Visualisation system based on image warping and Delaunay triangulation of the space[1]

Przemysław Kozankiewicz
Institute of Computer Science,
Warsaw University of Technology,
ul. Nowowiejska 15/19,
00-665 Warsaw, Poland
pkozanki@ii.pw.edu.pl

ABSTRACT

In this article we present a visualisation algorithm based on image warping. A source data for the algorithm is a set of images with colour and depth information rendered for the observer located in a number of reference positions. For each position there are 6 images rendered for the directions of axes forming the "image cube". The algorithm computes a 3D Delaunay triangulation on these positions and divides the space inside the convex hull into tetrahedra. During the visualisation process the observer situated in one tetrahedron sees warped images from the vertices of this tetrahedron. We also describe the automatic algorithm for selection of new reference positions to improve the quality and remove "holes". For both algorithms we utilize 3D hardware acceleration.

Keywords
image-based rendering, warping, visualisation, Delaunay triangulation

1. INTRODUCTION

Visualisation subsystem in virtual reality systems is one of the most important elements. Its efficiency and quality affect the amount and the quality of the impressions of the user. Typical visualisation subsystem creates images by drawing textured triangles.

In our project we investigate the possibility of using image-based rendering in the visualisation process instead of triangle meshes. Image-based rendering relies on describing objects and scene with point sets. These points are stored in a rectangular table (screen memory). Each cell contains the information about the point colour and the distance from the observer. Warping is a method of transforming such images and it allows, for example, to compute the image for new observer position or orientation. In many cases the warping operation does not need to compute 3D positions of points, thus it is fast.

In many applications, eg. flight simulators, real-time visualisation is needed. In the case of complex scenes, such systems require very efficient graphical systems. Pre-rendering of scenes for a number of different observer positions and then using warping to interpolate these rendered images during visualisation is a way to make the system independent of the scene complexity.

In our paper we investigate the possibility of using warping of pre-rendered images for the observer placed in the nodes of the 3D mesh created by a Delaunay triangulation.

In the next section we mention the previous works in the field of image warping and searching for reference positions. In section 3 we describe the source data for our visualisation algorithm. In the next two sections we present the algorithm for visualisation and

[1]This paper was funded by The State Committee for Scientific Research in years 2003-2004 under the grant no. 4 T11C 036 25

WSCG SHORT Communication papers proceedings
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.

the algorithm for automatic choosing of reference positions. In section 6 we discuss our implementation.

2. PREVIOUS WORK

2.1. Warping

There are few methods developed so far for scene and objects representation based on images or point sets.

Two similar methods, Light field [Levoy96a] and Lumigraph [Gortl96a] are used to describe single objects and are based on a discretisation of the plenoptic function (see [Adels91a]). Plenoptic function $c = f(x,y,z,\theta,\phi)$, describes colour c visible from the point (x,y,z) in the direction (θ,ϕ). Light field and Lumigraph are 4-dimensional subsets of this 5-dimensional function and describe the light leaving some bounded region.

Another two interesting representations of objects are LDI structures and relief textures. LDI structure (see [Shade98a]) is a set of images with depth that represent the subsequent "surface layers". Visualisation in such system consists of independent rendering of all layers from back to front. Relief textures (described in [Olive00a]) are the textures supplemented with the depth of texels. Their advantage is the possibility to exploit hardware graphics acceleration during visualisation.

Mark in paper [Mark99a] presented an algorithm for accelerating the rendering of an interactive animation. With this algorithm system generates only some frames with conventional rendering (for example one reference frame every second) and in-between frames are computed by warping reference frames. His algorithm requires to foresee the future observer position for reference frames.

Point set-based object representation was first described in paper [Levoy85a]. The advantage of such approach is the possibility of visualising objects with complex geometry, hard to represent with other methods. During the last few years a lot of work has been done in the field of interactive rendering large point sets with maintaining good visualisation quality (see [Rusin00a, Gross98a, Pfist00a]).

