Real-Time Per-Pixel Viewpoint Interpolation

D. Porquet
D. Ghazanfarpour
University of Limoges - France
porquet@msi.unilim.fr
ghazanfarpour@unilim.fr

J.M. Dischler
University of Strasbourg - France
dischler@dpt-info.u-strasbg.fr

Abstract

Real-time rendering of complex scenes is a crucial problem in computer graphics. In this paper we present a simple and efficient real-time (high frame rate) rendering method in which the computational cost is almost independent of the scene geometric complexity. The main advantages of our method compared to other height field warping approaches such as Relief Texture Mapping, is that the scene has no restriction concerning the nature of its geometry. This method can be considered as an hybrid geometry and image based representation of the scene, viewed along a 3D camera path. The final image in an arbitrary viewpoint on this path is rendered using a very small number of images corresponding to the reference viewpoints. Real-time rendering is performed using the pixels shaders functionality of current graphics hardware.

Keywords : real time rendering, viewpoints interpolation, graphics hardware, interactive walkthrough.

1. Introduction

Rendering at real-time rates of complex scenes composed of huge sets of triangles has always been an important research in computer graphics. Many different approaches have been proposed during the past decades that either consist in optimizing visibility computations (such as for instance the occlusion maps [19, 21]), or using textures and impostors [2, 1, 7, 11, 14, 18], or pure image-based techniques [4, 5, 8, 12, 13]. Hybrid methods combining image- and geometry based approaches [6, 10, 15, 16, 17, 20] have also known some noticeable success but all of these methods still raise several problems like blurring, poping or they have some restrictions concerning the nature of the scene geometry. In fact, currently there is no general method that allows us to render very complex scenes at real-time rates independently of the scene geometric complexity and with a high degree of rendering quality.

In this paper, we propose a new hybrid geometry and image based method to provide a solution to this problem. Our method is suitable for high quality real-time (high frame rate) rendering of scenes with an arbitrary geometric complexity. It has only one restriction: the viewpoints have to move along a specified 3D path on which we define key view points used for per-pixel interpolation. The rendering is performed fully in hardware and the computation cost is almost independent of the scene geometric complexity. In addition memory consumption remains very low. The main difference with previous techniques is that we do neither deform the reference images nor do we apply any interpolation among pixel colors. We also do not replace parts of the geometry by texture maps. Instead, we retrieve the most accurate point of the "real scene" using the key-images to perform the reconstruction. Since we do not need to interpolate pixel colors, our method requires only a very small set of reference images. Because our technique reconstructs on each pixel the "real scene surface", we further can relight the scene on the fly with standard hardware based shading models. Essentialy, our method consists of two passes :

- In the first pass, the scene surface is reconstructed according to a viewpoint lying on the user-specified path.
- In the second pass, the scene image is computed using a small set of reference images. We apply our real-time per-pixel view interpolation technique to reconstruct this image (see section 3).

Thus, our approach is based on a pre-computed data structure that consists of a set of z-buffers. For each pixel, we approximate the depth variation due to the camera motion along the specified path with a small set of linear varying intervals stored in the form of textures associated to the z-buffers. The pre-computed reference images are independent of the surface reconstruction and can be obtained with any software or hardware.

The paper is organized as follows. We briefly present the related works in section two. The scene surface reconstruction, for a given viewpoint, is presented in section three.

The image based appearance reconstruction of the scene is explained in section four. Real-time implementation is presented in section five. The results of our implementation are presented in section six, followed by the conclusions and future works in section 7.

2. Related works

Our method is related to Image-Based Rendering by Warping (IBRW) methods [5, 10, 11, 13, 14], which create 3D scenes by deforming or interpolating several 2D images with depth information. These methods generally limit the viewpoints allowed : they must lie on a fixed view cell or near the reference viewpoints. If the viewpoint exceed the limits, cracks and holes appears [5], due to unseen parts of the scene. In a last resort, holes can be filled while splatting a pixel of the input image on several pixels of the output image [10, 14], but the quality of the final image decreases. To deal with these disocclusion artifacts, hybrid techniques, such as Layered Depth Images [3, 10] capture parallax effects in a special data structure. This structure store the invisible part of the scene, and is used to fill the holes. But this operation increases the rendering complexity, furthermore, not all of the holes are filled. Impostors [1, 2, 6, 7, 15, 18], replace distant geometry with textured planes. Nevertheless, the replaced geometry must not generate parallax effects, otherwise the same holes and cracks problems appear in the reconstructed view. Another way to reconstruct the image of the scene, is the image based approach. Lightfields and Lumigraph [8, 12] methods generate new images of the scene by filtering and interpolating large sets of reference images. One main problem is to interpolate correctly these images to avoid blurring effects. The Light fields rely on oversampling while the Lumigraph uses an approximation of the geometry. Due to the huge amount of reference images, the main issue of these methods is data compression. Furthermore, as the sets of source images are taken from real scene photographs, relighting is very difficult.

