Multi-layered storage and transmission for animated 3D polygonal meshes

K. Selçuk Candan and M. G Wagner

Arizona State University,
Computer Science and Engineering Department,
Arizona State University,
Tempe, AZ 85287-5406.

ABSTRACT

Highly detailed geometric models are rapidly becoming common-place in computer graphics and multimedia. MPEG4 is introducing standards for three-dimensional polygon meshes. Analogously to video, highly complex polygon meshes challenge not only rendering performance but also transmission bandwidth and storage capacities. Current work focuses only on the compression, and progressive refinement and transmission of static polygon meshes. On the other hand, geometric content is no longer static, it allows user interaction and complex changes within the geometry. Since this emerging problem has to be addressed in any future multimedia standard this paper aims to open the research by exploring similarities and differences in transmission and storage of dynamically changing polygon meshes and geometry compared to other stream data, such as video. In this paper we propose techniques for multi-layered storage and transmission of animated geometry in low bandwidth environments. Our approach extends existing technologies for multi-layered representation of static polygon meshes to dynamic architectures.

Keywords: 3D media servers, geometric data transfer, geometric compression, virtual reality, computer animation

1. INTRODUCTION

Very recently the Internet has emerged from a pure communication tool to a complex multimedia retrieval system. Part of this development was the introduction of the VRML (Virtual Reality Modeling Language) international standard in 1997[1] or more recently the introduction of Chromeffects with Windows98. These technologies allow users to view and experience three-dimensional virtual worlds in a standard Web browser. While there is no doubt about the importance of this development which is also manifested in the upcoming MPEG4 standard,[2] it imposes serious challenges to research and content development alike.

VRML and Chromeffects have been designed to describe complex three dimensional virtual environments. In contrast to standard HTML based web pages this information is distributed over three dimensional space and time. The question whether and how some visual information is perceived by the viewer hence also depends on his location within the virtual environment. This paper presents the first approach for storage and transmission of dynamically changing 3D geometric data that is suitable for view dependent rendering and transmission of a virtual environment. We therefore focus on the description of polygonal meshes which is sufficient to describe arbitrary geometric information. In particular we deal with animated or morphing quadrilateral meshes.

1.1. Problem Statement

This paper aims to open the research in transmission and storage of dynamically changing polygon meshes and geometry. Since, both animations and captured video have visual and temporal characteristics, the problem of efficient storage and transmission of animated 3D polygon meshes is related to that of video.

Since, video information has been captured, stored, and manipulated for a longer time, there already exusts an excellent and extensive body of literature that addresses the challenges associated with video. Rangan *et al.*[3] provide an excellent survey of video storage and retrieval. Other researchers[4–6] develop disk models that facilitate

Other author information
K.S.C.: email: candan@asu.edu
M.G.W.: email: wagner@asu.edu
This work was supported by an ASU-FGIA Grant F-014-98.

supporting different kinds of VCR operations such as fast forward, rewind, pause, etc. Ng and his colleagues[7,8] have developed efficient buffer management techniques to handle video data and ensure that the clients' consumption rate requirements are met. In Ref. 9, we first proposed a *formal model* for continuous-media retrieval from heterogeneous disk servers. We then developed algorithms for optimally using resources in a multi-user interactive video-on-demand server.

Despite the similarities between video and animated quadrilateral meshes, there are significant differences between the two. These differences can be listed as follows:

- *The characteristics of data sizes are different:* An uncompressed video stream has a fixed bandwidth requirement:
$$B_{video} = width \times heigh \times color\_depth.$$
On the other hand, an uncompressed morphing mesh (or animation) stream has a time, scene, and user dependent bandwidth requirement:
$$B_{animation}(t, s, u) \sum_{i=1}^{num\_objects(s,t)} \sum_{j=1}^{num\_visible\_points(obj_i,u,s,t)} num\_bits\_per\_vertex,$$
where $num\_objects(s, t)$ is the number of objects in the scene, s, at time t; and $num\_visible\_points(obj_i, u, s, t)$ is the number of points in object obj_i visible to user u in the scene s at time t. Note that the number of visible points depends on the position of the user in the scene at the given point in time, as well as on user's preferences in terms of object priorities. Furthermore, since identifying the visible vertices may be a costly process, in some cases, the number of vertices sent may be larger than the actually visible vertices. In both cases, however, the number of vertices communicated will be a function of the scene, time, and user's view point.
- *The interaction characteristics are different:* In a video stream, the user interaction is generally limited to the fast-forward, rewind, pause, replay, or random access operations. These operations do not alter the content of the video stream. Consequently, to enable interactive presentation of video streams by multiple users, various techniques, such as prefetching, caching, and user grouping, can be employed.
On the other hand, in 3D animations, users may interact with the content of the stream, altering objects, object placements, their view points, or story line. From this respect, 3D animation streams are similar to authored multimedia documents, rather than recorded video. In an earlier work,[10,11] we developed a probabilistic model of interactive multimedia presentations where the probabilities reflect the likelihood of certain user interactions. Based on this probabilistic model, we developed prefetching algorithms that anticipate how users will interact with the presentation.
- *The characteristics of application domains are different:* Although application areas of video are very spread, from telecommunications (which accepts lossiness) to medicine (which does not accept any lossiness), 3D streamed animations did not achieve such a wide application spectrum yet. Currently, the WWW is the main application domain. Consequently, the amount of bandwidth available for transmission is very limited (28Kb to 56Kb in most cases). Although, this may eventually change with the emergence of newer application domains, a current standard must take this limitation into account.

Because of these differences between video streams and interactive 3D animations, we see that the current video standards can not be directly applied for animated quadrilateral meshes. Consequently in this paper we are addressing the problem of *efficient storage and transmission of animated 3D animations* and we are proposing techniques for low-bandwidth interactive animation streaming and storage. The proposed techniques are part of a research for defining a 3D streamed animation standard.

Note that, in this paper, we assume that users can interact with the system by changing their view points, object priorities, and the presentation direction and speed. Consequently, clients can not change the content of the animation, yet the information regarding their view point can let the server identify the set of objects and vertices that should be transmitted.