2.2. Search for the reference positions

There are only few papers concerning warping that describe the selection of new reference positions, most of papers leave this problem to the user. In Mark's paper [Mark99a] algorithm needs reference frames that are close to the future positions of the observer. The algorithm computes them by an extrapolation of the observer movement.

Stürzlinger [Stürz99a] shown the algorithm for finding a minimal set of reference positions that allow to capture all object surfaces. This algorithm is iterative and is based on the simulated annealing method. Unfortunately, this algorithm does not optimise the quality of captured surfaces and considers only the surface visibility.

The problem stated in our paper is similar to the *next best view* (NBV) problem. The NBV algorithms are used for automatic 3D scanning of objects. Their goal is to find the next optimal scanner position that allows to retrieve as much as possible of information about surfaces which were invisible before.

Most of the papers concern automatic recognition with laser scanner of objects put in a bounded volume and use volumetric methods [Massi98a, Wong98a]. After each scan, these methods update information in voxels describing that they are empty, not empty or not yet seen. In some cases (see [Massi98a]) when texture reconstruction quality is also needed, information about it is additionally stored in voxels. In these papers the selection of a new reference position is performed by checking all potential positions from some semi-regular grid. This is possible because of the bounded volume.

Other approach to optimisation is presented in the paper [Werne00a]. The authors utilise a linear programming to choose an optimal small set of reference images among large set of source images.

The paper [Klein00] describes a system of a robot control that examines and automatically reconstructs real scene. For each reference position the system creates a structure called *view-cube* made from 6 images with depth information, created by projecting the scene surfaces on the faces of a cube. Optimisation in this system consists in maximisation of an objective function expressing an amount of unknown surfaces visible from a chosen position. Positions for which the function is computed are taken from a semi-regular grid. During optimisation the 3D hardware acceleration is exploited for visibility tests.

The paper [Seque96a] also describes the robot control system but surfaces are represented as triangle meshes. After each scanning the mesh is updated. New positions are computed with the following algorithm. For each hole in the mesh the system points out the area in which the hole is visible. The algorithm tries to find a non-empty union of the largest

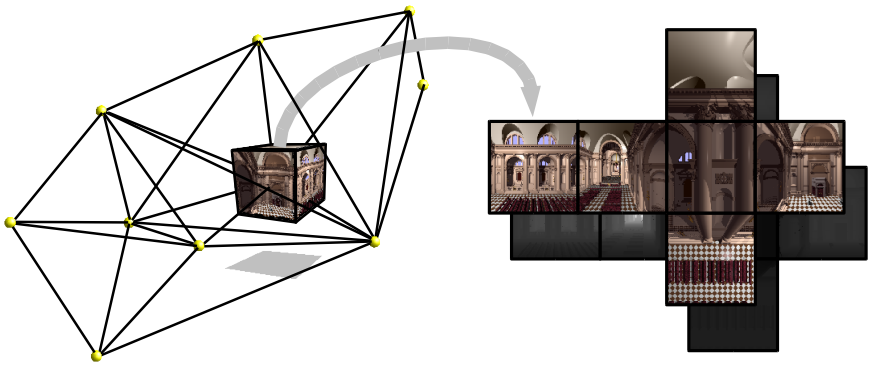

Figure 1. Example images for a single position and a space partition.

number of these areas. New reference position is put in this union set.

The paper [Fleis99a] presents a method of selecting the reference positions for the purposes of image-based rendering algorithms. The goal of this algorithm is to find such reference positions inside the defined volume where the surfaces are visible with suitable quality. The algorithm works in two phases. In the first phase, it chooses a base set of cameras on the boundary of that volume. In the second phase, it chooses a small subset of cameras fulfilling the quality requirements.

3. SOURCE IMAGES

In this section we present the source data for our algorithm.

The algorithm takes, as a source data, images created for given observer positions. For a single position the input is six square images with the depth for directions **-x**, **+x**, **-y**, **+y**, **-z**, **+z**. The field of view for each image is 90 degrees. Thus the images describe all surfaces visible from this position. These images may be put on faces of a cube centred around the position.

Images for a certain position are used for rendering when an observer is inside a tetrahedron containing that position as a vertex.