3. Surface reconstruction

The aim of our method is the reconstruction of the per-pixel depth information, which correspond to the scene surface, for an arbitrary viewpoint lying on the user-specified path. That is, for a particular viewpoint, we want to retreive the surface of the scene. But instead of projecting the whole scene onto the screen using a Z-buffer as commonly done with geometric approach or instead of simply warping an image as done with IBRW methods, we determine these depths using a compressed representation of the pixels depth variation along the path. The path corresponds to a recorded camera motion between two 3D points, i.e. a fixed set of observer positions and viewing directions as shown in figure 1.

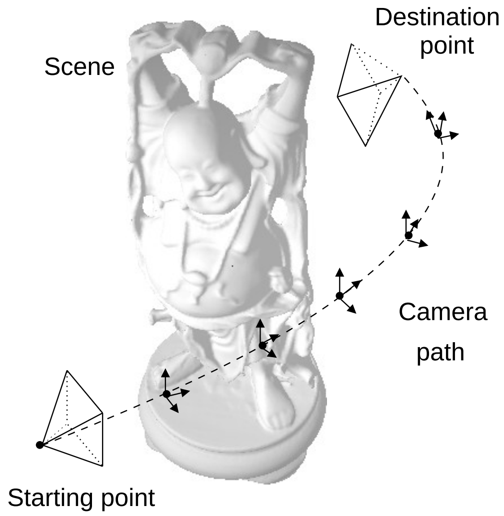


Figure 1. Camera path

In our implementation, we fix the viewing directions to the scene center so that we can retrieve all the camera locations by means of a simple linear interpolation in spherical coordinates between the starting and the destination points. However, other types of motions could be used, such as splines for example. We simply chose this approach for that sake of implementation simplicity.

3.1. Retreiving the scene surface

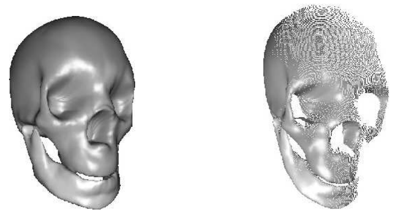


Figure 2. Scene surface from a viewpoint. Left, an image of the scene. Right, the scene surface points retreived and observed with another point of view

We consider that the scene surface viewed throughout a camera corresponds to the set of scene points projected onto each pixel of the camera plane. (see figure 2) For a specific viewpoint lying on the path, we obtain the scene surface via the z-buffer of its image. (See figure 3) So, we have to make the conventional rendering of the full geometric scene mesh to obtain the depth information for each pixels. These depth values are used to retrieve the surface as shown in the figure 3 : considering an isolated pixel on the screen, at coordinates (u_p, v_p), we create the 3D point $P_c = (u_p, v_p, z)$, in camera space coordinates. The z component is the depth of the pixel, directly taken from the z-buffer. To finally get the point in scene space, we simply have to transform P_c with the inverted camera transformation matrix, that is $P_s = M^{-1}.P_c$.

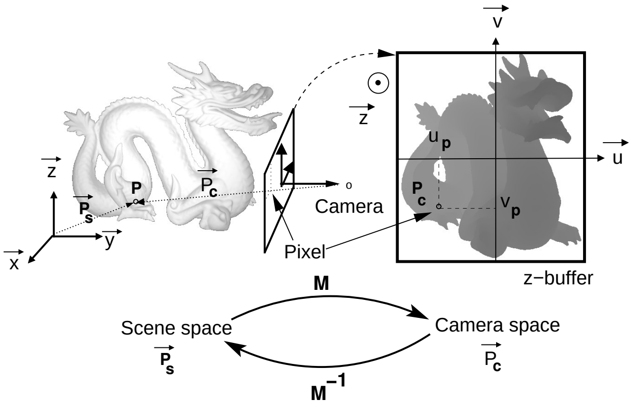


Figure 3. Retreiving the surface of the scene using the z-buffer of a viewpoint.

