Workshop in Virtual Reality Interactions and Physical Simulation VRIPHYS (2008)
F. Faure, M. Teschner (Editors)

Corotated Finite Elements Made Fast and Stable

Joachim Georgii[1] and Rüdiger Westermann[1]

[1]Computer Graphics and Visualization Group, Technische Universität München, Germany

Abstract

Multigrid finite-element solvers using the corotational formulation of finite elements provide an attractive means for the simulation of deformable bodies exhibiting linear elastic response. The separation of rigid body motions from the total element motions using purely geometric methods or polar decomposition of the deformation gradient, however, can introduce instabilities for large element rotations and deformations. Furthermore, the integration of the corotational formulation into dynamic multigrid elasticity simulations requires to continually rebuild consistent system matrices at different resolution levels. The computational load imposed by these updates prohibits the use of large numbers of finite elements at rates comparable to the small-strain finite element formulation.
To overcome the first problem, we present a new method to extract the rigid body motion from total finite element displacements based on energy minimization. This results in a very stable corotational formulation that only slightly increases the computational overhead. We address the second problem by introducing a novel algorithm for computing sparse products of the form RKR^T, as they have to be evaluated to update the multigrid hierarchy. By reformulating the problem into the simultaneous processing of a sequential data and control stream, cache miss penalties are significantly reduced. Even though the algorithm increases memory requirements, it accelerates the multigrid FE simulation by a factor of up to 4 compared to previous multigrid approaches. Due to the proposed improvements, finite element deformable body simulations using the corotational formulation can be performed at rates of 17 tps for up to 12k elements.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction and Related Work

Fast and reliable methods for predicting shape changes of deformable 3D bodies under load are of major relevance in a number of computer graphics applications like computer animation and virtual surgery simulation. In the past decades, three-dimensional finite element (FE) analysis was performed in a number of approaches to predict the mechanical response of deformable materials. FE methods are attractive because they can realistically simulate the dynamic behavior of such materials. Most commonly they have been used to simulate isotropic linear elastic materials based on Hooke's law, with the inhomogeneous Young's modulus E being defined on a per-element basis. A thorough overview of the state of the art in the field can be found in [NMK*05], which also discusses alternative approaches based on finite differences [TPBF87, TF88] or mass-spring systems [LTW95, BW98, DSB99], as well as multiresolution approaches using adaptive refinement techniques [CGC*02, DDCB01, GKS02].

FE methods for simulating elasticity effects, on the other hand, are numerically involved and not suited in general for usage in real-time applications. This is especially due of the non-linear relationship between strain and displacement, yielding a non-linear algebraic system to be solved [ZC99, WDGT01, PDA01, DDCB01, ML03]. Therefore, the linearized strain tensor, i.e. the Cauchy strain tensor, is commonly used in such applications [BNC96, CDA99]. While this approximation is appropriate for small deformations, it leads to non-realistic results if elements undergo large deformations. In particular, as the Cauchy tensor is not invariant under rotations, the strain distribution and thus the final element displacements tend to diverge from the correct solution (see Figure 1).

A good trade-off between the Cauchy and the full non-linear strain tensor can be achieved by using the so-called

Figure 1: *On a desktop PC, the tetrahedral grid consisting of 67k elements (left image) was interactively deformed at 3 tps using the corotational finite element formulation (middle image). Additionally, the same model was deformed using the small-strain formulation at 17 tps (right image). The artificial volume increase of the mouth can easily be perceived.*

corotational formulation of finite elements. It is a linear approximation just as the Cauchy strain, but it accounts for the geometric non-linearity by respecting per-element rotations in the strain computation. In the corotational formulation the motion of every element is split into a rigid body motion with respect to the element's initial configuration, and a deformational motion in the so-called corotated configuration. The use of the principle of corotational coordinates in FE analysis goes back to the early work of Wempner [Wem69] and Belytschko and Hsieh [BH79]. The element-independent corotational formulation underlying our approach, where per-element rigid body motions are removed and added before and after the finite element analysis, has been introduced by Rankin and Brogan [RB86]. For a thorough overview on this concept we refer the reader to the summary paper by Felippa and Haugen [FH05].

In the context of FE modeling the corotational formulation was introduced to the computer graphics community by Müller and co-workers [MDM*02, MG04], Etzmuss et al. [EKS03] and Hauth and Strasser [HS04], who proposed two different approaches for extracting the rigid body motion of triangular and tetrahedral elements. While in the former approaches this motion was computed purely based on geometric considerations, in the later approaches a polar decomposition of the per-element deformation gradient was employed. For large element deformations, however, both formulations may introduce numerical instabilities. While in the geometric approach such instabilities can be contributed to the uncertainty in determining the corresponding rigid body mode from a largely deformed element, numerical instabilities in the computation of the deformation gradient in largely deformed elements can become a problem otherwise. Furthermore, an approach based on a QR decomposition has been developed [NPF05], but it introduces vertex-ordering dependent anisotropies. To improve the robustness of the corotational formulation, extensions have been proposed that recover inverted elements [ITF04, ST08].

