Imaging all Visible Surfaces

Or

How many Reference Images are needed for Image-Based Modeling?

Wolfgang Stürzlinger
Dept. of Computer Science, University of North Carolina at Chapel Hill
stuerzl@cs.unc.edu

Abstract

Today many systems exist to generate geometric models of existing scenes and objects. However, no accurate data about surface appearance such as colors and textures is stored in this process. Such data can be captured as a series of images that, collectively, capture all surfaces of the object.
This work introduces a method to compute a minimal set of camera positions for this purpose. Taking images from the computed positions can then be used to derive a complete set of surface appearance data. A slightly different application of the presented method is the computation of a minimal set of viewpoints for reference images to be used in image-based rendering methods.
First a method to determine an optimal set of viewpoint regions for a given scene is introduced. It uses a hierarchical visibility method to preprocess the scene. Then a technique to find an optimal set of viewpoint regions is presented and the solution is used to derive an optimal set of viewpoints. Results and visualizations of the computed solutions are presented.

1. Introduction

For many man-made objects CAD (computer aided design) data is available today. Typically the data set describes the geometry of the scene exactly, but does not contain accurate or content specific data about surface appearance (such as color or texture). Therefore, a visualization of the object will only show the geometric structure but will not match the appearance of surfaces.
For objects with known geometry but unknown surface appearance, methods to derive textures from pre-recorded images (see e.g. [5]) can be used. This is the case when visualizing existing structures from unreachable viewpoints, visualizing modifications to a real scene, or generating realistic textures for virtual training systems. In general there are some parts of the scene that are not visible in the images, therefore a (heuristic) hole-filling algorithm has to used to generate a complete surface description. For objects with unknown geometry it is possible to reconstruct the geometry from pre-recorded images with image-based modeling methods. One successful approach [5] helps the user to construct an approximate geometric model.
It is hard to judge for a human if a set of pre-recorded images is complete in the sense that every part of every surface and object is visible in at least one of the images. Furthermore, it is time-consuming to set up camera and lighting so that high quality images (e.g. without highlights and/or reflections) can be recorded.
This work addresses the visibility portion of the problem with a method to compute an optimal set of viewpoints from which to capture images. The method assumes that the geometry of the scene is known. Approximate geometry can be used as well, but the quality of the results depends on the quality of the approximation. Taking photographs from all the computed viewpoints will guarantee that every part of every surface is visible in at least one image. Furthermore, the method assumes that *potentially* a spherical image can be obtained from every viewpoint. Compositing multiple images to a spherical image [4] can be used to generate such images.
Note that this work addresses the visibility issues of the problem only. Issues such as field of view, image resolution, surface sampling density, depth of field, and the avoidance of highlights or reflections are not addressed in this article as they are of a different nature.
Another view of the presented work is that it computes a lower bound on the number of reference images needed for image-based rendering and modeling. Image-based rendering is a new computer graphics method that utilizes a set of pre-recorded reference images to produce new views of the scene. The benefit of image-based rendering is that a new view of an arbitrarily complex scene can be generated in time proportional to the number of pixels of the reference images. One possibility is to 'warp' each pixel of the reference images into the destination view. For more detailed explanations of image warping see [3], [12]. A prominent open question in image-based modeling is how many images

are needed to represent a scene completely? Phrased differently, how many reference images are needed to be able to render the scene from any viewpoint without artifacts from missing information? The presented method computes a lower bound on the number of needed images and computes an (near-) optimal set of viewpoints for scenes with known geometry.
The hierarchical visibility algorithm used in this work is based on the idea of linking all surfaces that are at least partially mutually visible (they 'interact') and to subdivide these interactions based on the potential error in visibility. A similar idea was introduced previously in the context of hierarchical radiosity [10]. Most recently, Drettakis and Sillion [6] exploited a similar method (termed line-space hierarchy) to store visibility information in a scene. In the hierarchical radiosity literature the (smaller) parts of a subdivided surface are called elements, and this convention is used here, too.

2. Imaging the Set of Visible Surfaces

The method operates on a predefined scene with known geometry. To make the problem tractable, the volume of all possible viewpoints for the later rendering stage needs to be defined as input. This volume is subdivided into a given number of smaller regions. For each of these sub-regions, visibility of all surfaces of the scene is pre-computed for use in the optimization phase. After computing a set of near optimal regions a set of near optimal viewpoints is derived from this solution.

