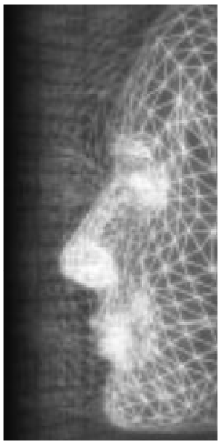
COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2006; **17:** 127–140
Published online 8 February 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cav.112

Implementation of 3D mesh streaming and compression techniques in NVEs



By Christos J. Bouras, Alexandros Panagopoulos and Thrasyvoulos Tsiatsos*

In this paper, we present a framework that integrates three-dimensional (3D) mesh streaming and compression techniques and algorithms into our EVE-II networked virtual environments (NVEs) platform, in order to offer support for large-scale environments as well as highly complex world geometry. This framework allows the partial and progressive transmission of 3D worlds as well as of separate meshes, achieving reduced waiting times for the end-user and improved network utilization. We also present a 3D mesh compression method focused on network communication, which is designed to support progressive mesh transmission, offering a fast and effective means of reducing the storage and transmission needs for geometrical data. This method is integrated in the above framework and utilizes prediction to achieve efficient lossy compression of 3D geometry. Copyright © 2006 John Wiley & Sons, Ltd.

Received: 24 March 2005; Accepted: 11 August 2005

KEY WORDS: 3D streaming; spatial partitioning; compression; networked virtual environments; high performance 3D graphics for distributed environments

Introduction

Networked virtual environments (NVEs) are one of the most promising uses of the virtual reality technology. Using NVEs as communication media, we can offer to members of virtual communities the advantage of creating proximity and social presence, thereby making participants aware of the communication and interaction processes with others.[1] However, three-dimensional NVEs can be very demanding in computational power and network communication. A server can host a large number of NVEs, consisting of virtual worlds that may be vast, geometrically complex and can be visited by a large number of users simultaneously. These virtual worlds can integrate many types of data such as 3D graphics, animated 3D models, multimedia (audio and video), text etc., which should all be shared among the clients. Thus, it is obvious that the size of data transmitted through the network is enormous. The amount of data increases according to the number of connected users and also according to the quality of the mesh representations used.

In the case of a large-scale environment, a user who connects via a low-bandwidth connection would have to wait excessively, until all data have been downloaded from the servers that host the virtual world, in order to start navigating the environment. Apart from the initial downloading, in a large-scale NVE, a large number of users interact with the environment and with each other. Each movement or interaction causes an event to be sent via multicast over the network, to all other users of the same NVE. These events may be large in size (e.g., when a user inserts a new 3D model in the world) or small but sent very frequently (such as the events related to the movement of the users' avatars). It is obvious that, unless the proper measures are taken, the network load could become excessive, resulting in resource starvation and excessive delays for the end-user. A number of methods can be employed, not only to reduce the amount of data sent to the clients, but also to reduce the delays observed for the same amount of transmitted data. Our goal is to provide large-scale environment support for EVE-II NVEs platform[2]

*Correspondence to: C. J. Bouras, Research Academic Computer Technology Institute, Greece; Computer Engineering and Information Department, University of Patras, Greece.
E-mail: bouras@cti.gr

[http://ouranos.ceid.upatras.gr/vr/], and improve its network behavior and user experience. EVE-II is an integrated platform for NVEs that provides: (a) high level of presence of the users in 3D multiusers virtual environments, (b) multimodal user-to-user interaction via chat, voice communication, and gestures. EVE-II can support VRML based 3D multiuser environments offering transparent sharing of objects, text- and voice-chat communication and a scalable, open architecture.[3] The implementation of the platform is mainly based on Web3D technologies such as VRML, VRML External Authoring Interface (VRML-EAI, www.web3d.org), Java, and ITU-T H.323 (www.itu.int). More information about EVE-II platform is presented at the third section of this paper.

Main objectives in order to upgrade our platform are: to reduce the amount of data transmitted over the network; to support large-scale virtual environments (LSVEs); to support immediate access of the users to the virtual environment over the network; and to make graphics quality adapt to the end-user's personal computer and network connection.

Main improvements in the EVE-II platform in order to achieve our goals are the following:

- *Streaming of virtual worlds and individual objects*: Using streaming, a user has to download only the visible portion of a world before he can interact with it. Also, streaming of individual objects allows users to view and interact with complex geometrical objects before they are fully downloaded. This dramatically reduces waiting times when a client enters an NVE.
- *Compression of the geometric data*: The usage of compression greatly reduces the amount of data that have to be transmitted to describe an object, resulting in the reduction of network traffic and downloading times. At the same time, compression should be integrated with streaming, to allow the progressive downloading of 3D objects.

This article is focused on the exploitation of well-known techniques in order to achieve streaming of 3D objects and to reduce the size of the data transferred over the network, using lossy compression of geometry. Our work has been based on the work presented in References [4,5].

The original contribution of our work is the development of a progressive transmission framework for the transmission of geometrical data over the network, using streaming and compression in order to support highly dynamic environments of arbitrary complexity, and should be able to make NVE applications usable over low-bandwidth connections and accessible through clients with low computational power. Another contribution is the integration of known solutions of sub-problems into a complete, working system. Furthermore, the experimental results, presented later in this article, confirm that (a) the progressive encoding allows the end user to view the models, at a lower image quality, before the models are completely streamed; (b) encoding only the error from the prediction yields a good compression ratio.

We initially (second section 'Related Work') present the previous work on the subject and, in the third section 'EVE-II Networked Virtual Environments Platform', general information about the EVE-II platform, which is the testbed for our work. In the fourth section ('Our Approach'), we present briefly the rationale of our approach. Afterwards, the algorithms used and the experimental results for 3D data streaming and spatial partitioning (fifth section '3D Data Streaming and Virtual World Partitioning'), as well as the implementation of 3D data compression (sixth section '3D Data Compression') are presented. Next, we describe our vision for the next steps of EVE-II. Finally, we present some concluding remarks.

Related Work

Much research work has been done on 3D data streaming and spatial partitioning as well as on 3D data compression. A conclusion is presented in the following paragraphs.

