EUROGRAPHICS 2002 / G. Drettakis and H.-P. Seidel
(Guest Editors)

Volume 21 (*2002*), *Number 3*

Multi-Resolution Rendering of Complex Animated Scenes

M. Wand and W. Straßer

WSI/GRIS, University of Tübingen

Abstract

We present a novel multi-resolution point sample rendering algorithm for keyframe animations. The algorithm accepts triangle meshes of arbitrary topology as input which are animated by specifying different sets of vertices at keyframe positions. A multi-resolution representation consisting of prefiltered point samples and triangles is built to represent the animated mesh at different levels of detail. We introduce a novel sampling and stratification algorithm to efficiently generate suitable point sample sets for moving triangle meshes. Experimental results demonstrate that the new data structure can be used to render highly complex keyframe animations like crowd scenes in real-time.

Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture / Image Generation – Display Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques – Graphics data structures and data types; G.3 [Mathematics of Computing]: Probability and Statistics – Probabilistic algorithms.

1. Introduction

Today, computer generated images of highly complex scenes of animated objects are found in many movie productions. „The Lion King“[22] for example features a sequence in which a stampede of buffaloes runs through a valley. Many recent movie productions like „The Mummy Returns“[24] or the „Lord of the Rings“[23] make heavy use of computer generated crowd animations. For many applications of interactive computer graphics like computer games or interactive visualization[20] it would also be desirable to integrate renderings of large animated data sets. However, rendering of highly complex animated models has high demands for computational resources so that it is usually limited to offline applications.

A general technique to reduce rendering costs is multi-resolution modeling: A hierarchy of levels of detail is built from the scene description. During rendering, levels of detail are chosen for each part of the scene to match the display resolution of the projected image. Previous work has shown that highly complex scenes can be handled by multi-resolution rendering in real-time. However, up to now, only a very few multi-resolution algorithms are known that can handle animated input data[5, 18]. In this paper, we present a new multi-resolution rendering algorithm for animated scenes that obeys to the point-sample rendering paradigm[8, 12, 15]. Our algorithm takes keyframe animations of triangle meshes as input and builds a hierarchy of point samples and triangles to represent different resolutions of the scene. The major technical problem is to find a set of sample points on the moving surfaces that are distributed sufficiently uniformly on the surfaces of the objects at any time during the animation. We present a novel sampling algorithm that consists of a randomized sampling step and a stratification step to efficiently calculate point sample sets that minimize oversampling. Similar to the surfels rendering technique[12], prefiltering is applied to reduce the variance of the color attributes of the sample sets. As suggested in recent work[3, 25], triangles which are large in respect to the sampling resolution are rasterized as triangles instead of points to guarantee that the multi-resolution approach is never slower than conventional rasterization and that the memory consumption of the point-sample data is limited without sacrificing model resolution. The algorithm leads to a rendering time that is independent of the complexity of the geometric model.

We apply our multi-resolution data structure to the rendering of large crowds of animated characters. Two different techniques are used: Smaller scenes with up to a few thousand objects can be controlled by simulating the behavior of each object individually. Larger crowds can be handled with a hierarchical instantiation scheme. It provides

less flexibility for the motion of the objects but permits the rendering nearly arbitrarily complex scenes. Using a prototype implementation of our new technique, we are able to render crowd animations consisting of up to some hundred million triangles in real-time with high image quality.

2. Related Work

In this section, we will summarize related work from multi-resolution rendering, especially multi-resolution rendering of animations.

Mesh Simplification: Automatic mesh simplification algorithms have been successfully applied to speed up the rendering of static scenes, see e.g. Puppo and Scopigno[13] for a survey. There are only a few methods that can handle animated data sets. Friedrich et al.[5] propose an algorithm for the interpolation of keyframe hierarchies based on the Rivara bisection scheme. Shamir et al.[18] propose a general data structure for the application of mesh decimation algorithms to animated data sets. Mesh simplification algorithms can speed up rendering substantially, if applicable. However, there are many scenes which do not permit sufficient simplification. Especially scenes of complex topology, like crowd animations with a huge amount of independent objects, cannot be simplified beyond a certain limit without a severe loss of image quality. In such cases, point sample rendering techniques as used in this paper are more appropriate.

Image Based Rendering: Image based techniques substitute precomputed images for complex geometry. Several methods have been proposed for static data [7, 9, 17] but only a few can handle animated scenes: Tecchia and Chrysanthou[21] describe a rendering technique for complex crowd animations. In a preprocessing step, images from several viewing directions are rendered into textures for some timesteps of the character animation. During rendering, the closest time step and viewing direction is determined and the corresponding texture is rendered. The method leads to very fast rendering times. However, the discretization of time and viewing angle causes parallax and continuity errors. This does not matter for a far field approximation, but the method cannot be used to simplify single, large animated objects of high complexity. Aubel et al.[1] describe a dynamic image caching algorithm for crowd animations. The image of a rendered person is reused as texture over several frames. The speed-up of this method is limited as coarsening the time discretization too much will result in jerky motion.

