Streaming of Plants in Distributed Virtual Environments

Sébastien Mondet, Wei Cheng, Géraldine Morin, Romulus Grigoras, Frédéric Boudon, Wei Tsang Ooi

Streaming of Plants in Distributed Virtual Environments

Sebastien Mondet
University of Toulouse, France

Wei Cheng
NUS, Singapore

Geraldine Morin
University of Toulouse, France

Romulus Grigoras
University of Toulouse, France

Frederic Boudon
CIRAD, Montpellier, France

Wei Tsang Ooi
NUS, Singapore

ABSTRACT

Just as in the real world, plants are important objects in virtual world for creating pleasant and realistic environments, especially those involving natural scenes. As such, much effort has been made in realistic modeling of plants. As the trend moves towards networked and distributed virtual environment, however, the current models are inadequate as they are not designed for progressive transmissions. In this paper, we fill in this gap by proposing a progressive representation for plants based on generalized cylinders. To facilitate the transmission of the plants, we quantify the visual contribution of each branch and use this weight in packet scheduling. We show the efficiency of our representations and effectiveness of our packet scheduler through simulations.

Categories and Subject Descriptors

I.3.2a [**Graphics Systems**]: Distributed/Network Graphics; C.2.4b [**Distributed Systems**]: Distributed Applications

General Terms

Design, Performance, Experimentation

Keywords

Streaming, Plant models, Multiresolution, Progressive coding, Progressive transmission, Networked Virtual Environment

1. INTRODUCTION

Networked virtual environment (NVE) is one of a few truly multi-media applications that involves many media types – 3D models, animation, images, audio, and video. These media data are typically stored on a server, collectively describing a virtual environment. A client connects to the server to navigate through the environment, requesting a subset of the media data based on its current viewpoint. The server transmits the requested media data to the client, which receives it and creates a partial/local 3D scene that is further rendered into a virtual environment at the client.

MM'08, October 27–November 1, 2008, Vancouver, Canada.

The multimedia research community have made much progress on audio and video transmissions, enabling high quality audio communications and video streaming within the NVE. The quality of 3D objects in NVEs, however, is still primitive and not realistic in general. Simplified models or image-based representations are commonly used in NVE to reduce both computational and bandwidth requirements. While Moore's Law and advances in GPU technology have made concerns on computational requirements less relevant, network bandwidth still remains a bottleneck. For instance, current generation of GPU is capable of rendering the Stanford's Thai Statue model with 10 millions triangles but the model, with a size of 122MB after compression, needs 1.6 minutes to download even on a fast 10 Mbps link. The latency induced by downloading completely such an object during a client navigation is unacceptable for interactive use. Thus, to enable realistic, high resolution 3D object in NVE, it is not feasible to render a 3D object only after it is completely received.

The technique of *progressive streaming* should be used to trade off between waiting time and quality. With progressive streaming, a low resolution version of the 3D object is first received and rendered. Subsequent received data, called *refinements*, further improve the quality of the 3D object. Progressive streaming is the key to enabling high quality and realistic 3D objects in NVE under bandwidth constraints. There are much on-going research on progressive streaming, focusing on mesh-based and point-based representations. These representations, however, are inadequate in representing plants.

Plants are important and common objects in a virtual world. Just as in the real world, plants help create a pleasant and realistic virtual environment, especially those involving natural scene. Realistic modeling of plants are crucial in NVE applications such as virtual forests or virtual botanical gardens, where users are expected to inspect a plant closely and possibly interact with plants. Previous work has focused on how to accurately model a plant [27, 4, 25, 24, 21] or making it easy to create a plant[1] within the virtual environment. Realistic and detailed plant models can require up to hundreds of thousands of polygons. Remolar et al. [27] estimated that a plant generated by XFrog, a well known plant modeling platform, can consist of 50,000 polygons to represent the branches. The plants can have 20,000 or more leaves, which themselves consist of polygons. Neubert et al. [21] reported the plant models that they used consist of up to 555,000 polygons. These numbers are for a single plant. In natural scenes, such as forests, one would expect the scene to contain tens to hundreds of plants. The size of these plants motivates the need to stream progressively, rather than to wait until the complete plant model is received before being displayed.

[1]Dyrad (http://dyrad.stanford.edu)

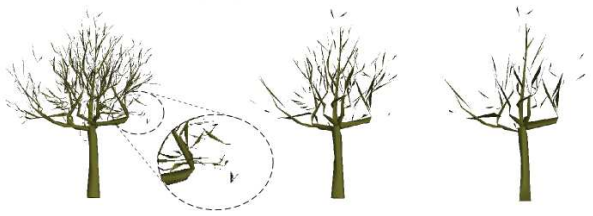

Figure 1: Mesh simplification on a *Walnut* model. The original model consists of 278632 triangles. From left to right: models consisting of 1%, 0.2% and 0.1% of the original model.

Progressive representation for general 3D objects, such as progressive meshes [16] are well studied. These representations, however, are not suitable for plants due to the topology structure of the branches. It is difficult to remove triangles above a certain level, and as a result, representation of plants by progressive meshes does not give satisfactory results [27]. Figure 1 illustrates that simplification of a mesh tree does not preserve the topology, in particular the connectivity, of the tree. Hence, progressive representations suited to the topology of plants are needed.

In this paper, we propose a progressive representation of plants that *preserves* the branching structure of a plant, even at a very low resolution. We focus on representing branches of a plant, and do not explicitly consider leaves in this paper. Our proposal is based on a skeletal representation of the tree, organized hierarchically into a data structure called n-tree and rendered as a set of generalized cylinders. Our representation can be coded efficiently, using differential coding.

The branching structure and differential coding introduce dependencies among the data representing a plant. These dependencies need to be accounted for when streaming our plant representation in a lossy environment. If a packet that a branch depends on is lost, the branch cannot be decoded even if it is received until the lost packet is retransmitted. This situation causes delay in rendering, and should be avoided especially in interactive applications. The other issue to consider for streaming of a plant is that not all branches contribute the same amount to the visual quality of the rendered plants. The branches that are more visually prevalent, then, should be sent first with higher priority.

In this paper, we adopted our previously proposed framework for progressive mesh streaming [9] to streaming of plants. Our framework considers the characteristics of the 3D model (dependencies and visual contributions) as well as characteristics of the network (retransmission delay, packet loss rate), and allows us to estimate the quality of the rendered 3D model at a given time. From this information, a packetization strategy and a sending order is determined in order to maximize the quality of the decodable model on the client side. A key to adopting our previous framework to our new representation of plants is quantifying the importance of each component of the progressive representation. In this paper, we use the size of the branch and distance from the viewpoint as metrics to determine its importance.

