can be neatly reformulated by using the Jacobian:

$$\begin{aligned} {}_t\beta'_{(a,b)} &\approx {}_t\beta_{(a,b)} + ([h_t v_A] - [h_t v_B]) \cdot n \\ &\approx {}_t\beta_{(a,b)} + h({}_t J_{(a,b)}[{}_t v_{A,B}]). \end{aligned}$$

Having obtained this expression, the second approximation is to extrapolate the validity of it to the rotational part of the bias update and assume that

$${}_r\beta'_{(a,b)} \approx {}_r\beta_{(a,b)} + h({}_r J_{(a,b)}[{}_r v_{A,B}]).$$

The above expands to

$${}_r\beta'_{(a,b)} \approx ([\sigma_a - \sigma_b] + h({}_r v_A \times \sigma_a - {}_r v_B \times \sigma_b)) \cdot n.$$

Comparing this to the exact value of ${}_r\beta'_{(a,b)}$ we see that it is an approximation of it that boils down to

$$\mathcal{R}_M(\alpha)\sigma \approx \sigma + (\alpha \times \sigma).$$

This is a first order approximation of rotation which can be obtained, for instance, from the Taylor expansion of the exponential of $\alpha$'s skew-symmetric matrix[23]. With these two approximations we arrive to

$$\begin{aligned} \beta'_{(a,b)} &\approx (\frac{-1}{h})({}_t\beta_{(a,b)} + h({}_t J_{(a,b)}[{}_t v_{A,B}]) \\ &\qquad + {}_r\beta_{(a,b)} + h({}_r J_{(a,b)}[{}_r v_{A,B}]) + hi) \\ &\approx (\tfrac{-1}{h})[{}_t\beta_{(a,b)} + {}_r\beta_{(a,b)} + hi] - J_{(a,b)}[v'_{A,B}] \\ &\approx \beta_{(a,b)} - J_{(a,b)}[v_{A,B}], \end{aligned}$$

Which is the update rule used in A3.

[22] Shabana, ***Computational Dynamics***, 105–10.

[23] Argyris, "An Excursion into Large Rotations," 91.

**IV.3.1.4 A Baked-in Formula** It is important to note that, as we pointed out, this was developed mainly with contact point constraints in mind. For other kinds of constraints the formula might not hold at all. For constant velocity motor constraints per example, the bias is the constant motor velocity (possibly with added drift control) and the update formula is another one. What happens in the code of ***Havok Physics*** is that such a constraint must undo the update rule's calculation in some way, paying the price for a constraint type based performance optimization that bakes in its update formula forcing it on all other types.

### IV.3.2 Approximation of the Jacobian

As seen in A4's listing, ***proc4:15*** skimps over the details of Jacobian approximation. This is because in the code of ***Havok Physics*** what happens depends on the type of the constraint and we shall not treat all the types here. Instead, we summarize the situation with a list.

- For contact point constraints, the approximation $n \approx n'$ that was already taken for granted in the bias update rule makes the local counterpart of the Jacobian independent of time and hence require no updates during sub-stepping.
- For some constraints, The Jacobian cannot be made time independent, per example when the vector used depends on the body positions or constrained point positions. Here, if $\breve{J}^{t+j\frac{h}{n}} \approx \breve{J}^{t+(j+1)\frac{h}{n}}$ is assumed, the gravity of its effect depends highly on the type of the constraint[‡].
- For 'stable' constraints such as the ***Havok Physics*** stiff spring, the Jacobian is indeed updated by first stepping the coordinates for the bodies involved and recalculating the Jacobian. This is however only an approximation because the update is usually done while solving the MLCP, the velocities used thereby not being the final ones, the ones used to step the coordinates at the end of the algorithm.