Point Sample Rendering: Point sample rendering replaces complex geometry by a cloud of roughly pixel-sized points. The usage of points as rendering primitives was already suggested by Levoy and Whitted[10] in 1985 and rediscovered in the late nienties[8, 16]. Several authors suggested multi-resolution algorithms that adapt the density of the point cloud locally to the display resolution[2, 12, 15, 19, 25]. Cohen et al.[4] describe a technique that combines point-based rendering and mesh simplification. Stamminger and Drettakis[19] describe a simple extension of their point sample rendering technique to animated scenes: The points are moved dynamically by a motion function. However, the sampling density is not adapted to the motion of the object but it is fixed in advance*. Our algorithm improves on this by allowing arbitrary motion. Using a dynamic hierarchy and time dependent stratification, it provides guaranteed limits for the oversampling.

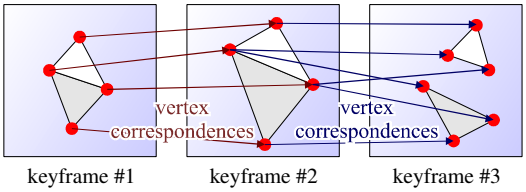


Figure 1: *Keyframe definition with vertex correspondences. The topology may change at keyframes.*

3. Multi-Resolution Hierarchy

In order to provide a general technique that is not dependent on a specific application, we need a general model to represent animated scenes. Our algorithm works on keyframe animations of triangle meshes. The keyframe animation consists of a sequence of triangle meshes of arbitrary topology and connectivity. For each pair of consecutive keyframes, *correspondences* between the vertices of the triangles must be specified, i.e. every vertex of a keyframe must be assigned a matching vertex in the other keyframe (Figure 1). During animation, the position (and all other vertex attributes, like normal or color) is interpolated linearly between the keyframe values. Triangles can be created or deleted by blending from one vertex position to three different positions and vice versa. The specification of vertex correspondences is part of the input to the algorithm, i.e. they are not established automatically but they must be specified by the user during modeling.

3.1. Data Structure

We will first describe our multi-resolution hierarchy for the static case. The extension to the animated case is discussed in section 3.3. Our algorithm is a generalization of the work of Pfister et al.[12] and Rusinkiewicz and Levoy[15], including ideas from Chen et al.[3] and Wand et al.[25].

* In their paper, they also describe a fully adaptive sampling technique for parametric surfaces. However, this method cannot be applied to general 3d-models.

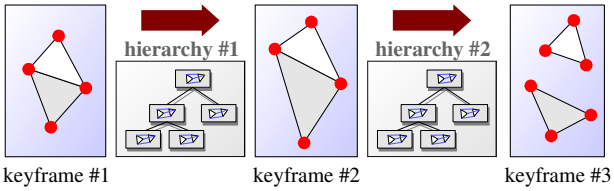


Figure 2: *Multi-resolution hierarchies are computed between consecutive keyframes*

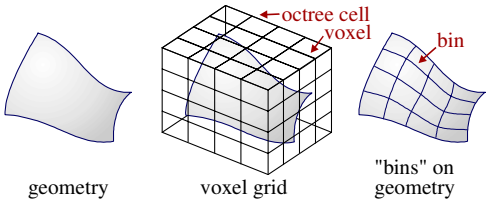


Figure 3: *Dividing Geometry into bins, each bin should receive at least one sample point*

Our data structure is an octree that partitions the scene spatially. Each node stores an approximate representation of its part of the scene with a fixed resolution in respect to the geometric size of the node. To build the octree, we start with a cube containing all triangles of the scene. Then the following algorithm is performed recursively, building the tree in top-down order: We choose sample points distributed uniformly on the surface area of the triangles and store them for the current box. The maximum distance between these sample points is chosen to be a constant fraction of the side length of the box. This means that we would obtain a fixed sampling density if we scaled the boxes to uniform size. The sampling strategy is detailed further in section 3.4. After sampling, we count the number of sample points that a triangle receives. Triangles receiving more than a few (say 1-3) sample points are also stored in the current box. These triangles are not considered any longer for point sampling in child boxes because at that sampling density, the point sample approximation becomes more expensive than conventional rasterization of the triangles. The current cube is then subdivided recursively into 8 smaller cubes and the remaining triangles are distributed among the child nodes. The recursion is performed until a node contains only a constant amount of triangles (e.g. at most 50 triangles) which are then stored in the box without sampling.

Distributing triangles among child boxes can lead to problems if the triangles are relatively large in comparison to the extents of the box. We adopted the same solution as Wand et al.[25], which works well in practice: A triangle is moved into that cube which contains its center. We allow triangles to exceed the boundary of the child-cube by at most a constant factor (say 10% of the side length of the cube). If the triangle is larger than that, we store it in the last inner node for which it does not exceed the tolerance zone.

3.2. Rendering

After this procedure, we obtain a hierarchy of point samples and triangles. Large triangles are stored near the root of the hierarchy and smaller ones towards the leafs. The sampling density increases if we traverse the hierarchy downwards: The maximum distance between the sample points is reduced to the half if we step from one node to its child node. For each node, the point samples represent all small triangles that have not been found yet on the path down from the root node.

To render a scene from this representation, we apply projective classification and view-frustum culling[2, 12, 15, 25]: We traverse the hierarchy downwards from the root recursively: If the current node is outside the view-frustum, it is excluded from consideration. Otherwise, the maximum distance between sample points in the current node is evaluated and projected on the screen: The projection is done by dividing the sample distance by the minimum depth of the bounding box (the minimum z-coordinate of a corner of the bounding box in camera coordinates) and scaling this value by the size of the viewport. If it is smaller than a user defined *splat size*, all point samples are drawn on the screen as splats of uniform color and depth. If the projected sample distance is still larger, the node is subdivided recursively. All triangles that are found during hierarchy traversal are drawn on the screen using conventional z-buffer rasterization. Previous work has shown that this rendering strategy is very efficient[12, 15, 3] even for highly complex scenes of billions of primitives[25].

