EUROGRAPHICS 2002 / I. Navazo Alvaro and Ph. Slusallek
(Guest Editors)

Visibility Preprocessing Using Spherical Sampling of Polygonal Patches

Oscar E. Meruvia Pastor

Department of Simulation and Graphics, Otto-von-Guericke University of Magdeburg, Germany

Abstract

A technique is presented that permits fast view-reconstruction of individual objects. This method improves a previous approach to solve the problem of approximated view reconstruction by combining clustering of polygons with visibility bitfields to determine visibility for novel viewpoints. The technique consists of three steps: patch creation, spherical sampling, and rendering. In the first stage, the input 3D model is tiled in polygonal patches. In the sampling stage images of the model are taken from several points on the surface of a viewing sphere. Patch-ID bitfields, which are structures that contain visibility information, are computed for each picture. In the rendering stage, a subset of the viewpoints computed for sampling is selected depending on the viewers position on the viewing sphere and the bitfields of the selected viewpoints are used to rebuild the visible parts of the model from the new viewpoint.

The overall system presented here makes a very efficient use of memory resources, and involves practically no overhead during rendering while significantly improving frame rate during interaction with large models. Although the technique is not conservative, our results show that the reconstructed views are practically identical to the original views of the model.

Key words: view reconstruction, visibility preprocessing, patch generation.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Traditional graphics systems compute views of a model by sending all polygons in the model to a graphics pipeline where one of the last stages is to determine the visible parts of the model using the z-buffer algorithm. The z-buffer algorithm is easily implemented in hardware and works for any model from any viewpoint. However, a problem with this graphics pipeline becomes apparent when the number of visible polygons at a given viewpoint is significantly less than the number of total polygons included in the model. In this case the system spends most of the time processing polygons which are not visible in the final rendition and introducing delay in the response time. In this work we present a technique to preprocess and then reconstruct external views of arbitrary polygonal models in real-time. Our technique relates to the area of computer graphics which deals with visibility computation and preprocessing, mainly occlusion culling but also image-based rendering and backface culling. Occlusion culling techniques attempt to determine the set of visible polygons of a model or scene from a particular viewpoint before sending all the polygons to the graphics pipeline. While occlusion culling techniques focus on computing visibility for a user who moves in a virtual environment, our technique computes visibility for individual objects in the scene. To reconstruct the visible regions of individual objects in real-time we apply two preprocessing steps to the input model: patch tiling and spherical sampling. During visualization, we determine which of the samples taken during preprocessing are relevant for the viewer depending on the viewer's position with respect to the input model. In image-based spherical sampling[12], images of a model are taken from different viewpoints on the surface of a sampling sphere which encloses the input model. An ID-bitfield encodes the list of visible polygons for each sample taken. The

ID-bitfield is a long bitfield of n bits, where n is the total number of polygons in the model. Each position in the bitfield is associated with a polygon in the model and indicates whether a given polygon is visible or not from the viewpoint where a picture was taken. The ID-bitfield and camera values for every picture taken are stored in a file which is later used by a renderer. During visualization, the task of the viewer is to determine which of the images previously taken are relevant for an arbitrary new viewpoint. A disadvantage of the ID-Bitfield approach is that the time required to construct the list of visible polygons is linear with respect to the number of polygons in the input model. Another drawback of the original technique is that the process of sampling was not automatic, which made it unpractical to take more than 60 samples.

In this work, we present several improvements to the technique above described. Specifically, the contributions of this paper are:

1. Reduction of complexity by using polygonal patches.
   When using patch-tiling, complete regions (patches) of a model are discarded at once when they are not visible. The time required to construct the list of visible patches is linear with respect to the number of patches in the model, thus, the reconstruction can be done in real-time if we split the model in a few hundred patches (see Figure 1). Additionally, patch-tiling facilitates sampling, (since it is easier to register a group of polygons than a single polygon from an ID-buffer), allows efficient use of memory resources, and eliminates long setup times .
2. Automatic sampling.
   A sampling program was developed which takes pictures of a model from viewpoints on the surface of a viewing sphere at any desired sampling level using a subdivision approach.
3. Optimized view selection.
   We now use a hierarchical sphere subdivision scheme, which permits fast selection of the relevant samples out of the faces of the sampling sphere when rendering.

The rest of the paper is organized as follows. Section 2 presents related work in visibility preprocessing techniques and object-centered rendering. In Section 3, we describe our technique step by step: patch generation, spherical sampling and view reconstruction. Section 4 describes the experimental results of the technique. Conclusions and future work are discussed in Section 5.

