This method can also be combined with adaptive time-stepping [PFTV92, Mir96b]. The goal of using an adaptive time step size is to achieve a predefined accuracy ε with a minimal computational effort. For the determination of the current step size, the truncation error of a step must be estimated. One way to do this is using the embedded Runge-Kutta formulas introduced by Fehlberg. Cash and Karp [CK90] use the general form of a fifth-order Runge-Kutta formula

\[
\begin{align*}
    k_1 &= h \mathbf{h}(t_i, \mathbf{z}_i) \\
    k_2 &= h \mathbf{h}(t_i + \frac{1}{2}h, \mathbf{z}_i + \frac{1}{2}k_1) \\
    \vdots \\
    k_6 &= h \mathbf{h}(t_i + ah, \mathbf{z}_i + b_6k_1 + \ldots + b_0k_5)
\end{align*}
\]

in combination with two different sets of parameters for the solution

\[
\begin{align*}
    \mathbf{z}_{i+1}^+ &= \mathbf{z}_i + \sum_{j=1}^{6} c_j k_j + O(h^5) \\
    \mathbf{z}_{i+1}^* &= \mathbf{z}_i + \sum_{j=1}^{6} c_j^* k_j + O(h^5).
\end{align*}
\]

The first set gives us a fifth-order Runge-Kutta. From the second we obtain an embedded fourth-order formula. The values for the parameters \(a_i, b_{ij}, c_i\) and \(c_i^*\) can be found in [CK90]. Now the error can be estimated as

\[\Delta\mathbf{z} = \mathbf{z}_{i+1} - \mathbf{z}_{i+1}^* = \sum_{i=1}^{6} (c_i - c_i^*) \cdot \mathbf{k}_i.\]

Since this error is of order five, a new step size can be determined by

\[h_{\text{new}} = h \left(\frac{\epsilon}{\Delta\mathbf{z}}\right)^{\frac{1}{5}}.\]

If the estimated error is smaller than the desired accuracy, the step size is increased. Otherwise if the desired accuracy could not be reached, the last step has to be simulated again with the new decreased step size.

### 4.2. Direct Methods

Direct methods are known to be computational heavy to use. Therefore, they are often not preferred for interactive simulation. However, their ability to deliver accurate solutions make them ideal to handle problems such as large mass ratios. Thus, for some applications direct methods are the only option. Among direct methods for linear complementarity problems (LCPs) based on pivoting are the Lemke method and the Keller method [CPS92a, Lac03]. We will present an incremental pivoting method in the spirit of [Bar94]. Before doing so we will first present a guessing approach that exploits that a LCP is a combinatorial problem. The LCP can be written as

\[
y \geq 0, \quad x \geq 0, \quad \text{and} \quad y^T x = 0,
\]

where \(y = Ax + b\). By algebraic manipulation we have,

\[
\begin{bmatrix} 1 & -A \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = b.
\]

Next we define the whole index set \(I = \{1, \ldots, n\}\) and introduce one index set of free variables \(y_i > 0\) and one of active variables \(x_i = 0\),

\[
\mathcal{F} \equiv \{i \mid y_i > 0\} \quad \text{and} \quad \mathcal{A} \equiv \{i \mid x_i > 0\}.
\]

We assume strict complementarity holds meaning we never simultaneously have \(y_i = 0\) and \(x_i = 0\). Thus, \(\mathcal{F} \cap \mathcal{A} = \emptyset\) and \(\mathcal{F} \cup \mathcal{A} = \{1, \ldots, n\}\). The idea is to create a method that can verify if any guess of \(\mathcal{F}\) and \(\mathcal{A}\) is a solution for the given LCP formulation. Using the index sets we make the partitioning

\[
\begin{bmatrix} 1_{\mathcal{F}} & -A_{\mathcal{A}\mathcal{A}} \end{bmatrix} \begin{bmatrix} y_{\mathcal{F}} \\ x_{\mathcal{A}} \end{bmatrix} = b,
\]

where \(1_{\mathcal{F}}\) and \(A_{\mathcal{A}\mathcal{A}}\) are the sub matrices given by the column indices \(\mathcal{F}\) and \(\mathcal{A}\). Our problem is simplified to verifying if the linear programming (LP) problem

\[
C \mathbf{x} = \mathbf{b} \quad \text{subject to} \quad \mathbf{x} \geq 0
\]

has a solution (same as \(b\) in positive cone of \(C\)). This can be done by first computing \(\mathbf{x}_A = -A_{\mathcal{A}\mathcal{A}}^{-1} \mathbf{b}_{\mathcal{A}}\) and verify if \(\mathbf{x}_A \geq 0\). Next one uses the feasible \(\mathbf{x}_A\) to compute \(y_{\mathcal{F}} = A_{\mathcal{F}\mathcal{A}} \mathbf{x}_A + \mathbf{b}_{\mathcal{F}}\) and finally verify if \(y_{\mathcal{F}} \geq 0\). If that last verification succeeds then a solution has been found. Observe that during the verification processes we only need to compute \(A_{\mathcal{A}\mathcal{A}}^{-1}\). If \(\|\mathbf{A}\| \ll n\) then verification will be fast.

In worst case the time complexity of guessing would be \(O(n^2 2^n)\) which is not computational very efficient. Another strategy is to be clever in making new guesses. For instance by applying a pivoting strategy or some strategy that builds up the index sets incrementally. Here we will focus on the latter idea. In the \(k^{th}\) iteration a new index will be selected from the current set of unprocessed indices, \(U \equiv I \setminus \{\mathcal{F} \cup \mathcal{A}\}\). The index sets \(\mathcal{F}\) and \(\mathcal{A}\) are initially both empty. Throughout, the complementarity conditions are kept as invariants. We will use superscript \(k\) to denote the values at a given iteration number. For any unprocessed index \(j \in U\) we implicitly assume \(x^0_j = 0\). Initially in the \(k^{th}\) iteration we use the partitioning

\[
\begin{align*}
\begin{bmatrix} y_{\mathcal{A}}^k \\ y_{\mathcal{F}}^k \\ y_{\mathcal{U}}^k \end{bmatrix} &= \begin{bmatrix} A_{\mathcal{A}\mathcal{A}} & A_{\mathcal{A}\mathcal{F}} & A_{\mathcal{A}\mathcal{A}}^j \\ A_{\mathcal{F}\mathcal{A}} & A_{\mathcal{F}\mathcal{F}} & A_{\mathcal{F}\mathcal{A}}^j \\ A_{\mathcal{U}\mathcal{A}} & A_{\mathcal{U}\mathcal{F}} & A_{\mathcal{U}\mathcal{U}} \end{bmatrix} \begin{bmatrix} x_{\mathcal{A}}^k \\ x_{\mathcal{F}}^k \\ 0 \end{bmatrix} + \begin{bmatrix} b_{\mathcal{A}}^j \\ b_{\mathcal{F}}^j \\ b_{\mathcal{U}}^j \end{bmatrix}.
\end{align*}
\]

The next candidate index to be processed in the method is selected as the index \(j \in U\) that minimize \(y^k_j\) as this corresponds to an index in \(U\) with a most violated complementarity constraint. If for the minimum value \(y_j^k \geq 0\) is fulfilled,
the method terminates as this would indicate that all the remaining unprocessed indices trivially fulfill the complementarity conditions. If no unique feasible minimum exists then one may pick a minimizing index at random. In the kth iteration we use the partitioning and keep the complementarity conditions as invariants implying $y_{A}^{k+1} = 0$ and $x_{F}^{k+1} = 0$, so

$$
\begin{bmatrix}
0 \\
y_{F}^{k+1} \\
x_{F}^{k+1}
\end{bmatrix} = \begin{bmatrix}
A_{A,A} & A_{A,F} & A_{A,j} \\
A_{F,A} & A_{F,F} & A_{F,j} \\
A_{j,A} & A_{j,F} & A_{j,j}
\end{bmatrix} \begin{bmatrix}
x_{A}^{k+1} \\
y_{F}^{k+1} \\
x_{F}^{k+1}
\end{bmatrix} + \begin{bmatrix}
b_{A} \\
b_{F} \\
b_{j}
\end{bmatrix}.
$$

The changes in $y_{F}$ and $x_{A}$ with respect to $x_{j}^{k+1} > 0$ are given by

$$
x_{A}^{k+1} = x_{A}^{k} + \Delta x_{A} x_{j}^{k+1},
$$

$$
y_{F}^{k+1} = y_{F}^{k} + \Delta y_{F} x_{j}^{k+1},
$$

$$
x_{F}^{k+1} = x_{F}^{k} + \Delta y_{j} x_{j}^{k+1},
$$

where

$$
\Delta x_{A} = -A_{A,A}^{-1} A_{A,j},
$$

$$
\Delta x_{F} = -A_{F,A}^{-1} A_{F,j},
$$

$$
\Delta y_{j} = -A_{j,j} - A_{j,A}^{-1} A_{A,j}.
$$

The idea is to increase $x_{j}^{k+1}$ as much as possible without breaking any of the complementarity constraints. Thus, $x_{j}^{k+1}$ is limited by the blocking constraint set

$$
B_{A} \equiv \left\{ \frac{x_{j}^{k}}{\Delta x_{q}} \quad q \in A \land \Delta x_{q} < 0 \right\},
$$

$$
B_{F} \equiv \left\{ \frac{y_{F}^{k}}{\Delta y_{r}} \quad r \in F \land \Delta y_{r} < 0 \right\}.
$$

If no blocking constraints exist then $x_{j}^{k+1}$ is unbounded by $A$ and $F$. Thus, each partition results in the bounds

$$
x_{A}^{k} = \left\{ \frac{\infty}{\min B_{A}} : B_{A} = \emptyset \right\},
$$

$$
x_{F}^{k} = \left\{ \frac{\infty}{\min B_{F}} : B_{F} = \emptyset \right\},
$$

$$
x_{j}^{k} = \left\{ \frac{-y_{F}^{k}}{\Delta y_{j}} : \Delta y_{j} < 0 \right\}.
$$

The solution for the value of $x_{j}^{k+1}$ will be the minimum bound. If a blocking constraint is found from $B_{A}$ then a pivot operation is initiated moving the blocking index from $A$ to $F$ and vice versa if a blocking constraint is found in $B_{F}$.