At the beginning we checked the possibility of using a uniform grid of $n_x \times n_y \times n_z$ cuboidal cells. During visualisation, the observer that was inside a particular cell, saw warped images from 8 vertices of this cell. It turned out that the source data for such grid was, first, redundant (if observer stayed far from objects, then the images from neighbouring positions contained similar information) and, second, it was hard to reveal this redundancy during the compression.

One possibility was an adaptive control of grid density by first creating a scarce grid and then making it more dense depending on the distance to objects surfaces.

However, we decided that it is better to use grid defined by the Delaunay triangulation for unrestricted reference positions. Such operation divides the space into tetrahedra. The visualisation system, for a set observer position, has to visualise images from only 4 reference positions. It speeds up the frame rate twice comparing to the grid of cuboids.

4. VISUALISATION

For a given set of positions, before the rendering, the visualisation algorithm computes three-dimensional Delaunay triangulation of these positions. We assume, that the observer moves only inside the convex hull of the positions. So the observer is inside (or on the boundary of) some tetrahedron. The visualisation algorithm displays warped images from four vertices of this tetrahedron ($6 * 4 = 24$ images).

Each source image is divided in square tiles (eg. 16 tiles), which are displayed independently. The purpose is to speed-up the algorithm by not displaying non-visible tiles.

All pixels of the images are stored in the format RGBZ, where RGB is the colour and Z is the z-depth. After reading a source image, the algorithm computes the space coordinates of all pixels (from the image camera parameters and z-depth of pixels).

Suppose that the scene is a convex solid and the camera is placed inside a tetrahedron. The algorithm is warping the images from vertices of this tetrahedron. Then we are certain, that any surface fragment is visible on at least one source image (see [Mark99a]). Un-

fortunately, in our case, we cannot have such assumption about the scene and we may expect that some "holes" appear in the visualised surfaces.

In a general case, the holes cannot be eliminated completely, but by thickening the mesh (by inserting new positions) the hole sizes may be reduced. In the next section we present the automatic algorithm for selection of new reference positions.

During the visualisation, the observer may pass from one tetrahedron to another. At this moment the algorithm switches instantly the displayed images. To prevent such unpleasant jumps in displayed images, the program needs to visualise additional images from 4 mesh vertices of tetrahedra adjacent by faces. Smooth transitions are obtained by using alpha-blending and transparent display of images from additional vertices. When the observer approaches the face common with the other tetrahedron, the α coefficient of the images for the other tetrahedron vertex increases (it reaches value 1 on the face). When the observer moves away from that face, the α coefficient decreases (it reaches value 0 on the other faces). The value of α coefficient is computed in the following way:

$$\alpha = \min_{\substack{i=1,2,3 \\ d_0+d_i>0}} \left\{ \frac{d_i}{d_0+d_i} \right\},$$

where d_0 is the distance to the face common with the considered adjacent tetrahedron, d_1, d_2, d_3 are the distances to the other faces.

5. AUTOMATIC ALGORITHM FOR INSERTING REFERENCE POSITIONS

In the case of simple scenes, if a small number of reference positions is required, they may be interactively given by a user. For example, the user moves the camera in the virtual scene and sees a hole in the surface or an insufficient texture quality, then he can decide to add a new reference point for this position and start the rendering process of source images for this position. In the case of complex scenes or in the case of greater number of reference positions, an automatic algorithm for inserting reference positions is useful. We describe such algorithm in this section.

Notice that by inserting positions we may change not only the tetrahedron containing that position, but also tetrahedra in some neighbourhood. That operation may worsen the visualisation quality inside these tetrahedra. However, by inserting sufficiently large number of positions, the sizes of tetrahedra decrease and thus the possible hole sizes also decrease. Thus the visualisation quality improves when the number of insertions is sufficiently large.

The algorithm for selecting reference positions consists of two phases executed alternately:

A. In this phase, for each tetrahedron, the algorithm searches for an "optimal" point, for which the observer sees the most of the holes and the space invisible on source images. The optimality of such point is defined by an objective function presented later in this section.

