Realtime Navigation in Highly Complex 3D-Scenes Using JPEG Compression

Nicolas Cuntz
cuntz@uni-paderborn.de
University of Paderborn

Jan Klein
janklein@uni-paderborn.de
University of Paderborn

Jens Krokowski
kroko@uni-paderborn.de
University of Paderborn

Prof. Dr. Friedhelm Meyer auf der Heide
Student Research Project of Nicolas Cuntz
GI-Department 4

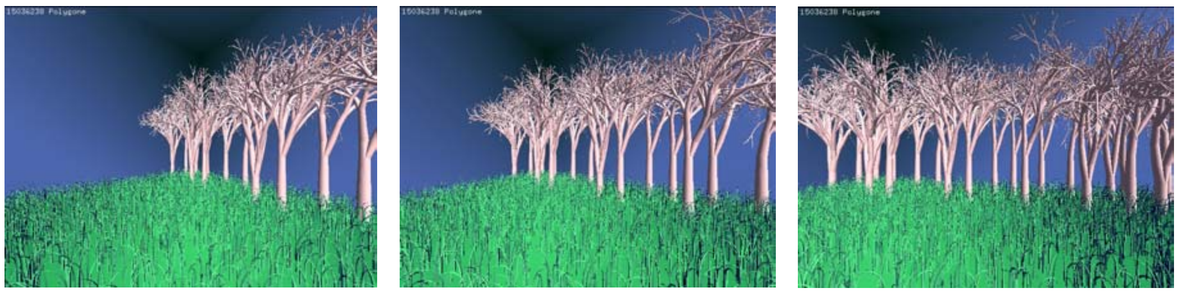

Figure 1: Landscape scene (≈ 15 million polygons) can be viewed interactively using JPEG compression

Abstract
We present a new and easy to use framework for navigating through scenes of arbitrary complexity and topology. In the preprocessing, images for discrete viewpoints and viewing directions are rendered and stored on an external volume. During navigation *each* image can be displayed within a very short time by loading it from the volume. For acceleration, our prefetching strategy loads possibly needed images for the next few frames if the viewer takes a break. The measurements show that we achieve interactive frame rates, whereby the difference between the minimal and maximal display time is very small. Our system works well with scenes modelled by polygons, but also digital photos can easily be used for describing a 3D scene.

1. Introduction

Despite the use of advanced graphics hardware, real-time navigation in highly complex virtual environments is one of the fundamental problems in computer graphics because the demands on image quality and details increase exceptionally fast. A virtual viewer moves on a plane (e.g. a meadow) or in a 3D space and wants to look in an arbitrary direction. In order to guarantee a smooth navigation, at least 20 images per second have continuously to be displayed on the screen.
In contrast to geometric rendering methods, e.g., see [Coh02, Pfi00, Rus00], the running time of Image Based Rendering, e.g., see [Kan99], is independent of the scene complexity. We make use of this advantage to realize a smooth navigation even in arbitrarily complex and detailed scenes. The scenes can be represented by 2D images or photos as usual with Image Based Rendering approaches. But our system can also use scenes modelled by polygons.

2. Related Work

In the following, we compare polygon based rendering methods with image based rendering approaches and show their advantages and drawbacks. The section concludes with a comparison to our method.

Polygon Based Rendering

Conventional rendering based on the well-known z-buffer algorithm [Cat74] were developed to visualize scenes described by polygons. The drawback of the easy to implement z-buffer algorithm, a running time linear in the number of polygons and the area of all projected polygons lead to developing more sophisticated algorithms, like occlusion culling algorithms [Coh02], level-of-detail and multiresolution methods [Hec94] or point sampling approaches [Pfi00, Rus00]. Often textures in combination with polygon models are used to minimize the number of polygons and to achieve a high image quality [Pfi00, Zwi01]. The advantages of all these algorithms are that the viewer can move to an arbitrary position in the scene and that there are no perspective distortions. But unfortunately, the running time is – depending on the used algorithm – linear or sublinear in the number of (visible) polygons or objects of the scene. Furthermore, the running time of many approaches strongly depends on the spatial arrangement of the polygons and the current camera position or the chosen perspective. Another drawback is that the running time can change rapidly from one frame to the next.

