View-Dependent Octree Image Rendering

Alexander Zhirkov
MSU Graphics&Media Lab
Moscow, Russia

Abstract

This paper considers view-dependent rendering method for 3D models represented as binary volumetric octree (BVO) together with reference images and corresponding cameras locations. This representation was proposed and accepted to MPEG-4 AFX (Animated Framework eXtension) as Octree Image Node in the Depth-Image Based Representations (DIBR) group. Originally DIBR-objects were assumed to be diffuse. In this paper we propose the techniques capable to represent and render such objects with dynamic lighting, semi-transparency and with image-based view-dependent effects. The dynamic lighting is achieved by reconstruction of normals for each voxels of BVO with subsequent object relighting. The information of semi-transparency is proposed to be stored as alpha channel of images. The correct transparency rendering is achieved by using BVO format capability to provide fast and correct back-to-front voxels traversal. Image-based view-dependence results from the images redundancy used in a special way in order to render voxels color.

Keywords: Octree Image, BVO, TBVO, DIBR, View-dependent Octree Image Rendering, MPEG-4 AFX, Implicit Light Field

1. INTRODUCTION

Because of increasing requirements to detail level and photorealism of 3D object representation and spreading of image-based and range-based model creation methods, there is a great interest in creating representations different from traditional textured polygonal ones. Because of their nature, those representations are mostly based on real photos and 3D points. Let's review the most important and modern in our view representations.

The group of representations that are closer to point-based than to image-based type consists of: *LDI*[1], *QSplats*[2], *Surfels*[3 and *RBF*(Radial Basis Function) Based 3D Objects[4]. In *LDI*(Layered Depth Image) points are organized in a multi-valued depth map. *Qsplat* is a multiresolution point system based on hierarchical structure of spheres of variable size. A more complex model is based on *Surfels*, a hierarchically structured representation of 3D object with the aid of local surface elements whose attributes include color, normal vectors and depth. A spatial cell of the corresponding tree structure contains projections of the object surface structured as 3 LDIs. The initial point representation is transferred to implicit surface representation using *RBF* [4].

Another group contains representations that are closer to image-based than to point-based type: *Plenoptic Modeling* [5], *Lumigraph* [6] and *Light Fields* [7]. The main characteristic of this group is that images are the main part of its representations and the geometry part is either completely absent or is very restricted.

Depth-image representation falls into the intermediate group between the two described above, because it contains a point set that is represented as an image. An important example of this group is *Relief Texture* [8]. The more universal representation in this group is a *DepthImage* format [9] that contains a set of depth maps and corresponding images.

Also we should mention *Light Fields Mapping* representation [10] that consists of specially organized images and polygons. This representation is different from those discussed in this article but also uses view-dependent image-based rendering as in our approach. For more details about the similarity of these approaches see section 3.2.

Ordinary voxel-based representations are positioned separately from point-based and image-based ones. In most of cases they are used to represent 3D volumes or used as auxiliary representations for polygonal representation [11] or for 3D reconstruction [12]. Also there are articles devoted to creating octrees for effective organization of object bounding space, for example [13].

We have already proposed a uniform 3D object representation that contains image-based and voxel-based representations [14]. This representation contains in a separate way color component in images and geometry component in binary-volumetric octree (*BVO*) form. At this moment this format is proposed and accepted to MPEG-4 AFX as the part of *DIBR* [9] and is called *OctreeImage*. This format has an effective compact representation, where images are compressed using existing lossy image compression formats and for *BVO* a specially developed compression scheme is applied with using adaptive arithmetic coding and contextual modeling. A more detailed explanation of node format can be found in its node specifications [15]. Besides compactness, the *OctreeImage* has advantages inherited mostly from volumetric representation, for example a possibility to use single data structure for fast, occlusion-compatible hierarchical warping, gap-free splatting-based rendering and easy level-of-detail selection.

Nevertheless, the separate representation of color and geometry also has some disadvantages. The main of them is connected with implicit relation between color and geometry components and necessity to explicitly determine it at the rendering stage. This task is time consuming, so to preserve real-time rendering it can be done at the preprocessing stage. But preprocessing can be used only for static objects. For streamed objects case there was developed another *OctreeImage* format that uses *Textured BVO* (*TBVO*) instead of *BVO*. The textured channel is a hierarchically compressed stream with explicit information about what camera image corresponds to each voxel [9,15].