We evaluate our method using a complex digitized walnut tree [29] and an apple tree [10] containing irregularities of a real natural branching shape, as well as using more regular trees generated using L-systems [24].

In summary, our major contributions in this paper are (i) an efficient and effective new progressive representation for branching structures in a plant, and (ii) an effective method for streaming this new representation over lossy network for applications such as networked virtual environments.

The rest of this paper is structured as follows. Section 2 reviews the state-of-the-art in 3D model streaming and plant representation. Section 3 describes our proposed representation. We evaluate the suitability of our proposed scheme in Section 4, and finally, we conclude in Section 5.

2. STATE OF THE ART

Before we present our work on progressive streaming of plants, we first review the existing literature on progressive streaming of 3D models and representation of plants.

2.1 Streaming of 3D Models

Previous research in streaming of 3D models have considered many different 3D representations and different aspects of the problem. A common theme of these work is how to improve the quality of the rendered mesh, given that the network is lossy.

One way to improve the quality is to encode the 3D models in a way that is resilient to losses. Park et al. and Yan et al. propose error resilient compression of progressive meshes to accomplish this goal [22, 33] through appropriate segmentation of the mesh to prevent error propagation. Point based representation of an object is inherently loss resilient, and is used by Rusinkiewicz and Levoy [28] and Tarin et al. [30]. Others have considered error control mechanisms for improving the rendered mesh quality [1, 6].

Using the right transport protocols can appropriately trade off between delay and robustness. This issue is considered by Li et al. [18], Al-Regib and Altunbasak [2], Harris III and Karvets [15].

Packet scheduling can affect the rendered quality of the 3D models as well. Ramanathan et al. [26] extended rate- distortion framework to streaming of light fields. Cheng et al. [9] proposed a greedy heuristic in deciding which vertex splits to send first when streaming a progressive mesh. Yang et al. [34] allocates bandwidth between mesh data and textures appropriately to improve the rendered quality.

One can send only segments of the 3D models that is visible to the viewers. Such view-dependent streaming has been explicitly or implicitly considered in these previous work. For instance, Meng and Zha [19] use user's gaze to guide the transmissions of point-based models. Cheng and Ooi [8] consider how to estimate the visibility and visual contributions of vertex splits at the receiver.

An important issue to consider is how to model the quality of the 3D model. Quantifying the visual contribution of a data unit can help deciding which data unit to send. Cheng and Ooi [8] estimate the quality of a vertex split in a progressive mesh using the screen-space area of the faces around the vertex split. In [26], authors measure the distortions of images in light fields. Tian and Al-Regib [31] and Cheng et al. [7] propose metrics to quantify the quality of a simplified mesh with textures.

As presented above, the literature on streaming of 3D models mainly concentrate on mesh-based, point-based or image-based representation of 3D objects. While each of these representations can be optimized to represent different types of objects within a virtual environment, none of them can effectively and progressively represent plants. In the next section, we briefly introduce the current state-of-the-art in plants representation.

2.2 Representation of Plants

Plant geometry is particularly complex and thus motivated a variety of representations dedicated to its specific needs [13, 5]. Branches and foliage are usually treated separately. From a modeling point of view, a string representation of the branching structure is coupled with rewriting rules, called L-systems [24], to simulate the growth of the plant, and with a LOGO style turtle that interprets

the symbols of the string as geometric commands [23]. In this system, the geometry of a symbol is built according to the geometry of previous elements. This idea inspired our work to propose a standardized representation of branches. In this case, leaves are instances at different places of the same geometric symbol. More generally, some high level representations for branches have been proposed based on parametric [4] or implicit surfaces [14]. They rely on a branching skeleton which is extended with radius (given by cross sections or implicit functions). Skeleton is defined as a set of connected parametric curves. These branching structure representations have the advantage to be compact compared to more discrete representations such as mesh and provide support for animation (which is not the case of the simplified models whose connectivity is lost in Figure 1). By default, however, they are not adapted for progressive description. The goal of this paper is precisely to fill this gap.

From a rendering point of view, some representations based on images [20, 11, 3], points [32, 12] or polygons [27] proposed adaptive schemes for displaying trees. These representations mainly focus on foliage (leaves) and thus can be seen as complementary to ours since they are usually complemented with polygonal representations of trunk and branches. If these representations offer some interesting results, they usually require a large amount of data, in particular with points and images. Polygonal representations on sparse geometry such as foliage are not totally convincing. These representations can be streamed with classic methods since they use classic primitives with low-level abstraction. By default, however, they seem more dedicated to static representations. They have to be attached to a skeleton representation to support animation.

3. STREAMABLE REPRESENTATION OF PLANTS

The lack of suitable, progressive, and dynamic representation of plants that allow plants to be streamed and rendered at multiple level of details motivates our work in this paper. In this section, we present our proposed representation of plants. We discuss how we code, compress, and stream our plant representation.

Figure 2 outlines the steps from encoding to streaming of our representation, and guide the presentation of this section. Our starting point is a natural scene using a plant model based on skeletal representation (Section 3.1). This representation served as the basis for our proposed compressed, progressive representation that decorrelates information into three components called branch models, instances, and detail vectors. The detail vectors are compressed with entropy coding (Section 3.2). We then convert the plant model into binary chunks. Each chunk is assigned an importance value, which is then used in packetizing and scheduling the chunks for streaming (Section 3.3).

3.1 Initial Plant Model

Our representation is focused on the branching structure of a plant and is thus based on a skeletal representation. A branch is represented by an axis curve, which is a Bézier of degree d and a radius along the branches. Such generic high level representation can then be displayed as generalized cylinders [4, 25], which is the case in this paper, or implicit surface [14] and is much more compact than a mesh representation. For example, the *Walnut* at full resolution only requires 6872 control points using our representation compared to 278632 triangles using a mesh model. Additionally, it can possibly be extended with kinetic informations for animation (not done in this paper). The branches are organized inside a n-ary tree data structure giving the structure of the plant. We call

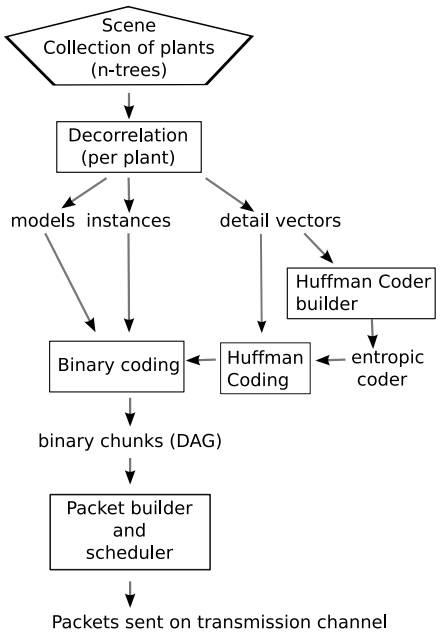

Figure 2: The encoding process for a model based on skeletal representation.

such a data structure a n-tree, to avoid confusion with the concrete plant object we model.

