Novel Applications of VR

Large-scale forest rendering: Real-time, realistic, and progressive

Guanbo Bao, Hongjun Li, Xiaopeng Zhang*, Weiming Dong

National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China

Abstract

Real-time rendering of large-scale forest landscape scenes is important in many applications, such as video games, Internet graphics, and landscape and cityscape scene design and visualization. One challenge in the field of virtual reality is transferring a large-scale forest environment containing plant models with rich geometric detail through the network and rendering them in real time. We present a new framework for rendering large-scale forest scenes realistically and quickly that integrates extracting level of detail (LOD) tree models, rendering real-time shadows for large-scale forests, and transmitting forest data for network applications. We construct a series of LOD tree models to compress the overall complexity of the forest in view-dependent forest navigation. A new leaf phyllotaxy LOD modeling method is presented to match leaf models with textures, balancing the visual effect and model complexity. To progressively render the scene from coarse to fine, sequences of LOD models are transferred from simple to complex. The forest can be rendered after obtaining a simple model of each tree, allowing users to quickly see a sketch of the scene. To improve client performance, we also adopt a LOD strategy for shadow maps. Smoothing filters are implemented entirely on the graphics processing unit (GPU) to reduce the shadows’ aliasing artifacts, which creates a soft shadowing effect. We also present a hardware instancing method to render more levels of LOD models, which overcomes the limitation of the latest GPU that emits primitives into only a limited number of separate vertex streams. Experiments show that large-scale forest scenes can be rendered with smooth shadows and in real time.

Article info

Article history:
Received 1 May 2011
Received in revised form 7 December 2011
Accepted 11 January 2012
Available online 1 February 2012

Keywords:
Real-time rendering
Level of details
Shadow mapping
GPU
Forest trees

1. Introduction

Virtual reality techniques are widely used in modern web applications. Rendering a realistic large forest scene in real time plays an important role in web applications, including natural environment simulation, animated television production, online immersive gaming and visual forestry management. Because a forest contains many highly complex geometric models, real-time forest rendering remains a challenge. More difficulties occur when rendering forests with rich plant detail and real-time shadows.

In this paper, we present a framework that integrates a set of novel techniques to realistically render highly detailed forest scenes with real-time shadows through a network. Because a forest with thousands of plants contains a vast amount of geometry, we present an efficient LOD algorithm to generate multi-resolution (MR) models following forest features (Section 3). To generate lightweight files for progressive transmission online, we carefully organize the MR models and the terrain’s digital elevation models (DEM) for efficient rendering on clients. We use scene management files to describe scene parameters, including the model’s distribution information, camera position, etc. The compressed files are uploaded to a server. After establishing a connection, the client first downloads a scene file list containing the MR models’ names, a DEM file name and a scene management file name. The corresponding files are then progressively downloaded. When processing transmissions to the client, a DEM takes priority over any single tree. Among the LOD tree models, coarse tree models with small geometry take priority over complex tree models. After the simplest model file is loaded into memory, the rendering system begins. In Section 4, we introduce a novel shadow map generation scheme suitable for a large-scale forest scene. We adapt parallel-split shadow mapping (PSSM) [1,2] with a LOD shadow maps generation scheme. This not only uses different sizes for shadow maps, but also updates shadow maps with different frequencies. We smooth the shadows using Gaussian filters implemented entirely on the graphics processing unit (GPU) for anti-aliasing, which achieves a soft-shadow effect. The new technique in this paper integrates GPU-based dynamic geometry LOD [3]. By taking advantage of the latest functionalities introduced in OpenGL 4, the number of driver invocations is dramatically reduced. Culling and LOD determining are entirely
implemented on GPU without breaking the drawing batches in other complicated and expensive CPU-based methods. Our contributions are as follows:

- We present a new framework for rendering large-scale forest scenes realistically and quickly that integrates extracting LOD tree models, rendering real-time shadows for large-scale forests and transmitting forest data to network applications.
- We present a novel hardware instancing method to render in real-time large-scale forest scenes containing over one million highly detailed trees.
- We integrate a novel LOD shadow maps generation scheme to render shadows in large-scale forests. This both reduces geometric complexity and saves GPU memory.
- We present a new leaf phyllotaxy LOD model to match original leaf models with textures. Our LOD strategy is also suitable for progressive rendering through a network.

2. Related work

Real-time forest rendering has been intensively studied, and many successful methods have been developed. The representative methods are displayed below.