Another concern arises from the integration of the corotational formulation in a FE solver. To account for the element rotations, in the corotational formulation the system matrix has to be re-built in every simulation step. Even though this does not pose a problem from a conceptual point of view, it considerably slows down the performance of the simulation process. This problem is amplified if multigrid methods are used to speed up the solution process as proposed in [WT04, GW05], meaning that quantities have to be computed on a hierarchy of different resolution grids. In particular, to ensure a consistent calculation of quantities on different resolution levels, the coarse grid matrices are derived by computing sparse products of the form RKR^{T}. Here, R and R^{T} are respectively the restriction and prolongation operator used to transfer quantities between different resolution levels, and K is the fine grid matrix. It is said that coarse grid operators constructed in this way satisfy the Galerkin condition, and it has been shown in previous work that such a construction leads to an optimal convergence of the multigrid method [Wes82, BHM00].

In case of unstructured mesh hierarchies, the corresponding matrix representations of the involved operators are sparse and non-zero entries are randomly scattered. As we will show in this paper, the update of the matrix hierarchy dominates the overall performance of dynamic multigrid methods, taking more than an order of magnitude longer than the system solver. The reason therefore is that sparse matrix products operate significantly below their theoretical peak performance due to the bottleneck of data transfer in the CPU memory hierarchy. A direct implication thereof is that standard algorithms for computing sparse matrix products can hardly achieve real-time performance in dynamic simulations for reasonably sized grid hierarchies.

1.1. Contribution

To overcome the aforementioned limitations, and thus to make FE based simulation of deformable 3D bodies stable and fast, we propose two novel approaches. Firstly, we introduce a method based on energy minimization to extract per-element rigid motions from their current state. The objective function we aim to minimize measures the similarity of the corotated configuration and the reference configuration. We show that by using this approach in FE elasticity simulations, large, yet stable deformations can be achieved, making this approach also attractive for deformation modeling in computer animation.

Secondly, we focus on improving the memory behavior of sparse matrix operations. We present a linear layout of computational cores for sparse-sparse matrix multiplication, which can effectively reduce the average memory access time. By reformulating the problem into the simultaneous processing of a sequential data and control stream, the locality of memory access operations can be improved, resulting in considerably less cache miss penalties. We include the proposed extensions into a multigrid approach for deformable bodies simulation. By means of our approach, we show an acceleration of the multigrid matrix hierarchy update by a factor of 10, yielding an overall speed up of the FE multigrid solver by a factor of up to 4.

Our paper is organized as follows: First, we shortly review the theory of the corotational FE formulation. In Section 3, we introduce the novel method to extract element rigid motions from their element displacements based on energy minimization. In the following section, we present the applied multigrid solver, and we describe in detail the developed data structure that accelerates the multigrid matrix hierarchy update. In Section 5, we present a detailed analysis of our approach, and finally we conclude with some aspects on future developments.

2. Corotated Finite Elements

Underlying our simulation is a linear elasticity model. In this model, we describe deformations as a mapping from the object's reference configuration Ω to its deformed configuration $\{x+u(x) \mid x \in \Omega\}$ using a displacement function $u : \mathbb{R}^3 \rightarrow \mathbb{R}^3$. The dynamic behavior of an object with linear elastic response is governed by the Lagrangian equation of motion

$$M\ddot{u} + C\dot{u} + Ku = f \tag{1}$$

where M, C, and K denote the mass, damping and stiffness matrix, respectively. u is a vector built from the displacement vectors of all vertices and f is analogously built from the per-vertex force vectors. The stiffness matrix K is constructed by assembling the element stiffness matrices K_e. Specifically, we are using a tetrahedral mesh, and every tetrahedral element has an associated element stiffness matrix with a form that is given by the strain tensor and the material law. To keep the matrices K_e linear with respect to u, the Cauchy strain tensor in combination with the generalized Hooke's law is used. For a more thorough derivation of the governing equations of linear elasticity let us refer to [BNC96, Bat02].

Since the Cauchy tensor is not invariant under rotations, especially if large deformations are applied, it tends to produce artificial forces yielding unrealistic results. To cure this problem, finite elements are first rotated into a configuration that matches best the reference configuration, such that the rigid body motions of the elements are eliminated before the strain is computed. This approach is known as the corotational FE formulation.

To consider element rotations in the finite element model, some modification have to be introduced. For clarity, we first focus on the element equations $K_e \in \mathbb{R}^{3m} \times \mathbb{R}^{3m}$ of one single finite element with m supporting vertices, i.e.,

$$K_e u_e = f_e.$$

To apply an element rotation to the deformed element configuration, we introduce element-dependent matrices Q_e, which are built from the respective orthogonal 3×3 element rotation matrices $\hat{Q}_e$ as follows:

$$Q_e = \begin{pmatrix} \hat{Q}_e & & \\ & \ddots & \\ & & \hat{Q}_e \end{pmatrix}.$$

The displacement from which the strain in the corotated configuration can be determined is $\tilde{u}_e = Q_e^{\mathrm{T}}(x_e + u_e) - x_e$. The obtained internal forces $K_e\tilde{u}_e$ then have to be rotated back into the deformed configuration, yielding

$$Q_e K_e Q_e^{\mathrm{T}}(x_e + u_e) = f_e + Q_e K_e x_e.$$

Thus, to handle finite elements in the corotational formulation the following steps have to be performed: a) The element matrices K_e have to be updated to $\tilde{K}_e = Q_e K_e Q_e^{\mathrm{T}}$ to account for the element rotations. b) A force correction vector $f_e^0 = Q_e(K_e x_e)$ has to be determined to account for the element rotations. These element-specific values have to be assembled into the global system matrix $\tilde{K}$ and the global force correction vector f^0, respectively.

