Progressive Transmission of Appearance Preserving Octree-Textures

Julien Lacoste[*]
LIUPPA
University of Pau

Camille Perin[†]
LIUPPA
University of Pau

Bruno Jobard[‡]
LIUPPA
University of Pau

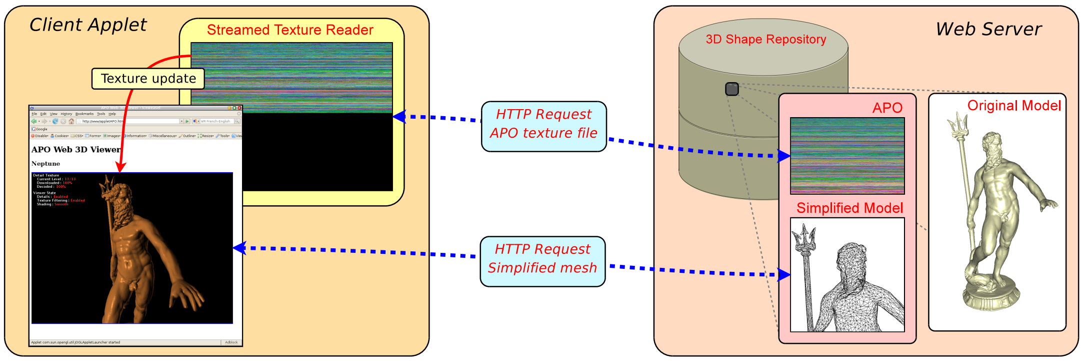


Figure 1: *Architecture of the APO web 3D viewer. The client, a web browser applet, progressively downloads the APO texture, while rendering the coarse model with previously downloaded details. The normal field is encoded in the APO which is an Appearance Preserving Octree-texture. Normal mapping on the simplified mesh is done on the client side by the fragment shader.*

Abstract

The development of shape repositories and 3D databases rises the need of online visualization of 3D objects. The main issue with the remote visualization of large meshes is the transfer latency of the geometric information. The remote viewer requires the transfer of all polygons before allowing object's manipulation. To avoid this latency problem, an approach is to send several levels of details of the same object so that lighter versions can be displayed sooner and replaced with more detailed version later on. This strategy requires more bandwidth, implies abruptly changes in object aspect as the geometry refines as well as a non negligible precomputing time. Since the appearance of a 3D model is more influenced by its normal field than its geometry, we propose a framework in which the object's LOD is replaced with a single simplified mesh with a LOD of appearance. By using *Appearance Preserving Octree-Textures* (**APO**), this appearance LOD is encoded in a unique texture, and the details are progressively downloaded when they are needed. Our APO-based framework achieves a nearly immediate object rendering while details are transmitted and smoothly added to the texture. Scenes keep a low geometry complexity while being displayed at interactive framerate with a maximum of visual details, leading to a better visual quality over bandwith ratio than pure geometric LOD schemes. Our implementation is platform independent, as it uses JOGL and runs on a simple web browser. Furthermore the framework doesn't require processing on the server side during the client rendering.

[*]e-mail:julien.lacoste@univ-pau.fr
[†]e-mail:camille.perin@etud.univ-pau.fr
[‡]e-mail:bruno.jobard@univ-pau.fr

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: Appearance Preserving, Octree-Textures, GPU, Normal mapping, streaming

1 Introduction

Many applications such as 3D databases, shape repositories or virtual environment are now available online, allowing the user to browse 3D models or navigate inside complex 3D scenes. In these online applications, more particularly in 3D databases, an interactive 3D viewer offers a greater capacity to observe meshes than a set of fixed pictures. In order to offer interactive applications, efforts have to be made to adapt the graphic pipeline to the network constraints. The goal is to provide a maximum of visual details with a limited usage of bandwidth, and on some mobile devices with limited hardware capacity.

The classic approach is to use decimated meshes to reduce 3D scenes or objects geometrical complexity. Another solution is to create several levels of detail of the models, then choosing the appropriate resolution according to the network or the client capacity. This solution provides an immediate rendering with lower resolution while more detailed versions of the object are still downloading. This solution suffers from several drawbacks:

- The server needs to compute and store multiple version of the same object, and the client must download all versions before viewing full resolution model, which implies a greater bandwidth consumption.
- With high detailed meshes, made of millions of polygons, it won't be possible to perform interactive framerates.
- The transition between two levels of detail will exhibit abrupt changes, as the two LOD could be very different and not spatially coherent in vertices positions.