Concerning 3D data streaming and spatial partitioning, there is a variety of well-known spatial partitioning techniques, such as grid (division of the virtual world in horizontal rectangular areas); quad-tree (division of the virtual world in *x*–*z* level using quaternary trees), octree (extension of quad-tree in the *x*–*y*–*z* level, division of the virtual world in cubes); kd-tree; BSP-tree; cells and portals. In case of a multiuser virtual world each technique is integrated with specific techniques (such as filtering) for better manipulation of the multi-user data and events transmitted to the user.[6] 3D data streaming is generally implemented by producing a progressive representation of mesh data and then transmitting them through the network. A significant number of automated methods for producing progressive representations of triangular meshes exist, which rely on edge collapses or on clustering neighboring vertices, which is equivalent to applying several edge collapses in sequence. Methods differ in the particular strategy used for collapsing edges.[7] The most common representation used for the progressive loading and transmission of 3D objects is called *progressive meshes* and was introduced by Hoppe.[4] A progressive mesh representation consists of a base mesh and a sequence of vertex split operations that refine it. This permits progressive loading and transmission and view-independent refinement. Other significant methods (such as vertex clustering for manifold or non-manifold meshes) are presented in Reference [7]. Furthermore, Isenburg and Lindstrom[8] have introduced a streaming format for polygon meshes that is simple enough to replace current offline mesh formats and is suitable for representing large data sets.

Concerning 3D data compression, the increasing popularity of web-based applications, compression and streaming techniques are today more important than ever.[9] Mesh compression algorithms can be divided in single-rate and progressive mesh coding. Since single-rate algorithms allow the display of the mesh only after all data have been transmitted, only progressive coding is of interest in our case. Significant work has been done on compression of both connectivity and geometry. Regarding geometry compression, quantization,[10] prediction and delta coding have been used to achieve high compression rates. Early work employed delta coding or linear prediction along a vertex ordering dictated by connectivity data after quantization,[11] while MPEG-4 uses the Parallelogram Rule.[5] Lee *et al.*[12] proposed the usage of prediction and quantization in angle space after prediction, while Isenburg and Alliez[13] generalized the methods to polygon meshes. A drawback of the above methods was that vertex order was dictated by mesh connectivity, which is not optimal. An attempt to solve this problem was the usage of prediction trees.[14] Furthermore, Isenburg *et al.*[15] have presented a method for compressing floating-point coordinates with predictive coding in a completely lossless manner. This method omits the initial quantization step and the predictions are calculated in floating-point. Also, several methods based on wavelet transformations have been proposed, achieving very good results.

Several algorithms have been proposed to compress connectivity as well. Hoppe[4] introduced the progressive mesh technique, which could be used to compress connectivity. Pajarola and Rosignac[16] grouped edge collapses and used a two-coloring of the mesh vertices, resulting in 3 bits/vertex. Alliez and Desbrun[17] and Karni *et al.*[18] improved the previous approaches further. However, this paper does not address connectivity compression, so these results will not be further examined.

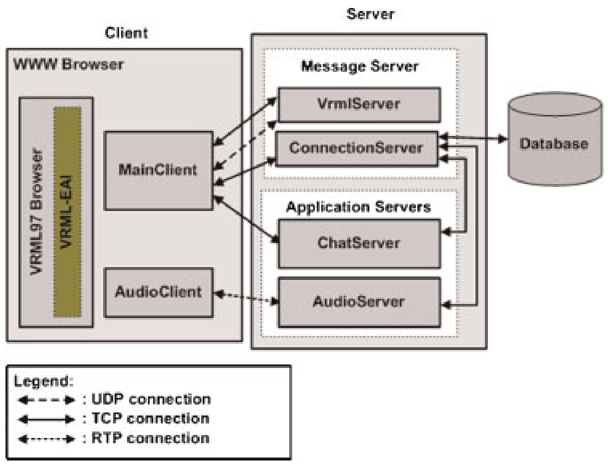


Figure 1. Architecture of EVE-II.

EVE-II Networked Virtual Environments Platform

EVE-II is a NVEs platform, which is used as testbed for the work presented in this paper. Its architecture (Figure 1) is based on a client–multiserver platform model. The current form of EVE-II constitutes an open and flexible architecture. The servers, on which the platform relies, are the message server and two application servers, a chat and an audio server. This model offers scalability and flexibility to the EVE-II architecture, because we can add more application servers in order to offer more functionality and furthermore the processing load is distributed among the above set of servers.

EVE-II is based on open technologies (mainly web 3D technologies) and international standards. More specifically the implementation of the platform is mainly based on:

- VRML, for the representation of the 3D worlds and for describing 3D objects.
- VRML external authoring interface (VRML-EAI), for implementing an interface between the 3D worlds and external tools.
- Java, for the development of the client–server model, and the network communication among the different components of our platform.
- H.323, for offering audio conferencing services over the internet.

Concerning the sharing of multiuser events, EVE-II goes beyond other platforms, offering very simple and almost transparent sharing of all types of events, while it hardly imposes any limitations to the way virtual worlds should be implemented in terms of VRML code.

Our Approach

This section presents the approach we adopted in order to implement streaming and use compression to reduce network traffic in our NVE platform.

As it is mentioned in the introductory part of this paper, a virtual world consists of a large number of objects and scripts, which describe the geometry, textures, special features, and interactivity of the environment. Virtual environments can be rather large and complicated, demanding the transmission of an equally large amount of data. This makes it hard for users to connect to the virtual environment over low-bandwidth connections, such as dial-up. Unless further measures are taken, the user would have to wait several minutes for the downloading of data to complete, before he/she can enter a regular 3D environment.

The way to deal with these problems is to reduce the amount of data that need to be sent to the client using compression, and to transmit this data progressively. Using progressive transmission, the user does not have to wait until the whole 3D environment is transmitted. As long as the important data that lie in the area the user perceives have been downloaded, the user can start navigating the world and interact with objects. As transmission continues, a larger portion of the world becomes accessible to the user, while complicated meshes are displayed with increasing detail.

Primary goal of this paper is to provide a framework and methods to assist the transmission of geometrical data over the network, using streaming and compression. These methods target highly dynamic environments of arbitrary complexity, and should be able to make NVE applications usable over low-bandwidth connections, such as dial-up. Furthermore, the computational overhead on both the server and the client should be low, since servers may support a large number of virtual worlds and users, while clients' machines may be low-end. The methods used target virtual environment applications that:

- consist of both high- and low-complexity 3D meshes,
- should be able to function adequately over low-bandwidth connections,
- are highly dynamic, and
- Do not demand high processing power neither from the server nor from the client side

In the following sections, our solutions will be presented.