Since only the calculation of the element strain is affected by the corotational formulation, the dynamic behavior of a deformable object is governed by the Lagrangian equation of motion with the modified matrix $\tilde{K}$ and the force correction vector f^0:

$$M\ddot{u} + C\dot{u} + \tilde{K}u = f + f^0. \tag{2}$$

By using an implicit time integration scheme like Euler or Newmark, a linear system of algebraic equations is obtained.

3. Extracting Element Rotations

To compute the element rotations in an accurate and stable way, we now introduce a novel approach based on energy minimization. Therefore, we define an energy E_e for every finite element Ω_e. E_e is used to measure the similarity between the corotated element configuration—which is determined by rotating the deformed configuration—and the reference configuration. The energy considers the pairwise distances between element points in the corotated configuration and corresponding points in the reference configuration, similar to the approach proposed in [MHTG05]. For every element its center of mass in the deformed configuration, c, is used as center of rotation to obtain the orientation in the corotated configuration. To match the reference configuration, we first translate the element such that the centers of mass coincide (c_0 denotes the center of mass in the reference configuration). Then, we compute the distance d between a point in the deformed element and its corresponding position in the reference configuration:

$$d = d(x) = Q_e^T(x+u-c) - (x-c_0).$$

The energy E_e is then derived by integrating the squared distances over the element domain:

$$E_e = \int_{\Omega_e} d^T d\, dx. \tag{3}$$

By minimizing this energy for every element we can determine the element rotations that minimize the average distances between positions in the reference and corotated configuration.

To solve this minimization problem, we first reformulate it by using quaternions [Sho85,LEF95]. In this formulation a vector v is rotated by applying quaternion multiplication $qv\overline{q}$. Here, q is the 4 component unit quaternion describing the rotation Q_e^{T}, and $\overline{q}$ is its conjugate. Therefore, in quaternion space the distance is determined as

$$d = d(x) = q\,(x+u-c)\,\overline{q} - (x-c_0).$$

We now search for a solution where the first derivative of the energy vanishes, i.e., $\frac{\partial E_e}{\partial q} = 0$. Since the minimum has to be found under the constraint that q is a unit quaternion, a Lagrange multiplier λ is added to the energy functional:

$$E_e = \int_{\Omega_e} d^{\mathrm{T}} d\, dx + \lambda(q^{\mathrm{T}}q - 1). \tag{4}$$

Then, the first derivatives become

$$\frac{\partial E_e}{\partial q_i} = 2\int_{\Omega_e} \frac{\partial d^T}{\partial q_i} d\, dx + 2\lambda q_i = 0, \tag{5}$$

$$\frac{\partial E_e}{\partial \lambda} = q^{\mathrm{T}}q - 1. \tag{6}$$

This system of non-linear equations is solved using a Newton solver, which iteratively determines the zero crossings of the set of equations (5) and (6). For this solver to work, the components of the Hessian matrix of E_e are computed as

$$\frac{\partial^2 E_e}{\partial q_j \partial q_i} = 2\int_{\Omega_e} \frac{\partial d^T}{\partial q_i}\frac{\partial d}{\partial q_j} + \frac{\partial^2 d^T}{\partial q_j \partial q_i} d\, dx + 2\lambda s(i,j), \tag{7}$$

$$\frac{\partial^2 E_e}{\partial \lambda \partial q_i} = 2q_i. \tag{8}$$

Here, $s(i,j)$ is a function returning 1 if i equals j and 0 otherwise. The unit quaternion describing the rotation that minimizes E_e is computed by starting with an initial state vector $\tilde{q}^k = (q^k, \lambda^k)$, and by iteratively updating this vector with respect to the following scheme:

$$\tilde{q}^{k+1} = \tilde{q}^k - \left(\frac{\partial^2 E_e}{(\partial \tilde{q})^2}\right)^{-1} \nabla E_e.$$

Both the gradient of E_e and its Hessian have to be computed in the current deformed configuration. Therefore, we first compute the center of mass c (c_0 is constant for each element), from which we derive the partial derivatives in equations (5) to (8) and finally the gradient and the Hessian via numerical integration.

4. Multigrid Solver

In this section, we describe an optimization technique to efficiently integrate finite elements using the corotational formulation into a multigrid FE solver. The solver we use to solve the linear system of equations $Ku = f$ employs geometry-specific operators to transfer quantities between different levels in a tetrahedral grid hierarchy [GW05]. Specifically, R_h and R_h^{T} are the restriction and the prolongation operators, respectively, which are used to transfer quantities to and from a particular level h.