2. Related Work

Several occlusion culling techniques perform visibility preprocessing when solving the problem of real-time visibility computation. Since a user normally moves freely within a virtual environment, these techniques have opted for subdividing the virtual environment in viewing cells and then computing visibility from within each cell[5, 21]. In Zhang et al.[21], for example, the model space is subdivided by a uniform grid of points, then a cube which surrounds each of the grid points is created and a list of visible objects from inside the cube is computed. While these techniques perform visibility preprocessing from within a given cell, our technique focuses on preprocessing visibility for individual objects in a scene, and could be used together with techniques that determine visibility at the object level. In this case, an algorithm such as the one by Gotsman et.al.[5], could find which objects are visible for a certain viewpoint, and our technique could be used to render the visible parts of the visible objects with respect to the user's position. Occlusion culling techniques can be divided in two categories, conservative and non-conservative. Conservative occlusion culling means that all polygons visible from a viewpoint are guaranteed to be rendered. Non-conservative occlusion culling means that most of the visible polygons are rendered, but some may not. The technique here presented is non-conservative, since we cannot guarantee that all visible polygons are present in the reconstructed view. In practice, however , it is hard to notice any difference between the original and the reconstructed views when sampling is thoroughgoing (1026 samples or more), as results show in Section 4.

A related family of techniques focuses on backface culling[7, 20]. Zhang and Hoff[20] present a technique where a normal mask (a bitfield of 2 bytes length) is stored for each polygon. Each bit in the normal mask is associated with a cluster of normal vectors in a normal-space partitioning scheme. With this approach significant culling rates and speed-ups are obtained. However, the visibility test still needs to be performed for all polygons in the input model, and the memory requirements are high. Kumar et al.[7] present a technique where polygons are clustered by normals in a preprocessing stage. However, this grouping scheme is not compatible with the way graphics pipelines process textured models, as discussed by Zhang[20]. To some extent, our technique combines both of these techniques because we group polygons (in patches), and use bitfields to encode preprocessed visibility information. Moreover, the technique presented here can also be applied to textured models. Additionally, OpenGL[13] permits hardware supported backface culling, but the time required for rendering is still linear with respect to the number of input polygons, because the culling test needs to be performed in real-time for each polygon. Our technique first approximates the set of visible polygons and then sends this set to the graphics pipeline.

The last major group of visibility preprocessing techniques is the group of image-based rendering. These techniques preprocess visibility by taking pictures of those parts of the scene which are farthest from the user. After this, images can be placed on 2D billboards in the place of 3D geometry at the back of the scene (similar to what is done in occlusion culling). In object-centered image-based rendering, images of a specific model are taken from several viewpoints around it[4, 14, 2], such that a small subset is selected for

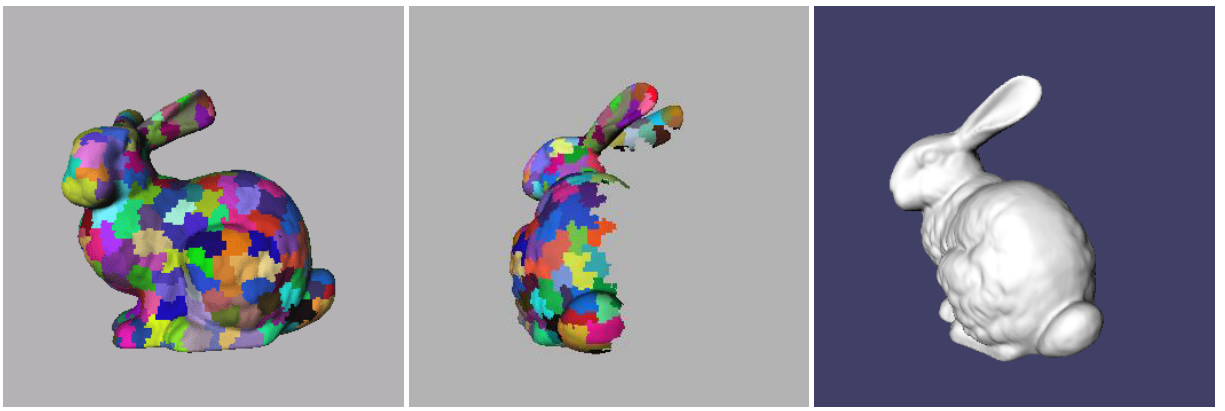

Figure 1: *This sequence of images shows an image-based sample as obtained from a patch-ID bitfield. The first image shows the sample from the point where the picture was taken, the following two images show the sample as it is being rotated. The last image shows the sample in the original color. The surface of the bunny was tiled in 300 patches.*

rendering according to the new viewpoint. Since images are valid only within a small viewing range, one needs either a large amount of images or images enhanced with depth information[14]. Systems that take large amounts of images require an efficient image selection mechanism and large amounts of memory, or schemes for image compression[17]. Finally, to appropriately react to changes in external illumination, images require surface normals.[2] In the system here presented, these problems do not occur, because we reconstruct the 3D geometry and pass it to the standard graphics engine. Furthermore, it is possible to sample a model without textures, and then use the model with textures when rendering, thus providing high quality renditions of the visible parts of a model. Finally, the approach requires a small amount of extra memory (the patch-bitfields encoding) to produce the appropriate view reconstruction.

