ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/2905027

Temporal and Spatial Level of Details for Dynamic Meshes

Article · November 2003

DOI: 10.1145/505008.505023 · Source: CiteSeer

CITATIONS
59

READS
31

2 authors:

Ariel Shamir
Interdisciplinary Center Herzliya
107 PUBLICATIONS **6,769** CITATIONS

SEE PROFILE

Valerio Pascucci
University of Utah
350 PUBLICATIONS **7,556** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Connectomics View project

All content following this page was uploaded by Valerio Pascucci on 11 December 2014.

The user has requested enhancement of the downloaded file.

Temporal and spatial level of details for dynamic meshes

Ariel Shamir
The Interdisciplinary Center
Herzelia
arik@idc.ac.il

Valerio Pascucci
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
pascucci@llnl.gov

ABSTRACT

Multi-resolution techniques enhance the ability of graphics and visual systems to overcome limitations in time, space and transmission costs. Numerous techniques have been presented which concentrate on creating level of detail models for static meshes. Time-dependent deformable meshes impose even greater difficulties on such systems. In this paper we describe a solution for using level of details for time dependent meshes. Our solution allows for both temporal and spatial level of details to be combined in an efficient manner. By separating low and high frequency temporal information, we gain the ability to create very fast coarse updates in the temporal dimension, which can be adaptively refined for greater details.

1. INTRODUCTION

Complex time-dependent meshes are becoming more frequent as part of animation sequences and virtual reality worlds. These types of meshes are often viewed as a time sequence of static meshes and as such, impose greater demands on the display systems in terms of rendering time and storage space. Multi-resolution techniques of static meshes have been widely used and studied as a means for overcoming time, storage and transmission restrictions. Many decimation and refinement techniques were developed and many spatial metrics defined for governing the quality and level of detail of multi- resolution static models [7, 15, 13, 10].

In this paper we introduce a multi resolution model for dynamic geometry sequence of meshes, which enables the combination of both *spatial* and *temporal* level of details to be employed. The key observation is that mesh modifications over time can be separated to "low frequency" global affine transformations and "high frequency" local vertex deformations (Figure 1). The low frequency information captures the most visually significant temporal displacements using a very coarse and inexpensive approximation. This approximation can in turn be refined adaptively using the high frequency information as needed to create higher resolutions and greater details in both time and space dimensions. Our model combines three different adaptation strategies: applying the low frequency temporal deformations over time, applying the high frequency temporal deformations, and applying spatial adaptation of level of details. This gives a visualization system the flexibility to comply with a wide range of timing restrictions.

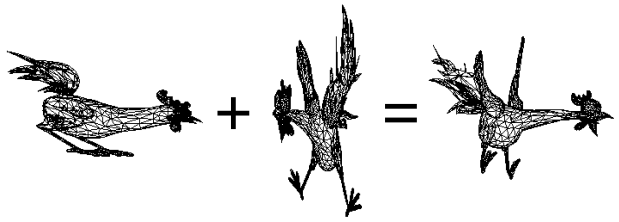

Figure 1: Separating low frequency (left) and high frequency (middle) deformations. This 'equation' is simply an illustration of the concept. The real calculation involves the multiplication of the vertices of the high frequency mesh (middle) by an affine matrix (which effect is symbolized by the left mesh).

The basis of our model is the TDAG structure (see [17] and Section 2) defined for supporting multi-resolution time dependent meshes. However, instead of encoding the original series of time dependent meshes in the TDAG, we first extract the low frequency deformations of the meshes, and encode only the resulting residual meshes. The TDAG stores all attributes and positional changes of the residual meshes over time (the high frequency temporal deformations), and enables the extraction of different and adaptive resolution meshes for each time-step (spatial adaptation).

1.1 Previous Work

It is far beyond the scope of this paper to describe the many approaches that have been developed for creating multi-resolution representations of static geometric data for graphics and visualization [7, 15, 13, 10]. In this paper we use edge contraction [8, 9] as the decimation primitive and the Quadric error metrics [5]. Other multi resolution schemes include vertex removal [1], triangle contraction [6], vertex clustering [16], and wavelet analysis [19, 2]. Similar to [4, 13, 1, 6] we organize the levels of detail structure as a DAG (Directed Acyclic Graph). Each node in the DAG represents a decimation operation and the edges represent dependencies between the different operations, which impose a certain partial order for applying them on the mesh. Every **cut** (informally, a cut is a group of edges that include one and only one edge from each path from the roots to the leaves) in this DAG defines a valid adaptive level of detail of the underlying mesh. To date, most of these schemes are based on the assumption that the fine resolution mesh is static. Our scheme aims to define a level of detail in both time and space for multi-resolution dynamic meshes.