The creation of multiresolution version of the models could avoid some of these problems, but the pre-processing step is more complex. Furthermore, the full resolution of a high detailed mesh would still be difficult to render at interactive framerate. These problems are linked to the fact that these LOD schemes rely on the object's geometry.

Since the visual appearance of a 3D model is more influenced by its normal field than its geometry, we can use a reduced geometry while still preserving object's appearance by encoding the details in a normal map. By using Appearance Preserving Octree-textures (APO), the need of a 2D parameterization, which is sometimes difficult to obtain, is avoided, and the hierarchical nature of the octree-texture can be seen as a LOD of appearance.

We propose an APO-based framework in which the original model is replaced by a simplified one along with an APO-texture. The details of the APO could be progressively downloaded. The use of this APO appearance LOD provides many advantages. The scenes keep a low geometry complexity while being displayed with a maximum of visual details, leading to a better visual quality with a minimal bandwith usage. The APO downloading could be stopped when finer details aren't necessary, so the bandwidth consumption is then optimal according to the visual details needed. During the rendering, the server has no processing to do, as the client just needs to the preprocessed simplified model and its APO. Then, it's possible to propose a web based implementation which is platform independent. This web based client uses JOGL and it allows the user to view objects with a simple web browser, with no software installation required.

2 Related Work

To generate several models with different resolution, one can use classical mesh decimation algorithm or methods such as [Cohen et al. 1996]. The smaller model can be used first while other models are downloaded. The switching between two resolutions will cause some visual artifacts, sometimes referred as popping. As each model is fully transmitted, this simple method greatly increases the bandwidth consumption. It also implies to save all the models on the server side.

In [Hoppe 1996], authors presented an algorithm which starts from a coarse mesh, and stores a sequence of vertex splits refining the mesh. This sequence is the invert of the sequence of edge collapse that had been used to decimate the mesh. The new vertices are added one at a time, their positions can be interpolated according to a function of time, and then the popping effect disappears. This algorithm had been improved in [Hoppe 1997] in order to perform view-dependent refinements. Regions outside the view frustum or oriented away from the viewer won't be refined. Authors also proposed to use a screen-space geometric error criterion, in order to split a vertex if the distance the approximate surface and the original one when projected on the screen is greater than a fixed value. These view-dependent criterion allow to transmit only needed information on the network. Algorithm of geometry compression, such as presented in [Alliez and Desbrun 2001] or [Pajarola and Rossignac 2000] could be used for network transmission since the model can be progressively reconstructed.

An alternate architecture to progressive transmission of huge meshes has been proposed in [Koller et al. 2004; Koller and Levoy 2005]. It consists in a lightweight client that manipulates only low resolution meshes. The client sends the rendering environment (lighting conditions, position etc) to a server which computes the rendered pictures. The client then receives and displays this resulting picture. This solution is well suited for privacy, as the end user never owns the full resolution model. Furthermore, the server slightly modify the rendered pictures to prevent their use with an image-based reconstruction algorithm. These modifications, which consist in altering lighting conditions beetween two successive rendering, altering pixel position are not visible nor prejudicial to the user.

For the remote visualization of large meshes, authors of [Rusinkiewicz and Levoy 2001] propose to use a system based on QSplat, a hierarchical structure of bounding spheres with splat rendering. These sphere will be the leaves of the constructed tree structure. During transmission, the hierarchical structure allows to request only visible geometry. When necessary data are not yet available, they are replaced while rendering with a splat corresponding to the parent sphere in the hierarchy. Compared to previous methods, the use of QSplat needs a greater amount of data to be transfered. Furthermore, the rendering of coarse structure levels with their large splats give less details than a simplified mesh.

Previous methods focus on transmitting and compressing the geometry. In [Hadim et al. 2007] authors present a method to transmit the appearance (normals and colors) of the model, using quantization to reduce the amount of data to transfer. The quantization is computed on-the-fly by the server GPU and then transmitted to the client which performs the dequantization. The bandwidth usage is decreased, but there is no level-of-detail on the geometry which must be fully transmitted. The use of appearance preserving simplification methods could prevent this full geometry transmission. In [Lacoste et al. 2007] authors proposed to use octree-textures to perform appearance preservation on simplified meshes. In the following sections we will see how to use the hierarchical nature of the octree to perform a progressive transmission of appearance.

3 Appearance Preserving Octree-Textures

Octree-Textures have first been simultaneously introduced by [(grue) DeBry et al. 2002] and [Benson and Davis 2002]. They proposed to use an octree to encode the color map, which avoids the computation of a 2D parameterization on the input mesh. Furthermore, octree-textures and their intrinsic adaptative structure reduce the waste of memory implied by fixed-resolution 2D or 3D textures. GPU implementation of octree-textures have been presented in [Lefebvre et al. 2005] and [Lefohn et al. 2006]. In [Lacoste et al. 2007] otree-textures are used to perform appearance preservation, by encoding the normal field of a full resolution mesh M in the octree-texture. This texture, called APO, is then mapped on a simplified version m during rendering. We review the APO creation process in the following sections.