The root of the n-tree is the trunk of the plant and branches borne by the trunk are the n-tree children of this trunk. Each child branch contains a parameter u $(0 \leq u \leq 1)$ giving the position of the attachment point on its bearing parent branch [25]. The parameter u defines the first control point of the Bézier curve of the child branch. The d remaining control points are encoded in the child branch by their three coordinates in space (c.f. Figure 3).

3.2 Compact Progressive Models for Plants

3.2.1 Multi-resolution model

To encode a plant as a compressed multi-resolution representation, we exploit the similarity of branches. The idea of the compression algorithm is to replace the absolute coding of a branch

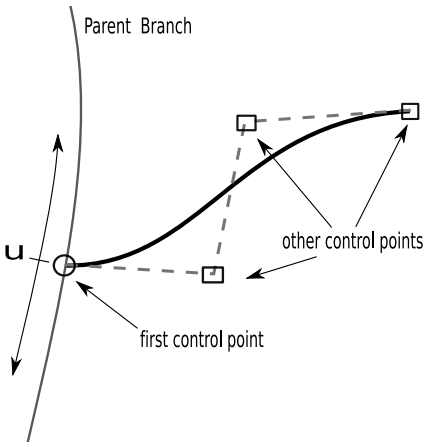

Figure 3: The Bézier curve and the u parameter.

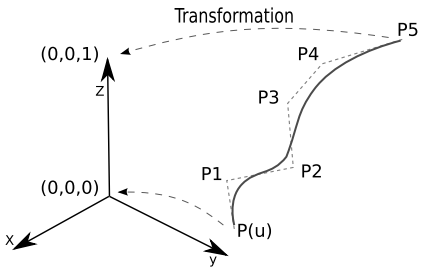

Figure 4: The standardization of a Bézier curves.

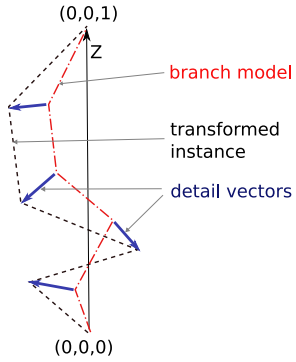

Figure 5: In regular dash, a standard representation of a branch, and in irregular dash, its model, an average Bézier curve of control points $\overline{P_i}$. The *details* vectors are the difference vectors between the control point of the standard representation, and its corresponding control point on the average curve.

control points by differences compared to an average Bézier curve for a chosen set of branches. Due to the similarity of the group of branches, these differences are small. Therefore they may be quantized with a fewer bits, leading to a compact coding. In this paper, we group the branches according to the underlying degree d of the Bézier curve. Other, more accurate, grouping policies could improve the similarity between branches, but that is part of our forthcoming work (c.f. section 5). We elaborate on this process in this section.

Instance. In order to compare and to code differences between two branches, a so called *standard representation* of the skeletal representation is necessary. An affine transformation converts back and forth between an original branch and its standard form. The affine transformation is defined so that P_0 and P_N, the first and last control points of the original Bézier curve b, map to the origin $(0,0,0)$ and the point $(0,0,1)$ respectively (c.f. Figure 4). We characterize this first mapping by two rotation angles, and a uniform scaling factor. Since we choose to apply a uniform scaling, there is a degree of freedom remaining, which corresponds to the rotation around the z, to completely define the affine transformation. We fix the rotation around the z axis so that the center of gravity (or average) of the control points, P^b, lies in the xz plane.

More specifically, the affine transformation mapping a branch to its standard representation is characterized by

- a translation of vector $t = -\overrightarrow{P_0}$,
- a scaling factor $s = \frac{1}{\|\overrightarrow{P_0 P_N}\|}$,
- three rotation angles such that

$$\overrightarrow{T(P_0)T(P_N)} = \vec{z} \text{ and } \overrightarrow{T(P_0)T(P^b)} \cdot \vec{y} = 0,$$

where $T(P)$ denotes the image of P by the affine transformation.

We call the set of transformation parameters the *instance* of the branch.

Branch Model. After obtaining the standard representation for each branch in the group, we can now calculate the average branch: the curve of the average branch is a Bézier curve of the same common degree, such that its i-th control point $\overline{P_i}$ is the barycenter of the i-th control points of the standard curves, mapped from the curves of degree d. We call this average branch the *branch model*.

Detail Vectors. For each branch, we now code in *differential form* the corresponding Bézier curve relatively to the branch model, storing, instead of the coordinates of the control points, its differences to the corresponding control point of the branch model $(\overline{P_i})$ (c.f. Figure 5). We call the differences *detail vectors*. It is thus possible to encode these detail vectors using a limited number of bits.

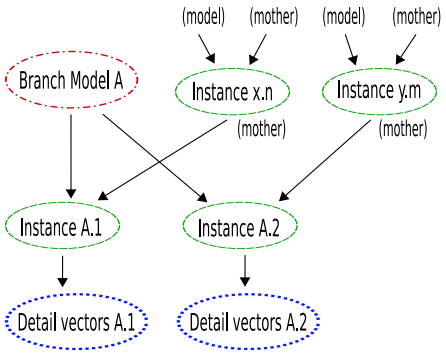

Figure 6: One level in the plant progressive representation.

It should be noted that since the curves are in standard form, the first and last control points do not need to be coded. For example, Bézier curves of degree 3 only need two intermediate points to be defined. The encoding of a branch is now defined by a set of instantiation parameters (transformation to standard form) and a set of differential data (from the branch model).

Our representation allows branches of a plant to be displayed progressively in two ways. First, parent branches are displayed before their children branches and descendants. Second, the branch instances are displayed first, showing an approximate shape of the branch. The detail vectors may refine the shape of the branch later.