The blocking constraint sets are changed as the active and free index sets $A$ and $F$ are changed by a pivoting operation. This implies that one could increase $x_{j}^{k+1}$ further after a pivoting step. Thus, we will continue to look for blocking constraints and perform pivoting on them until no more blocking constraints exist. Depending on the final value of $x_{j}^{k+1}$ index $j$ is assigned to either $F$ or $A$.

Noticing that the pivot step only swaps one index and therefore only changes the size of $A$ by one an incremental factorization method can be used for computing $A_{A,A}^{-1}$. There exist incremental factorization running in $O(n^2)$ time complexity. Baraff [Bar94] proved that the outer loop runs at most $O(n)$. Thus, a positive overall time complexity for the pivoting method is $O(n^3)$.

The pivoting method is capable of finding an accurate solution for the LCP whereas the iterative methods we cover in Section 4.3 and 4.4 only find approximate solutions. However, the accuracy is at the expense of having to form the $A$-matrix in the first place whereas the iterative methods often can exploit a factorization of the $A$-matrix given by the constraint Jacobians and the mass matrix, $JM^{-1}JT$. These matrices are extremely sparse and one can evaluate matrix-vector products more efficiently using the factorization than by first assembling the $A$-matrix which can be very dense even if it consists of products of sparse matrices.

If we let $b$ be the number of rigid bodies then storage complexity of the matrix product factorization $JM^{-1}JT$ is $O(b + n)$ compared by the $O(n^2)$ storage complexity for the full $A$-matrix. The assembly of the $A$-matrix takes at worse $O(n^2 + b^2 + b^3)$ time factorized. The iterative methods often only need to compute matrix-vector products using the matrix-product factorization since we often have $b \ll n$ this takes $O(n)$ time compared against a full matrix which takes $O(n^2)$ time.

### 4.3. Iterative Fixed Point Schemes

Most open source software for interactive real-time rigid body simulation uses the Projected Gauss–Seidel (PGS) method for computing contact forces. This includes the two most popular open source simulators Bullet and Open Dynamics Engine. PGS is computational very efficient with an iteration cost of $O(n)$, careful memory layout of sparse matrices allows for a memory footprint of $O(n)$. In addition to being computational and memory-wise efficient PGS is very robust and can deal gracefully with even bad or erroneous problems. For these reasons PGS is well suited for interactive applications like computer games.

#### 4.3.1. Matrix Splitting Methods

We introduce the matrix splitting $A = M - N$. Next we let $e^{i} = b - Nx$ then the LCP

$$
Ax + b \geq 0,
$$

$$
x \geq 0,
$$

$$
(x)^{T}(Ax + b) = 0.
$$

© The Eurographics Association 2012.
becomes
\[ M\mathbf{x}^{k+1} + c^k \geq 0, \]  \hspace{1cm} (66a)  
\[ x^{k+1} \geq 0, \]  \hspace{1cm} (66b)  
\[ (x^{k+1})^T(Mx^{k+1} + c^k) = 0. \]  \hspace{1cm} (66c)

This results in a fixed-point formulation where we hope that for a suitable choice of \( M \) and \( N \) the complementarity sub-problem might be easier to solve than the original problem. The splitting method can be summarized as

**Step 0** Initialization, set \( k = 0 \) and choose an arbitrary non-negative \( x^0 \geq 0 \).

**Step 1** Given \( x^k \geq 0 \) solve the LCP (66).

**Step 2** If \( x^{k+1} \) satisfy some stopping criteria then stop otherwise set \( k \leftarrow k + 1 \) and go to step 1.

The splitting is often chosen such that \( M \) is a Q-matrix. This means that \( M \) belongs to the matrix class of matrices where the corresponding LCP has a solution for all vectors \( c^k \).

Clearly if \( x^{k+1} \) is a solution for (66) and we have \( x^{k+1} = x^k \) then by substitution into the subproblem given by (66) we see that \( x^{k+1} \) is a solution of the original problem (65).

Next we will use the minimum map reformulation on the complementarity subproblem, this is equivalent to
\[ \min(x^{k+1},Mx^{k+1}+c^k) = 0. \]  \hspace{1cm} (67)

Subtract \( x^{k+1} \) and multiply by minus one,
\[ \max(0,-Mx^{k+1}-c^k+x^{k+1}) = x^{k+1}. \]  \hspace{1cm} (68)

Again we re-discover a fixed-point formulation. Let us perform a case-by-case analysis of the \( j^{th} \) component. If
\[ (x^{k+1}-Mx^{k+1}+c^k)_j < 0 \]  \hspace{1cm} (69)

then \( x^{k+1}_j = 0. \) Otherwise
\[ (x^{k+1}-Mx^{k+1}+c^k)_j = x^{k+1}_j. \]  \hspace{1cm} (70)

That is
\[ (Mx^{k+1})_j = c^k_j. \]  \hspace{1cm} (71)

For a suitable choice of \( M \) and back-substitution of \( c^k = b - Nx^k \) we have
\[ (M^{-1}(Nx^k-b))_j = x^{k+1}_j. \]  \hspace{1cm} (72)

Combining it all we have derived the closed form solution for the complementarity subproblem,
\[ \max(0,(M^{-1}(Nx^k-b))) = x^{k+1}. \]  \hspace{1cm} (73)

Iterative schemes like these are often termed projection methods. The reason for this is that if we introduce the vector \( z^k = M^{-1}(Nx^k-b) \) then
\[ x^{k+1} = \max(0,z^k). \]  \hspace{1cm} (74)

That is the \( k+1 \) iteration is obtained by projecting the vector \( z^k \) onto the positive octant. In a practical implementation one would rewrite the matrix equation (74) into a for loop that sweeps over the vector components and updates the \( x \)-vector in place. The result is the same pseudo code as given in Section 4.3.2.

One would want to use a clever splitting such that the inversion of \( M \) is computationally cheap. Letting \( L, D \) and \( U \) be the strict lower, diagonal and strict upper parts of \( A \), then three popular choices are: the projected Jacobi method \( M = D \) and \( N = L + U \), the projected Gauss–Seidel (PGS) method \( M = (L+D) \) and \( N = U \), and the projected Successive Over Relaxation (PSOR) method \( M = (D + \gamma L) \) and \( N = ((1-\gamma)D - \gamma U) \) where \( 0 \leq \gamma \leq 2 \) is the relaxation parameter. More about this parameter in Section 4.3.2.

It is worthwhile to note that \( A \) must at least have nonzero diagonal for these splittings to work. In general for non-symmetric matrices one may experience divergence. This means we can not apply these methods directly to the LCP model. Thus, in computer graphics an alternative model has been used which drops the principle of maximum dissipation. This alternative allows for a matrix splitting method to be derived [PNE10]. One may improve the accuracy and convergence rate of the resulting numerical method by using sub-space minimization [NSEL01] or a nonsmooth nonlinear conjugate gradient method [SNE10b].

It seems that all hope of using matrix splitting for the LCP model is lost. However, as we show in Section 4.3.3 and 4.3.4 a blocked version of the matrix splittings can be used for the LCP model.

### 4.3.2. Using A Quadratic Programming Problem

In our second approach for deriving the iterative methods PGS and PSOR we will make use of the quadratic programming (QP) problem reformulation. Our derivation follows in the footsteps of [Man84]. The reformulation allows us to prove convergence properties of the PGS and PSOR methods. We assume that \( A \) is symmetric and positive semi-definite then the LCP can be restated as a minimization problem of a constrained convex QP problem
\[ \mathbf{x}^* = \arg \min_{\mathbf{x} \geq 0} f(\mathbf{x}) \]  \hspace{1cm} (75)

where \( f(\mathbf{x}) \equiv \frac{1}{2}\mathbf{x}^T A\mathbf{x} + \mathbf{x}^T \mathbf{b} \). The first order optimality (Karush-Kuhn-Tucker) conditions [NW99] is equivalent to the LCP (65).