Time dependent data structures which include hierarchical de-

Copyright 2001 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by a contractor or affiliate of the U.S. Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
VRST'01, November 15-17, 2001, Banff, Alberta, Canada.
Copyright 2001 ACM 1-58113-427-4/01/0011 ...$5.00.

compositions are presented in [20] for the extraction of iso-surfaces from dynamic volumetric data, and in [18] for volume rendering. These structures allows very efficient time-dependent iso-surface extraction and volume rendering respectively, but are tailored for these specific types of visualization primitives and do not deal with general dynamic deformable meshes.

Earlier in [3], a model for multi-resolution video was presented. Temporal as well as spatial level of details were possible by using a binary time tree where each node corresponds to some spatial averaging of all the images of its time span. The node holds a spatial quad-tree built from this average image. This structure supports multi-resolution in the temporal dimension by accessing the average images, and seems very appropriate for video sequences. Our approach covers more general meshes (considering images can be viewed as planar meshes) and treats the temporal dimension a little differently: instead of averaging the meshes over time spans, our temporal resolution defines the length of the intervals between each time sample.

Recently efforts on behalf of the MPEG4 standard organization [14] defined specific interpolators or behaviors for human figures or faces as well as rigid body transformations to efficiently encode dynamic meshes. Dealing with general dynamic meshes as in this paper was postponed for later dates. Moreover, 3D geometry compression either concentrate on static meshes [21, 22], or assume low geometry bandwidth [12, 23]. In [11] a method is proposed for compression of time dependent geometry. The vertex positions matrix is decomposed into $P \cdot V \cdot G$, where P is the time interpolation, V is the vertex positions at key time-steps and G is the geometry or spatial interpolation. The gross movement of the geometric mesh is extracted from V and encoded with a small set of controls. By separating the low and high frequency temporal information the residual magnitudes are reduced. The gross movement is encoded using affine transformations and the residuals are quantized at low bit rate. Our method uses a similar approach for separating low and high frequency temporal information, but uses them at two different levels of temporal details.

The rest of this paper is structured as follows: in Section 2 we briefly describe the TDAG structure, its construction and supported queries. Separating low and high frequency temporal information is described in Section 3, and the construction of the multi-resolution model in Section 4. Section 5 discusses the possible uses of the model and Sections 6 and 7 show an example and outline future work.

2. THE TDAG STRUCTURE

This paper focuses on triangular meshes. A triangular mesh $\mathcal{M}$ is a tuple $\mathcal{M} = (P, F, I)$ of vertices $P = \{p_i\}$ in E^3, faces $F \in P \times P \times P$, and some vertex attributes I such as position, normal and color. We consider a time-sequence of meshes:

$$\mathcal{M}_{t_0}, \mathcal{M}_{t_1}, \ldots, \mathcal{M}_{t_k}$$

where $t_0 < t_1 < \ldots < t_k$. All mesh components, i.e. attributes, positions and adjacency, become a function of the time t_i:

$$\mathcal{M}_{t_i} = (P_{t_i}, F_{t_i}, I_{t_i})$$

. The actual time values are irrelevant, and so we normalize the time-steps to unitary intervals $\{t_i\} \rightarrow i$.

We classify the modifications between two consecutive meshes $\mathcal{M}_i$ and $\mathcal{M}_{i+1}$ into four levels: attribute changes (e.g. change in vertex color), position changes (e.g. change in vertex positions), connectivity changes (changes to the set F), and topological changes (e.g. the genus of the mesh changes).

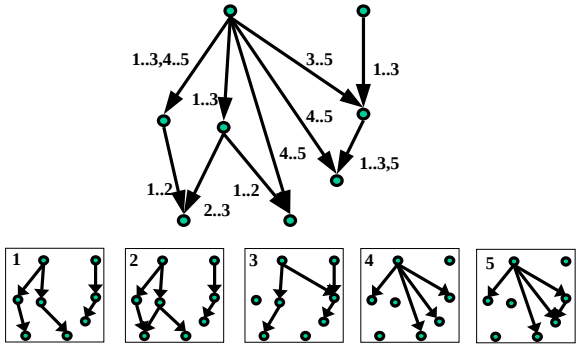

Figure 2: The child links in a T-DAG structure with five time-steps (top), and the five DAGs it represents (bottom). Each child link edge in the top T-DAG carries a tag depicting the range of time-steps in which it is active.

The TDAG (Temporal Directed Acyclic Graph) is a multi resolution data structure, which uses time-tags for all time dependent information. The TDAG has the ability to encode a large class of dynamic models, which may also include connectivity and topology changes. In particular, it deals with the symbolic information such as mesh connectivity and decimation dependencies in a similar manner as the numeric information such as attributes and positions of vertices (see Figures 2). All such information is stored as fields in the nodes of the TDAG.