2. FROM STATIC TO ANIMATED MESHES

State of the art modeling and scanning systems frequently produce meshes of extremely high complexity. As an example, Figure 1 shows a Chinese dragon obtained from a standard 3D scanning system at Stanford University. In its original configuration, the object consists of 871,414 triangles. Due to this extremely high volume of data it is impossible to use this object in a virtual environment which is distributed through a low or even medium bandwidth network. In addition, the amount of data obtained by the data acquisition tools does not necessarily correlate to the actual visual information associated with the object. In many multimedia applications, a simplified version of the dragon consisting of only 7,402 triangles would be much more appropriate.

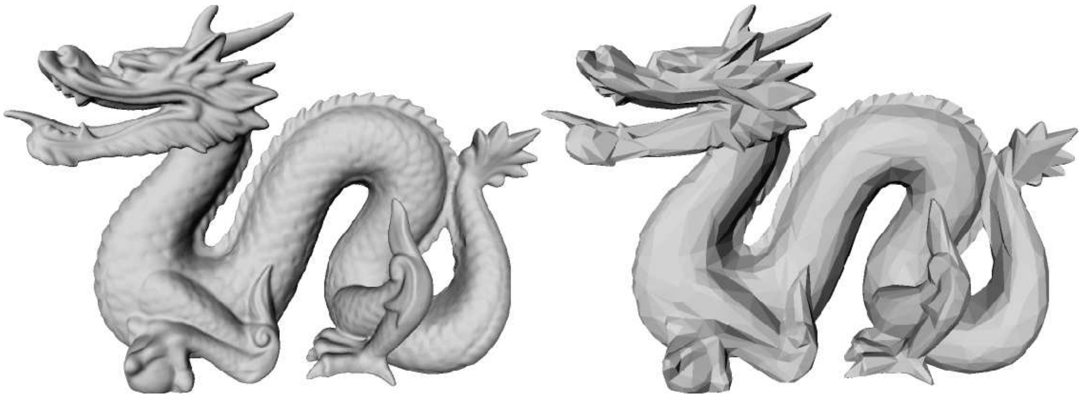


Figure 1. Two mesh models of a Chinese dragon. The original left model consists of 871,414 triangles, the simplified model on the right of 7,402 triangles.

Mesh simplification. It has therefore been necessary to study techniques that allow the automatic simplification of a mesh with respect to a given error tolerance. Various approaches have been proposed in the past including vertex removal,[12] edge collapsing,[13–15] and wavelet based techniques.[16–18] These approaches can be divided in two fundamentally different categories.

- *Topology preserving algorithms.* These algorithms preserve the topological type of the mesh and thus the genus of the approximated surface. Most currently known algorithms fall into this category.[13–15] Quite often, this is considered to be a disadvantage. In Ref. 16, Eck *et al'.* had to compute a base mesh of the same topological type which then served as the parameter domain for their multi-resolution approach. This computationally expensive precomputation step is necessary due to the topology preserving nature of their wavelet based algorithm.
- *Topology independent algorithms.* These algorithms do not necessarily produce simplified meshes of the same topological type as the original mesh.[19] Although preferable, the corresponding approaches usually require the mesh to be rearranged after each simplification step which leads to computationally more expensive algorithms. In Ref. 14, Popovic and Hoppe introduced so-called progressive simplical meshes which solve this problem by employing concepts from algebraic topology. Nevertheless, as their examples show, the algorithm produces rather unexpected results for very coarse mesh approximations.

It has to be noted that all of these approaches have one fundamental concept in common. They simplify the mesh by reducing the number of vertices. In distributed multimedia applications, however, this might not always be the most suitable approach. Instead of reducing the number of points one could also try to optimize the way in which the point coordinates are encoded. In image and video processing, this is a very common concept resulting in lossy compression schemes such as JPEG. In computer graphics, however, this concept is mostly unused since previous research has been more concerned with rendering speed rather than transmission speed or storage size.

Progressive meshes. File sizes of multimedia objects challenge not only the bandwidth of the used communication lines but also the patience of the user who wants to retrieve a specific object. It is therefore preferred to transmit multimedia objects in a progressively refining way. This allows the user to decide during transmission, if the specified object justifies waiting for the download to be completed. Such progressive encoding techniques are well established for image media. For polygonal meshes, progressive transmission is achieved by generating a multi-layered data structure from a sequence of different approximation levels of the polygonal mesh. Therefore, one can use an almost arbitrary mesh simplification algorithm. The only requirement is that each finer approximation of the mesh must be computable from the coarser version by adding some detail features. Commonly used techniques use either different versions of a method called vertex split, which is the inverse procedure of the above mentioned edge collapse,[20,14] or are based on a wavelet analysis of the mesh.[16,17]

Automatic level-of-detailing Similarly, progressive meshes can be applied to partially solve the view dependent rendering problem by so-called automatic level-of-detailing.[20] In this case, the different detail layers are used to generate approximating meshes of different complexity. Depending on the distance of the viewer to the object, one can then choose the appropriate level-of-detail which optimizes rendering performance. So far, however, most automatic level-of-detail algorithms are global in the sense that they are incapable of using different levels of detail for different areas of a single object at the same time.

Animated meshes. With the introduction of VRML, geometric data is no longer static. Objects can change their shape over time either by predefined deformations or due to interaction with the viewer. As a result it has become important to consider animated polygonal meshes and to investigate possibilities for optimizing rendering, storage and transmission speed. In any case, the main challenge is the enormous amount of data necessary to describe dynamically changing meshes.

To give an example, consider the simplified version of the dragon in Figure 1. It consists of about 7,500 triangles, which, considering the connectivity information of the mesh, ad up to an estimate of 60 kB of data. Let us consider that we want to generate an animation by describing each frame individually. In the sequel, we will refer to this technique as the frame based approach to animated meshes. Todays browser are capable of rendering about 15 frames per second. Although this is not considered to be completely sufficient to generate a smooth animation, it still results in about 900 kB of file size per second of animation. If we would like to stream such an animated mesh over a network, it would require us to have the full bandwidth of a 10 Mbits per second network connection available. This is unrealistic even in the more distant future.