B. In this phase, the algorithm inserts a single position to the tetrahedron with the greater value of an objective function. After this operation the Delaunay triangulation is updated and source images for inserted position are computed.

In phase B, the Delaunay triangulation is modified in the neighbourhood of the inserted position. Owing to this, the objective function must be computed only for new tetrahedra in this neighbourhood. Values for other tetrahedra may be cached and do not need to be computed again in phase A.

Objective function is defined in the following way. For each point X inside the considered tetrahedron, the algorithm computes k_{un} – the number of pixels visible during visualisation that are unknown ("holes") or space visible through them is undefined on all reference images (thus it can be not empty). The value of the objective function is given by the following equation:

$$D_X = (k_{\text{un}} - k_{\text{dist}}) * V,$$

where V is the volume of the tetrahedron, k_{dist} is the number of pixels closer than given depth threshold $dist_{\text{min}}$. The component k_{dist} does not allow to approach surfaces too close. Without this component, an observer moving towards the surface would see this surface as separate pixels with holes around them. In this case the objective function would grow constantly while getting closer to the surface. Factor V is required to prioritize tetrahedra based on their volume when two tetrahedra have similar number of "hole" pixels ($k_{\text{un}} - k_{\text{dist}}$).

To speed-up the computation of the objective function value we use the OpenGL library and the 3D hardware

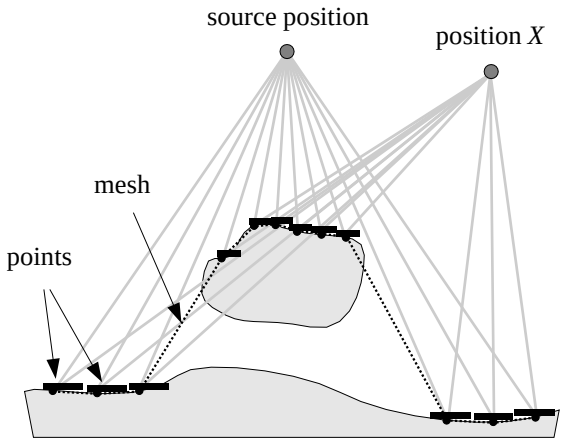

Figure 2. Two-dimensional example of objects rendered on images $P_{d,s}$. Thin dotted line denotes triangle mesh (drawn with colour c_b), thick short horizontal lines denote points (colour c_p).

acceleration. The algorithm renders 24 images $P_{d,s}$, where $d = 1,\ldots,6$ is one of 6 directions in which the image is rendered, $s = 1,\ldots,4$ is a number of a tetrahedron vertex for source images. Each source image is rendered twice, first as a triangle mesh (with the background colour c_b) and second as points drawn as 2x2 pixel squares (with the colour c_p). These points from the second pass are slightly moved towards the observer. Then the points on surfaces perpendicular to the observer viewing direction would not be covered by the triangle mesh. The figure 2 shows a two-dimensional example.

After computing images $P_{d,s}$, the algorithm merges them. For each of the 6 directions it computes image $P_{d,\mathrm{all}}$. This merging is performed independently for each pixel, the rule is the following:

- If the pixel has colour c_p on at least one of the 4 images, then on the destination image the colour is c_p. Otherwise the colour is c_b.

The rule indicates that if the surface is visible (the colour is c_p) on at least image $P_{d,s}$ then there is no occluding object between the surface and the observer.

The goal of the optimisation algorithm is to find the position inside each tetrahedron with maximum objective function value. The domain is three-dimensional. The computation of the exact or approximate value of a gradient is hard, thus for such problem the best approach is to use a randomised method, such as some evolutionary algorithm. We chose the genetic algorithm. The other possibility is, for example, simulated annealing. Genetic algorithm is chosen because of the crossover operator. This operator speed-ups the maxima search in the situation, when 2 positions to be crossed are on the opposite sides of the same maximum.

6. IMPLEMENTATION

The algorithm presented in this article was implemented in the C++ (GCC) on Linux. For visualisation we used the SDL library along with the OpenGL library for 3D hardware acceleration.