But if we want to use view-dependent rendering instead of diffuse one the *TBVO* can not help us to render without preprocessing because it stores only information about diffuse voxels colors. It is view-dependent rendering that is the main topic of this article; hence we will discuss only *OctreeImage* with BVO structure.

2. OCTREE IMAGE FRAMEWORK OVERVIEW

Fig. 1 shows simplified framework for use of *OctreeImage*.

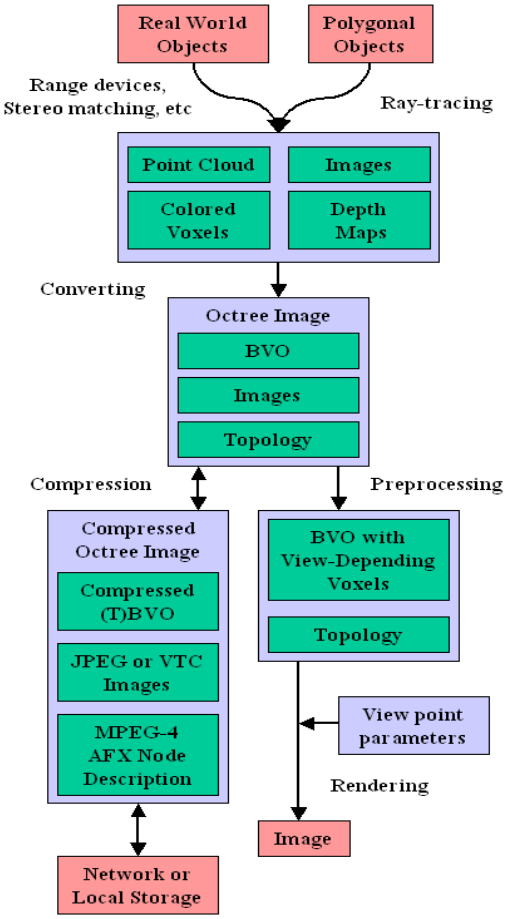


Figure 1 Framework of *OctreeImage* Representation

Such stages as creation of *OctreeImage* from data obtained from synthetic or real world objects and *OctreeImage* compression are described in detail in the previous articles [14,9]. Therefore the main attention in this article will be given to view-dependent image-based rendering with the use of *View-dependent Voxels* representation. The main distinction compared with the original rendering is the generalization of diffuse rendering. Diffuse rendering using fix-colored voxels is replaced with rendering using view-dependent voxels. View-dependence means that not only coordinate projection depends on the point of view of an object but also does the color of voxel.

3. VIEW-DEPENDENT RENDERING

As has been already mentioned, to render of *OctreeImage* with BVO at real-time it is necessary to construct *BVO* that contains view-dependent voxels for every octree node including leaf nodes. For simple diffuse rendering it is enough to assign one color to each node but for view-dependent rendering such attributes as normal vector, transparency and view-dependent voxels can be also used.

After the viewer position and camera parameters are determined, for each voxel color and transparency values are assigned, depending on viewing direction and lighting conditions.

The next stages such as back-to-front octree traversal, splatting and subpixel filtering are similar to ordinary diffuse rendering [14].

3.1 View-dependent voxels

Further we will name voxels, whose projection depends on projection direction view-dependent voxels.

Let's consider three view-dependent rendering types that exist for view-dependent voxels:

- Dynamic lighting rendering
- Semi-transparency rendering
- Image-based view-dependent rendering

The key feature of *OctreeImage* is that geometry and color components are represented essentially separately and in different ways. Moreover it is common some voxels can be visible from multiple cameras and others can be invisible from all the cameras.

Let's discuss all these unclear cases in detail and make clear how to use available geometry and color information in the most efficient way. We will consider methods of explicit computation of colors colors, their normals, semi-transparency and image-based view-dependent information for each voxel. These methods are based on *Implicit Light Field* function representation described below.