3.3. Animation

Up to now, we neglected the fact that the objects of the scene are moving (with motion defined by a sequence of keyframes). In order to deal with animated scenes, we use one separate multi-resolution hierarchy between each pair of consecutive keyframes (see Figure 2). This reduces the problem to defining a multi-resolution hierarchy for triangles with vertices that are moving on linear paths with constant speed (due to the linear keyframe interpolation). We will now consider one pair of consecutive keyframes and describe the extensions to our multi-resolution hierarchy: We apply the construction algorithm described in Section 3.1 to the first of the two keyframes. In the resulting hierarchy, we store the bounding box of the triangles for both the start and the end keyframe. To calculate the sampling density, we calculate the maximum side length of the start and the end bounding box and set the maximum sample distance to a fixed fraction of this maximum side length (typically 1/8 - 1/16).

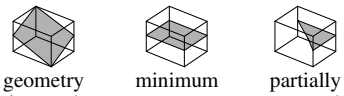


Figure 4: *left: part of a closed surface in a voxel, middle: minimum area created by a closed surface, right: voxel is only partially covered by the surface*

During rendering, we will need a hierarchy for a position in time between the two keyframes. This is calculated dynamically: The bounding box of a node is obtained by linear interpolation between the start and the end bounding box. It is easy to see that the interpolated bounding box is always a correct bounding volume for any position in time between the two keyframes due to the linear motion of the vertices. Thus, we use interpolated hierarchies during hierarchy traversal and interpolated vertex positions and vertex attributes for the point samples and triangle vertices for rendering the primitives. Using sampling distances proportional to the maximum side length ensures that the sampling density will grow monotonically when descending in the hierarchy, as assumed by the rendering algorithm.

Of course, interpolated hierarchies rely on temporal coherence: If all vertices move to random positions between two keyframes, the hierarchy will be destroyed if we move on to time steps beyond the start keyframe, towards the end keyframe. However, such a situation is rarely found in applications. In practice, the worst case is that groups of triangles move in opposite directions, for example parts of the two legs of a walking human. In this case, the bounding volume grows if we move forward in time. If it becomes too large, the sampling density will become too small and the box will be subdivided during rendering. The spatial subdivision will then (usually in one or at most two steps) divide the moving parts from each other. Actually, we did not observe any problems due to hierarchy distortion in our experiments.

3.4. Choosing Sample Points

Up to now, we described the basic rendering algorithm. The problem that remains is how to calculate the sample points for each octree box. We must cover moving surfaces with point sets with roughly uniform distances between the points: On the one hand, we must not exceed a maximum sample point distance on the surfaces of the objects at anytime between the two keyframes. On the other hand, we should use as few points as possible in order not to slow down rendering. We solve this problem by a two step approach: First, we choose a large set of sample points that safely covers all moving surfaces. The points are fixed on the surface and moved when the geometry is animated. Then we use a stratification algorithm to choose a subset of this candidate set for discrete time steps that has only a small oversampling.

3.4.1. Candidate Sets

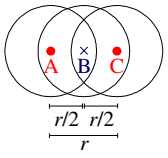
The problem in choosing suitable sample sets is that the structure of the triangle motion is not known. A general technique to deal with problems of complex structure is randomization. Here, one relies on statistical properties to guarantee independence of the concrete structure of the problem. Randomization was already applied successfully to the generation of surface sample point sets[25] for static scenes and dynamic scenes with small deformations[19]. We generalize the technique to arbitrary keyframe animations:

Again, we assume first that our scene is static. We choose random surface points, uniformly distributed on the surface of the triangle mesh: We choose a random triangle with probability proportional to its area and choose a point from the triangle as a random linear combination of its vertices with uniform probability. It is obvious that we are able to cover the surface of the scene with arbitrary density by repeating this process many times. But how many sample points are sufficient in order to assure a given sample density? To analyze this question, we consider a three dimensional grid of cubes in space that contains the geometry (Figure 3). The grid divides the geometry into bins. We want that every voxel in the grid that is fully covered by a piece of surface will receive at least one sample point (Figure 4, left). If we set the maximum sample distance to two times the diagonal extents of a voxel and chose an appropriate projected splat size this will guarantee that surfaces are reconstructed on the screen without holes.

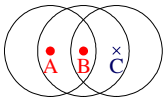
To guarantee that every bin receives at least one sample point, we first determine the minimum area of a piece of surface that fully covers the voxel: This is the side length of the voxel squared (Figure 4, middle). To obtain an upper bound for the number of bins, we divide the total area of the triangle mesh by the minimum area of a bin. Now we know that we have (at most) n bins. According to the so-called "coupon-collectors-theorem"[11] the expected number of random points we must draw from the surface is $n \cdot \ln n$. This result is asymptotically sharp[11]. Therefore, we can guarantee that every bin receives a point with very high probability by enlarging the sample size by a small constant factor (say 3 or 5). These arguments are similar to those in Wand et al.[25]. However, they use randomized sampling to generate sample points dynamically during rendering while we apply it already in the preprocessing stage.

