Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2008)
M. Gross and D. James (Editors)

Fast Adaptive Shape Matching Deformations

Denis Steinemann[1], Miguel A. Otaduy[2] and Markus Gross[1]

[1] Computer Graphics Laboratory, ETH Zürich, Switzerland
[2] Grupo de Modelado y Realidad Virtual, URJC Madrid, Spain

Abstract

We present a new shape-matching deformation model that allows for efficient handling of topological changes and dynamic adaptive selection of levels of detail. Similar to the recently presented Fast Lattice Shape Matching (FLSM), we compute the position of simulation nodes by convolution of rigid shape matching operators on many overlapping regions, but we rely instead on octree-based hierarchical sampling and an interval-based region definition. Our approach enjoys the efficiency and robustness of shape-matching deformation models, and the same algorithmic simplicity and linear cost as FLSM, but it eliminates its dense sampling requirements. Our method can handle adaptive spatial discretizations, allowing the simulation of more degrees of freedom in arbitrary regions of interest at little additional cost. The method is also versatile, as it can simulate elastic and plastic deformation, it can handle cuts interactively, and it reuses the underlying data structures for efficient handling of (self-)collisions. All this makes it especially useful for interactive applications such as videogames.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism: Animation.

1. Introduction

Simulation of deformable objects can greatly enhance the level of engagement of many computer graphics applications, and this importance has lead to the development of many and diverse deformation models over the last twenty years. In computer animation for feature films, physical realism is often the most important aspect. In interactive applications such as video games or surgery simulation, however, computational efficiency and robustness are the dominant aspects, trading physical accuracy for plausibility.

Computer graphics has recently sought the rise of shape-matching deformation models for robust and efficient computation of large deformations, with application to video-game-like settings. Müller et al. [MHTG05] presented a meshless simulation technique that pulls deformable points toward a globally consistent deformed shape, resulting in unconditionally stable and extremely fast deformation. Rivers et. al [RJ07] have extended this technique to simulate many more degrees of freedom (DOFs) using *Fast Lattice Shape Matching* (FSLM). They overlap many (rigid) clusters of points in a lattice, and exploit the regularity of the lattice for designing an efficient algorithm. The use of a regular lattice induces however unsolved limitations, such as lack of flexibility for distributing DOFs, poor scalability in terms of resolution, homogeneous mechanical stiffness, or large cost for applying topological changes.

Our Contribution

In this paper, we present a novel dynamic deformation technique based on shape matching that solves many of the limitations of previous methods and extends their applicability to scenes and models exhibiting interactive complex topological changes (see Figure 3), inhomogeneous mechanical behavior (Figure 4), independently deformed thin features (Figure 1), and adaptive and dynamic LOD selection (Figure 6). Our simulation algorithm enjoys the same algorithmic simplicity as FLSM, and its runtime cost is also linear in the number of deformation points. However, our adaptive sampling framework enables in practice the simulation of much thinner features than FLSM at a much lower cost, as shown in Figure 1.

The technical contributions of our work may be listed as:

- A hierarchical fast summation algorithm for shape-matching deformations with adaptive discretizations. It

is sustained on *octree-based sampling* and *interval-based definition* of shape-matching regions.
- An algorithm for dynamic resampling of the octree representation that allows interactive topological changes and LOD selection.
- A fast method for computing distances on the octree setting, which is used in the dynamic update of shape matching regions.

We continue with a discussion of related work, focusing on earlier shape matching algorithms in Section 3. In Section 4 we present our octree shape matching algorithm. In Section 5 we describe fast resampling under topological changes and dynamic LOD selection, while in Section 6 we introduce an efficient shortest path algorithm used for dynamic resampling. In Section 7 we present experiments and results, and conclude with a discussion of future work.

2. Related Work

Physically-based simulation of deformable materials was introduced to computer graphics more than twenty years ago [TPBF87]. Since then, many researchers have aimed at obtaining robust deformation models with low computational cost. Some of their approaches include fast solutions to implicit integration of FEM models [BNC96], corotational FEM for large deformations with implicit integration [MDM*02], boundary element methods focused on the surface [JP99], or mass-spring models with additional volume conservation constraints [THMG04]. Modal analysis methods also enable fast and robust computation of global deformations on geometrically complex objects by extracting the main deformation modes [PW89, JP02, BJ05]. Another way of achieving fast simulations on complex geometry is to compute the deformation on a lattice or low-resolution mesh [CGC*02a]. Meshless methods feature other interesting properties such as robust simulation of large deformations, state-transitions, or topological changes [BLG94, MKN*04, SOG06]. For a more extensive discussion of physically-based deformation models, please refer to surveys on the topic [GM97, NMK*05].