3.1 APO Creation

The first step consists in building the octree structure around m. This is done by creating a initial coarse octree with a restricted depth and recursively subdividing each leaf according to an error citerion that controls the precision of the encoded details in the APO, and therefore its size. The criterion, based on the $L^{2,1}$ metric presented in [Cohen-Steiner et al. 2004], measures the normal field variation of M in each leaf. The subdivision stops when the error criterion is fulfilled or when the maximum depth had been reached. Once the octree has been built, each leaf is filled with a representative sample of the normal field of M, obtained by a ray-casting procedure. The octree is used as a space partitionning structure to speed up the ray-casting procedure. As the closest intersection between the ray and M will most of the time be in the starting leaf, the set of cells intersected by the ray must be searched only if no intersection were found in the leaf node. The preprocessing time are then decreased by 40% compared to the results presented in [Lacoste et al. 2007] (see table 1). Normals in internal nodes are computed by averaging the normals of their children.

3.2 APO Encoding in 2D Textures and GPU Mapping

The constructed APO is encoded in a 1D array by enumerating nodes in a breadth first order, as illustrated in figure 2. All siblings are contiguous in the array, so we only need one pointer from each leaf toward its first child. To avoid waste of space, only non

Model	Size	Creation	Texture	FPS
Stan. Dragon	$874k$ / $3k$ tri.	$50s$	$3.6MB$	58
Grog	$1.7M$ / $5k$ tri.	$2m06s$	$12.3MB$	48
Neptune	$4M$ / $15k$ tri.	$3m10s$	$10.3MB$	68
XYZ Dragon	$7.2M$ / $15k$ tri.	$6m43s$	$30.6MB$	46
Statue	$10M$ / $20k$ tri.	$9m50s$	$46MB$	58

Table 1: *Creation time, APO size and rendering FPS with different models, with original and simplified size. The error bound is set to 0.05 and the screen resolution is* 1024×1024.

empty nodes are stored in the array. With the breadth first ordering, the coarser levels of the octree are in the beginning of the array, and deeper levels are in the end. An internal node will be encoded

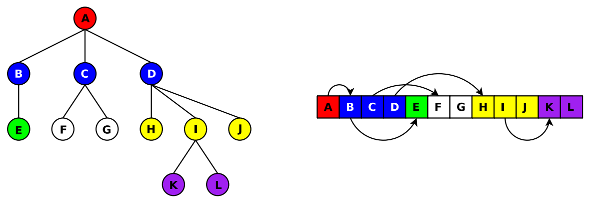


Figure 2: *Encoding an octree in a 1D array by breadth-first ordering. Arrows show the pointer between parent nodes and their first child.*

in eight bytes and a leaf on four bytes. Alpha channels are used as set flags giving information on node children, RGB values are interpreted either as normal value or offset toward the first child. The 1D array is directly written in a 2D texture, in order to make the APO available for the GPU. During the rendering, an octree traversal is performed for each fragment. Starting at the octree's root, which is at index 0 in the array, and recursively accessing the child owning the fragment 3D position until a leaf is reached. In [Lacoste et al. 2007] an adaptive rendering of the APO was performed by computing during its traversal in the fragment shader the projected cell size in screen space. The traversal was stopped when this projected cell size fell below the size of one pixel and therefore avoid sub-pixel precision. We propose an acceleration of the adaptive rendering by computing once for each fragment the depth the APO traversal should stop. By using the GLSL *dFdx* and *dFdy* functions on the interpolated fragment position in world space, we estimate the distance on the mesh between two fragments. A small distance between two neighbour fragments on the mesh means that the object is rather in close-up view and many details should be provided by traversing more deeply the APO. Let $D = min(dFdx(pw), dFdy(pw))$ be the distance on the mesh where pw is the world space position in $[0;1]^3$ of the current fragment. The depth of the traversal is directly computed using the formula $depth_{stop} = \lceil \log_2(1/D) \rceil$. This is the depth of the first node in which the distance D can't fit, which means that D is covered by several cells. To avoid the projection of several nodes on the same fragment the traversal stops at this computed depth.

4 Progressive Transmission of APO

The APO web viewer is based on a client/server architecture (see figure 1). The server is a simple web server which store the precomputed simplified model and its APO. The APO is transmitted to the client as a compressed raw binary file to limit the bandwidth usage, as the texture could size more than 40Mb. The server will also hold an information file, saving the dimensions of the texture, and how many pixels compose each level of the octree.