3D Data Streaming and Virtual World Partitioning

Using streaming for the transmission of 3D objects, we can support quick users' connection in a NVE as well as immediate interaction with shared virtual objects, without requiring the downloading of the whole 3D scene and the geometrical data. In order to support large-scale virtual environments, EVE-II partitions the virtual world and employs a selective transmission mechanism, to send to the clients only the data of 3D objects that are close to them, and defer the transmission of objects lying far away from the user. Only 3D geometry is subject to partitioning and deferred transmission. Nodes such as transformations, scripts etc. must all be transmitted before the virtual environment is displayed to the user. The application of both 3D data streaming and spatial partitioning techniques in multiuser virtual environments helps the reduction of network traffic and is crucial in order to offer good scalability, as the size of the virtual environments grows. In the following sections, the algorithms applied in EVE-II platform and the experimental results concerning 3D data streaming and 3D world partitioning are presented.

Implementation of 3D Data Streaming in EVE-II Platform

In order to implement 3D Data streaming, EVE-II: (a) exploits spatial partitioning by adopting a combination of octree partitioning with cells and portals, (b) introduces a progressive transmission framework, and (c) uses progressive mesh representation. These solutions are presented in the following sections in detail.

Space Partitioning. EVE-II is intended to support every kind of virtual world, including both indoor and outdoor environments. These environments could be of a very large size, so space partitioning is necessary to handle them. The solution, we adopted is a combination of octree partitioning with cells and portals.

Every closed room in a virtual environment is considered as a cell. The openings that connect a cell to its adjacent cells are called portals. The outdoor environment is considered as a single cell with some special properties. This partitioning of virtual space into cells is extremely suitable for indoor environments, and that is the reason why it is very common in games and other real-time 3D applications. The user lies in exactly one cell. The space that is visible for him/her consists of the cells that are connected to the cell the user is in with 'open' and visible portals. That way, it is quite straightforward to determine the data that are necessary to be sent to a user at a given time, and also the data that might probably be needed in the near future.

The cells and the portals have to be defined by the creator of the virtual world, and their use is optional. A cell is defined only by the nodes it contains. A portal is defined by a bounding box covering the area of the corresponding 'opening,' the cells it connects and a boolean property indicating whether it is open or closed. When a user opens, for example, a door, the corresponding portal becomes open, and the cell on the other side of the door visible. So, if the data have not yet been sent to the client, they are sent urgently.

A cell may cover a large portion of virtual space, containing a lot of objects. The special cell that is used for the whole outdoor environment (and all nodes which do not belong to a particular cell) covers the whole extent of the virtual world, and consists of a vast number of objects in a large-scale environment. It is obvious that large cells have to be further subdivided. This space subdivision is implemented via a simple octree structure. The octree provides a tree structure that partitions space, and allows for efficient spatial queries. This way, it is fast to determine which nodes are close to the user, and which are not.

An octree structure is based on the recursive partitioning of box-shaped areas. The root of the tree includes the whole virtual world. Each node is subdivided to eight child nodes, each one covering a box-shaped area with one-eighth of the volume of the parent. Node subdivision stops when a node is encountered, which includes fewer objects than a certain threshold. Spatial queries can be answered by traversing this tree, which is very efficient.

We define an area of interest for each user. This area of interest is a simple sphere around the user, with a fixed radius. All nodes of the octree that lie into this area are considered important for the user's perception of the world, and have to be sent to the client as soon as possible.

Progressive Transmission Framework. We have implemented a progressive transmission framework, which includes:

- One or more data pools for deferred data.
- Priority calculation for each node, depending on its position, size, and internal attributes.
- Different classes for deferred data, relative to their importance.
- Support for streaming of individual node data.
- Automatic updates, when the user moves, rotates, or when new objects are added or removed from the world.
- Transmission of deferred data parallel to the normal event traffic.

Progressive transmission is used (for the time being) only for VRML nodes of type *IndexedFaceSet*. These nodes are the ones that contain the geometry of 3D meshes, and generally account for the 80–90% of the total data of a virtual world. Furthermore, these nodes do not include any functionality. So, as soon as the rest of the world is available to the user, the latter has access to the full functionality of the virtual environment. Mesh data are transmitted progressively only for those nodes that lie inside the area of interest. The rest of the mesh data is transferred only when the client's connection is idle, since these data do not correspond to areas visible to the user.

This framework is able to accommodate all transmission needs of the platform. A deferred data pool is created for each virtual world. When the world is sent over the network, all mesh data whose transmission can be delayed are put into the deferred data pool, instead of being transferred immediately. The rest of the data is sent to the client, and the normal operation of the platform begins, before any of the meshes has been sent.

For each node put into the deferred data pool, the server calculates a priority value and an importance class. Priority is based on the node's position and size, as well as on internal node data. For example, internal node data taken into account when a progressive mesh is streamed through the network are the portion of data already sent, and the error value associated with the next vertex split. Along with the position, some of the data are marked as 'urgent'. These data lie inside the area of interest for a specific user, and have to be sent as soon as possible, because they are considered important for the user's experience. Priorities are recalculated often, so as to adopt when the user moves or new objects are added or removed. When an object in the deferred data pool is to be transmitted, it is asked to send the next block of its data. Objects that cannot be streamed will, at this point, send all of their data and be removed from the deferred data pool. Streamable objects will send only a small block of data, as asked, alter their priority and then wait for their turn to send data again. This way, fairness is guaranteed among the different nodes that rival for transmission. When all data of a node have been sent, this node is removed from the data pool. The transmission of deferred data is carried out in parallel with regular event traffic, generated by the interaction between users and the virtual environment. Events are given higher priority than deferred data. However, data that are considered as 'urgent' will never wait more than a fixed, small amount of time. Data that are outside the area of interest, on the other hand, will only be transmitted when the client's connection remains idle. This strategy balances the traffic generated by user interactions and the one generated by deferred data transmissions, and achieves optimal usage of the client's connection.

As already mentioned, the only nodes that are subject to delayed transmission are, for the time being, the *IndexedFaceSet* nodes, which contain 3D mesh data. Mesh data, which are sent in parts, include vertex positions, normals, vertex colors, texture coordinates, and indices. All of these data are streamed, if a progressive mesh is used to represent the *IndexedFaceSet*, or sent as a whole otherwise.