As an alternative to physically-based methods, shape-matching deformation models rest on purely geometric grounds [MHTG05, RJ07]. They move deformation points toward goal positions defined by the rest geometry, thus guaranteeing stability of the simulation. Effectively, they offer the robustness of implicit integration in physically-based methods, with a cost comparable to explicit integration, making them a great candidate for plausible simulation in interactive applications. Due to the strong connection between our work and that in [MHTG05, RJ07], we discuss these approaches in detail in the next section. Shape-matching deformation models have also seen application in geometric modeling. The prism-based deformation model of Botsch et. al [BPGK06] aims at finding per-prism rigid transformations that minimize a deformation energy. However, the method solves a global optimization problem, hence it is not quite suited for interactive simulations with many DOFs. This work was later extended to deal with adaptive sampling [BPWG07]. The position-based dynamics technique of Müller et al. [MHHR06] is somewhat kindred to shape-matching models, as it moves points in a deformable model toward goal positions defined by local constraints.

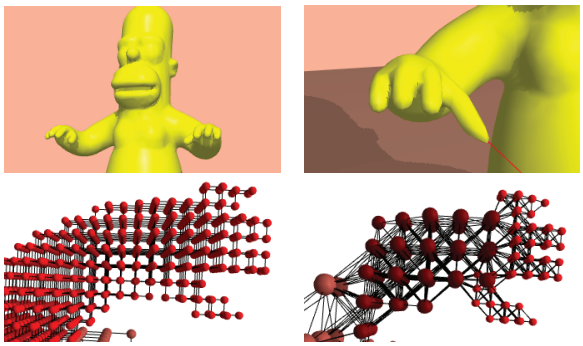

Figure 1: *An object is deformed with our octree shape matching approach. Notice the independent deformation of a finger in the top-right. FLSM requires* 35 000 *nodes to correctly represent thin fingers (bottom-left), while our adaptive approach requires only* 661 *nodes (bottom-right), providing* 88*-time speed-up in the simulation.*

Two of the main applications of our technique are efficient handling of adaptive simulation and topological changes. There has been extensive work on adaptive simulation in computer graphics [DDCB01, GKS02, CGC*02b, OGRG06], although orthogonal to ours. Regarding topological changes, our work shares some of the problems of dynamic resampling in meshless deformations [PKA*05, SOG06].

3. Deformation through Shape Matching

In this section, we review the previous approaches by [MHTG05] and [RJ07] and discuss limitations.

3.1. Meshless Shape Matching

Given a set R_r of simulation points, with $\mathbf{x}_i^0$ and $\mathbf{x}_i$ their initial and deformed positions, the technique by Müller et. al [MHTG05] computes a rigid rotation $\mathbf{R}_r$ and a translation vector that transform the $\mathbf{x}_i^0$ such that the distance between initial and deformed configurations is minimal in the least-squares sense. A linear transformation

$$\mathbf{A} = \Big(\sum_{i \in R_r} m_i \mathbf{p}_i \mathbf{q}_i^T\Big)\mathbf{A}_{qq} = \mathbf{A}_r \mathbf{A}_{qq} \qquad (1)$$

is computed first, with $\mathbf{p}_i = \mathbf{x}_i - \mathbf{c}_r$ and $\mathbf{q}_i = \mathbf{x}_i^0 - \mathbf{c}_r^0$. $\mathbf{c}_r$ and $\mathbf{c}_r^0$ are the centers of mass in the deformed and the initial setting, and $\mathbf{A}_{qq}$ is a symmetric matrix containing only scaling. $\mathbf{R}_r$ is extracted from $\mathbf{A}_r$ using polar decomposition such that $\mathbf{A}_r = \mathbf{R}_r\mathbf{S}$. Next, the goal position of each node is computed as $\mathbf{g}_i = \mathbf{R}_r(\mathbf{x}_i^0 - \mathbf{c}_r^0) + \mathbf{c}_r = \mathbf{T}_r\mathbf{x}_i^0$, $\mathbf{T}_r = [\mathbf{R}_r\ \mathbf{t}_r] \in \mathbb{R}^{3x4}$. Goal

positions are then used in an unconditionally stable numerical integration scheme:

$$\mathbf{v}_i(t+h) = \mathbf{v}_i(t) + \frac{\mathbf{g}_i(t) - \mathbf{x}_i(t)}{h} + h\frac{\mathbf{f}_{ext}(t)}{m_i}, \quad (2)$$

$$\mathbf{x}_i(t+h) = \mathbf{x}_i(t) + h\mathbf{v}_i(t+h). \quad (3)$$

Müller et. al extended the basic definition to linear and quadratic deformation modes. They also increased the number of DOFs in the deformation by clustering points into several regions, but deformation artifacts may appear due to region discontinuities, and clustering itself is problematic.