The client applet must first download the simplified model and the information file. The mesh is generally composed of few polygons, and the informations are written in a small text file, so the downloading time of these two data is negligible. Just as the methods presented in [Koller et al. 2004; Koller and Levoy 2005], the system based on the APO ensures the privacy of the model. Indeed, only the simplified models are transmitted to the client, the original meshes are never transmitted to the end user.

4.1 APO Texture Reconstruction

Once the simplified model had been transmitted the download of the APO starts. The applet can immediately render the simplified model while the first APO levels are read. The client progressively downloads the compressed array containing the APO. The data are decompressed on the fly and are used to fill a buffer. The APO texture is reconstructed by calling OpenGL texture functions with the data read in the buffer. Texture reconstruction can be driven by two policies:

On buffer completed update: Each time a complete octree level had been downloaded a texture update is performed. The number of completed levels is sent to the GPU as a uniform variables, so the fragment program stops when it reaches the current deepest leaf. This solution minimize memory transfer between the CPU and the GPU, as there will be as many textures updates as the number of levels in the APO. But the downloading time of each level is uneven, as there are much more leaves in the deeper levels. For instance, the APO used for figure 3 is made of 13 levels which are dowloaded as follows: the 7 first levels at once (0.2MB) and then each level individually: 8 (0.7MB), 9 (2.4MB), 10 (6.5MB), 11 (11.3MB), 12 (12.5MB) and 13 (8.4MB). The user immediatly sees the coarse details and must wait longer for finer leaves.

Fixed time step update: A texture update is performed at regular fixed intervals. This policy implies a greater memory transfer toward the GPU, but the normal details progressively appears on the mesh, just as a progressive JPEG refinement for 2D pictures. The fragment program must be modified since the details could be in the last complete downloaded level, or one level further. Let these two levels be called j and k with $k = (j+1)$. The octree traversal goes up to the level k. If the pixel at this level is black, the fragment normal hadn't been yet downloaded, so the leaf crossed at level j must be used. To avoid costly conditional instructions on the shader, the used leaf is found by using an interpolation function between the two leaves. The coefficient is computed using RGB values of the pixel at level k in order to equal 0 if the pixel is black, 1 otherwise.

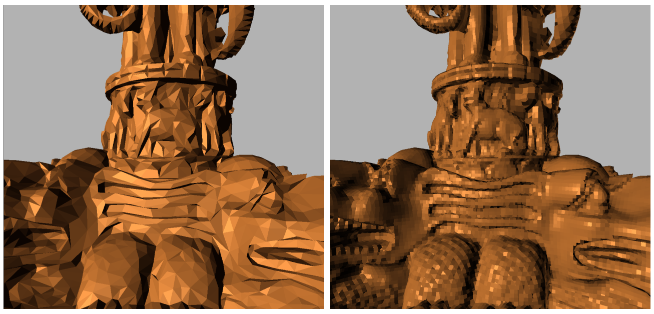
The buffer is cleared after each texture update so that the client remains light in CPU memory. The figure 3 shows the details refinement at different stages of the APO download.

4.2 Data Transmission Control

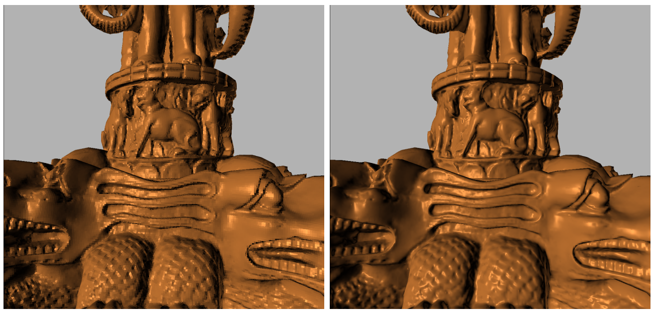
The transmission of the APO could be paused or stopped to avoid unnecessary bandwidth usage. In [Lacoste et al. 2007], the APO structure is used to perform an adaptative rendering. During the octree traversal, if a crossed node is projected in less than one pixel on the screen, the loop stops and the filtered node normal is used for normal mapping. For progressive transmission, a similar method could be used to stop the data transmission when no more detail are necessary. At each frame, the screen-space dimensions of the deepest node is computed. If the projection is smaller than one pixel on the screen, the transmission is stopped and starts again when the object gets closer to the screen.