Source images were rendered with a ray-tracer called Rayshade [Kolb87a]. Each image was saved to the separate file. We used adaptive anti-aliasing by subsampling where the colour of adjacent pixels was greater than some threshold, but the pixels depth was not to big (in the opposite case, the pixels came from different surfaces).

Pixels are drawn as points of a suitable size (such that there would be no holes between adjacent points). In the drawing procedure the coordinates are transmitted to the OpenGL library as Vertex Arrays and the colour is put by an automatic texture projection.

To improve the interactivity of our program, we implemented the loading of new images in the background thread. Unfortunately, this may cause that the surfaces not loaded so far are not displayed.

Figure 3 shows sample frame from visualisation. Squares depict mesh vertices, white lines are the edges of tetrahedra. The test scene is the model of the church created by Nathan O'Brian [O'Brie93a].

The test configuration was a computer Duron 800MHz, 512 MB RAM, NVidia GeForce 2 MX graphics card with 32 MB RAM. The resolution of the reference images was 400x400 pixels. The visualisation algorithm rendered images of the resolution 640x480 with the field of view 20°. The achieved frame-rate was 5-6 frames per second (about 1M points were rendered per second[2]).

In the automatic algorithm of selecting reference positions, we used 3D hardware acceleration for rendering scene views. We used an OpenGL extension called P-buffer for an off-screen rendering. To speed-up the algorithm, the resolution of source and destination images was diminished 4 times in both axes directions

[2]Remark: points were drawn as small squares, what slowed down the visualisation comparing to drawing points as single pixels.

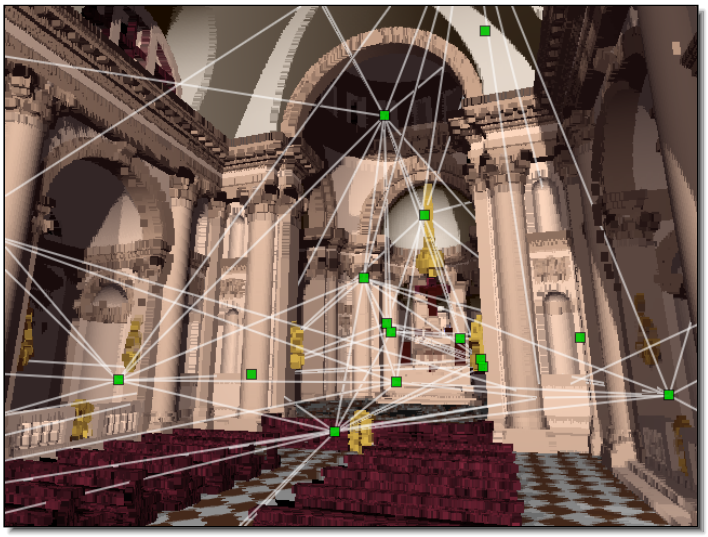

Figure 3. Sample scene with reference positions and tetrahedra edges shown.

(100x100). The time of computing of the objective function was approximately 1s (it involved rendering of 24 images $P_{d,s}$, and 6 merging of images $P_{d,\text{all}}$).

For each tetrahedra the genetic algorithm executed 50-150 iterations (depending on the test). There were 4 individuals (the position is an individual in this case) in the population, in each iteration 2 individuals were produced and 2 worst individuals were removed. Individuals to cross and reproduce were randomly selected, the probabilities of individuals with bigger objective function were bigger.

7. CONCLUSION AND FURTHER WORK

In this article we presented the visualisation algorithm based on warping and space partitioning by three-dimensional Delaunay triangulation. The main advantage of the presented algorithm is the independence from the object complexity. The quality of visualised surfaces is similar and does not depend on the observer position.

In some cases, we may not be able to remove holes, even if we increase the density of the mesh by inserting new positions (the size of holes decreases, but the holes do not disappear). Such situation may occur, for example, when an observer sees distant surface through a small hole in the closer surface. A very small observer movement may cause completely new part of the distant surface to be visible through that hole.