Given the \( i^{th} \) unit axis vector \( \hat{e}_i \) where \( \hat{e}_j = 0 \) for all \( j \neq i \) and \( \hat{e}_i = 1 \) then the \( i^{th} \) relaxation step consists in solving the one dimensional problem
\[ \tau^* = \arg \min_{\tau \geq 0} f(\mathbf{x} + \tau \hat{e}_i) \]  \hspace{1cm} (76)

and then setting \( \mathbf{x} \leftarrow \mathbf{x} + \tau \hat{e}_i \). One relaxation cycle consists of one sequential sweep over all \( i^{th} \) components.
The one dimensional objective function is rewritten as
\[
f(x + \tau \frac{\partial f}{\partial x}) = \frac{1}{2} x^2 A_{ij} + \tau (A x + b) + f(x).
\]
From which we find the unconstrained minimizer as \( \tau_0 = -\frac{1}{2} \frac{\partial f}{\partial x} \). Considering the constraint \( x_i + \tau \geq 0 \) we find the constrained minimizer to be \( \tau_c = \max(\tau_0 - x_i) \) which yields the final update rule for the relaxation step
\[
x_i \leftarrow \max \left( 0, x_i - \frac{\tau_c}{A_{ii}} \right).
\] (78)

This is algebraic equivalent to the \( \tau_{ib} \) component in the PGS update (74). Consider the polynomial \( g(\tau) = \frac{1}{2} A_{ii} \tau_i + \tau_x \).

We know \( A_{ii} > 0 \) so the legs of the polynomial are pointing upwards. The polynomial has one trivial root \( \tau = 0 \) and a minimum at \( \tau = -\frac{\tau_c}{A_{ii}} \) where \( g \left( -\frac{\tau_c}{A_{ii}} \right) = -\frac{\tau_c^2}{2 A_{ii}} < 0 \). The other root is found at \( \tau = -2 \frac{\tau_c}{A_{ii}} \). Thus, any \( \tau \) value in the interval between the two roots has the property
\[
\tau_c = -\gamma \frac{\tau_c}{A_{ii}} \Rightarrow g(\tau_c) < 0, \quad \forall \gamma \in [0, 2].
\] (79)

From this it follows that
\[
f(x + \tau_c \frac{\partial f}{\partial x}) = g(\tau_c) + f(x) \leq f(x), \quad \forall \gamma \in [0, 2]
\] (80)
with equality if \( \tau_c = 0 \). This results in the over relaxed version
\[
x_i \leftarrow \max \left( 0, x_i - \gamma \frac{\tau_c}{A_{ii}} \right).
\] (81)

This is in fact algebraic equivalent to the \( \tau_{ib} \) component of the PSOR update and contains the PGS method as a special case of \( \gamma = 1 \). Observe that by (80) we are guaranteed a non increasing sequence of iterates by our relaxation method. The complete iterative method can be listed as

1: \textbf{method} PSOR(N, \gamma, x, A, b)  
2: \textbf{for} k = 1 \textbf{to} N  
3: \textbf{for} all \( i \)  
4: \( r_i \leftarrow A_{ii} x + b_i \)  
5: \( x_i \leftarrow \max \left( 0, x_i - \gamma \frac{r_i}{A_{ii}} \right) \)  
6: \textbf{next} i  
7: \textbf{next} k  
8: \textbf{end method}

where \( N \) is the maximum number of allowed iterations and \( \gamma \) is the relaxation parameter.

4.3.3. The Blocked Gauss–Seidel Method

The matrix splitting and QP reformulation approaches imply that Gauss–Seidel methods can not be used for the LCP contact model due to its zero diagonal values and non symmetry of \( A \). However, the splitting idea can be applied in a blocked version. This results in a numerical method that is very easy to implement and still preserves the good numerical properties of the PGS method. A block is defined as all variables from one contact point. In the case of a four sided friction pyramid the \( ib \) block will consist of the normal impulse \( x_{ni} \), four friction impulses \( x_{fi} \), \( x_{bi}, x_{ni}, x_{fi}, \) and one slack variable \( b \). We introduce the block notation \([x]_i = [x_{ni}, x_{fi}, \cdots, b]_T \). Similar \([A]_ij \) is the sub block of \( A \) corresponding to the \( ib \) and \( jb \) contact point variables. Thus, the blocked LCP can be written
\[
[y]_i = \sum_j [A]_{ij} [x]_j + [b]_i \geq 0 \quad \forall i, \quad (82a)
\]
\[
[x]_i \geq 0 \quad \forall i. \quad (82b)
\]
\[
[y]_i^T [x]_i = 0 \quad \forall i. \quad (82c)
\]

Now we may apply the Gauss–Seidel splitting to the blocked LCP. The result is a blocked Gauss–Seidel (BGS) method,

1: \textbf{method} BGS(N, x, A, b)  
2: \textbf{for} k = 1 \textbf{to} N  
3: \textbf{for} all \( i \)  
4: \[ b]_i \leftarrow [b]_i - \sum_j [A]_{ij} [x]_j \]  
5: \textbf{solve-sub-lcp}([x]_i, [A]_{ij}, [b]_i)  
6: \textbf{next} i  
7: \textbf{next} k  
8: \textbf{end method}

The intuition behind the numerical method is that all contact point variables other than the \( ib \) block are momentarily frozen while solving for the variables of the \( ib \) block. The BGS approach is also known as a “sweeping process” or as the non-smooth contact dynamics (NSCD) method [Mor99, Jea99].

The sub block LCP in line 5 can be solved using any LCP solver one wants. Usually one would apply yet a splitting dividing the sub block LCP into a normal impulse sub block and a frictional sub block. The normal part is a 1D problem and can be solved by a projection. The frictional part would in our case be a 5D problem. It is a bit unpleasant as we have zero diagonal terms and non-symmetry of the frictional sub block part of \( A \). However, the low dimensionality would allow for an efficient direct enumeration approach or one may drop the principle of maximum dissipation – changing the contact model – but allowing us to reduce the number of variables to a 2D problem with a symmetric positive semi-definite frictional sub block matrix.

From a computer science viewpoint an implementation of this method is indistinguishable from an implementation of the propagation model. The main difference is that this is a numerical method for solving a simultaneous contact model whereas the other is a model in itself. Besides the former solves for force impulses whereas the latter solves for collision impulses. The similarity with the propagation model also give intuition to some of the traits of the numerical method. One may see propagation effects even though one is using a simultaneous model.

The blocked Gauss–Seidel method offers many possibil-
In Section 4.3.4 we divide a LCP into two sub blocks one with normal variables only and the other containing the rests. In fact one may use any kind of partitionings to create the sub blocks. For instance If the LCP includes joints one may create a sub block for all the joint variables. This joint sub block of the LCP is known to be equivalent to a symmetric positive semi-definite linear system. Thus, one may use a preconditioned conjugate gradient (PCG) solver to solve for joint impulses rather than a PGS method. As PCG has the same per-iteration cost as PGS but better convergence rate the result is much less joint drifting errors at the same cost as PGS. If the number of joints is sufficiently small one may use an incomplete Cholesky factorization to solve for joint impulses resulting in very accurate solutions. One may even take the BGS idea one step further and solve the joint sub block with a completely different approach like the reduced coordinate formulation in Section 3.2. In the extreme case BGS can be used to partitioning a configuration into sub blocks where one can apply specialized solvers for each sub block. This has been termed hierarchical solvers by the graphics and gaming community.

4.3.4. A Staggered Approach

One may combine the ideas of splitting the LCP and use QP reformulations. The idea is referred to as staggering [Lot84, KSP08]. We partition the LCP variables into three index sets, one corresponding to normal impulses $N$, and one to friction impulses $F$ and the last one is simply the slack variables $a$. Applying our partition would require us to solve the two coupled LCPs,

$$A_Nx_N + (b_N + A_Fx_F) \geq 0 \quad \perp x_N \geq 0$$

and

$$A_Fx_F - e^T \geq 0 \quad \perp x_F \geq 0$$

Taking a staggered approach one solves the top-most LCP first (normal force problem) and then the bottom-most LCP second (the friction force problem) and continues iteratively until a fixed-point is reached. This is in fact a blocked Gauss–Seidel splitting method.

Observe that the normal force problem has a symmetric positive semi-definite coefficient matrix $A_N$ making QP reformulations possible whereas the frictional problem has an non-symmetric matrix. One may exploit a QP reformulation anyway. Because the friction LCP is the first order optimality conditions of the QP problem

$$\mathbf{s}_F = \arg \min \frac{1}{2} x_F^T A_F x_F + c_F x_F$$

subject to

$$x_F \geq 0 \quad \text{and} \quad c_N - e^T x_F \geq 0,$$

where $c_N = \mu x_N$ and $c_F = b_F + B_F x_N$. Thus, any convex QP method can be used to solve for the normal and friction forces and one is guaranteed to find a solution for each subproblem. Whether the sequence of sub QP problems converge to a fixed point is not obvious.