3.2.2 *Dependencies in the progressive representation*

In order to efficiently handle a large model (e.g. load it into memory or transmit it over the network) with a progressive representation of the branch system, we need to express the dependencies between pieces of data: A depends on B meaning that the decodability of A *requires* that B has already been decoded.

There are two main families of dependencies: topological dependencies and those generated by the differential coding. The first family is related to the n-tree structure of the plant: a branch depends on the parent branch it attaches to. The second family includes the dependencies due to differential coding, that is, on one hand the dependence between a branch and its branch model, and

on the other hand the dependence between a set of detail vectors and its corresponding instance.

Figure 6 shows these three types of dependencies. Instance $A.1$ depends on its parent branch instance $x.n$ (topological dependence). Note that a child branch is independent of the detail vectors of its parent branch; Section 3.3.2 uses this independence and shows how we prioritize between child branches and detail vectors. Instance $A.1$ also depends on its model A. The set of detail vectors $A.1$ depends on the instance $A.1$. These two last dependences appear with differential coding.

The progressive representation created by the model, instance, and detail vector nodes and their dependencies does not contain cycles and therefore fits in a DAG (Direct Acyclic Graph).

3.2.3 *Quantization of Detail Vectors*

One advantage of multi-resolution differential coding is the ability to quantize small detail vectors with a small number of bits, and to choose accurate binary representative *symbols* according to their distribution. In this section, we first show that the evaluation of our resulting detail vectors leads to a beneficial usage of an entropy coder, and then, we present our quantization method followed by our implementation of a Huffman coder for encoding the detail vectors.

To evaluate the accuracy of using an entropy coder in our method, we have computed, for a given quantization (i.e. a given number of bits per floating point number), the induced error and the theoretical entropy of the represented data. The maximal induced error gives the accuracy of the quantization, while the computed theoretical entropy gives the mean number of bits to expect after Huffman coding.

The quantization is carried out in two steps. First we compute the AABB (Axis-Aligned Bounding Box) of all detail vectors (by finding the min and max of the x,y,z coordinates). Then, to quantize each coordinate into c bits, we build a 3D grid corresponding to 2^{3c} vectors uniformly distributed in the AABB. Each detail vector is then represented by the symbol of the nearest of the vectors discretized on the grid. The quantization error is thus the distance from the quantized vector to the original detail vector. To reconstruct the quantized vectors, a header containing the AABB of the vectors (6 floating point numbers) and the number of bits per coordinate is sufficient.

The resulting error for a given number of bits per coordinate could still be decreased by processing a few iterations of a classification algorithm such as k-means. However, the resulted gain would be offset by increased header size, since transmission of the actual values of the representing symbols chosen by the classification would be necessary.

Once each detail vector is mapped to a symbol we can build the entropy coder. First, we build an entropy histogram, giving the number of detail vectors per symbol. We have summarized the resulting plots for one sample tree (the *Walnut*, c.f. 4) in Figure 7. To improve plot readability, we sort the symbols in increasing order of the number of detail vectors it represents, and we only show effectively used symbols. The shape of each curve shows very promising entropic coding capabilities – a few symbols represent most of the detail vectors, and most symbols are linked to zero or one vectors (e.g. for $c = 6$, only 997 symbols among the 262144 available are actually used).

From the built histograms we can compute the theoretical (Shannon) entropy with the formula:

$$H = -\sum_{w_i>0} w_i \log_2 w_i$$

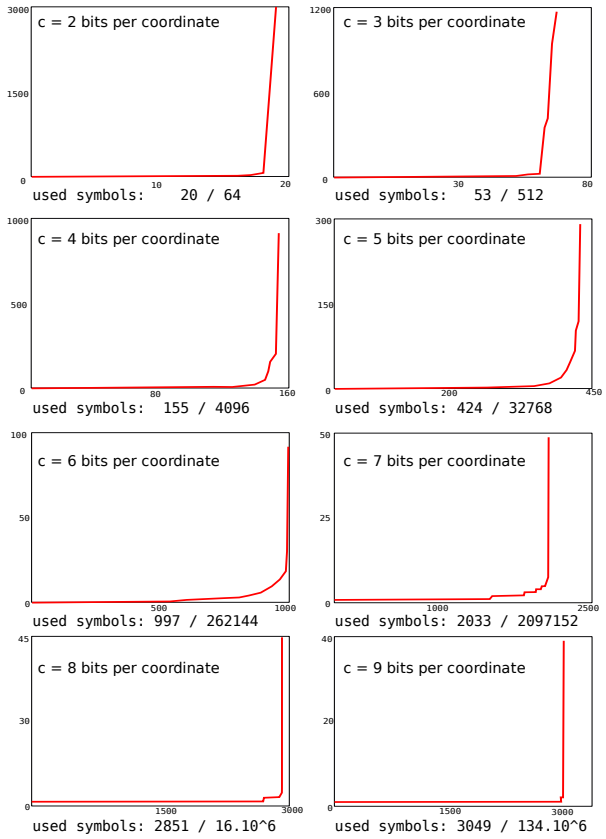

Figure 7: Entropy histograms computed for $c \in [2, 9]$, they show the number of represented vectors, per symbols actually used.

where w_i is the weight of the i-th represented value (i.e. its probability). The results obtained for *Walnut* are displayed in Table 1 and in Figure 8. Table 1 also shows the effective entropy after Huffman coding, which includes the header overhead (Huffman table and parameters). We should note that, for this plant, choosing $c = 4$ or $c = 5$ give the best trade off between entropy and mean error. We evaluate the compression efficiency of our method in Section 4.

3.3 Streaming of Plants

The previous section showed how we can progressively represent and code a plant. This section describes how the plant can be streamed. We present how we encode the data into binary chunks and pack the chunks into packets for transmission.

3.3.1 *Binary Coding*

A progressive representation of a plant consists of four types of data: base data, branch models, instantiation parameters, and detail vectors. Base data contains general characteristics of the plant, information about the trunk, and the entropy coder (the dictionary of symbols). Base data needs to be received first in order to setup the data structures for a plant. Special care has been taken in order to minimize the number of bits used for representing various pieces of information (in particular IDs and pointers).