Animation parameters. The use of animation parameters is currently the only approach used in dealing with the massive amount of data necessary for describing animated objects. In this case, an object's deformation is described and communicated by a relatively small set of parameters which are then used by the client in order to reconstruct the mesh. This approach is especially well studied for human body and facial animation where the movements are described by joint angles or positions of certain reference points, respectively. It is also a vital part in MPEG4.[2]

Nevertheless, there are reasons why this approach leaves an unsatisfactory impression. First of all it requires the client to do all the computation since only the animation parameters are transmitted. Second, it requires the definition of animation parameters for each and every object individually. While there is a broad consensus on how the animation parameters of a humanoid should look like,[2] there exist applications, such as the animation of animals or more abstract morphing shapes, which have drawn very little or no interest at all yet. Finally, all current approaches produce only very coarse approximations of realistic movements. Unfortunately, the way an object changes its shape during a motion is very often too complex to be described by a small set of parameters.

Frame based mesh animation. The logical consequence is that despite the necessity of pursuing an approach based on animation parameters for various specialized applications, it also appears to be necessary to look into the possibility of frame based mesh animation for generic applications. In literature this has not been considered yet. It is important to point out that this approach can coexists with the standards included in the MPEG4 proposal.

3. ANIMATION AND USER MODELS

In this section, we describe how we model animations, users, user interactions, and stored and transmitted animation streams. In the following sections, we will discuss how we achieve the multi-level compression required by this model. We start by the modeling of an animation:

Animation. We model an uncompressed 3D animation, a, as a 4-tuple $\langle V, C, S, O, loc(v,t)\rangle$, where V is a set of vertices, C is a mesh which describes the connectivity of the vertices, S is the static properties of the vertices in V, such as their color values or IDs of the objects they belong to, O is a mapping from object IDs to the constituent sets of vertices, and $loc(v,t)$ is function, $V \times T \rightarrow R^3$, describing where in 3 dimensional space a given vertex, $v \in V$, is located at time $t \in T_a$ (T_a is the animation timeline).

Although animations can be created in declarative or functional languages, such as VRML, the above model describes an animation in terms of the positions of the vertices involved. Note that animations modeled as such are not interactive as the model does not describe how users can alter the content of the animation. However, by assigning some of the objects as *interaction objects* and switching animation (story) timelines when users interact with these objects, interactivity of the animation could be increased. Note also that, we do not include the properties of mesh faces in our model, but the model can easily be extended to encapsulate face information.

User. We model a user, u, viewing an animation, a, as a 4-tuple $\langle B(t), E(t), W(t), FF(t)\rangle$, where $B(t)$ is the bandwidth available for the user, $E(t)$ is the position and direction of user's eye in the 3 dimensional space, $W(t)$ is the shape, position, and size of user's view window, and $FF(t)$ is the speed by which user watches the animation ($FF(t) > 0$ means animation moves forward, $FF(t) = 0$ means the presentation is paused, and $FF(t) < 0$ means the presentation moves backward). Here, $t \in T_w$, which means that t is the real world timeline.

The above model of the user describes the resources available to the user as well as his or her presentation requirements (such as the presentation speed and direction as well as the view point). Since different users accessing the same animation may have different resources and since they may have different presentation specifications, the information that is sent to each one of them may be different.

User projection of an animation. The projection, $a(u)$, of an animation, a, onto a user, u, is a 4-tuple $\langle V(t), C, S, o, loc(v,t), T(t)\rangle$, where $V(t)$ is the set of vertices visible to user u at time $t \in T_w$, C is a mesh which describes the connectivity of the vertices, S is the static properties of the vertices in V, such as their color values or IDs of the objects they belong to, O is a mapping from object IDs to the constituent sets of vertices, and $loc(v,t)$ is a function, $V(t) \times T_w \rightarrow R^3$, describing where in 3 dimensional space a given vertex, $v \in V(t)$, is located at time $t \in T_w$. In addition T(t) is a function, $T_w \rightarrow T_a$, which maps the points in the real world timeline into points on the animation timeline.

Server. We model an animation server, s, as a 4-tuple $\langle Buf, SB, Seek_U, Seek_W\rangle$, where Buf is the size of the buffer, SB is the processor bandwidth (i.e., the amount of vertices that can be served by the server each second), $Seek_U$ is the amount of time wasted each time the server context switches from one user to an other, and $Seek_W$ is the amount of time wasted each time a user changes view point and direction.

Note that the seek times given above are not the disk seek times used in modeling video or disk servers, though disk bandwidth and seek and rotational delays are also used in calculating the animation server bandwidth and seek times. Especially in the case of $Seek_W$, one of the major bottlenecks is the amount of time required to identify the set of vertices that need to be sent to the user.

Stream. A stream, $stream(s,a,u)$, of an animation, a, from a server, s, to a user, u, is equal to $a(u)$.

An animation stream is a function of the user for two reasons: the user interaction and the user view point. However, as defined such, the animation stream does not depend on the bandwidth, available to the user. Below, we define compressed stream which scales up and down in terms of bandwidth requirements according to the user bandwidth.

Compressed stream. A compressed stream, $cstream(s,a,u)$, of an animation, a, from a server, s, to a user, u, is a 4-tuple $\langle V^*(t), C, S, loc^*(v,t), T(t)\rangle$, where

- $t \in T_w$,
- C is a mesh which describes the connectivity of the vertices,
- S is the static properties of the vertices in V,
- $V^*(t) = \{V_1(t), \ldots, V_n(t)\}$, where $V_i(t)$ is a set of vertices not necessarily in $V(t)$,
- $loc^*(t) = \{loc_1(v,t), \ldots, loc_n(v,t)$, where $loc_i(t)$ is function, $V_i(t) \times T \rightarrow R^3$, describing where in 3 dimensional space a given vertex, $v \in V_i(t)$, is located at time t,
- T(t) is a function, $T_w \rightarrow T_a$, which maps the points in the real world timeline into points on the animation timeline, and

there exists a group of functions, *compress*, *uncompress*, *bandwidth_requirement*, such that