Point-/line-based rendering: point and line models efficiently render small polygons. Reeves et al. first used point models to render plants [4]. Both points and lines usually combine with polygons to construct hybrid tree models. Polygonal geometry is used for close distances, but more points and lines represent trees by randomly sampling as the viewer moves away from the trees. Point-based rendering of trees is effective only for distant objects, as it is visually unacceptable at close range.

Image-based rendering (IBR): IBR is efficient and has been used to represent tree models for many years. Billboards are the simplest of the IBR techniques. They consist of triangles or quadrilaterals covered by a semi-transparent 2D texture. IBR techniques efficiently render complex natural objects, such as grass. They render more efficiently than polygon-based rendering because a single primitive replaces a large amount of geometry, and they remain the best choice for many recent industrial simulators. Several kinds of billboards have been designed. The simplest is a billboard that represents an entire model constantly facing the camera. This method shows no parallax when the camera is moving; however, when inspected closely, it appears unrealistic due to weak geometry. Therefore, this type of billboard is often used in the background. A better approximation of trees is provided by the fixed crossed quadrilaterals of two or three billboards crossing one another to produce a more three-dimensional impression of trees, resulting in a cardboard appearance. However, artifacts occur in the direction parallel to a billboard. Billboard clouds, introduced by Decorret et al. [5], are a means of extreme simplification. This method represents geometry through a set of arbitrarily oriented billboards to generate visually pleasing results with smooth geometry models. However, it fails to work optimally for plant models with high complexity, mainly because the plane-space transform used does not provide significant viewing effects due to noncompact tree model geometry.

Many methods have been introduced to deal with plant models, including those by Fuhrmann et al. [6], Behrendt et al. [7], Candussi et al. [8], and Bao et al. [9]. These modified methods use billboard clouds to represent models’ geometry through a set of arbitrarily oriented billboards, significantly reducing the number of polygons. The simplified models, however, lose much of the original geometry information. These simplified trees are unsuitable for close-up viewing because the weak parallax artifacts make dynamic lighting difficult to achieve. Ismael et al. [10] created too many texture images for tree simplification (Fig. 2). The large number of images increases transmission time when their method is used in network applications.

Zhang et al. [11] proposed a forest model representation using hierarchical layered depth images. Discrete textured quadrilaterals are generated from sampled depth images of polygonal tree models and organized with a hierarchy structure. This method realizes the fast display of a large-scale forest. However, its...
storage cost is too high to achieve real-time shadow and dynamic lighting effects.

Liu et al. [12] proposed an approach that adopts a dynamic quad stream on GPU buffers for view-dependent rendering. Their method is based on hierarchical layered depth images and generates a set of presampled parallel billboards to represent a tree. A quad stream, arranged as billboards, represents tree models. In the runtime rendering stage, they apply geometry tessellation to increase the emitted primitives to progressively refine the tree models. This approach represents trees with billboards for rendering speed and with detailed quad meshes to provide high visual quality. It can render large-scale forest scenes with high efficiency and reduce memory costs and data transfer overhead. The limitation of this image-based algorithm is also clear: receiving a realistic rendering effect, such as real-time shadowing, is difficult (Fig. 3).

**Volume-based rendering:** A volume is a 3D reference, e.g., a box containing one or more instances of the object to be rendered. The GPU-based lighting and shadowing of complex natural scenes proposed by Cohen et al. [13] adds realism to volume-encoded forest scenes; however, these visually pleasing effects are achieved by precomputing filtering textures and horizon maps. Decaudin et al. [14] utilized no periodic volumetric texture tiles to render vast and dense forests with the texturing power of graphic hardware. These volumetric textures were generated by converting the plants’ geometry into a series of parallel image slides and tiled over the ground to represent forests. Volume representation offers a full parallax effect from any viewing direction: when the viewer moves, objects are correctly rendered with no flatness, which differs from using billboards and is advantageous for use in flight simulation. However, walking through and into the forest is not yet possible. Taking advantage of the GPU geometry shader, Decaudin et al. improved their previous method and introduced volumetric billboards [15]. A set of volumetric textures called volumetric billboards presents plant model foliage. Walking through and into the forest is possible, but the memory cost is so high that few trees can be rendered.

**Polygon-based rendering:** The polygon is the primitive used to represent the geometric model of a plant. Many geometric simplification methods have been explored to eliminate minor geometric details. Multi-resolution modeling and LOD algorithms are frequently used methods that address polygon decimation and geometry compression. These methods efficiently simplify smooth objects, but cannot be applied to trees without changing their overall appearances because trees possess complex topologies.

Deng et al. [16] proposed a LOD-based simplification method for coniferous leaves that used two representations, i.e., cylinder and line, to represent nearby and faraway coniferous leaves, respectively, in which lines can be merged for further simplification. Automatically generating multiple LODs for trees is a nontrivial task, and those methods do not naturally support plants, as foliage contains many isolated surface patches.