3.2. Per-pixel depth variation

Our method is based on the per-pixel depth variation while moving the viewpoint along the path. To obtain this information, we render the scene and store the depth component of all pixels during the viewpoint evolution along the path. (i.e., we store the z-buffers). To do this, we discretize the path with an arbitrary number of viewpoints, the more we have, the better will be the reconstruction. Considering the depth variations of one isolated pixel, the aim of our method is to approximate its variation with a small set of linear varying intervals (figure 4), which will be explained in the next section.

3.3. Depth variation segmentation

The number of linear varying intervals is fixed to control the amount of memory required for the recontruction (see section 3.4). So, the problem is to find the most suitable 2D segments which best approximate the real depths variations. In figure 4, we have represented the camera position on the abscissa and the depth of the pixel on the ordinates axis.

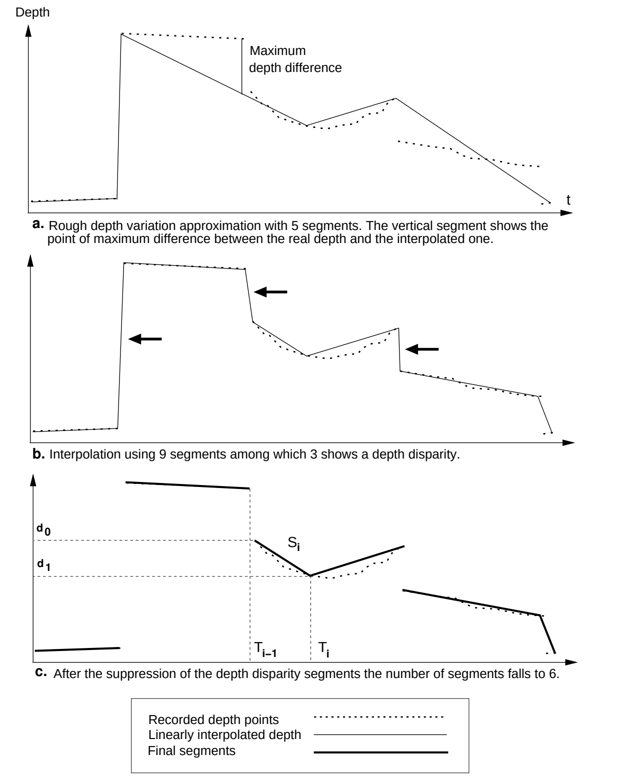


Figure 4. Segmentation of the depth variation for a pixel. In abscissa, the viewpoint coordinate along the path. In ordinates, the pixel depth. The real real recorded depths correspond to the black points.

Our algorithm is adaptive : we first start the segmentation by one segment joining the depth of the starting point and the last depth. Then, the segment is subdivided at the point which subsists the greatest depth difference between the linearly interpolated depth and the real depth (figure 4.a).

The process is repeated and we choose among the segments list the one with the greatest depth difference in order to subdivide it. When the fixed number of segments is reached, we stop the subdivisions.

During the segmentation, we delete all the segments showing a depth disparity, that is the segments which links two consecutive points (4.b). The depth disparity corresponds, in the z-buffer of the scene, to a brutal depth variation commonly due to an edge of the scene, as shown in figure 5.

Finally, the data structure stores a sorted list of segments,

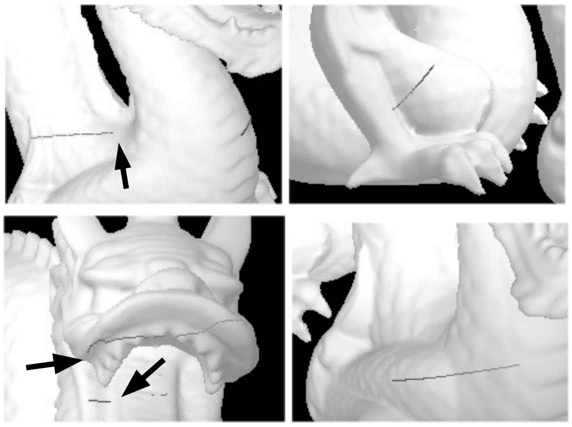


Figure 5. Scene points projected onto a pixel, when the camera moves along the path. The left images shows the depth disparities.

for each pixel. One segment correspond to a motion intervall, and consist of the following informations (see figure 4.c) :

- a starting depth d_0
- an ending depth d_1
- the abscissa T that define the ends of the segment.