- the i^{th} layer of compressed animation is equal to the set of vertices, $V_i(t)$,
$$compress(\langle V(t), loc(t)\rangle, i) = \langle V_i(t), loc_i(t)\rangle$$
- uncompressing a compressed stream of animation using all the compression layers results in an animation visually equivalent to the uncompressed animation. Note that there is no concrete definition of visual equivalence; in this work, we use the equivalence of vertices and their locations to denote visual equivalence. One important difference between image or video compression and the animation compression is that although a large error in a single pixel may not render an image visually wrong; a large error in the location of a single vertex may result in a drastic decrease in the visual quality. Consequently, the visual equivalence have to be defined more carefully and the errors that are introduced by the compression algorithm must not vary to much from vertex to vertex:
$$uncompress(\langle V_i(t), loc_i(t)\rangle \mid_{i=1}^{\infty}) \equiv_{vis} \langle V(t), loc(t)\rangle$$
- omitting any compression layers results in a visual decrease in quality:
$$\forall_{k\in Z+} uncompress(\langle V_i(t), loc_i(t)\rangle \mid_{i=1}^{k}) \leq_{vis} \langle V(t), loc(t)\rangle,$$
- the more layers one omits, the lower the final visual quality is:
$$\forall_{k_1<k_2\in Z+} uncompress(\langle V_i(t), loc_i(t)\rangle \mid_{i=1}^{k_1}) \leq_{vis} uncompress(\langle V_i(t), loc_i(t)\rangle \mid_{i=1}^{k_2}),$$
- Bandwidth requirements monotonically increase with the amount of detail involved:
$$\forall_{k_1>k_2\in Z+} \sum_{i=1}^{k_1} bandwidth\_requirement(V_i(t), loc_i(t)) \geq \sum_{i=1}^{k_2} bandwidth\_requirement(V_i(t), loc_i(t)),$$
- The total bandwidth used for transmission is less than the bandwidth available for the user:
$$\sum_{i=1}^{n} bandwidth\_requirement(V_i(t), loc_i(t)) \leq B(t), \text{ and}$$
- An additional layer of detail would violate the bandwidth constraint of the user:
$$\sum_{i=1}^{n+1} bandwidth\_requirement(V_i(t), loc_i(t)) > B(t).$$

A compressed animation stream consists of a set of layers of animation, such that the layers fit into the bandwidth requirement of the user at all times. This means that, in scenes which do not have too many vertices, more layers can be sent to the user and in scenes which contains many vertices, the number of layers can be reduced. A key observation, however, is in contrast to the video transmission, the number of vertices to be transmitted can vary significantly depending on the user's view point in the scene. Consequently, *there is a larger room for prefetching when there are not too many point to transmit. This is one of the driving forces of our research.*

User request. A user request, $req(u, s, a)$, from user, u, to server, s, for animation, a, is one of the followings:

- *Registration request:* $\langle a, B, E, W, FF\rangle$, where B is the bandwidth available for the user, E is the position and direction of user's eye in the 3 dimensional space, W is the shape and size of user's view window, and FF is the speed by which user watches the animation (again, $FF > 0$ means animation moves forward, $FF = 0$ means the presentation is paused, and $FF < 0$ means the presentation moves backward).
- *Bandwidth change request:* $\langle B_{new}\rangle$, where B_{new} is the new bandwidth available for the user.
- *View change request:* $\langle E_{new}, W_{new}\rangle$, where E_{new} is the new position and direction of user's eye in the 3 dimensional space and W is the shape and size of user's view window.
- *View direction and speed change request:* $\langle FF_{new}\rangle$ where FF_{new} is the new speed.
- *Random access request:* $\langle t\rangle$, where $t \in T_a$ is a time point with respect to the start of the animation.
- *Stop request:* $\langle\perp\rangle$, which means that the user wants to stop viewing the animation.

The user interaction model given above does not assume that users can change the content of the animation. However, users can change the set of vertices that they will see through view point changes as well as through random access in animation. Also, if the animation contains interaction objects, i.e., those objects that change the way animation timeline (or the animation story) moves, users can also interact with those objects through

- *interaction request:* $\langle o_i, act_j\rangle$,

where o_i is an object and act_j is an action defined on act_j). Some of the user interaction, such as view point changes, random access patterns, and presentation direction and speed changes are harder to predict (and subject of ongoing research). However, for prefetching and user clustering purposes, some criteria can be more easily defined:

Overlapping users. Two users, u_1 and u_2, are overlapping at time $t \in T_w$ if and only if

$V_{u_1}(t) \cap V_{u_2}(t) \neq \emptyset$, $FF_{u_1}(t) = FF_{u_2}(t)$, and $T_{u_1}(t) = T_{u_2}(t)$.

Overlapping users are those users who see a common set of objects in the animation simultaneously and with the same direction and speed.

Future overlapping users. Two users, u_1 and u_2, are future overlapping at time $t \in T_w$ if and only if

$V_{u_1}(t) \cap V_{u_2}(t + \Delta t) \neq \emptyset$, $FF_{u_1}(t) = FF_{u_2}(t)$, and $T_{u_1}(t) = T_{u_2}(t + \Delta t)$.

Future overlapping users see a common set of objects, with the same speed and direction, but with a phase difference.

Potentially overlapping users. Two users, u_1 and u_2, are potentially overlapping at time $t \in T_w$ if and only if

$V_{u_1}(t) \cap V_{u_2}(t) \neq \emptyset$, $FF_{u_1}(t) = FF_{u_2}(t)$, and $T_{u_1}(t) = T_{u_2}(t + \Delta t)$.

Potentially overlapping users see a common set of objects, with the same speed and direction, but with a phase difference. Furthermore, at (real) time t, the objects that they see overlap. However, since users can change their view point or direction at any time, it is not certain that the user coming from behind will see the same objects as the other one at the phase-compensated time point in the animation.

Master-slave users. Two users, u_1 and u_2, are master-slave at time $t \in T_w$ if and only if

$V_{u_1}(t) \cap V_{u_2}(t) \neq \emptyset$, $(0 < FF_{u_1}(t) < FF_{u_2}(t))$ or $(FF_{u_2}(t) < FF_{u_1}(t) < 0)$, and $T_{u_1}(t) = T_{u_2}(t + \Delta)$.

