Visual interactive simulation 7.5p 5DV058 2009

Kenneth Bodin, Umeå University

Notes on the SPOOK method for simulating constrained mechanical systems

Contents

This is a description of a method for numerically integrating a mechanical system with constraints, for example contact constraints and distance constraints. An iterative Gauss-Seidel procedure for solving the resulting mixed linear complementarity system (MLCP) is also described. Note that the methods are described in a prescriptive format. For the full derivation, stability analysis and more, please refer to the doctoral thesis of Claude Lacoursière [1]. Also the textbook, Physics Based Animation by Kenny Erleben et.al. is a useful reference [2]. A white paper for Game Developers Conference by Erin Catto also has a clear and pedagogical presentation of a closely related method [3]. In particular, Catto's paper has an alternative and more general description of the book keeping and mapping procedure used in Gauss-Seidel iterations.

The method described here can be varied in a number of ways, in particular in how contacts are split into resting contacts and high velocity impacts. Here, impacts are treated using a Newton-Coulomb impulse law. The friction model described here is called scaled box and has its limitations. A number of different alternative friction models exist, and overall, dry frictional contacts in multi body systems is by no means a solved problem! Also, the Gauss-Seidel solver is just one example of how the equations can be solved.

The time integration is done with a leapfrog type of method, derived from a discrete variational principle. With no constraints present, everything just follows leapfrog.

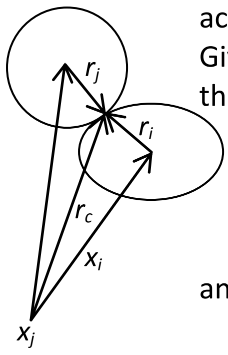
Algorithm description

Find intersection pairs

Find all interacting pairs (i,j), and for each contact find the contact set, that is the contact point, the penetration depth, and the contact normal. You may want to use a broad-phase method for screening out contacts in large systems, for example spatial hashing or sweep-and-prune.

Resolve high velocity impacts

Loop over all intersecting pairs. If colliding, compute a Newton-Coulomb impulse according to the following procedure:
Given a relative velocity $\boldsymbol{u}$, at the contact point $\boldsymbol{r}_c$ we first assume stick friction and the relative post collision velocities at the contact are,



$$\begin{aligned} u_n' &= -e\,u_n \\ u_t' &= 0 \end{aligned} \qquad 1)$$

and for this case we compute the impulse from,

$$\boldsymbol{J} = K^{-1}(\boldsymbol{u}' - \boldsymbol{u}) \qquad 2)$$

where $\boldsymbol{K}$ is the collision matrix. We can only use this impulse if it is within the allowed friction cone,

$$J_t \leq \mu\, J_n \qquad 3)$$

otherwise the contact is sliding and we compute the impulse from,

$$j = \frac{-(1+e)u_n}{\boldsymbol{n}^T K(\boldsymbol{n} - \mu \boldsymbol{t})} \qquad 4)$$

$$\boldsymbol{J} = j\,\boldsymbol{n} - \mu\, j\, \boldsymbol{t} \qquad 5)$$

Apply the impulse and impulsive torque to the linear and angular velocities of both bodies,

$$\boldsymbol{v}_{i,j} = \boldsymbol{v}_{i,j} \pm m_{i,j}^{-1}\boldsymbol{J} \qquad 6)$$

$$\boldsymbol{\omega}_{i,j} = \boldsymbol{\omega}_{i,j} \pm I_{i,j}^{-1}(\boldsymbol{r}_{i,j} \times \boldsymbol{J}) \qquad 7)$$

where $I_{i,j}$ is the inertia tensor of bodies i and j, respectively. Note that the inertia tensor is typically represented in body coordinates, so to use it here in the equations of motion you must transform it to world coordinates.

