EUROGRAPHICS 2005 / M. Alexa and J. Marks
(Guest Editors)

Volume 24 (2005), Number 3

Efficient Raytracing of Deforming Point-Sampled Surfaces

Bart Adams[1] Richard Keiser[2] Mark Pauly[3] Leonidas J. Guibas[3] Markus Gross[2] Philip Dutré[1]

[1]Katholieke Universiteit Leuven [2]ETH Zürich [3]Stanford University

Abstract

We present efficient data structures and caching schemes to accelerate ray-surface intersections for deforming point-sampled surfaces. By exploiting spatial and temporal coherence of the deformation during the animation, we are able to improve rendering performance by a factor of two to three compared to existing techniques. Starting from a tight bounding sphere hierarchy for the undeformed object, we use a lazy updating scheme to adapt the hierarchy to the deformed surface in each animation step. In addition, we achieve a significant speedup for ray-surface intersections by caching per-ray intersection points. We also present a technique for rendering sharp edges and corners in point-sampled models by introducing a novel surface clipping algorithm.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling–Surface Representations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism–Raytracing

1. Introduction

Point-based surface representations have recently become popular in computer graphics and geometric modeling [AGP*04]. As an alternative to traditional surface models, such as spline patches or polygonal meshes, they allow for simple and efficient dynamic re-sampling due to minimal consistency requirements [PKKG03]. At the same time, their explicit nature supports highly detailed surfaces, which is crucial for realistic computer animations in feature films and games. These two properties make point-based surface representations particularly suitable for animating complex deformable objects, as has recently been demonstrated in [MKN*04], [CZ04], and [PPG04].

To visualize point-sampled surfaces, splat-based approaches have been most popular, e.g., [RL00], [ZPvG01]. However, as has been shown in [AAN05], splatting results in poor image quality under magnification. Moreover, splatting-based rendering algorithms typically do not account for secondary effects such as shadows or reflections, which greatly enhance the realism of an animation. Raytracing, on the other hand, naturally incorporates these secondary rendering effects. Various authors have proposed raytracing schemes for static point-sampled surfaces ([SJ00], [AA03a], [AA03b], [WS03]) using static data structures, such as kd-trees, octrees, or bounding sphere hierarchies to improve the performance of ray-surface intersection calculations.

In this paper we propose a new raytracing algorithm for rendering **deforming** point-sampled surfaces, and address the challenges that arise when balancing the efficiency of dynamic updates vs. spatial queries on the data structure used to accelerate the intersection tests.

We assume a reduced deformation model, where the displacement of the object surface can be expressed with significantly fewer parameters than the number of degrees of freedom of the surface itself. A typical example is the free-form deformation approach of [MKN*04], where a high-resolution surface is embedded within a low-resolution simulation domain. We render such a deformable model by constructing and maintaining a bounding sphere hierarchy to accelerate ray-surface intersections. Rendering performance is greatly improved by exploiting different aspects of spatial and temporal coherence:

- The compact representation of the deformation allows efficient local updates of the sphere hierarchy by bounding the deviation of the displacements of the surface elements.
- The surface representation and corresponding bounding sphere tree are maintained in a lazy fashion when the object is deformed. Only those elements are updated that po-

tentially experience a ray intersection, while other parts of the model remain unchanged.

- For each ray, we first test against the intersected sphere node from the previous time step. This gives a good upper bound for the current intersection depth, significantly culling the number of spheres to be updated and tested for intersection.

We also present a novel technique for rendering sharp edges and corners in point-sampled models by introducing a new surface clipping algorithm.

2. Related Work

Raytracing Point-Sampled Geometry. Schaufler and Jensen [SJ00] were the first to propose a ray-surface intersection algorithm for point-sampled surfaces. They define the intersection as a weighted average of disk intersections within a cylinder around the ray. This leads to a slightly view-dependent surface, which can be problematic when rendering animations. Adamson and Alexa [AA03b] presented a different technique based on an iterative projection procedure. The ray is intersected with spheres which enclose the surface, and inside the spheres with local polynomial surface approximations. As a result, their surface definition is view-independent. Our surface intersection algorithm is based on their follow-up work [AA03a] (see also [AA04]) on approximating and intersecting surfaces from points. The surface is implicitly defined based on normal directions and weighted average point positions. The core of the intersection algorithm consists of three steps. First, a support plane is constructed using an approximate normal direction. Then a polynomial approximation is constructed over the support plane serving as a local approximation of the surface. Finally, the ray is intersected with this polynomial. This procedure is repeated until convergence. As a spatial data structure they use a bounding sphere hierarchy which has shown to be very effective. [AAN05] proposes an extension of this scheme to speed up ray-surface intersections by exploiting image- and object-space coherence. Surface intersection points are used to construct view-dependent bilinear surface approximations, which are rendered as quads on the GPU using forward projection. These patches are reused in consecutive frames, leading to performance gains of up to 50% for non-deforming objects.

Accelerated Raytracing. As reviewing all of the acceleration techniques is out of the scope of this paper we refer to [Gla88], [Gla89], [Shi00], and [3DO] and the excellent state of the art report on real-time raytracing by Wald and coworkers [WPS*03]. Most relevant to this paper is the second part of their report where various approaches to accelerate raytracing in dynamic environments are discussed, e.g., [RSH00], [LAM01] and [WBS03]. However, most of this work focuses on (hierarchical) motion, where whole groups of triangles are moved under the same affine transformation. Our technique on the other hand focuses on highly detailed point-sampled surfaces which deform in a free-from manner.