3. Sampling and Rendering Polygonal Patches

3.1. Patch Creation

A polygonal patch is a grouping of polygons which encloses a region of the model. Organizing the model into polygonal patches for sampling and rendering is central to our technique. Two main benefits derive from this clustering of polygons. First, image-based sampling of polygon groups is easier than sampling individual polygons. The second advantage is that by grouping the model's polygons in patches, the time required to reconstruct the view is bound by the number of patches and not of by the number of polygons in the model.

The algorithm of patch creation works as follows. In the beginning every single polygon of the model is seen as an individual patch and the information about the neighbours of each patch is taken from the connectivity graph derived from the original model. The patches are then fused with their neighbors iteratively until the desired patch count is reached. In each iteration the smallest patch (by area) is fused with one of its neighbours such that the increase in the extension of the patch is minimized (the extension of a patch is measured by the length of the diagonal of the patch bounding box). By doing this, we ensure that patches grow in a compact way (see Figure 1).

3.2. Spherical Sampling

We generate bitfields that indicate which patches are visible from viewpoints on the surface of a sampling sphere. We call this process spherical sampling. To sample the model we set a camera which moves around an object located at the center of our sampling sphere, in a similar way as done when manipulating objects with the Arcball[18], or the Open Inventor Viewer.

3.2.1. Generating Sampling Positions

We uniformly sample the model from positions on the surface of a sphere enclosing the model. The samples are taken looking towards the center of the object at a distance such that the entire model lies within the viewing area.

To calculate the position of each sample, we adapted a face subdivision algorithm commonly used to generate facet approximations to a sphere[1]. We begin with an octahedron as the starting shape. In each subdivision step each face is subdivided into 4 faces, and the vertices of the faces are projected to the surface of the sphere. This algorithm generates new sampling points and new faces in a uniform distribution on the surface of the sphere at an exponential rate (base 4). This sampling is more dense in each subdivision level, filling the "holes" of the previous level.

Each vertex of the polyhedron corresponds to a camera

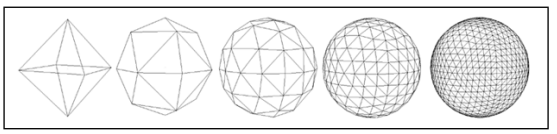

Figure 2: *Levels of subdivision of the sampling sphere.*

position from where pictures of a model located at the center of the sphere are taken. Figure 2 shows the first five approximations (which contain 6, 18, 66, 258 and 1026 different viewpoints respectively) obtained using this algorithm.

3.2.2. Generating Patch-ID Bitfields

In this stage the bitfields that correspond to each picture taken are generated. The ID-bitfield is computed by scanning the ID-buffer[19, 16] of the pictures obtained from each sampling point. In our case we initially assigned a unique color to each patch and a position in an indexed array, which is the patch-ID bitfield. In addition, a map is created from the set of available colors to the set of available positions in the array. Then we proceed to sample the model by taking pictures of it from all the viewpoints generated by the sphere subdivision algorithm with the model rendered in colored patches.

The (color) indices of all visible patches from a given viewpoint are obtained by inspection of the color buffer of the rendered model. Each index found is used to set the bit of the corresponding patch, indicating that the patch is visible from the viewpoint where the picture was taken. Once all pictures have been taken, the bitfields and corresponding camera positions are saved in a configuration file. The size of this configuration file is relatively small since we need only to store one bit per patch per picture and one camera position for each picture taken.

Sampling a polygon patch is easier than sampling individual polygons, because patches cover larger areas of the color buffer than individual polygons do. The risk of failing to sample polygons of sub-pixel size is a matter of image resolution, and is practically eliminated when doing patch sampling, because all polygons in the input model are guaranteed to belong to a patch and the patches themselves are compact, i.e. they enclose all polygons inside a given area. The risk of missing a whole patch depends more on visibility issues (where the density of spherical sampling plays a more important role) than on image resolution.

3.3. View Reconstruction and Rendering

To interactively render a model, we first determine which of the generated sampling points are relevant for the new viewpoint, and then we construct the parts of the model that correspond to the sampling points selected.

3.3.1. Selecting a Set of Samples

Given the high density sampling that is performed, selecting the three closest samples to an arbitrary viewpoint effectively allows us to do a good reconstruction of the model.