Solve for contact and constraint forces

Assuming all impacts have been resolved, we next turn to contacts. We use SPOOK to stabilize contacts, and start from the matrix equation (we refer to this as the SPOOK equation on Schur complement form),

$$(GM^{-1}G^T + \Sigma)\,\lambda = -a\,G\,q - b\,G\,W - h\,G\,M^{-1}f \qquad 8)$$

where h is the timestep, G the Jacobian matrix, M the mass matrix, q the generalized position vector, W the generalized velocity vector and f the generalized force vector. Σ is a diagonal matrix $\Sigma = \epsilon\hat{I}$. The constant parameters are,

$$a = \frac{4}{\mathrm{h}(1+4d)} \qquad 9)$$

$$b = \frac{4d}{1+4d} \qquad 10)$$

$$\epsilon = \frac{4}{\mathrm{h}^2 k(1+4d)} \qquad 11)$$

with k being a spring constant, and d is the number of timesteps it takes to stabilize the constraint. k is chosen somewhat freely to both relax the constraint. It also improved the convergence rate since the regularization parameter, ϵ, improves the condition number of the matrix problem. The relaxation is typically chosen to $d \approx 3-5$ timesteps. In general, both d and k can be set specifically for each constraint, but for simplicity we use global values of these here.
The matrix equation form is very general and contains all the degrees of freedom of the simulated system. The mass matrix is block diagonal and contains the mass and inertia of all bodies,

$$M = \begin{bmatrix} m_1\hat{I} & & & & 0 \\ & I_1 & & & \\ & & m_2\hat{I} & & \\ & & & I_2 & \\ 0 & & & & \ddots \end{bmatrix}, \qquad (12)$$

with $\hat{I}$ here being a 3x3 unit matrix.

The generalized velocity vector holds both linear and angular velocities of all bodies,

$$W = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{\omega}_1 \\ \boldsymbol{v}_2 \\ \boldsymbol{\omega}_2 \\ \vdots \end{bmatrix}. \qquad (13)$$

The generalized force vector holds all linear forces and torques acting on all bodies,

$$f = \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{\tau}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{\tau}_2 \\ \vdots \end{bmatrix} \qquad (14)$$

The Jacobian matrix represents all the constraints and is typically very sparse with the constraints as matrix blocks. This also means that the "spring" used for constraint stabilization acts in generalized coordinates. It acts in a direction perpendicular to the constraint surface, whether the constraint is modelled in position coordinates, angular coordinates or a combination thereof.

Also note that the entire right hand side of the SPOOK matrix equation can be precomputed before we solve for λ since it does not change during the iterations.

Once we have solved for λ we can compute new generalized velocities at timestep $(k+1)$ from the by updating the velocities from the previous step (k) with the now known forces,

$$W_{k+1} = W_k + M^{-1}G^T\lambda_k + h^{-1} M^{-1} f_k \,. \tag{15}$$

Solving SPOOK with Gauss-Seidel iterations

The SPOOK equation above has the following general form

$$S_\epsilon \lambda = B \tag{16}$$

with the obvious definitions of S_ϵ and B. We can solve this using Gauss-Seidel iterations. We first split $S_\epsilon = D + L + U$ where D is the diagonal and L and U are the upper and lower parts of the matrix S_ϵ. Note that while G usually is very sparse, S_ϵ is not necessarily sparse, so we do not compute S_ϵ explicitly. On matrix form, the Gauss-Seidel iterative procedure prescribes that the solution vector in iteration $(\nu + 1)$ is computed from the previous iteration (ν), according to

$$\lambda^{\nu+1} = (D+L)^{-1}(B - U\lambda^\nu) \tag{17}$$

In practise, we need this on component form,

$$\lambda_i^{\nu+1} = \frac{1}{S_{\epsilon,ii}}\left[B_i - \sum_{j<i} S_{\epsilon,ij}\,\lambda_j^{\nu+1} - \sum_{j>i} S_{\epsilon,ij}\,\lambda_j^{\nu}\right] \tag{18}$$