Master slave users see a common set of objects, at one point in real world time. Furthermore, one of the users is watching the animation faster than the other one and catching in animation timeline. Consequently, the slave user will watch everything that has been watched by the master until catching up (unless there is a change in view parameters).

4. PRIORITIZED MULTI-LAYERED STREAMING AND PREFETCHING

In the above section, we described the parameters required for multi-layered streaming of animated 3D meshes. In this section, we describe how we can benefit from the very dynamic characteristics of these animated meshes in increasing the utilization of the available bandwidth. As we mentioned earlier, in contrast to video, the number of vertices to be transmitted in an animation can vary significantly depending on the user's view point in the scene. Hence, due to this variance, there is a larger room for *prefetching*.

Since users are free in their movements in the animation time and space, given a user, $u = \langle B(t), E(t), W(t), FF(t) \rangle$, and an animation $a = \langle V, C, S, O, loc(v,t) \rangle$, the user projection of the animation $a(u)$ can not be generated beforehand. Consequently, prefetching has to benefit from additional user behavior models and animation properties.

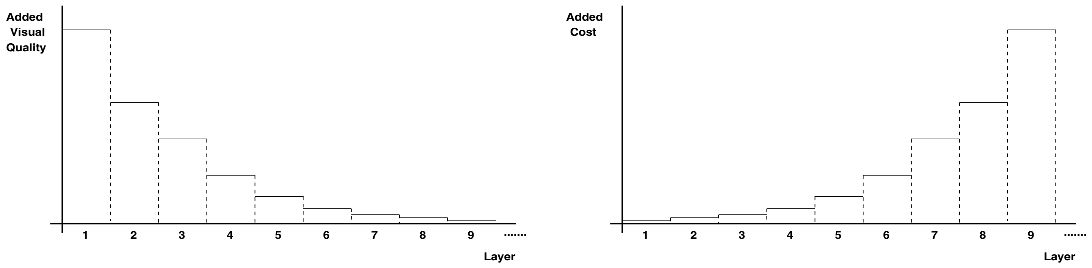
In this section, we will assume that each object, $o \in O$, has an associated priority, $pri(o)$, which can either be specified in the animation or provided by the user, u. We will also assume that, given any $st_1 = \langle E_1, W_1, t_1 \rangle$ and $st_2 = \langle E_2, W_2, t_2 \rangle$ (3-tuples denoting the view characteristics of the user in the animation space-time), there is a probability, $p(st_1, st_2)$, which denotes the probability of user's view point to switch from st_1 to st_2. This probability may depend on the structure of the animation, the interactive objects within the animation, as well as the user's behavior model. Consequently, it is possible to calculate two new values

- $pv(o, \delta t, st_1)$, which describes the probability of object o being visible within δt time units, given a view description, st_1, in animation time and space.
- $ed(o, \delta t, st_1)$, which describes the expected closest distance of o to the user within the next δt time units.

Given a time $t \in t_a$, and user view point, direction, and window properties, E and W, the objects to receive are decided using the object prefetching algorithm (OPA):

Input:

- A time $t \in t_a$,
- User view point, direction, and window properties, E and W,
- A set of objects O in the animation,
- A bound δt on the future.
- $B_{prefetch}$, is the amount of prefetch bandwidth available at time t.



(a) (b)

Figure 2. (a) Visual gain and (b) cost per compression layer.

1. Let st be $\langle E, W, t\rangle$,
2. For each object $o_i \in O$, calculate $gain(o_i, k) = pv(o_i, \delta t, st_1) \times pri(o_i) \times visual\_quality(o_i, k, ed(o_i, \delta t, st_1))^*$, as the system gain to prefetch k layers of the object.
3. For each object $o_i \in O$, calculate $cost(o_i, k)$ as the bandwidth required to retrieve k layers of the object.
4. Identify a set of objects, O_{retr}, and a mapping, $num\_layers(o)$, from O_{retr} to integers, denoting the number of layers to be transmitted, such that

$$\sum_{o_i \in O_{retr}} gain(o_i, num\_layers(o_i)) \quad \text{is maximum, and}$$

$$\sum_{o_i \in O_{retr}} cost(o_i, num\_layers(o_i)) \leq B_{prefetch}.$$

Figure 2 shows how each layer in the presentation adds to the visual quality and the transmission cost. The increase in cost with the increase in the layer number is because higher layers contain more (detail) information.

When $k = 1$, identification of the suitable objects for prefetching is an NP-complete problem (knapsack) with a pseudo-polynomial solution. When $k > 1$, the problem can be solved with a pseudo-polynomial algorithm modified to make sure that object layers are added in the proper order. The complexity of the modified algorithm is $O(|O| \times number\_of\_compression\_layers \times B_{prefetch})$, where $|O|$ is the number of objects, $number\_of\_compression\_layers$ is the number of compression layers per object, and $B_{prefetch}$ is the bandwidth available for prefetching. In addition, polynomial time approximations to the knapsack problem (with appropriate modifications) can be used to identify these objects. Note that, if we allow fractions of the layers being transmitted, then there is a polynomial, $O(number\_of\_compression\_layers \times |O| \times \log(|O|))$ time, greedy algorithm. However, this requires further research in identifying appropriate fraction of a layer for transmission and the corresponding visual effects on the animation.

The above algorithm is based on the assumption that it is possible to compress 3D data in a layered form. In the next section, we describe how the multi-layered compression of 3D geometric information for storing and transmitting animated data is accomplished in the proposed framework.

5. LOSSY COMPRESSION OF ANIMATED QUADRILATERAL MESHES

Our main objective is to derive lossy compression schemes for frame based mesh animation that lends themselves into creation of compression layers. The existing edge collapsing based approaches do not naturally lead into compression layers. As outlined above, one of the major challenges that we need to address is the massive amount of data associated with this approach. In order to solve this problem, our approach employs a two step process: (1) encoding and compression of static polygon meshes and (2) efficiently encoding sequences of animated meshes.