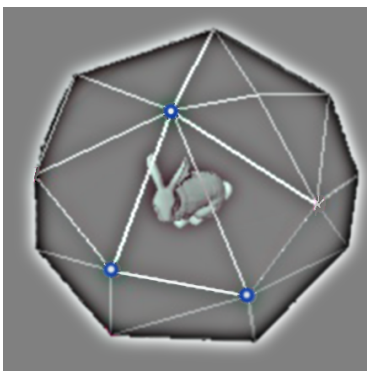

Figure 3: *The relevant samples for view reconstruction are the (highlighted) vertices of the face of the viewing sphere intersected by the viewing ray.*

Since these three samples correspond to the vertices of one face of the sampling sphere, the problem of finding the three closest samples translates to the problem of finding the face which is intersected by a (viewing) ray sent from the center of projection to the viewer's position, as shown in Figure 3. However, testing a linear array of hundreds or thousands of faces to determine which one intersects this ray is time consuming.

We developed an algorithm which takes advantage of the hierarchy implicitly created by the face subdivision process used to create the sampling sphere (Section 3.2.1). The first eight faces of the octahedron represent the top level of the hierarchy. Each time a face is subdivided, it is divided in four new faces, which are considered descendants of the original face at the following level of the hierarchy.

The algorithm first tests which of the initial faces of the base octahedron intersects with the viewing ray. If a viewing ray intersects one face of the sphere at any subdivision level, there is one face among its descendants which also is intersected by the viewing ray, so we look for the intersecting face only among the descendants of the parent face. We recursively traverse the hierarchy until we find a face which has no descendants, i.e. until we reach the last level of subdivision on the sampling sphere. This will be the face we are looking for. The number of tests to find the intersecting face is linear with respect to the subdivision level (at most 4 tests per subdivision level). Even when the number of faces and viewpoints grows at an exponential rate, the algorithm finds the relevant face in linear time.

3.4. Joining the Selected Samples

To join the information of the samples corresponding to the vertices of the selected face, we perform a join operation (inclusive OR) between the bitfields. After that, we include in the scene graph those patches which have their bit set, so that all visible patches from each selected viewpoint are sent to the graphics pipeline. An advantage of this approach with respect to the ID-Bitfields[12] is that we do not construct a view on a per-polygon basis, using a bitfield with as many bits as the number of polygons in the model. Instead, we construct a view on a per-patch basis with a limited number of patches. The time required to construct the scene is linear to the number of patches in the model, as shown in the experimental analysis.

4. Experimental Results

We tested culling performance (in terms of culling rate) and interaction performance (in terms of frame rate) under several conditions of sampling and patch tiling for different models. All results were obtained on a SGI Onyx2 Infinite Reality, with 2 195MHz MIPS R10000 CPUs, 900Mb main memory, and video output set at 72.0 Hz. Sampling was done using the Open Inventor offline renderer at a resolution of 510 by 480 pixels.

Conditions of sampling were derived from the possible number of samples generated by the sphere generation algorithm using 3, 4, 5 and 6 subdivisions which produce 66, 258, 1026 and 4098 sampling points. Sampling is the most time consuming part of our algorithm. Sampling times vary according to the size of the input model: small models (wing, 6,100 polygons) can be sampled at a rate of 3 samples per second, large models (brain model, 288,344 polygons) require up to 3.5 seconds per sample (due to the offline rendering), which accounts for up to 4 hours in the highest sampling condition. The number of patches in which the models were tiled was selected dynamically as the experiments occurred, since we tried to find the most interesting regions in terms of culling and frame rate optimization.

4.1. Analysis of Culling Performance

We defined a measure for culling performance called culling rate as the average number of culled polygons on all possible views from the model with respect to the total number of polygons in the original model.

We found that culling rate depends directly on both the number of samples and on the amount of patches in which the model is tiled. Figure 4 illustrates that culling rate increases with the number of samples. However, this increase tends to be smaller as sampling increases. This is because on each subdivision level the sampling points lie closer to one another. We also observe from Figure 4 that varying the number of patches directly affects the culling rate for the same model (Horse). What is more, we can see that the effects of sampling add up with the effects of patch tiling in culling rate.

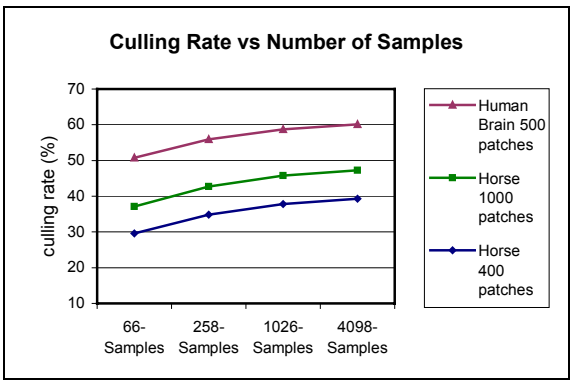

Figure 4: *Average of culling rate across number of samples for different models and different number of patches.*