Image Based Rendering

Image based methods, e.g., see [Kan99], do not use polygons but photos or computer generated images for the navigation through a scene. Often new images are computed by interpolating two or more existing 2D images. In contrast to the methods mentioned above, they have the great advantage that they are independent of the scene complexity and thus they can visualize scenes of arbitrarily high quality with constant running time. The disadvantages however are the high memory consumption and possible perspective distortions. A special kind of methods in the area of image based rendering deals with spherical or cylindrical panorama pictures [Kan99] that can be looked at by viewer (e.g. Apple QuickTime® VR [Che95]). But these viewers are not optimized for walkthroughs in 3D scenes. So on the one hand, the well known methods [Lip80, Mil92, Che95] do not support an automatic generation of the image material or movie out of a polygon model. This means the scene has already to be given in 2D images. On the other hand, there are no prefetching policies [Zac02] that can load images in advance while the viewer pauses in order to decrease the display time. Especially for very complex scenes whose image material cannot completely be stored in main memory, prefetching policies as developed for our approach are of great advantage.

Based on these considerations, we have developed a system that does not show these disadvantages and displays images in realtime independent of the scene complexity. Instead of composite panorama pictures, we use single images that show the corresponding view for a certain viewpoint and viewing direction. The single images can be compressed and stored on an external volume. In the preprocessing, they can be automatically generated from a polygon model. But also a realistic environment can easily be recorded by a digital camera and used as a scene (see section 4).

3. Realtime Navigation Using JPEG Compression

The system consists of two main components, a *preprocessor* and a *navigation program*. In the preprocessor numerous views of a complex scene are rendered, based on a geometrical model. The computed images are compressed with the JPEG algorithm and are stored on a volume. Preprocessing is not necessary if the scene is described by digital photos stored in the JPEG format.

During navigation the images are loaded and decompressed. Only those views of the scene that were produced before in the preprocessor can be displayed. Since the number of pictures is limited by the capacity of the external volume, we must also restrict the number of positions and viewing directions.

A data structure (the *position grid*), that is integrated in both the preprocessor and the navigation program, defines a suitable set of points lying on a plane. During the navigation, the user can rotate or move forward and backwards on this plane.

Positions and viewing directions

We are looking for a regular alignment of the positions, because the distance between two (view-)points should be the same for all points to realize a smooth navigation. So the points can be located on a grid consisting of squares (4 viewing directions) or of isosceles triangles (6 directions), see Figure 2. It would be a problem to align the positions on a grid with 5 or more than 6 directions. In this case no regular grid could be found: The number of points would become infinite (e.g., see Figure 2 (right)).

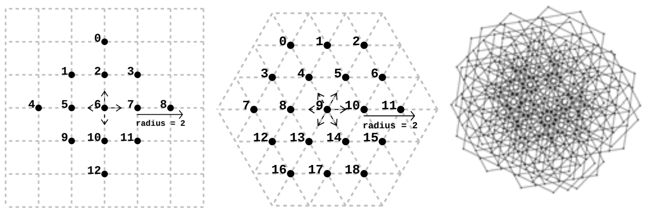

Figure 2: Position grids with different numbers of directions. From left to right: 4, 6, and 5 directions. For 5 directions the number of points would become infinite, if we did not abort the recursive generation of the grid.

However the restriction to 4 or 6 viewing directions would make a smooth rotation impossible. To overcome this problem, the following approximation is performed: We allow more than 4 or 6 viewing directions per point, but if the user makes a step leading to a point *p* that is not contained in the position grid, then a grid point with minimum distance to *p* will be taken for displaying the appropriate image. This raises an error, which subjectively is not as disagreeable as the restriction to 4 or 6 directions. Now if the user goes on, the old point *p* is taken as origin. The program permits only an even number of directions so that a backward motion leads exactly in opposite direction.

Image Generation

The size as well as the density of the grid and the number of directions can be adjusted in the preprocessor program. So the grid can be adapted to arbitrarily extended scenes. After the selection of the parameters the computation of all pictures is performed. Note that this computation could be strongly accelerated by sophisticated algorithms, e.g., see [Coh02, Hec94, Pfi00]. Until now our program only uses the conventional z-buffer algorithm for rendering the single pictures.

Prefetching

The navigation program has to load and decompress the appropriate pictures depending on the inputs of the user. In order to accelerate the handling of a large amount of data, a prefetching algorithm is developed, that loads certain images in advance if the viewer takes a break. Figure 3 shows the points of the position grid that can be reached in three time units starting from a certain position. Thereby, we must count forward steps and each rotation. Backward steps were not considered here.