The segments are stored following the T order, that is, on the figure 4.c, from left to the right.

3.4. Depth recovering

To recover the depth of a pixel for a given time t on the path, we just have to find the first segment S_i which ends with $T_i > t$, starting the lookup at the beginning of the pixel sorted list. Given the segment data, we obtain, along with the previous segment of the list, the depth d with a simple linear interpolation :

$$d = d_0 + \frac{t - T_i}{T_i - T_{i-1}}(d_1 - d_0) \tag{1}$$

Finally, we are able to recover the depth of each pixel, for each moment of the motion. The full scene surface reconstruction is performed while finding all the depths of all pixels of the camera. To complete the scene reconstruction for a given viewpoint, we now have to retrieve the scene visual appearance.

4. Depth appearance reconstruction

With the per-pixel depth information, i.e. the scene surface, for a given viewpoint, the scene appearance reconstruction is straightforward. We use a small set of depth images as reference viewpoints. Each of these viewpoints covers a part of the scene surface. These viewpoints are external to the path, and positioned by the user. For each pixel, we retrieve its object space 3D point P, as shown figure 6.

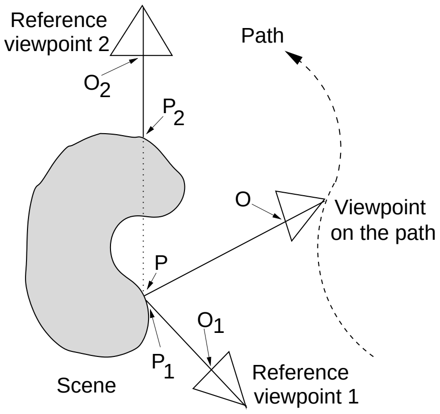


Figure 6. Scene appearance recontruction for one particular pixel (O) based on two reference viewpoints.

This point is projected on the camera plane of each reference viewpoint. We obtain a set of 2D points lying on the camera plane of each viewpoint (O_1 and O_2 in this example). Each of these points correspond to a point located on the scene surface (P_1 and P_2). Among these pixels, the best is the one for which its associated scene space point is nearest to P. In this example (figure 6), the associated 3D point of P viewed throughout the reference viewpoint 2 , is the point P_2 ; throughout the viewpoint 1, it is P_1. As shown for this pixel, the viewpoint 1 is the best viewpoint for this part of the surface : the distance between P_1 and P is smaller than the distance between P_2 and P. So we color the pixel O with the color of the pixel O_1.

Finally, we have to do these comparisons for all of the pixels of the camera to obtain the final pixels colors displayed on the screen.

5. Real-time rendering

Real-time rendering is achieved by using the latest hardware functionality. Pixels Shaders functionality allow us to do expensive computations, (such as matrix transformation) and memory bandwidth expensive operations (like texture lookup), per pixel, in real-time . Our method is implemented in two passes : the first pass reconstructs the surface of the scene (see part 3) and the second pass, does the color computation of the image (see part 4). Presently, we have implemented the second pass fully in hardware, using the NVidia fragment shaders OpenGL extensions on a GeForce FX 5600. The surface reconstruction step outputs a depth texture, representing the reconstructed surface. This texture is used for the second step which colors those depth pixels, and ends the rendering of the scene.

5.1. Surface color restitution

Because of the limitation of the fragment shader size, we can't use an arbitrary number of viewpoints to reconstruct the scene appearance. In practice, we have implemented two methods :

- The first one is suitable if the scene viewed from the path is fully covered by using three viewpoints. The rendering is done in one pass with the direct implementation of the algorithm described above.
- The other one is used if we must use more than three viewpoints. Then the rendering process is done in n passes, n corresponding to the number of reference viewpoints.

In the second case, one pass consists in computing the distance between the surface point P and the 3D point stemming of the viewpoint (see figure 6) and output this distance into the z-buffer of the video card, along with the retrieved color. This operation is repeated for each of the reference viewpoints. The z-buffer comparison function allows only the colors associated with a depth value smaller than the previous stored one to be displayed. So, the rendered color is the one associated with the smallest distance, which is what we desire.

In fact, because of the z-buffer precision, we do not use directly the distance between the two 3D points. Instead, we reproject the two points into the camera plane, and compute the distance in 2D.