The pseudo-code, which shows the program logic is the following:

```
repeat_forever
{
  for_each node in the deferred data pool
  {
     node.updatePriority();
  }
  sortNodes(data_pool);
  Node node = data_pool.getTopNode();
  if (node.priorityClass = = URGENT ||
    event_pool.getPendingEventsNum() = = 0 ||
    getCurrentTime() - node.lastTransmissionTime > TIMEOUT)
  {
     node.transmitNextDataBlock();
  }
  else if (event_pool.getPendingEventsNum() > 0)
  {
     Event event = event_pool.getNext Event();
     transmitEvent(event);
  }
}
```

Mesh Representation. For the representation of the 3D data, we use the progressive mesh[4] representation, a mesh format that allows the generation of different levels-of-detail, the progressive transmission of the 3D data, and will also provide the base to develop a compression scheme for the meshes. In progressive mesh form, an arbitrary mesh M is represented by a coarse base mesh M^0 and a sequence of n refinement operations. Each of these refinement operations encodes the data associated with a vertex split operation, an elementary transformation that adds one vertex to the mesh. The sequence of these refinement operations indicates how to refine the coarse mesh M^0 into a mesh $M^n = M$. Applying each refinement operation sequentially, we obtain on each step a new approximation of the original mesh M. The sequence of approximations $M^0 \ldots M^n$ represents n different levels-of-detail of the original mesh, with the last approximation M^n being exactly the same as the original mesh M.

Streaming transmission of this mesh representation is quite straightforward. Only the coarse base mesh M^0 has to be sent as a whole, before the mesh can be rendered. Afterwards, refinement operations are sent sequentially, with each one adding a new vertex to the mesh shown to the user. The transition between the sequential levels of detail is smooth enough, and can become smoother with the introduction of geomorphs.[4]

Only the transmission of the base mesh is considered urgent, if the object lies in the user's area of interest.

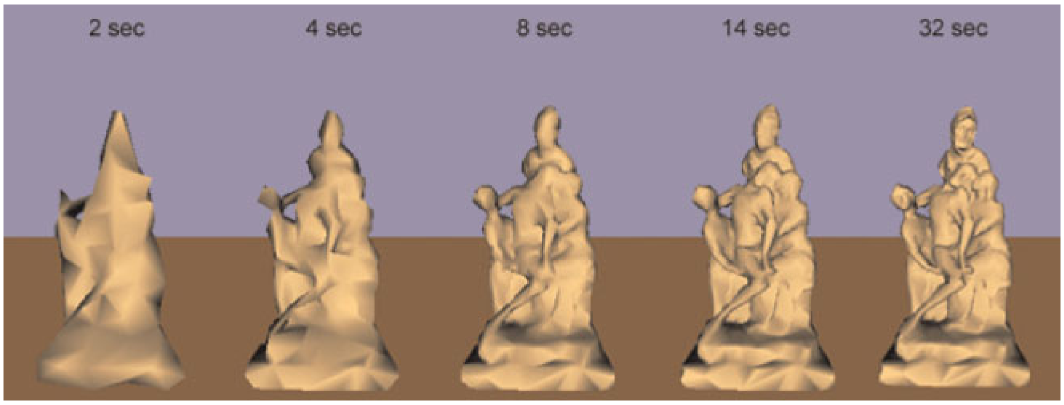


Figure 2. Downloading of pieta mesh.

Base mesh	LoD 10%	25%	50%	100%
2 s	4 s	8 s	14 s	32 s

Table 1. Download times for the pieta mesh

Experimental Results

In Figure 2, some steps in the transmission of a complex mesh can be seen. The user can see a coarse version of the mesh only a few seconds after the transmission of the world has started. The user-waiting time for an average virtual world (using a local area network connection) is 32 s without using progressive transmission and only 2 s when progressive transmission is used. The total download time though is approximately the same in both cases.

Download times for the pieta mesh (Figure 2) are listed in Table 1.

3D Data Compression

EVE-II targets to NVEs of high complexity, in order to provide a good experience to the users. Such environments typically take up several Mbytes to be stored. However, since the platform should be usable even over low-bandwidth dial-up connections, the transmission of such a large amount of data is not acceptable, and data size should be reduced as much as possible.

On the other hand, many of the objects that constitute a virtual environment usually have a low number of faces, since world creators are aware of the bandwidth limitations. Triangulation and accuracy are more important on these meshes. The compression used should keep the original triangulation of the objects, and respect mesh attributes. Furthermore, the compression should seamlessly integrate with the progressive transmission of meshes. Finally, it would be desired to be able to adjust the compression ratios according to the characteristics of the client's connection. Mesh data can be divided into two main groups: the vertex data and the index (connectivity) data. Vertex data include all vertex attributes, such as position, color, texture coordinates, normal vector etc. The indices, on the other hand, define the faces that constitute the mesh. Vertex data take up the 60–70% of the overall mesh size. For the time being, EVE-II supports the compression of only the vertex data and not of the indices. Compression of connectivity data (indices) will be added in the near future, following well-known methods (triangle strip techniques, topological surgery,[19] Cut Border,[20] method by Tauma and Gotsman[5]).

The method, we implemented utilizes the progressive mesh format, and tries to reduce the size of transmitted data by using prediction filters and quantization.

Some of the advantages of this solution are:

- It is inherently progressive.
- It works with arbitrary meshes.
- Its implementation is simple and flexible.
- It can be applied to all vertex attributes.
- There is no need for preprocessing (apart from the creation of the progressive mesh structure).

In the following sections, the algorithms applied in EVE-II platform and the experimental results regarding 3D data compression are presented.

Implementation of 3D Data Compression in EVE-II Platform

The compression method used to reduce the size of transmitted geometry in the EVE platform builds on the existing progressive mesh framework, keeping data streaming in mind.

For each vertex sent to refine a progressive mesh, a prediction algorithm is used to predict the expected value of each attribute (position, color, or texture coordinate) of this vertex. This prediction algorithm is based only on data already transmitted, so that it can be executed both on the server and the client, yielding the exact same results. The only data sent is a correction vector, representing the difference between the predicted and the actual value. This value is quantized and encoded. The magnitude of the correction vector is very small, which means that it can be highly quantized without significant loss of precision. When the difference between the predicted and the actual value is below a certain threshold, its transmission can be skipped. In this case, the client uses only the predicted value.

The tactics of using a prediction filter, quantizing and encoding the correction vectors, and omitting some of them can be used not only on vertex positions, but also on all other vertex attributes. In fact, the prediction is easier when applied on vertex colors or texture coordinates, and a larger portion of these data can be skipped when transmitting the mesh. Vertex normals can be wholly skipped, since they can be recalculated by the vertex data.