In case of an unstructured mesh hierarchy, the corresponding matrix representations of these operators are sparse and non-zero entries are randomly scattered. The operators are used to construct the coarse grid system matrices K_{h+1} from a given fine grid system matrix K_h, where $K_0 = K$. To ensure a consistent calculation of quantities on different resolution levels, for all but the finest hierarchy level the coarse grid operators are obtained by solving products of the form $R_h K_h R_h^{\mathrm{T}}$. It is said that coarse grid operators constructed in this way satisfy the Galerkin condition, and it has been shown in previous work that such a construction is the natural choice for defining the coarse grid operators [Wes82].

Since in our application the matrix K_0 has to be re-built in every simulation step to account for the corotational formulation, the entire matrix hierarchy has to be adapted accordingly. As we will show in Section 5, this update operation dominates the overall performance of multigrid methods, taking about a factor of 10 longer than the system solver. The reason therefore is that sparse matrix products operate significantly below their theoretical peak performance due to the bottleneck of data transfer in the CPU memory hierarchy. A direct implication thereof is that standard algorithms for computing sparse matrix products can hardly

achieve real-time performance in dynamic simulations for reasonably sized grid hierarchies. To overcome this limitation, we focus on improving the memory behavior of sparse matrix operations in this paper. We present a linear layout of computational cores for sparse matrix multiplication, which can effectively reduce the average memory access time. By reformulating the problem into the simultaneous processing of sequential streams, the locality of memory access operations can be improved, yielding considerably less cache miss penalties.

4.1. Matrix Data Structure

To store the matrices involved in the simulation, we apply a *row-compressed* format (Yale format) [EGSS82]. To store a sparse matrix K, we encode the non-zero entries and their respective column indices in two separate arrays, row after row. Additionally, an index is stored for every row, which references the first non-zero entry in that row (see Figure 2). The set of column indices stored for each row i is denoted by S_i^K in the following.

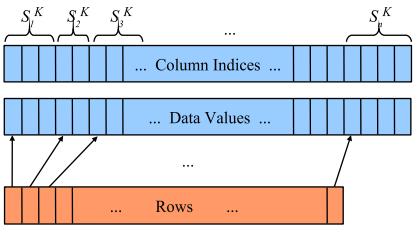

Figure 2: *Row-compressed matrix format.*

In the underlying implementation, we extend the row-compressed (RC) matrix format. Instead of just one single value, a non-empty block of entries is associated with each row and column index. This format is beneficial, because the stiffness matrix, in general, consists of 3×3 blocks of non-zero entries associated with each 3D vertex of the simulation mesh. Therefore, the memory overhead to store column indices can be reduced significantly, and the matrix entries can be updated more efficiently. In the following, we will refer to this format as *block-row-compressed* (BRC) matrix format.

4.2. Matrix Assembly

In the corotational formulation, in every simulation time-step the element stiffness matrices have to be updated before they get assembled into the global stiffness matrix. By storing the element matrices K_e of every finite element, matrices are updated as $\tilde{K}_e = Q_e K_e Q_e^{\mathrm{T}}$ once the element rotations are available. Since $\tilde{K}_e$ consists of 3×3 blocks of non-zero values for each pair of element vertices, these blocks can directly be written into the block-row-compressed global stiffness matrix K. To determine the respective destination block in K from the global vertex indices, one has to find the column index in the respective row. Although this can be done by a binary search if the column indices are sorted, it requires the entire set of column indices to be available in the CPU caches. To reduce this memory transfer in the assembly process, for each pair of vertices of an element we directly store a reference to the respective block in the global matrix. The memory overhead introduced by this approach is not significant, since it adds only 4 bytes to a dense 3×3 block of double precision floating point numbers in the block-row-compressed matrix format. Overall, this results in a memory increase by a factor of 1.05.

4.3. Sparse Matrix Products

Before we are going to present a specialized algorithm to efficiently calculate sparse-sparse matrix products of the form $E = RKR^{\mathrm{T}}$, let us first outline the naïve approach for doing so. To simplify things, we assume in this discussion, as well as in all upcoming discussions, a row-compressed matrix format. It should become clear from the text, however, that the extension to the block-row-compressed matrix format is straight-forward.

A particular entry in the matrix E is computed as

$$E_{ij} = \sum_{l \in S_i^R} R_{il} \left(\sum_{k \in S_l^K \cap S_j^R} K_{lk} R_{jk} \right). \tag{9}$$

The outer sum only considers the non-zero entries in the i-th row of R, denoted by the index set S_i^R. The inner sum can be optimized in such a way that it only accounts for indices in the intersection of the index sets S_l^K and S_j^R, since both rows of K and R are sparse. If the intersection is empty, the resulting term is zero. The expansion of the product has to be carefully designed such that all matrix data structures are accessed in a row-wise order.