Figure 7 shows a reconstructed scene at different moments of the interpolation : in this example we have used three reference viewpoints, they are highlighted with different colors : red for the first viewpoint, green for the second and blue for the third one. Thus we can see which viewpoint is used to color a pixel.

5.2. Depth disparity problems

During our experimentation, we have noticed some problems. The figure 9.a shows those artifacts. They result from the pixel lookup when projecting P (Figure 6) onto the reference viewpoints. The problem is detailed on figure 8. P is projected on the reference viewpoint onto a pixel giving the P2 point instead of P1 because of the discretization of the camera plane. To solve this problem, we compute the depth of the pixel stemming from the reference viewpoint, with

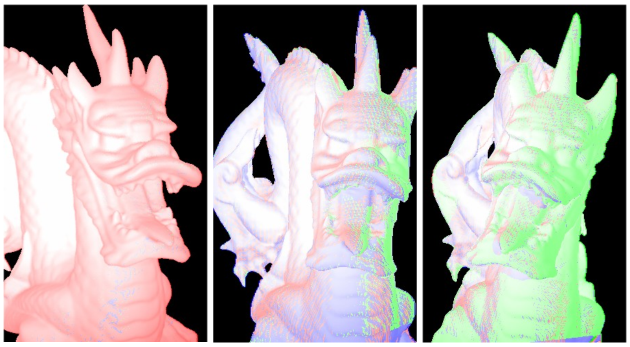


Figure 7. Pixels origin during the real-time rendering

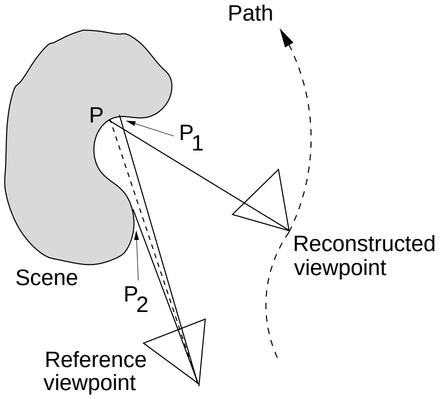
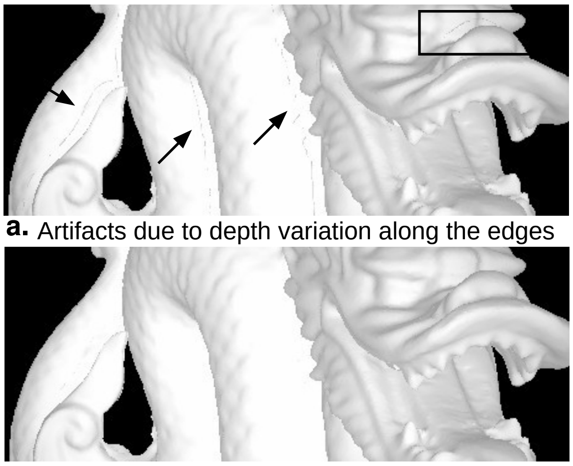


Figure 8. Discretization problem

a bilinear interpolation of its four adjacent pixels. The result is shown on the figure 9.b.

6. Results

Figure 10 shows results for a scene composed of the Buddha mesh of the Stanford Computer Graphics Laboratory. All our tests were done with a screen resolution of 512x512 pixels. Using this resolution, the pre-computation time for our structure is about 30 seconds. For these three meshes we use 15 segments to approximate the depth variation with no visual differences between the real depth, as shown in the attached video clip. With 15 segments, the memory used is about 20 Mb. Table 1 shows some details on the differents meshs we use during our experimentations. The table 2 shows the average frame rate we obtain using our method for a different number of reference images. The second column shows the results without the depth segmentation. The last column shows the results using bilinear depth filtering and depths segmentation. Our rendering time is independent of the scene complexity and heavily depends



a. Artifacts due to depth variation along the edges

b. With depth bilinear filtering

Figure 9. Depth filtering

on the graphics card bandwidth between main memory and texture memory. The second column shows this fact : with one viewpoint, there is a small number of texture lookups per pixel and the frame rate is high. With more than three viewpoint, the number of texture lookup decrease the rendering speed below 10 frames per second, but we used a low end graphics card and these results would be far higher with the latest hardware. The depth filtering decreases the rendering performances for the same reasons. The last line show the results using an ATI Radeon 9700 Pro and DirectX 9 with Pixel Shaders 2.0. This card gives the best frame rates, but is limited in terms of shader length and we can't use the depth filtering. Moreover we have to use the multi-pass algorithm detailed in section 5.1 which decreases the precision of the interpolation. Figure 11 shows the visual difference between our reconstructed image and the real one. The memory used is about 20 Mb for 15 segments, 13 Mb for 10 segments, 10 Mb for 8 segments and 6 Mb for 5 segments. Compared to video compression, our method is able to relight the scene in real-time : we just have to store a normals texture along with the z-buffer, for each viewpoint.