In other words, we compute the new contact force $\lambda_i^{\nu+1}$ using the best current guess of all other contact forces in the system, either from the previous iteration (ν) or the ones already computed in the current iteration $(\nu + 1)$.
We may rewrite the procedure slightly by also including the previous value of λ_i^ν itself on the right hand side. We do this by changing from $j > i$ to $j \geq i$ in the sum. Rather than computing a new value for $\lambda_i^{\nu+1}$, we then rather compute a new contribution $\Delta\lambda_i^{\nu+1}$

$$\Delta\lambda_i^{\nu+1} = \frac{1}{S_{\epsilon,ii}}\left[B_i - \sum_{j<i} S_{\epsilon,ij}\,\lambda_j^{\nu+1} - \sum_{j\geq i} S_{\epsilon,ij}\,\lambda_j^{\nu}\right] \tag{19}$$

We get the new value from accumulating

$$\lambda_i^{\nu+1} = \Delta\lambda_i^{\nu+1} + \lambda_i^{\nu} \, . \tag{20}$$

For the right hand side we can precompute B, but we also need to gather information from all previously computed λ's. Rather than gathering this information every time, recomputing terms of the type, $S_{\epsilon,ij}\,\lambda_j^{\nu+1}$, we can scatter and accumulate the information every time we solve for a new λ and thus have the required information stored for each new $\lambda_i^{\nu+1}$ that we solve for.
To do this, we observe that terms of the type $S_{\epsilon,ij}\lambda_j = (GM^{-1}G^T + \Sigma)_{ij}\lambda_j$ have units of velocity. Since Σ is a diagonal matrix, it is nonzero only for $i = j$, and we can split this into $S_{\epsilon,ij}\lambda_j = (GM^{-1}G^T)_{ij}\lambda_j + \epsilon\delta_{ij}\lambda_j$. Since constraint impulses are given by $M^{-1}G^T\lambda$ it turns out that the sums over $(GM^{-1}G^T)_{ij}\lambda_j$ measure the contribution to the constraint velocities from the current approximation we have of our constraint impulse λ. So, it is convenient to gather information about the current values of λ as velocity contributions on the bodies that participate in the constraint, and then compute current constraint velocities from this. Thus, whenever we solve for a new contribution to the constraint impulse $\Delta\lambda_i^{\nu+1}$ we also compute the updated velocities of the two bodies of constraint i from this impulse. We will summarize this below in the overall algorithm description.

The contact Jacobian

A non-penetration constraint is defined as

$$g = \left(\boldsymbol{x}_j + \boldsymbol{r}_j - \boldsymbol{x}_i - \boldsymbol{r}_i\right) \cdot \boldsymbol{n}_i \, , \tag{21}$$

where definitions from the figure above apply. $\boldsymbol{n}_i$ is defined to be the normal pointing out from body i. The corresponding kinematic constraint is obtained by taking the time derivative (using the chain rule)

$$\begin{aligned} \frac{dg}{dt} &= \left(\boldsymbol{v}_j + \boldsymbol{\omega}_j \times \boldsymbol{r}_j - \boldsymbol{v}_i - \boldsymbol{\omega}_i \times \boldsymbol{r}_i\right) \cdot \boldsymbol{n}_i \\ &+ \left(\boldsymbol{x}_j + \boldsymbol{r}_j - \boldsymbol{x}_i - \boldsymbol{r}_i\right) \cdot (\boldsymbol{\omega}_i \times \boldsymbol{n}_i) \end{aligned} \tag{22}$$

The penetration is small and therefore the second term is often neglected,

$$\frac{dg}{dt} \approx \left(\boldsymbol{v}_j + \boldsymbol{\omega}_j \times \boldsymbol{r}_j - \boldsymbol{v}_i - \boldsymbol{\omega}_i \times \boldsymbol{r}_i\right) \cdot \boldsymbol{n}_i \, . \tag{23}$$

This is often a good approximation, but in certain cases when rotations dominate the motion that leads to a contact, it can give artefacts. On the other hand, including the second term gives a Jacobian that contains velocities, and is hard to deal with. We know that the Jacobian describes a mapping between the generalized velocities and the kinematic constraint,

$$\frac{dg}{dt} = GW \, . \tag{24}$$

From this we can identify the Jacobian of the contact constraint,

$$G_c = \left[-\boldsymbol{n}^T \quad -(\boldsymbol{r}_i \times \boldsymbol{n})^T \quad \boldsymbol{n}^T \quad (\boldsymbol{r}_j \times \boldsymbol{n})^T\right], \tag{25}$$

where we have dropped the index i from $\boldsymbol{n}$.In this case, it can be shown (do it!) that the term $G_c M^{-1} G_c^T$ for each contact contains a block of the form $\boldsymbol{n}^T K \boldsymbol{n}$ where K is called the "collision matrix" for that particular contact (see lecture notes). The collision matrix describes the effective inverse inertia seen by the constraint force.