There exist many variations over this staggering scheme [LL11]. For instance one variation is to use a blocked Gauss–Seidel method for the frictional problem rather than a QP reformulation. This is mostly due to performance. Keeping a QP solver for the normal problem helps getting accurate normal forces which are needed to deal with large mass ratios whereas accurate friction can be given up to some degree in interactive applications which means a Gauss–Seidel method is suitable for the friction problem.

4.4. Newton Methods

The PGS methods from Section 4.3 may suffer from viscous artifacts due to linear convergence rate. One remedy is to use Newton methods. These can provide quadratic convergence rates and thus offers more accurate solutions at a slightly higher per iteration computational cost than PGS methods. PATH [Pat05] is a well known Newton type solver for LCPs and used by many researchers in graphics and robotics. One drawback of PATH is that computing time scales quadratically in the number of contacts $O(n^2)$. Here we will present a specialized Newton type solver and an open source implementation can be found in [Ear11].

The Fischer function is defined as

$$\phi(a, b) = \sqrt{a^2 + b^2} - (a + b) \quad \text{for } a, b \in \mathbb{R}. \quad (85)$$

If one has the complementarity problem $a \geq 0 \perp b \geq 0$, a solution $(a^*, b^*)$ is only a solution if and only if $\phi(a^*, b^*) = 0$. This may be proven by a case-by-case analysis of the signs of $a$ and $b$. Now consider the LCP

$$x \geq 0 \quad \perp y = Ax + b \geq 0 \quad (86)$$

where $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$ are given constants. Using the Fischer function the LCP may be reformulated as the nonsmooth root search problem

$$F(x) = F(x, y) = \begin{bmatrix} \phi(x_1, y_1) \\ \vdots \\ \phi(x_n, y_n) \end{bmatrix} = 0. \quad (87)$$

Thus, our problem is changed to that of finding the root of a nonlinear nonsmooth equation. This problem may be solved using a generalized Newton method which is an iterative method. In the $k^{th}$ iteration the Newton method solves the generalized Newton system

$$J \Delta x^k = -F(x^k) \quad (88)$$

for the Newton direction $\Delta x^k$. Here $J \in \partial F(x)$ is any member from the generalized Jacobian $\partial F(x)$. After having computed the Newton direction one performs a Newton update
to obtain the next iterate,
\[ x^{k+1} = x^k + \delta^k \Delta x^k. \]  
(89)

Here \( \delta^k \) is the step length of the \( k \)th Newton direction. A line search method will be used to determine the value \( \delta^k \).

We will briefly introduce some definitions and theorems from nonsmooth analysis. Let \( F: \mathbb{R}^n \rightarrow \mathbb{R}^n \) and let \( D \subseteq \mathbb{R}^n \) denote the set of all \( x \in \mathbb{R}^n \) where \( F \) is continuously differentiable. Assume \( F \) is Lipschitz continuous at \( x \) then the B-subdifferential of \( F \) at \( x \) is defined as
\[ \partial_B F(x) \equiv \{ H \in \mathbb{R}^{n \times n} | \exists x'(v) \subseteq D \text{ and } \lim_{x \rightarrow x'} \nabla F(x') = H \}. \]

Clarke’s generalized Jacobian of \( F \) at \( x \) is defined as the convex hull of the B–subdifferential [ClA90],
\[ \partial F(x) \equiv \text{co} (\partial_B F(x)). \]  
(90)

As an example consider the Euclidean norm \( e: \mathbb{R}^2 \rightarrow \mathbb{R} \) then for \( z \in \mathbb{R}^2 \setminus \{0\} \) we have
\[ \partial e(z) = \partial_B e(z) = \nabla e(z) = \frac{z}{\|z\|} : \forall z \neq 0. \]  
(91)

For \( z = 0 \) we have
\[ \partial_B e(0) = \{ y^T | y \in \mathbb{R}^2 \} \quad \text{and} \quad \| y \| = 1 \]  
\[ \partial e(0) = \{ y^T | y \in \mathbb{R}^2 \} \quad \text{and} \quad \| y \| \leq 1. \]  
(92a)

For \( z = [a \ b]^T \in \mathbb{R}^2 \) we write the Fischer function as \( \phi(a,b) = \phi(z) = e(z) - f(z) \) where \( f(z) = \left( \begin{bmatrix} 1 & 1 \end{bmatrix}^T z \right) \) is a everywhere continuous differentiable function. From this we find
\[ \partial \phi(z) = \partial_B \phi(z) = \nabla f(z) \]  
(93a)

Hence for \( z \neq 0 \),
\[ \partial \phi(z) = \partial_B \phi(z) = \left\{ \frac{z}{\|z\|} \cdot \left[ \begin{array}{c} 1 \\ 1 \end{array} \right] \right\} \]  
(94)

and
\[ \partial \phi(0) = \{ y^T \cdot \left[ \begin{array}{c} 1 \\ 1 \end{array} \right] | y \in \mathbb{R}^2 \} \quad \text{and} \quad \| y \| = 1 \]  
\[ \partial \phi(0) = \{ y^T \cdot \left[ \begin{array}{c} 1 \\ 1 \end{array} \right] | y \in \mathbb{R}^2 \} \quad \text{and} \quad \| y \| \leq 1. \]  
(92b)

The Clarke generalized Jacobian of the Fischer reformulation (87) can be written as
\[ \partial F(x) \equiv D_a(x) + D_b(x) A \]  
(96)

where \( D_a(x) = \text{diag} \{ a_1(x), \ldots, a_n(x) \} \)
\[ \text{diag} \{ b_1(x), \ldots, b_n(x) \} \in \mathbb{R}^{n \times n} \]
are diagonal matrices. If \( y_i \neq 0 \) or \( x_i \neq 0 \) then
\[ a_i(x) = \frac{x_i}{\sqrt{x_i^2 + y_i^2}} - 1, \]  
(97a)
\[ b_i(x) = \frac{y_i}{\sqrt{x_i^2 + y_i^2}} - 1, \]  
(97b)
else if \( y_i = x_i = 0 \) then
\[ a_i(x) = \alpha_i - 1, \]  
(98a)
\[ b_i(x) = \beta_i - 1 \]  
(98b)
for any \( \alpha, \beta \in \mathbb{R} \) such that \( \| [\alpha \ \beta]_T \| \leq 1. \)

Proof: Here we will only show the case for \( y_i \neq 0 \) or \( x_i \neq 0 \). The differential of the \( i \)th component is given by
\[ dF_i(x, y) = d \left( \frac{x_i^2 + y_i^2}{2} \right)^{\frac{1}{2}} - d (x_i + y_i). \]  
(99)

Using the chain rule we have
\[ dF_i(x, y) = \frac{1}{2} \left( \frac{x_i^2 + y_i^2}{x_i^2 + y_i^2} \right)^{\frac{1}{2}} - \frac{d (x_i + y_i)}{\sqrt{x_i^2 + y_i^2}} - \frac{d x_i - d y_i}{\sqrt{x_i^2 + y_i^2}} \]
\[ = \left[ \begin{array}{c} \frac{x_i}{\sqrt{x_i^2 + y_i^2}} - 1 \\ \frac{y_i}{\sqrt{x_i^2 + y_i^2}} - 1 \end{array} \right] \left[ \begin{array}{c} d x_i \\ d y_i \end{array} \right]. \]

Finally, let \( A_i \) be the \( i \)th row of \( A \) then we have \( d y = A d x \), so \( d y_i = A_i d x \), substitution of this result in
\[ dF_i(x, y) = \left( a_i(x) \phi_i + b_i(x) A_i \right) d x. \]
(101)

The case \( x_i = y_i = 0 \) follows from the previous examples.

We can choose any element in the generalized Jacobian. If \( x_i = y_i = 0 \) we could choose \( \beta_i = 1 \) and \( \alpha_i = 0 \). Thus, resulting in using the negative \( i \)th unit axis vector as the \( i \)th row of \( J \). A more practical implementation approach would simply consist in whenever \( x_i = y_i = 0 \) one would use \( x_i = x_i + \varepsilon \) in place of \( x_i \) when evaluating the generalized Jacobian where \( \varepsilon \) is a sufficiently small value.

A line search method is often used to achieve global convergence of the Newton method. We propose a backtracking line search with an Armijo condition to ensure sufficient decrease and that the chosen step length is not too small [NW99]. The line search uses the natural merit function of \( F(x) \) as a measure of convergence. The natural merit function is defined as \( \Psi(x) = \frac{1}{2} \| F(x) \| ^2 \). The Armijo condition is given by
\[ \Psi(x + \Delta x) \leq \Psi(x) + c \Delta x \]  
(102)

where the sufficient decrease parameter is \( c \in (0, 1) \) and
the gradient of the merit function is given by \( \nabla \Psi(x^k) = J^T F(x^k) \).

The objective of the line search method is to find a step length \( \bar{x}^k \) such that (102) is satisfied. The back tracking approach starts with the guess of \( \bar{x}^k = 1 \) and then test if (102) holds. If not \( \bar{x}^k \) is reduced by a step reduction fraction and the test is repeated. This continues until the test passes and one will have obtained the final value \( \bar{x}^k \).