Model	triangles	Size (Mb)	FPS
Armadillo	345944	30,5	4.5
Buddha	1087716	95	2
Dragon	871414	76,5	2.5

Table 1. Conventional rendering

# vp.	No compr.	15 seg.	+ depth filter.
1	80	16	15
2	22	16	10
3	14	13	6
3 (ATI)	80	30	N.A.
4 (ATI)	75	25	N.A.

Table 2. Frame rates using our method

7. Conclusions and future work

We have presented a system for the real time visualization of complex static scenes, along a fixed path. We use hybrid geometric - image based methods to obtain interactive frame rates, in the order of 10-25 images per second. The main advantage of our method is to provide such a frame rate independently of the scene geometric complexity, at a low memory cost, yet providing high quality images (e.g. nearly undistinguishable from usual z-buffer rendering). To our knowledge, this is the first method achieving this level of quality. We have also described a new technique to reconstruct the scene image, based on a very small set of reference images. Presently, the first pass (reconstruction of the surface) of our algorithm is done in software. However, an hardware implementation with Pixels Shaders should be feasible for this part of the algorithm. This is one of the objectives of our future works. An error measurement between the real depth and the approximated depth would be straightforward to provide. It will permit the user to control the quality of the reconstructed surface as a function of the memory requirement. Another objective is to extend the segmentation algorithm to reconstruct the scene for arbitrary viewpoints, with no predefined path.

References

[1] D. G. Aliaga. Visualization of complex models using dynamic texture-based simplification. *IEEE Visualization '96*, 1996.

[2] D. G. Aliaga and A. A. Lastra. Smooth transitions in texture-based simplification. *Computers and Graphics*, 22(7):71–81, January 1998.

[3] C. Chang, G. Bishop, and A. Lastra. Ldi tree : A hierarchical representation for image-based rendering. *Proc. SIGGRAPH 99*, pages 291–298, August 1999.

[4] S. Chen. Quicktime vr - an image-based approach to virtual environment naviguation. *Proc. SIGGRAPH 95*, pages 29–38, August 1995.

[5] S. Chen and L. Williams. View interpolation for image synthesis. *Proc. SIGGRAPH 93*, pages 279–288, August 1993.

[6] P. Debevec, C. Taylor, and J. Malik. Modeling and rendering architecture from photographs : A hybrid geometry- and

image-based approach. *Proc. SIGGRAPH 96*, pages 11–20, August 1996.

[7] X. Décoret, F. Durand, F.-X. Sillion, and J. Dorsey. Billboard clouds for extreme model simplification. *Proc. SIGGRAPH 2003*, August 2003.

[8] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. *Proc. SIGGRAPH 96*, pages 43–54, August 1996.

[9] E. Hidalgo and R. Hubbold. Interactive rendering of complex and animated scenes : Hybrid geometric - image based rendering. *Computer Graphics Forum (Eurographics 2002)*, 2002.

[10] Y. Horry, K. Anjyo, and K. Arai. Tour into the picture: using a spidery mesh interface to make animation from single image. *Proc. SIGGRAPH 97*, pages 225–232, 1997.

[11] S. Jeschke and M. Wimmer. Textured depth meshes for real-time rendering of arbitrary scenes. *Eurographics Workshop on Rendering*, 2002.

[12] M. Levoy and P. Hanrahan. Light field rendering. *Proc. SIGGRAPH 96*, pages 33–42, 1996.

[13] L. McMillan. An image based approach to three-dimensional computer graphics. *Technical report, Ph.D. Dissertation, UNC Computer Science TR97-013*, 1999.

[14] M. Oliveira, G. B. G, and D. McAllister. Relief texture mapping. *Proc. SIGGRAPH 2000*, pages 359–368, 2000.