Gumbau et al. [17] presented a foliage pruning method for real-time vegetation rendering. They base their algorithm on the principle of some of the foliage not being visible, depending on the viewer’s location. In a preprocess step, LOD tree models’ foliage is divided into a cloud of cells, and each cell’s visibility is computed from a set of external viewpoints surrounding the foliage. A run-time stage interactively alters the LOD models by detecting less-visible foliage in real time. First, each cell evaluates the viewpoint position to calculate a percentage value. Next, a list of polygons that compound the visible leaves is generated based on this value. Finally, the size and color of the remaining geometry are altered for rendering.

This method is efficient for the GPU, renders with a decreased LOD to improve efficiency and considerably reduces the extraction and visualization times of the foliage-representing geometry. However, the method cannot efficiently achieve real-time shadowing effects. In the shadow maps generation pass, the LOD models can only be altered in the light space frustum to record correct depth information, which is an extra rendering burden. Shadows in Fig. 4 are not recalculated every time, but only on the first frame.

Bao et al. [18] proposed a leaf geometry modeling approach to have leaf models match leaf textures, so that the visual effect and model complexity can be balanced well. The proposed leaf model has an advantage over other models in that different texture images (in alpha format) can be changed conveniently to the same leaf model without damaging the visual effects. In this paper, we present a novel LOD shadow maps generation scheme to render large-scale forest shadows. We also present a novel hardware instancing method which overcomes the limitation of the latest graphic cards that emits primitives into only a limited number of separate vertex streams, and renders more levels of LOD. Our LOD model strategy is also suitable for progressive rendering through networks. The rendering performance is greatly improved.

With their impressive performance improvement, current GPUs allow real-time rendering of much more complex scenes. Modern computer applications, especially video games, raise the realism levels of forests in their scenes; detailed models are more suitable for realistic rendering.

**Progressive transmission:** To transmit model data over a communication line to show progressively better approximations, the progressive mesh (PM) proposed by Hoppe [19] can naturally support progressive transmission without requiring additional transmission time. The view-dependent streaming of progressive meshes presented by Kim et al. [20] accounts for visual perception characteristics, giving priority to those on the visual impact of transmitting large patches and reducing user wait time. View-dependent
progressive transmission of the realization process is complex, which limits its flexibility. Tian and Al Regib [21] introduced a bit-allocation framework, BaTex3, for progressive transmission. They carefully organized the bit stream to quickly present high-resolution visual experience. These methods are unsuitable for tree simplification and previews allows a highly responsive image viewing and browsing framework, BaTex3, for progressive transmission. They carefully limit its flexibility. Tian and Al Regib [21] introduced a bit-allocation method that can handle 3D objects associated with attributes, including colors, while producing high-quality intermediate LODs. This method also has efficient adaptation mechanisms to optimize LOD management of 3D scenes according to constraints, including network bandwidth, the device’s graphic capability, display resolution and user preferences. This method does not account for more complex constraints in the adaptation parts.

In our application, we present a special data-encoding method that can effectively transmit progressive forest scene data. We use full polygon representations for near trees and LOD controls for rendering, which can also be used for progressive transmission.

Hardware instancing: The visualization of large-scale forests has always been a challenging problem not only due to the high geometric complexity, but also due to the batch problem [23]. This is caused by generating large numbers of graphics API draw calls in every frame. Due to this, large-scale forest rendering becomes often CPU bound.

Most previous methods, such as [24], work like this: in each frame, they first update all geometry instances in the view frustum and then put the updated data into several vertex streams and finally render the streams in a few draw calls. These methods still need CPU intervention, such as culling instances against the view frustum and determining appropriate LOD models. The method of [25] uses geometry shaders to perform culling and dynamic LOD selecting on CPU. However, this approach is not suitable for modern graphics hardware and needs too many rendering passes to separate LOD models.

Shadow mapping: Shadow mapping has been used extensively in 3D games, film productions and other applications for producing shadows. Standard shadow mapping suffers from its well-known aliasing problem. This occurs when projecting the view frustum into the shadow map, as sampling rates near the viewpoint are much higher than farther from it. Many techniques have been developed for anti-aliasing. These approaches can be classified into warping and partitioning methods [26].