The performance of the naïve approach is mainly limited due to the following properties: Firstly, to determine the intersections $S_l^K \cap S_j^R$ the entire (ordered) sets S_l^K and S_j^R have to be processed even though their intersection is typically very small or even empty. Secondly, the indices l and j themselves are determined by processing sparse index sets. Note, that the entries E_{ij} are processed as they are stored in the sparse matrix format, and therefore—once the structure of E has been determined—j is obtained from a sparse index set, too. Accessing these sets, namely S_l^K and S_j^R, produces scattered read operations that can most likely not be served from cache.

4.3.1. Stream Acceleration

To overcome the aforementioned limitations of the naïve approach for computing matrix products of the form RKR^{T}, we now introduce a cache-efficient algorithm to perform such operations. Specifically, we construct a *data* and *control* stream that is aligned with the data structure of the matrix K.

The data stream contains values of R and respective indices into E. The indices are used to scatter the multiplied entries from K and R into the destination matrix E. Note that because R and R^T do not change over time, their contributions to the product can be encoded into the stream. The control stream is used to encode how many pairs of data values and indices have to be processed for each non-zero entry of the matrix K. A single byte of the control stream is interpreted as follows: The first bit indicates whether the next non-zero entry of the matrix K should be fetched or the previous entry of K is used in the current calculations. The remaining seven bits indicate the number of data value/index pairs from the data stream that have to be processed. Note that at most 127 pairs can be encoded in one single byte. If an entry of K is scattered into the result matrix more than 127 times, an additional control byte has to be used with the first bit set to 0.

Due to this particular layout, scattered memory read operations to access pre-computed intersections $S_l^K \cap S_j^R$ can be avoided and only the final write operation accesses the memory randomly. An overview of the streaming approach is given in Figure 3.

The respective control and data streams are constructed as listed in Algorithm 1. The algorithm essentially performs the operations given in Equation 9. Since the streams are aligned to the matrix data structure of K rather than to that of E, the ordering of the sums in Equation 9 has to be changed, such that we immediately obtain all terms for each entry of K. This entries can then directly be encoded into the data stream.

In the pseudo code, the method E.getIndex(i, j) determines the index of an element (i, j) in the linearized data array of E. This index allows to quickly access the respective element of E in the stream processing stage. The stream's push() operation is used to encode the required information into the control and data stream. Besides storing the given value/index pair in the data stream, it also increments the number stored in the last control byte. If the maximum value of 127 is exceeded, a new control byte with the first bit set to 0 is appended to the stream. The stream's setNext() operation appends a new control byte to the stream. Since the first bit is set to 1, it refers to the next non-zero element of K.

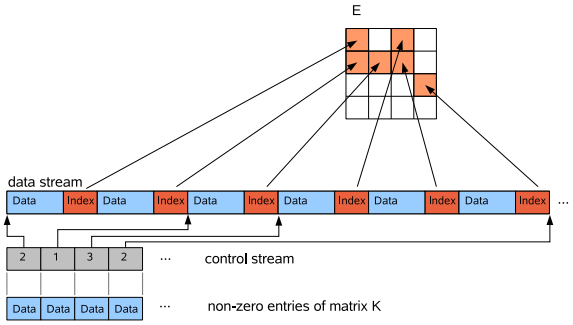

Figure 3: *Overview of the stream acceleration data structure.*

Algorithm 1 Stream Construction (In-Place)

Require: Matrices K, R^{T}, structure of matrix E
Ensure: $E = RKR^{\mathrm{T}}$

```
for i = 1 to E.numRows do
  for j ∈ S_i^E do
    E_ij = 0;
  end for
end for
for l = 0 to K.numRows do
  for k ∈ S_l^K do
    for i ∈ S_l^{R^T} do
      for j ∈ S_i^E ∩ S_k^{R^T} do
        E_ij = E_ij + K_lk · R_li^T · R_kj^T;
        stream.push(R_li^T · R_kj^T, E.getIndex(i, j));
      end for
    end for
    stream.setNext();
  end for
end for
```

Computing the destination matrix E is performed by sequentially traversing the entire stream and repeating the following two steps for each control byte (l and k denote the row/column index of the current non-zero entry of K, and they are initialized to index the first non-zero entry of K):

Step 1: If the first bit of the current control byte equals 1, the index k is advanced to the next non-zero entry of K in the row l. If such an entry is not available, the row index l is incremented to the next non-empty row, and k is set to the first non-zero column index of that row. Then, the value K_{lk} is stored in a temporary register t. The last 7 bits of the control byte encode the number p of weight/index pairs that have to be processed.

Step 2: Do p times:
Read a data value w and an index value i from the data stream, determine the product $w \cdot t$ and add it to the entry $E(i)$ of the destination matrix ($E(i)$ addresses the i-th position in the linearized representation of E).

5. Results

In this section, we analyze the performance of the Multigrid finite element solver using the proposed corotational formulation of finite elements in combination with the cache-efficient sparse matrix representations. All benchmarks were run on a standard desktop PC with an Intel Core$^{\mathrm{TM}}$ 2 Duo 6600 2.4 GHz processor equipped with 2 GB RAM.