Without loss of generality, we may assume that the meshes we work with have quadrilateral structure. The reason for this restriction is that due to the specific structure of quadrilateral meshes, they allow much higher compression and are therefore more suitable for our purpose. If we are dealing with a more general mesh, we assume that the mesh has first been converted to a quadrilateral mesh or a patchwork of quadrilateral meshes. For that purpose the quadrilateral submeshes may show singular parameterization.

$^*visual\_quality(o_i, k, ed(o_i, \delta t, st_1))$ describes the visual quality that will be achieved by displaying the fitst k layers of the object o_i when the object is $ed(o_i, \delta t, st_1)$ units away.

5.1. Encoding and compression of static quadrilateral meshes

Standard mesh simplification schemes can only be applied up to a certain level. If the simplification is too coarse, the mesh will loose significantly in visual quality. In the dragon example above, this level is reached at about 7,000 triangles. Any further simplification would lead to poor visual results. 7,000 triangles, however, is still too much to achieve a reasonable transmission rate. In order to overcome this problem, the only possible way is to encode the simplified mesh such that the amount of data is reduced without any further reduction of the number of points. The proposed compression scheme therefore consists of a two step process: (1) Mesh simplification down to the minimum number of points necessary to maintain visual quality. (2) Proper encoding of the point coordinates to reduce file size and application of a lossy compression technique. It is the second step in this process which is completely new in the context of polygonal meshes.

Why not wavelets? Since we are restricting the present work to quadrilateral meshes, one could address the encoding and compression problem by applying standard algorithms used in image compression, for example by employing wavelets. However, it has to be considered that the information given by a mesh is significantly different to that of an image. Since we first apply a mesh decimation algorithm we have to assume that the uncompressed mesh is only an approximation of the actual object lacking high detail information. Consequently, wavelet compression would give only a relatively poor compression ratio. Furthermore, if the mesh has been generated by a CAD software package we have to consider that most CAD systems work with cubic representations of surfaces for example in the form of tensor product Bézier or b-spline surface patches. A compression scheme which exploits the properties of these types of surfaces is therefore much more suitable for the particular application we are interested in. The approach we are proposing takes advantage of the properties of cubic surface patches. In short we will encode the mesh with respect to the deviation of an "optimally" smooth surface mesh. The actual encoding will be done in a filter bank algorithm very similar to standard wavelet encoding.

Optimally smooth point sets. For a curve it is well known, that optimal smoothness is achieved, if the bending energy $E = \int \kappa^2(s)ds$ is minimized. Here, κ describes the curvature of the curves and s the so-called arc length parameter. Since this variational problem cannot be solved explicitly, it is common strategy to linearize the problem. It is well known that under certain boundary conditions, a curve minimizes the linearized bending energy if and only if it is a piecewise cubic curve. Such a curve is characterized by the property that the fourth derivative vector vanishes throughout the parameter domain. In case of point sets we now simply replace the fourth derivative vector with the so-called discrete fourth derivative vector.[21] It is then easy to see that a set of five points are optimally smooth, if they satisfy a simple geometric relation shown in Figure 3. This property has been used in[21] as basis of a linear fairing algorithm.

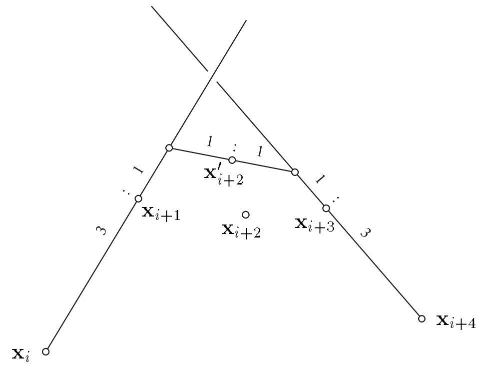


Figure 3. Optimal point $\mathbf{x}'_{i+2}$ with vanishing fourth discrete derivative vector.

Encoding Scheme. Let us consider a set of five consecutive points $\mathbf{x}_i, \ldots, \mathbf{x}_{i+4}$. By exchanging the point $\mathbf{x}_{i+2}$ by $\mathbf{x}'_{i+2}$ and deleting the points $\mathbf{x}_{i+1}$ and $\mathbf{x}_{i+3}$, we obtain a first level approximation $\mathbf{x}_i, \mathbf{x}'_{i+2}, \mathbf{x}_{i+4}$. It is easy to see that the original set of points can be computed from this approximation by adding the third discrete and fourth discrete derivative vectors

$$\begin{aligned}
\mathbf{x}_i^{(3)} &= \mathbf{x}_{i+3} - 3\mathbf{x}_{i+2} + 3\mathbf{x}_{i+1} - \mathbf{x}_i \\
\mathbf{x}_i^{(4)} &= \mathbf{x}_{i+1}^{(3)} - \mathbf{x}_i^{(3)}
\end{aligned}$$

as detail information. In matrix notation we obtain

$$\begin{pmatrix} \mathbf{x}_i \\ \mathbf{x}'_{i+2} \\ \mathbf{x}_{i+4} \\ \mathbf{x}_i^{(3)} \\ \mathbf{x}_i^{(4)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -\frac{1}{6} & \frac{4}{6} & 0 & \frac{4}{6} & -\frac{1}{6} \\ 0 & 0 & 0 & 0 & 1 \\ -1 & 3 & -3 & 1 & 0 \\ 1 & -4 & 6 & -4 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{x}_i \\ \mathbf{x}_{i+1} \\ \mathbf{x}_{i+2} \\ \mathbf{x}_{i+3} \\ \mathbf{x}_{i+4} \end{pmatrix}$$

Since we can assume that the point set is smooth, the fourth discrete derivative vector which essentially describes the deviation of the point set to the 'optimal' point set will be very small. In fact, if the model is sampled from a cubic TP b-spline surface, such as used by a CAD or animation software package, the fourth discrete derivative vector will vanish in most cases. Similarily, the third discrete derivative vector essentially describes the deviation from a parabola and will therefore vanish if quadratic b-splines are used. This motivates that the above algorithm will transform the original representation of the mesh in a representation where the detail vectors are very small or vanish completely.