5. Parallel Processing and Optimizations

Parallelization is an important topic since multi-core systems and massively parallel GPUs are very common today.

OpenMP (Open Multi-Processing) or MPI (Message Passing Interface) are often used for developing parallel applications for multi-core systems. OpenMP is designed for shared memory computers and provides a very simple and flexible interface for programmers. A programmer can use simple compiler directives in order to parallelize his code. MPI runs also on distributed memory architectures and can be used on a wider range of problems than OpenMP but it is harder to program. In general a MPI program consists of multiple processes that communicate by messages in order to solve a problem in parallel.

The parallel programming of GPUs is a far more complex task than programming a multi-core CPU. GPUs have a SIMD architecture since they were designed for rendering. The first parallel simulation methods on GPUs were implemented as shader programs which were executed in the render pipeline for each pixel or each vertex of a special scene. Such a scene had exactly the same number of pixels or vertices as required program executions. The data of the simulation had to be encoded as textures. The introduction of high-level languages for programming GPUs like OpenCL and NVIDIA’s Compute Unified Device Architecture [?, NV11] made General Purpose Computation on Graphics Processing Unit (GPGPU) more interesting for the community. A GPU program (also called kernel) can access different kinds of memories with different sizes and different performance characteristics. Therefore, the memory access and memory layouts play an important role for getting a high performance. OpenCL and CUDA provide access to global, local and shared memory. The interaction between CPU and GPU is also important since memory transfers between both are costly. The number of kernel calls also influence the performance significantly. Therefore, it is desirable to reduce the number of calls to a minimum. Since the parallelization on a GPU is not straightforward, efficient data structures and algorithms are required that are optimized for parallel rigid body simulations.

For the simulation of bilateral constraints one has to solve a system of linear equations (see Section 3.2). This can be done in parallel by using a solver like PARDISO [SG04] which is optimized for multi-core processors. Alternatively, there exist multiple methods for solving such a system on the GPU. Bolz et al. [BFGS03] as well as Krüger and Westermann [KW03] used shader programs and special textures to implement different parallel solvers on the GPU. Optimized data structures for sparse matrix operations on the GPU have been developed in [BG09] and [BCL09]. For achieving a high performance a good memory layout of these structures is very important. The optimized matrix operations allow the efficient solution of sparse systems which generally occur in multibody simulations with bilateral constraints. Another approach was presented by Bayer et al. [BBD09]. They create groups of independent constraints in a precomputation step. Then, all constraints in a group can be solved independently from each other. This is done in parallel using different pixel shader programs. The dependencies between the groups are resolved by a Gauss-Seidel iteration approach. A similar approach was used in [?].

For the computation of contact forces we have unilateral constraints in the simulated multibody system. Since we have inequalities in this case, the unilateral constraints cannot be solved by a linear solver. Therefore, the parallel computation of contact forces was also a research topic of interest in the last years.

Harada [Har08] used rigid bodies that are represented by sets of particles. This representation makes a parallelization of the collision detection and response very simple. For the collision detection each particle is represented by a sphere which results in an efficient detection due to a very simple collision test. The accuracy and the performance of the collision detection directly depend on the resolution of the particle representation. For the collision response Harada used a discrete element method (DEM) where a repulsive force, a damping force and a shear force are computed for each colliding particle.

Tasora et al. [TNA08, TNA10] used a Cone Complementarity Problem (CCP) formulation instead of a classical LCP solver with a polyhedral approximations of the friction cone in order to parallelize the contact problem. One
of the challenges when working on the GPU is to avoid concurrent updates of shared data. Due to the high latency on GPU memory access, global atomic operations can be computationally costly. Tasora et al. argued that the probability for a concurrent velocity update of contacts associated with the same body is very small for large scenarios with hundreds of thousands of contacts. Harada showed how to efficiently solve this problem by partitioning, synchronizing, and scheduling the operations using local atomics within each compute unit. An open source implementation as shown in Figure 12 is available as part of a rigid body simulation pipeline running entirely on the GPU for the Bullet physics engine.

Courtecuisse and Allard [CA09] introduced a parallel Gauss-Seidel iteration method for dense matrices. Their method works on multi-core processors and GPUs. It maintains the invariant that in each block row, the diagonal element is the last to be updated. This is used to schedule the block computations, eliminating the need for global synchronization.

The research in the area of parallelization shows us that the performance of simulations can be increased significantly taking advantage of multi-core processors and GPUs. But this performance gain is not achieved straightforward, it demands a computational rethinking of the used algorithms.

6. Collision Detection for Rigid Body Dynamics

Collision detection provides important information used by rigid body dynamics. We briefly discuss the most relevant shape representations, collision detection queries, and contact generation methods. A more complete overview of the field is available in the collision detection surveys [LG98, JTT00] and books [Ber04].

6.1. Shape Representations

The geometry type of simulated rigid bodies is important for collision detection and contact point generation. It significantly influences the performance and the complexity of the simulation system. This section will discuss the role of geometry in simulations.

Structured polygonal models are very popular in the graphics community. There exist many tools and efficient algorithms for this type of geometry. These models are also popular in the field of interactive rigid body simulations. The mass properties of a polyhedral body can be determined fast and accurately [Mir96a]. Furthermore, there exist different very fast collision detection methods which only work for closed convex polygonal meshes.

Since fast collision detection methods are essential for an interactive simulation, the usage of convex polygonal models is often required in this area. Therefore, non-convex shapes have to be decomposed in convex parts in a precomputation step (see Figure 13). There are several ways to decompose a closed non-convex polygonal mesh into a convex decomposition. The decomposition can be generated either manually or automatically [MG09].

Non-moving concave world geometry is often represented as a concave triangle mesh, and collision queries are performed on individual triangles. As mentioned before, midphase acceleration structures can be used to cull most triangles.

6.2. Collision Queries

The exact collision queries between two objects are known as narrow phase collision detection. The choice of algorithm and complexity of the query depends on the collision shape representation of the objects involved (see Figure 14). Aside from the shape type, we can classify queries into discrete and continuous queries. Discrete methods perform the collision check at a specific time instant, while continuous collision detection (CCD) methods take the motion of the objects into account over a time interval.

Discrete Collision Detection Various discrete collision queries exist, ranging from a simple intersection test to full
contact information generation. A discrete intersection test produces a boolean result that determines whether collision shapes overlap or not. When using a simulation loop with an adaptive timestep, the intersection test can be used to search for the time of impact using a technique called bisection.

When objects are separated by a positive distance, we can compute this closest distance and the closest points, also known as witnesses. The GJK algorithm \cite{GJK88,Ber04} is versatile and it has been used for different queries between convex shapes. GJK can be used to perform an intersection test, and when objects are separated by a positive distance, it can compute the distance and the corresponding closest points (one on each body).

If objects are overlapping, we can compute the penetration depth. A common way to define penetration depth is the shortest relative translation of the objects to eliminate the overlap. In addition to the penetration depth vector, we can compute witness points on both objects where the object will touch. The penetration depth between general convex shapes can be computed using the expanding polytope algorithm \cite{Ber04}, while the separating axis test (SAT) can be used between convex polyhedra. Concave shapes can be represented as a union of convex shapes \cite{See61}. Alternatively, a collision check is performed for each triangle in a concave triangle mesh (or triangle soup).

A single contact point pair is often not sufficient for stable resting contact in rigid body dynamics. Section 6.3 will provide more information about contact point generation.

**Continuous Collision Detection** Discrete collision checking algorithms can fail to detect a collision due to temporal aliasing, which commonly occurs with fast moving or small objects that can pass completely through an object in one time step. To avoid this, we can take the motion into account for a certain time interval and compute the time of impact. This information can be used to subdivide the simulation timestep, or it can be used to formulate contact and distance constraints to prevent penetration \cite{See61}.

Mirtich \cite{Mir98} determined a lower bound for the time of collision for each pair of bodies. These times are stored in a heap which has to be updated after each collision since the bodies then change their motion. The minimum collision time of a pair of bodies describes how long a simulation can run at least without a penetration occurring. The detection is accelerated by bounding volumes which take the ballistic motion of the bodies into account.

Continuous collision detection methods approximate the motion of the bodies during a time step. The collision detection is performed on the resulting trajectories. Different methods were introduced to perform the motion approximation for unconstrained bodies, which are based on interpolation \cite{RKC00,RKC02, KR03}. In contrast, Redon presented a continuous collision detection which is designed for articulated bodies \cite{RKLM04,ZRLK07}. Continuous collision detection has the advantage that no collision is missed but at the price of a higher computational complexity.

### 6.3. Contact Point Generation

The quality of the contact point information greatly influences the overall robustness and stability of an interactive simulator. Since all subsequent contact force and stabilization computations are affected by the quality of the contact point information, the requirements of the method for contact point generation are: high performance, robustness and consistency.

