Comparative Evaluation of Photon Mapping Implementations

Tomáš Lysek*
Supervised by: Pavel Zemčík†

Department of Computer Graphics and Multimedia
University of Technology
Brno / Czech Republic

Abstract

The paper focuses on the photon mapping method, which is one of the global illumination methods used in computer graphics. The paper presents a short summary of the photon mapping method and proposes decomposition of photon mapping into a set of simpler algorithms. Each of these algorithms is evaluated in the experimental implementation in order to identify bottleneck(s) in the method. The results of the experimental evaluation are presented in the paper and some suggestions regarding alternative implementation and/or optimization of the computationally expensive parts are presented as well. Finally, the paper presents the results of some of the optimization and draws conclusions.

Keywords: photon mapping, global illumination, kd tree, nearest neighbors

1 Introduction

Photon mapping is one of the advanced rendering methods, which belong to the global illumination methods. Photon mapping is capable of creation of images using advanced photorealistic elements, such as indirect illuminaton and causitcs. Among the global illuminaton tehniques, photon mapping is one of the fastest one which is able to render highly photoroealistic images. Recently, new techniques for acceleration of computer graphics were discovered. For this reason, it is interesing to test these techniques and algorithms with photon mapping.

Photon mapping was first introduced by Henrik Wann Jensen in 1996 [6]. Jensen also wrote a great book [7] about photon mapping where he is comparing photon mapping with other global illumiation methods and presents advanced techniques in photon mapping such as subsurface scattering or rendering participating media. Many extensions to Photon mapping exists, like Progressive Photon mapping [4] and Stochastic Progressive Photon Mapping [3].

Many researchers worked on fast ray-triangle intersection. Good results were achieved by Wald [11] or Shevtstov [10]. The currently best performance has the new method by Havel [5]. In his paper, very good comparison on ray-triangle intersection methods are shown.

Kd-tree was first presented by J. L. Bentley in 1975 [2]. Using KD-tree with triangles is a little more complicated then using it with points. For this reason, complicated heuristic must be performed. Paper by Wald [12] and Havran is dedicated to fast creation of KD-tree on triangles with complicated heuristics. The paper by Zhou [13] shows even more speedup of KD-tree construction using GPGPU.

For fast nearest neighbor searching, it is possible to combine clasical nearest neighbor searching with aproximate searching. The aproximate searching was first presented by Area and Mount [1] in 2000.

In the presented work, the main focus was on the above techniques that were examined, measured on real photon mapping datasets, the best combination was proposed In order to achieve the fastest solution.

2 Photon Mapping

Photon mapping is a two-pass rendering method. It was introduced by Henrik Wann Jensen in 1996 [6]. It is based on aproximation of rendering equation [8] by calculating the incoming radiance of the selected point, where local illumination model is computed, through the nearest photons. In photon mapping, the Photon is a bigger particle than photon known from physics, that, which carries a certain amount of light energy (higher than real photon) but its behavior is similar to photon. Before searching for nearest photon, photon map has to be created for the whole scene. Photon map is a set of distributed photons on the scene which represents illumination of the scene. For this reason, photon mapping is two-pass rendering method. In the first pass, photons are emitted from light sources. These photons are propagated through scene and if the photon hits a diffuse surface, the value of the photon energy is stored into the photon map. Consequently, these photons are recursively being sent to the scene with direction based on surface characteristics [7]. The photons, which get propagated through any transparent objects, are creating caustics on diffuse surfaces and these caustic photons are

*xlysek03@stud.fit.vutbr.cz
†zemcik@fit.vutbr.cz

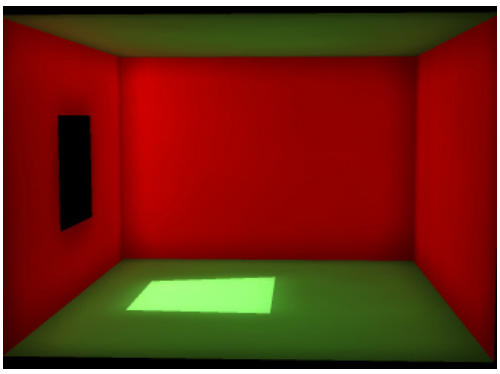

Figure 1: Indirect Illumination

stored in a separate photon map. Caustics are refracted light rays which are concentrated into small areas and they are creating shiny places on diffuse material.