3.2 Implicit Light Field

It's logical to start the description with the analysis of *Surface Light Fields* approach [10] dealing with approximation of view-dependent function on the surface. *Surface Light Field* is a 4-dimensional function $f(r,s,\theta,\phi)$ that makes it possible to define the outgoing radiance in every viewing direction for every point on the surface of an object. The first pair of parameters r,s of this function describe the surface location and the second pair of parameters describes the viewing direction:

$$f(r,s,\theta,\phi) = \sum_{k=1}^{K} g_k(r,s) h_k(\theta,\phi) \quad (1)$$

In our approach we will approximate view-dependent function not only on the surface but also on the whole volumetric space. Instead of approximation using a sum of two-dimensional functions we will use a sum of samples f_i (2). These samples are not samples of original function but only their approximates. So we will approximate 5D view-dependent function $f(x,y,z,\theta,\phi)$ and it can be called *Implicit Light Field*:

$$f(x,y,z,\theta,\phi) = \sum_{i=1}^{N} K_i(x,y,z,\theta,\phi) f_i \quad (2)$$

$$\sum_{i=1}^{N} K_i(x,y,z,\theta,\phi) = 1 \quad (3)$$

$$K_i(x,y,z,\theta,\phi) = k_i \delta_{coord}(\rho_{coord}((x,y,z),(x_i,y_i,z_i)))\delta_{dir}(\rho_{dir}((\theta,\phi),(\theta_i,\phi_i))) \quad (4)$$

Functions ρ_{coord}, ρ_{dir} are distance functions in coordinate and viewing direction space. Functions δ_{coord}, δ_{dir} are approximates

of δ- function. Currently we use the following approximation: $\delta_{abc}(x) = \frac{1}{(x^a + b)^c}$.

The *Implicit Light Field* system (2-4) is close to general approach of *Radial Basis Functions* but differs from it because it is necessary to normalize procedure (3) and to make interpolation requirements much softer: $f(x_i, y_i, z_i, \theta_i, \phi_i) \approx f_i$.

If function *f* is determined we can use it for assigning color for each voxel by using voxel center and its viewing direction as f parameters (2). So, if we determine a set of vectors $\{x_i, y_i, z_i, \theta_i, \phi_i, k_i, f_i\}_{i=1}^{N}$ we will be able to view-dependently render the *OctreeImage.* We will denote these vectors as *oriented points* , k_i as the significance of an oriented point and f_i as the point value.

For the purpose of vector determining we will project all non-zero voxels on all cameras to obtain depth maps. Let's stick together the obtained depth maps with reference camera images and after some additional computations (see in 3.2.3 below) we will obtain color, alpha, depth field and normal field at each pixel that is not empty on this generalized image. Then we will convert generalized images to oriented points by attaching viewing direction of corresponding depth maps. Significance coefficients should be chosen to satisfy (3). They should be chosen in reverse proportion to corresponding depth map pixel magnitude gradient (analogous to normal computation, see section 3.4) to increase visual rendering quality. Original generalized image pixel values are assigned to the oriented point values.

The following three subsections consider three particular use cases of the described view-dependent function approximation.

3.3 Image-based View-dependent Rendering

It is possible to achieve any view-dependent effects using image-based view-dependent rendering, but it has some limitation: illumination and other scene conditions relative to *OctreeImage* object are considered static. The more complex view-dependent effect we want to produce, the more images we need to store in *OctreeImage*.

To implement image-based view-dependent rendering it is sufficient to use color as value of view-dependent function f.

By varying parameters of view-dependent function we can vary the view-dependence power:

- If we set ρ_{dir} function to constant we get simple diffuse rendering without view-dpending effects
- The larger view-dependence is achieved by making δ_{coord}, δ_{dir} functions closer to δ-function. And in the limit we can achieve equality: $f(x_i, y_i, z_i, \theta_i, \phi_i) = f_i$.

In practical use high view-dependent power leads to rendering of close view point positions that is not fluent enough. Hence, it is better to use the golden mean between those two modes. The rule can be formulated like this: the more images there are in *OcreeImage* the bigger image-based view-dependence power should be used and as a consequence the higher photo-realism will be achieved.

3.4 Rendering with Semi-transparency

Semi-transparent components can be stored in *OctreeImage* images in alpha-channel and they can represent one more value of view-dependent function.

Using image-based view-dependent rendering it is also possible to achieve effects of semi-transparency, but explicit semi-transparency channel is much more effective.