We exploit temporal coherence using a lazy updating scheme similar to [MSH*92]. They use an octree acceleration data structure, which is only built up to a certain level in each frame. Nodes at lower levels are only updated when necessary. In our setting, sphere nodes of the bounding sphere hierarchy and surface patches are only updated when queried, i.e. tested for ray intersection.

A caching scheme similar to ours is used in [AK87], where recently referenced hypercubes are cached and retrieved for intersection testing in the next time step. We store per-ray intersected sphere nodes from frame to frame and test cached spheres for intersection first. This allows efficient culling of unnecessary sphere nodes for increased rendering performance.

Finally, the update of our bounding sphere hierarchy is inspired by the technique presented in [JP04]. By bounding the deformation of the surface within each sphere, a conservative estimate of the updated sphere radius can be computed for each time step.

3. Surface Animation

Our method for rendering deformable objects is based on the animation framework of [MKN*04] that allows physics-based simulation of elastically and plastically deforming solids. To solve the equations of continuum mechanics, the simulation volume is discretized with a set $\{p_j\}$ of simulation nodes (see Figure 1, left). The boundary surface of the solid is represented by a (typically much larger) set $\{s_i\}$ of surface elements (surfels). When deforming the material, the displacements of the surfels are determined from spatially adjacent simulation nodes using a free-form deformation approach. In principle, however, our rendering algorithm can be used with any animation method that applies the idea of an embedded surface, e.g., mass-spring based systems or FEM-based approaches.

Initially, we assign to each surfel s_i a set of neighboring simulation nodes p_j (see Figure 1, middle). After an animation step, the new position $\mathbf{x}'_{s_i}$ of s_i is computed using a first order accurate approximation of the displacements $\mathbf{u}_{p_j}$ of the neighboring simulation nodes p_j as [MKN*04]:

$$\mathbf{x}'_{s_i} = \mathbf{x}_{s_i} + \sum_{p_j} \overline{\omega}^{h_i}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}} (\mathbf{u}_{p_j} + \nabla^T_{\mathbf{u}} \mathbf{u}_{p_j} \mathbf{d}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}), \tag{1}$$

where $\mathbf{d}_{\mathbf{x},\mathbf{y}} = \mathbf{y} - \mathbf{x}$, $\overline{\omega}^h_{\mathbf{x},\mathbf{y}} = \omega^h_{\mathbf{x},\mathbf{y}} / \sum_{\mathbf{y}} \omega^h_{\mathbf{x},\mathbf{y}}$, and $\omega^h_{\mathbf{x},\mathbf{y}}$ is a smoothly decaying weight function with support radius h. We use the compactly supported radial spline function

$$\omega^h_{\mathbf{x},\mathbf{y}} = \omega(r) = \begin{cases} 1 - 6r^2 + 8r^3 - 3r^4 & r \le 1 \\ 0 & r > 1, \end{cases} \tag{2}$$

where $r = (\|\mathbf{x} - \mathbf{y}\|)/h$. We reuse the MLS approximation

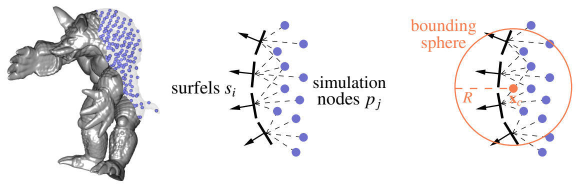

Figure 1: *Left and middle: the surfels are embedded in the simulation domain. Right: bounding sphere. Note that the simulation nodes are not necessarily contained by the sphere.*

of $\nabla_{\mathbf{u}}\mathbf{u}_{p_j}$, which is computed when solving the continuum mechanics equations as described in [MKN*04]. Similar to [PKKG03], both the surfel center and its tangent axes are deformed, yielding the deformed surfel normal $\mathbf{n}'_{s_i}$.

By de-coupling the sampling of the simulation domain from the sampling of the boundary surfaces, this method allows efficient animation of highly detailed models using the smooth displacement field $\mathbf{u}$. We exploit the implicit spatial coherence for efficient updates of the bounding sphere hierarchy, as will be discussed below.

4. Rendering Framework

In this section, we describe the rendering framework that we use to generate subsequent frames of an animated model. We first discuss the bounding sphere hierarchy used to accelerate ray-surface intersections. Next, we outline the ray-surface intersection algorithm which is at the core of our rendering algorithm. We describe how rendering time can be reduced by caching of per-ray intersected sphere nodes, before we introduce clipping relations for rendering sharp edges and corners. Finally, we discuss how these individual pieces are integrated to yield an efficient raytracing algorithm for deforming point-sampled surfaces.

4.1. Bounding Sphere Hierarchy

To accelerate the ray-surface intersection tests, we use a bounding sphere hierarchy [AA03b]. The hierarchy is only built once for the undeformed object and dynamically updated in each time step to conform with the deformed object [JP04].

4.1.1. Initial Sphere Hierarchy

The hierarchy is built top-down starting with a sphere wrapped around all surfels. Each sphere is recursively split into two child spheres until eventually a sphere contains only one single surfel. Splitting is done according to the plane defined by the longest axis of the surfels' bounding box, similar to [RL00]. We compute the initial sphere centers $\mathbf{x}_c$ and optimal radii R using the miniball algorithm of [Gar99]. For each sphere bounding surfels s_i, we also keep a list of the simulation nodes p_j that define the displacement of the surfels s_i (see Figure 1, right).

4.1.2. Sphere Update

As building a new sphere hierarchy in each time step is computationally too expensive, we dynamically update the hierarchy built for the undeformed object using the deformation field $\mathbf{u}(\mathbf{x})$ (see also [JP04]).