Table 1 shows the performance of the proposed multigrid solver at different FE model resolutions. We use one Gauss-Seidel step both for pre- and post-smoothing in the multigrid

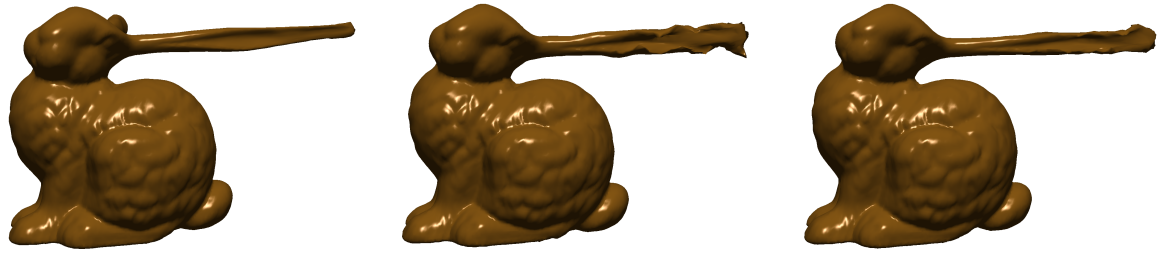

Figure 4: *Comparison of different approaches to estimate per-element rotations. Left: novel approach based on energy minimization. Middle: polar decomposition of the deformation gradient. Elements start fluttering due to large stretching. Right: spatial smoothing of element rotations. Fluttering is reduced but still noticeable.*

V-cycle. In the first column we list the used model (see Table 2 for a more detailed model analysis). Next, we give the times required to compute per-element rotations based on energy minimization. The third column presents the computation times for re-assembling the system matrix (including the rotation of the element stiffness matrices). Then, we analyze the performance of the procedure used to update the multigrid matrix hierarchy, which already exploits the novel stream acceleration data structure. In the next column, the time that is required for solving the system $Ku = f$ is given. Finally, we show the performance of the multigrid solver in time steps per second (tps).

The most striking performance gain results from using the stream acceleration structure to update the multigrid hierarchy. Typically, the Galerkin update of the coarse level matrices is performed by successively performing the naïve approach (see Equation 9). The time required for this naïve approach is given in Table 1 in brackets for our models. Due to the proposed acceleration structure, the same operation is now about 10 times faster. It is especially interesting to note that this performance gain is achieved despite the additional memory that is required to store the data and control streams (between 15 MB and 132 MB in the current examples). This result clearly indicates that the performance of sparse matrix operations is vastly dominated by cache-miss penalties, which can be reduced significantly by the streaming approach presented in this paper.

Summarizing, we achieve a performance of the advanced finite element multigrid solver that is about some orders of magnitudes faster than current FE approaches. The approach of Wicke et al. [WBG07] is roughly a factor of 20 slower for an equal number of degrees of freedom to be solved. Even for the most recent FE approach by Kaufmann et al. [KMBG08]—running on the same hardware as our implementation—, we observe a speedup of a factor of more than 10.

To further validate our method against previous approaches, we have also integrated the polar decomposition of the deformation gradient [HS04] into the multigrid solver. Due to the aforementioned limitations of this approach regarding stability, we apply a spatial smoothing of the computed rotation field on the tetrahedral grid. The smoothing is performed in quaternion space, meaning that we compute quaternion averages in $\mathbb{R}^4$ and re-normalize the weighted quaternion to place it on the unit sphere S^3. The centroid is computed as $\sum_i w_i q_i / \| \sum_i w_i q_i \|$, where q_i are unit quaternions on S^3 and w_i are the respective weights. Even though this kind of re-normalization can cause undesirable effects, in the current examples it could significantly improve the stability of the approach.

Per-element rotations are first transformed into corresponding quaternions. Spatial smoothing is then performed by gathering quaternions from the 1-ring neighborhood of each element, and by computing quaternion averages as described. Weights are chosen according to the volume of averaged elements. As can be seen from Figure 4, which provides a visual comparison between the different methods used to extract per-element rotations, even though spatial smoothing

Model	Calculation times [ms] Rotate	Assemble	Update	Solve	Total [tps]
Bunny	21	18	11 (108)	10	17
Horse	67	58	39 (354)	26	5.2
Dragon	112	103	94 (1183)	51	2.8

Table 1: *Timing statistics for the multigrid FE solver using the corotational finite element formulation. Per-element rotations are computed by energy minimization as introduced in Section 3. In the Update column, times in brackets denote the multigrid update operation using standard operations instead of the novel stream acceleration approach.*

Model	# Elements	# Vertices	# Levels	Total [tps]
Bunny	11241	2918	3	31
Horse	33689	8632	4	10.1
Dragon	67309	16943	4	5.0

Table 2: *Model and timing statistics for the multigrid FE solver using the corotational finite element formulation. Per-element rotations are computed by a polar decomposition of the deformation gradient and a spatial smoothing of the rotation field.*