Case with a single contact

For a single contact (l) without friction, between bodies (i, j), the matrix equation reduces to (verify),

$$({\boldsymbol{n}_l^T}^T K_l \boldsymbol{n}_l + \epsilon)\, \lambda_l = \left(-a\, q_l - \, b\, \boldsymbol{u}_l \cdot \boldsymbol{n}_l - \boldsymbol{u}_l^f \cdot \boldsymbol{n}_l\right). \tag{26}$$

The contact impulse λ_l thus compensates for the penetration depth, q_l, the normal contact velocity $\boldsymbol{u}_l \cdot \boldsymbol{n}$ and the change in the normal contact velocity, $\boldsymbol{u}_l^f \cdot \boldsymbol{n}$ due to external forces. Since we only have a single equation, the Gauss-Seidel process is exact and converges in one step. An example of this would be a sphere on a plane, and it is most useful to use this for debugging since numeric print-outs can be compared with your manual calculations. If your physics engine doesn't deal with this correctly, it sure won't handle the many-body case either!

The contact constraint is a one-sided, unilateral, constraint. This means that the contact force can only be repulsive (otherwise contacts can be sticky). In a Gauss-Seidel procedure this is solved through *projection*, i.e. if we find a negative λ, we just set it to zero.

Case with many coupled contacts

When we have multiple coupled contacts, l, the iterate for each contact can be written on the form,

$$\Delta\lambda_l^{\nu+1} = \frac{1}{(\boldsymbol{n}_l^T K_l \boldsymbol{n}_l + \epsilon)}\left(-a\, q_l - \, b\, \boldsymbol{u}_l \cdot \boldsymbol{n}_l - \boldsymbol{u}_l^f \cdot \boldsymbol{n}_l \, - \, \boldsymbol{u}_l^\lambda \cdot \boldsymbol{n}_l - \epsilon \lambda_l^\nu\right) \tag{27}$$

or, wrapping the constant terms on the right hand side into B_l, we get,

$$\Delta\lambda_l^{\nu+1} = \frac{1}{(\boldsymbol{n}_l^T K_l \boldsymbol{n}_l + \epsilon)}\left(B_l \, - \, \boldsymbol{u}_l^\lambda \cdot \boldsymbol{n}_l - \epsilon \lambda_l^\nu\right) \tag{28}$$

Here, $\boldsymbol{u}_l^\lambda$ is the velocity at the contact point that comes from the contribution of all previously computed λ's, including any previous approximation of λ_l itself. The term $\boldsymbol{u}_l^f$ residing from external forces and torques can be computed if we know,

$$\boldsymbol{v}_i^f = \frac{h}{m_i}\, f_i \tag{29}$$

$$\boldsymbol{\omega}_i^f = I_i^{-1}\, h\, \boldsymbol{\tau}_i \tag{30}$$

Thus for the contact l, this gives us an updated contact force (initially zero),

$$\lambda_l^{\nu+1} = \Delta\lambda_l^{\nu+1} + \lambda_l^{\nu} \tag{31}$$

As mentioned, the *total* contact force *must* be repulsive, so $\Delta\lambda_l^{\nu+1}$ must be clamped,

$$if\ \lambda_l^{\nu+1} < 0\ then\ \lambda_l^{\nu+1} = 0 \text{ and } \Delta\lambda_l^{\nu+1} = -\lambda_l^{\nu}\ . \tag{32}$$

Note that each contribution $\Delta\lambda_l^{\nu+1}$ can indeed be negative, but the total constraint impulse must be positive.