The most prominent warping method is perspective shadow maps (PSM), introduced by Stamming and Drettakis [27], which applies a perspective transformation to the scene before rendering it into the shadow map. This creates more samples near the center of projection and fewer samples near its far plane. This method is quickly computed and reduces shadow map aliasing with almost no overhead. Light-space perspective shadow maps (LiPSM), proposed by Wimmer et al. [28], is another prominent approach that wraps the camera frustum so it does not change the light source directions. A new light frustum is built with a viewing ray perpendicular to the light’s direction. After the warp, objects near the viewer appear bigger in the shadow map and therefore receive more samples. Martin and Tan’s trapezoidal shadow maps (TSM) method [29] is a similar approach that builds a bounding trapezoid (instead of the frustum in LiPSM) of the camera frustum as observed from the light. These warping methods only generate one shadow map, which is insufficient for a large-scale forest; even the largest shadow map’s size is used, and they are, therefore, not commonly used in computer games.

In contrast to warping methods, partitioning methods approximate the ideal sample distribution through multiple shadow maps. Parallel-split shadow mapping (PSSM), introduced by Zhang et al. [12], splits the view frustum into different depth ranges using split planes parallel to the view plane, then renders multiple shadow maps for the split parts. In PSSM, most of the forest scene rendering burden lies with the rendering of an extensive amount of geometry in every frame for each shadow map. Therefore, we combine PSSM with a novel LOD shadow maps generation scheme and smooth filters to produce real-time and anti-aliasing shadows for large-scale environments.

3. Tree modeling and model processing

3.1. Leaf phyllotaxy model

By defining a leaf vein quadratic interpolation function, leaf geometry can be obtained and leaf LOD models can be easily extracted [30]. In practice, except for leaf LOD models, phyllotaxy models can be generated for LOD application. Phyllotaxy models can be constructed by specifying geometry parameters. These parameters include the average leaf length $L$, average leaf width $W$, the angle $\theta$ between a leaf vein and the phyllotaxy axis, the angle $\alpha$ between the leaf and the axis, leaf number at one node, and the length between adjacent internodes. The angle $\beta = 90 - \alpha$, which defines the leaf direction, signifies the angle between the axis and the normal vector of leaf plane (Fig. 5a). For example, when $L = 10.0 \text{ cm}$, $N = 3$, $\theta = 80.0$, and $\alpha = 89$, a whorled phyllotaxy is generated (Fig. 6); Fig. 7(a) shows an opposite phyllotaxy, where $L = 6.0 \text{ cm}$, $N = 2$, $\theta = 80.0$, and $\alpha = 1$. Other phyllotaxies can be generated, including alternate and fascicled types. Leaf phyllotaxies are at the nodes of high-level branches along branch directions. To increase their visual effect, each parameter employs random factors.

In addition, when specifying these parameters, adjacent leaves should not intersect. Fig. 5(b) shows a mesh phyllotaxy model, where adjacent leaf boxes intersect; their texture leaves are well...
arranged in Fig. 5(c), indicating that leaf contour is a factor in modeling phyllotaxy.

Leaf phyllotaxy LOD models are generated simply in our experiments. Via leaf union, a phyllotaxy is represented with a quadrilateral accompanied by a texture image. Figs. 6(b) and 7(b) illustrate the method.

3.2. Branch model processing

Tree skeleton models are obtained using the plant modeling software AMAP. A series of static MR models are constructed [26], which are used in real-time scene rendering, as discussed below. We construct branches using this method because it generates continual LOD models. Continual LOD models are useful for both efficient memory cost and model switching in a forest.

3.3. Extraction of plant LOD models

A large forest often contains thousands or millions of trees. If each tree is represented by the most detailed information possible, the memory will easily be exhausted. LOD models often used to save memory while maintaining realism. However, for a large forest, too many LOD models also exhaust the memory. Therefore, we use a series of less than eight LOD models to represent a tree instance.

Because forest occlusion is common, lower-resolution models can reduce cases of extreme occlusion. Using the simplification methods introduced in Section 3.1, Five LOD models have been selected according to their rendering effects. The closer the model is to the viewpoint, the larger its size. Therefore, we use a series of less than eight LOD models to represent a tree instance.

Because forest occlusion is common, lower-resolution models can reduce cases of extreme occlusion. Using the simplification methods introduced in Section 3.1, Five LOD models have been selected according to their rendering effects. The closer the model is to the viewpoint, the larger its size. Fig. 8 illustrates the LOD series used in Section 5. The subtitle of each subfigure shows the number of polygons in each model.

3.4. Upload data to server

For web data transmission, extracted LOD models, digital elevation models (DEM) and scene management files that contain scene parameters including the models’ distribution information and camera position are formed with light weights. They are saved as binary files and compressed in ZIP format with security passwords. The compressed files are uploaded to a server for loading by clients.