In order to deal with animated scenes, we calculate for each triangle the maximum area that it will have between the two keyframes* and use this value as area value during

* As easy to see, the area of a triangle with linearly interpolated vertices is a 4^{th} degree polynomial in time. Thus, the time of maxi-



(a) Point B can be deleted.



(b) After point C has been deleted, point B must not be deleted anymore.

Figure 5: *Stratification, avoiding cascaded deletion.*

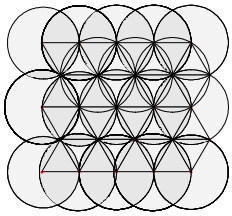


Figure 6: *The worst-case sample distribution is a tightest packing of circles.*

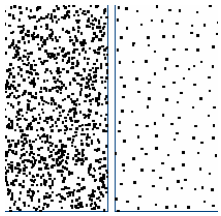


Figure 7: *comparison of sampling patterns. Left: random, Right: stratified.*

randomized sampling. The key observation is now that our sampling process does not assume anything about the concrete structure of our scene. The bins are only abstract amounts of surface area, not fixed regions in space. Thus, if we assume maximum area values for all triangles, our coverage arguments apply for any time step between the two keyframes. Due to the randomization, our sample set is universal, suitable for arbitrary deformations of the underlying geometry. This is the main argument for the randomized technique. We could of coarse use a deterministic sampling pattern to sample the triangles. However, the structure of the motion of the triangle mesh is difficult to analyze by the sampling algorithm. Therefore, we could obtain a nearly random sample set even if the sample sets for each triangle or even for a whole keyframe are chosen deterministically. The randomized construction allows us to ignore the structure of the input data. However, this has some costs: The oversampling due to the randomization is $\ln n$. In practice, this is a value between 5 and 10. Additionally, the triangles may change their area value between the keyframes. It is easy to see that the triangle area may be a 4^{th} degree polynomial over time. So we obtain a worst-case overestimation of area of:

$$\int_0^1 adt \Big/ \int_0^1 at^4 dt = 5 \quad \text{(for some constant a)}$$

This means that the worst-case oversampling of the purely randomized sampling technique is in the range of 25-50. This value is too large for rendering in real-time, therefore we need an additional stratification step.

3.4.2. Stratification

In static scenes, a fixed voxel grid in space is often used for stratification of surface sample points[12, 25]: Space is divided into a regular grid of voxels and in every voxel one sample point is chosen. Usually, the point is also quantized to lay in the middle of the voxel. To analyze the oversampling, we assume that a flat, 2-dimensional surface with a high density candidate set is the input to the stratification step. The ratio between the minimum and the maximum sample point distance is the ratio between the diagonal and the side length of a voxel and this is $\sqrt{3}$. The maximum oversampling for a planar surface is this ratio squared. Thus, we obtain a worst-case oversampling of 3, which is better than that of the randomized technique. However, this technique cannot be generalized easily to animated scenes because the objects can move arbitrarily while the grid is static. We would need new sample sets every few time steps.

To avoid these problems, we use a criterion without a fixed spatial grid. Let us once again first assume that the scene is not animated. The idea of the criterion is fairly simple: We can remove sample points from the candidate sets as long as all points are still covered by splats from the remaining points in screen space. Let r be the maximum distance between sample points allowed and assume that we have given a high density candidate set with maximum sample distance no larger than r. Then we can remove points as long as all points still have a neighboring point within a radius of $r/2$ that has not been deleted yet (Figure 5a). Our stratification algorithm is a simple greedy algorithm: It takes all points in random order and tries to delete the point. A point can be removed if it is already covered by a sphere of radius $r/2$ around another point *and* if it is not the only point to cover one of the previously deleted points within a radius of $r/2$. The second part of this condition is important: If we just delete any point with at least one neighbor within $r/2$ this can lead to cascaded deletion, deleting all sample points until only one is left (see Figure 5b). The criterion is similar to Poisson-disk sampling[6]. However, for the generation of surface sample points, we must guarantee a maximum sample point distance rather than a minimum distance.

How efficient is the suggested stratification technique? This obviously depends on the candidate set and on the order, in which the deletion operations are performed. We perform the deletion in random order on a random candidate set. Thus, we can only analyze the worst case. In the analysis, we assume again that a flat, 2-dimensional surface is the input to the sampling algorithm because this is a good model for a typical case when surface models are simplified*. The worst case, i.e. the largest sample set from which no more samples can be deleted, is a tightest packing of spheres of radius $r/2 - \varepsilon$ (for arbitrarily small ε) in the plane

mum area can be calculated by solving the derivative which is a 3^{rd} degree polynomial.

* Of course, the algorithm will work in any case, i.e. on arbitrary volumes of sample points.

(Figure 6). The largest distance between two points may be as large as r. Therefore, we must use splats with a radius of at least $r/2$. Thus, the worst-case sampling density is $((r/2)^2 \cdot \sin 60°)^{-1}$ while the splatting renderer must assume a density of r^{-2}. Therefore, the worst-case oversampling is $4 \cdot \sin^{-1} 60° \approx 4.6$. However, this is only a worst-case value. If we apply our implementation of the algorithm to a flat, thin plate we obtain an empirical average value of 2.6. The deviation from this value is very small (we measured a standard deviation of 3%).