### IV.3.3 Coordinate Stepping Approximation

Instead of multiplying all the rotations into an exact final one as done in ***proc3:19***, A4, again extrapolating a translational formula (this time the translation stepping at ***proc4:19***), makes use of the approximation

$$\mathcal{R}_q(\frac{h}{n}{}_r\breve{v}^1) \circ \cdots \circ \mathcal{R}_q(\frac{h}{n}{}_r\breve{v}^n) \approx \mathcal{R}_q(\sum_{k=1}^{n}[\frac{h}{n}{}_r\breve{v}^k]).$$

By this it appeals to the vectorial nature (and hence additivity) of angular velocity and assumes it holds for small rotations. The approximation is first order as can be seen in the section on small rotations.

## IV.4 Havok Stepping, Actual

***Algorithm 5*** is what one find in the code of ***Havok Physics***. The changes it makes to its predecessor introduce no new ideas. Instead expressions are refactored and shuffled around to minimize the number of computations using two main devices which we now describe.

### IV.4.1 Additional Jacobian Approximation

The sub-stepping of $\mathcal{C}$ is refactored. Any recalculation of the Jacobian is now done while micro-iterating[24] and the bias update rule is transformed such that some more calculations are collapsed. This is motivated by the observation that in A3 we have

$$\beta^{x+1} = \beta^x - \breve{J}^x\breve{v}^x = (\beta^{x-1} - \breve{J}^{x-1}\breve{v}^{x-1}) - \breve{J}\breve{v}^x.$$

The above can be continued until $\beta^1$ is reached. A final approximation is made here. It is assumed that alone for the bias calculation, it is safe to assume that $\breve{J}^{t+j\frac{h}{n}} \approx \breve{J}^{t+(j+1)\frac{h}{n}}$, effectively approximating all Jacobians with $J^1$. This is admitted while at the same time the Jacobians are actually still recalculated for some (the 'stable') constraints. Because of this, our pseudocode listing of A5 removes the time-indicating symbol corner of the Jacobian altogether due to the fuziness of what happens in the actual code. Heuristically, this approximation drags the expected quality of the actual implementation below that of even $\widehat{\Gamma}^1(t, h, \widetilde{n})$, further towards $\widehat{\Gamma}^{\mathrm{n}}(t, h)$ and away from $\widehat{\Gamma}^1(t, \frac{h}{n})$.

[24] This means that the Jacobian changes at each micro-iteration. This makes it difficult to quantify this approximation because normally, problems are held fixed during their iterative solution.

With this, all the Jacobians on the righ-hand-side are identified and we get

$$\beta^{x+1} \approx \beta^1 - \breve{J}^1 \sum_{i=1}^{x-1} \breve{v}_x.$$

#### IV.4.2 Refactoring

The stabilization factors are baked into the quantities they affect. Surrounding the ensuing variables with a box to emphasize this fact, and introducing two new variable, A5 defines the following (taking A4 as a reference for the variables on the right)

$$\begin{aligned}
\boxed{\beta} &= \frac{\tau}{\kappa}\beta, \\
\boxed{s\breve{v}^x} &= \frac{\tau}{\kappa}\sum_{i=1}^{x-1}\breve{v}^x, \boxed{s\breve{v}^1} = 0, \\
\boxed{m\breve{v}^x} &= \breve{v}^x + \boxed{s\breve{v}^x}, \\
\boxed{M^{-1}} &= \kappa M^{-1} \\
\boxed{\dot{\lambda}} &= \frac{1}{\kappa}\dot{\lambda}
\end{aligned}$$

We let these variables stand out so much because while working with the code of ***Havok Physics***, one encounters the two velocity related variables under the names 'Velocity' and 'SumVelocity', which can be confusing when one mistakenly assumes to be dealing with $\breve{v}$ and $\sum_{i=1}^{x}\breve{v}_x$ instead. A similar problem occurs for with the mass related quantity.