Center Update. The displaced sphere center $\mathbf{x}'_c$ is computed in the same manner as we compute the displaced surfel positions:

$$\begin{aligned} \mathbf{x}'_c &= \mathbf{x}_c + \sum_{p_j} \overline{\omega}^{R}_{\mathbf{x}_c,\mathbf{x}_{p_j}} \left(\mathbf{u}_{p_j} + \nabla^T_{\mathbf{u}}\mathbf{u}_{p_j}\mathbf{d}_{\mathbf{x}_c,\mathbf{x}_{p_j}}\right) && (3) \\ &\equiv \mathbf{x}_c + \mathbf{u}_c. && (4) \end{aligned}$$

Radius Update. The new radius R' is conservatively estimated from the maximal distance between the deformed surfels (Equation 1) and the new sphere center (Equation 4) using basic linear algebra and the triangle inequality:

$$\begin{aligned} R' &= \max_{s_i} \|\mathbf{x}'_{s_i} - \mathbf{x}'_c\|_2 && (5) \\ &= \max_{s_i} \|(\mathbf{x}_{s_i} - \mathbf{x}_c) + \sum_{p_j} \overline{\omega}^{h_i}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}} (\mathbf{u}_{p_j} - \mathbf{u}_c) \\ &\quad + \sum_{p_j} \overline{\omega}^{h_i}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}} \nabla^T_{\mathbf{u}}\mathbf{u}_{p_j}\mathbf{d}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}\|_2 && (6) \\ &\leq \max_{s_i} \|\mathbf{x}_{s_i} - \mathbf{x}_c\|_2 + \sum_{p_j} \max_{s_i} |a_{s_i,p_j}| \|\mathbf{u}_{p_j} - \mathbf{u}_c\|_2 \\ &\quad + \sum_{p_j} \max_{s_i} \|\mathbf{b}_{s_i,p_j}\|_2 \|\nabla^T_{\mathbf{u}}\mathbf{u}_{p_j}\|_F && (7) \\ &\equiv R + \sum_{p_j} \mathbf{A}_j \mathbf{U}_j + \sum_{p_j} \mathbf{B}_j \nabla \mathbf{U}_j && (8) \\ &= R + \mathbf{A}^T\mathbf{U} + \mathbf{B}^T\nabla\mathbf{U} && (9) \end{aligned}$$

where $a_{s_i,p_j} = \overline{\omega}^{h_i}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}$, $\mathbf{b}_{s_i,p_j} = \overline{\omega}^{h_i}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}\mathbf{d}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}$, and $\|\nabla^T_{\mathbf{u}}\mathbf{u}_{p_j}\|_F$ is the Frobenius norm of the directional gradient of the displacement. We can bring $\mathbf{u}_c$ into the summation since the weights $\overline{\omega}^{h_i}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}$ sum up to 1 by construction. The entries $\mathbf{A}_j$ and $\mathbf{B}_j$ remain constant during the animation and can thus be precomputed once in the reference system (see also Section 4.1.3). Note that the center and radius updates have time complexity linear in the number of simulation nodes associated with a bounding sphere, not in the number of bounded surfels. This is important as the number of simulation nodes is typically much smaller than the number of surfels. The radius update is always done with respect to the initial (optimal) bounding spheres, i.e., the radius can both increase and decrease over time. The sphere hierarchy thus maintains its tight fit even for highly elastic materials that expand and shrink significantly during an animation.

The above updates of the sphere center and radius are only performed for non-leaf nodes. Since each leaf node bounds a single surfel, we can use the updated surfel's position (Equation 1) and compute the updated radius from the surfel's deformed tangent axes.

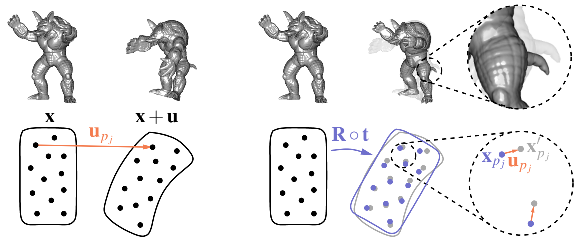

Figure 2: *Left: the displacements are computed relative to the reference system. Right: the displacements are computed relative to the optimally transformed reference system. This results in smaller relative displacements.*

4.1.3. Optimal Rigid Transformation

As illustrated in Figure 2, the displacements $\mathbf{u}$ of surfels and simulation nodes are computed relative to the undeformed positions $\mathbf{x}$. The latter are defined in the reference system (material coordinates), while the displaced positions $\mathbf{x}'$ are specified in the deformed system (world coordinates). Although the bounding radius update of Equation 9 is invariant under uniform translations (since then $\mathbf{U}_j = 0$ and $\nabla\mathbf{U}_j = 0$), the sphere radii grow under uniform rotations.

We can address this problem by factoring out the *optimal rigid transformation*. After each iteration, we transform the reference system rigidly by computing the optimal global rotation and translation based on geometric algebra [LFDL98], similar to [TW88]. This will align the transformed reference system as closely as possible with the deformed system, resulting in smaller $\mathbf{U}_j$'s and $\nabla\mathbf{U}_j$'s and thus in smaller bounds for the updated sphere nodes (see Figure 2, right).