In order to appreciate the relative weight of various components, details of the *Walnut* model are shown in Table 2. For coding dependencies, only 4 (resp. 11) bits are required to identify each of the 11 models (resp. 1870 instances) nodes. For *Walnut*, the base

Bits per coordinate	Theoretical entropy	Effective entropy (w/ header)	Maximal and mean error
2	0.206	0.412 (0.454)	0.1991 0.0769
3	0.835	0.843 (0.943)	0.1061 0.0616
4	1.411	1.425 (1.694)	0.0526 0.0296
5	2.224	2.234 (3.062)	0.0255 0.0150
6	2.991	3.004 (5.204)	0.0131 0.0075
7	3.573	3.594 (8.743)	0.0065 0.0038
8	3.800	3.826 (11.810)	0.0033 0.0019
9	3.842	3.868 (13.342)	0.0017 0.0009

Table 1: Computed entropies and quantification errors, for *Walnut*, $c \in [2, 9]$. The number of coordinates coded is $3128 \times 3 = 9384$.

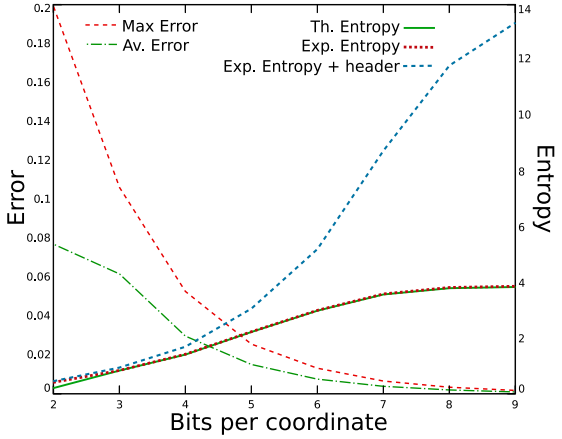

Figure 8: The entropies and errors for the quantization of the *Walnut*

data size is 3020 bits and the total size is 246660 bits (30833 Bytes). Note also that model nodes represent only 2.5% of the data.

Once we encode a plant into binary chunks, the next step is to pack the chunks into packets for transmissions. As with packetizing audio and video data, this process packs binary chunks one-by-one into a packet, until the MTU of the packet is reached. The packet is then passed to the transport layer for transmission. A question that arises here is in what order should the binary chunks be sent. While it is clear that the base data should be sent first, determining what order to pack the other type of chunks such that the best rendered quality is achieved at the receiver, is non-trivial. We describe our approach in the next two sections.

3.3.2 *Quality Metric*

First, let us consider the case where there is no packet loss. In this ideal case, the best way to send the data is in decreasing visual contribution of a chunk – i.e. how much a chunk contributes to the rendered quality of the plant. Doing so would ensure that the

Type	Number of chunks	Size (bits)			
		min	avg	max	total
Models	11	10	533.64	1354	5870
Instances	1870	103	103.00	103	192610
Detail vectors	1870	17	24.15	50	45160

Table 2: Binary coding: data chunks and their size for *Walnut*.

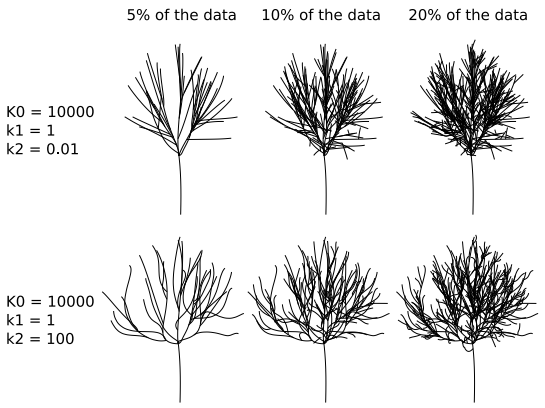

Figure 9: The influence of the choice of (k_0, k_1, k_2) on the structure of *Walnut* after decoding 5%, 10% and 20% of the data.

receiver can view, at any given time, the plants with the best quality possible.

The question thus is how to quantify the visual contribution, or importance, of a chunk. We describe a quality metric for each chunk as follows:

- the importance of a branch model is a constant k_0,
- the importance of an instance is the value of the scaling factor, corresponding to the size of the branch,
- the importance of detail vectors is the importance of the corresponding instance multiplied by the average length of the detail vectors.

The next question is how to relate these three metrics to each other: we choose to have the importance of instances and detail vectors comparable using two constants (knobs), k_1 and k_2, respectively. Intuitively, these can be chosen depending on the application.

Figure 9 illustrates the impact of the choice for k_1 and k_2 for the *Walnut* during the first steps of a progressive decoding. For a botanist, detail vectors are important for the plant to look realistic (so k_2 will be chosen larger than k_1, as shown by second row of the figure); for a computer game player, density of the branches may be of higher relevance (first row). The figure only shows the static visual influence of the coefficients. One should also note that when detail vectors are delayed too much, a *move popping effect* can be observed as branches which carry many others are deformed when their details are decoded.

Finally, if $k_0 \gg k_2$ and $k_0 \gg k_1$, then all branch models are sent before the instances and detail vectors.

The proposed metric is for a single plant. In a scene containing multiple plants, we can adjust the importance of a plant according to its distance from the viewpoint. This importance leads to a simple view-point dependent streaming: plants closer to the viewpoint are streamed first.

3.3.3 *An Analytical Model for Streaming*

For scheduling, two simple strategies may be used: *Naive* which features dependence-only ordering (we send only ready-to-decode data); and *FIFO* which adds importance ordering between binary chunks. *FIFO* can be seen as *almost* optimal in the case of a stream transmission (no packet reordering due to losses). In this section, a

more elaborated *Greedy* streaming strategy is presented; it modifies the *FIFO* ordering to take packet loss into account.

When there are packet losses, one needs to consider dependencies in deciding the sending order. Suppose there are two chunks P and Q, with P depends on Q. If we send P and Q separately in different packets, if the packet that contains Q is lost, then P cannot be decoded even if it is received, until Q is retransmitted. Thus, ideally one should put P and Q into the same packet.

The discussion above shows that the ideal order to send the chunks (with the goal of optimizing the quality of the plant), needs to consider both the dependencies and importance of the chunks. The ideal order also depends on network characteristics – packet loss rate, round trip time, and available bandwidth. The latter two parameters determine time to retransmit a loss packet. In our previous work [9], we have developed a model for estimating the expected quality of the received 3D model, in the context of progressive mesh. In this work, we adopt the model for streaming of plants. We briefly highlight the results from this previous work in the rest of this section for completeness. Interested readers are referred to the original paper for details [9].