5 Implementation Details

We have implemented the APO Web viewer as a JOGL applet and tested it on several GPU running from nvidia GeForce 6200 to 8800. The applet is easily integrated in web pages. When the browser access the APO web viewer page, it downloads the application as a jar file which is 350KB large. The parameters of the applications are the url to download mesh, information file and APO on the server are written in the HTML source code. This solution is platform



(a) *Left:* first, the simplified mesh (200KB) is quickly downloaded and displayed while the first levels of details are transmitted in background. *Right:* as soon as the 7 first levels of the APO are received (200KB) the details are progressively mapped on the simplified mesh.



(b) visual results after receiving 2 more levels of details (3MB). Texture filtering is enabled on the right picture.

Figure 3: *APO progressive download. An original 20M triangles model (100MB) is replaced with a 20K triangles simplified mesh (200KB) augmented with a 13 levels of details APO (42MB).*

independent and doesn't require any software installation on the client side. The compressed array is decoded on the fly with the java gzip stream object.
Since the first levels of the APO are nodes that cover wide areas on the simplified model, they don't carry useful details. In order to avoid the loss of visual details, the APO mapping is not used until the first seven level of the APO are available for the client. The first texture update is performed when these levels are downloaded. Before this initial texture creation, the mesh is rendered with no details. On the worst case (a complete octree) the seven levels take less than 3Mb in APO that can weight more than 40Mb, so the waiting for details is negligible compared to the transmission of the full texture.
Java applets generally run with 64Mb of memory, which is enough for a large APO texture and a coarse mesh. When less memory are available, it's not possible for a java applet to allocate a bigger memory. The use of a buffer which is cleared after each texture update makes the applet runnable even on in case of low CPU memory environment.

6 Conclusion

We have proposed a scheme for the network visualization of large meshes, and an implementation on light web browser applet. By using the APO, the client only handles a simplified version of the mesh while appearance details are progressively downloaded. The hierarchical nature of the APO allows to control the bandwidth usage, as the necessary details of a refinement are downloaded only when they are needed on the rendering. Our architecture needs no computation on the server side during client rendering, as the applet only needs to download data. A simple web server holding meshes and their APOs is sufficient for allowing network rendering by several client applets. The creation of APO is fast and must be precomputed only once for each model. On the client side, the applet require few CPU time to decompress the APO texture array, and remains light in memory as the downloaded data are cleared after they have been transmitted to the GPU. For future work we plan to include color details in the texture to render colored objects.

References

ALLIEZ, P., AND DESBRUN, M. 2001. Progressive compression for lossless transmission of triangle meshes. In *SIGGRAPH 2001, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., 195–202.

BENSON, D., AND DAVIS, J. 2002. Octree textures. *ACM Siggraph*.

COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, F., AND WRIGHT, W. 1996. Simplification envelopes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 119–128.

COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM SIGGRAPH*.

(GRUE) DEBRY, D., GIBBS, J., PETTY, D. D., AND ROBINS, N. 2002. Painting and rendering textures on unparameterized models. *ACM Siggraph*.

HADIM, J., BOUBEKEUR, T., RAYNAUD, M., GRANIER, X., AND SCHLICK, C. 2007. On-the-fly appearance quantization on gpu for 3d broadcasting. In *ACM SIGGRAPH Web3D*, ACM Press, ACM.

HOPPE, H. 1996. Progressive meshes. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 99–108.

HOPPE, H. 1997. View-dependent refinement of progressive meshes. *Computer Graphics 31*, Annual Conference Series, 189–198.

KOLLER, D., AND LEVOY, M. 2005. Protecting 3d graphics content. *Commun. ACM 48*, 6, 74–80.

KOLLER, D., TURITZIN, M., LEVOY, M., TARINI, M., CROCCIA, G., CIGNONI, P., AND SCOPIGNO, R. 2004. Protected interactive 3d graphics via remote rendering. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA.

LACOSTE, J., BOUBEKEUR, T., JOBARD, B., AND SCHLICK, C. 2007. Appearance preserving octree-textures. In *GRAPHITE '07: Proceedings*, ACM, New York, NY, USA, 87–93.

LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. *GPU Gem's 2: Octree Textures on the GPU*.

LEFOHN, A. E., KNISS, J., STRZODKA, R., SENGUPTA, S., AND OWENS, J. D. 2006. Glift: Generic, efficient, random-access gpu data structure. *ACM Transaction on Graphics*.

PAJAROLA, R., AND ROSSIGNAC, J. 2000. Compressed progressive meshes. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (/), 79–93.

RUSINKIEWICZ, S., AND LEVOY, M. 2001. Streaming qsplat: a viewer for networked visualization of large, dense models. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 63–68.