A solution to this problem may be the following. Instead of visualising individual images bound to the tetrahedra vertices, one should visualise unorganised point sets bound to separate tetrahedra. However, still it is an open problem, how to optimise a subset of points that has to be exchanged after moving from one tetrahedron to another.

REFERENCES

[Adels91a] Adelson, E.H. and Bergen,J.R.: Computational Models of Visual Processing, *MIT Press, Cambridge, MA, 1991*, ch. 1 (The Plenoptic Function and the Elements of Early Vision).

[Olive00a] Oliveira, M.M. de, Bishop, G. and McAllister, D.: Relief texture mapping, *SIGGRAPH 2000 Conference Proceedings*, 2000.

[Fleis99a] Fleishman, S., Cohen-Or, D., and Lischinski, D.: Automatic camera placement for image-based modelling, *Computer Graphics Forum, 19 (2000)*, pp. 101–110, 2000

[Gortl96a] Gortler, S.J., Grzeszczuk, R., Szeliski, R. and Cohen, M.F.: The Lumigraph, *SIGGRAPH 96 Conference Proceedings*, pp. 43–54, 1996

[Gross98a] Grossman, J. P. and Dally, W.: Point sample rendering, *in Rendering techniques '98*, 1998, pp. 181–192.

[Klein00] Klein, K. and Sequeira, V.: The view-cube: An efficient method of view planning for 3d modelling from range data, *in 5th IEEE Workshop on Applications of Computer Vision (WACV'2000)*, Palm Springs (CA), USA, December 2000.

[Kolb87a] Kolb, C. and Bogart, R.: *Rayshade ray tracer*, 1987-1998, http://www-graphics.stanford.edu/ cek/rayshade/.

[Levoy85a] Levoy, M. and Whitted, T.: The use of points as a display primitive, *Tech. Rep. TR 85-022, University of North Carolina at Chapel Hill*, 1985.

[Levoy96a] Levoy, M. and Hanrahan, P.: Light field rendering, *SIGGRAPH 96 Conference Proceedings*, pp. 31–42, 1996

[Mark99a] Mark, W.R.: Post-Rendering 3D Warping: Visibility, Reconstruction and Performance for Depth-Image Warping, *Ph.D. thesis, Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, North Carolina*, 1999

[Massi98a] Massios, N. A. and Fisher, R. B.: A best next view selection algorithm incorporating a quality criterion, *in 9th British Machine Vision Conference, Southampton, England*, 1998, pp. 780–789.

[O'Brie93a] O'Brien, N.: Rayshade model of the interior of Palladio's Il Redentore in Venice, 1993.

[Pfist00a] Pfister, H., Zwicker, M., Baar, J. van and Gross, M.: Surfels: Surface elements as rendering primitives, *SIGGRAPH 2000 Conference Proceedings*, 2000, pp. 335–342.

[Rusin00a] Rusinkiewicz, S. and Levoy, M.: QSplat: A multiresolution point rendering system for large meshes, *SIGGRAPH 2000 Conference Proceedings*, 2000, pp. 343–352.

[Seque96a] Sequeira, V., Goncalves, J. and Ribeiro, M. I.: Active view selection for efficient 3d scene reconstruction, *Proceedings of the 13th International Conference on Pattern Recognition (ICPR)*, Vienna, Austria, August 1996, pp. 815–819.

[Shade98a] Shade, J. W., Gortler, S. J., He, L. wei and Szeliski, R. : Layered Depth Images, *SIGGRAPH 1998 Conference Proceedings*, 1998, pp. 231–242.

[Stürz99a] Stürzlinger, W.: Imaging all visible surfaces, *Proceedings of Graphics Interface 99*, 1999.

[Werne00a] Werner, T., Pajdla, T., Hlaváč, V., Leonardis, A. and Matoušek, M.: Selection of reference images for image-based representation, *Tech. Rep.*, Center for Machine Perception, Czech Technical University, Prague, Czech Republic, December 2000.

[Wong98a] Wong, L. M., Dumont, C. and Abidi, M. A.: Determining optimal sensor poses in 3-d object inspection, *Conference on Quality Control By Artificial Vision*, November 1998, pp. 371–377.