3.2. Fast Lattice Shape Matching

Rivers et. al [RJ07] applied the technique of [MHTG05] on cubic lattices and overlapped many clusters through a region-based convolution of rigid shape-matching transformations, resulting in smooth deformation. The object's surface is embedded in the lattice and deformed using trilinear interpolation of lattice vertices. In the FLSM of Rivers et al., each lattice point represents a simulation node i, and is associated to a shape matching region R_i composed of i and all other nodes closer than a distance w (according to the max-norm metric). FLSM computes per-region transformations $\mathbf{T}_r$ as described above, and per-node goal positions are obtained by averaging the transformations of all influencing regions. With a symmetric definition of regions, goal positions are defined as $\mathbf{g}_i = <\mathbf{T}_r\mathbf{x}_i^0>_{r\in R_i} = \mathbf{T}_i\mathbf{x}_i^0$.

Naively multiplying and summing up vectors for each region as in (1) would yield a cost $O(w^3n)$, where n is the number of nodes. Instead, FLSM exploits summation redundancy in the lattice setting and yields a total cost linear in the number of nodes. We refer to [RJ07] for the exact definition of their fast-summation operator

$$\boldsymbol{F}_{i\in R_r}\{v_i\} \equiv \sum_{i\in R_r} v_i, \quad (4)$$

which indicates that a quantity v_i is summed over a region R_r. In essence, it requires three recursive passes along all simulation nodes, and it is used for computing region transformations and node goal positions. As we will show in section 4.3, our algorithm follows the same steps as FLSM, but replaces $\boldsymbol{F}$ with a new hierarchical fast-summation operator that works on adaptive discretizations.

3.3. Limitations of the Lattice Setting

FLSM allows for more DOFs and smoother deformation than the original method by [MHTG05], but the use of a regular lattice yields several important limitations:

- Small features yield an explosion of the runtime cost. A small surface feature may require fine sampling in order to be deformed independently from non-adjacent material, but this fine sampling must be applied to the whole object. Figure 1 shows how our adaptive sampling correctly resolves thin features efficiently, while in FLSM the simulation cost grows cubically with lattice resolution.
- Mechanical stiffness, related to region half-width w, is a global parameter. A varying width w would break the regularity required by FLSM. Figure 4 shows material inhomogeneity in our octree setting.
- Dynamic restructuring due to topological changes is computationally expensive. The fast-summation algorithm becomes particularly intricate in regions where the lattice is not regular, e.g., near boundaries, as several sums must be maintained per node. Although [RJ07] show the ability to perform fracture, the definition of these sums is typically handled as pre-processing, and is expensive at runtime.

4. Octree Shape Matching

In this section, we will introduce our new deformation method based on octree shape matching.

4.1. Adaptive Octree Sampling

As opposed to the uniform lattice sampling of [RJ07], we propose an octree-based sampling of the deformable objects. Octree-based sampling lays a framework for adaptive discretization of the shape-matching deformation model. Moreover, as it will become clear later, the octree representation will allow for a hierarchical definition of a fast summation operator, where a high node in the octree stores the sum of all its leaves. When appropriate, nodes will reuse high-level sums without visiting subtrees.

We begin by creating a very coarse cubic lattice, referred as *base lattice*, that embeds the object's surface. The choice of resolution for the base lattice is guided by the maximum desired stiffness, and has 28 nodes in the example in Figure 1. We then do an octree subdivision of the lattice, following some user-defined criterion. One possibility is to subdivide until all surface features of a certain size are resolved, as shown in Figure 1, but it is also possible to subdivide at runtime based on, e.g., user interaction as in Figure 6, view-dependent LOD selection, etc. We place a simulation node at the center of each leaf cell, and a *virtual node* at the center of each non-leaf cell. As mentioned earlier, virtual nodes will store sums of all their descendant simulation nodes. Masses of simulation nodes are set based on cell volume and density.

4.2. Interval-Based Shape Matching Regions

FLSM exploits lattice regularity for avoiding the region-size-dependency of brute-force shape matching. Instead, we propose an interval-based definition of regions that, together with the octree representation, ensures that each summation operation need only operate on $O(1)$ summands.

Given a simulation node n_i with tentative region width w_i, we define the region R_i of n_i in the following way: if a node n_j is closer than w_i from n_i, then it belongs to R_i; and if it is further than $(1+\varepsilon)w_i$, then it does not belong. As a result, nodes in the distance interval $[w_i, (1+\varepsilon)w_i]$ *may or may not* be included in R_i. This definition bears some resemblance with that of $(1+\varepsilon)$-spanners [GGN06]. Recall that

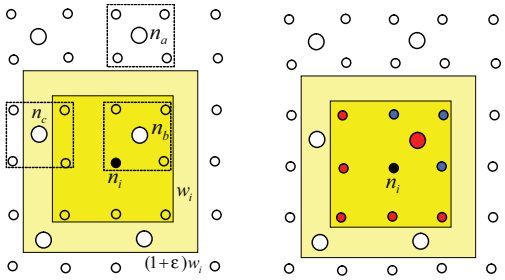

Figure 2: *Hierarchical sampling of an object. (a) Intervals of n_i and virtual nodes. The virtual node n_b stores a distance interval $[0,1]$ (in max-norm metric) and is added as summation node of n_i; n_a stores $[2,3]$ and is discarded; and n_c stores $[1,2]$ and is refined. (b) In red, the summation nodes of region R_i associated with n_i.*

region width is related to local mechanical stiffness, and our interval-based region definition effectively implies a small variance in the actual mechanical stiffness.