This leads to an encoding procedure very similar to standard wavelet encoding. The only difference is that here five points are encoded in three new points with two detail vectors. In case of quadrilateral meshes it is possible to apply this procedure in the two different parameter directions. Finally, in order to achieve lossy compression one simply has to set small detail vectors equal to zero. Although very simple, this procedure turns out to give surprisingly good results. As an example, figure 4 shows two versions of the same mesh consisting of 4,225 points. The left image shows the original mesh, the right image shows the mesh after deleting 96% of its detail vectors. The total file size necessary to store this data is about 500 bytes which corresponds to about 1 bit per point. Furthermore note that the algorithm preserves boundary points and is therefore suitable to compress patchworks or closed surface meshes.

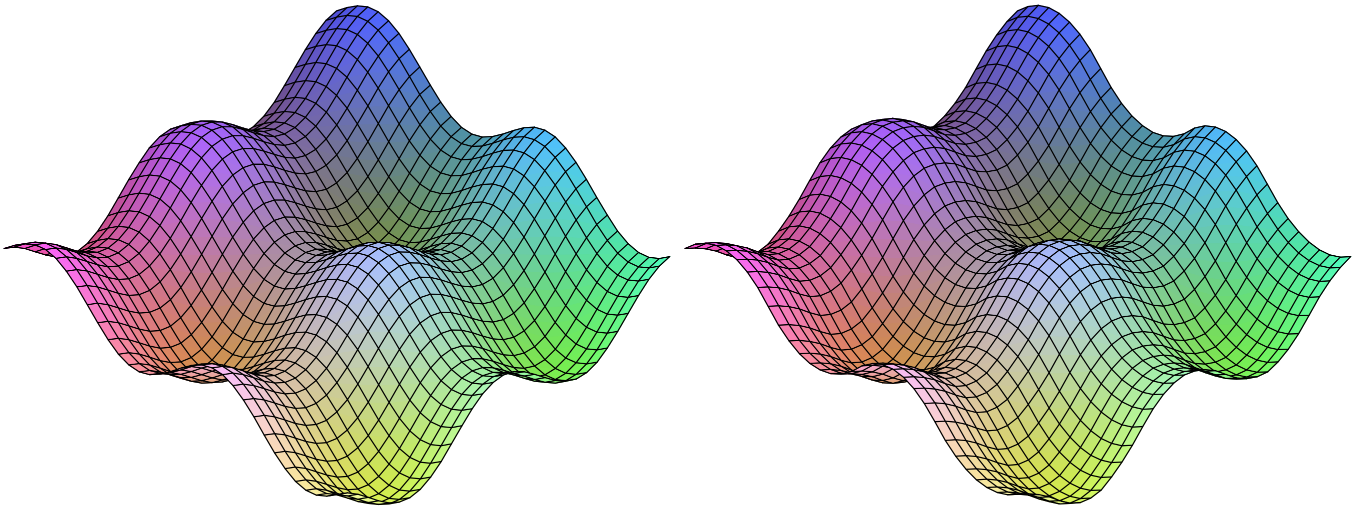


Figure 4. Two versions of the same mesh consisting of 4,255 points. The right version shows the mesh after applying a lossy high compression scheme. The amount of data needed was about 500 bytes which corresponds to approximately 1 bit per point. The error is visually almost neglectable.

Although this extremely good result is partly due to the special structure of a quadrilateral mesh, it indicates that there exist possibilities to reduce the file size of a polygonal mesh significantly without reducing the number of mesh points.

Approaching the magical number. During our tests it turned out that the minimum number of bytes necessary to store a quadrilateral mesh without loosing visual information, does not as much depend on the mesh itself but on the underlying surface geometry. For example, figure 5 shows the same surface sampled in two different resolution. In both cases, the minimum amount of data necessary to describe the visual information was about the same.

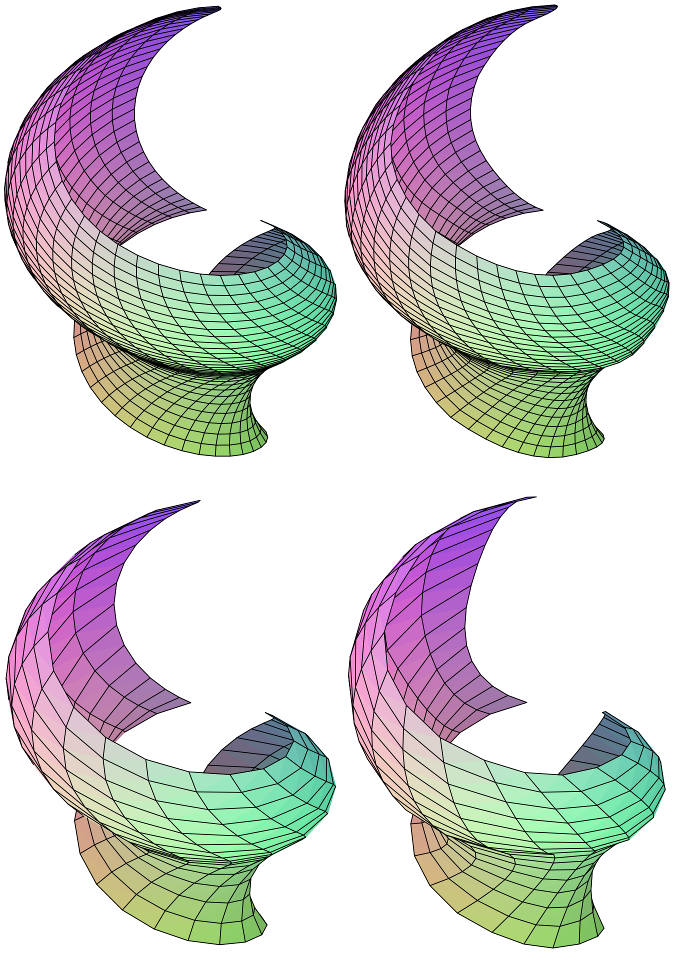


Figure 5. Two versions of the same surface at different sampling rates. In both cases, the compressed version to the right needed the same amount of data which was about 700 bytes. This indicates that there is a certain measurement for surface entropy which describes the minimum amount of data necessary to describe the shape of a surface independent from its sampling rate.