The data is sent progressively. We can treat each sequence of values of each vertex attribute as a separate data sequence. For each of these data sequences, we apply the appropriate prediction filter, omit values that are near zero and quantize and transmit the rest of them. Next we will examine the prediction filters used for each vertex attribute sequence, and the results we got.

We should note that the method used offers the advantage that it does not pre-calculate a compressed mesh representation at a given compression ratio. Instead, it reduces the amount of data when transmitting the mesh to the client, so that the compression ratio can be altered to adapt to the needs of each client.

Furthermore, the complexity added to the initial progressive mesh representation is minimal, and the additional computational cost for both the server and the client is almost negligible.

The whole 3D data compression scheme is depicted in Figure 3. The high-level pseudo-code describing how the program logic in the case of vertex coordinates is the following:

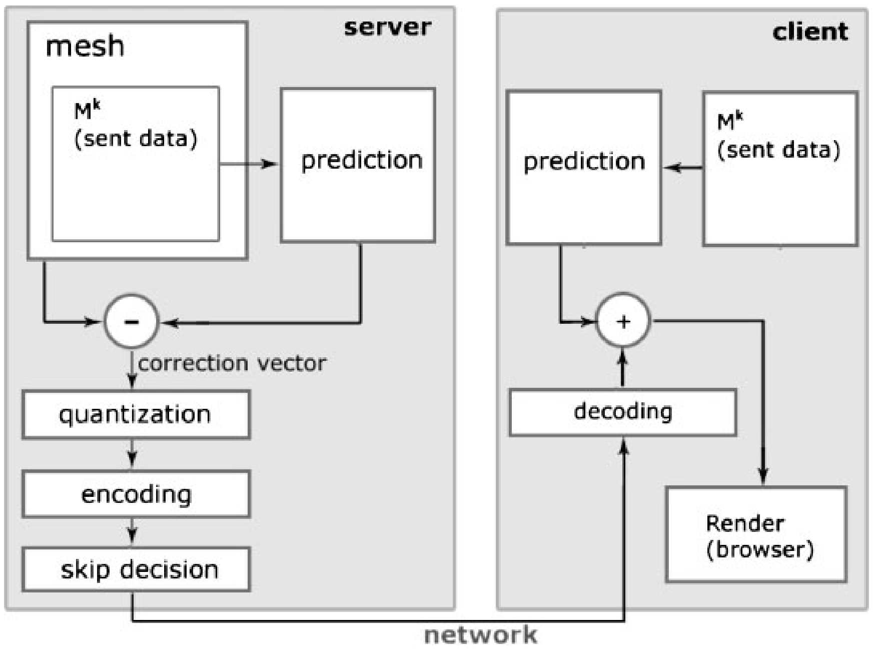


Figure 3. 3D data compression in EVE-II platform.

```
vsplit_transmit()    // server-side
{
  Vector3 pred_pos = predict(sent_mesh, cur_vertex);
  error = predictionErrorMetric(pred_pos, actual_pos);
  if (error > ERROR_THRESHOLD)
  {
    Vector3 pos_error = actual_pos - pred_pos;
    byte[ ] binData = quantizeAndEncodeVector(quant_pos_error);
    transmit_data(binData);
  }
  else
    markSkippedVertex(cur_vertex);
}
vsplit_receive()    // client-side
{
  Vector3 pred_pos = predict(received_mesh, cur_vertex);
  bool skip_flag = receiveSkipFlag();
  if (skip_flag = = true)
    return pred_pos;
  else
  {
    byte[ ] binData = receive_data();
    Vector3 pos_error = decodeVector(binData);
    return pred_pos + pos_error;
  }
}
```

In the following sections, the compression algorithm we used will be discussed in more detail.

Prediction. When a vertex split k is performed, available on the client side is only the topology of the M^k approximation of the original mesh M. We have to determine which will be the new edge that will be added to the mesh, which triangles shall be added, and which indices must be changed, in order to convert M^k to M^{k+1}. These data must be sent to the client, along with the values of all vertex attributes.

The new edge (v_1, v_2) to be created is determined by the id of the edge vertex that is already in M^k, v_2. So, the prediction algorithm will work on data that consists of the topology of M^k, the id of v_2, the indices of the triangles to be added and the indices to be changed on the existing index list.

To predict the new vertex's position, we need to know the indices that will change. The vertices that are referred by these indices are depicted in Figure 4(a) as bold (i.e., A_1, A_2, and A_3). Let A be the set of these vertices. Let also v_2 be the vertex of the edge to be added, which is already in the transmitted mesh.

The average of the positions of these vertices, v_{base}, is used as a base for the prediction of the new vertex position. v_{base} is actually the predicted position projected onto the plane of the nearest triangle.

$$v_{\text{base}} = \frac{1}{n}\sum_{i=0}^{n} \frac{v_{A,i} + v_2}{2} \tag{1}$$

In order to predict the vertex position, we need to take one more factor into account, which is surface curvature. We used the normal of the existing edge vertex, n_2, and the average of the normal vectors of vertices in A, n_{base}. These two normal vectors are used to 'project' v_{base} away from the mesh surface according to the following equation (k is a constant selected so that the prediction is accurate on spherical surfaces):

$$p = v_{\text{base}} + k(n_{\text{base}} + n_2)(1 - n_{\text{base}} \times n_2)|v_{\text{base}} - v_2| \tag{2}$$

The final predicted position p is the sum of v_{base} and a vector in the direction of the average of n_{base} and n_2, and length proportional to the distance between v_{base} and

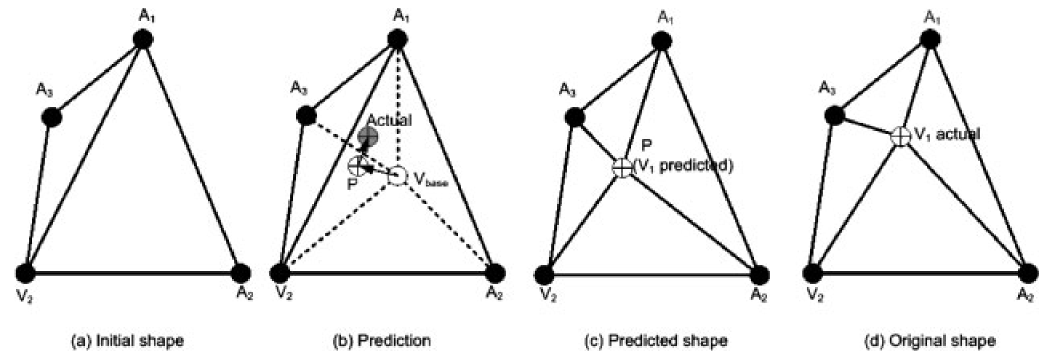


Figure 4. Prediction of a v_{split}.

v_2 and the angle formed between n_{base} and n_2. The predicted position p is shown in Figure 4(b). The prediction is usually close to the original vertex (Figure 4(c) depicts the predicted shape whereas Figure 4(d) depicts the original shape). In fact, the prediction equation is tailored to give optimal results on spherical surfaces—that is to give almost exact predictions of the actual positions. However, this proved to work satisfactory on most curved surfaces.