There are two basic types of fields in a TDAG node: single-valued fields and multiple-valued fields. We define a *single-valued* field of a node n as a function $F_s : \Re \rightarrow \Re \cup \{\perp\}$, and a *multiple-valued* field as a function $F_m : \Re \rightarrow 2^{\Re} \cup \{\perp\}$. Error estimation, vertex coordinates, or color components have only a single possible value, and therefore are defined as single value fields. Parent and child links in the DAG have several values associated with them all valid at the same time, and therefore are defined as multiple-valued fields. Consider a dynamic mesh with k time-steps, for every t, $0 \leq t \leq k-1$, $F_s(t)$ is a real number, and $F_m(t)$ is a groups of numbers (for simplicity of notation even pointers are assumed to be represented as real numbers).

In a static multi-resolution DAG, all the node's fields would be static. In a dynamic setting they are time dependent and need to be represented as functions of time. In the TDAG this is done by attaching different time-tags to different values in the fields of nodes. These tags are in the form of sequences of ranges (i, j), with $i \leq j$, each defining a continuous time interval (t_i, t_j) where the specific value is *alive* in its field. A value x is defined as *not alive* at time step t iff its field returns $\perp$. If x has a time tag (i, j), then we call i its *birth time* and j its *death-time*. Note that x can have be multiple birth/death times.

> *The* **T-DAG** *is the collection of all values for all time steps of all the fields in all the nodes.*

Note that the whole TDAG can be a general graph but for any specific time-step t, the collection of all alive parent or all alive child links form an actual DAG at time t (see Figure 2). In general, a node n_0 in the TDAG might be a descendent of node n_1 at time t_i, and an ascendant of node n_1 at time t_j, as long as $(i \neq j)$.

The construction of the TDAG is done incrementally using an online algorithm. When each new mesh $\mathcal{M}_{i+1}$ in the sequence is presented, it is merged into an existing TDAG which encodes the meshes $\mathcal{M}_0, \ldots, \mathcal{M}_i$. The key idea behind the incremental TDAG construction algorithm is to create at each time-step a multi-

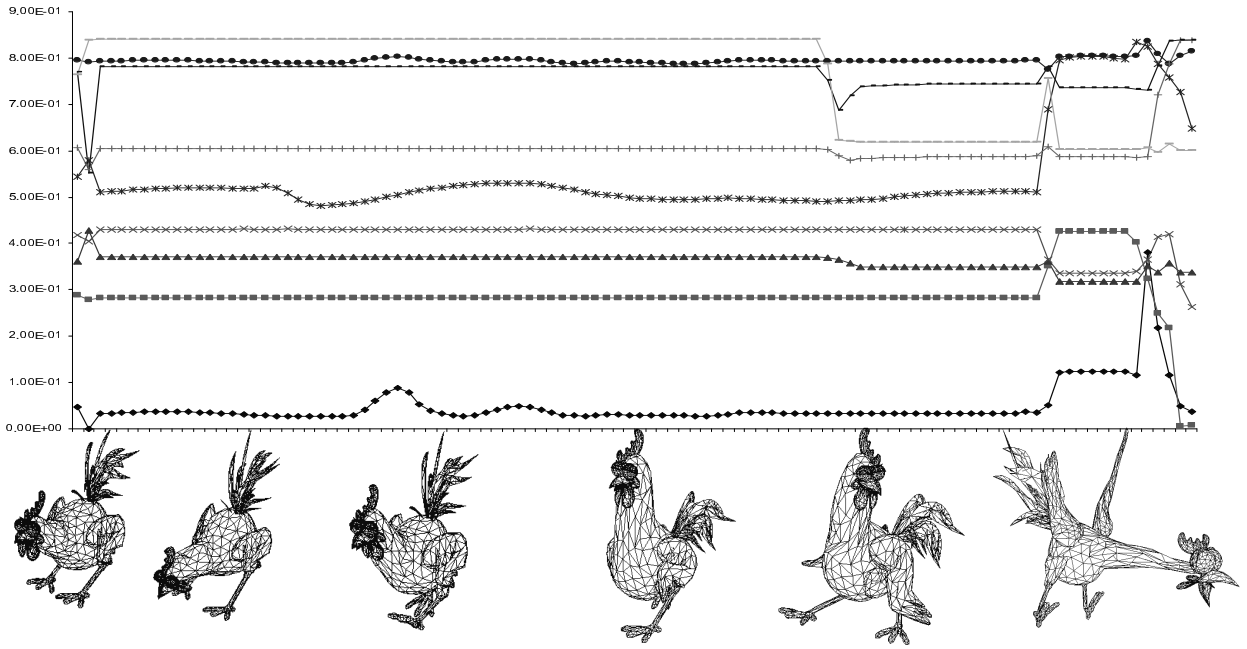

Figure 3: The decimation cost of different edges as a function of time: the x axis is time and the y axis is cost. Each plot series represents the cost of contracting one edge at different time steps. As can be seen, the cost can rise or drop depending on the local geometry of the mesh when the contraction is performed (the bottom line shows some snapshots of the mesh closely bellow their position in time). Edges that are part of the main body of the chicken tend to have smaller and more stable costs, while the cost of edges that are part of the wing or neck have a large jump in cost at the time of the surprised chicken.