This opens the question whether there is a magical number of bytes necessary to describe a surface independent from how the surface has been sampled into a mesh. However, it is very unlikely that such a surface entropy measure can be obtained easily. Its definition is subject of future research.

5.2. Lossy compression of animated polygonal meshes

Once it is possible to reduce the size of an individual frame significantly, making the transition to animated polygonal meshes is straightforward. In this case we take an approach similar to standard video encoding. Instead of transmitting each frame individually, a sequence of key frames, also referred to as I-frames, will be chosen. In order to estimate the intermediate frames we then use the velocity and acceleration information of the points which can be obtained from the point trajectories using standard velocity estimation methods in computer aided geometric design (e.g., see Ref. 22). Similarly to the original mesh, the set of velocity and acceleration vectors form a quadrilateral mesh which can be encoded in the very same way. Finally, we apply cubic or quintic Hermite interpolation (e.g., see Ref. 22) in the decoding procedure in order to obtain a parameterized point trajectory for each point of the mesh. These point trajectories can then be easily sampled into the appropriate number of frames.

Note that, since, unlike videos, the view point and direction is not known in advance, object motions have to be described in a view independent manner. However, the view point and direction information can be used in choosing which portion of the motion information to be sent to a particular user with which level of detail. Furthermore, if the users can alter the animation content (object locations and motion), motion vectors may need to be created or updated in real-time, making the problem harder than that of video.

6. CONCLUSION

In this paper, we introduced the concepts of storage and transmission of animated 3D meshes. We, first, compared it with the problem of storing and transmitting video, and we showed the differences between these two problems. We then introduced animation, user, and server models, which we use for handling prioritized multi-layered streaming and prefetching. We then introduced the concept of lossy compression of animated meshes and we described how 3D scenes can be compressed in a layered fashion suitable for adaptive storage and transmission of animations. The proposed techniques are part of a research for defining a 3D streamed animation standard.

REFERENCES

1. *VRML International Standard* ISO/IEC 14772-1:1997, http://www.vrml.org/Specifications/.
2. *MPEG4 Standard* (Tokyo version), http://drogo.cselt.stet.it/mpeg/public/w2196.htm.
3. P. Venkat Rangan and H.M. Vin, "Efficient Storage Techniques for Digital Continuous Multimedia," *IEEE Transactions on Knowledge and Data Engineering,* vol. **5**, no. 4, pp. 564-573, August 1993.
4. H.J. Chen, A. Krishnamurthy, T.D.C. Little, and D. Venkatesh, "A Scalable Video-on-Demand Service for the Provision of VCR-like Functions," *Proc. IEEE Intl. Conf. on Multimedia Computing Systems,* pp. 65-72, 1995.
5. M.-S.Chen, D.D. Kandlur, and P.S. Yu, "Support for Fully Interactive Playout in a Disk-Array-Based Video Server," *Proc. ACM Multimedia'94*, pp. 391-398, 1994.
6. J.K. Dey-Sircar, J.D. Salehi, J.F. Kurose and D. Towsley, "Providing VCR Capabilities in Large-scale Video Servers," *Proc. ACM Multimedia'94*, pp. 25-32, 1994.
7. D.J. Makarof and R.T. Ng, "Schemes for Implementing Buffer Sharing in Continuous-media Systems," *Information Systems*, vol. **19**, no. 4, pp. 33-54, 1994.
8. R.T. Ng and J. Yang, "An Analysis of Buffer Sharing and Prefetching Techniques for Multimedia Systems," *ACM-Springer Multimedia Systems Journal*, 1995.
9. K.S. Candan, E. Hwang, and V.S. Subrahmanian, "An Event-Based Model for Continuous Media Data on Heterogeneous Disk Servers," Accepted for publication in *ACM-Springer Multimedia Systems Journal,* 1998.
10. K.S. Candan, E. Lemar, and V.S. Subrahmanian, "Interactive Result Visualization and View Management in Multimedia Databases," Submitted to the Special Issue of the *VLDB Journal* on Multimedia Systems, 1998.
11. K.S. Candan, E. Lemar, and V.S. Subrahmanian, "Management and Rendering of Multimedia Views," Accepted for publication at the *Fourth International Workshop on Multimedia Information Systems*, 1998.
12. W.J. Schroeder, J.A. Zarge, and W.E. Lorensen, "Decimation of Triangle Meshes," *Computer Graphics (SIGGRAPH 92 proceedings)* pp. 65–70,1992
13. H. Hoppe, "Progressive Meshes," *Computer Graphics (SIGGRAPH 96 proceedings)*, pp. 99–108, 1996.
14. J. Popovic and H. Hoppe, "Progressive Simplicial Complexes," *Computer Graphics (SIGGRAPH 97 proceedings)*, pp. 217-224, 1997.
15. L. Kobbelt, S. Campagna, and H-P. Seidel, "A General Framework for Mesh Decimation," to appear in *Graphics Interface 98 Proceedings*, 1998.
16. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle, "Multiresolution Analysis of Arbitrary Meshes," *Computer Graphics (SIGGRAPH 95 proceedings)*, pp. 173–182, 1995.
17. M. Lounsbery, T. DeRose, and J. Warren, "Multiresolution Analysis for Surfaces of Arbitrary Topological Type,"*ACM Transactions on Graphics,* **16**, pp. 34–73, 1997.
18. E. Stollnitz, T. DeRose, and D. Salesin, *Wavelets for Computer Graphics*, Morgan Kaufmann Publishers, 1996.
19. W.J. Schroeder, "A Topology Modifying Progressive Decimation Algorithm," *IEEE Proceedings Visualization 97*, pp. 205–212, 1997.
20. H. Hoppe, "View–Dependent Refinement of Progressive Meshes," *Computer Graphics (SIGGRAPH 97 proceedings)*, pp. 189–198, 1997.
21. M.G. Wagner, "Affine Invariant Fairing of Point Sets," in *Proceedings of the CISST '98,* Las Vegas, NV, pp. 370–374. July 6–9, 1998.
22. G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*, 4th ed., Academic Press, Orlando 1992.