Now we can apply this stratification technique to animated scenes with little modification: We divide the time between two keyframes in discrete time intervals (say 3 or 4). In each time interval, we determine which points are fully covered by spheres of radius $r/2$ around the other points. Then we apply the stratification algorithm to this coverage data and store different point sets for each time interval. The coverage information can be calculated efficiently by using a spatial data structure. We use a regular grid with the sphere radius as cell spacing. A separate data structure at the start and the end of each time interval is used to determine which points are overlapping at the beginning and the end of the time interval. If two points cover each other at both the start and the end time, they must cover each other throughout the whole interval because the distance between two linearly moving points can have only one local minimum and no local maximum.

This stratification technique is slower than simple grid-quantization. We need about 20-25 seconds to build a hierarchy of stratified sample points with one time interval between two keyframes of a 30.000 triangles model of a walking character. The complete animation of 9 keyframes takes about 3-4 minutes.

3.5. Attribute Filtering

The stratification algorithm calculates some representative points R which safely cover the underlying surface. Now we must assign surface attributes to the points. We first calculate a larger set of sample points F which cover the same triangles that are represented by the stratified points. Typically, the sampling density of F is ten times higher than that of the candidate set for R. For performance reasons, F is not stratified. Then, we place a three dimensional radial Gaussian function around each representative point of R and calculate the weighted average of the attributes found in the neighboring points of F. As the support of the Gaussian filter kernel is relatively small, we can use again a spatial grid to accelerate the neighborhood query. We calculate filtered attributes for the start and the end time of each point sample and blend linearly in time to avoid popping artifacts.

3.6. Hierarchical Instantiation

A typical application area of our algorithm is the rendering of large animated crowds of characters. For a limited number of characters in the crowd (say no more than some thousands), it is possible to place each object individually in space, maybe according to some behavioral rules. If we want to animate larger crowds, we can apply hierarchical instantiation: We construct multi-resolution hierarchies out of a set of multi-resolution sub-hierarchies. The sub-hierarchies may represent different animated models of single objects and the super-hierarchy represents a whole crowd of objects. Usually, both the motion of the sub-hierarchies and the super-hierarchy are periodical, but the super hierarchy may have a larger periodicity (i.e. a larger number of keyframes).

A generalization of our data structure to hierarchical instantiation is straightforward: We allow not only triangles to be inserted in the data structure but also sample points. Then we can perform the same sampling, stratification and filtering operations as before just by mixing “new” point samples from triangles and “old” point samples from instances. To assure a correct sampling and filtering, each sample point is assigned a weight proportional to the area it represents after sampling. The weight is calculated by dividing the area of the original model (triangles or instance) by the amount of sample points (random points or stratified points from the instance).

4. Implementation

We implemented the algorithm in C++ using OpenGL for rendering points and triangles. The interpolation of triangle vertex attributes and point sample vertex attributes is performed using the `NV_VERTEX_PROGRAM` extension: For each vertex the start and end position and the corresponding color attributes are specified. The vertex program interpolates the vertices linearly according to the time between two keyframes and then performs a perspective transformation. All vertex attributes are stored in vertex arrays. All benchmarks were performed on a 2 Ghz Pentium 4 with 1 GB of ram and a GeForce3 graphics board.

5. Results

5.1. Parameters

There are some parameters in our algorithm which should be determined experimentally.

Sampling density per box: An important parameter is the maximum spacing of sample points used in the sampling and stratification step. We write this parameter as fraction of the side length of the octree box for which the sampling is done. If we choose a large sample distance, only a few

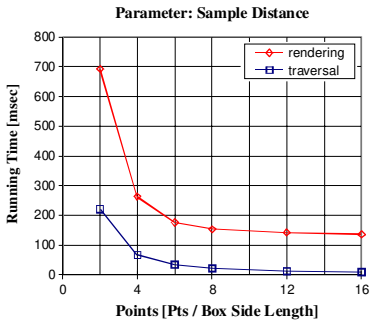


Figure 8: *Rendering time for different sampling densities per octree box*

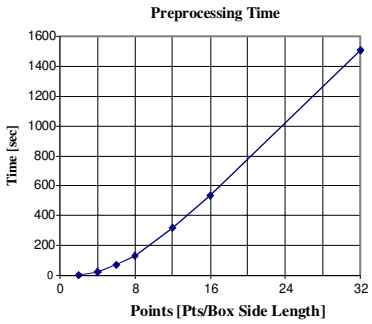


Figure 9: *Preprocessing time for different sampling densities*

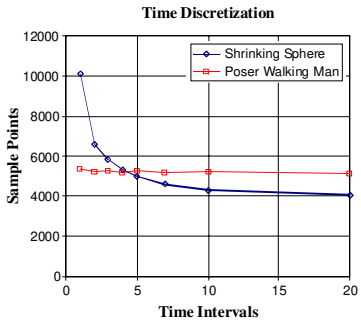


Figure 10: *Time discretization during stratification for different models*

points are found in every box and we have a lot of traversal efforts in the rendering algorithm. If we use larger blocks we have less traversal costs but the approximation of the ideal sampling density is less exact. Additionally, the preprocessing costs are higher because more sample points have to be processed per octree node. Figure 8 and Figure 9 show the experimental results for a typical test scene (Figure 11d). The overhead for small boxes is rather high while the approximation error introduced by larger boxes is not a severe problem. We found a sample distance of 1/8 - 1/16 of the maximum side length of the octree box to be a good compromise in practice. Using larger sampling densities does not lead to a significant reduction of the rendering cost but to unnecessarily high preprocessing costs.