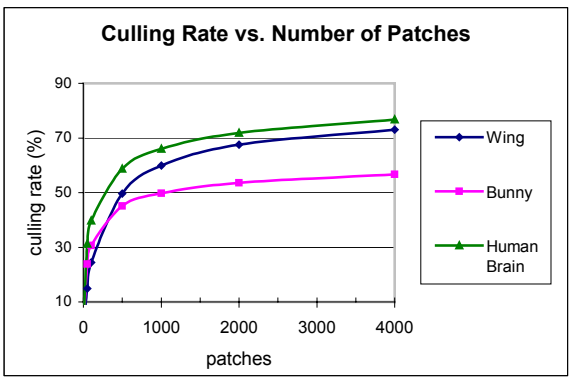

Figure 5: *Variation of culling rate across number of patches for several models. 1026 samples were used in all conditions.*

In Figure 5 we present a more detailed analysis of the effects on the culling rate by changes in the number of patches in which the model is tiled. We can see that even when culling rates vary according to the model, the general behavior of all curves is the same, that is, a greater number of polygonal patches increases the culling rate, but the improvements are less significant as the number of patches increase.

We conclude that it is better to perform higher sampling as a way to provide better culling quality. On the other hand, even as we observed that a higher number of patches also improves culling, increasing the number of patches has some drawbacks in terms of frame rate as is shown in the next section.

4.2. Analysis of Performance in Terms of Frame Rate

We measured the average frame rate achieved by our system by rendering a model which was constantly rotating around its vertical axis of rotation for a period of five minutes under each condition. Models were rendered using double buffering and the maximum frame rate was bound by the refreshing rate of the video output (72Hz). We tested frame rate versus sampling density and number of patches separately.

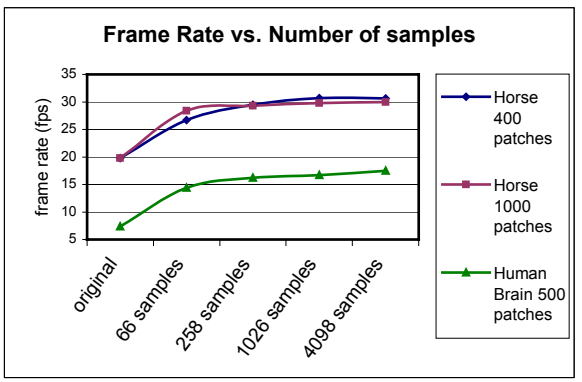

Figure 6: *Frame rate across number of samples.*

Figure 6 shows how frame rate increases as the sampling density increases. This occurs because culling improves as sampling increases as described in the earlier section. Also, sample selection takes advantage of the spherical subdivision algorithm described in Section 3.3, so even when the number of samples increases exponentially, we can find the relevant samples in linear time with respect to the subdivision level. Figure 6 also illustrates how frame rate improves as more samples are taken, but starts reaching a plateau at 258 samples (which speaks for keeping the number of samples low). As the model is sampled above this level, frame rate is bound by the amount of visible polygons.

Figure 7 illustrates the relationship between frame rate and the number of patches for several models. All models were sampled with 1026 samples. In all cases we observe a plateau of performance for frame rate, which is reached even with a relatively small number of patches (we should consider that the Wing, Bunny and Brain models have approximately 6,100, 69,000 and 288,000 polygons each). After that, we can observe that increasing the tiling above 1,000 patches (500 for the Wing model) starts affecting negatively performance in terms of frame rate. Because the three performance curves are very different for each model, we analyze each case separately.

The Wing (or X-wing) model is relatively small, the whole model can be rendered in real-time by our system, so any decrease in performance can be blamed on the experimental conditions. We can observe that frame rate is kept at 72 frames per second (fps) practically until the model is tiled

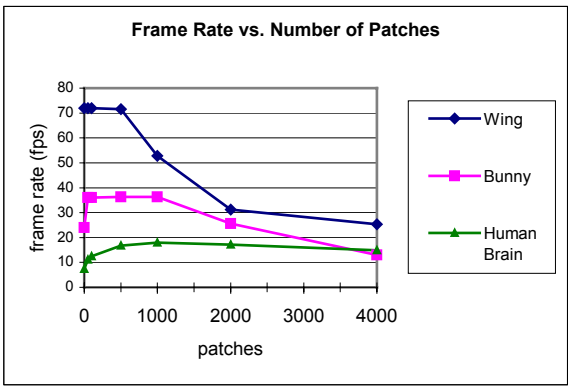

Figure 7: *This figure illustrates how performance (in terms of frame rate) is affected by the number of patches in the sampled model. 1026 samples were used in all conditions.*

in 500 patches. After that, frame rate starts to decrease and falls significantly at 2000 patches. The data line for Wing shows the maximum frame rate attainable for any model in function of the number of patches alone, when using 1026 samples, and speaks for keeping the number of patches low.