Collision detection methods like the GJK algorithm often return only one pair of points representing either the minimum distance or the maximal overlap distance. For accurate collision resolution and stable resting contact handling between rigid bodies, we need more than a single contact point in general. In practice, post processing is often done to generate the complete contact region between objects. However, having too many contact points between two rigid bodies can cause performance and stability issues.

Feature based contact point generation was among the first approaches \cite{Bar90}, it has since been extended to cover continuous collision detection \cite{SMT08}. Basically, contacts are represented by either edge-edge or vertex-face feature pairs. Since edge-face feature pairs correspond to penetrations, these are used when objects are overlapping. Feature based contacts are local definitions and might show inconsistences on a global scale. Further, some methods have a tendency to generate redundant feature pairs \cite{CTM08}. Collision envelopes are used to ward off numerical imprecision, round-off and truncation errors in floating point arithmetic.

When using the feature based contact point generation, one method needs access to the actual features of the mesh representation.

Another approach for generating a contact region is to track contact points over time. In each time-step, new contact points are added to a region while filtering out old ones that no longer agree with the current contact plane. Contacts can be tracked based on features between convex polyhedra \cite{Mir98}. Feature information is not always available, so to track contacts between general convex objects a heuristic based on the distance between closest points can be used \cite{Cou05}.

For convex polyhedra an approximation of the entire contact region can be computed at once. In the case of two colliding bodies, first, a separating plane is determined. The collision geometry can then be clipped and projected onto the separating plane. The contact region is determined by intersecting the resulting convex polygon in the plane \cite{BG10}.

Signed distance fields have been popular for deformable models \cite{MAC04}, cloth \cite{BMF03} and rigid bodies \cite{GBF03}. According to Erleben \cite{Erl05}, signed distance fields add a
certain smoothness to the contact point generation, which avoids many of the difficulties in choosing contact normals and computing penetration depths. On the other hand, the smoothness is related to the resolution of the distance field and can be troublesome for stacking configurations. The memory footprint can make signed distance fields intractable for interactive simulations. Other approaches using discrete Voronoi diagrams also exist [SGG+06], these tend to be similar to signed distance fields methods, apart from using a Voronoi diagram as the basic representation.

A general issue in contact point generation, is choosing between global and local solutions [KOLM02]. In theory, contact point generation is a global issue, however, in practice the local solution is often used to satisfy performance considerations. There is also some discussion on how a good penetration depth measure is defined. For convex polytopes the generalized penetration depth is same as the translational penetration depth [ZKVM06].

7. Rigid body dynamics in practice

The literature on rigid body dynamics, robotics and contact mechanics are vast, cross-disciplinary and have a long history. Thus, several attempts have been made in the past to classify previous work in order to make differences more clear to the communities. In this section, we shortly review some of the terminology that has been used in the past.

Furthermore, we want to give an overview over existing commercial and open source simulation software as well as a classification for this software. We also want to give a survey of benchmark papers in computer graphics and discuss the common practice regarding benchmarking and validation of simulators.

7.1. The Simulation Paradigms and Contact Models

In the terminology of Moreau [Mor99] one may classify simulation methods as being event-driven, smoothing, or contact dynamics approaches. Event-driven approaches are classified by models where motion in between events are assumed to be sufficiently smooth and not changing too much. This can be understood in the sense that the contact regions between objects are non-changing and reaction forces do not change direction. Changes then only occur at specific single events in time and must be dealt with specifically at these events. An example of such a method could be one that assumes that interactions only consist of instantaneous collisions. Like the impulse-based model of Hahn [Hah88], and Mirtich and Canny [MC95, Mir96b]. Here it is assumed that objects are in free ballistic flight in between collisions. Another example is the acceleration-level based formulations like the ones in Baraff’s work [Bar89, Bar90, Bar93a, Bar93b, Bar94, Bar95, Bar96]. Here, the equation of motion is formed and treated as second order ordinary differential equation – assuming the motion is continuous differentiable. Discontinuous instantaneous changes in velocities and accelerations must then be treated at specific events. Smoothing approaches essentially apply some kind of regularization, like replacing a nonsmooth non-penetration law by a stiff repulsion law. The smoothing can be applied both in time and space. For instance in the work by Moore and Wilhelms [MW88] springs are cleverly used to model both sustained contact as well as instantaneous collisions. Contact dynamics approaches are described by Moreau as time stepping algorithms that determine the evolution of the velocity function by applying the principles of dynamics and assumed force laws. This means that no concept of acceleration is needed and the detailed dynamics over a single time-step is treated and resolved in a one step manner. The later work by Stewart and Trinkle, Moreu, and Jean [ST96, Mor99, Jes99] are examples of velocity based formulations that apply a fixed time-stepping procedure to advance the simulation state. Recently, this type of methods is simply referred to as time-stepping schemes [Stu08].

Event-driven approaches are often not the preferred choice for interactive simulation. The reason being that it can be highly unpredictable how many events need to be processed before reaching the next frame in ones simulation. Thus, sometimes ones simulator appears to be fast and at other times it may even stagnate. For configurations with a lot of dynamic and fast moving objects that bounce around an event-driven approach can be very efficient. However, for large piles or stacks of objects undergoing some transient salient motion the rate of events can explode and stagnate the simulation. This is one of the reasons why fixed time-stepping schemes are preferred as they always take one step ahead in time no matter what the interaction is. The computational cost of a time-stepping scheme often scales in the number of constraints. However, with iterative methods this scaling can be as fast as linear and often the number of iterations can be bounded yielding a fast simulator with a highly reliable predictable performance. The smoothing approach bare some similarity to the penalty-based paradigms which we cover later and suffers from the same difficulties.

Baraff [Bar93b] applies the terms Continuous methods (originally Baraff termed this continuum methods but community seems to have converged on the term continuous collision detection [vdB05,ZRLK07]) and discrete methods for dealing with the numerical time aspect of collision detection and contact point generation methods. The discrete setting can be thought of as taking a photograph and compute all geometric and physical information from that time instant. In such an approach one really does not know what occurs between two consecutive discrete points in time. The continuous methods on the other hand resolves what occurs over time intervals. Baraff describes two groups of simulation methods for dealing with constraints. One is termed constraint-based methods and the other is termed the penalty methods. In the first group constraint forces are solved for
analytically such that they exactly fulfill the constraints of the system. In the second method constraints are rephrased as penalty functions in an optimization sense [NW99]. Later Baraff [Bar94] adopted the term analytical methods to describe methods that compute contact/constraint forces in an analytical setting (like solving a linear system of equations or an linear complementarity problem) that fulfills the imposed constraints. These terms are essentially a classification of which type of numerical method that is used to find a solution for a system of constraints whether that is expressed as a linear system of equations or a more complex mathematical formulation like a complementarity problem formulation. More recently people make the distinction between direct methods and iterative methods [Stu08, KSJP08, BDCDA11]. Again, this is a classification of the numerical method applied.

Early work tended to use direct methods based on pivoting for solving complementarity problem formulations [Bar94, ST96, AP97b]. For interactive simulations it was quickly recognized that these type of numerical methods scaled too poorly although they were accurate. To deliver a fast performance that scales well iterative methods have been employed. In particular Gauss–Seidel like methods [Mor99, Jea99, Er107, Stu08, CA09] have been investigated. Proper exploitation of matrix factorizations allow these type of iterative methods to scale linearly in the number of constraints. The poor convergence of Gaussian–Seidel type solvers have been countered by Newton-type algorithms [AC91, Ort07, EO08, SNE09, BDCDA11] that offer a theoretical second order convergence rate over the linear rate of Gaussian–Seidel type solvers. Linear scaling can be obtained for Newton-type methods resulting in Quasi-Newton methods this sacrifices the convergence rate though. A well-known Newton type solver is the PATH solver [FM99, Pat05]. One downside of PATH is that it needs a global coefficient matrix and for that reason it scales quadratic in the number of constraints.

Other authors refer to models that are based on the dynamics and assumed force laws formulated as constraints as constraint-based paradigms and make the distinction of whether they are formulated on a position, velocity or acceleration based level [ST96, AP97b, CR98, MS01, KEP05, Er105, TNA08, KSJP08, TNA10, BDCDA11]. This type of paradigm shares some similarity traits with the contact dynamics approaches of Moreau and the constraint-based methods of Baraff. Constraint-based paradigms are often further subdivided into being maximal or reduced coordinate formulations. This refers to whether knowledge of joint constraints are used to remove unneeded degrees of freedom from ones system of equations. If such action is taken, a reduced formulation is created containing a smaller number of variables, hence the term “reduced”. Maximal coordinate formulations on the other hand do not reduce the number of variables but rather keep joint constraints as an extra set of equations that must be fulfilled. The work of Armstrong and Green, and Featherstone [AG85, Fea87] are examples of numerical methods that in a recursive manner very efficiently finds solutions to a reduced coordinate formulation whereas Baraff [Bar96] is an example that works with a maximal coordinate formulation where sparsity pattern of the first-order optimality conditions (known as the KKT-matrix [NW99]) is exploited to find solutions for Lagrange multipliers in linear time. The sparsity pattern arises from tree-like jointed mechanism. The term penalty-based paradigms seems to have converged on the meaning that some type of repulsive force is used to penalize violations of constraints or penetrations. Recent work tend to compute penalty forces based on volume violation, i.e. the actual overlapping volume [AFC10]. In the same spirit impulse-based paradigms refers to models that approximate continuous contact with a series of instantaneous contacts [Mir96b, Mir00, GBF03]. Thus, the above simulation paradigms each classify the underlying model of the physical interaction as being based on penalty forces, collision impulses (i.e. instantaneous impacts) or some simultaneous mathematical formulation of the whole system.