In order to construct shape-matching regions, we employ the interval-based definition above and we follow a hierarchical algorithm. For each simulation node n_i, we represent its shape-matching region R_i through a set of *summation nodes*, which may contain both simulation and virtual nodes. Note again that a virtual node will store summed values of all its descendant leaves, hence using a virtual node as summation node prevents us from visiting all its leaves. A shape-matching region R_i is constructed in the following way:

1. For every node n_j of the octree, compute an interval $[a_j, b_j]$ that captures the minimum and maximum distances from all descendant leaves of n_j to n_i.
2. Traverse the octree top-down, and for every node n_j:
   if $a_j > w_i$, discard n_j and its subtree.
   else if $b_j < (1+\varepsilon)w_i$, insert n_j in R_i.
   else process the children of n_j.
3. Enforce region symmetry, $n_i \in R_j \iff n_j \in R_i$.

Figure 2 shows an example situation with summation nodes at two different levels. There exists a minimum value ε such that the number of summation nodes in every region is bounded by a desired constant. In practice, we use $\varepsilon = 0.5$, which yields on average 6 summation nodes per region in the adaptively sampled model in Figure 1.

Note that our definition of shape-matching regions is independent of the distance metric and the algorithm for computing node distances. We describe the efficient computation of distances in our implementation in Section 6.

4.3. Hierarchical Fast Summation

Given $O(1)$ summation nodes per shape-matching region as defined above, we can now define our linear-cost *Hierarchical Fast-Summation* operator

$$\underset{i \in R_r}{\boldsymbol{HF}} \left\{ v_i \right\} \equiv \sum_{i \in R_r} v_i. \tag{5}$$

It consists of two steps:

1. *Depth summation*: For all octrees, compute sums of v_i in bottom-up fashion, such that virtual nodes contain the sum of their children's values.
2. *Breadth summation*: For each region, sum up the readily available values of all its summation nodes.

For an object with n simulation nodes, the depth-summation is $O(n)$ if we assume that the octrees are roughly balanced. The breadth summation is $O(n)$ as outlined above, hence the $\boldsymbol{HF}$ operator has linear cost.

Similarly to the $\boldsymbol{F}$ operator in FLSM, our novel hierarchical fast-summation operator $\boldsymbol{HF}$ is used for computing the per-region transformations $\mathbf{c}_r$ and $\mathbf{A}_r$, as well as per-node goal positions $\mathbf{g}_i$ (See Section 3.1 for their definition). The only difference w.r.t. FLSM is that we weight per-region transformations by the mass m_r of their associated node, and we then normalize the sum by the summed mass M_i, which can be precomputed.

Our octree shape matching algorithm is now summarized:

1. Compute per-region translations
$$\mathbf{c}_r = \frac{1}{M_r} \boldsymbol{HF}_{i \in R_r} \left\{ m_i \mathbf{x}_i \right\}. \tag{6}$$
2. Compute per-region linear transformations
$$\mathbf{A}_r = \boldsymbol{HF}_{i \in R_r} \left\{ m_i \mathbf{x}_i \mathbf{x}_i^{0^T} \right\} - M_r \mathbf{c}_r \mathbf{c}_r^{0^T}. \tag{7}$$
3. Extract rotations $\mathbf{R}_r$ using polar decomposition and compose rigid transformations $\mathbf{T}_r$.
4. Compute per-node goal positions
$$\mathbf{g}_i = \frac{1}{M_i} \boldsymbol{HF}_{r \in R_i} \left\{ m_r \mathbf{T}_r \right\} \mathbf{x}_i^0. \tag{8}$$
5. Apply the integration scheme from (2) and (3).

It now becomes apparent that our octree shape matching shares the same algorithmic structure and linear-cost as FLSM, with the notable difference that it supports adaptive sampling, and thereby the possibility to simulate much thinner features at a lower total cost. Damping, described in [RJ07], is also directly applicable with our $\boldsymbol{HF}$ operator.

5. Dynamic Resampling

When topological changes are applied or when new LODs are locally (de)activated, objects must be dynamically resampled, and summation nodes of affected shape matching regions must be recomputed. In this section, we describe a general, robust, and efficient algorithm for dynamic resampling under our octree-based setting. While describing the algorithm, we assume the existence of a method for computing distances between pairs of nodes. Our particular method, described in Section 6, uses a visibility graph, and here we will also refer to updates to the graph during resampling.