Conceptually, the refactoring can be seen as driven by dividing both sides of $\mathcal{C}$ as used in ***proc4:11*** by $\kappa$. This produces the related problem $\boxed{\mathcal{C}}$, which we illustrate while temporarily removing all decorations.

$$\begin{aligned}
J(\tfrac{1}{\kappa}\kappa\, v + \tfrac{1}{\kappa}M^{-1}J^T\lambda) &= \tfrac{1}{\kappa}\tau\beta \quad : \mathtt{C}\widehat{\dot{\lambda}}, \\
J(v + \underbrace{M^{-1}J^T[\frac{1}{\kappa}\lambda]}_{\frac{\delta}{\kappa}}) &= \tfrac{\tau}{\kappa}\beta \quad : \mathtt{C}\widehat{\dot{\lambda}}.
\end{aligned}$$

Compared to $\mathcal{C}$, this MLCP would solve for the impulse $\frac{\delta}{\kappa}$. We modify the problem again such that it solves for $\delta$. For that we use our modified mass and reach

$$J(v + \underbrace{\overbrace{(\kappa M^{-1})}^{\boxed{M^{-1}}}J^T[\frac{1}{\kappa}\lambda]}_{\frac{\kappa}{\kappa}\delta}) = \frac{\tau}{\kappa}\beta : \mathtt{C}\widehat{\dot{\lambda}}.$$

Finally, we use the bias's approximate expansion in terms of the sum of velocities to obtain the form of $\boxed{\mathcal{C}}$ as used in the actual algorithm:

$$J(\overbrace{v^j + \boxed{s\breve{v}^j}}^{\boxed{m\breve{v}^j}} + \underbrace{\overbrace{(\kappa M^{-1})}^{\boxed{M^{-1}}}J^T\overbrace{[\frac{1}{\kappa}\lambda]}^{\boxed{\dot{\lambda}}}}_{\frac{\kappa}{\kappa}\delta}) = \boxed{\beta^1} : \mathtt{C}\widehat{\dot{\lambda}}.$$

## V Small Rotation Approximations

In this section we provide an infinitesimal oriented proof showing that the orientation stepping approximation as explained previously is of first order. We reserve the variable name $a$ for axis-angle rotations and denote them by $a = (\theta, \mathbf{n})$. Similarly, we reserve the variable name $q$ for quaternions and denote them by $q = (s, v_i\mathbf{e_i})$ where $v_i\mathbf{e_i}$ is a shortcut for $\sum_{i=1}^{3} v_i\mathbf{e_i}$ where $\mathbf{e_i}$ are the conventional unit vectors. Finally, by the inifinitesimality of all angles involved, we shall freely use the approximations $\cos(x) \approx 1$ and $\sin(x) \approx x$, identifying '$\approx$' with '$=$'.

### V.1 Proof of Commutativity

Consider two rotations $a_1 = (2\theta, \mathbf{n})$ and $a_2 = (2\alpha, \mathbf{m})$. By infinitesimality, their corresponding quaternions are

$$q_1 = (1, \theta n_i \mathbf{e_i})$$

and

$$q_2 = (1, \alpha m_i \mathbf{e_i}).$$

Using quaternion multiplication we obtain

$$q_1 \cdot q_2 = (1 - \circledcirc^2, [\theta n_i + \alpha m_i + \circledcirc^2]\mathbf{e_i}),$$

where each $\circledcirc^2$ denotes some infinitesimal of second order, obtained by the multiplication of first order infinitesimals, e.g $\alpha\theta = \circledcirc^2$. Eliminating second order infinitesimals we get

$$q_1 \cdot q_2 = (1, [\theta n_i + \alpha m_i]\mathbf{e_i}).$$

It is easy to see that we obtain the same result for $(q_2 \cdot q_1)$ and commutativity is thus proved.

## V.2 Proof of Additivity

Consider the rotation vectors $\mathbf{u}$ and $\mathbf{v}$ corresponding to $a_1$ and $a_2$ above. We shall prove that for small rotations, the simple componentwise addition $(\mathbf{u}+\mathbf{v})$ is a rotation vector that parametrizes the same rotation as the one obtained by the (commutative) composition of $a_1$ and $a_2$.
We have

$$\mathbf{u} = 2\theta n_i \mathbf{e_i},$$

$$\mathbf{v} = 2\alpha m_i \mathbf{e_i},$$

and their sum

$$\mathbf{w} = 2[\theta n_i + \alpha m_i]\mathbf{e_i}.$$

On the other hand, with

$$a_3 = (||\mathbf{w}||, \frac{\mathbf{w}}{||\mathbf{w}||}),$$

its corresponding quaterion $q_3$, in terms of $\mathbf{w}$ is

$$q_3 = (\cos(\frac{||\mathbf{w}||}{2}), \frac{\mathbf{w}}{||\mathbf{w}||}\sin(\frac{||\mathbf{w}||}{2})).$$