Penalty-based paradigms are notoriously hard to work with, since it requires extensive parameter tweaking to perform optimally. Physical plausibility is hard to achieve with this paradigm, in part because collisions are never solved exactly, making stable stacking nearly impossible to simulate. The impulse-based paradigms is simple to implement, however, stable stacking is often difficult to achieve. This has been improved upon in later work with a technique of shock-propagation [GBF03]. The constraint-based paradigm has become the paradigm of choice in many interactive rigid body simulators [Sm00, Cou05] as it offers both great control and stability.

Maximal coordinate formulations are in computer animation dominated by complementarity formulations. There exist alternatives on kinetic energy [MS01] and motion space [RKC03]. However, the former solves a more general problem but is not attractive for performance reasons, and the latter is of limited use for realistic animation since it does not include friction. Recently Kaufman et al. [KEP05] presented a velocity-based method using projections onto convex subspaces of feasible velocities. The authors used a contact model, which is based on limit surfaces and principle of maximum dissipation [GRP89] , together with an ad-hoc model for bounciness and an approximation of momentum conservation. Kaufman et al. [KSJP08] also explored an iterative staggered approach for solving a velocity-level linear complementarity problem for contact problems by splitting the solver iterating into a normal force only solve followed by a friction only solve phase. Neither the 2005 nor the 2008 work was for interactive simulation. Recently, Newton type methods have been explored [BDCDA11] which apply a blocking strategy for solving the Newton system. The model is very similar to original work by Alart and Curnier [AC91]. The focus in this work is contact problem for hair and not rigid bodies, in fact the solver has problems dealing with the often overdetermination.
and large mass ratio properties encountered in rigid body dynamics. Complementarity formulations come in two flavors: acceleration-based formulations [Bar94, Bar95] and velocity-based formulations [ST96]. Acceleration-based formulations cannot handle collisions, and one must stop at the point of collision and switch to a impulse-momentum law [BWAK03, PW96, AP97b, Cha99]. Further, acceleration-based formulations suffer from indeterminacy and inconsistency [Ste00]. Although mostly overlooked in the computer graphics literature, the velocity-based formulation suffers from none of these drawbacks.

One other way to classify methods is by examining the underlying assumptions applied in their models of contact. For instance many impulse-based simulators apply a sequential (or propagating) contact model. Here, a local contact model of what happens during an instantaneous collision at a single point of contact between two rigid bodies is applied in a one-by-one sequential manner. Whereas many complementarity-based formulations take a more global view and use a simultaneous contact model that describe how the dynamics is coupled through multiple contacts between multiple objects [PG96, CR98, Mos07]. Impulse-based methods are examples of sequential models whereas complementarity problem formulations are simultaneous models of contact. One can further distinguish a contact model as being a hard (also called nonsmooth) or smooth contact model. The complementarity constraints used for non-penetration constraints are examples where a hard contact model is applied whereas the penalty force formulation is an example of an application of a smooth model. Of course one may add compliance or regularization to a hard contact model making it more smooth. It is often easier to model propagation/wave effects using local contact models or soft contact models whereas nonsmooth constraints often rely on simultaneous contact models and disregard any kind of propagation/wave effect.

A real-life working rigid body simulator is often not limited solely to one type of paradigm or contact model. Often things are combined in an ingenious and careful manner. Each piece of a simulator combats different artifacts and helps ensuring robustness and stability of the simulator. For instance position-level constraint-based formulation may be used to compute projections of rigid bodies in order to remove penetration errors or a penalty-based paradigm may be used to stabilize discretization errors of the ordinary differential equations as they evolve the system state in time. For example joint drifting in maximal coordinate formulations are countered through stabilization terms acting much like penalty forces. Compliance is added to nonsmooth contact models by adding a penalty force term making pure rigid bodies behave as quasi-rigid bodies.

For interactive simulators some common trends appear to be velocity-based constraint-based paradigms using fixed time-stepping methods. Reduced coordinate formulations are also widely popular for character animation as these by “design” do not visually appear to suffer from discretization errors and usually can deal with high speed moving limbs of a character.

7.2. Commercial and Open Source Software Solutions

There exist many open source alternatives like Bullet, Open Dynamics Engine (ODE), Newton Game Dynamics, daVinci code (dVC3d), Dynamo, dynamY, LMGC90, Jingine, Box2D, OpenTissue, IBDS as well as more commercial alternatives such as Vortex from CMLabs, PhysX from NVIDIA, Havok or Algoryx.

Most of these are multi-purpose physics engines and usually implement several methods: several different constraint solvers, friction models, collision detection method etc. ODE for example has both a pivoting/direct method and iterative methods such as blocked projected Gauss–Seidel (PGS). Bullet have similar iterative methods and is exploring nonsmooth nonlinear conjugate gradient methods as well. Recently many has focused on using GPUs in their simulators such as ChronoEngine, SOFA, PhysX and Bullet. The algorithmic choices are still based on iterative methods in these works.

Most game physics engines use a constraint solver based on iterative methods like the ones known from ODE and Bullet. The methods are confusingly named “sequential impulses” by the computer gaming community but the models are based on the constraint based paradigm and are solved using an iterative method such as PGS or similar. Although the constraint based paradigm has rooted itself as a very dominant method there are examples of other types of rigid body simulators. SIMPACK is such an example using a penalty based paradigm based on computing contact forces from volume overlaps. One difficulty of the penalty based approaches is often that they have many parameters and stability can be hard to come by at times this makes one abandon using physical meaningful parameter values. Examples of the impulse based paradigm and reduced coordinate method may be found in DynaMech.

The robotics community have made good use of rigid body simulators and large open source simulation frameworks such as Gazebo or Weebots are build on top of the ODE simulator. Commercial alternatives also exist like Microsofts Robot Simulator that utilizes the PhysX engine. The frameworks offer a large library of existing joint and motor models as well as different kinds of controllers. Thus, allowing robot designers to test ideas and do off line programming of their control algorithms. Very detailed contact simulations can be done with simulators such as Adams from MSC software. These are often based on finite element methods and penalty based paradigms and are often not even near to interactive simulation.

Many authoring tools like Blender, Maya, Cinema 4D, LightWave, Houdini, 3ds Max, Autocad etc. offer rigid body
J. Bender et al. / Interactive Rigid Body Simulation

simulation as their features. Most of such tools even offer a variety of plug-ins allowing users to pick and choose between which rigid body simulator they wish to use. Render engines like Ogre3D provide plug-ins for most of the popular rigid body simulators. The need for being able to switch one simulator for another has motivated initiatives such as COLLADA which is an XML scheme for describing rigid body physics in digital content and open PAL (Physics Abstraction Layer) which offers a common application programming interface for many rigid body simulators.

7.3. Benchmarking and Validation of Simulators

Few examples of benchmark tests and simulator comparison tests exist in the computer graphics literature [LH00a, LH00b, AS06, BB07, WM09, WSM∗10]. These limit themselves to case-by-case studies comparing simulated results against analytical results or by feature list comparisons. No benchmark databases exist with both simulation setups and ground truth data. Looking at experiments and results from research papers several tendencies about common practice can be extracted. In summary, computer graphics rarely perform temporal convergence studies, rather focus is on creating stable and robust simulators that work with large time steps. Thus, there is little point in examining what happens if the time-step size goes to zero. In a similar fashion constraint errors are rarely examined in detail. From animation viewpoint this can be justified as a simulation may appear plausible as long as any constraint error is less than the size of a pixel and thus can not be seen by the naked eye. Computer graphics tends to care more about generality and robustness of a simulator than whether it can accurately reproduce a contact force to some given numerical precision. This is often exemplified by showing several simulation results from production environment like scenarios taking robustness and generality to the extreme. These are of course over-simplifying statements that would not be valid in all application areas of interactive rigid body simulation but holds for many cases of entertainment where focus is more on creating interesting motions than correct motions.

Statistics Many works settle with reporting the number of bodies, constraints, contact points and the frame time for selected test cases [Bar94, GBF03, WTF06, WGF08, SSF09]. Some work report more detailed wall-clock times as a function of frames [KEP05]. Others investigate convergence rates of iterative solvers [Lac03, Erl07]. Often single test cases are constructed showing how a simulator can deal robustly with different types of simulation [Mir96b, AS06, BB07]. In our view for interactive simulation it is often of interest not only to know average numbers, but also minimum, maximum and variance are important to figure out the range of applications a simulator can be used for.