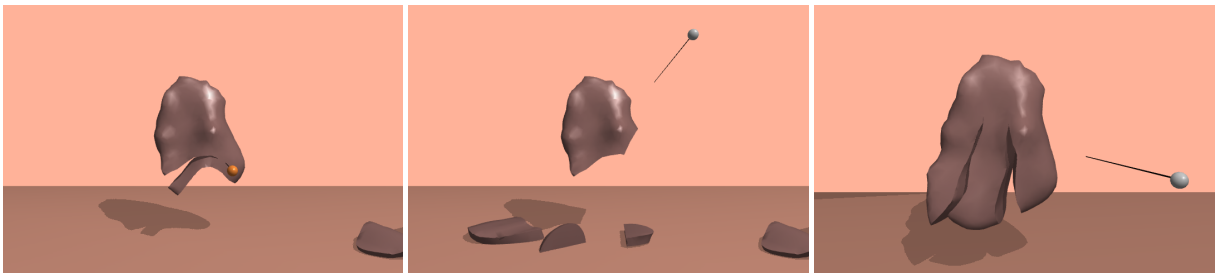

Figure 3: *A hanging liver model is interactively cut, while shape matching regions are efficiently recomputed, and self-collisions are also interactively handled. The model starts with* 500 *nodes and ends with* 1 550.

5.1. Topological Changes

Let us define as N_{update} the nodes for which shape matching regions must be updated or computed from scratch (i.e., summation nodes must be identified). A topological change is detected when an edge $e(n_i, n_j)$ of the visibility graph is cut. In this case, N_{update} must include the simulation nodes n_i and n_j, plus all other simulation nodes whose regions include either n_i or n_j.

5.2. Dynamic LOD Updates

To refine an existing simulation node n_i, its associated cell is subdivided according to user-defined criteria, and nodes are created for each new cell. All new leaf nodes become simulation nodes, while n_i becomes virtual and its associated region is removed. To coarsen (sibling) simulation nodes $\{n_i\}$, they are removed from the tree, while their parent is set as new simulation node. Both when refining or coarsening, N_{update} consists of the newly created simulation nodes and those nodes n_j whose region R_j includes a removed node. The visibility graph must be updated by deleting the edges incident on removed nodes, and setting visibility edges for the newly added nodes (See Section 6.1 for more details).

5.3. Updating Shape Matching Regions

Once N_{update} is determined, either after dynamic LOD updates or topological changes, we can recompute summation nodes. For all nodes $n_i \in N_{update}$ we do the following:

1. Recompute the distance to other simulation nodes. Note that distances do not need to be computed for nodes further than $(1+\varepsilon)w_i$. Distance recomputation amounts to more than 80% of the computation time when recomputing summation nodes.
2. For virtual nodes, compute distance intervals in a bottom-up manner.
3. Traverse the octrees in a top-down manner, determining summation nodes by checking the distance intervals. Under topological changes, distances in the undeformed configuration cannot grow, hence it is sufficient to start the top-down traversal at the old summation nodes of n_i.
4. Once summation nodes are determined, recompute the constant quantities $\mathbf{c}_r^0$ and M_r.

6. Efficient Distance Computation

In this section we describe our algorithm for efficiently computing distances between simulation nodes. We connect nodes using a visibility graph and define distances as shortest paths along the graph, similar to [SOG06]. We first describe the initialization of the graph, and then a novel bucket-based version of Moore's algorithm for shortest path computation [Moo59].

6.1. Graph Initialization

After sampling an object as described in Section 4.1, we create a visibility graph in a way similar to [SOG06]. Given a simulation node n_i at an octree level where the distance between nodes is d_i, we set edges to all other simulation nodes closer than or equal to d_i. Then, we remove duplicate edges, as well as edges that cross the surface at concave regions, in order to account for material discontinuities. In our implementation, we use the ℓ^∞ distance metric, i.e., max-norm. Please refer to Figure 1-d, where edges of the visibility graph are visualized.

6.2. Bucket-Moore Algorithm

Once the visibility graph is initialized, it remains to compute shortest distances along the graph, which are used for defining summation nodes as described in Section 4.2. We have found maximum efficiency by adapting Moore's algorithm [Moo59] for distance computations on a regular grid where all edges have unit length, instead of using general-purpose shortest path algorithms such as Dijkstra or Floyd-Warshall [CLR90] or adapted versions [SOG06]. Moore's original algorithm resembles breadth-first search (BFS) and computes shortest distances from a node n_0 to all other nodes in the graph. It maintains two buckets, B_0 storing the current front of BFS, and B_1 storing the next front. It visits the nodes in B_0 and places their unvisited neighbors in B_1. Once B_0 is emptied, B_1 is shifted to B_0 and the integer distance is incremented. A node's distance is set upon removal from B_0. Moore's algorithm can compute all pairwise distances shorter than D_{max} in time $O(mn)$, where n is the number of nodes and m is the average number of nodes closer than D_{max}.