By infinitesimality, the above simplifies to

$$q_3 = (1, \frac{\mathbf{w}}{2}) = (1, [\theta n_i + \alpha m_i]\mathbf{e_i}),$$

which is precisely $(q_1 \cdot q_2)$ as worked out in the commutativity proof, concluding the demonstration.

# VI Listing of Algorithms

**output** : $q', v'$

**Procedure** proc1$(q, v, h, [ext, M^{-1}, J, \underset{\frown}{\beta}_{(h)}, \overset{\frown}{\lambda}])$

  **begin** Step velocities

    Solve for $\grave{\lambda}$ in

    $$J([v + ext] + \underbrace{M^{-1}J^T\grave{\lambda}}_{\grave{\delta}}) = \underset{\frown}{\beta}_{(h)} : \mathtt{C}\overset{\frown}{\lambda}$$

    $v' = v + \grave{\delta}$

  **begin** Step coordinates

    ${}_tq' = {}_tq + h_tv'$

    ${}_rq' = \mathcal{R}_q(h_rv') \circ {}_rq$

**Algorithm 1:** Essential Stepping

**output** : $q', v'$

**Procedure** proc2$(q, v, h, [ext, M^{-1}, J, \underset{\frown}{\beta}_{(h)}, \overset{\frown}{\lambda}], [\sigma w, \alpha p, \omega])$

  **begin** Step velocities

    $\widehat{v'}^0 = (v + ext)$

    **begin** Iterate

      **while** ***convergence*** $(\sigma w)$ ***or iteration cap*** $(\alpha p)$ ***not reached*** **do**

        **foreach** ***Jacobian row*** $x$ **do**

          Solve $\grave{\lambda}$ in

          $$J_x(\widehat{v'}^\diamond + \underbrace{M_x^{-1}J^T{}_x\grave{\lambda}}_{\grave{\delta}}) = \underset{\frown}{\beta_x}_{(h)} : \mathtt{C}\overset{\frown}{\lambda_x}$$

          $\widehat{v'}^{\diamond+1} = (1 - \omega)\widehat{v'}^\diamond + (\omega)\grave{\delta}$

    $v' = \widehat{v'}^\diamond$

  **begin** Step coordinates

    ${}_tq' = {}_tq + h_tv'$

    ${}_rq' = \mathcal{R}_q(h_rv') \circ {}_rq$

**Algorithm 2:** Essential Stepping, Iterative

**output** : $q', \breve{v}'$

**Procedure**

proc3$(q, \breve{v}, h, n, [\kappa, \tau], [\mathit{ext}, \breve{M}^{-1}, \breve{J}, \underline{\beta}_{(\frac{h}{n})}, \widecheck{\lambda}], [m])$