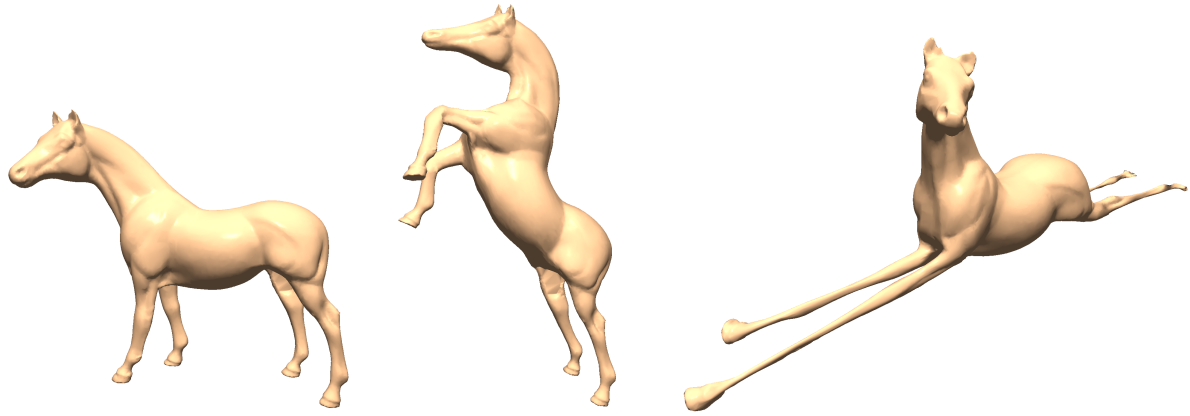

Figure 5: *Large-scale deformations using the FE model based on the corotational formulation. Energy minimization is used to extract per-element rigid motions.*

weakens the limitations of the polar decomposition it cannot achieve the same stability as the energy minimization. Furthermore, since energy minimization is only about a factor of two slower than spatial smoothing (see Table 2), it positions itself as an effective and efficient means for FE simulations using the corotational formulation of finite elements.

Figure 5 demonstrates the stability of the proposed FE model using the corotational formulation. For the horse model, we show two deformations. In the middle image, the horse is modeled with heterogeneous material parameters (stiff legs and soft knees), and it is fixed on the hooves of the hind legs. This allows bringing the model into the presented pose by pulling the shoulders to the top and folding the fore legs. It is worth noting that all applied forces are accumulated to keep the deformed body in its pose. In the right image, the horse is modeled with homogeneous material parameters, while the fixation is unchanged. The horse is then largely stretched to demonstrate the stability of energy minimization for the determination of per-element rotations. When using geometric methods or polar decomposition of the deformation gradient to estimate these rotations, the model starts fluttering and becomes increasingly unstable. However, since our energy minimization extracts rotations only locally, spatial consistency cannot always be ensured.

6. Conclusion

In this paper, we have presented novel algorithms to improve the simulation of deformable bodies based on the corotational formulation of finite elements. Firstly, we have introduced a new method to extract the rigid body motion from total finite element displacements based on energy minimization. This formulation greatly improves the stability of corotated finite elements, and it only leads to a slight increase of the computational load compared to previous approaches. Secondly, we have presented a cache-efficient data structure to significantly accelerate the matrix update procedure in Galerkin multigrid approaches. By reformulating the sparse matrix product into the simultaneous processing of a data and a control stream, cache-miss penalties can be effectively reduced. The new approach accelerates the update of the multigrid matrix hierarchy by a factor of 10 compared to previous multigrid approaches, which yields an overall acceleration of the simulation by a factor of up to 4.

In the future we will go into three different research directions: a) we will analyze whether the energy minimization approach can be efficiently incorporated into a global minimization problem thereby accounting for further improved spatial consistency of the rotations. b) we will investigate the parallelization of the multigrid solver on multi-core systems like the Intel Larrabee visual computing architecture. This will allow us to considerably speed up physics-based FE simulations, at the same time enabling the handling of very large data sets. c) we will try to integrate cutting mechanisms into the multigrid FE simulation. In particular we will explore the integration of finite polyhedral cells [WBG07] into the multigrid setting, and we will investigate techniques to efficiently handle local topology changes in this setting.

References

[Bat02] BATHE K.-J.: *Finite Element Procedures*. Prentice Hall, 2002.

[BH79] BELYTSCHKO T., HSIEH B. J.: Application of higher order corotational stretch theories to nonlinear finite element analysis. *Computers & Structures 10* (1979), 175–182.

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A Multigrid Tutorial, Second Edition*. SIAM, 2000.

[BNC96] BRO-NIELSEN M., COTIN S.: Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Proceedings of Eurographics* (1996), pp. 57–66.

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proceedings of SIGGRAPH* (1998), pp. 43–54.

[CDA99] COTIN S., DELINGETTE H., AYACHE N.: Real-time elastic deformations of soft tissues for surgery simulation. In *IEEE Transactions on Visualization and Computer Graphics* (1999), pp. 62–73.

[CGC*02] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIĆ Z.: A multiresolution framework for dynamic deformations. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), pp. 41–47.

[DDCB01] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A. H.: Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of SIGGRAPH* (2001), pp. 31–36.