resolution DAG using decimations which will conform to the existing TDAG. This is done by decimating $\mathcal{M}_{i+1}$ (for example, using edge contraction) with a metric function which combines current spatial constraints (such as quadric error metric) with global temporal ones. During the decimation of $\mathcal{M}_{i+1}$ an enhanced priority function is used for choose the next decimation element, introducing some history considerations which augment the regular priorities of decimation cost. The first part of the priority function optimizes the structure of the level of detail DAG for the current mesh, while the history constraints are targeted at preserving the structure of the DAG as much as possible over time. A weighted sum of the two components is used where the weights can be changed to adjust the conformity of the TDAG. More details on this can be found in [17].

As a model for the dynamic meshes $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k$, the TDAG is parametric in two dimensions: resolution and time. Therefore, every valid TDAG query must include these two parameters. There are two types of fundamental queries supported by the TDAG: the random type queries and the incremental type queries. Given *(time = t,tol =* ϵ_1*)*, the random query returns an approximation $\mathcal{M}_t^{\epsilon_1}$ of the mesh $\mathcal{M}_t$ which does not differ from $\mathcal{M}_t$ by more than ϵ_1 under some given error metric. Incremental query in time (resolution) will return $\mathcal{M}_{t+1}^{\epsilon_1}$ ($\mathcal{M}_t^{\epsilon_2}$ when $\epsilon_2 \neq \epsilon_1$).

Although the model described in this paper can support all query types, we will concentrate our discussion on progressive solutions to the incremental time query. In virtual reality worlds and animation sequences, this is the type of query that is mostly used. In such cases, the level of detail mesh of current time-step is progressively updated to arrive at the next time-step level of detail mesh.

3. SEPARATING DIFFERENT TEMPORAL FREQUENCY

If we track the trajectories of each vertex in the time sequence meshes $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k$, it could be difficult to distinguish between global mesh movements (such as rigid body transformations) and local fluctuations or deformations. In order to separate between these low and high frequency deformations we want to map all meshes to a uniform general position in space. We collect all n vectors of homogeneous coordinates of the vertices of mesh $\mathcal{M}_t$ into a global $4 \times n$ matrix V_t. Similar to [11], we select either the first time-step matrix V_0 or an average mesh matrix (created by the average vertex positions of all meshes over time) and search for A_t, a 4×4 affine matrix which is the best least square solution to the equation:

$$A_t V_0 = V_t$$

In practice, we solve the following equation and find A_t for each time-step t:

$$A_t = V_t V_0^T (V_0 V_0^T)^{-1}$$

Gathering, for all t, the affine coefficients of A_t defines the low frequency temporal information of the sequence $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k$. Applying A_t on V_0 and using the connectivity defined by $\mathcal{M}_0$ gives a first crude approximation for the mesh sequence (see Figure 4). The residual meshes $R\mathcal{M}_t$ are found by applying A_t^{-1} on each V_t, mapping each mesh $\mathcal{M}_t$ to the uniform position of $\mathcal{M}_0$ (see Figure 5).

When topology or connectivity changes occur over time, we must calculate the affine map by using a sub-set of vertices valid in both V_t and V_0. However, as will be described in Section 4, dynamic cut update is more difficult to implement in these situations, and often the mesh will need to be reconstructed from the roots of the DAG when a cut update is requested.