2.1. Viewing Regions

A viewing region is defined as a volume of empty space with the interpretation that it represents the union of all viewpoints inside this volume. The volume of all possible viewpoints is called the *global viewing region.* This global viewing region is subdivided later into smaller viewing regions. Figure 1 depicts possibilities of global viewing regions for some simple scenes.

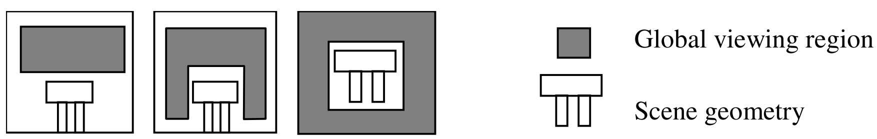

Figure 1: Examples for global viewing regions (see text).

The global viewing region is not restricted to be a convex volume - it might even consist of discrete parts. The only requirement is that it can be subdivided into a pre-defined number of smaller viewing regions. The global viewing region can also be a polygon in space. For instance this happens when the viewpoint is constrained to remain at average eye height level, a common restriction in many walkthrough systems.
There is a set of limitations for the global viewing region as it is impractical to assume that the camera can be positioned everywhere in a real scene. One obvious limitation is that the camera cannot be placed inside objects. To guarantee that a given global viewing region is valid, the global viewing region may be tested for intersections with the known geometry. Also for real cameras a certain minimum distance from each object has to be observed due to the non-zero physical size of the camera and the necessary distance to focus. This can be accomplished by generating appropriate offset surfaces, intersecting the global viewing region with the offset surface and classifying the results.
The global viewing region is subdivided hierarchically into a pre-defined number of viewing regions. A general property of a (global or subdivided) viewing region is that the set of surfaces (or elements) visible from this volume is the union of all surfaces visible from all possible viewpoints inside this volume. Using this property we can define a sufficient set of viewing regions as a set that 'sees' every part of every surface at least once. An optimal set avoids redundancies as far as possible and has a minimal number of viewing regions. The same definition applies to viewpoints. A sufficient set of viewpoints 'sees' every surface part at least once. An optimal viewpoint set has a minimal number of viewpoints. Figure 2 shows a simple scene and an associated global viewing region. An example of an optimal set of viewing regions and an optimal set of viewpoints is shown.

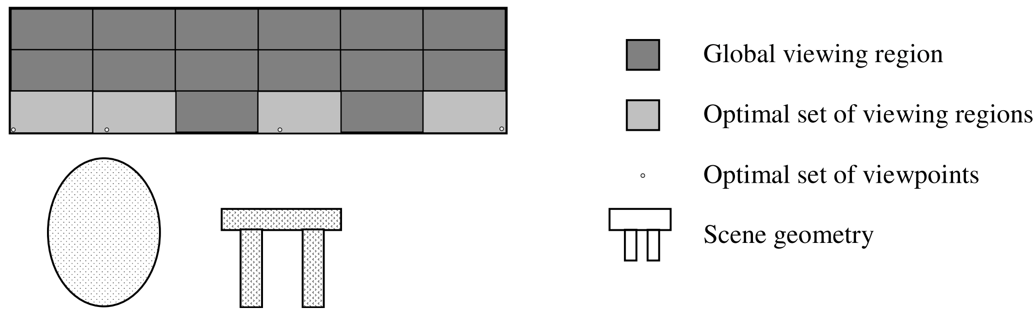

Figure 2: Scene, global viewing region and optimal set of viewing regions.

Note that the optimal set of viewing regions is not necessarily unique. There may be other sets with the same number of viewing regions that provide an equally good solution. An important property of an optimal set of viewing regions is that *at least one* reference image has to be taken from inside each region to ensure the correctness of results during rendering. It *may* also be necessary to generate multiple reference images from inside a particular viewing region, to cover all possible viewing angles. But with decreasing volume the probability that multiple viewpoints are needed decreases as well.

The output of the presented method is a (near) optimal set of viewing regions and an (near) optimal set of viewpoints. The optimality of the solution depends mainly on the implementation of the hierarchical visibility-preprocessing step, see the next section

The size of an optimal set of viewing regions is also a good lower bound on the optimal number of viewpoint for a given scene. This is true because with decreasing viewing region size the probability that more than one viewpoint needs to be placed inside the viewing region decreases as well. An informal proof of this can be stated as follows:

- It is clear that the global viewing region will contain more than one viewpoint for nontrivial scenes.
- Subdivision decreases the volume of each viewing region and also splits the set of viewpoints into smaller sets.
- As the viewing regions get very small (epsilon regions), there will be a one-to-one correspondence between viewing regions and viewpoints.