Time discretization: In the stratification step we have the option to do the stratification multiple times for several time intervals between each pair of keyframes. This should reduce the rendering costs for models with strongly varying triangle areas. We varied this parameter for two different models. The first scene is an artificial scene of a sphere shrinking to 1/100 of its original area. The second is an animated poser character. For such a scene we anticipate only minimal improvements as the triangle area shows only minimal variation between two keyframes while the first scene should profit from multiple stratification steps in time.

The experimental results agree with this (Figure 10): For the "shrinking-sphere" scene, The number of sample points can be reduced to about 40% by using a fine granular discretization. A good choice for the number of intervals is a value between 3 and 5 as this already provides most of the reduction of the rendering time. For animated characters, time discretization is not necessary. The reduction of rendering time is below 5% in the best case.

5.2. Applications

We applied our algorithm to two different example scenes, representing different application scenarios. Table 1 summarizes the results. Table 2 shows the number of primitives used for rendering. Snapshots from the animation are shown in Figure 11 and in the color section.

Simulation: The scene in Figure 11b shows a herd of horses running through a landscape. We implemented a simple simulator to control 2500 instances of an animated horse model. Our behavioral model is a simplified version of the boids model[14]. The objects are controlled by local force fields around each object and each obstacle. A spatial grid data structure is used to efficiently retrieve the local neighbors of each object to integrate the forces. The scene has a total complexity of about 50 million triangles. We were able to render the scene with up to 5 frames per second, depending on the splat size. The simulator took about 10% of the rendering time of the fastest rendering settings.

Hierarchical instantiation: If we want to create scenes with a larger number of objects, the efforts for placing the object instances become dominant. To allow larger crowds of objects, we can use hierarchical instantiation. We applied this technique to an artificial scene that consists of a replication of four different animated characters (Figure 11a) on a

scene	complexity [triangles]	# keyframes	preprocessing time [sec]	data structure size [MB]	rendering time for splat size [sec] 1×1	2×2	3×3
4 poser characters	25584	9	573	97	0.018	0.016	0.013
simulation (horses)	58 million	15	251	208	0.738	0.319	0.202
football stadium	105 million	15	1315	302	1.036	0.373	0.188
grid replication	576 million	9	708	123	0.861	0.294	0.135

Table 1: *Running times for application examples, see also Figure 11 and color section*

scene	1×1	2×2	3×3
horses simulation	1608 / 715	645 / 195	316 / 95
football stadium	2667 / 860	1030 / 298	527 / 137
grid replication	2552 / 398	850 / 84	379 / 24

Table 2: *Simplification of the example scenes (see Figure 11): points / triangles used for rendering [in thousands] for different splat sizes*

300×300 grid resulting in a scene complexity of 576 million triangles (Figure 11d). A 640×480 rendering of the scene took between 0.14 and 0.86 seconds, depending on the splat size (see Table 1). The hierarchical instantiation technique can also be applied to real world scenes: Figure 11c shows a football stadium with 16416 football fans. The scene consists of 105 million triangles and can be rendered with up to 5 frames per second.

6. Conclusions and Future Work

We presented a new multi-resolution rendering technique for animated scenes based on point-sample rendering. The algorithm works on linearly interpolated triangle meshes of arbitrary topology. The algorithm uses a precomputed hierarchy of triangles and sample points. Thus, the rendering time is independent of the complexity of the input meshes. Experiments show that the algorithm can be used to animate large crowds of individually moving objects. Hierarchical instantiation can be applied to animate even larger scenes of some hundred million primitives. To our knowledge, this is the first multi-resolution rendering algorithm in this generality providing real-time performance.

There are several directions for future work. First, we would like to implement a better anti-aliasing technique that can also handle partially transparent point samples. A very interesting approach is surface splatting[26] although it currently still requires a software rendering pipeline. A second, important topic would be the integration of an occlusion culling algorithm to reduce the dependency of the rendering time on the projected area. We are also interested in a generalization of the animation model. It should be easy to generalize the algorithm to higher order interpolation schemes. The convex hull property of many spline techniques could be used to calculate an interpolated hierarchy and vertex shaders could also be used to interpolate point samples on higher order curves efficiently.

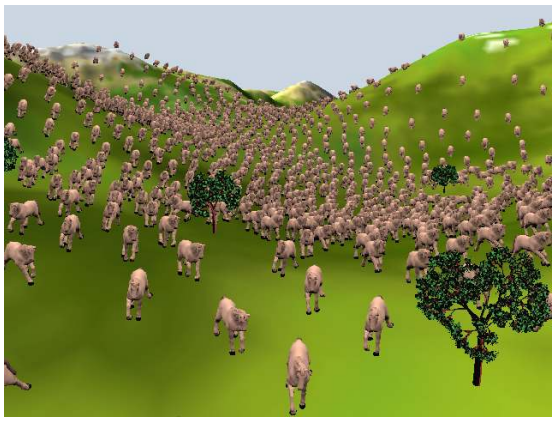
Acknowledgements

The authors wish to thank Stefan Gumhold and Matthias Fischer for valuable discussion, Amalinda Oertel and Martin Frisch for modeling the test scenes, Matthias Mueller for implementing the Poser import filter, and Tobias Hüttner/EgiSys for providing the Poser Animation Software.