4. Model transmission and real-time forest rendering

Once a client establishes a network connection with the server, the client first downloads a scene file list containing MR model names, a DEM file name and a scene management file name. It then downloads the files in the list. The client then unzips the compressed files using the authorization password and deletes the unzipped binary files after loading them into memory.

4.1. Shadows for large-scale forest scenes

In 3D applications, shadows offer better perceptions of the 3D shapes of the objects displayed. Shadows can dramatically enhance the reality of both virtual environments and computer-generated images. In this section, we describe our shadowing and lighting approaches, which run with high realism in real time.

4.1.1. LOD shadow maps generation scheme

Shadow mapping has been a popular algorithm for fast shadow generation since its inception and is widely used in many applications, e.g., film production and the game industry.

Shadows generated by a single shadow map usually suffer from the inherent aliasing artifact. This artifact is more obvious in large-scale outdoor spaces when rendering objects with fine details, such as trees with thin branches. Many anti-aliasing shadow generation algorithms have been developed. We choose PSSM to alleviate the aliasing effect of using only one depth image and adapt PSSM with a novel detail shadow maps generation scheme to make it suitable for real-time large-scale forest rendering.

PSSM splits the camera view frustum into several partitions using clip planes parallel to the view plane and renders separate shadow maps for each partition. Each object in the scene is thus rendered more than twice per frame. Rendering such a complex scene in real time represents a significant barrier. To reduce rendering overhead, we cull trees against the light frustum to avoid unnecessary rendering before generating each shadow map (the clip method is introduced in Section 4.2). We employ a novel LOD shadow maps generation scheme to achieve a good balance between visual effect, rendering speed and memory cost. We update shadow maps with different frequencies and use different depth image resolution sizes to generate shadows. Trees near the viewer are mapped by high-resolution depth images with high

Fig. 7. An opposite phyllotaxy and its leaf union. (a) Most detail and (b) leaf union.

Fig. 8. LOD series of a black poplar tree. (a) 24,073, (b) 19,302, (c) 12,667, (c) 4720, and (d) 2451.
updating frequencies to cast detailed shadows; trees in the far distance are mapped by low-resolution depth textures with low updating frequencies. Our strategy is as follows: we give the closest shadow map a high image resolution (4096 × 4096) and update it every frame, the second map a middle resolution (2048 × 2048) and update it every two frames, and the last the lowest resolution (512 × 512), updating it every eight frames.

For a PSSM scheme with \( n \) partitions, a naive implementation requires \( n \) shadow rendering passes to generate \( n \) shadow maps. To avoid performance degradation from generating multiple shadow maps, we apply frame buffer arrays to render split shadow maps into different depth textures. An appropriate depth texture is chosen in the fragment shader during the rendering stage.

4.1.2. Shadows smoothing

The generated vegetation shadows still suffer from obvious aliasing artifacts. The irregular thin shapes of highly detailed plant models primarily account for this. As shown in the left column of Fig. 9, the upper subfigure (a) is the shadows of the trees' branches in winter and the lower one (d) is the shadows of the trees' leaves in summer. Several thin branch shadows are disconnected and split into several partitions; the brims of the shadows still contain many jaggies. To reduce these aliasing artifacts, two-dimensional Gaussian filters are employed to smooth shadows in the fragment shader.

Fig. 9 (b), (c) and (e), (f) shows filtering results from using two filters with different sampling sizes, which give the rendering scene a soft-shadowing effect. Fig. 10 shows a close-up view of our rendering result with a 5 × 5 smoothing filter. Fig. 11(a) and (b) shows the rendering results of two different single trees using our system.

4.2. Hardware instancing

We extract \( l \) levels LOD tree models to compress the overall forest geometric complexity. However, most modern GPUs emit primitives into only a limited number \( (n) \) of separate vertex streams. Usually, \( n \) is less than four even for many high performance graphics cards such as Geforce 470 and 560Ti. This means only \( n \) LOD levels can be determined at once. To solve this problem, we use \( \lceil l/n \rceil \) rendering loops when \( l \) is larger than \( n \).

**Instancing in the view frustum:** We separate \( l \) LOD levels into \( \lceil l/n \rceil \) LOD segments. Each LOD segment contains \( n \) continuous LOD levels. The first LOD segment is determined and rendered in
the first loop. Additional loops are used to render the left LOD segments until all LOD levels have been determined. In the first rendering pass, each type tree’s instance data are organized into an array of four-component vectors. Each vector is composed of the world-space position and a randomly rotating degree. To achieve a good performance, we would like to minimize the number of state and texture changes in our application, thus polygons with the same texture are gathered into one group. All groups in a single model share one transform feedback object. Each group is rendered in the way of Fig. 12 to render more than n levels of LOD models. LOD models are determined based on the object’s distance from the viewpoint. We sent the appropriate distance parameter $d_i$ to the geometry shader to determine LOD segment. During the actual rendering pass, each vertex in the vertex shader is multiplied by a rotation matrix to achieve different orientations.