The alpha channel is interpolated analogously to color components only view-dependent effects are not used: $\rho_{dir} \equiv 1$.

So, for each voxel only one semi-transparency value is determined. This condition speeds up rendering but in principle it is possible to use view-dependent semi-transparency and to represent objects with very impressive properties.

To make correct semi-transparency rendering BVO is traversed in back-to-front order with color mixing.

3.5 Normal Reconstruction and Dynamic Lighting

Dynamic lighting is achieved by using normals obtained from BVO and cameras topology and external to *OctreeImage* format information about light position and material properties.

So the task is to find normal vectors for each voxel.

Firstly, all opacity voxels are projected to all reference cameras using z-buffer to obtain depth images. Then adaptive blurring of depth image is done: the more local depth dispersion the more blur is applied. This type of adaptation makes depth images smoother and simultaneously preserves places of depth discontinuties.

After smoothing depths normals for each pixel with depth, the normals for each voxel are computed:

$$N_i(x,y) = \left(-\frac{D_i(x+1,y) - D_i(x-1,y)}{2}, -\frac{D_i(x,y+1) - D_i(x,y-1)}{2}, 1 \right),$$

$$N_i^*(x,y) = \frac{1}{2 + \sqrt{(N_i(x,y)[0])^2 + (N_i(x,y)[1])^2}} Norm(N_i(x,y)),$$

$$N(x,y,z) = \sum_i (T_i * N_i^*(T^{-1}(x,y,z)[0], T^{-1}(x,y,z)[1]) - T_i((0,0,0)),$$

where:

$D_i(x,y)$ – depth value at pixel (x,y) for the i-th camera

T_i – 4-by-4 transform matrix corresponding to i-th camera

$N_i(x,y)$ – normal vector at pixel (x,y) for the i-th camera

$N_i*(x,y)$ – weighted normal at pixel (x,y) for the i-th camera.

$N(x,y,z)$ – normal vector for the voxel (x,y,z), obtained by weighted summation of the normals for the cameras that 'see' this voxel.

4. RESULTS

The rendering results for all three types of the discussed view-dependent rendering types are shown further.

a.1 a.2

b.1 b.2

Figure 2: Comparison of point-based (a.1,a.2) and image-based view-dependent *OctreeImage* (b.1,b.2) rendering

Figure 2 compares point-based rendering and image-based view-dependent *OctreeImage* rendering with originally similar representation that contains 24 depths images located on the surrounding sphere. It is visible that image-based view-depending rendering visualizes correctly view-depending surface regions.

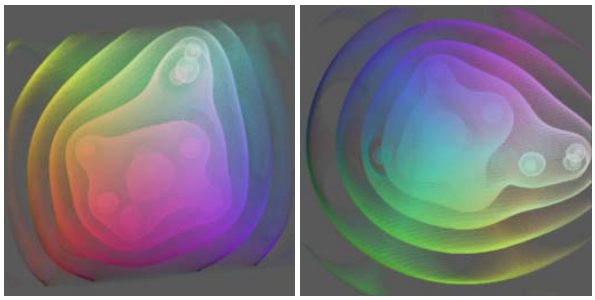
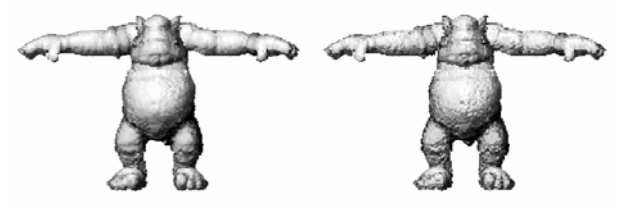
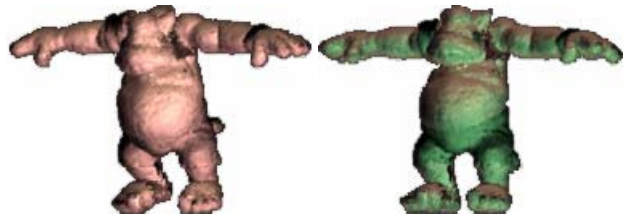


Figure 3: Semi-transparent rendering

Figure 3 is a sample of using semi-transparency *OctreeImage* rendering for multiple iso-surface visualization.



a b



c.1 c.2

Figure 4: Comparison of smoothed (a) and non-smoothed depth normal channel (b) and corresponding to smoothed model Phong illumination *OctreeImage* rendering results (c.1, c.2).