Our analytical model considers a sender sending packets at an average (normalized) rate of one packet per unit time. We consider retransmission-based protocol. A retransmitted packet always takes precedence over new packets. Let T_d be the average time between sending a packet and discovering that it is lost (either NACK or timeout-based methods can be used). We pack the data to send into packets, and indexed the packets as 1, 2, 3, etc. We let S_i be the time a packet i is sent, and R_i be the time a packet i is received. The average loss rate of the network is p. We can estimate the sending time, receiving time of a packet using the lemmas below.

LEMMA 1. *If $i \geq T_d$, then for any $k \geq 0$,*

$$E[S_i] \quad = \quad (i - T_d + 1)\frac{1}{1-p} + T_d - 1.$$

Otherwise, if $i < T_d$, then $S_i = i$.

LEMMA 2.

$$Pr(R_i = t) = \begin{cases} (1-p)p^{n_{i,t}} & \text{if } (t - S_i) \mod T_d = 0 \\ 0 & \text{otherwise} \end{cases}$$

where $n_{i,t} = \lfloor (t - S_i)/T_d \rfloor$ is the number of times packet i was lost when $R_i = t$.

LEMMA 3.

$$Pr(R_i \leq t) = 1 - p^{n_{i,t}+1}.$$

Let D_v be the decoding time of a chunk v, and $P(v)$ be the set of chunks v depends on. Then, we have the theorem below.

THEOREM 1.

$$Pr(D_v = t) = \sum_{j \in \mathcal{P}(v)} \frac{Pr(R_j = t)}{Pr(R_j < t)} \prod_{k \in \mathcal{P}(v)} Pr(R_k < t) \qquad (1)$$

The expected decoding time of a chunk v is thus

$$E[D_v] = \sum_{j=t}^{\infty} jPr(D_v = j), \qquad (2)$$

This model we developed previously allows us to estimate when a chunk can be decoded, considering dependencies and network characteristics. We can use this estimation to help us decide the sending order of the chunks. We proposed the following greedy heuristic in our previous work as well [9].

Suppose a chunk i has an importance w_i, as calculated from the previous section. We consider the chunks that have not been sent. For each chunk i, if all chunks that i depends on has either been pack or sent, we decide whether to send i in the current packet, or in the next packet. We compute a metric called *penalty* δ_i, given by

$$\delta_i = w_i(E[D_i^{next}] - E[D_i^{curr}]), \qquad (3)$$

where D_i^{curr} and D_i^{next} are the decoding time of i if i is packed in the current packet and next packet respectively. Minimizing the penalty maximizes the difference in decoded plant quality. The greedy heuristic therefore simply packs the branch with highest penalty at each step. We shall see in section 4.2.2 the advantage of the greedy strategy when an important node is lost.

4. EXPERIMENTS

In order to validate our multi-resolution coding scheme, we have at first evaluated the resulting compressed representation, then tested the multi-resolution interdependent organization of the binary data over a lossy network.

Compression and streaming have been applied to three plants. We have used two digitized plant models: a 20 year old *Walnut* tree [29] and an apple tree [10]. The walnut tree is 7.5 meters high and 5.8m large. It took two weeks to digitize using a Polhemus 3Space Fastrack electromagnetic device. We pre-process it by fitting Bézier curves to series of digitized points representing branches. Our representation is thus composed of approximatively 1900 branches with a total of 6900 control points. The apple tree is 6 year old, 2.8m high and 2m large and is made of 430 branches and 1350 control points.

To extend our experimental range of models, we have also generated some examples using L-systems. For example, we use a spruce-like tree composed of 4300 branches and 17500 control points. Of course, if used in an application, L-systems models would have been more efficiently transmitted by sending their generative rules and parameters. But determining generative process of a given tree is not always possible, in particular for measured tree.

4.1 Compression of Plants

In order to appreciate the efficiency of our compressed model we have chosen to compare it with a well-known compression method (bzip2). Results are shown in Table 3. First row contains the size of a basic serialization of geometry and topology of the Bézier n-tree (with floats and integers coded on 32 bits). Second row shows the performance after compression with bzip2. Third row shows results for our method if binary chunks are concatenated. Even if simple, this concatenation keeps an important property of our model: it is progressive. Naturally this is appropriate for either file storage (with progressive loading) or network transmission (with progressive rendering on client). Moreover, we could save a little more by decreasing the pointer overhead: if binary chunks are completely ordered, then some IDs (e.g. instance and detail IDs) can be removed.

Results of Table 3 show that, if five bits are used for coding a coordinate for *Walnut*, bzip2 compression applied to the basic coding reduces data size to 77% of the original, whereas our coding method brings it down to 36%.

4.2 Transmission of a Set of Trees

As the main goal of our coding scheme is the progressive transmission of large natural scenes, we evaluated our transmission schemes over a lossy network to see how our interdependent bi-

Figure 10: Rendering after progressive decoding of the *Walnut* (quality metric parameters: $k_0 = 10000, k_1 = 1, k_2 = 10$)

Tree name	Walnut	Apple Tree	L-system
Basic	89964	17980	228504
Basic + bzip2	69659 (77%)	12395 (69%)	132758 (58%)
Our method ($c = 4$)	30833 (34%)	6606 (37%)	68618 (30%)
Our method ($c = 5$)	32438 (36%)	6864 (38%)	68831 (30%)
Our method ($c = 6$)	34950 (39%)	7152 (40%)	69326 (30%)
Our method ($c = 7$)	39102 (43%)	7381 (41%)	69964 (31%)

Table 3: Comparison of coding performance of three methods: basic coding, basic coding compressed with bzip2 and our progressive coding (for a given c number of bits per detail vector coordinate). The sizes are given in bytes.

nary chunks can be efficiently packetized, transmitted over a lossy network, and progressively decoded.

4.2.1 Experimental Streaming Setup

We run our experiments using a client-server streaming setup. On the server-side, the 3D scene data is loaded as a DAG of binary chunks (Figure 2). This DAG structure is packetized by the scheduler which implements a given strategy (c.f. 3.3.3) and transmitted. Lost packets are retransmitted; the retransmission order differs depending on the strategy. On client-side (Figure 11), binary chunks are demultiplexed between objects and a progressive decoder associated with the plant decodes them as soon as possible to reconstruct the plant for rendering (Figure 10).

We use unreliable transport (DCCP) with retransmission in our experiments so that packets after a loss can still be processed before repairing the loss (TCP buffers packets to deliver them in order). For reproducibility, we first capture a packet trace of random packet data over given network conditions, and use the trace to simulate transmissions of plants over realistic network conditions and replay it locally while decoding progressively the packetized plants.