For the book keeping of $\boldsymbol{u}_l^{\lambda}$ we keep track of the velocity contribution accumulated from each (projected) $\Delta\lambda_l^{\nu+1}$ on the two bodies (i,j) participating in the constraint, also taking Newton's third law into account,

$$\boldsymbol{v}_{i,j}^{\lambda} = \boldsymbol{v}_{i,j}^{\lambda} \pm \frac{1}{m_{i,j}}\ \Delta\lambda_l^{\nu+1}\boldsymbol{n}_l \tag{33}$$

$$\boldsymbol{\omega}_{i,j}^{\lambda} = \boldsymbol{\omega}_{i,j}^{\lambda} \pm I_{i,j}^{-1}\ \Delta\lambda_l^{\nu+1}(\boldsymbol{r}_{i,j} \times \boldsymbol{n}_l) \tag{34}$$

These velocity contributions can be used to compute $\boldsymbol{u}_l^{\lambda}$ whenever we solve for a new $\Delta\lambda_l^{\nu+1}$. When we have run our iterations for λ we can compute the new final velocities at time (k+1) by adding all velocity contributions from external forces and constraint forces to the velocity of the previous timestep,

$$\boldsymbol{v}_i^{k+1} = \boldsymbol{v}_i^{k} + \boldsymbol{v}_i^{f} + \boldsymbol{v}_i^{\lambda} \tag{35}$$

$$\boldsymbol{\omega}_i^{k+1} = \boldsymbol{\omega}_i^{k} + \boldsymbol{\omega}_i^{f} + \boldsymbol{\omega}_i^{\lambda} \tag{36}$$

Integrating the position variables

With the procedure described above, we now have computed new velocities. We use these to update the positions $\boldsymbol{x}$ and quaternions $\boldsymbol{Q}$, for each body i,

$$\boldsymbol{x}_i^{k+1} = \boldsymbol{x}_i^{k} + h\,\boldsymbol{v}_i^{k+1} \tag{37}$$

$$q_i^{k+1} = q_i^{k} + \frac{h}{2}\ \omega_i^{k} q_i^{k} \tag{38}$$

It is important that you renormalize the quaternions regularly, since they tend to drift in this numerical scheme, and if they aren't normalized they actually don't correspond to rotations at all.

The notation for the quaternion integration is non-standard, but customary, and requires some explanation. To be written on this form, ω requires a special construction as a 4-vector,

$$\omega = [0 \quad \omega_x \quad \omega_y \quad \omega_z]. \tag{39}$$

See Chapter 18 in [2] for further explanation.

Distance constraint

A distance constraint constrains two points, one on each body, to be a certain distance apart. It is very similar to the non-penetration constraint, but with a normal defined to be the direction between the two points, and the constraint error is the distance between the two points minus the desired distance. This is a bilateral constraint, so no projection is required.
The distance constraint is often called a ball-and-socket constraint, since this is what it typically looks like in a real mechanism. The ball is free to rotate in the socket, but the position is otherwise fixed.

Using a distance constraint, it is possible to model not only jointed mechanisms, but also ropes and cloth.

Friction

Dry frictional contacts is a nonlinear problem in general and therefore very hard to solve unless certain approximations are made. Here we describe one of the simpler methods, called *scaled box friction*.

We use two kinematic constraints to model friction,

$$\dot{g}_{t_1} = \left(\boldsymbol{v}_j + \boldsymbol{\omega}_j \times \boldsymbol{r}_j \boldsymbol{v}_j - \boldsymbol{v}_i - \boldsymbol{\omega}_i \times \boldsymbol{r}_i\right) \cdot \boldsymbol{t}_1 \,, \tag{40}$$
$$\dot{g}_{t_2} = \left(\boldsymbol{v}_j + \boldsymbol{\omega}_j \times \boldsymbol{r}_j \boldsymbol{v}_j - \boldsymbol{v}_i - \boldsymbol{\omega}_i \times \boldsymbol{r}_i\right) \cdot \boldsymbol{t}_2 \,. \tag{41}$$

The two constraints act to constrain motion along the two tangential directions $\boldsymbol{t}_1$and $\boldsymbol{t}_2$. The tangents are constructed to satisfy $\boldsymbol{n} = \boldsymbol{t}_1 \times \boldsymbol{t}_2$. Obviously, this splitting into two tangential directions is artificial, and rather than constraining the friction force into a cone, we constrain it in a box.