The Bunny model exhibits a surprising behavior, because the maximum performance is reached as soon as the model is tiled in as few as 50 patches. We can hypothesize that for certain model shapes (like a sphere), tiling the model in a small number of patches is enough to cull away large portions of the model, provided sampling is thoroughgoing. In addition, a small list of patches can be managed easily.

An analysis of the Brain model lets us observe the expected trade-off between the number of patches and frame rate. If the number of patches is too small, culling and frame rates are not optimal. As the number of patches increases, the culling rate improves, but the time required to manage the patches list and create a new scene increases as well. We can observe that the best frame rate is obtained when the model is tiled into 1000 patches, and that increasing from this point on only worsens frame rate, although not in a dramatic way, because frame time is spent more on rendering the large amount of visible polygons, rather than on constructing the view. These results speak for keeping the number of patches low, and sampling around between 258 and 1026 samples to obtain optimal results.

Finally, Table 1 shows the improvements in performance for the best configurations of number of patches for the five test models. 1026 samples were used in all cases, since the difference in frame rate to the next level of subdivision is not significant. Results show that we can improve frame rate from 0.4 up to 1.4 times the original values. However, as the Dragon, Brain and the Skeleton Hand models show, the benefits of visibility preprocessing are bound by the number of visible polygons at any given time. For large models such

Model	Polygons	Patches	Culling rate	Original f.r. (fps)	Optimal f.r. (fps)	Frame rate Improvement
Bunny	69,451	50 - 1000	15 - 60 %	24.9	36.0	50%
Horse	96,966	400	38%	19.8	30.7	55%
Dragon	871,414	600	44%	2.2	3.6	67%
Human Brain	288,334	1000	66%	7.4	18.0	144%
Hand	654,666	1000	38%	2.8	4.0	43%

Table 1: *Improvements in terms of frame rate for the optimal number of patches. The Wing model is originally rendered at 72 fps, the maximum frame rate. The Bunny model reached optimal frame rate under a range of number of patches (see Figure 7).*

as these, Level-of-detail techniques[6, 11, 15] are required as a preprocessing stage to attain interactive frame rates (30 fps and higher).

In general the results show that we can efficiently improve performance by choosing an appropriate number of patches and by performing a dense sampling (1026 sampling points).

4.3. Analysis of Visual Quality

Since it was hard to distinguish the original models from the reconstructions, we used a test program to compare images of the reconstructed views with images of the original model from viewpoints selected at random positions on the viewing sphere. The program reported that all test images were identical to the images of the complete models. This is the case for all images shown in Figure 8. We do not contend that there are no positions where a difference could be noticed, but our observations are that it is hard to find these positions with a sampling density of 1026 viewpoints. On the other hand, the quality of the reconstruction depends to a great extent on the model. We can imagine there are models which represent a greater challenge to this technique, like a tree, but then it does not seem practical to attempt visibility preprocessing of such a model.

A particular situation arises when the user tries to see a model from a distance closer than that where the samples are taken. In this case uncovered areas of the model may appear. In these situations we recommend to switch to the full model, since it is our experience that the user will not only try to see the model at a closer distance, but he will also try to inspect specific regions of the model changing even the focus of attention and the center of rotation to a specific region of the model. This problem does not occur for viewpoints located farther from the sampling sphere, because the projective distortion is more significant when looking closer at an object, than when looking farther from a given viewpoint (see videos[12]).

4.4. Memory requirements

The largest amount of memory is required for the patch tiling algorithm. Since tiles can be removed from memory once they have been used in a patch fusion iteration, we only require the original model plus the tile connectivity information at each iteration. The sampler and the renderer require the memory necessary to store the (tiled) model, the patch-bitfield (one bit per-patch per-sample), plus one image buffer for the sampler. Configuration files containing the patch bitfield information to be used by the renderer vary in size, according on the number of samples encoded and patches in the model. For instance, the human brain model configuration file for 50 patches tiling, containing 1026 samples occupies 183Kb of disk storage; while the configuration file for the same model divided in 2000 patches, 1026 samples uses 709Kb. The human brain model itself occupies 12Mb of disk space, either in the original form or tiled in patches.

5. Conclusions and Future Work

We have described a technique which provides high quality non-conservative view-reconstruction in real time for external views of a model. A preprocessing stage divides the model in polygonal patches. In the sampling stage, image-based visibility preprocessing is done by taking pictures of a model from a viewing sphere. Finally, the rendering stage consists in sample selection for a specific viewpoint. This selection is performed with an algorithm that takes advantage of the sphere subdivision scheme used in the sampling stage.

The technique is conceptually simple and can be applied to any polygonal model, improving interaction in terms of frame rate from 50% to 144% while providing high quality renditions of a model.