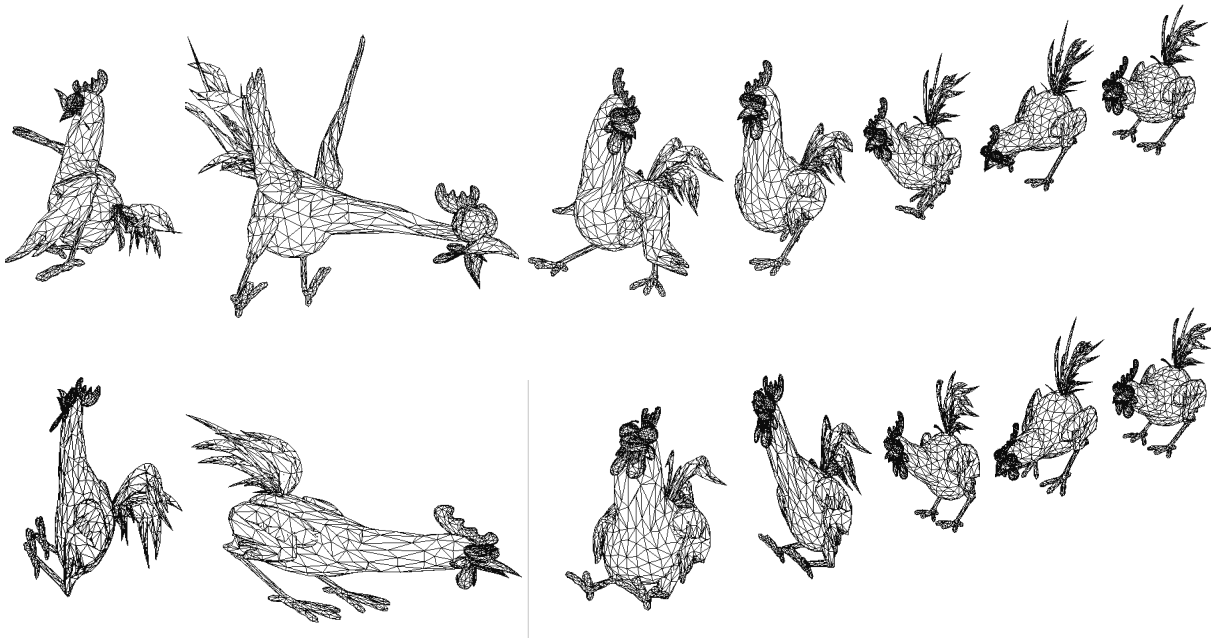

Figure 4: The low frequency deformation information of the chicken sequence (top) is applied to the first time-step mesh, creating the bottom sequence. Note that the mesh itself does not deform much, but its position, orientation and size (affine attributes) adhere to those of the original sequence.

4. HIERARCHY CONSTRUCTION

The multi-resolution TDAG imposes no restrictions on mesh deformations over time in terms of connectivity and topology changes. However, when the sequence of meshes do not change their connectivity over time (and therefore, also their topology), it is possible to use the same DAG structure for all time-steps. In this case, every cut in this DAG defines a valid mesh in all time-steps. This enables the use of an easy dynamic update algorithm of the cut, which creates the current level of detail of the mesh by updating the previous time-step cut (either expanding or contracting the cut depending on the costs at the nodes of the new time-step).

In order to store all time dependent meshes using the same DAG we need to use the exact same sequence of decimation operation for all time-steps. Note that this does not mean that the multi-resolution model will be constant over time. As can be seen in Figure 3, the same decimation operation may have different costs in different time-steps. The reason for this is that the mesh attributes, which effect the cost of decimation, change over time. This also means that for the same tolerance in different time-steps we might get different cuts of the DAG, and there is a need for updating the cut over time even if the tolerance is constant.

In an online scenario, we start constructing the TDAG by decimating $\mathcal{M}_0$. If we choose to use the same DAG for all time-steps, then this decimation determines the decimation order for all consequent time-steps. We can try and minimize the amount of cost changes over time if we abandon the online requirement for creating the TDAG. However, finding the set and order of the decimation operations that will minimize the cost change and the total cost over all time-steps is a very complex and costly optimization problem.

As a measure for the quality of decimation on all time-steps, we examine the maximum cost reached over time for every specific edge contracted. We then measure the average and median of those maximums over all edges. If E is the number of edges in the mesh, let n_i be the number of times edge i was used (contracted in one time-step mesh) during the construction of the TDAG. We define $N = \sum_{i=1}^{E} n_i$. Let c_{ij} be the cost of contracting edge i at time-step j. We define $c_{ij} = 0$ if edge i wasn't contracted at time j. Our average quality measure will then be:

$$\frac{\sum_{i=1}^{E} \max_j(c_{ij})}{N}$$

In our example (see Section 6) we used edge contraction and the quadric error metric [5] and chose to compare between choosing the first mesh, an "average" mesh (by averaging vertex positions over all time-steps) and an arbitrary mesh (number 277) to govern the decimation of more than 300 time-steps. We also measured the optimal decimation, where each mesh is decimated separately. The result of decimating the chicken sequence using an "average" mesh was not significantly better than decimating according to the first or to an arbitrary chosen mesh. In all of these cases the average quality measure doubled compared to the optimal decimation (see Figure 6). This stems from the fact that the high frequency deformation of this sequence is large (see the figures in this paper) and the "average" mesh is as far from the other meshes as any arbitrary mesh.

In this case (and probably also in many dynamic cases), not much is gained by using an average mesh to justify the abandonment of the online algorithm. Moreover, if we examine the median measure in Figure 6 it is much closer to the optimal case. This means that most of the costs of decimations are actually not much different from the optimal case. The large difference in the average measure probably comes from higher levels in the hierarchy of the DAG, which are used in cases where quality is much less a factor (e.g. when the mesh is very far from the viewer or for back faces).