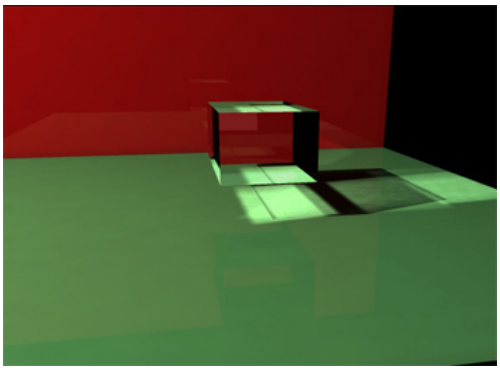

Figure 2: Caustics

Rendering of photon map is possible using several different methods. Probably the most photorealistic results can be achieved by distributed raytracing [7]. Also very good results are achievable by the classic raytracing extended by computing indirect illumination and caustic by nearest photons in photon map [7].

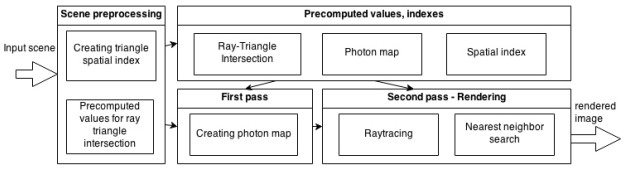

Figure 3: Block diagram

The Photon mapping task is possible to divide into several functional blocks, subtasks, which will be described later and analyzed:

- **Ray-triangle Intersection**
  There are many methods for computing intersection of ray with triangle. In paper Yet Faster Ray-Triangle Intersection [5] is presenting the currently fastest method. Also, during the performed work, it was measured how the another ray-triangle intersection method compares with others. In this comparison, their method has best result. For exploitation of this method, some precalculated values have to be prepared for each triangle.

- **Spatial index (spatial partitioning)**
  Spatial index is data structure which divides space into more smaller subspaces. For each subspaces, all triangles which lie in a subspace are stored in a list connected to the individual subspace. When a ray is being shot through the scene, the ray-triangle tests are performed only on those triangles, which lie in subspaces which intersect with that ray.

  There are many spatial indexes. Performance of each spatial index is highly dependent on scene setup. It is impossible to determine fastest index.

  The most used spatial indexes are octree and KD-tree. Octree is a spatial index, which recursivly divides its space into eight subspaces of the same size. The recursive division is performed to specific level or recursivly dividing is terminated when count of triangles in subspase falls under some specific threshold.

  KD-tree is spatial index which recursively divides space into two subspaces using a dividing plane. This dividing plane is parallel with one of the axes. Several methods exist to determine the dividing plane. The simpler methods include median split on one of the axes (e.g. circularly changed). The more advanced methods use heuristics for determining where the best dividing plane should lie. With these heuristics, is possible to accomplish better results.

  Probably the most used heuritic, used with KD-trees in spatial triangle indexing, is Surface Area Heuristic [13]. This heuristic attempts to maximize the area of subspaces and quantity of triangles in these subspaces.

  Another possible optimization of KD-tree is ropes [9]. Ropes are connections between leafs of the spatial index tree. Using this extension, it is possible to traverse the tree directly through leaf subspaces and avoid slow crawling up and down the tree.

- **Creating photon map**
  In this block, light propagation is simulated from the light sources into the scene and the process results in the photon map. The simulation itself is performed by discrete sampling of light transmission. One sample represents the photon and carries fraction of the light source energy. The light transmission is calculated for example by rejection sampling [7], in which photon is sent from the light source in random direction and then it is propagated through the scene.