**Instancing in the light frustums**: Instance rendering in each light frustum is similar to that in the view frustum, but with a different strategy. Here we employ only one rendering loop even if there are more than n LOD levels. For each type of trees, only two of the finest models and $n-2$ of simplest models are used in rendering for shadow maps generation.

Fig. 13 shows a detailed flow chart of our rendering system. For each frame, the view frustum is split into subfrustums according to the viewpoint position. Shadow maps for each subfrustum are updated in the light space using our LOD shadow generation scheme. LOD models are selected, rendered and projected in light space for shadowing. The Gaussian filter is implemented in the fragment shader for anti-aliasing during the depth test. Tree fragments lying in shadows are shaded, achieving a realistic rendering result.

### 4.3. Progressive transmission of forest scene data

Fig. 15 shows the organized rendering data stream for effective progressive forest scene data transmission over the Internet.

The data stream generated is transmitted through a TCP/IP network in the client–server mode. A terrain’s DEM takes the highest priority over other transmitted data. The scene management file, containing information such as the model distribution information, is sent second. Simple LOD tree models take priority over complex tree models. The simplest LOD models for all trees are sent first. Instead of sending triangles individually, LOD tree models’ trunks are arranged as a set of triangle strips, where each subsequent triangle reuses two vertices from the previous triangle. Hence, the transmitted data are reduced, and the client’s rendering performance improves with less data sent to the GPU. To generate lightweight files, all files are saved as binary. The rendering system uses the simplest LOD models and invokes a thread to search the downloaded models. When the DEM and simplest models are obtained, the user can see the lowest-resolution forest as quickly as possible (Fig. 14(a)). As more complex models arrive, the forest scene becomes increasingly clear (Fig. 14(b)). After the highest-resolution models have been downloaded, the most complex and realistic forest is rendered (Fig. 14(c)).

Fig. 16 demonstrates overall forest scene data processing. Once a client establishes a connection with the server, its data download module downloads remote forest scene data from the server. Meanwhile, the data import module searches for any downloaded models and parses them before sending them to the GPU for rendering. The rendering module renders the data with real-time shadowing effects and responds to the user’s interaction.

### 5. Experiments and results

Our rendering technique has been implemented in the C++ language with OpenGL. All shaders are written in the OpenGL Shading Language (GLSL). In our experiments, the rendering performance is measured on a PC with an Intel Core i7 2.66 GHz CPU, 3 GB RAM and a NVIDIA GeForce GTX260 graphic card with 896M video memory. All data were transmitted in the client–server mode through a TCP/IP network.

The DEM files are one-channel gray images compressed in JPEG format. A DEM image with a resolution of 1024 x 1024 is...
smaller than 40 kB. The ground textures are three-channel images, also in JPEG format. A ground texture with a resolution of \(2048 \times 2048\) is 844 kB. The scene management file is a binary file of usually no more than 200 kB.

In the LOD extraction phase, we extract eight LOD models for each tree. Constructing LOD models for each tree takes less than 1 min. After compression, the generated lightweight model files range from approximately 100 kB to 700 kB. Table 1 displays each tree sample model's data file size and polygon number in Fig. 1. The full data of the forest scene we built is 9.8 MB, and the model's data count is more than 85% of its total. The storage cost of a forest is \(O(N)\) (\(N\) is the number of tree sample models used in the forest).

Fig. 18 shows a forest with nine types of trees, in which each type of tree has eight LOD levels. This forest has 39.1 MB of data. Table 3 displays each tree sample model's data file size and polygon numbers. Tree models account for 97.3% of the total data. Although these types of forest data are not a problem for users with access to high network bandwidth, it takes several minutes to download the data with a 100 kB/s bandwidth. Users may lose patience with low transmission speeds, requiring the use of our method to deal with the situation.