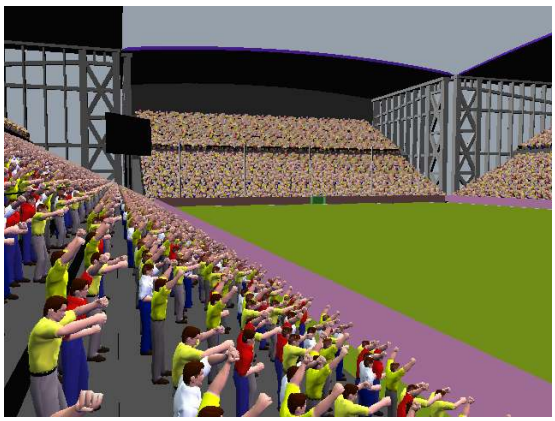
References

1. Aubel, A., Boulic, R., Thalmann, D.: Real-time Display of Virtual Humans: Levels of Detail and Impostors. In: *IEEE Transactions on Circuits and Systems for Video Technology*, 2000.
2. Chamberlain, B., DeRose, T., Lischinski, D., Salesin, D., Snyder, J.: Fast Rendering of Complex Environments Using a Spatial Hierarchy. In: *Proc. Graphics Interface '96*, 132-141, 1996.
3. Chen, B., Nguyen, M.X.: POP: A Hybrid Point and Polygon Rendering System for Large Data. In: *Proceedings of IEEE Visualization '2001*, 2001.
4. Cohen, J.D., Aliaga, D.G., Zhang, W.: Hybrid Simplification: Combining Multi-resolution Polygon and Point Rendering. In: *Proceedings of IEEE Visualization '2001*, 2001.
5. Friedrich, A., Polthier, K., Schmies, M.: Interpolation of Triangle Hierarchies. In: *Proceedings of IEEE Visualization '98*, 1998
6. Glassner, A.S.: *Principles of Digital Image Synthesis*. Morgen Kaufmann, 1995.
7. Gortler, S. J., Grzeszczuk, R., Szeliski, R., Cohen, M. F.: The Lumigraph. In: *SIGGRAPH 96 Proceedings, Annual Conference Series*, 43-54, 1996.
8. Grossman, J. P., Dally, W.: Point Sample Rendering. In: *Rendering Techniques'98*, 181–192, Springer, 1998.
9. Levoy, M., Hanrahan, P.: Light Field Rendering. In: *SIGGRAPH 96 Proceedings, Annual Conference Series*, 31-42, 1996.
10. Levoy, M., Whitted, T.: *The Use of Points as a Display Primitive.* Technical report, University of Norh Carolina at Chapel Hill, 1985.
11. Motwani, R., Raghavan, P.: *Randomized Algorithms*. Cambridge University Press, 1995.
12. Pfister, H., Zwicker, M., van Baar, J., Gross, M.: Surfels: Surface Elements as Rendering Primitives. In: *SIGGRAPH 2000 Proceedings, Annual Conference Series*, 335-342, 2000.
13. Puppo, E., Scopigno, R.: Simplification, LOD and Multiresolution Principals and Applications. In: *EUROGRAPHICS 97 Tutorial Notes*, 1997.
14. Reynolds, C. W.: Flocks, Herds, and Schools: A Distributed Behavioral Model, In: *SIGGRAPH 87 Proceedings, Annual Conference Series*, 25-34, 1987.
15. Rusinkiewicz, S., Levoy, M.: Qsplat: A Multiresolution Point Rendering System for Large Meshes. In: *SIGGRAPH 2000 Proceedings, Annual Conference Series*, 343-352, 2000.
16. Shade, J., Gortler, S., He, L., Szeliski, R.: Layered Depth Images. In: *SIGGRAPH 98 Proceedings, Annual Conference Series*, 231-242, 1998.

(a) 4 animated poser characters, 25 584 triangles, rendering time 18 msec (splat size 1×1)



(b) 2 500 horses ("boids" simulation), 58 million triangles, rendering time 319 msec (splat size 2×2)

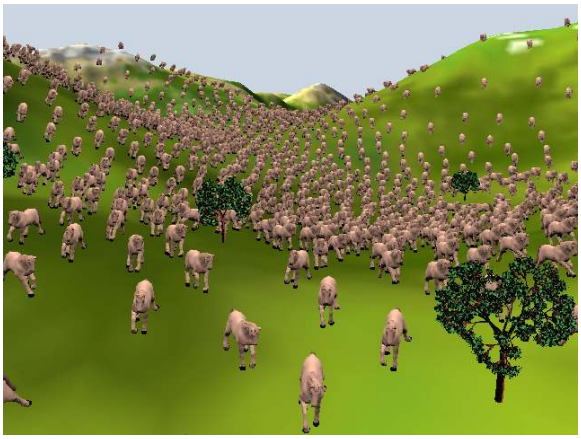


(c) football stadium, 16 416 objects, 105 million triangles, rendering time 373 msec (splat size 2×2)