  The photons are propagated through the scene similarly to the rays in raytracing. If a photon hits diffuse

surface, energy, direction, and position of the photon are stored. Recursively, another photon from this position is sent further to the scene. The direction of such photon is based on material properties and e.g. if the material is shiny and transparent, two photons are investigated one reflected, second transmitted. For this purpose, an acceleration technique called Russian roulette was created. When the Russian roulette is being used, actions and generation of photon (storage, reflection, refraction) are based on random number with threshold depending on material properties [7].

Photons, which pass through transparent object create caustics. For photorealistic caustic rendering, a large number of photons, which pass through transparent object, is needed. For this purpose, new simulation is started but photons in this simulation go only through transparent objects and save values only to caustic photon map. So, in the end, photon mapping has two photon maps, one for indirect illumination and a second one for caustics.

- **Raytracing**
  For the final rendering of the scene, classical raytracing is used. The values representing indirect illumination and caustics are treated as a local illumination model. These values are obtained by searching N nearest neighbors in photon maps. With increasing N, the quality of the photorealistic results are improved.

- **Finding nearest photons**
  Nearest neighbor search is performed for each computation of local illumination, hence this block is very critical and it has very important role in the rendering performance. The speed of this block is dependent on the size of the photon map and on the number of the neighbors.

  For acceleration of the nearest neighbor search, it is appropriate to use searching index. Many different methods focusing on nearest neighbor search exist. Some of them are available in libraries. One of the interesting ones is ANN Aproximate Nearest Neighbor Library and another one is FLANN Fast Library for Approximate Nearest Neighbors.

  ANN is an older library, this library is used for searching of the KD-trees and BD trees [1]. The FLANN library is used for indexing using the randomized KD-tree. Both libraries are very often used in computer vision for nearest neighbor searching in image features and they are very well optimized for mulidimensional datasets. In our case, analysis in lower dimensionality 3D is needed. Both libraries provide approximate searching search with a small acceptable error is enabled.

3 Experimental Evaluation

The purpose of the experiments is to test selected methods on photon mapping datasets. On these datasets, their performance is evaluated, compared against the other methods, and the method which fit best for photon mapping is then selected.

Experiments which were performed are:

- **Spatial subdivision test** - comparing spatial indexes, octree and KD-tree for acceleration ray-triangle intersection.
- **Creating photon map** compare speed of creating searching indexes on KD-tree and BD-tree in FLANN and ANN libraries
- **Nearest searching - neighbor number** - Comparing nearest neighbor search time with increasing number of neighbors to find.
- **Nearest searching - size of map** - Comparing nearest neighbor searching time with increasing size of photon map.
- **Approximate nearest search** - identification of maximum acceptable error and comparison of the approximate searching times.

All the experiments were performed on laptop with Intel core i7 M620 processor @ 2.67 GHz with 2x 2GB DDR3 RAM 1066Mhz, 7-7-7-20. For compiling, the MSVC 11 (Visual studio 2012) compiler was used. All the measurements were performed on real photon mapping data.

Spatial subdivision test

This test compares speed of spatial indexing methods. In this test, three methods are being compared. First, the naive method with no indexing simply bruteforce method was measured. Then the method is compared to octree and to KD-tree with ropes.

To compare these methods, I created a simple scene with 12140 triangles and performed 100 000 ray-triangle intersection test with this scene. All intersection tests pointing on same spot.

Name	Speed	Precomputing
Naive	47.567s	-
Octree	1.159s	**0.04s**
KD-tree with ropes	**0.453s**	0.16s

Table 1: Comparing spatial indexes

The results of the comparison show that KD-tree with ropes is approximately two times faster than octree. This test also shows that using spatial indexes is indeed efficient and any of the methods outperforms the naive approach.

The KD was 88 times and octree 44 times faster than the naive method.

These test also shows speed of creation of the spatial index. Octree is approximately four times faster than KD-tree. As the spatial index is created only once in this method while the ray-triangle intersections are performed many times - in photon map creating and raytracing. The KD-tree with ropes is generally the best of the tested ones.

Creating photon map

This test compares times of spatial index creation in photon maps depending on the total number of photons in the map. ANN library and FLANN library were used for nearest neighbor searching on multidimensional datasets. From the ANN library, KD-tree index and BD-tree were chosen.