The corresponding Jacobian for the kinematic friction constraints is,

$$G = \begin{pmatrix} -\boldsymbol{t}_1^T & -(\boldsymbol{r}_i \times \boldsymbol{t}_1)^T & \boldsymbol{t}_1^T & \left(\boldsymbol{r}_j \times \boldsymbol{t}_1\right)^T \\ -\boldsymbol{t}_2^T & -(\boldsymbol{r}_i \times \boldsymbol{t}_2)^T & \boldsymbol{t}_2^T & \left(\boldsymbol{r}_j \times \boldsymbol{t}_2\right)^T \end{pmatrix}. \tag{42}$$

This allows us to model the direction of the friction force. In order to model the magnitude we apply scaling laws on the corresponding constraint forces,

$$-\mu m g \le \lambda_{t_1} \le \mu m g \,, \tag{43}$$
$$-\mu m g \le \lambda_{t_2} \le \mu m g \,. \tag{44}$$

Here, μ is the friction coefficient (we assume it is the same for static and dynamic friction), g is the gravitational acceleration ($9.81\ m/s^2$) and m is a “representative mass”. This mass is typically set to the the reduced mass of the bodies participating in the constraints, distributed over the number of constraints. This may sound like an awkward approximation, and it is, but it still gives an acceptable friction behaviour in most cases. However, it gives artefacts for bodies rotation flat against a plane, since the direction of friction changes non-smoothly. It also gives artefacts in stacking problems since the maximum friction force at the top of the stack is as large that the maximum friction forces at the bottom of the stack even though normal forces are much larger at the bottom and therefore should allow for larger friction forces too.

Alternative friction models

You are free to experiment with alternative friction models. In particular you may try replacing the limits of the scaled box model with actual normal forces. For example, you may run a number of iterations on contact with zero friction to compute approximate normal forces. Next you can use these normal forces as limits of the frictional forces Then you turn on friction and run additional iterations where you use these approximate normal forces for computing the limits of the friction forces.

Also, rather than using the box directions, $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, you may experiment with using the current estimate of the tangential contact velocities for computing the direction of friction, just like we did for high velocity impacts.

However, note that the methods above *may* have problems with convergence. In general, the coupling between the friction force and the normal force is non-linear and attempts to patch up this during the iterations can cyclic behaviour in the iterations, so that e.g. the direction of friction flips alot. At the end of the day this can introduce jitter and instabilities in a friction problem.

More research is required to improve models of dry friction in contacting multibody systems.

Some notes on the Gauss-Seidel method

The Gauss-Seidel method is referred to as a linear relaxational method. The reason is that the global error in the current approximation of λ fall of linearly with the number of iterations. This is relatively slow, and if a high precision solve is required, we would need a very large number of iterations. This can be understood if we consider a case with two coupled constraints where one constraint force is 100 times larger than the other constraint force. If we start by guessing that both constraint forces are zero, and then compute the first, and thereafter the second, we end up having a large error. The iterations will "relax" this error by smearing it out between the two constraints at a rate of $1/N$, so if the error is 1 to start with and we'd like to go to 10^{-6}, then we actually need 10^{6} iterations! This can become a real problem in stiff systems, for example if we have a body of weight 1 kg with a body weighing 1000 kg resting heavily on top of it. Errors in the resulting large constraint force don't affect the heavier body very much, but they can have dramatic effects on the lighter body. Another example is a wire consisting of light rigid body elements attached to each other with distance constraints. If the system is static and the wire carries a heavy load, each mass element will experience large constraint forces, but zero net force. Even small errors in the constraint force will severely limit stability and physical realism of this wire.

Often these Gauss-Seidel errors lead to very springy constraints, so the wire becomes much more elastic than it should be.

The regularization Σ improves the convergence and stability for such systems. Convergence can also be improved by warm starting the iterations and reusing the λ's from the previous timestep as a starting point for the iterations. However, in practise the Gauss-Seidel method is still too inefficient for systems with large mass ratios or systems with stiff constraint forces.