[DSB99] DESBRUN M., SCHRÖDER P., BARR A.: Interactive animation of structured deformable objects. In *Proceedings of Graphics Interface* (1999), pp. 1–8.

[EGSS82] EISENSTAT S. C., GURSKY M. C., SCHULTZ M. H., SHERMAN A. H.: Yale sparse matrix package. *International Journal of Numerical Methods for Engineering* (1982), 1145–1151.

[EKS03] ETZMUSS O., KECKEISEN M., STRASSER W.: A fast finite element solution for cloth modelling. In *Proceedings of Pacific Conference on Computer Graphics and Applications* (2003), p. 244.

[FH05] FELIPPA C., HAUGEN B.: A unified formulation of small-strain corotational finite elements: I. theory. *Computer Methods in Applied Mechanics and Engineering 194* (2005), 2285–2335.

[GKS02] GRINSPUN E., KRYSL P., SCHRÖDER P.: CHARMS: a simple framework for adaptive simulation. In *Proceeding of SIGGRAPH* (2002), pp. 281–290.

[GW05] GEORGII J., WESTERMANN R.: A Multigrid Framework for Real-Time Simulation of Deformable Volumes. In *Proceedings of the 2nd Workshop On Virtual Reality Interaction and Physical Simulation* (2005), pp. 50–57.

[HS04] HAUTH M., STRASSER W.: Corotational simulation of deformable solids. In *Proceedings of WSCG* (2004), pp. 137–145.

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics symposium on Computer Animation* (2004), pp. 131–140.

[KMBG08] KAUFMANN P., MARTIN S., BOTSCH M., GROSS M.: Flexible simulation of deformable models using discontinuous galerkin fem. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), pp. 105–115.

[LEF95] LORUSSO A., EGGERT D. W., FISHER R. B.: A comparison of four algorithms for estimating 3-d rigid transformations. In *BMVC '95: Proceedings of the 1995 British conference on Machine vision (Vol. 1)* (Surrey, UK, UK, 1995), BMVA Press, pp. 237–246.

[LTW95] LEE Y., TERZOPOULOS D., WALTERS K.: Realistic modeling for facial animation. In *Proceedings of SIGGRAPH* (1995), pp. 55–62.

[MDM*02] MÜLLER M., DORSEY J., MCMILLAN L., JAGNOW R., CUTLER B.: Stable real-time deformations. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), pp. 49–54.

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proceedings of Graphics Interface* (2004), pp. 239–246.

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (2005), 471–478.

[ML03] MENDOZA C., LAUGIER C.: Simulating soft tissue cutting using finite element models. In *Proceedings of IEEE International Conference on Robotics and Automation* (2003), pp. 1109–1114.

[NMK*05] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically based deformable models in computer graphics. In *Proceedings of Eurographics* (2005), pp. 71–94.

[NPF05] NESME M., PAYAN Y., FAURE F.: Efficient, physically plausible finite elements. In *Eurographics (short papers)* (august 2005), Dingliana J., Ganovelli F., (Eds.).

[PDA01] PICINBONO G., DELINGETTE H., AYACHE N.: Nonlinear and anisotropic elastic soft tissue models for medical simulation. In *Proceedings of IEEE International Conference on Robotics and Automation* (2001), pp. 1370–1375.

[RB86] RANKIN C., BROGAN F.: An element-independent corotational procedure for the treatment of large rotations. *ASME J. Pressure Vessel Tchn. 108* (1986), 165–174.

[Sho85] SHOEMAKE K.: Animating rotation with quaternion curves. *SIGGRAPH Comput. Graph. 19*, 3 (1985), 245–254.

[ST08] SCHMEDDING R., TESCHNER M.: Inversion handling for stable deformable modeling. *The Visual Computer 24* (2008), 625–633.

[TF88] TERZOPOULOS D., FLEISCHER K.: Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. In *Proceedings of SIGGRAPH* (1988), pp. 269–278.

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. In *Proceedings of SIGGRAPH* (1987), pp. 205–214.

[WBG07] WICKE M., BOTSCH M., GROSS M.: A finite element method on convex polyhedra. In *Proceedings of Eurographics* (2007).

[WDGT01] WU X., DOWNES M. S., GOKTEKIN T., TENDICK F.: Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes. In *Proceedings of Eurographics* (2001), pp. 349–358.

[Wem69] WEMPNER G. A.: Finite elements, finite rotations and small strains of flexible shells. *International Journal of Solids and Structures 5* (1969), 117–153.

[Wes82] WESSELING P.: A robust and efficient multigrid method. In *Multigrid Methods*, Hackbusch W., Trottenberg U., (Eds.), vol. 960 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1982, pp. 614–630.

[WT04] WU X., TENDICK F.: Multigrid integration for interactive deformable body simulation. In *Proceedings of International Symposium on Medical Simulation* (2004), pp. 92–104.

[ZC99] ZHUANG Y., CANNY J.: Real-time simulation of physically realistic global deformation. In *Proceedings of IEEE Visualization* (1999), pp. 270–273.