Translation. The optimal translation $\mathbf{t} = \bar{\mathbf{x}}'_m - \bar{\mathbf{x}}_m$ is the difference between the centers of mass $\bar{\mathbf{x}}'_m = \sum_{p_j} \overline{m}_{p_j}\mathbf{x}'_{p_j}$ and $\bar{\mathbf{x}}_m = \sum_{p_j} \overline{m}_{p_j}\mathbf{x}_{p_j}$ of the displaced and the reference simulation nodes respectively, where $\overline{m}_{p_j} = m_{p_j}/\sum_{p_j} m_{p_j}$, and m_{p_j} is the mass of a simulation node p_j.

Rotation. The optimal rotation $\mathbf{R} = \mathbf{V}\mathbf{U}^T$ is computed in a least squares sense by computing the singular value decomposition of $\mathbf{F} = \mathbf{U}\mathbf{W}\mathbf{V}^T$, where

$$\mathbf{F} = \sum_{p_j} m_{p_j}^2 (\mathbf{x}_{p_j} - \bar{\mathbf{x}}_m)(\mathbf{x}'_{p_j} - \bar{\mathbf{x}}'_m)^T. \qquad (10)$$

This linear transformation is applied to the reference position of all simulation nodes, surfels and bounding sphere centers when used during raytracing. The quantities $\mathbf{d}_{x,y}$ and $\overline{\omega}^h_{\mathbf{x},\mathbf{y}}$ remain constant under rigid transformations and therefore the entries $\mathbf{A}_j$ and $\mathbf{B}_j$ of Equation 9 need to be computed only once as discussed before.

4.1.4. Tightness of Radius Update

For the example given in Figure 6, the updated radius is on average 4 times larger than the radius of the smallest sphere wrapped around the deformed points (using the algorithm of [Gar99]). If we look at the lowest levels, the ratio is only 1.0, 1.2, 1.2, ... However, if we go up the hierarchy, the radius estimate depends on more and more simulation nodes and therefore this ratio increases to up to 20.0 for the root node (see Figure 3). In our setting, this is somewhat problematic as these spheres are also the biggest ones and are thus hit by many rays during raytracing.

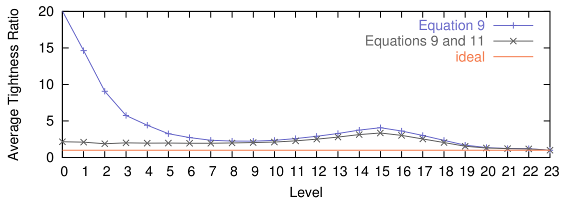

Figure 3: *Average ratio of updated sphere radius and radius of smallest enclosing sphere at all levels (level 0 is the root node) for the cannon ball armadillo sequence (Figure 6).*

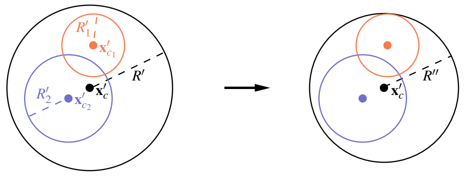

Figure 4: *Left: The parent and child spheres are updated according to Equations 4 and 9. Right: In this case, a tighter bound for the parent's radius can be found by looking at the children's radii using Equation 11.*

To alleviate this problem, we compute a second conservative radius estimate R'', and choose the smallest of the two estimates to update the bounding sphere. Assume that both child nodes of a sphere have already been updated in the current animation step. Let $\mathbf{x}'_{c_1}$, $\mathbf{x}'_{c_2}$ and R'_1, R'_2 be the corresponding updated sphere centers and radii, respectively. We know that the surfels in the parent node are bounded by the union of the two child spheres. As illustrated in Figure 4, we can thus compute R'' as

$$R'' = \max(\|\mathbf{x}'_c - \mathbf{x}'_{c_1}\| + R'_1, \|\mathbf{x}'_c - \mathbf{x}'_{c_2}\| + R'_2). \qquad (11)$$

Since the time complexity of this expression is constant, the additional radius estimate comes essentially for free, as compared to the estimate in Equation 9, which has time complexity linear in the number of associated simulation nodes. If one of the children is not updated yet in this animation step, we choose R' as the radius update, otherwise we choose $\min(R', R'')$.

The combination of these two bounds leads to an average increase of the sphere radius by only a factor of 1.9 as compared to the tightest possible sphere. For the highest level, i.e. the root sphere, the average ratio drops to 2.1 instead of the aforementioned 20.0 (see Figure 3). This results in 48% less unnecessary ray-sphere intersection tests on average.

4.2. Ray-Surface Intersections

We start by briefly describing the ray-surface intersection algorithm for point-sampled surfaces. Next, we demonstrate how we exploit temporal coherence to increase the performance of intersection tests.

4.2.1. Intersection Algorithm

When a ray hits a leaf node, we intersect it with the ellipse defined by the node's surfel. This intersection point $\mathbf{x}$ serves as an initial guess for the following iterative procedure [AA04]:

- Compute the weighted average $\mathbf{a}(\mathbf{x})$ of the deformed surfel positions and the weighted average $\mathbf{n}(\mathbf{x})$ of surfel normals in the neighborhood of $\mathbf{x}$ within a distance h:

$$\mathbf{a}(\mathbf{x}) = \frac{\sum_{s_i} \omega^h_{\mathbf{x},\mathbf{x}'_{s_i}} \mathbf{x}'_{s_i}}{\sum_{s_i} \omega^h_{\mathbf{x},\mathbf{x}'_{s_i}}}, \quad \mathbf{n}(\mathbf{x}) = \frac{\sum_{s_i} \omega^h_{\mathbf{x},\mathbf{x}'_{s_i}} \mathbf{n}'_{s_i}}{\|\sum_{s_i} \omega^h_{\mathbf{x},\mathbf{x}'_{s_i}} \mathbf{n}'_{s_i}\|}. \quad (12)$$

  These define a plane $f(\mathbf{x}) = \mathbf{n}(\mathbf{x})^T(\mathbf{x} - \mathbf{a}(\mathbf{x})) = 0$.
- Test for convergence. The iterative intersection algorithm has converged if $|f(\mathbf{x})| < \varepsilon$. If not converged, intersect the ray with this plane. This yields a new intersection point $\mathbf{x}'$. If $\mathbf{x}'$ falls outside the node's bounding sphere, the iteration is stopped and the intersection point is rejected.
- Repeat the above steps until convergence.