The *OctreeImage* rendering can be completely CPU-based or use hardware accelerated rendering. There are fast algorithms for visualizing *BVO* structure in software, so the speed for both types of rendering is roughly the same. For example, when we use P4-1700 processor in software mode or GeForce-2 video accelerator in hardware-accelerated mode the speed of average objects in 256x256x256 volumetric space, the frame rates are comparable and are more than 30 fps. But the current implementation of the described view-dependent rendering exists only in software mode and is not optimized. Hence the average frame rate is 3 times slower than diffuse rendering.

In conclusion, we can say that the proposed enhanced rendering scheme can significally increase the application domain and photo-realistism of MPEG-4 AFX *OctreeImage* node format.

Acknowledgments

Samsung Advanced Institute of Technology (SAIT) supported this research. I wish to express my gratitude to Leonid Levkovich-Maslyuk for his comments and suggestions. I would like to thank Boris Mihailovich and Anton Konushin who helped me to construct the models.

5. REFERENCES

[1] Steven Gortler, Li-wei He, Richard Szeliski. *Layered Depth Images. SIGGRAPH '98.*

[2] S. Rusinkiewicz and M. Levoy. *"QSplat: A multiresolution point rendering system for large meshes". Proc. of SIGGRAPH'00, pp. 343-352, July 2000.*

[3] H. Pfister, M. Zwicker, J. Baar, and M. Gross, "*Surfels: Surface elements as rendering primitives", Proc. of SIGGRAPH'00, pp. 335-342, July 2000.*

[4] Carr et al. "*Reconstruction and Representation of 3D Objects With Radial Basis Functions", Siggraph 2001*

[5] L. McMillan and G. Bishop, "*Plenoptic modeling: An image-based rendering system", Proc. of SIGGRAPH'95, pp. 39–46, August 1995.*

[6] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. "*The Lumigraph", Proc. of SIGGRAPH'96, pp. 43–54, August 1996.*

[7] M. Levoy and P. Hanrahan. *"Light field rendering, Proc. Of SIGGRAPH'96", pp. 31-42, August 1996.*

[8] Manuel M. Oliveira, Gary Bishop, McAllister. "*Relief Textures Mapping". SIGGRAPH 2000.*

[9] Yuri Bayakovski, Leonid Levkovich-Maslyuk, Alexey Ignatenko, Anton Konushin, Dmitri Timasov, Alexander Zhirkov, Mahnjin Han, In Kyu Park. *"Depth Image-Based Representation for Static and Animated 3D Objects", IEEE International Conference on Image Processing, USA, New-York, 2002.*

[10] Wei-Chao Chen, Jean-Yves Bouguet, Michael H. Chu, Radek Grzeszczuk. *"Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light" Fields. Proc. of SIGGRAPH'01, pp. 43–54, August 1996.*

[11] Rocchini, P. Cignoni, C. Montani, and R. Scopigno, *"The Marching Intersections algorithm for merging range images". Technical Report B4-61-00, I.E.I. -C.N.R., Pisa, Italy, June 2000*

[12] S. M. Seitz and C. R. Dyer. *"Photorealistic scene reconstruction by voxel coloring". Int. J. of Computer Vision, 35(2):151–173, 1999.*

[13] C.H. Chien, J.K. Aggarwal, *"Computation of Volume/Surface Octrees from Contours and Silhouettes of Multiple Views", IEEE Computer, 1986, pages 250-255*

[14] A.Zhirkov. *"Binary Volumetric Octree Representation for Image Based Rendering". Graphicon'01. September of 2001.*

[15] ISO/IEC JTC1/SC29/WG11 N5397: FDAM of ISO/IEC 14496-16, *Awaji Island, Japan, December 2002.*

About the author

Alexander Zhirkov is a Ph.D. student of the Keldysh Institute of Applied Mathematics, Russian Academy of Sciences. His research interests are in real time photorealistic image-based rendering, video and sound compression, fractal and multiscale analysis, object and speech recognition, artificial neural networks, human-centered interfaces, chaos and synergetic.