Areas of extension for this work are:

- User-centered interaction. Use of this technique in user-centered interaction, where visibility is preprocessed for every object in a scene with respect to the currrent camera position. Additionally, a space partitioning scheme would help determine visibility inside a cell and spherical sampling would help determine visibility of the individual objects according to the user position with respect to the models.
- Spatial partition. Introduction of a distance parameter to improve culling and discovering of hidden faces when the user zooms into a model when using an orthographic camera.
- Inventor Selection Node. Creation of a selection node for use of the technique with individual objects contained in larger scenes.
- Enhancement of the sample selection process. It is possible to develop an algorithm to determine the minimal set of positions from where a picture of the model should be taken to cover all visible regions of a model. This would be an object-centered variant of the “art gallery problem”

that could be effectively used to reduce the set of samples taken[3].

Acknowledgements

The author would like to thank Prof. Thomas Strothotte for his valuable comments and suggestions, and Lourdes Peña Castillo for reviewing drafts. Figure 2 included with permission of P. Bourke[1].The Human Brain model was provided by Thomas Witzel, the X-wing (Wing) model was obtained from the VRML Object Supermarket[10]. The Horse model is available at the Large Geometric Models Archive[8]. The Skeleton Hand, Bunny and Dragon models are from the Stanford 3D Scanning Repository[9]. This work was supported by a scholarship of the state of Sachsen-Anhalt, Germany.

References

1. P. Bourke. *Sphere Generation*. Swinburne University of Technology, 2002. http://www.swin.edu.au/astronomy/pbourke/modelling/sphere. 3, 8
2. Q. Dinh, R. A. Metoyer, and G. Turk. Real-time lighting changes for image based rendering. In *Proc. of the IASTED International Conference, Computer Graphics and Imaging*, pages 58–63, 1998. http://www.cc.gatech.edu/~metoyer/. 2, 3
3. S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic camera placement for image-based modeling. In *Pacific Graphics'99 Conference Proc.*, 1999. http://www.math.tau.ac.il/~shacharf/publications.html. 8
4. S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumighraph. In *SIGGRAPH 96 Conference Proceedings*. ACM, 1996. http://research.microsoft.com/MSRSIGGRAPH/96/Lumigraph.htm. 2
5. C. Gotsman, O. Sudarsky, and J. Fayman. Optimized occlusion culling using five-dimensional subdivision. In *Computers and Graphics, 23(5)*, pages 645–654. Eurographics, 1999. http://www.cs.technion.ac.il/~sudar/cag.ps.gz. 2
6. P. Heckbert and M. Garland. *Survey of Polygonal Surface Simplification Algorithms.* Carnegie Mellon University, 1997. http://www-2.cs.cmu.edu/~ph. 7
7. S. Kumar, D. Manocha, B. Garret, and M. Lin. Hierarchical backface culling. In *7th. Eurographics Workshop on Rendering.*, pages 231 – 240. Eurographics, 1996. ftp://ftp.cs.unc.edu/pub/users/manocha/PAPERS/VISIBILITY/backface.pdf. 2
8. *Large Geometric Models Archive.* Georgia Institute of Technology, 2002. http://www.cc.gatech.edu/projects/large_models/. 8
9. *Stanford 3D Scanning Repository.* Stanford University, 2002. http://graphics.stanford.edu/data/3Dscanrep/. 8
10. *The VRML Object Supermarket.* The University of Edinburgh, 2002. http://www.dcs.ed.ac.uk/home/objects/vrml.html. 8
11. D. Luebke. Developer's survey of polygonal simplification algorithms. In *IEEE Computer Graphics & Applications/ (May 2001).*, May 2001. http://www.cs.virginia.edu/~luebke/publications.html. 7
12. O. Meruvia and T. Strothotte. Approximated view reconstruction using precomputed id-bitfields. In *Eurographics '2001 Short Presentations.*, 2001. http://isgwww.cs.uni-magdeburg.de/~oscar/. 1, 5, 7
13. J. Neider and T. Davis. *OpenGL Programming Guide*, chapter 2. Addison-Wesley Publishing Company, 1997. http://www.opengl.org/developers/documentation/specs.html. 2
14. M. M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 359–368. ACM Press/Addison-Wesley Publishing Co., 2000. http://doi.acm.org/10.1145/344779.344947. 2, 3
15. E. Puppo and R. Scopigno. Simplification, lod and multiresolution - principles and applications. In *Technical report, Eurographics '97 Tutorial Notes*, 1997. http://www.disi.unige.it/person/PuppoE/. 7
16. T. Saito and T. Takahashi. Comprehensible rendering of 3-d shapes. In *SIGGRAPH 90 Conference Proceedings*, pages 197 – 206. ACM, 1990. 4
17. J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings*, pages 75–82. ACM Press, 1996. http://doi.acm.org/10.1145/237170.237209. 3
18. K. Shoemake. *Arcball Rotation Control, Graphics Gems IV*, pages 175–192. Academic Press, 1994. 3
19. T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann Publishers, April 2002. 4
20. H. Zhang and K. Hoff III. Fast backface culling using normal masks. In *Proc. of the 1997 Symposium on Interactive 3D Graphics*, pages 103–106. ACM, August 1997. 2
21. H. Zhang, D. Manocha, T. Hudson, and K. Hoff III. Visibility culling using hierarchical occlusion maps. In *SIGGRAPH 98 Conference Proceedings*, pages 77 – 88. ACM, 1998. 2