As shown in [AA03a] this procedure quickly converges depending on the initial guess. In our examples we need 3 iterations on average before convergence.

We avoid expensive nearest neighbor queries by using a static surfel neighborhood for the leaf nodes. The surfel neighbors are precomputed in the undeformed system. Whenever a ray hits a leaf node we use the stored surfels s_i as neighbors for the initial intersection point. Note that the weights $\omega^h_{\mathbf{x},\mathbf{x}'_{s_i}}$ are computed against the current intersection point, not against the center of the leaf node. For the examples given in this paper we used fixed neighborhoods of 16 surfels.

4.2.2. Sphere Node Caching

We can reduce the number of ray-sphere intersection tests by caching intersected sphere nodes from frame to frame. For each ray we store the sphere node where we found the closest intersection point (i.e. the one with the smallest t-value). In the next frame, we first test for intersection using this cached sphere node (if there is any). In many cases this gives us a good upper bound for the intersection depth of the ray. Next, we test against the root node and descend the hierarchy as usual. However, thanks to the upper bound of the intersection depth, many more spheres are trivially culled resulting in less sphere updates and less ray-surface intersection tests. We experienced a decrease of 27% on average in the number of performed intersection tests thanks to the sphere node caching.

4.3. Clipping

To be able to render sharp edges and corners we have incorporated a new clipping technique. Surfels are grouped in surfel collections between which we explicitly store clipping relations. Suppose we have a sharp edge defined as the intersection of two surfaces S_1 and S_2. Whenever a ray intersects a surfel of surface S_1 we test if the intersection point is clipped by surface S_2. If so, the intersection is rejected.

To be able to perform the clipping test, we store for each surfel of S_1 the two nearest surfels of S_2 and vice versa (these are precomputed and cached for the undeformed surfaces). Clipping is performed using these two nearest surfels of the other surface using the technique proposed in [WTG04]. Note that clipping is done in (deformed) object space, as opposed to [WTG04] where clipping is performed in image space. This has the advantage that we are able to anti-alias the sharp edges and corners by super-sampling (see Figure 8).

4.4. Putting It All Together

In this section we describe how the individual pieces described above are combined to yield the final rendering algorithm for deformable models.

Precomputation. For each surfel s_i compute its surfel neighbors. If there are clipping relations, precompute the clipping neighbors for the relevant surfels. Also compute the simulation nodes p_j and the corresponding weights $\overline{\omega}^{h_i}_{\mathbf{x}_{s_i},\mathbf{x}_{p_j}}$ that define the displacement of s_i. Build the initial bounding sphere hierarchy for the undeformed object. For each sphere keep a list of simulation nodes p_j used by the surfels bounded by the sphere and keep the corresponding weights $\overline{\omega}^{R}_{\mathbf{x}_c,\mathbf{x}_{p_j}}$. Precompute the entries in $\mathbf{A}$ and $\mathbf{B}$ for each sphere (Equation 9).

Rendering. For each ray, test for intersection against the cached sphere node before testing against the whole hierarchy. When a node is visited for the first time, update the center and the radius using Equations 4 and 9. If a node is already visited before, update the radius using Equation 11, if possible. When eventually the ray hits a leaf node, update the surfel s_i corresponding to this node and its surfel neighbors using Equation 1 (if not already done before in this time step). For leaf nodes we use the center of the associated surfel as the center of the sphere. Intersect the ray with the ellipse defined by s_i. Perform the iterative intersection algorithm. If necessary, check whether the intersection point is rejected by one or more clipping surfaces.

Note that both the update of the sphere nodes and the update of the surfels are done in a lazy manner. Only the nodes and surfels visited in this time step are updated, leading to additional savings in computation time.

Optimization For Shadow Rays. When traversing the hierarchy for shadow rays, we can reduce the number of ad-

Model	#Surfels	#Sim. Nodes	#Spheres	#Levels
Armadillo	170k	453	346k	23
Goblin	100k	502	200k	24
CSG Head	91k	253	182k	23
Elastic Ball	6k	88	10k	14

Table 1: *Sampling statistics for the different models used in our examples.*

ditional spheres to be updated. If a shadow ray intersects a parent sphere of which only one child is already updated, we first test the ray against the updated child. As for shadow rays any intersection point counts (i.e. we do not have to search for the closest one), most of the time the other child branch does not have to be updated nor traversed.

5. Results

We have tested our implementation using four animation sequences, all rendered at a resolution of 512 by 384 pixels. The sampling statistics are given in Table 1, average rendering times (averaged over 3 runs on a Pentium 4, 3GHz, 512Mb ram) are indicated in Figure 5.

The first animation shows the armadillo being hit by a cannon ball (Figure 6). As can be seen on the left plot in Figure 5, we achieve an average speedup of a factor 2.1 (8.5 seconds per frame compared to 18.5 seconds per frame on average) compared to the naive technique, where the sphere hierarchy is rebuilt in each time step. In the beginning and end of the animation we clearly gain from caching sphere nodes as the object does not deform significantly. In the middle of the animation we also gain from the fact that a large part of the armadillo is occluded by the cannon ball. This results in less sphere node and surface updates.