In the rendering phase, we render all scenes with our LOD shadow maps generation scheme. Table 2 displays detailed rendering performance results, including FPS, tree number in current view frustum, parameters for set LOD distances, tree and polygon numbers in each LOD and polygon numbers in the current view frustum. It contains a pedestrian view of a sparse forest scene and a bird's eye view of a dense forest. Both scenes use four shadow maps and a \(5 \times 5\) Gaussian filter. The forest shown in Fig. 1 contains 7446 trees and is observed from a pedestrian point of view, rendered with real-time realistic shadow effects; 757 trees were rendered in the view frustum, containing more than one and a half million polygons, with an average frame rate of 33.91 FPS. Fig. 17 is a bird's eye view of another forest with over 43,000 highly detailed trees. There are 1028 trees with over 14 million polygons in the view frustum. The scene was rendered with an average frame rate of 19.04 FPS. Both scenes contain five types of trees, and each tree type has four LOD models. The frame rate depends on the number of trees and the geometric complexity in the view frustum. The results show that the rendering is efficient and realistic.
Our hardware instancing method can render over one million highly detailed trees in real-time and with real-time shadows as shown in Fig. 19. The forest contains five tree samples and each sample has eight LOD levels. The performance of Fig. 19 is tested on a NVIDIA GeForce GTX 560 graphics card.

6. Comparison with state-of-the-art works

We compare our method with the three advanced representative forest rendering techniques mentioned in Section 2: dynamic quad stream (DQS) [12], view-dependent pruning (VDP) [17] and
multi-resolution foliage (MRF) [31]. Table 4 summarizes several aspects of this comparison.

Hardware evolution has dramatically enhanced hardware instancing performance. Our method benefits from this because our models’ geometries are nearly unchanged. Geometry modification occurs frequently in the other techniques when the view position moves; therefore, those methods cannot benefit from hardware instancing. This also explains why those techniques cannot handle real-time shadowing well, as they must recalculate the models’ geometry in the light frustums. The recalculation of the geometry creates a considerable overload.

VDP, MRF and our method use small primitives to represent tree foliage. These small primitives cause aliasing when rendering

Table 2
Detailed results of rendering performance.

<table>
<thead>
<tr>
<th>Scene parameter</th>
<th>Fig. 1</th>
<th>Fig. 17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frame per second</td>
<td>33.91</td>
<td>19.04</td>
</tr>
<tr>
<td>Trees in view frustum</td>
<td>757</td>
<td>1028</td>
</tr>
<tr>
<td>LOD distances</td>
<td>899; 2861; 14,201</td>
<td>499; 1861; 4201</td>
</tr>
<tr>
<td>LOD trees</td>
<td>28; 35; 173; 521</td>
<td>78; 121; 199; 630</td>
</tr>
<tr>
<td>LOD polygons</td>
<td>463,130; 206,230; 322,117; 534,996</td>
<td>7,638,686; 2,303,844; 1,894,918; 2,370,769</td>
</tr>
<tr>
<td>Polygons in view frustum</td>
<td>1,526,473</td>
<td>14,208,217</td>
</tr>
</tbody>
</table>

Table 3
Each tree sample model’s data file size and polygon number in Fig. 18 of this paper (kB/polygons).

<table>
<thead>
<tr>
<th>Tree sample</th>
<th>LOD 1</th>
<th>LOD 2</th>
<th>LOD 3</th>
<th>LOD 4</th>
<th>LOD 5</th>
<th>LOD 6</th>
<th>LOD 7</th>
<th>LOD 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Birch</td>
<td>683/6474</td>
<td>589/5835</td>
<td>491/4657</td>
<td>387/3017</td>
<td>246/2068</td>
<td>218/1765</td>
<td>143/1523</td>
<td>105/1102</td>
</tr>
<tr>
<td>Elm</td>
<td>721/8398</td>
<td>651/6542</td>
<td>572/5802</td>
<td>443/2395</td>
<td>233/1974</td>
<td>199/1783</td>
<td>162/1243</td>
<td>102/1065</td>
</tr>
<tr>
<td>Apple_S</td>
<td>699/7065</td>
<td>572/5974</td>
<td>523/5011</td>
<td>433/3917</td>
<td>313/2157</td>
<td>214/1383</td>
<td>178/1061</td>
<td>102/702</td>
</tr>
<tr>
<td>Apple_M</td>
<td>659/6686</td>
<td>512/5174</td>
<td>434/4334</td>
<td>336/2331</td>
<td>236/1917</td>
<td>178/1517</td>
<td>154/1203</td>
<td>116/971</td>
</tr>
<tr>
<td>Apple_M2</td>
<td>698/6582</td>
<td>576/5415</td>
<td>473/4741</td>
<td>394/3234</td>
<td>267/2515</td>
<td>225/1876</td>
<td>163/1653</td>
<td>95/854</td>
</tr>
<tr>
<td>Apple_M3</td>
<td>714/8179</td>
<td>673/6341</td>
<td>561/5322</td>
<td>453/2185</td>
<td>253/1814</td>
<td>197/1581</td>
<td>161/1253</td>
<td>112/1039</td>
</tr>
<tr>
<td>Maple</td>
<td>683/7015</td>
<td>565/5831</td>
<td>509/4611</td>
<td>413/3210</td>
<td>347/2517</td>
<td>243/1873</td>
<td>182/1038</td>
<td>101/709</td>
</tr>
<tr>
<td>Bottle</td>
<td>669/6365</td>
<td>552/5154</td>
<td>465/4214</td>
<td>356/3035</td>
<td>268/1712</td>
<td>195/1531</td>
<td>146/1191</td>
<td>96/871</td>
</tr>
</tbody>
</table>