Therefore with a decreasing viewing region size, the number of viewing region will provide an increasingly better lower bound on the number of necessary viewpoints. However, it is impractical to decrease the size of the viewing regions to the point where it can be guaranteed that only one viewpoint is needed per viewing region. The most prominent limitation is that visibility pre-processing gets progressively more expensive for deeper subdivisions. The alternative of evaluating the visibility on the fly is very costly, too, as the number of visibility computations needed to find the optimum grows quickly with subdivision depth.

2.2. Hierarchical Visibility

Assume that the scene is given as a set of surface polygons organized in a hierarchy. The interior nodes of this hierarchy are bounding volumes (also known as clusters) and the leaves are surfaces that are subdivided further into elements on demand. The hierarchy can be constructed during modeling or automatically by clustering nearby surfaces together (e.g. [2]). A hierarchical visibility method subdivides this scene hierarchy depending on the relative visibility of objects.

The hierarchical visibility algorithm used in this method starts with two nodes. One is the top level bounding box enclosing the whole scene. The other is the global viewing region. The visibility between these two objects is represented as a link. Each link and with it the objects are subdivided recursively until the potential error in visibility falls below a predefined threshold. For simplicity a regular subdivision pattern is used. Figure 3 shows a visualization of the initial situation and two examples for subdivided links.

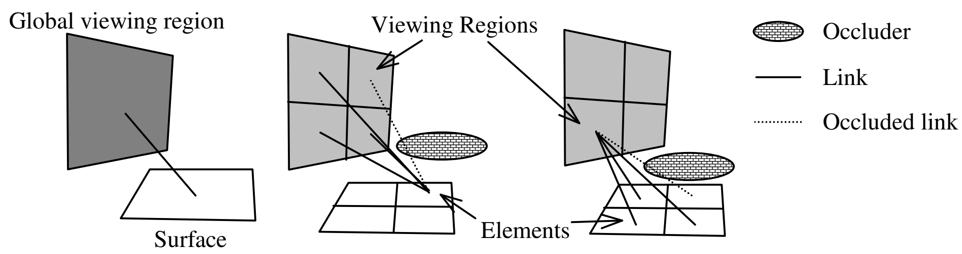

Figure 3: Initial link (left) and examples for refined links.

If there is no occluder between the two nodes a link is stored. Otherwise the larger node is subdivided and the process is applied recursively. Shaft culling [9] is used to speed the refinement process. Note that shaft culling can be optimized in the recursive refinement by temporarily storing the current list of potential occluders during recursive subdivision and using this list during further refinement of the link [1], [6]. For each link created the list of potential occluders is stored for later use. Figure 4 shows pseudo-code.

```
RefineRecursive (Occluder_List)
       If scene object is a cluster
              RefineRecursive (Occluder_List) & return
       Do shaft culling on Occluder_List & create Temp_Occluder_List
       If the occluder list is empty
              Store visible link & return      // totally visible
       If there is one occluder which bisects the shaft
              Return                           // invisible -> no link
       If there is a set of occluders which occlude the shaft
              Return                           // invisible -> no link
       If viewing region and element area smaller than threshold
              Store visible link & return      // maybe partially visible
       Refine viewing region or element and create new links
       Call RefineRecursive (Temp_Occluder_List) for all new links
       Free Temp_Occluder_List.
```

Figure 4: Pseudo-code for hierarchical visibility computation.

If there is one occluder that occludes the whole shaft, the nodes are mutually invisible and subdivision stops. This test is a simple addition to the shaft-culling test [1]. Often there is no single polygon that blocks the shaft completely. This causes unnecessary subdivision and potentially false results - some areas are falsely classified as partially visible. To evaluate the visibility for each shaft exactly, a different method such as the visibility skeleton [14] has to be used.

A different description of the above algorithm is to imagine a hierarchical radiosity algorithm that ignores all energy values when refining links and refines only based on visibility. The presented algorithm is then equivalent to refining one shooting interaction of a hierarchical radiosity system.

The final result of the hierarchical visibility algorithm is a set of links that associates each viewing region to the elements, which are at least partially mutually visible from the region. Note that the visibility information stored in the links is conservative: as long as there is one point in the viewing region that can potentially see a part of a surface, a link is created.