The second sequence shows a goblin creature practicing gymnastics on the parallel bars (Figure 7). Timing statistics are given on the second graph in Figure 5. Again, we obtain a speedup of a factor 2.1 (6 seconds per frame compared to 12.8 seconds per frame on average). In this sequence we gain from the fact that large parts of the goblin are self-occluded and do not have to be updated from frame to frame. Note that the middle curve touches the upper curve around frame 160. Here the goblin is deformed the most, resulting in the worst fit of the bounding sphere hierarchy.

The third sequence shows three bouncing heads (Figure 8). Each head consists of three surfaces between which explicit clipping relations are defined (see Section 4.3). The timings on Figure 5 indicate a speedup of a factor 2.2 (19 seconds per frame compared to 42.8 seconds per frame on average).

Finally, the last animation shows 40 elastic balls being thrown inside a hollow cube (see Figure 9). Thanks to the dynamic sphere updates and the sphere node caching we obtain an average speedup of a factor 3 compared to the naive technique (15 seconds per frame compared to 45 seconds per frame on average).

6. Discussion and Future Work

As demonstrated in Figure 5, exploiting spatial and temporal coherence leads to significant performance gains over the naive approach. Speedups are particularly high if the number of surfels is large compared to the number of simulation nodes, since sphere updates are linear in the number of simulation nodes. Our method also performs well for animations where large parts of the surface are occluded and thus not touched by any rays (e.g. for the animation of Figure 9).

Situations may arise however where refitting (parts of) the sphere hierarchy is more efficient than lazy updating. As lazy updating results in sub-optimal bounding spheres, it also results in more ray sphere intersection testing compared to the optimal sphere hierarchy. Our experiments showed that the break-even point (speedup of 1) varies significantly with the specific sequence. Therefore, one might want to devise heuristics to decide whether to use lazy updating or to perform a (full) sphere hierarchy update.

Storing and reusing spatial and temporal proximity information naturally leads to increased memory consumption. The main bottlenecks are the sphere nodes: for each node we keep a list of simulation nodes and the corresponding weights $\overline{\omega}^R_{\mathbf{x}_c,\mathbf{x}_{p_j}}$, and entries $\mathbf{A}_j$ and $\mathbf{B}_j$ (see Equation 9). However, for the armadillo sequence, where all the surfels have 16 simulation node neighbors, the sphere nodes have to store only 16.4 simulation nodes on average, since child nodes share their simulation nodes with the associated surfels. Storing static surfel neighborhoods requires an additional 16 pointers per surfel. For the caching of per-ray intersected sphere nodes we can trade off cache size with cache hit rate by setting a maximum cache size.

Memory overhead can be significantly reduced when many instances of the same model are present in a scene (e.g. the animations of Figures 8 and 9). In such cases, we can reuse the same (undeformed) sphere node hierarchy, even if the individual instances deform differently. Since the weights $\overline{\omega}^R_{\mathbf{x}_c,\mathbf{x}_{p_j}}$, entries $\mathbf{A}_j$ and $\mathbf{B}_j$, and neighborhood relations are constant over all instances, we only need to store the properties of the deformed simulation nodes, sphere nodes, and surfels for each instance.

Tighter bounds on the radius update could be obtained using a hierarchical approach when computing the optimal rigid transformation (see Section 4.1.3). Instead of computing a single transformation for the whole object, one could segment the simulation nodes in different sets and compute an optimal transformation for each set similar to [MHTG05]. For example, the model of Figure 7 could be split into two sets, one for the body and one for the hands. This yields smaller simulation node displacements and thus smaller bounds for the updated radii.

Currently, our method is only suitable for animated models with fixed topology. For highly dynamic substances, such as fracturing solids or fluids, our caching schemes would fail since neighborhood relations change too frequently. However, we plan to extend our method with dynamic up- and down-sampling of surfels and simulation nodes to better handle models that experience extreme deformations. As resampling is often a very localized operation, we expect that updating of cached neighbors will be fairly efficient.

We want to stress that even though our method is implemented in the context of a point-based animation framework, our dynamic sphere hierarchy can also be used for other surface representations. For example, if the boundary surface of the solid is given as a polygonal mesh embedded in an FEM simulation grid [MG04], the same algorithms can be used by applying the deformation field to the mesh vertices.

7. Conclusion

We have introduced a new method for efficient raytracing of animated point-sampled surfaces. Central to our method is a dynamic bounding sphere hierarchy that is used to accelerate ray-surface intersection tests. By de-coupling the sampling of the simulation domain from the sampling of the boundary surfaces, we are able to update the hierarchy by only looking at the deformation of the simulation nodes. Moreover, the update is done in a lazy manner, only touching spheres and surfels that are actually used during rendering. We show how this dynamic update and additional caching of intersected sphere nodes lead to an efficient raytracing algorithm obtaining speedups of over a factor of 2 compared to existing techniques. Finally, our method incorporates the rendering of sharp edges and corners which are explicitly defined using surface clipping relations.

Acknowledgements. The authors wish to acknowledge support from NSF grants CARGO-0138456, ITR-0205671, ARO grant DAAD19-03-1-033, and NIH Simbios Center grant 1091129-1-PABAE. Bart Adams is funded as a Research Assistant by the Fund for Scientific Research - Flanders, Belgium (F.W.O.-Vlaanderen).