Each vertex has a number of attributes other than position, which should also be predicted. The most common of these attributes are texture coordinates, normal, and color. Vertex colors and texture coordinates are predicted using a simple interpolation between their respective values for v_1 and v_2. Since the values of these attributes do not usually exhibit large changes among neighboring vertices, this simple method proved to be quite successful. Vertex normals are not transmitted at all, since they can easily be recalculated by the vertex positions.

Quantization. The values predicted for the various vertex attributes may not be accurate, but are always near the original values. That way, the correction vectors have small magnitude, and can be highly quantized. We used 24 bits for the correction vector of vertex positions, 16 bits for texture coordinates, and 4 bits for each color component.

Encoding. After quantization, the numerical values are encoded, using simple Huffman encoding. Most of the numerical values are near zero, since the difference between the predicted and the actual values is usually small. This fact makes the application of Huffman encoding effective, further reducing the size of the data to be transmitted.

Skipping vertices. When the prediction is very close to the actual values, the transmission of the correction vector can be skipped, as already mentioned. For the decision of whether to transmit or skip a given vertex, we used an error metric, which takes into account the difference between the predicted and actual values, the area of the neighboring faces and the curvature of the surface at that point. As seen in the experimental results, the number of skipped vertices is usually large.

Experimental Results

We tested the compression algorithms described above with a variety of real-world models, with both high- and low-polygon counts. Our algorithm displayed good behavior, and very good results for all models. As shown in Table 2, a compression of 5–10 bits/vertex is always possible.

Mesh	No. of Vertices	Data size (KBytes)	Maximum visually acceptable compression	Compressed data size (KBytes)
Head (low-detail)	3322	318	7.5 bits/vertex	25
Head (mid-detail)	24 662	2367	5.3 bits/vertex	132
Head (high-detail)	42 660	4095	4.6 bits/vertex	196
Unicycle	6973	669	9.1 bits/vertex	63

Table 2. Experimental results

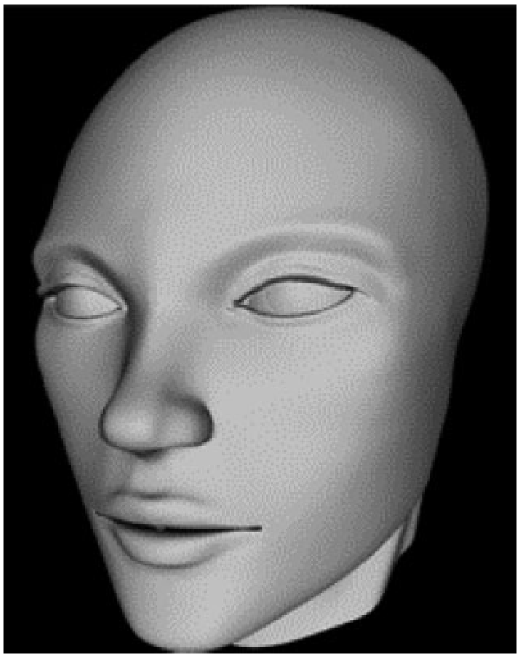


Figure 5. Original head model.

The first model we used to test our algorithm was a high-polygon-count model of a human head, shown in Figure 5.

As shown in Figure 6(d), when compressed at about 5.7 bits/vertex, the model seems almost identical to the prototype. Even at 4.2 bits/vertex (Figure 6(b)), the quality of the model is good, except from some high-frequency noise. At even higher compression rates, however, this noise becomes very obvious and disturbing. This can be seen at Figure 6(a), where the model is compressed at 3 bits/vertex.

The model of Figure 5 was a typical example of a high-polygon-count organic model, full of curved surfaces. This is a good case for most compression algorithms. In order to test our implementation against a bad read-world case, we used the model of a unicycle shown in Figure 7.

It can be seen from Figure 8 that our algorithm behaves very well even in this bad case. When compressed at 11 bits/vertex (Figure 8(b)), the unicycle model has very small distortions, and at 7.4 bits/vertex (Figure 8(a)) it is still visually acceptable. It can be noticed there are no specific features of the model that were excessively distorted or destroyed.

It should be noted that, when attempting to compress a mesh with a very low number of faces, no real compression could be done, since almost all vertex information is important. In this case, our algorithm reduces to a simple quantization of coordinates.

Further Improvements: Future Work

Generally speaking, the performance of EVE-II platform has been significantly improved. However, further work should be done in order to improve the networking behavior of the platform. The main improvements in order to reduce network traffic will be the following:

- *Exploitation of spatial partitioning for selective transmission of events*: We propose a better manipulation of the shared events in order to send to a specific user events that are referred only in nodes concerning objects inside his/her viewpoint. An area of influence will be defined for each node and checked versus the user's area of interest, as well as the area of interest of other nodes that receive events, in order to filter and reduce event traffic on the network.
- *Implementation of a caching mechanism*, which will cache objects and resources that are frequently downloaded by the user. This caching mechanism will reside on the client side, and will apply on geometry data as well as textures and other media content.