2.3. Determining the Set of Optimal Viewing Regions

The result of the above hierarchical visibility algorithm is a set of links that express the visibility relation between the set of viewing regions and the surfaces of the scene. This information is used to determine an optimal set of viewing regions. The number of combinations to consider increases exponentially with the size of the viewing region set. This prohibits an exhaustive search for non-trivial subdivisions of the global viewing region. Furthermore, it is very important for any optimization method that the benefit of a proposed combination of viewing regions can be evaluated very quickly. Therefore the link information is processed to generate a better-suited data structure.

The method starts by enumerating all viewing regions and all elements by traversing the corresponding hierarchies. Then a two-dimensional visibility array is created and filled. It is indexed by viewing region and element number. To identify all elements that are visible from one particular viewing region

the links of the viewing region and all links of its parents have to be traversed. Each entry is set if a link exists.
By combining the entries of all rows corresponding to a set of viewing regions all elements visible from this set can be computed quickly. Furthermore, combining all rows gives all elements visible from the whole global viewing region. This set is used later to detect if the optimization has reached the optimum.
The problem of finding the best combination of a set of objects is a combinatorial optimization problem. One simple way to define the benefit or 'goodness' of a particular set of viewing regions is the total number of visible elements. As the hierarchical subdivision of the surfaces does not necessarily produce elements with equal areas, it is better to calculate the benefit by summing the areas of all visible elements. This puts more emphasis on finding large visible elements first during the global optimization.
The method calls the optimization procedure repeatedly for increasing numbers of viewing regions. As soon as the maximum set of elements is reached the loop terminates.

2.4. Computing an Optimal Set of Viewpoints

The above computed set of viewing regions is primarily of scientific interest as viewing regions are an abstract concept. To make the results useful for more practical applications the method has to compute a set of viewpoints. But the solution for the viewing regions can be used to derive an optimal set of viewpoints. A simple, yet efficient, heuristic approach was adopted, which works as follows:

```
Benefit (Current_Viewpoint)
        Z = empty                  // list of locally visible elements
        For all links from the current viewing region
                Use stored occluder lists for point-polygon shaft culling
                If linked element at least partially visible
                        Z += element
        Compute benefit value by summing element areas in (X + Z)

Compute_Optimal_Viewpoints ()
        X = empty                  // global list of visible elements
        Loop as long as X does not include every visible element
                Pick the next viewing region (cyclical).
                Optimize for optimal viewpoint inside current viewing region
                Y = set of elements visible from optimal viewpoint
                If Y adds something new to X
                        Add viewpoint to set of optimal viewpoints.
                        X += Y.
        End-Loop
```

Figure 5: Pseudo-code for optimal viewpoint computation.

By cycling through all possible viewing regions the loop in Figure 5 tries to find viewpoints that maximally improve the current solution. In each viewing region under consideration simulated annealing is used to find an optimal viewpoint position. Note that the simulated annealing is here used to optimize for the optimal *position* (floating-point coordinates). The stored occluder lists speed up the benefit computation significantly as shaft culling for the current viewpoint needs to consider a much smaller set of geometry. Full visibility is assumed if the shaft is not completely occluded, therefore the method is conservative.

3. Implementation

In this section a few noteworthy details of our implementation are discussed. The viewing region is currently defined by hand, as an automatic method was considered outside the scope of this work. Our current implementation can also handle only one two-dimensional viewing region due to an early implementation choice. But we can argue that the set of everything visible from an empty cubical volume in space is equivalent to the combination of everything visible from all six faces of this cube. Therefore, the generalization to three-dimensional viewing regions is considered to be relatively straightforward.
The hierarchical visibility pre-processing was implemented by modifying a public domain radiosity package [1]. Ray tracing is used if there is more than one potential occluder between a viewing region (or viewpoint) and an element. If all random rays are occluded our implementation (falsely) concludes

that the shaft is occluded. We considered replacing this part with an implementation of an exact visibility method (e.g. [14]), but the additional implementation effort is substantial.

For efficiency the table encoding the mutual visibility of the viewing regions and elements was realized as an array of bit-vectors. All needed operations can then be expressed as efficient Boolean operations.

To find the optimal solutions the presented method uses simulated annealing. It works by changing the vector of solutions randomly. The magnitude of the potential changes is non-monotonically decreased over time. Simulated annealing has proven to be a very general and efficient global search method in a wide variety of applications and also for combinatorial optimization problems. While it cannot be guaranteed that simulated annealing will find the global optimum it consistently finds at least values very close to the optimum in reasonable time (see e.g. [11]). The interested reader is referred to the literature for further details (see e.g. [11], [13]).