- **begin** Step velocities
  - **begin** Sub-step
    - $\breve{v}^0 = \breve{v}, \;\; \breve{J}^1 = \breve{J}, \;\; \underline{\beta^1}_{(\frac{h}{n})} = \underline{\beta}_{(\frac{h}{n})}$
    - **for** $j = 1$ **to** $n$ **do**
      - $\breve{v}^j = (\breve{v}^{j-1} + \mathit{ext})$
      - **begin** Micro-iterate
        - $\widehat{\breve{v}^j}^0 = \breve{v}^j$
        - **for** $i = 1$ **to** $m$ **do**
          - **foreach** *block constraint* $x$ **do**
            - Solve for $\grave{\lambda}$ in $\breve{J}^j_x(\kappa\, \widehat{\breve{v}^j}^\diamond + \underbrace{\breve{M}_x^{-1} \breve{J}^{T,j}{}_x \grave{\lambda}}_{\grave{\delta}}) =$ $\tau \underline{\beta^j_x}_{(\frac{h}{n})} : \mathtt{C}\widecheck{\lambda_x}$
            - $\widehat{\breve{v}^j}^{\diamond+1} = \widehat{\breve{v}^j}^\diamond + \grave{\delta}$
        - $\breve{v}^j = \widehat{\breve{v}^j}^\diamond$
      - **begin** Sub-step $\mathcal{C}$ to $(t + j\frac{h}{n})$
        - Calculate $\breve{J}^{j+1}$ and $\underline{\beta^{j+1}}_{(\frac{h}{n})}$
- **begin** Step coordinates
  - $\breve{v}' = \breve{v}^n$
  - ${}_t q' = {}_t q + \sum_{k=1}^n [\frac{h}{n}\, {}_t \breve{v}^k]$
  - ${}_r q' = {}_r q \circ \mathcal{R}_q(\frac{h}{n}\, {}_r \breve{v}^1) \circ \cdots \circ \mathcal{R}_q(\frac{h}{n}\, {}_r \breve{v}^n)$

**Algorithm 3:** Havok Stepping, Conceptual

**output** : $q', \breve{v}'$

**Procedure**

proc4$(q, \breve{v}, h, n, [\kappa, \tau], [\mathit{ext}, \breve{M}^{-1}, \breve{J}, \underline{\beta}_{(\frac{h}{n})}, \widecheck{\lambda}], [m])$

- **begin** Step velocities
  - **begin** Sub-step
    - $\breve{v}^0 = \breve{v}, \;\; \breve{J}^1 = \breve{J}, \;\; \underline{\beta^1}_{(\frac{h}{n})} = \underline{\beta}_{(\frac{h}{n})}$
    - **for** $j = 1$ **to** $n$ **do**
      - $\breve{v}^j = (\breve{v}^{j-1} + \mathit{ext})$
      - **begin** Micro-iterate
        - $\widehat{\breve{v}^j}^0 = \breve{v}^j$
        - **for** $i = 1$ **to** $m$ **do**
          - **foreach** *block constraint* $x$ **do**
            - Solve for $\grave{\lambda}$ in $\breve{J}^j_x(\kappa\, \widehat{\breve{v}^j}^\diamond + \underbrace{\breve{M}_x^{-1} \breve{J}^{T,j}{}_x \grave{\lambda}}_{\grave{\delta}}) =$ $\tau \underline{\beta^j_x}_{(\frac{h}{n})} : \mathtt{C}\widecheck{\lambda_x}$
            - $\widehat{\breve{v}^j}^{\diamond+1} = \widehat{\breve{v}^j}^\diamond + \grave{\delta}$
        - $\breve{v}^j = \widehat{\breve{v}^j}^\diamond$
      - **begin** Sub-step $\mathcal{C}$ to $(t + j\frac{h}{n})$
        - Calc. $\breve{J}^{j+1}$ or set it to $\breve{J}^j$
        - $\underline{\beta^{j+1}}_{(\frac{h}{n})} = \underline{\beta^j}_{(\frac{h}{n})} - \breve{J}^j \breve{v}^j$
- **begin** Step coordinates
  - $\breve{v}' = \breve{v}^n$
  - ${}_t q' = {}_t q + \sum_{k=1}^n [\frac{h}{n}\, {}_t \breve{v}^k]$
  - ${}_r q' = {}_r q \circ \mathcal{R}_q(\sum_{k=1}^n [\frac{h}{n}\, {}_r \breve{v}^k])$