Fig. 19. A view of a forest which contains 1,038,525 trees. There are 55,923 trees with over 11.6 millions polygons in the view frustum. FPS is 9.38. The resolution is 1920 × 1018 pixels. Experimented on a Geforce GTX 560 graphics card.

Table 4
Comparison with three representative forest-rendering techniques.

<table>
<thead>
<tr>
<th>Method</th>
<th>Instance rendering</th>
<th>Real-time shadow</th>
<th>Aliasing processed</th>
<th>Shading realism</th>
<th>Foliage texture</th>
<th>Geometry fidelity</th>
<th>Memory cost</th>
<th>LOD continuity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQS [12]</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>Low</td>
<td>×</td>
<td>×</td>
<td>High</td>
<td>×</td>
</tr>
<tr>
<td>VDP [17]</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>High</td>
<td>√</td>
<td>√</td>
<td>Low</td>
<td>√</td>
</tr>
<tr>
<td>MRF [31]</td>
<td>×</td>
<td>×</td>
<td>√</td>
<td>Middle</td>
<td>×</td>
<td>√</td>
<td>Middle</td>
<td>√</td>
</tr>
<tr>
<td>Our method</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>High</td>
<td>√</td>
<td>√</td>
<td>Middle</td>
<td>×</td>
</tr>
</tbody>
</table>
trees in the far distance. To address anti-aliasing, MRF and our method merge small leaves with large ones to render trees at far distances.

Polygon-based approaches can achieve high shading realism. MRF does not account for foliage texture, so its rendering is inferior to those of VDP and our method. Geometric fidelity is also high in polygon-based approaches.

In our method, model files range from approximately 100 kB to 700 kB. The GPU memory cost complexity of a forest is O(N) (where N is the number of tree models used in the forest). DQS occupies approximately 15 MB to load a single tree model's textures onto the GPU. VDP's memory cost was the smallest of all methods used. Our method falls in the middle in this aspect. Our method, however, must extract more LODs to reach a continuous model transition.

We present a novel LOD shadow maps generation scheme to render large-scale forest shadows. Our approach not only reduces geometric complexity, but also saves GPU memory. By updating each shadow map with different frequencies and using different sizes for each shadow map according to the distance from the viewer, our shadow map surpasses that of PSSM. The scheme greatly increases the FPS. In addition, we implement anti-aliasing filters in the GPU for efficiency. We present a hardware instancing approach for rendering over one million highly detailed trees in real-time. We use extra rendering passes to perform culling and determining appropriate LOD models. Our approach overcomes a limitation of the latest graphic cards that allows primitives into only a very limited number of separate vertex streams, to render more levels of detail than a straightforward approach would permit. We also present a new leaf phyllotaxy LOD model to match original leaf models with textures, balancing the visual effect and the model complexities. Our LOD strategy is also suitable for progressive rendering through networks because each model is compressed as a whole. The lowest-resolution model is lightweight and can be transmitted and rendered in advance.

Our techniques can be easily applied to computer games, Internet graphics applications and virtual forestry. Realistic large-scale forest scene rendering applies to forest management, where the growth situation of each tree should be visualized before applying management strategies[32]. This technique can also create natural environments for web-based games, in which forests provide immersive effects in many scenarios.

In future work, the following aspects can enhance rendering speed and quality. Using the GPU's tessellation capability in shader model 5, terrain rendering can be more visually pleasing using an adaptive LOD method. Furthermore, fast global illumination methods such as ambient occlusion can be integrated into our rendering.

Acknowledgments

All tree models in this paper are from the output of AMAPGenesis™ software. This work was supported by the National Natural Science Foundation of China with project Nos. 60970093, 60902078, 60872120, and 61172104.

Appendix A. Supplementary material

Supplementary data associated with this article can be found in the online version of 10.1016/j.cag.2012.01.005.

References