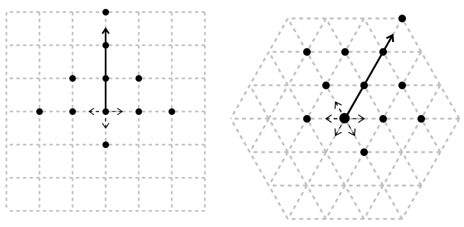
Fig. 3: Reachable points

This strategy can be improved by the observation that users usually turn around their own axis or directly run several steps forward. Therefore, it is more effective to prefetch only the pictures on a straight line in view direction rather than for all the points showed in Figure 3. Since the memory consumption for *decompressed* images is relatively high (an example: In the case of a resolution of 640x480 pixels and 24 bits of color information, each individual bitmapped picture needs 900 KB. Thus over 10 MB of image data would have to be stored per viewpoint in main memory.), the appropriate images are loaded into a buffer in the compressed status and are decompressed when needed.

Real Photographs

As mentioned above, our system is able to use real photographs for navigation. So everybody can create his own walkthrough for any real environment. With a digital camera and a tripod, images can easily and cheaply be produced. If a position grid with 4 directions combined with 12 viewing directions is chosen, all necessary photos of, e.g., a room can be taken within a very short time. Figure 2 shows the order of the camera positions; small deviations of the camera alignment are tolerable. The images are used by the navigation program of our system without any postprocessing. Only the number of viewpoints and viewing directions has to be set. A distance of about 2 meters between the grid points already gives a very good spatial impression. Furthermore, 12 viewing directions per point (rotation angle of 30°) allow an acceptable rotation (see Figure 4).

4. Implementation and Results

The project is completely written in C++ and platform independent, so that it can be compiled and executed on Linux and on Windows. Therefore, the drawing procedures make use of OpenGL [Sil02] in conjunction with GLUT [Kil02]. The JPEG compression was realized with the libjpeg library [Ljp02]. In the following, we present measurements – performed on an Athlon XP 1500, 512 MB, GeForce 3 Ti 200 – for two different scenes:

Scene 1 (polygon based model)

The first scene (see Figure 1) represents a landscape with trees and grass modelled by 15,036,238 polygons. We have developed a program for generating natural looking trees depending on different parameters, e.g., size, precision, angle etc. The degree of the image compression was adjusted in such a way that a quality difference is subjectively not perceptible. With a resolution of 640x480 pixels we get file sizes of approx. 50 KB. The image generation for a grid with 19 positions (Figure 2, centre) and 20 viewing directions per point took about 30 minutes.

Scene 2 (real photographs)

The second scene (see Figure 4) was composed from manually taken photographs. Although the graphic data of the whole scene consumes only about 9 MB on the external volume, the user can look into 12 directions and move to 13 different positions, which cover a surface of 6x6 m in the real world. We needed about 25 minutes in order to take all 156 photos.

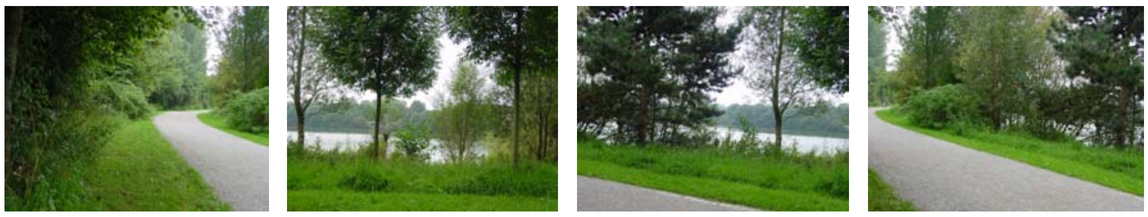
Figure 4: Photos taken with 30° rotations, the average size of one JPEG picture amounts to 60 KB (640x480)

Measurements

The measurements (see Figure 5) confirm that the time for loading, decompressing and displaying the images is largely independent of the scene complexity: For the measurements of scene 1 (Figure 5 (left)), we took a route for which different views of the scene must be loaded. In some pictures we see trees modelled by a lot of polygons and other pictures show only grass modelled by fewer polygons. Nevertheless, the time for loading and displaying the images is constant. Note that although the images are loaded from CD-ROM (8x), 20 frames per second are displayed. The small variations in the diagrams can be explained by the fact that the file size of the images differs a little bit. Figure 5 (right) shows that also for displaying the photographs only constant time is needed, which is similar to that in the left diagram (20 fps). Besides, it turned out that the time for loading can be decreased only slightly with faster drives. This can be explained by the fact that decompressing the images is more expensive than loading them, if the corresponding files are relatively small (about 50 KB). A faster CPU would increase the speed in this case. With our prefetching strategy, only the loading but not decompressing of the files can be accelerated. So the most acceleration by our prefetching can be achieved, if the access to the data is relatively slow (e.g., when using a network or CD-ROM) or if large files have to be loaded (> 500 KB).