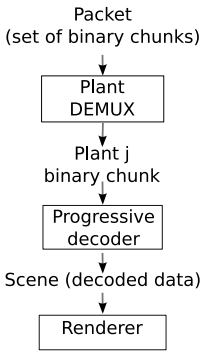

Figure 11: The decoding process.

Experimenting with DCCP has proved to be very challenging. First, because of security restrictions on firewalls, DCCP ports needs to be opened. And because DCCP uses TCP's port's system (client receives data on a random port), this makes the task of configuring port traversal via firewalls tricky. The second problem came from the backbone itself: experiments carried out between Toulouse and Singapore showed that somehow some backbone routers do not route DCCP traffic. The solution to the last problem was to implement an application-level UDP tunnel that carries out DCCP traffic.

Two main congestion control mechanisms have been implemented in DCCP (CCID2 and CCID3 in DCCP's RFC [17]). CCID2 implements TCP-like congestion control whereas CCID3 uses TFRC (TCP-friendly Rate Control). We have chosen to present here experiments with CCID2, as we found its implementation in recent Linux kernels more robust.

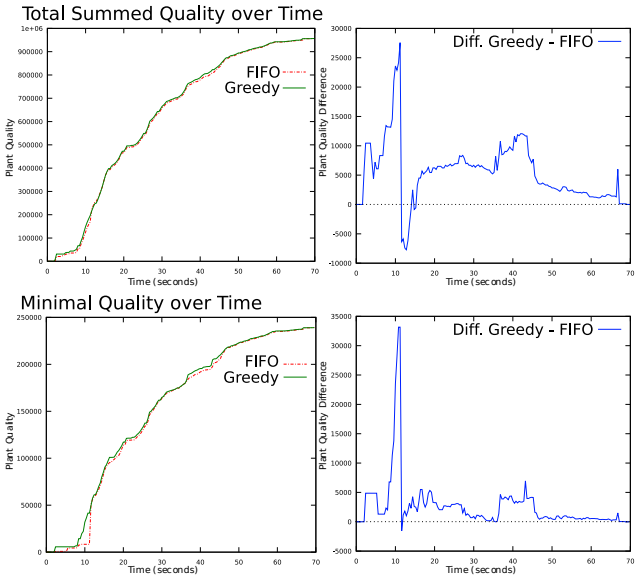

Figure 12: Comparison of the efficiency of packetization strategies *Greedy* and *FIFO*, during the progressive transmission of 4 walnuts between Toulouse and Singapore.

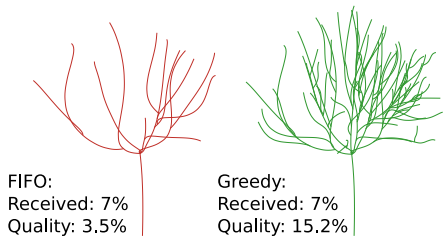

Figure 13: Example showing the structure of the same tree of the scene better packetized by the *Greedy* strategy.

Experiments with DCCP have been carried out on both WAN (with the tunnel) and LAN (a WAN was simulated by adjusting packet loss and delay with traffic control tools of the Linux kernel). For conciseness, we shall only present here results on a real WAN.

4.2.2 *Transmission Results*

As an example of our experiments, we give results for the transmission of a set of four *walnut* trees (considered independent from each other) over the Internet, between Toulouse and Singapore. The dynamic importance proposed at the end of section 3.3.2, that is, the importance is *scaled* by the distance to the view point is computed. We used DCCP + retransmission (with CCID2) in our UDP tunnel (c.f. 4.2.1). For this network capture, we have measured a packet loss weaving between 10% and 15%.

Two packetization strategies have been tested: *FIFO* strategy which takes into account both dependencies and importance ordering between binary chunks and the proposed *Greedy* strategy (see section 3.3.3).

In Figure 12, we show the evolution of the quality of the set of trees (in the first row, the sum of the tree qualities; in the second row, the quality of the tree of minimal quality) over time for *FIFO* and *Greedy* strategies (the right column shows the difference between these two curves). Both experiments use the same set of packet trace. So, at a given time, both have received exactly the same amount of data. On the arrival of a packet, we reconstruct the trees with the available binary chunks and compute the quality. Therefore, the plots tend to confirm that the proposed *Greedy* strategy of the binary chunks improves the amount of decodable data over transmission time. Figure 13 shows the reconstruction of one of the trees after receiving 7% of its data during both transmissions. We can observe that most of the data received in the *FIFO* case is unusable due to the lack of one or more binary chunks on which many others depend. The aim of the *Greedy* packetization strategy is rightly to prevent those "accidents" to happen.

5. CONCLUSION AND PERSPECTIVES

We have proposed an original progressive representation of branching systems adapted to the streaming of 3D scenes. This representation allows efficient compression of the plant geometry represented by generalized cylinders. Our method outputs a set of interdependent binary pieces of data well suited for packetization and progressive transmission over lossy networks with the help of a quality metric.

There are several directions we can take to continue this research.

For the progressive representation of plants, we plan to design more accurate methods for grouping branches, for example, using PCA (Principal Component Analysis) on their geometry. Doing so would lead to more relevant branch models, allowing to delay the transmission of detail vectors and more aggressive quantization. Moreover, it would be interesting to find out if sharing branch models between different plants of the scene can increase the compression. The challenge is to have more accurate models while keeping a good ratio between the numbers of models and instances.

For progressive transmission of plants, the quality metric could be made more *dynamic* by considering the viewpoint of the navigating user more accurately. For example, at the scene level, scene data close to the central region of the view frustum should have a higher importance and therefore be streamed first. The current quality metric also depends on the tunning of parameters k_0, k_1 and k_2 (3.3.2). A user survey may be useful to evaluate the subjective impact of the quality metric, especially in the presence of undesired effects such as *popping*, a common problem of most 3D multi-resolution models.

The efficiency of our progressive representation could be evaluated in other applications requiring progressive models for plants, for example 3D visualisation on mobile devices or plant modeling or animation software.

Finally, we also think of adapting standardization and instantiation to represent leaves. Our tree representation can be extended with some reference leaf symbols that will be instantiated similarly to branches to encode the foliage.

6. ACKNOWLEDGMENTS

This work has been partly funded by the *NatSim* ANR project (French National Research Agency 05-MMSA-0004-01) and National University of Singapore Academic Research Fund R-252-000-306-112.