It is important to stress that the cost of decimation should be calculated on the actual meshes $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k$, and not on the residual meshes $R\mathcal{M}_t$. This is because the quality of approxima-

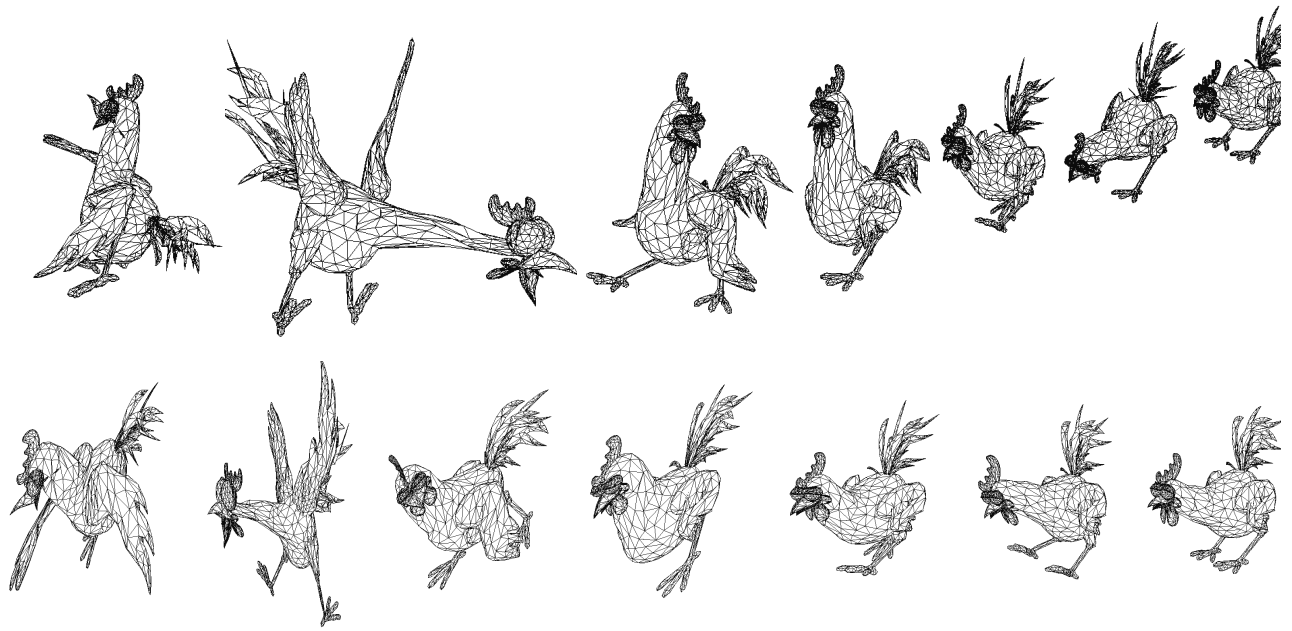

Figure 5: The application of the inverse of the affine transformation matrix to all meshes maps them to a general position, size and orientation in space. Only the high frequency deformation information of the original mesh sequence (top) changes in the residual meshes (bottom).

Governing Mesh	No. of decimations	Total decimations	Average Measure	Median Measure
First	2884	868084	2.14	0.666
Arbitrary	2890	869890	1.90	0.650
Average	2885	868385	2.19	0.687
Optimal	-	865480	1.09	0.629

Figure 6: A comparison between different strategies for governing the creation of the TDAG. In the first three lines all meshes were decimated using the same edges and the same order which is governed either by the first, the average or an arbitrarily chosen mesh. In the last line each time-step mesh is decimated separately optimally. As can be seen, there is not much difference in choosing a single mesh to govern the decimation, in all cases the average measure is about doubled with respect to the optimal. Therefore the first mesh is as good as any other mesh to be chosen in an online algorithm.

tion should be governed by the real mesh that will be displayed and not the residual.

Before discussing the possible usage of our model we summarize the online construction of the time dependent model. A mesh $\mathcal{M}_t$ from the sequence of meshes $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_k$ is encoded in the following way: first, we extract and store A_t, the affine transformation with respect to the first mesh. We then create the residual mesh $R\mathcal{M}_t$ by applying A_t^{-1} to $\mathcal{M}_t$. Then, we encode the positional and attribute changes of $R\mathcal{M}_t$ from $R\mathcal{M}_{t-1}$ in the TDAG structure.

5. LEVEL OF DETAIL UTILIZATION

Although level-of-detail models can be utilized to reduce the amount of time-dependent updates (by restricting them only to *active* vertices that are part of the current level of detail mesh), one must remember that dealing with time dependent meshes means the update of nodes attributes (position, color etc.) must be done continually through time for correct rendering.