Furthermore, the mesh compression algorithm could be significantly improved. These improvements will not only affect the compression rates, but also the way compression is used to provide a better experience to the user. Some of our next steps towards this direction are the following:

- Compression of mesh connectivity, in the same way as vertices are compressed now. This is a necessary step, so that geometry compression in EVE-II can be considered complete. The existing framework and way of thinking will be used in this case as well, while we will exploit well-known methods based on triangle strips.
- Improvement of the prediction methods used for vertex positions, and, if necessary, addition of a post-processing step to remove the noise introduced by the compression.
- Categorization of clients in different classes, depending on their machine and connection speed: then, higher compression can be used for those clients that connect via a dial-up connection, and the progressive transmission of meshes can be stopped before the full-resolution mesh has been transmitted, when the

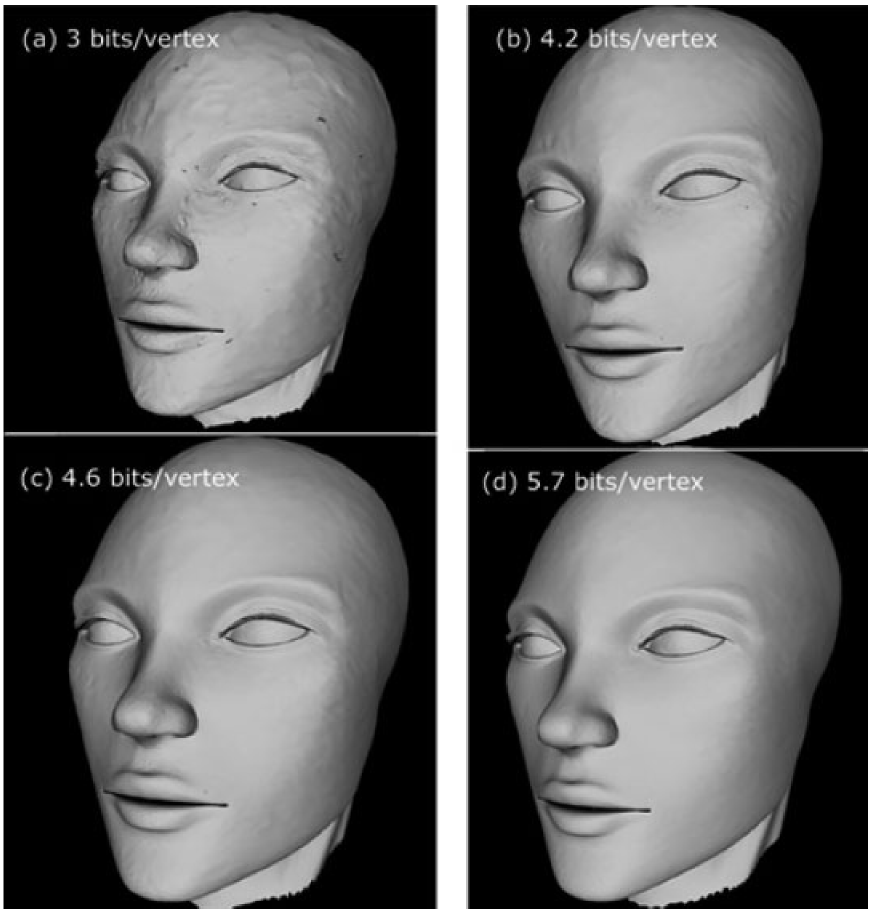


Figure 6. Head model, at various compression rates.

client's machine is not fast enough to display the world at acceptable frame rates. The introduction of such techniques will allow users who do not have access to fast equipment to use the platform, while privileged users will still be able to enjoy very good graphics quality.

Conclusions

This paper describes the techniques used in EVE-II in order to reduce the bandwidth requirements of the platform and the waiting times for users. A framework was described, which supports the streaming transmission of 3D objects, and the partial, progressive downloading of the virtual world. Also, a method used to compress 3D meshes was presented. The combination of the above methods significantly enhanced the usability and efficiency of EVE-II.

In particular, the framework for streaming and progressive downloading of the world enables the partial transmission of a virtual world, so that only objects that are inside the user's area of interest are sent to the client at first. That way, the initial download time is independent of the size of the environment, so that LSVEs are possible.

Furthermore, the streaming of 3D meshes allows a coarse representation of the mesh to be displayed to the user very fast, while downloading continues in the background. The meshes are refined, as download progresses, until the full-detail mesh is reached. That way, the user does not have to wait until the full-detail mesh has been received.

Also, an effective mesh compression method was presented, which highly compresses the vertex data and integrates seamlessly with the streaming framework. As shown, compression ratios of 10–30 to 1 are achievable.

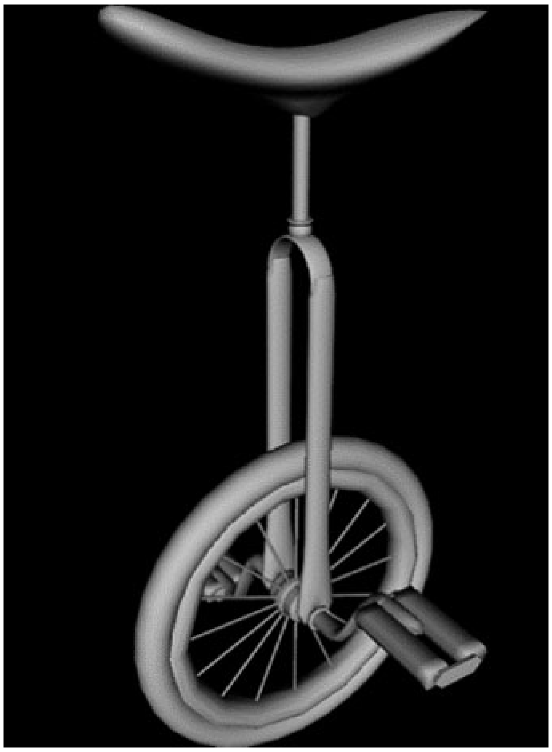


Figure 7. Original unicycle model.

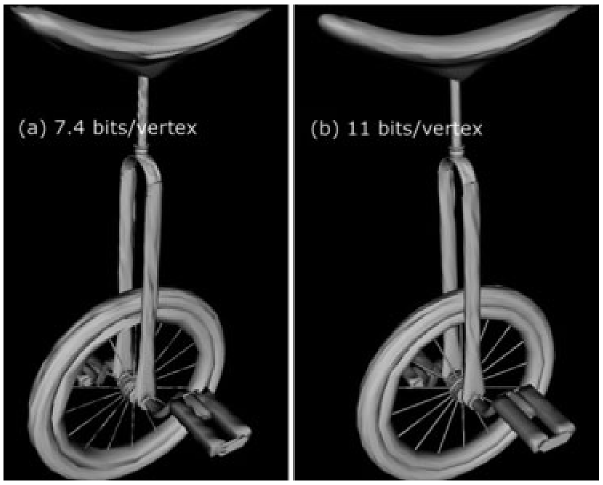


Figure 8. Unicycle model, at various compression rates.

The above additions to the platform substantially improved the network behavior of EVE-II, as well as the experience perceived by the final user. The times the latter has to wait in order to access a virtual world were dramatically decreased, while the reduction in the amount of transmitted data made possible the use of the platform even by users that connect via low-bandwidth lines. At the same time, the use of large-scale worlds is now feasible, expanding the possible applications of our platform.