One consequence of using simulated annealing is that our implementation cannot guarantee that it finds the globally optimal solution. In other words the set of viewing regions and/or viewpoints might be larger than necessary. But experiments with a local search method to improve the solution further show that the solution computed by simulated annealing is at least a very good local optimum.

4. Results and Discussion

The experimental results obtained for two living room scenes are presented here. All statistics were measured on a SGI Max Impact and timings are given in seconds.

Figure 6 shows a simple living room scene with 316 polygons subdivided into 1514 elements. The global viewing region was subdivided into 64 viewing regions. An optimal set of seven viewing regions is visualized. The shown intensity encodes how many times each element is visible. Black signifies that the corresponding element is not visible from anywhere in the viewing region (e.g. some elements under the table). The entire viewing region is depicted in dark gray and the set of optimal viewing regions is shown in medium gray. Some images of the following images also show wire-frame outlines to depict the structure of the scene more clearly.

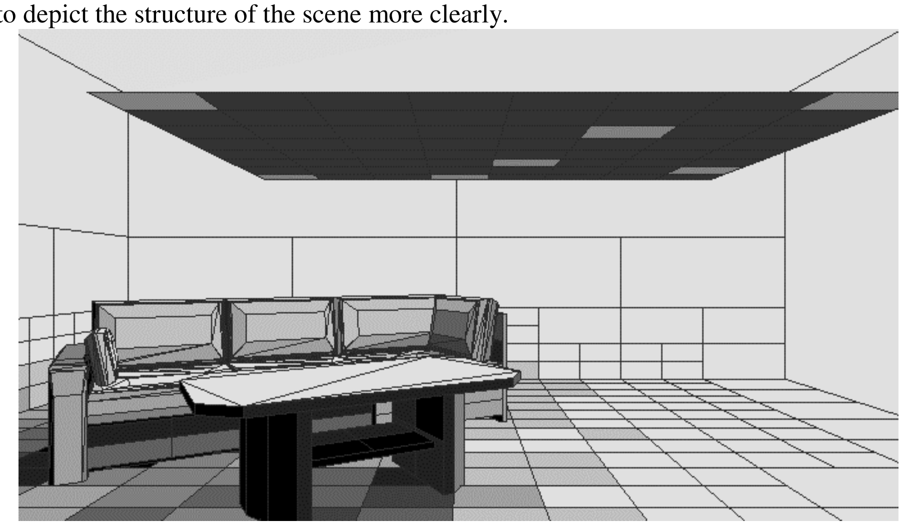

Figure 6: Visualization of solution for simple environment with 7 viewing regions.

The same environment was subdivided more finely into 2822 elements with 256 viewing regions (Figure 7).

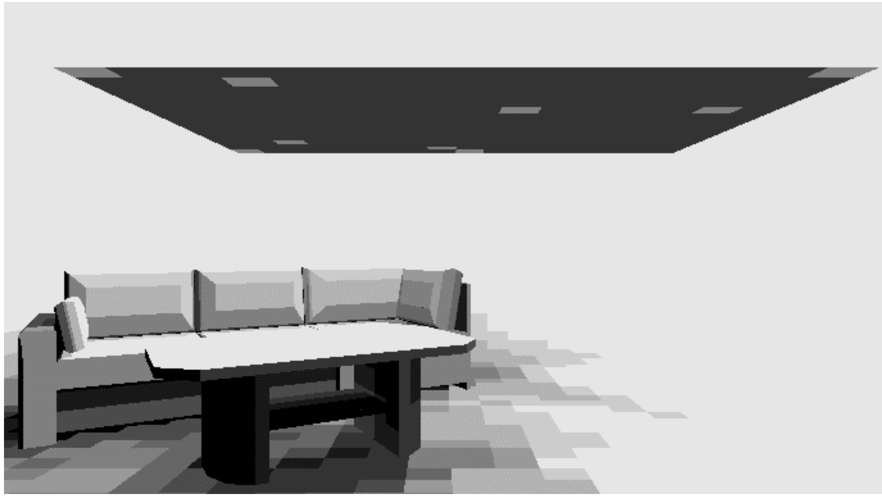

Figure 7: Visualization of solution for simple environment with 9 viewing regions.