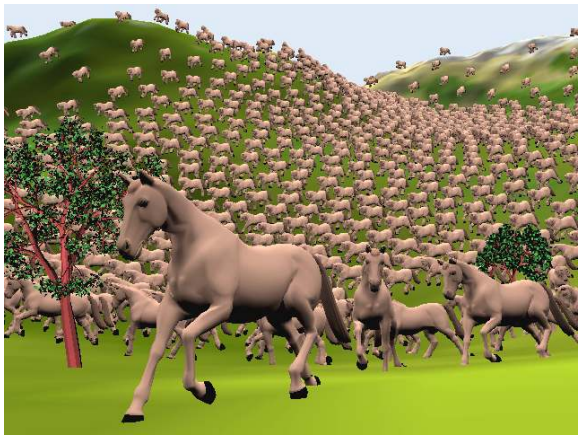
(d) replicated poser models, 90 000 objects, 575 million triangles, rendering time 294 msec (splat size 2×2)

Figure 11: *Application examples. Resolution 640×480 pixel.*

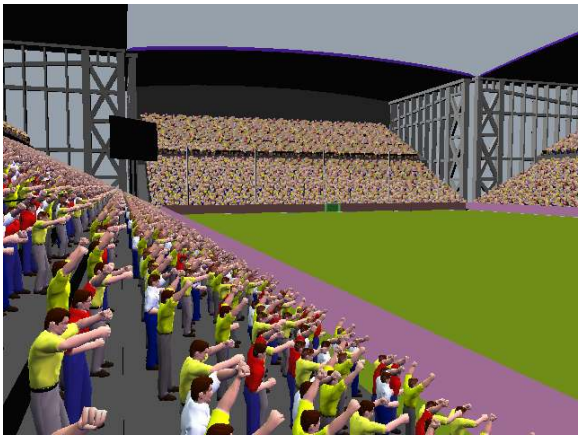
17. Shade, J., Lischinski, D., Salesin, D. H., DeRose, T., Snyder, J.: Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In: *SIGGRAPH 96 Proceedings, Annual Conference Series*, 75-82, 1996.
18. Shamir, A., Valerio, P., Chandrajit, B.: Multi-Resolution Dynamic Meshes with Arbitrary Deformations. In: *Proceedings of IEEE Visualization 2000*.
19. Stamminger, M., Drettakis, G.: Interactive Sampling and Rendering for Complex and Procedural Geometry. In: *Rendering Techniques 2001*.
20. Stoev, S.L., Straßer, W.: A Case Study on Interactive Exploration and Guidance Aids for Visualizing Historical Data. In: *Proceedings of IEEE Visualization 2001*.
21. Tecchia, F., Chrysanthou, Y.: Real-Time Rendering of Densely Populated Urban Environments. In: *Rendering Techniques 2001*.
22. *The Lion King*, Disney Pictures, 1994.
23. *The Lord of the Rings*, New Line Productions, 2001.
24. *The Mummy Returns*, Universal Pictures, 2001.
25. Wand, M., Fischer, M. Peter, I., Meyer auf der Heide, F., Straßer, W.: The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes. In: *SIGGRAPH 2001 Proceedings, Annual Conference Series*, 361-370, 2001.
26. Zwicker, M., Pfister, H., van Baar, J., Gross, M.: Surface Splatting. In: *SIGGRAPH 2001 Proceedings, Annual Conference Series*, 371-378, 2001.



2 500 horses (“boids” simulation), 58 million triangles, rendering time 319 msec (splat size 2×2)



simulation scene, close-up, rendering time 410 msec (splat size 2×2)



football stadium, 16 416 objects, 105 million triangles, rendering time 373 msec (splat size 2×2)

football stadium, close-up, rendering time 579 msec (splat size 2×2)

replicated poser models, 90 000 objects, 575 million triangles, rendering time 294 msec (splat size 2×2)

replicated poser models, close-up, rendering time 308 msec (splat size 2×2)

Figure: *Wand/Straßer: Multi-Resolution Rendering of Complex Animated Scenes. The left column shows snapshots from the animations discussed in the paper (see Table 1 and Table 2 for details). The right column shows additional close-ups.*

Some Corrections:

The original paper has been to pessimistic concerning the efficiency of the novel stratification technique. Due to a bug in the implementation, the sampling distance of the neighborhood-based point removal stratification strategy has been too small by a factor of sqrt(3), leading to a higher measured average oversampling. Additionally, the worst case for the (standard) grid stratification technique is not a plane parallel to the coordinate axes; a skew plane can intersect more grid cells, leading to a much larger oversampling. Therefore, both the average and the worst-case oversampling of the proposed novel stratification technique are indeed *better* than the traditional grid based technique, not slightly worst, as stated in the paper.

A second minor bug is in the area estimated for moving triangles: The area of a triangle with vertices moving on linear paths over time is not a 4th degree polynomial but the *squared area* is a forth degree polynomial (page 4). This does not affect the determination of the maximum area (to determine the maximum, we search for the maximum in the squared area, which is equivalent). However, for the case of a triangle growing to maximum size, we do not obtain $O(t^4)$ growth but only $O(t^2)$, leading to an oversampling factor of 3 not 5 (page 5). Again, the original estimate has been too pessimistic.

The results of the paper still hold; only the theoretical analysis for comparison with traditional stratification techniques has been to pessimistic.

Additional Remark:

Meanwhile, we have implemented an optimized rendering backend using DirectX 9. The football stadium scene animations shown in the paper can be rendered at framerates of 10-20Hz with *1 pixel* reconstruction resolution on a 1.5Ghz Pentium-M with GeForce FX Go 5650 graphics. This is considerably faster than the original implementation.

Details:

Details on these topics can be found in my PhD-thesis, which is available at http://www.gris.uni-tuebingen.de/areas/pbr/phd/index.html.