References

[3DO] http://www.realtimerendering.com/int/. 2

[AA03a] Adamson A., Alexa M.: Approximating and intersecting surfaces from points. In *SGP '03: Proc. of the Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2003), Eurographics Association, pp. 230–239. 1, 2, 5

[AA03b] Adamson A., Alexa M.: Ray tracing point set surfaces. In *Proc. of Shape Modeling International* (2003), pp. 272–279. 1, 2, 3

[AA04] Alexa M., Adamson A.: On normals and projection operators for surfaces defined by point sets. In *Proc. of the Eurographics/ACM SIGGRAPH Symposium on Point-Based Graphics* (2004), Eurographics Association. 2, 5

[AAN05] Adamson A., Alexa M., Nealen A.: Adaptive sampling of intersectable models exploiting image and object-space coherence. In *Proc. of the ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games* (2005). 1, 2

[AGP*04] Alexa M., Gross M., Pauly M., Pfister H., Stamminger M., Zwicker M.: *Point-Based Computer Graphics*. ACM SIGGRAPH, 2004. Course Notes. 1

[AK87] Arvo J., Kirk D.: Fast ray tracing by ray classification. In *SIGGRAPH '87: Proc. of the 14th annual conference on Computer graphics and interactive techniques* (1987), ACM Press, pp. 55–64. 2

[CZ04] Chang J., Zhang J. J.: Mesh-free deformations. In *Comp. Anim. Virtual Worlds* (2004), pp. 211–218. 1

[Gar99] Gartner B.: Fast and robust smallest enclosing balls. In *Proc. of the European Symposium on Algorithms 1999* (1999), pp. 325–338. 3, 4

[Gla88] Glassner A. S.: Spacetime ray tracing for animation. *IEEE Comput. Graph. Appl. 8*, 2 (1988), 60–70. 2

[Gla89] Glassner A. S. (Ed.): *An introduction to ray tracing*. Academic Press Ltd., 1989. 2

[JP04] James D. L., Pai D. K.: Bd-tree: output-sensitive collision detection for reduced deformable models. *ACM Trans. Graph. 23*, 3 (2004), 393–398. 2, 3

[LAM01] Lext J., Akenine-Moeller T.: Towards rapid reconstruction for animated ray tracing. In *Eurographics 2001–Short Presentations* (2001), pp. 311–318. 2

[LFDL98] Lasenby J., Fitzgerald W. J., Doran C. J. L., Lasenby A. N.: New geometric methods for computer vision. *Int. J. Comp. Vision 36(3)* (1998), 191–213. 4

[MG04] Müller M., Gross M.: Interactive virtual materials. In *GI '04: Proc. of the 2004 conference on Graphics interface* (2004), Canadian Human-Computer Communications Society, pp. 239–246. 7

[MHTG05] Mueller M., Heidelberger B., Teschner M., Gross M.: Meshless deformations based on shape matching. In *Proc. of ACM Siggraph 2005* (2005). to appear. 6

[MKN*04] Müller M., Keiser R., Nealen A., Pauly M., Gross M., Alexa M.: Point based animation of elastic, plastic and melting objects. In *In Proc. of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), ACM Press, pp. 141–151. 1, 2, 3

[MSH*92] McNeill M. D. J., Shah B. C., Hebert M.-P., Lister P. F., Grimsdale R. L.: Performance of space subdivision techniques in ray tracing. In *Computer Graphics Forum* (1992), vol. 11, pp. 213–220. 2

[PKKG03] Pauly M., Keiser R., Kobbelt L. P., Gross M.: Shape modeling with point-sampled geometry. In *Proc. of ACM Siggraph* (2003), ACM Press, pp. 641–650. 1, 3

[PPG04] Pauly M., Pai D. K., Guibas L. J.: Quasi-rigid objects in contact. In *In Proc. of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), ACM Press, pp. 109–119. 1

[RL00] Rusinkiewicz S., Levoy M.: Qsplat: A multiresolution point rendering system for large meshes. In *Proc. of ACM Siggraph* (2000), pp. 343–352. 1, 3

[RSH00] Reinhard E., Smits B. E., Hansen C.: Dynamic acceleration structures for interactive ray tracing. In *Proc. of the Eurographics Workshop on Rendering Techniques* (2000), Springer-Verlag, pp. 299–306. 2

[Shi00] Shirley P.: *Realistic ray tracing*. A. K. Peters, Ltd., 2000. 2

[SJ00] Schaufler G., Jensen H. W.: Ray tracing point sampled geometry. In *Rendering Techniques 2000: 11th Eurographics Workshop on Rendering* (June 2000), pp. 319–328. 1, 2

[TW88] Terzopoulos D., Witkin A.: Physically-based models with rigid and deformable components. In *Proc. Graphics Interface* (June 1988), pp. 146–154. 4

[WBS03] Wald I., Benthin C., Slusallek P.: Distributed Interactive Ray Tracing of Dynamic Scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)* (2003). 2

[WPS*03] Wald I., Purcell T. J., Schmittler J., Benthin C., Slusallek P.: Realtime Ray Tracing and its use for Interactive Global Illumination. In *Eurographics State of the Art Reports* (2003). 2

[WS03] Wand M., Strasser W.: Multi-resolution point-sample raytracing. In *Graphics Interface 2003 Conference Proceedings* (2003). 1

[WTG04] Wicke M., Teschner M., Gross M.: Csg tree rendering of point-sampled objects. In *Proc. of Pacific Graphics 2004* (2004). 5

[ZPvG01] Zwicker M., Pfister H., van Baar J., Gross M.: Surface splatting. In *Proc. of ACM Siggraph 2001* (2001), pp. 371–378. 1