7. REFERENCES

[1] G. Al-Regib and Y. Altunbasak. An unequal error protection method for packet loss resilient 3D mesh transmission. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, pages 743–752, New York, NY, June 2002.

[2] G. Al-Regib and Y. Altunbasak. 3TP: An application-layer protocol for streaming 3D models. *IEEE Transactions on Multimedia*, 7(6):1149–1156, December 2005.

[3] S. Behrendt, C. Colditz, O. Franzke, J. Kopf, and O. Deussen. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum*, 24(3):507–516, September 2005.

[4] J. Bloomenthal. Modeling the mighty maple. *ACM Computer Graphics (SIGGRAPH'85)*, 19(3):305–311, July 1985.

[5] F. Boudon, A. Meyer, and C. Godin. Survey on computer representations of trees for realistic and efficient rendering. Research Report 2301, LIRIS, Université Claude Bernard Lyon 1, 2006.

[6] Z. Chen, J. F. Barnes, and B. Bodenheimer. Hybrid and forward error correction transmission techniques for unreliable transport of 3D geometry. *Multimedia Systems*, 10(3):230–244, March 2005.

[7] I. Cheng, L. Ying, and A. Basu. Packet-loss modeling for perceptually optimized 3D transmission. *Advances in Multimedia*, 2007(1):11–11, January 2007.

[8] W. Cheng and W. T. Ooi. Receiver-driven view dependent streaming of progressive mesh. In *Proceedings of the 18th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Braunschweig, Germany, May 2008.

[9] W. Cheng, W. T. Ooi, S. Mondet, R. Grigoras, and G. Morin. An analytical model for progressive mesh streaming. In *Proceedings of the 15th ACM International Conference on Multimedia*, pages 737–746, Augsburg, Germany, September 2007.

[10] E. Costes, H. Sinoquet, J. Kelner, and C. Godin. Exploring within-tree architectural development of two apple tree cultivars over 6 years. *Annals of Botany*, 91:91–104, 2003.

[11] P. Decaudin and F. Neyret. Rendering forest scenes in real-time. In *Proceedings of the 15th Eurographics Symposium on Rendering*, pages 93–102, Norrköping, Sweden, June 2004.

[12] O. Deussen, C. Colditz, M. Stamminger, and G. Drettakis. Interactive visualization of complex plant ecosystems. In *Proceedings of IEEE Visualization*, Boston, MA, October 2002.

[13] O. Deussenand and B. Lintermann. *Digital Design of Nature: Computer Generated Plants and Organics*. Springer-Verlag, 2005.

[14] C. Galbraith, P. MacMurchy, and B. Wyvill. BlobTree trees. In *Proceedings of Computer Graphics International*, Crete, Greece, June 2004.

[15] A. F. Harris (III) and R. Kravets. The design of a transport protocol for on-demand graphical rendering. In *Proceedings of the 12th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 43–49, Miami, FL, May 2002.

[16] H. Hoppe. Progressive meshes. In *SIGGRAPH'96 Conference Proceedings*, 1996.

[17] E. Kohler, M. Handley, and S. Floyd. RFC 4340 - Datagram Congestion Control Protocol, 2006.

[18] H. Li, M. Li, and B. Prabhakaran. Middleware for streaming 3D progressive meshes over lossy networks. *ACM Transactions on Multimedia Computing, Communication and Applications*, 2(4):282–317, 2006.

[19] F. Meng and H. Zha. Streaming transmission of point-sampled geometry based on view-dependent level-of-detail. In *Proceedings of 4th International Conference on 3-D Digital Imaging and Modeling*, pages 466–473, Banff, Canada, October 2003.

[20] A. Meyer, F. Neyret, and P. Poulin. Interactive rendering of trees with shading and shadows. In *Proceedings of the Eurographics Workshop on Rendering Techniques*, pages 183–196, London, UK, June 2001.

[21] B. Neubert, T. Franken, and O. Deussen. Approximate image-based tree-modeling using particle flows. *ACM Transactions on Graphics*, 26(3):88, July 2007.

[22] S.-B. Park, C.-S. Kim, and S.-U. Lee. Error resilient 3-D mesh compression. *IEEE Transactions on Multimedia*, 8(5):885–895, October 2006.

[23] P. Prusinkiewicz. Graphical applications of L-systems. In *Vision Interface*, pages 247–253, Vancouver, BC, May 1986.

[24] P. Prusinkiewicz and A. Lindenmayer. *The algorithmic beauty of plants*. Springer Verlag, 1990.

[25] P. Prusinkiewicz, L. Mündermann, R. Karwowski, and B. Lane. The use of positional information in the modeling of plants. *ACM Computer Graphics (SIGGRAPH '01)*, 22(4):289–300, 2001.

[26] P. Ramanathan, M. Kalman, and B. Girod. Rate-distortion optimized streaming of compressed light fields. In *Proceedings of International Conference on Image Processing*, pages III–277–80, Barcelona, Spain, September 2003.

[27] I. Remolar, M. Chover, O. Belmonte, J. Ribelles, and C. Rebollo. Geometric simplification of foliage. In *Eurographics'02 Short Presentations*, pages 397–404, Saarbrücken, Germany, September 2002.

[28] S. Rusinkiewicz and M. Levoy. Streaming QSplat: a viewer for networked visualization of large, dense models. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pages 63–68, Research Triangle Park, NC, March 2001.

[29] H. Sinoquet, P. Rivet, and C. Godin. Assessment of the three-dimensional architecture of walnut trees using digitising. *Silva Fennica*, 31(3):265–273, 1997.

[30] B. Tari, Y. Yemez, O. Ozkasap, and R. Civanlar. Progressive view-dependent transmission of 3D models over lossy netowrk. In *Proceedings of the 13th European Signal Proceesing Conference*, Antalya, Turkey, September 2005.

[31] D. Tian and G. Al-Regib. FQM: a fast quality measure for efficient transmission of textured 3D models. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, pages 684–691, New York, NY, USA, October 2004.

[32] J. Weber and J. Penn. Creation and rendering of realistic trees. *ACM Computer Graphics (SIGGRAPH '95)*, 29(3):119–128, August 1995.

[33] Z. Yan, S. Kumar, and C.-C. Kuo. Error-resilient coding of 3-D graphic models via adaptive mesh segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(7):860–873, July 2001.

[34] S. Yang, C.-H. Lee, and C.-C. J. Kuo. Optimized mesh and texture multiplexing for progressive textured model transmission. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, pages 676–683, New York, NY, October 2004.