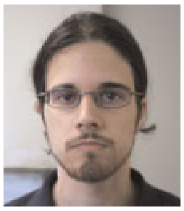
References

1. Bouras C, Tsiatsos T. Distributed virtual reality: building a multi-user layer for the EVE platform. *Journal of Network and Computer Applications (JNCA)* [Academic Press] 2004; **27**(2): 91–111.
2. Bouras C, Panagopoulos A, Theoharis N, Tsiatsos T. EVE-II—II: an integrated platform for networked virtual environments. In *Proceedings of the Tenth International Conference on Distributed Multimedia Systems*—DMS'2004. San Francisco, USA, 2004 September 8–10.
3. Bouras C, Tsiatsos T. Educational virtual environments: design rationale and architecture. *International Journal of Multimedia Tools and Applications (MTAP)*, 2006 (to appear).
4. Hoppe H. Progressive meshes. In *Proceedings of ACM SIGGRAPH 1996*, 1996, pp. 99–108.
5. Touma C, Gotsman C. Triangle mesh compression. *Graphics Interface 98 Conference Proceedings*, 1998, pp. 26–34.
6. Diehl S. 2001. *Distributed Virtual Worlds, Foundations and Implementation Techniques Using VRML, Java, and CORBA*. Springer-Verlag: Berlin Heidelberg, Germany, ISBN 3-540-67624-4.
7. Guéziec A, Taubin G, Horn B, Lazarus F. A framework for streaming geometry in VRML. *IEEE Computer Graphics and Applications* 1999; **19**(2): 68–78, ISSN:0272-1716.
8. Isenburg M, Lindstrom P. Streaming Meshes, Report UCRL-CONF-201992, Lawrence Livermore National Labs, 2005.
9. Cohen-Or D, Levin D, Remez O. Progressive compression of arbitrary triangular meshes. In *Proceedings of IEEE Visualization 99*, San Francisco, CA, 1999, pp. 67–72.
10. Sorkine O, Cohen-Or D, Toldeo S. High-pass quantization for mesh encoding. In *Proceedings of Eurographics Symposium on Geometry Processing*, 2003.
11. Deering M. Geometry compression. *ACM SIGGRAPH Conference Proceedings*, 1995, pp. 13–20.
12. Lee H, Alliez P, Desbrun M. Angle-analyzer: a triangle-quad mesh codec. In *Eurographics Conference Proceedings*, 2002, pp. 383–392.
13. Isenburg M, Alliez P. Compressing polygon mesh geometry with parallelogram prediction. In *IEEE Visualization Conference Proceedings*, 2002, pp. 141–146.
14. Kronrod B, Gotsman C. Optimized compression of triangle mesh geometry using prediction trees. In *Proceedings of 1st International Symposium on 3D Data Processing, Visualization and Transmission*, 2002, pp. 602–608.
15. Isenburg M, Lindstrom P, Snoeyink J. Lossless compression of redicted floating-point geometry. *Computer-Aided Design* 2005; **37**(8): 869–877.
16. Pajarola R, Rossignac J. Compressed Progressive Meshes. *IEEE Transactions on Visualization and Computer Graphics* 2000; **6**(1): 79–93.

17. Alliez P, Desbrun M. Progressive encoding for lossless transmission of 3D meshes. In *ACM SIGGRAPH Conference Proceedings*, 2001, pp. 198–205.
18. Karni Z, Bogomjakov A, Gotsman C. Efficient compression and rendering of multi-resolution meshes. In *IEEE Visualization Conference Proceedings*, 2002.
19. Taubin G, Rossignac J. 1996. Geometric compression through topological surgery. *ACM Transactions on Graphics* 1998; **17**(2), and IBM Technical Report RC-20340, January 1996.
20. Gumhold S, Strasser W. Real time compression of triangle mesh connectivity. In *SIGGRAPH'98 Conference Proceedings*, 1998, pp. 133–140.

Authors' biographies:

Christos J. Bouras, associate professor. Christos Bouras obtained his Diploma and Ph.D. from the Computer Science and Engineering Department of Patras University (Greece). He is currently an associate professor in the above department. Also he is a scientific advisor of Research Unit 6 in Research Academic Computer Technology Institute (CTI), Patras, Greece. His research interests include Analysis of Performance of Networking and Computer Systems, Computer Networks and Protocols, Telematics and New Services, QoS and Pricing for Networks and Services, e-learning, Networked Virtual Environments, and WWW Issues. He has extended professional experience in Design and Analysis of Networks, Protocols, Telematics, and New Services. He has published 200 papers in various well-known refereed conferences and journals. He is a co-author of seven books in Greek. He has been a PC member and referee in various international journals and conferences. He has participated in R&D projects such as RACE, ESPRIT, TELEMATICS, EDUCATIONAL MULTIMEDIA, ISPO, EMPLOYMENT, ADAPT, STRIDE, EUROFORM, IST, GROWTH, and others. Also he is member of, experts in the Greek Research and Technology Network (GRNET), Advisory Committee Member to the World Wide Web Consortium (W3C), IEEE Learning Technology Task Force, IEEE Technical Community for Services Computing WG 3.3 Research on Education Applications of Information Technologies, and W 6.4 Internet Applications Engineering of IFIP, Task Force for Broadband Access in Greece, ACM, IEEE, EDEN, AACE, and New York Academy of Sciences.



Alexandros Panagopoulos obtained his Diploma from the Computer Engineering and Informatics Department of the University of Patras (Greece). His research interests include computer graphics, computer vision, networked virtual environments, virtual reality, and distance learning applications.

Thrasyvoulos Tsiatsos obtained his Diploma, his Master's Degree and his Ph.D. from the Computer Engineering and Informatics Department of Patras University (Greece). He is currently an R&D Computer Engineer at the Research Unit 6 of Research Academic Computer Technology Institute (CTI), Patras, Greece. His research interests include Computer Networks, Telematics, Distributed Systems, Networked Virtual Environments, Multimedia and Hypermedia. More particular he is engaged in Distant Education with the use of Computer Networks, Real-Time Protocols and Networked Virtual Environments. He has published more than 40 papers in Journals and in well-known refereed conferences and he is co-author in two books. He has participated in various Greek and European R&D projects such as OSYDD, RTS-GUNET, ODL-UP, VES, ODL-OTE, INVITE, EdComNet, and VirRAD.