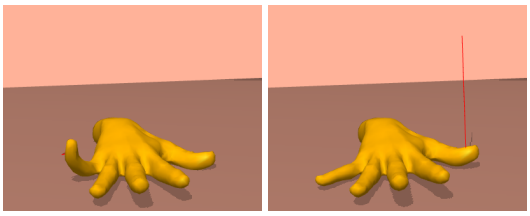

Figure 4: *Deformation of a hand with varying mechanical stiffness (The pinky is soft, while the thumb is hard). Our framework efficiently handles shape-matching regions of varying width.*

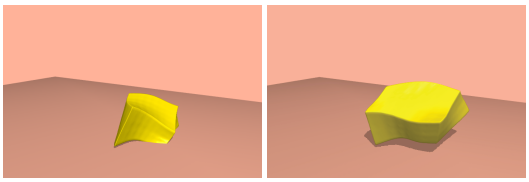

Figure 5: *Block undergoing collisions and plastic deformations, efficiently incorporated to our hierarchical fast summation.*

With the use of the max-norm distance metric, we can quantize edge lengths with integer values, assigning a length of one to an edge between two adjacent nodes at the maximum octree resolution. We similarly approximate the region widths w_i and $(1+\varepsilon)w_i$ with integer values. In this setting, we propose a bucket-based version of Moores's algorithm that computes shortest paths when all edge lengths are integers in a small range $[1, d_{max}]$. It also runs in $O(mn)$, although the constants are somewhat larger. The algorithm maintains $d_{max}+1$ buckets, and operates by visiting the nodes on bucket B_0. When a node n_i is removed from B_0, a node n_j adjacent to n_i at distance d may be added to bucket B_d. Once bucket B_0 is empty, buckets are shifted $B_d \leftarrow B_{d+1}$. Nodes that are visited store a temporary minimum distance, which may be later reduced. Our proposed *Bucket-Moore Algorithm* for finding shortest distances from a node n_0 to all other nodes closer than D_{max} may be summarized as follows:

1. Initialization:
   $k = 0$.
   For all nodes, unmark and set $d_{min} = \infty$.
   Put n_0 in B_0.
2. While B_0 is not empty
   Remove the first node n_i from B_0.
   If n_i is marked, discard it.
   Else: mark n_i; for each neighbor n_j of n_i:
   Compute $d = k + d(n_i, n_j)$.
   If ($d \geq n_j.d_{min}$ or $d \geq D_{max}$), discard n_j.
   Else: $n_j.d_{min} \leftarrow d$; add n_j to $B_{d(n_i,n_j)}$.
3. $k \leftarrow k+1$; shift buckets $B_d \leftarrow B_{d+1}$; $B_{d_{max}} = \{\}$.
4. If $k < D_{max}$ and some bucket is non-empty, repeat 2.

7. Implementation and Results

All our experiments were carried out on a 3.4 GHz Pentium-4 PC with 1 GB of memory. Next we describe several of the effects that can be achieved with our approach, we discuss implementation details for several features, and compare performance and features with the FLSM algorithm [RJ07].

Surface animation: We animate the surface by interpolating the deformation fields defined by nearby simulation nodes, similar to [MKN*04]. We have implemented the surface animation on the CPU, but it would be possible to do it directly on the GPU as described by [RJ07]. In order to efficiently detect nearby nodes after topological changes, we augment the visibility graph as described in [SOG06].

Comparison with FLSM - Performance, adaptivity, and inhomogeneity: Figure 1 shows an example where we compare FLSM and our octree shape matching. Using a regular lattice, 35 000 nodes are needed for correctly capturing the fingers. FLSM runs then at an average 2.5 fps, while our approach (with regular sampling, no octree) runs at 4.6 fps. Fast summation / shape matching, polar decompositions, and damping each consume roughly one third of overall computation time in our approach, and the other simulation examples have a similar distribution. In both FLSM and our approach, we have measured timings without low-level code optimizations, while the timings reported in [RJ07] include such optimizations (as reported by the authors in personal communication), and with these FLSM would run at about 10 fps.

But the power of our approach lies in its ability to accomodate adaptive sampling. Figure 1 also shows the same model with adaptive sampling. The resolution on the surface is as high as before when needed, but the resolution in the interior is much coarser. The model consists of 661 nodes, and runs at 222 fps with our method, almost two orders of magnitude faster than required by FLSM for resolving surface features. Note that in this example adaptive sampling is done once as a preprocess, and our algorithms for dynamic resampling and efficient distance computation are not required. One could use any other method for computing distances and determining summation nodes.

Figure 4 shows a hand model with different stiffness at each finger (Please watch the accompanying video). Such material inhomogeneity is achieved by varying the width w of shape matching regions. Our octree shape matching framework naturally allows this feature, which would however break the regularity required by FLSM.