Accuracy and Error Correction Piles of objects are very popular, and stable stacking like towers or card houses [GBF03, KEP05, WTF06, Erl07, KSJP08, AFC∗10]. Examples are shown in Figure 15. Some works address scaling by increasing the number of objects or joints and reporting how computation time is affected by this [MS04, Ben07]. Some papers introduce test cases like a block sliding on an inclined plane which allows one to validate the friction law or visually identify creeping problems [Mir96b, SSF09]. Dense structured stacks are very good for verifying the accuracy of contact force computations and to determine if error correction is working properly [Erl07]. Dense stacks usually severely suffer from constraint overdeterminacy (i.e. redundancy in contact point information) and therefore stress the numerical methods. Iterative methods often deal very well with overdeterminacy whereas other methods suffer from the singularity that appears in the coefficient matrix due to the overdeterminacy. Static structures like the card house or dry stone masonry structures like a stone arch are excellent for testing a simulator’s ability to accurately reproduce static friction. In real-life it is basically static friction that holds these kind of structures in place [Erl07, KSJP08]. Dominoes falling down is also often used to test if friction makes the dominoes come to rest [Mir96b].

Proper Event Handling Dynamic examples include the see-saw which is great for stressing event-driven approaches and showing impulse propagation [Mir96b]. The Newton
cradle (see Figure 16) can be used to determine whether a simulator’s underlying model is a simultaneous contact model or a sequential contact model. Falling dominoes are often used as a test case for event-driven schemes [Mir96b]. Destruction of buildings as shown in Figure 17 can be helpful to test if sleeping policies (freezing) is too aggressive or even if it is present [Erl05]. Configurations with an in-build designed jam (like billiard balls in a racket) can be used to test whether simulators terminate in case they use a sequential contact model or if they can deal gracefully with errors and symmetries.

Numerical Methods Examples of objects being wedged have a tendency to stress the numerics. In particular if the setup is created in an illegal state. These types of tests mimic the effect of a real-life user doing “bad” things. The tests are thus helpful in determining if an interactive simulator is robust and stable. Large mass ratio tests are extremely good at stressing the numerical properties of simulators like a heavy box placed on top of a light box [SNE09]. Structured stacks exhibit some of these mass-ratio problems as the bottom-most objects feel the weight of all objects lying on top of them. Most iterative methods have difficulties dealing accurately with large mass ratios within a limited time budget or they converge so badly that even infinite amount of iterations won’t help. Thus, large mass ratios are well suited as worst-case scenarios for the numerical methods.

Joints Vehicles or spinning wheels during a turning motion are stressful for most maximal coordinate formulations as turning the high speed wheel axis can make wheels numerically fly off. This is caused by discretization errors that are enhanced by applying large-time steps and using infinite orientations. Most engines have special fixes for this well-known problem. To stress joint drifting errors in maximal coordinate formulations large mass objects can be connected through joints to an articulated figure that is pinned down. Much like a prisoner in a jail with a ball and chain attached to the legs.

Gyroscopic Forces The tippe top and rattleback (see Figure 18) are famous test cases for testing if gyroscopic forces are working as expected in a simulator. Many game engines have a tendency to simplify the gyroscopic forces as they are a major cause to numerical stability problems prohibiting large time-steps. Thus, a case such as the tippe top can be used for identifying these problems.

Interactivity Performance is critical to achieve interactive interaction. Thus, scalability testing is very interesting to see how large problems one can deal with at interactive rates. Robustness in interactive applications is more difficult to verify. Goal oriented tasks like stacking objects and tipping the stack without making it fall down have been used for simulators running under extreme conditions to test robustness towards user interaction as shown in Figure 19 and Figure 20.

Generality and Robustness Often simulations of complex scenes or interactions like interlocked gears or moving belt tracks are used to demonstrate generality and robustness of a simulation method [WGF08, SNE10a]. Complex geometry is often of interest in particular from a production viewpoint where concave surfaces and/or sharp features of the geometries may be encountered. However, from a contact mechanics viewpoint most contact laws being used are planar thus computing any contact forces at any given discrete instant in time is completely independent of the curvature of the surface. It is only when time-stepping comes in play that things may go wrong as surfaces may have higher “order” than the contact laws and time-integration methods being used. For fixed-time-stepping methods it is often not meaningful to use higher order integration schemes due to the fact that the contact laws are planar. Letting a box slide down a curved slope can be a good test case for illustrating these kind of problems. Some illustrations of complex scenes are shown in Figure 21.
Physical Correctness Time plotting mechanical energy as well as kinetic and potential energy can help analyze physical correctness of ones simulator. Time plots of linear and angular momentum can be useful in this manner [AS06]. This allows one to validate if the physical conservation laws are fulfilled. One may test physical laws from simple physical test systems. For instance for impact laws one may test the proper effect of setting the coefficient of restitution. Sliding friction can be validated using for instance a box sliding down a plane. Inertia can be tested by spinning objects and verify if the proper axis of rotation is reached. In many cases one can from sufficient simple tests derive analytical results to compare against simulated results. For interactive simulators it can be quite useful to compare against a non-interactive but very high-fidelity simulator to confirm that similar simulation results are obtained.

Performance and Numerical Properties Many works list performance measurements as average frame times or frames per second achieved for a small portfolio of test cases [Bar94, GBF03, WTF06, WGF08, SSF09]. There are examples of papers doing complexity analysis and comparing against scalability studies, i.e. plotting computing time as a function of problem size [RGL05, PNE10, SNE10b]. For multi-core approaches speedup factors and floating point operations per second are plotted [CA09]. It is however not always clear what is the base reference that is compared against. In our view best practice should be an optimal tuned state-of-the-art solution. Detailed time-measurements of each sub-part of a simulator can be very helpful in analyzing performance bottlenecks too [KEP05]. Convergence plots of iterative methods are very helpful for determining a solvers ability to converge fast or slow [Lac03, Erl07]. Investigating how convergence error changes as a function of parameter choices (parameter studies) is a very methodical approach to find values suitable for robust interactive applications [BD- CDA11]. Others compare against state-of-the-art competing methods [KSJP08, SNE10b].

Perception The saying that in computer graphics if it looks good then it is good is often stated by simulation people. The question is how one can quantify and measure if this is true. Although the literature is sparse, perception and user sensitivity studies have been performed [OD01, ODGK03, RP03, NLB07, RO09]. Among other things these studies have shown that users find it hard to detect abnormalities in spinning objects and more difficult to detect abnormalities in collisions between complex objects than between simpler objects such as spheres. From these studies perception error metrics are developed. In principle such metrics could be used to validate if ones simulation results looks “good”. However, simulation papers rarely do so.

8. Conclusion and Future Work
Interactive rigid body simulations have become an important part in different application areas. Such a simulation requires efficient and accurate methods for handling joint and contact constraints as well as a fast collision detection.

The simulation of more complex scenes and improvements of accuracy are current goals in this field. To reach these goals massively parallel GPUs and multi-core processors are taken into account. This parallelization trend requires a computational rethinking and provides the possibility to develop new efficient algorithms.

In the last years much research has been done on coupling of rigid body simulations with other animation and simulation techniques. One topic in this area is the combination of techniques like inverse kinematics with rigid bodies. Another important one is the coupling of rigid bodies with fluids [CMT04, RMSE08, RMEF09], cloth and deformable bodies [SSEF07, SSF08]. Coupling allows the usage of different kinds of bodies in the same simulation environment by simulating the interaction between these bodies.

Simulation is a good way to generate realistic looking animations. But compared to keyframe techniques there is one big drawback. The results of a simulation can only be controlled indirectly by manipulating simulation parameters or adding forces to the system. Many physical parameters have
to be defined for a simulation. It is hard to reach certain predefined goals just by tweaking these parameters. Therefore, more control over the simulation is required. In order to solve this problem different methods have been developed which give a high-level control to the user. Some works propose inverse dynamics methods [PSE03, TJ08], others perform multiple simulations and discard unfitting ones [TJ07]. These methods let the user sketch a desired motion or define specific goals which must be reached by the simulation. But controlling the simulation is still a problem where much work has to be done.

References


© The Eurographics Association 2012

J. Bender et al. / Interactive Rigid Body Simulation


[BS06a] Bender J., Schmitt A.: Constraint-based collision and contact handling using impulses. In Proceedings of the 19th international conference on computer animation and social agents (Geneva (Switzerland), Switzerland), July 2006, pp. 3–11. 19


[Cha99] Chatterjee A.: On the realism of complementarity conditions in rigid body collisions. Nonlinear Dynamics 20, 2 (October 1999), 159–168. 32

[CK00] Cash J. R., Karp A. H.: A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. ACM Transactions on Mathematical Software 16, 3 (1990), 201–222. 21


[Cou12] Coumans E.: Coumans experiments. Published online at https://github.com/erwincoumans/experiments, January 2012. Open source experiments and research for the Bullet physics engine. 28


© The Eurographics Association 2012.
Interactive Rigid Body Simulation


© The Eurographics Association 2012.


© The Eurographics Association 2012.