As stated earlier, we focus on incremental queries in time: given a specific tolerance ϵ governing the overall level of detail, and given a level of detail mesh for the current time step mesh $\mathcal{M}_t$, we would like to create the next time-step level of detail mesh approximating $\mathcal{M}_{t+1}$. The complete procedure for creating the new mesh approximation involve three stages in the following order:

1. Updating the cut of $R\mathcal{M}_{t+1}$ according to the costs at $t+1$ (and possibly also a change in other parameters such as the direction of, and distance from the viewpoint).
2. Updating the attributes and positions of the active vertices in the resulting approximation residual mesh.
3. Applying the affine transformation A_{t+1}^{-1} on the resulting residual mesh.

Examining these three stages, it is evident that they are also ranked by their complexity. The first stage in creating the next time-step mesh is the most complex and time consuming, and the last is the most simple and rapid. However, in terms of their visual significance, the last stage is the most significant and the first stage the least.

The key idea in utilizing our multi-resolution model is to reverse the order of execution of these stages. This is feasible since those

three stages can be carried out almost independently, resulting in three levels of temporal approximations for the mesh:

1. The coarsest approximation is created using the previous time step spatial level of detail mesh by applying the new time-step affine transformation. Note that such operation can sometimes be as simple as calling `glPushMatrix()` and `glMultMatrix()` in OpenGL.

2. A better approximation is created using the previous time-step level of detail mesh, applying the affine transformation, and also updating the attribute and position of the active vertices that are shared by both time-steps. Hence, both the low and the high frequency deformations are applied, but using the previous time-step level-of-detail mesh instead of the current.

3. The best approximation is created by carrying out all the stages of mesh creation: updating the cut, updating the attributes of active vertices and applying the transformation.

The relative independence of the three mesh update stages can also be utilized in a multi-threaded virtual environment. The rendering thread, which reads the mesh and sends it down the graphics pipeline, can be responsible for the low-cost operation such as the affine transformations. The mesh creation thread, which writes the mesh, can be responsible for the cut update. This scheme resembles the internal double-buffer scheme of the graphics pipeline, using two working meshes - one for reading and one for writing and swapping between them. Similar to the frame buffer, if the writing thread lags behind in updating the cut, there will be no swapping of meshes. When a specific frame rate is required the writing thread can be tuned in advance to skip some time-steps and calculate the cut for every other time-step, when the reading thread fills the gaps with the affine transformations. Since the visual significant part of the temporal update will be carried out, the unpleasant artifacts such as jumps in position, orientation and scale will be reduced. The usual artifacts created by using multi-resolution models will still need to be addressed. The implementation of such multi-threaded rendering scheme is still under way and remains as future work.

6. RESULTS

The chicken sequence of meshes display a chicken character crossing a road realizing a truck is heading towards it, turning and trying to escape. The sequence includes 400 time-steps of 3030 vertices and 5664 triangles each. An uncompressed binary representation of the mesh takes around 14 megabytes and the TDAG and affine maps around 37.5 megabytes. On a 500 Mhz Intel Pentium 3 with 128 Mb of RAM, it takes around 0.3 second to calculate the affine map of each time-step and create the residual mesh and an average of 30 seconds to merge a new mesh into the TDAG.

Figure 4 presents samples of the original sequence of meshes (top) and the low frequency temporal information represented by applying the affine maps of each time step to the first mesh (bottom). Note how the general structure of the mesh remains more or less the same while the affine transformations change the position, orientation and stretch of the object. Figure 5 presents the sequence of meshes (top) and the high frequency temporal information, which are the residual meshes created by applying the inverse affine transformations of each time-step to the original mesh. Note in contrast to Figure 4 how the position and orientation remain constant, while the local and internal deformations of the meshes are revealed.

Lastly, Figure 7 shows examples of the results of the three levels of approximations for the mesh: applying only the affine maps (top row), applying the affine map and the attributes update (second from top), and then also updating the cut (the bottom three rows show level of detail created by three different tolerances)

7. SUMMARY

This paper presented a scheme for creating a level-of-detail model for time dependent meshes, which allows the utilization of both temporal and spatial level-of-detail approximations. This is made possible by a specific decomposition of the temporal information into low and high frequency deformations and the usage of a multi-resolution model for the spatial information.

As discussed in Section 5 the advantages of using this scheme can be exploited in any multi-threaded virtual environment. The first item in future extensions is the creation of a "double meshed" multi-threaded rendering system for time-dependent meshes. Other directions include lowering the size of the TDAG file representation by using fewer dependencies in the DAG, and exploiting the high temporal coherency in the TDAG for the use of some compression mechanism.

8. ACKNOWLEDGEMENTS

The chicken character is © Copyright 1996, Microsoft Corporation. It was created by Andrew Glassner, Tom McClure, Scott Benza, and Mark Van Langeveld.