Plasticity: Figure 5 shows plastic behavior of a deforming block under collisions. We achieve this behavior by adapting the plasticity model of [MHTG05] to our setting. In their model, each cluster stored a plasticity transformation matrix

Figure 6: *Complex scene with* 40 *deforming flowers. We employ a coarse sampling when the flowers are moved by the wind, for a total of* 5 680 *nodes in the scene. But we dynamically refine the models when touched by the user, as shown in the left image. Our (dynamic) adaptive sampling framework allows interactivity (*20 *fps) at the feature level in this complex scene.*

$\mathbf{S}_r$, which can be adopted in our model by modifying the computation of the per-region transformations $\mathbf{A}_r$ and $\mathbf{A}_{qq}$ (See Sections 3.1 and 4.3 for more details):

$$\mathbf{A}_r = \left(\boldsymbol{HF}_{i\in R_r}\left\{m_i\mathbf{x}_i\mathbf{x}_i^{0^T}\right\} - M_r\mathbf{c}_r\mathbf{c}_r^{0^T}\right)\mathbf{S}_r^T, \quad (9)$$

$$\mathbf{A}_{qq} = \left[\mathbf{S}_r\left(\boldsymbol{HF}_{i\in R_r}\{m_i\mathbf{x}_i^0\mathbf{x}_i^{0^T}\}\right)\mathbf{S}_r^T - \mathbf{S}_r\left(M_r\mathbf{c}_r^0\mathbf{c}_r^{0^T}\right)\mathbf{S}_r^T\right]^{-1} \quad (10)$$

Note that $\mathbf{A}_{qq}$ must be recomputed when $\mathbf{S}_r$ or the region itself changes. Finally, $\mathbf{S}_r$ is included in the region transform such that $\mathbf{T}_r = [\mathbf{R}_r\mathbf{S}_r \quad (\mathbf{c}_r - \mathbf{R}_r\mathbf{S}_r\mathbf{c}_r^0)]$.

Topology changes: Figure 3 shows a liver model being cut interactively. The model starts with 500 nodes and ends with 1 550. During cutting, we update the visibility graph, resample simulation nodes, and recompute summation nodes, as described in Section 5. Cutting also involves synthesizing new surfaces, and we follow the approach of [SOG06] for this purpose. The simulation, including collision handling, takes between 3.7 and 15.5 ms per frame. Resampling takes between 62 and 124ms.

(Self-)collision handling: The same Figure 3 also depicts interactive handling of self-collisions between cut surfaces. We reuse the shortest-path information (see Section 6) for processing collisions and self-collisions very efficiently. We approximate a distance field inside an object through a simple flooding algorithm that is seeded at the simulation nodes near the object's surface. We transform octree leaf cells with the transformations of corresponding simulation nodes, and we then test them for intersection with other nodes using the spatial hashing algorithm of [THM*03]. The penetration depth is given by the approximate distance field and is used for computing repulsive forces.

Dynamic LOD selection: Figure 6 shows a complex scene with 40 deformable flowers moving in the wind. When viewed from far away, each flower is discretized with 142 simulation nodes, and the total simulation runs at 20 fps. When the user interacts with a flower, we dynamically refine the sampling to capture the complexity of surface features and allow them to move independently. With the FLSM approach, the resolution required to resolve the thin petals would produce an explosion of the number of nodes. With our octree shape matching algorithm and dynamic LOD selection, however, the total number of nodes increases only by 6%, allowing full interactivity. Dynamic LOD updates are efficiently executed, and 3 simultaneous levels of refinement near the petals (352 new nodes) took only 121 ms. These timings do not include surface animation and rendering, which took 4 ms per flower.

8. Limitations and Future Work

We have presented a novel shape matching deformation algorithm that allows (dynamic) adaptive sampling. It enjoys the robustness and efficiency of other shape matching deformation models, but it also enables features like interactive topological changes or dynamic LOD selection. It is applicable in settings that favor plausibility, robustness, and efficiency, such as video games or surgical simulation.

Our method also presents some limitations. One of them, common to other geometric deformation methods, is the lack of physical fidelity and the difficulty to tune mechanical behavior based on measurable parameters. However, our method efficiently supports local control of mechanical stiffness, unlike previous shape matching methods.

Topological changes are much more efficient with our method than with previous shape matching approaches, but there is a practical bound on the number of regions that can be updated in an interactive manner. The same is true for dynamic LOD selection, and very drastic LOD changes could stall the method. These are, however, known limitations for all techniques that support dynamic adaptivity.

Our shape matching deformation model relies on the existence of a volumetric sampling, and cannot be directly used

for simulating shells or rods. It would be interesting to define shape matching deformation models for such objects.

Acknowledgements

We would like to thank the anonymous reviewers, members of the Computer Graphics Lab in Zurich and Alec Rivers and Doug James for their helpful comments. This research was supported in part by the NCCR Co-Me of the Swiss National Science Foundation.

References

[BJ05] BARBIČ J., JAMES D. L.: Real-time subspace integration for St. Venant-Kirchhoff deformable models. *Proc. of ACM SIGGRAPH* (2005).