As for the FLANN library, the randomized KD-tree and special single index KD-tree was chosen. Single index KD-tree is optimized for lowerdimensional spaces. This KD-tree is optimized for lower dimensional data and it is called single index.

For this test, I created a simple scene [1] and photon maps with 50k, 100k, 200k, and 500k photons. To generate this amount of photons, I have used photon map block so this test was performed on the real photon mapping data.

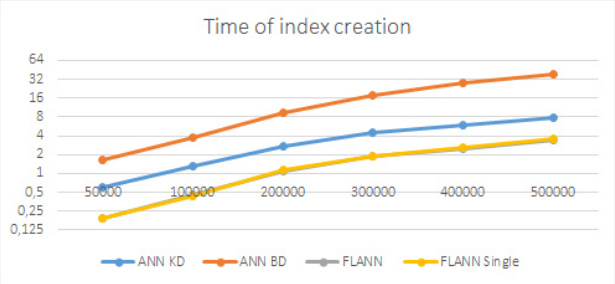

Figure 4: Dependence of map creation time on photon count.

Size of Map	50000	100000	200000	300000	400000	500000
ANN KD	0,591234	1,327776	2,751757	4,408452	6,009744	7,812247
ANN BD	1,660428	3,72888	9,296865	17,34999	27,49657	39,44626
FLANN	**0,187711**	0,458326	**1,066961**	1,878107	**2,476542**	**3,484599**
FLANN Single	0,190411	**0,439525**	1,131965	**1,873107**	2,61855	3,626007

Figure 5: Average times of creating photon map with increasing photons count.

The results show that BD-trees have the worst time of index creation. Both of the FLANN indexes KD-tree and single index have approximately the same time of index creation in fact, KD-tree is little faster than single KD-tree. The ANN KD-tree is approximately five times faster than BD-tree, but two times slower than both of FLANN indexes.

[1] Scene is available at http://lyso.cz/dp/house.zip

Nearest search

This test was performed in order to compare the indexing methods depending on the number of neighbors to search for. The indexing methods for this test were the same as in the previous test - ANN KD-tree + bd tree as well as FLANN randomized KD-tree + single index KD-tree.

The same scene as in the previous tests was used. The photon map with 500 000 photons was created and on this map, the search indices were created. During the testing, the progressively increasing number of nearest neighbor to find were used.

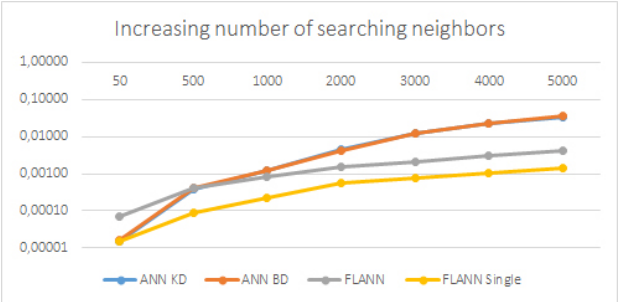

Figure 6: Dependence of searching time on photon count.

N	50	500	1000	2000	3000	4000	5000
ANN KD	0,00001	0,00040	0,00121	0,00452	0,01205	0,02204	0,03394
ANN BD	0,00002	0,00041	0,00121	0,00432	0,01218	0,02281	0,03590
FLANN	0,00007	0,00043	0,00080	0,00151	0,00203	0,00304	0,00410
FLANN Single	0,00001	0,00009	0,00022	0,00054	0,00078	0,00107	0,00141

Figure 7: Average time for searching one photon depending on number of neighbors to search for.

The results show that both ANNs indices and FLANN single index have the best performance for cases in which little number of photons is required to be searched for. However, with the increasing number of photons to search for, ANN is increasingly worse and FLANN KD-tree becomes better than the other two indices.

This test is similar to the previous one but in this case, the number of neighbors is fixed and the size of the photon map is changing. In this experiment, the number of neighbors was set to 5 000. The size of the photon maps ranges from 50 000 to 500 000.

The result shows that increasing size of photon map does not have too big influence on the achieved speed and that the size of the number of photons to search for has large impact on speed.