Another test with 1024 viewing regions showed that the number of viewing regions does not increase any further, which shows that the result is optimal. One effect that causes the difference in the number of viewing regions between the two shown examples is that the algorithm assumes full visibility if the subdivision limit is reached. Also the visibility is not evaluated exactly in all cases. The result is that a finer subdivision generates a better solution. But under the stated limitations (see the implementation section) the set of viewing regions is optimal.

Figure 8 shows a more complex living room scene with 64 viewing regions (1228 polygons subdivided into 4227 elements).

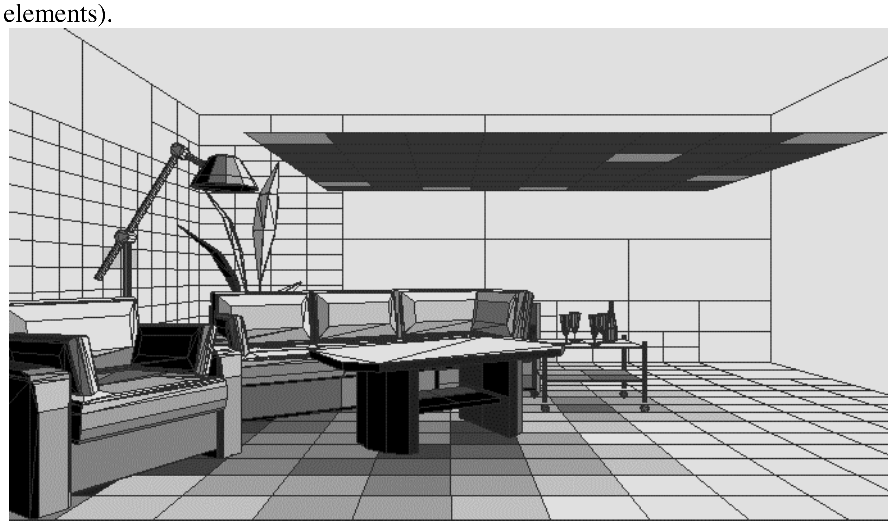

Figure 8: Visualization of solution for complex enviroment with 7 viewing regions.

The same environment was subdivided more finely into 5959 elements with 256 viewing regions (Figure 9).

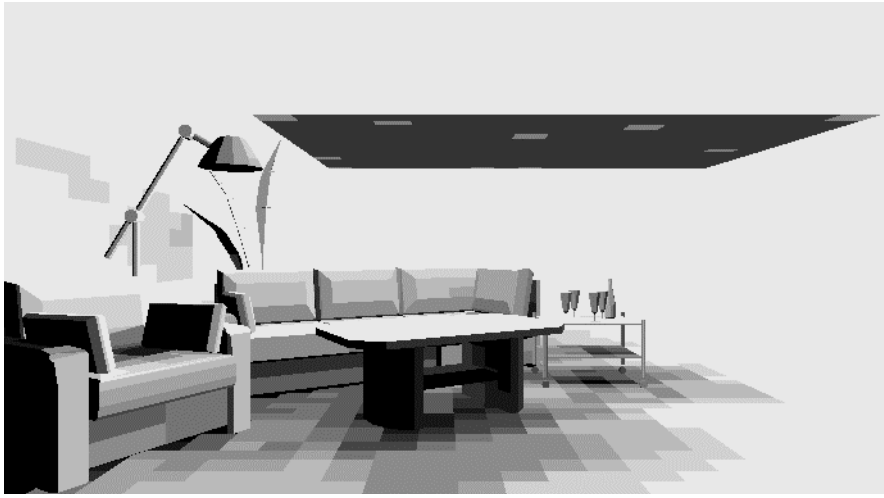

Figure 9: Visualization of solution for complex enviroment with 10 viewing regions.

Again the effects of inexact visibility evaluations are noticeable. An experiment with 1024 viewing regions was aborted after more than one hour of processing due to memory limitations. Table 1 summarizes the statistics for the test cases:

Figure	Elements	Visibility pre-processing time (sec)	Links created	Opt. viewing regions	Optimization time (sec)	Memory (kb)
6	1514	74	18350	7	31	1538
7	2822	311	96936	9	143	5624
7a (not shown)	4690	974	360416	9	374	24799
8	4227	239	59423	7	118	5307
9	5959	817	251381	10	956	16252

Table 1: Statistics for optimal viewing region determination.