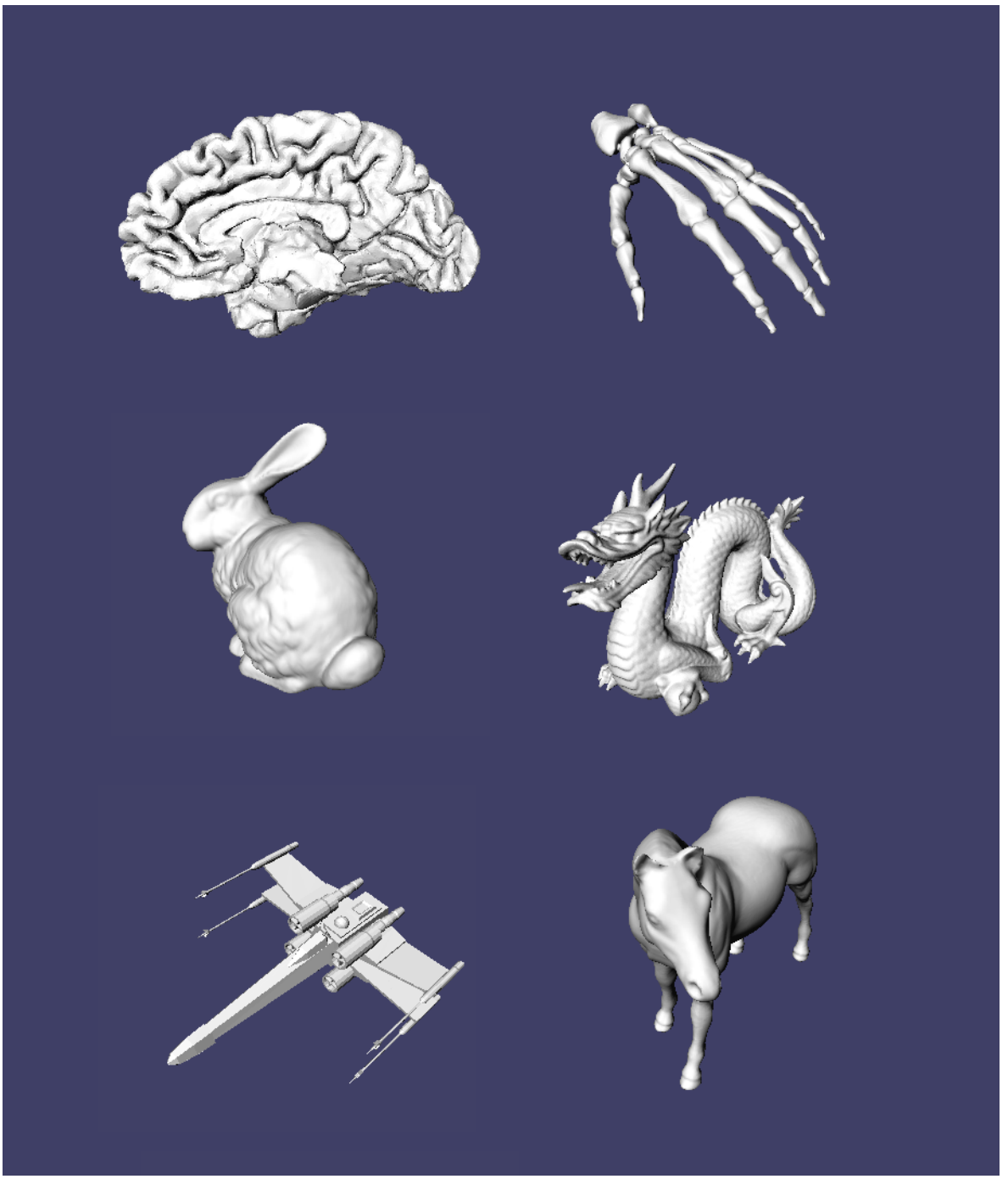

Figure 8: *Some examples of the visualizations obtained by our technique. All images were identical to the images obtained when the original models were rendered. In all cases 1026 samples were taken.*