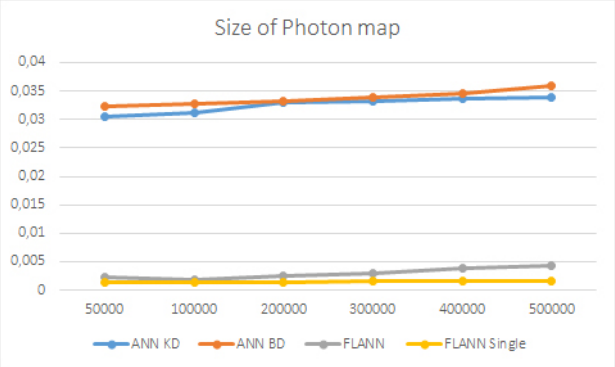

Figure 8: Dependence of searching time on photon map size.

Size of map	50000	100000	200000	300000	400000	500000
ANN KD	0,030481	0,03107	0,032918	0,033227	0,033702	0,03393
ANN BD	0,032248	0,032861	0,033306	0,033909	0,034564	0,035993
FLANN	0,002246	0,001861	0,00254	0,003049	0,003783	0,004382
FLANN Single	0,001426	0,001393	0,001396	0,001489	0,001509	0,001528

Figure 9: Average time for searching one photon depending on number of searching neighbors.

Approximate nearest search

The FLANN and ANN libraries provide also an approximate searching method search in which some error is allowed in the result and which is somewhat faster than the exact case. It should be interesting to find out how much influence the approximate searching has on the quality of rendered images and how much acceleration can be achieved. As the approximate searching leads into worse results in terms of quality, it should be found out how much error is acceptable and then how much it influences the speed.

For this test I created a simple scene, where only the indirect illumination was rendered. This indirect illumination was achieved by search for the nearest photons in the photon map. Photon map size was 500 000 photons and in every calculation of indirect illumination 5 000 photons were used.

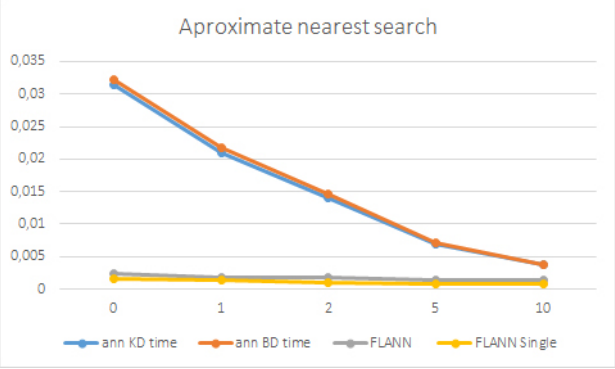

Figure 10: Dependence of searching time on epsilon.

Epsilon	0	1	2	5	10
ann KD time	0,031451	0,020913	0,01414	0,006924	0,003739
ann BD time	0,032258	0,021839	0,014725	0,007142	0,003817
FLANN	0,002356	0,00175	0,001694	0,00145	0,001465
FLANN Single	0,001499	0,001328	0,000981	0,000881	0,000817

Figure 11: Average time for searching one photon depending on epsilon.