[15] W. Paulo, C. Maciel, and P. Shirley. Visual navigation of large environments using textured clusters. *Symposium on Interactive 3D Graphics.*, pages 95–102, 1995.

[16] J. Shade, S. Gortler, L. w. He, and R. Szeliski. Layered depth images. *Proc. SIGGRAPH 98*, pages 231–242, July 1998.

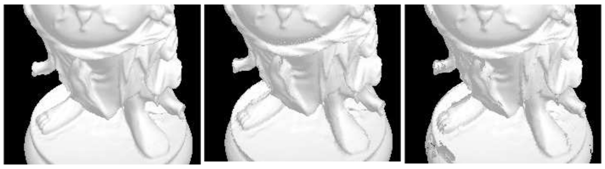
[17] L. Wang, X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H. Shum. View-dependent displacement mapping. *Proc. SIGGRAPH 2003*, 2003.

[18] M. Wimmer, P. Wonka, and F. Sillion. Point-based impostors for real-time visualization. *Eurographics Workshop on Rendering*, 2001.

[19] P. Wonka and D. Schmalstieg. Occluder shadows for fast walkthroughs of urban environments. *Computer Graphics Forum (Eurographics '99)*, pages 51–60, 1999.

[20] S. Yamakasi, R. Sagawa, H. Kawasaki, K. Ikeuchi, and M. Sakauchi. Microfacet billboarding. *Proc. of the 13th Eurographics Workshop on Rendering.*, pages 175–186, 2002.

[21] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. *Proc. SIGGRAPH 97*, pages 77–88, 1997.



a) The real mesh b) 12 seg. compression c) 8 seg. compression

Figure 10. results

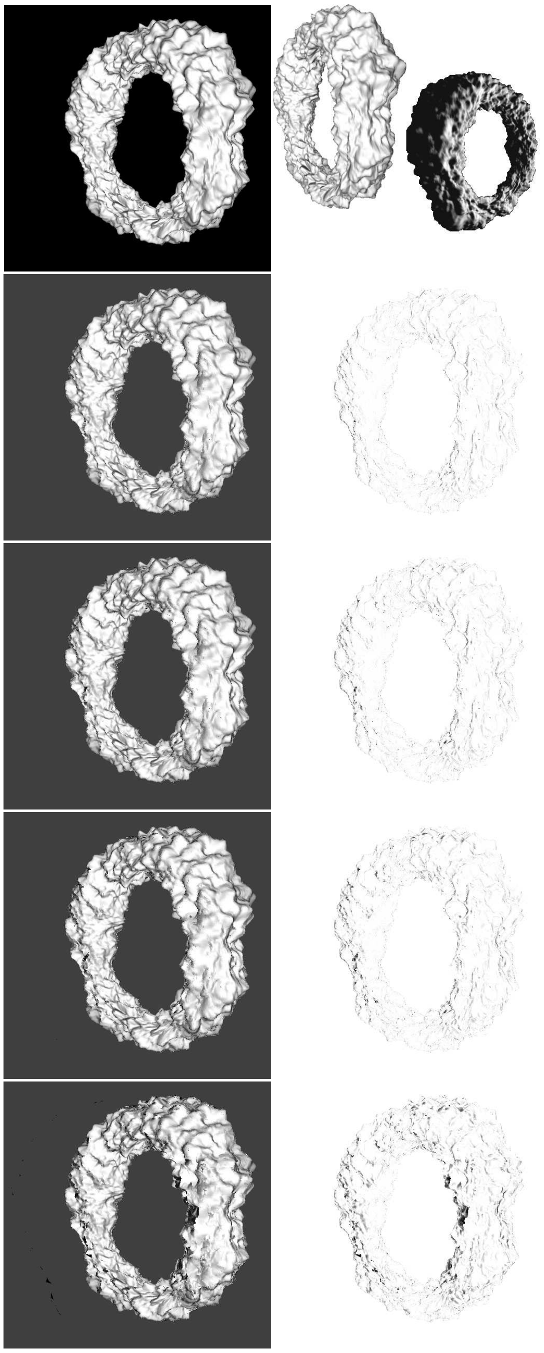


Figure 11. Compressions comparison. First line, left : the real mesh ; right : the starting and ending viewpoints. Next lines : results using different compression rates : 15, 10, 8 and 5 segments, top to bottom. Right row shows the visual difference with the real mesh.