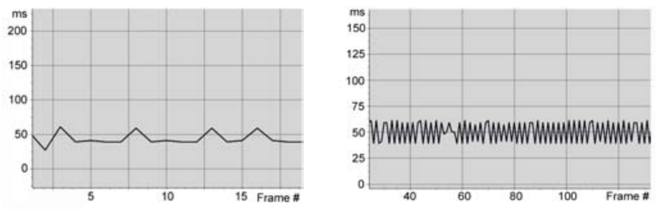

Figure 5: Time (y-axis) for loading and decompressing single pictures (x-axis). Left: Scene 1, 15,036,238 polygons, drive: Mitsumi 8x CD-ROM. Right: Scene 2, real photographs, IBM 40 GB, 7200 RPM

5. Conclusion and Future Work

We developed an approach for realtime navigation in scenes of arbitrary complex topology. In both scenes an interactive walkthrough with about 20 fps is possible. Our prefetching policy accelerates our basic algorithm, especially when the images are stored on external drive with a slow access time. Although the result is satisfying, numerous extensions and improvements are possible. It would be conceivable, e.g., to add collision detection in order to avoid walking through trees or other objects. Furthermore, instead of displaying the entire scene with our method, one can combine it with geometrical procedures in order to save memory and to avoid the restriction to fixed viewpoints by our position grid. Moreover, an interpolation between the precomputed pictures can improve the navigation quality.

References

[Cat74] E. Catmull. A Subdivision Algorithm for Computer Display of Curved Surfaces, Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, 1974

[Che95] S. E. Chen. QuickTime® VR – An Image-Based Approach to Virtual Environment Navigation. In *Computer Graphics (Proc. SIGGRAPH 1995 Conference Proc.)*, pages 29–39, 1995

[Coh02] D. Cohen-Or, Y. Chrysanthou, C. Silva and F. Durand. A Survey of Visibility for Walkthrough Applications. To appear in: *The IEEE Transactions on Visualization and Computer Graphics (TVCG02)*, 2002

[Hec94] P. Heckbert and M. Garland. Multiresolution Modeling for Fast Rendering. In *Proc. Graphics Interface 1994*, pages 43–50, 1994

[Kan99] S. B. Kang. A survey of image-based rendering techniques. *In Videometrics VI (SPIE International Symposium on Electronic Imaging: Science and Technology), Vol. 3641*, pages 2–16, 1999

[Kil02] M. Kilgard, ported to Win32 by N. Robins, http://www.xmission.com/~nate/glut.html

[Lip80] A. Lippman. Movie Maps: An Application of the Optical Videodisk to Computer Graphics. In *Computer Graphics (Proc. SIGGRAPH 1980 Conference Proc.)*, pages 32–43, 1980

[Ljp02] Free library for JPEG image compression, http://www.ijg.org

[Mil92] G. Miller, E. Hoffert, S. E. Chen, E. Patterson, D. Blackketter, S. Rubin, S. A. Applin, D. Yim, and J. Hanan. The Virtual Museum: Interactive 3D Navigation of a Multimedia Database. In *The Journal of Visualization and Computer Animation, (3)*, pages 183–197, 1992

[Pfi00] H. Pfister, M. Zwicker, J. van Baar and M. Gross. Surfels: Surface Elements as Rendering Primitives. In *Computer Graphics (Proc. SIGGRAPH 2000 Conference Proc.)*, pages 335–342, 2000

[Rus00] S. Rusinkiewicz and M. Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Computer Graphics (Proc. SIGGRAPH 2000 Conference Proc.)*, pages 343–352, 2000

[Sil02] Silicon Graphics Inc., Open Graphics Library, http://www.opengl.org/

[Zac02] C. Zach and K. Karner. Prefetching Policies For Remote Walkthroughs. In *Technical report 2002-010, VRVis Research Center*, 2002

[Zwi01] M. Zwicker, H. Pfister, J. van Baar and M. Gross. Surface Splatting. In *Computer Graphics (Proc. SIGGRAPH 2001 Conference Proc.)*, pages 371–378, 2001