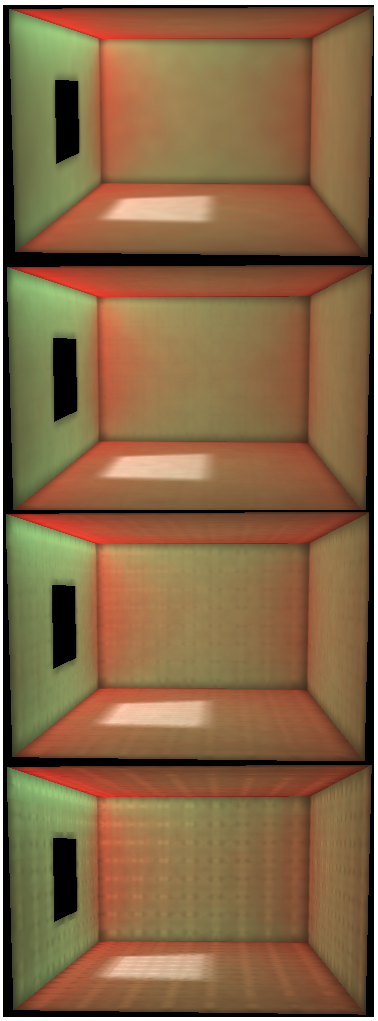

Figure 12: This images shows what influence the allowed error epsilon has on quality of the rendered image. The top images are those with epsilon equal to 0, the second epsilon equal to 1, the third epsilon equal to 2 and the fourth epsilon equal to 4

For measurement acceleration of such approximate searching, another test must be created. For this test, the same simple scene as described above was used. The photon map with 500 000 photons was created and 5 000 photos were used for indirect illumination.

The results show that with the increasing error rate epsilon, the time for searching decreases. The times are decreasing faster when ANN library is used with increasing epsilon but not enough to cause the ANN library to out-

perform the FLANN library. Also, usage of epsilon higher than one has side effects in bad quality of rendered images, as it was mentioned and as it was shown above. For this reason, this approach is generally unusable as it does not produce good enough photorealistic images.

In these experiments, testing of several types of acceleration structures was successfully accomplished. From the results of these experiments, it can be seen what are the best fastest in the application acceleration structures: For spatial index on ray-triangle intersection it is KD-tree with ropes and for acceleration on searching in photon map is the single index KD-tree from FLANN library.

4 Conclusion

In this paper, photon mapping was described along with some selected acceleration techniques. The photon mapping method was subdivided into smaller functional blocks and these blocks were analyzed and their acceleration attempted the acceleration was specifically performed on the slowest blocks of the whole computational process. First experiment was comparing the spatial indices for ray-triangle intersection search. From the results of this text, the KD-tree with ropes was selected as the better one compared to the octree. The experiments with photon map were intended to measure time of creation of the index of a photon map. Several indexing methods were tested and the best performance was accomplished by FLANN indices. Another experiment was performed on evaluation of time for searching for photons in a photon map with increasing number of photons to be searched for. From this experiment single index KD-tree from FLANN library was better. Yet another experiment was performed only to demonstrate that the strongest influence on photon mapping comes from the number of the photons to be searched not size of photon map. Also, when the images are rendered using photon maps, the size of photon map should be bigger. The final experiment was performed on the approximate searching that seemed quite promising. However, this test shows that using approximate searching has side effects in worse quality of rendered image. From all of these experiments on photon maps, it is clear that the single index KD-tree from FLANN library is the best and should be chosen for photon mapping applications. Further work includes more acceleration structures exploration more complex scenes as well as attempt to speedup that could be accomplished by using paralelism for example port photon mapping into GPGPU.

References

[1] Sunil Arya and David M. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17(3-4):135–152, December 2000.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[3] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. *ACM Trans. Graph.*, 28(5):141:1–141:8, December 2009.

[4] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Trans. Graph.*, 27(5):130:1–130:8, December 2008.

[5] Jiri Havel and Adam Herout. Yet faster ray-triangle intersection (using sse4). *IEEE Transactions on Visualization and Computer Graphics*, 16(3):434–438, 2010.

[6] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, pages 21–30, London, UK, UK, 1996. Springer-Verlag.

[7] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[8] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[9] Stefan Popov, Johannes Günther, Hans-Peter Seidel, and Philipp Slusallek. Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum*, 26(3):415–424, September 2007. (Proceedings of Eurographics).

[10] Maxim Shevtsov, Alexei Soupikov, and Er Kapustin. Ray-triangle intersection algorithm for modern cpu architectures. In *in Proceedings of GraphiCon 2007*, pages 33–39.

[11] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics roup, Saarland University, 2004.

[12] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in o(n log n). In *IN PROCEEDINGS OF THE 2006 IEEE SYMPOSIUM ON INTERACTIVE RAY TRACING*, pages 61–70, 2006.

[13] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. *ACM Trans. Graph.*, 27(5):126:1–126:11, December 2008.