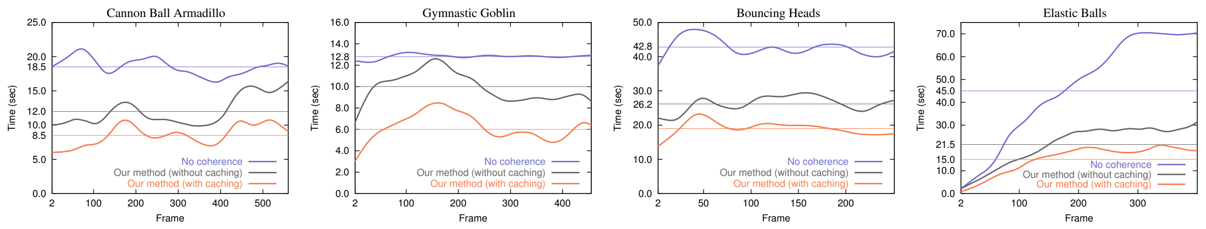

Figure 5: *Timing statistics for the different sequences. From left to right: cannon ball armadillo sequence, gymnastic goblin sequence, bouncing heads sequence and elastic balls sequence. The straight lines represent the average rendering time over all frames. The upper curve is the timing without considering any coherence. The middle curve is timed using our technique without sphere node caching enabled. The lower curve shows the average time per frame employing all acceleration techniques presented in this paper.*

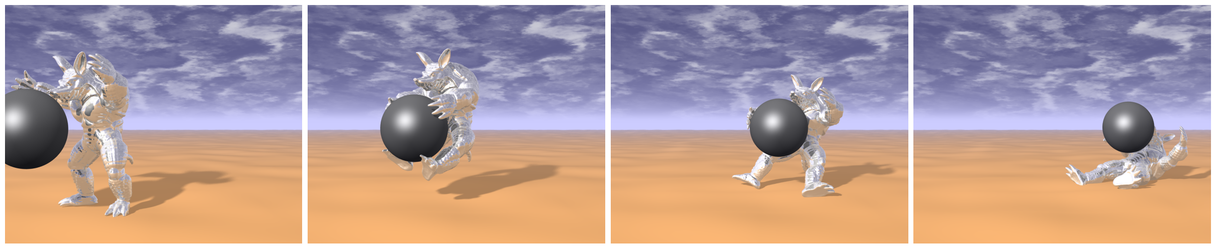

Figure 6: *Four frames of the cannon ball armadillo sequence.*

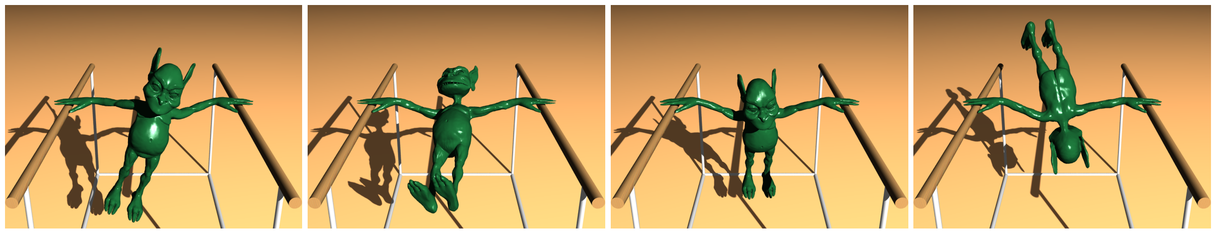

Figure 7: *Four frames of the gymnastic goblin sequence.*

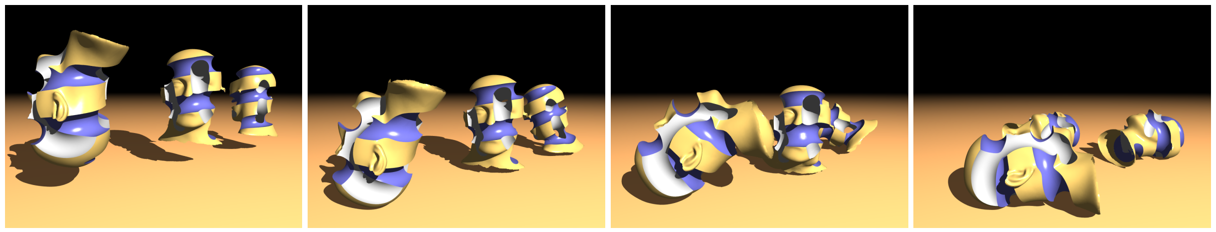

Figure 8: *Four frames of the bouncing heads sequence.*

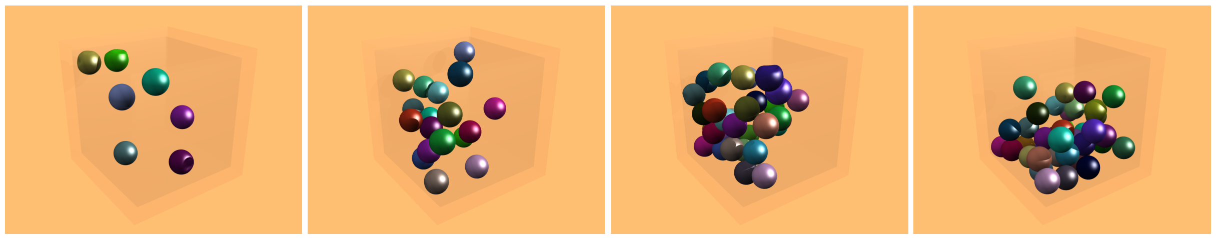

Figure 9: *Four frames of the elastic balls sequence.*