An alternative iterative method could be the preconditioned conjugate gradient method (CG). However, CG is not overly efficient for MLCP's since the projection often ruins the convergence rate. It is known to have good global convergence, but the local convergence can even be worse than with Gauss-Seidel. This is not optimal since local convergence often is important in visual simulation.

Another relevant alternative is to use a direct sparse method, which can deliver the correct solution to machine precision after a fixed number of algebraic manipulations of the equations. In practise, a direct sparse solver is right now the only alternative when solving for high fidelity physics in stiff systems and systems with large mass ratios. For sparse systems (e.g. a wire) it has the same 1/N computational complexity as the Gauss-Seidel procedure but delivers a solution with 10^{10} times better precision! For less sparse systems and degenerate problems (many constraint do the exact same thing) or ill-posed problems (constraints counter-act eachother) it can be hard to find a solution at all, whereas Gauss-Seidel then tends to deliver a leat square minimization of the problem (if $|S_\epsilon \lambda - B| = 0$ doesn't have a solution, Gauss-Seidel can find $\min\left(|S_\epsilon \lambda - B|\right)$ - but it may also not converge at all!). There is yet *much* research to be done on solvers for rigid body dynamics and multiphysics systems! This area is also likely to influence processor design and multiprocessor communication paths in the future.

Sometimes the Gauss-Seidel procedure is referred to as an impulse propagation method. As described earlier, this is because the constraint forces can be accumulated into velocities and impulses updating these velocities can be said to propagate through the system. Although this is intuitively attractive, it is a bit of a fallacy since the direction of the impulse propagation depends entirely on the iteration order and therefore does not have physical relevance. It is possible to improve the convergence rate of Gauss-Seidel dramatically by chosing an efficient iteration order. For example, in a stack dominated by gravitational forces, the direction of gravity can be used in the iteration order. If the number of iterations are kept low it may also be wise to randomize the iteration order between timesteps (or even within the iteration) to avoid error accumulation.

References

[1] Claude Lacoursière, *Ghosts and machines: regularized variational methods for interactive simulations of multibodies with dry frictional contacts*, Doctoral thesis, Dept. Comp. Science, Umeå University, (2007). http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-1143

Claude's thesis covers everything described in this document and more. It has a thorough derivation of the SPOOK method from a discrete variational principle, and analyzes the stability and integration properties of the method in comparison with many other methods. It is important to understand that the SPOOK method that we use here only is a special case of SPOOK on Schur complement form. This works well for contacts and distance constraints, where we can do book keeping relatively easy on the velocities to preserve sparsity, but for many other constraint problems the underlying saddle-point matrix formulation is much more efficient, in particular when working with constraint motors. The thesis also has one of the best accounts of quaternion algebra for physics simulation in the entire literature. It also has sections covering the Gauss-Seidel procedure and other solution methods.

[2] Kenny Erleben, Jan Sporring, Knud Henriksen, Henrik Dohlman, *Physics Based Animation*, Charles River Media, (2005).

The book of Kenny Erleben et.al. is a good reference for many things in this document, including the derivation of the collision matrix and a thorough discussion on collision laws including the Newton-Coulomb law. As is often done in the literature, it divides methods into impulsive methods and constraint methods, which is questionable since it can be shown that impulsive methods are just special solution strategies and parameter choices of a constraint formulation. The book does not cover the SPOOK method, but it can be shown that the methods described in the constraint method chapter also can be derived from the SPOOK framework. The appendices are very useful and summarize rigid body mechanics, quaternion algebra, iterative methods and much more.

[3] Erin Catto, Iterative Dynamics, white paper for Game Developers Conference (2005). http://www.gphysics.com/downloads (also see other downloads, slides etc).

This paper has made strong impact in the game physics area and computer graphics since it is a clear and pedagogical presentation of a constraint stabilization scheme solved using Gauss-Seidel iterations. The formulation is very similar to SPOOK, but not as consistent in terms of parameters, and it doesn't at all apply a regularization term Σ.