Note that a (relative) moderate number of viewing regions suffices for the used scenes. The time spent in visibility pre-processing grows with the complexity of the environment and the subdivision level. The last entry in Table 1 emphasizes that the optimization times grow more than linearly with the number of viewing regions due to the exponential growth in possible combinations. This is also known as the 'curse of dimensionality' in the optimization literature (e.g. [11]). To demonstrate this further, the statistics of the optimization process for Figure 9 are summarized in Table 2:

Number of viewing regions	2	3	4	5	6	7	8	9	10
Percent of optimum	97.11	98.37	99.43	99.64	99.85	99.82	99.97	99.97	100
Visibility evaluations.	360	86	249	404	4805	4296	3452	3429	4402
Time (sec)	10	9	19	25	60	98	191	224	317

Table 2: Optimization statistics for optimization process for Figure 9.

Optimization times increase strongly with the number viewing regions as the more dimensions are added to the problem. The statistics show also that the optimization process quickly finds a near optimal solution for a small number of viewing regions. But the optimal solution can only be found with an appropriate number of viewing regions. In contrast subdividing the surfaces more finely results only in a proportional increase in computation as the number of array entries for the benefit computation is increased linearly.

For the test cases shown in Figure 7 and Figure 9 we also computed an optimal set of viewpoints (again under the limitations of the implementation). Figure 10 and Figure 11 show the viewpoints as small light gray dots inside the optimal set of viewing regions.

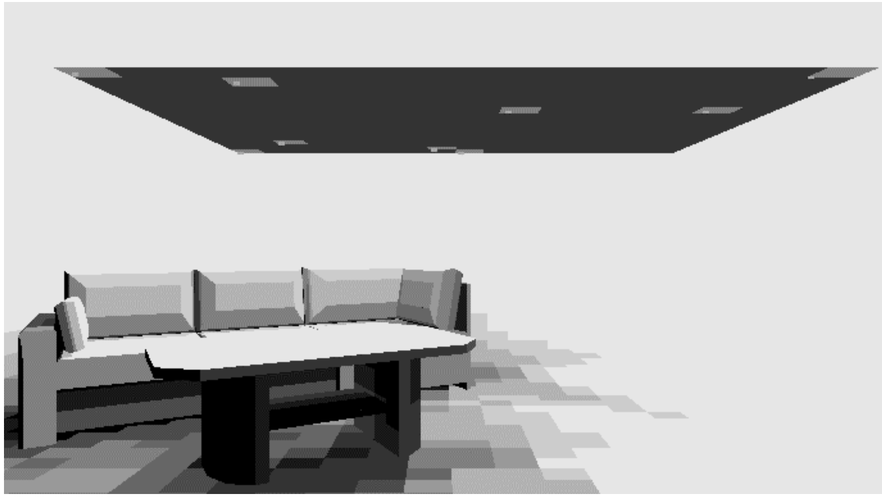

Figure 10: Optimal viewpoint set for simple environment.

Figure 11: Optimal viewpoint set for complex environment.

The method took 166 and 535 seconds to find the shown optimal set of viewpoints. In both cases the number of optimal viewpoints is equal to the number of optimal viewing regions. Therefore we can conclude that this method computes indeed an optimal set of viewpoints. Table 3 shows how long the iterations of the viewpoint optimization for the complex environment took and how optimal the current solution was.

Percent of optimum	91.26	95.72	96.62	97.33	97.48	98.5	99.15	99.52	99.74	100
Optimization time (sec)	52	49	55	87	47	51	47	56	47	44

Table 3: Optimization statistics for optimal viewpoint determination.

Again the system quickly reaches an almost optimal solution. But for a complete solution the full number of viewpoints is needed. Optimization times for each viewpoint are approximately constant as the stored occluder lists simplify the visibility computation.

5. Conclusions and Future Work

This work introduced a method to compute an optimal set of viewpoints for a given scene. First the scene is preprocessed with a hierarchical visibility algorithm. Then an optimal set of viewing regions is computed. Finally optimal viewpoints are found inside each of the viewing regions. Taking images from the computed optimal set of viewpoints will show each part of every surface at least once. Such a set of images can be used to generate a complete set of textures for the photographed scene.

Our current implementation produces sub-optimal results due to inexact visibility evaluations. In our experiments we observed that the computed number of viewpoints is equal to the number of viewing regions, therefore we conclude that our implementation indeed computes an optimal set of viewpoints under the given limitation. If the visibility were evaluated exactly (e.g. with [14]) the method would indeed produce optimal results.