[BLG94] BELYTSCHKO T., LU Y. Y., GU L.: Element-free Galerkin methods. *International Journal of Numerical Methods in Engineering 37* (1994).

[BNC96] BRO-NIELSEN M., COTIN S.: Real-time volumetric deformable models for surgery simulation using finite elements and condensation. *Computer Graphics Forum 15*, 3 (1996).

[BPGK06] BOTSCH M., PAULY M., GROSS M., KOBBELT L.: PriMo: Coupled prisms for intuitive surface modeling. *Proc. of Eurographics Symposium on Geometry Processing* (2006).

[BPWG07] BOTSCH M., PAULY M., WICKE M., GROSS M.: Adaptive space deformations based on rigid cells. *Proc. of Eurographics* (2007).

[CGC*02a] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIC Z.: Interactive skeleton-driven dynamic deformations. *Proc. of ACM SIGGRAPH* (2002).

[CGC*02b] CAPELL S., GREEN S., CURLESS B., DUCHAMP T., POPOVIC Z.: A multiresolution framework for dynamic deformations. *Proc. of ACM SIGGRAPH SCA* (2002).

[CLR90] CORMEN T., LEISERSON C., RIVEST R.: *Introduction to Algorithms, 2nd Ed.* MIT Press, 1990.

[DDCB01] DEBUNNE G., DESBRUN M., CANI M. P., BARR A. H.: Dynamic real-time deformations using space and time adaptive sampling. *Proc. of ACM SIGGRAPH* (2001).

[GGN06] GAO J., GUIBAS L. J., NGUYEN A.: Deformable spanners and its applications. *Computational Geometry: Theory and Applications 35*, 1 (2006).

[GKS02] GRINSPUN E., KRYSL P., SCHRÖDER P.: CHARMS: A simple framework for adaptive simulation. *Proc. of ACM SIGGRAPH* (2002).

[GM97] GIBSON S. F., MIRTICH B. V.: *A Survey of Deformable Modeling in Computer Graphics*. Tech. rep., Mitsubishi Electric Research Laboratory, 1997.

[JP99] JAMES D. L., PAI D. K.: ArtDefo: Accurate real-time deformable objects. *Proc. of ACM SIGGRAPH* (1999).

[JP02] JAMES D. L., PAI D. K.: DyRT: Dynamic response textures for real-time deformation simulation with graphics hardware. *Proc. of ACM SIGGRAPH* (2002).

[MDM*02] MÜLLER M., DORSEY J., MCMILLAN L., JAGNOW R., CUTLER B.: Stable real-time deformations. *Proc. of ACM SIGGRAPH Symposium on Computer Animation* (2002).

[MHHR06] MÜLLER M., HEIDELBERGER B., HENNIX M., RATCLIFF J.: Position based dynamics. *Proc. of VRIPhys* (2006).

[MHTG05] MÜLLER M., HEIDELBERGER B., TESCHNER M., GROSS M.: Meshless deformations based on shape matching. *Proc. of ACM SIGGRAPH* (2005).

[MKN*04] MÜLLER M., KEISER R., NEALEN A., PAULY M., GROSS M., ALEXA M.: Point-based animation of elastic, plastic, and melting objects. *Proc. of ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (2004).

[Moo59] MOORE E. F.: The shortest path through a maze. *Annals of the Harvard Computation Laboratory 30* (1959), 285–292.

[NMK*05] NEALEN A., MÜLLER M., KEISER R., BOXERMANN E., CARLSON M.: Physically based deformable models in computer graphics. *Eurographics STAR* (2005).

[OGRG06] OTADUY M. A., GERMANN D., REDON S., GROSS M.: Adaptive deformations with fast tight bounds. In *Proc. of SCA* (2006).

[PKA*05] PAULY M., KEISER R., ADAMS B., DUTRE P., GROSS M., GUIBAS L. J.: Meshless animation of fracturing solids. *Proc. of ACM SIGGRAPH* (2005).

[PW89] PENTLAND A., WILLIAMS J.: Good vibrations: Modal dynamics for graphics and animation. *Proc. of ACM SIGGRAPH* (1989).

[RJ07] RIVERS A. R., JAMES D. L.: FastLSM: Fast lattice shape matching for robust real-time deformation. *Proc. of ACM SIGGRAPH* (2007).

[SOG06] STEINEMANN D., OTADUY M. A., GROSS M.: Fast arbitrary splitting of deforming objects. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2006).

[THM*03] TESCHNER M., HEIDELBERGER B., MÜLLER M., POMERANETS D., GROSS M.: Optimized spatial hashing for collision detection of deformable objects. *Proc. of VMV* (2003).

[THMG04] TESCHNER M., HEIDELBERGER B., MÜLLER M., GROSS M.: A versatile and robust model for geometrically complex deformable solids. *Proc. of Computer Graphics International* (2004).

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. *Proc. of ACM SIGGRAPH* (1987).