**Algorithm 4:** Havok Stepping, Approximative

**Algorithm** `proc5()`
`proc5(`$q, \breve{v}, h, n, [\kappa, \tau], [ext, \breve{M}^{-1}, \breve{J}, \underline{\beta}_{(\frac{h}{n})}, \widehat{\lambda}], [m]$`)`
**begin** Step velocities
  **begin** Sub-step
    $\breve{v}^0 = \breve{v},\ \boxed{m\breve{w}^0} = \breve{v}^0,\ \underline{\underline{\boxed{\beta^1}}}_{(\frac{h}{n})} = \frac{\tau}{\kappa}\underline{\beta}_{(\frac{h}{n})}$
    **for** $j = 1$ **to** $n$ **do**
      **begin** Micro-iterate
        $\widehat{\boxed{m\breve{w}^j}}^0 = \boxed{m\breve{w}}^j$
        **for** $i = 1$ **to** $m$ **do**
          **foreach** *block constraint* $x$ **do**
            Solve for $\boxed{\grave{\lambda}}$ in
            $\breve{J}_x(\widehat{\boxed{m\breve{w}^j}}^{\diamond} + \underbrace{\boxed{\breve{M}_x^{-1}}\breve{J}^{T,j}{}_x\boxed{\grave{\lambda}}}_{\grave{\delta}}) =$
            $\underline{\underline{\boxed{\beta_x^1}}}_{(\frac{h}{n})} : \mathtt{C}\widehat{\lambda}_x$
            $\widehat{\boxed{m\breve{w}^j}}^{\diamond+1} = \widehat{\boxed{m\breve{w}^j}}^{\diamond} + \grave{\delta}$
          $\boxed{\breve{w}^j} = \widehat{\boxed{m\breve{w}^j}}^{\diamond}$
        **begin** Sub-step $\mathcal{C}$ to $(t + j\frac{h}{n})$
          $\breve{v}^j = \boxed{m\breve{w}^j} - \boxed{s\breve{v}^j}$
          $\boxed{s\breve{v}^{j+1}} = \boxed{s\breve{v}^j} + \frac{\tau}{\kappa}\breve{v}^j$
          $\breve{v}^{j+1} = (\breve{v}^j + e\breve{x}t)$
          $\boxed{m\breve{w}^{j+1}} = \breve{v}^{j+1} + \boxed{s\breve{v}^{j+1}}$
**begin** Step coordinates
  $\breve{v}' = \breve{v}^n$
  ${}_tq' = {}_tq + \sum_{k=1}^{n}[\frac{h}{n}{}_t\breve{v}^k]$
  ${}_rq' = {}_rq \circ \mathcal{R}_q(\sum_{k=1}^{n}[\frac{h}{n}{}_r\breve{v}^k])$

**Algorithm 5:** Havok Stepping, Actual

# References


Argyris, J. "An Excursion into Large Rotations." ***Computer Methods in Applied Mechanics and Engineering*** 32, no. 1-3 (1982): 85–155. http://www.scopus.com/inward/record.url?eid=2-s2.0-0019613550/&/#38;partnerID=40.

Baraff, David. "Linear-Time Dynamics using Lagrange Multipliers." ***Computer Graphics*** 30, no. Annual Conference Series (1996): 137–46. http://citeseer.ist.psu.edu/baraff96lineartime.html.

Diebel, James. "Representing attitude: Euler angles, unit quaternions, and rotation vectors." ***Matrix*** 58 (2006): 1–35. doi:10.1093/jxb/erm298.

Erleben, Kenny. "Numerical Methods for Linear Complementarity Problems in Physics-based Animation." In ***ACM SIGGRAPH 2013 Courses***. SIGGRAPH '13. New York, NY, USA: ACM, 2013. doi:10.1145/2504435.2504443.

———. "Stable, Robust, and Versatile Multibody Dynamics Animation." Edited by Knud Henriksen. PhD thesis, University of Copenhagen, 2004.

Nohra, Jad. ***The Mathematics of Havok's Solver***. (Work In Progress). Havok; Internal Technical Report, 2014. //depot/Other/Physics/Presentations/Internal/TheMathematicsOfHavoksSolver.pdf.

Saad, Yousef. ***Iterative Methods for Sparse Linear Systems***. Second. SIAM, 2003.

Shabana, A. ***Computational Dynamics***. New York: Wiley, 1994.

Strunk, Oliver. ***Solver Introduction***. Havok; Product Documentation, 2004. //depot/Other/Personal/Jad.Nohra/ByOthers/SolverIntroduction.doc.