The results are not directly applicable to real applications such as photographing and generating textures for scenes with known geometry as the computed solution relies on the ability to generate (potentially) a spherical image with high enough resolution at each viewpoint. It is easy to show that it might not be necessary to take a spherical image at each viewpoint in general - e.g. the ceiling needs to be photographed only from one viewpoint. But this holds only if it is a homogeneous surface. Phrased differently the frequency of the color variations on the visible surfaces determines the image resolution necessary. A trivial example is that a photograph of a textured wall from across the room will not provide a texture with enough resolution for close-up views. Because of the added complexity of the sampling problem such issues are outside the scope of this work. Still, the size of the optimal viewpoint set is a good lower bound on the number of total images to take for image-based modeling (taking the sampling problem into account can only increase the number).

Future work includes:

- One area for future experimentation is the subdivision level needed to compute results quickly. But for meaningful experiments a correct visibility evaluation method is needed.
- Better tuning of the optimization process. Almost all possibilities for tuning the optimization package were not exploited in the current implementation. It might well be that a significant speedup is possible if the parameters of the global optimization algorithm are tuned appropriately. Another option is to experiment with alternative search algorithms for the combinatorial optimization problem (e.g. Tabu-search [7], [8]).
- The sampling issues and other issues related to real images and photographs (image resolution, field of view, depth of field, highlights, and reflections) need to be handled before the results can be applied (and experimentally verified) in a real system. We are currently exploring ways to incorporate this into the system.

Acknowledgements

Thanks to Anselmo Lastra, Gary Bishop, Mary Whitton for comments on early drafts of this paper. Many thanks to Chun-Fa Chang - he suggested the improved benefit computation method. The author acknowledges support from the Austrian Science Foundation (grant number J01317-TEC).

References

[1] Bekaert P., de Laet F. S., Dutre, P., “RenderPark: A Photorealistic Rendering Tool”, http://www.cs.kuleuven.ac.be/cwis/research/graphics/RENDERPARK, 1997.

[2] Cazals, F., G. Drettakis, C. Puech, “Filtering, Clustering and Hierarchy Construction: a New Solution for Ray-Tracing Complex Scenes”, Proceedings of EUROGRAPHICS ’95, pp. 371-382, 1995.

[3] Chen, S. E., L. Williams, “View Interpolation for Image Synthesis”, Proceedings of SIGGRAPH ’93, pp. 270-288, 1993.

[4] Chen, S. E., “QuickTime VR- An Image-Based Approach to Virtual Environment Navigation”, Proceedings of SIGGRAPH ’95, pp. 29-38, 1995

[5] Debevec, P. E., C. J. Taylor, J. Malik, “Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach”, Proceedings of SIGGRAPH ’96, pp. 11-21, 1996.

[6] Drettakis, G., F. X. Sillion, “Interactive Update Of Global Illumination Using A Line-Space Hierarchy”, Proceedings of SIGGRAPH ’97, pp. 57-64, 1997.

[7] Glover, F., “Tabu Search - part I”, ORSA Journal on Computing 1(3), pp. 190-206, 1989.

[8] Glover, F., “Tabu Search - part II”, ORSA Journal on Computing 2(1), pp. 4-32, 1990.

[9] Haines, E. A., “Shaft Culling for Efficient Ray-Traced Radiosity”, In Brunet and Jansen, editors, “Photorealistic Rendering in Computer Graphics”, Springer Verlag, pp. 122-138, 1993.

[10] Hanrahan, P., D. Saltzman, L. Aupperle, “A Rapid Hierarchical Radiosity Algorithm”, Proceedings of SIGGRAPH ’91, pp. 197-206, 1991.

[11] Ingber, L., “Adaptive Simulated Annealing (ASA)”, http://www.ingber.com/#ASA-CODE and http://www.ingber.com/asa_papers, 1989.

[12] McMillan, L., G. Bishop, “Plenoptic Modeling: An Image-Based Rendering System”, Proceedings of SIGGRAPH ’95, pp. 39-46, 1995.

[13] Press, W. H., B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, “Numerical Recipes in C”, Cambridge University Press, 1988.

[14] Durand, F., G. Drettakis, C. Puech, “The Visibility Skeleton: A Powerful And Efficient Multi-Purpose Global Visibility Tool”, Proceedings of SIGGRAPH ’97, pp. 89-100, 1997.