This work was performed under the auspices of the U.S. Department of Energy by University of California, Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

9. REFERENCES

[1] M. de Berg and K. T. G. Dobrindt. On levels of detail in terrains. *Graphical Models and Image Processing*, 60:1–12, 1998.

[2] M. Eck, T. DeRose, T. Duchamp, T. Hoppe, H. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *ACM Computer Graphics Proceedings, SIGGRAPH'95*, Annual Conference Series, pages 173–180, 1995.

[3] A. Finkelstein, C. E. Jacobs, and D. H. Salesin. Multiresolution video. In *ACM Computer Graphics Proceedings, SIGGRAPH'96*, Annual Conference Series, pages 281–290, 1996.

[4] L. De Floriani, P. Magillo, and E. Puppo. Data structures for simplicial multi-complexes. In *Proceedings Symposium on Spatial Databases*, Hong Kong, China, July 1999.

[5] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In Turner Whitted, editor, *ACM Computer Graphics Proceedings, SIGGRAPH'97*, Annual Conference Series, pages 209–216. ACM SIGGRAPH, Addison Wesley, August 1997.

[6] Tran S. Gieng, Bernd Hamann, Kenneth I. Joy, Gregory L. Schussman, and Issac J. Trotts. Constructing hierarchies for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 4(2):145–161, April 1998.

[7] P. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. In *ACM Computer Graphics Proceedings, Annual Conference Series, SIGGRAPH'97, Multiresolution Surface Modelling, Course Notes No. 25*, 1997.

Figure 7: Examples of the three approximation levels for a multi-resolution dynamic mesh. The top row shows the coarsest level by applying just the affine transformations to the first time-step level of detail mesh. The second row shows the middle approximation by applying also the attributes update on the first mesh. The bottom three rows are created using the full update procedure for each time-steps, but using a different tolerance in each row. Note, for instance the differences in the triangles around the neck of the chicken in the different update schemes: in the coarsest and middle approximation (top two rows), the set of triangles does not change. In the third row in particular, the update of the cuts changes the set of triangles.

[8] H. Hoppe. Progressive meshes. In *ACM Computer Graphics Proceedings, SIGGRAPH'96*, Annual Conference Series, pages 99–108, 1996.

[9] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings IEEE Visualization'98*, pages 35–42. IEEE Comp. Soc. Press, 1998.

[10] R. Klein and J. Kramer. Multiresolution representations for surface meshes. In *Proceedings of the SCCG*, 1997.

[11] J. E. Lengyel. Compression of time-dependent geometry. In *Proceedings of the 1999 ACM Symposium on Interactive 3D Graphics*, Atlanta, Georgia, April 1999.

[12] M. Levoy. Polygon-assisted jpeg and mpeg compression of synthetic images. In *ACM Computer Graphics Proceedings, SIGGRAPH'95*, pages 21–28, 1995.

[13] A. Maheshwari, P. Morin, and J. R. Sack. Progressive tins: Algorithms and applications. In *Proceedings 5th ACM workshop on Advances in geographic information systems*, Las Vegas, 1997.

[14] INTERNATIONAL ORGANISATION FOR STANDARDISATION CODING OF MOVING PICTURES and AUDIO ISO/IEC JTC1/SC29/WG11 N2995. *MPEG4 standard specifications*, http://drogo.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm edition.

[15] J. Rossignac and P. Borrel. Multi-resolution 3d approximation for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Geometric Modeling in Computer Graphics*, pages 455–465. Springer Verlag, 1993.

[16] William J. Schroeder. A topology modifying progressive decimation algorithm. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization 97́*, pages 205–212. IEEE, November 1997.

[17] A. Shamir, V. Pascucci, and C. Bajaj. Multi-resolution dynamic meshes with arbitrary deformation. In *Procceedings of the IEEE Visualization Conference VIS'00*, pages 423–430, 2000.

[18] H. Shen, L. Chiang, and K. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *Proceedings of the IEEE Visualization Conference VIS'99*, pages 371–378, 1999.

[19] E. J. Stollnitz, T. D. DeRose, and D. H. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, 1996.

[20] P. M. Sutton and C. D. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree (t-bon). In *Proceedings of the IEEE Visualization Conference VIS'99*, pages 147–154, 1999.

[21] G. Taubin, A. Gueziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *ACM Computer Graphics Proceedings, SIGGRAPH'98*, Annual Conference Series, pages 123–132, 1998.

[22] C. Touma and C. Gotsman. Triangle mesh compression. In *Proceedings of Graphics Interface '98*, pages 26–34, 1998.

[23] D. S. Wallach, S. Kunapalli, and M. F. Cohen. Accelerated mpeg compression of dynamic polygonal scenes. In *ACM Computer Graphics Proceedings, SIGGRAPH'94*, pages 193–197, 1994.

View publication stats