Exact Shooting - a two-level overshooting technique for progressive refinement radiosity

Libor Šindlar
Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, 118 00 Praha 1
email: sindlar@sun1.ms.mff.cuni.cz

Abstract:

We introduce a new overshooting technique for progressive refinement radiosity. The technique is based on a relatively cheap computation at low level of accuracy when a scene consists of a small number of planes. The computed radiosity is used when we compute at the high level of accuracy. We solve the radiosity equation at two levels. The high level describes a relationship between pairs of patches, the low level between patches and elements. Furthermore, this method allows us to select whether we prefer the shortest time of the final result or the best quality of the actual result during computing. For displaying of the actual result, the ambient is replaced by the results of high-level computing. Finally, the method is also compared with ambient overshooting and hierarchical radiosity.

Introduction:

One of the most important drawbacks of the classic radiosity method was the great amount of required memory. For solving of radiosity of a scene consisting of *N* elements and *M* patches we had to compute and remember a *M*N* matrix of form factors. This fact restricted the method to scenes consisting of no more than several thousand planes.

This drawback was overcome by the progressive refinement [3]. The progressive method does not remember form factors. Instead, they are computed again if we need them once more. The method spends more time but it requires less memory. It depends on the scene (and especially on the average reflection) how much time the method needs. The final result usually requires 2-10 times more time than the classic full-matrix radiosity.

The progressive radiosity shoots energy from patches at elements. As patches are selected according to their unshot energy, the primary light sources are processed first. Then the method continues by processing of less important patches. During the whole computation, patches are receiving radiosity from an environment. Therefore, it is usually necessary to shoot the radiosity from most of patches more than once. To minimize the number of shots, the overshooting technique [6] is often used. Overshooting tries to estimate the amount of radiosity that the patch will receive in later iterations. Such estimated radiosity is then shot together with the actual unshot radiosity. There are several overshooting techniques [4]. One of the best methods is the ambient overshooting which will be used as our reference. The ambient overshooting estimates the amount of radiosity received in later iterations according to the global ambient.

Radiosity and accuracy of computing

We can find out two main goals when we use the progressive radiosity for computation of the light distribution in a scene. The first goal is to achieve the best result in a given time. The second goal is to minimize time for computation of the final image. Until now, these two goals were usually mixed together. We talked only about the speed of convergence and during computation we always wanted to make the biggest step toward the final result [7]. On the other hand, our method allows us to select one of these goals and then we can achieve a significant speedup with respect to the selected goal. The details will be described later.

Inaccuracy of the actual solution in progressive radiosity can be roughly divided into two main types. The first one is an average radiosity over the whole patch. This average value depends on the radiosity of all patches in the scene. Either it can be estimated by the ambient light or computed by relatively great amount of radiosity shots. The second type of error is an exact description of the shadow and semi-shadow boundaries. The most important shadows are created by occlusions of the primary light sources and the most luminous planes. Therefore, this second type of radiosity error can be radically decreased by several exact shoots from such planes. Into this type of error, we can also classify radiosity variance caused by a small distance of patches (corner of a room).

If we want to increase the accuracy of shadow boundaries, we have to compute at lower level of the plane hierarchy and shoot radiosity from patches at elements. As the number of elements can be very high, we must compute form factors with high accuracy. For example, we can use a hemicube [3] with high resolution. Another reason, why we must compute form factors at high level of accuracy, is the adaptive subdivision of patches. As the adaptive surface subdivision can create very small elements, we must use a large hemicube to avoid aliasing.

If we want to compute the average radiosity over a whole patch, we can use form factors computed with less absolute accuracy (but with the same relative accuracy). It enables us to use less expensive algorithm and therefore we can carry out more radiosity shots within a given time.

In conclusion, to achieve the best result, we must carry out several high accuracy steps and a lot of low accuracy ones. The results of high-level iterations between pairs of patches give us a very good estimation of the average radiosity for each patch and it is used for the shooting at elements. This technique increases the quality of the actual result and decreases time needed for the computation of the final result. If the solution of the computation at the level of patches were accurate, it would be enough to shoot from all patches at elements exactly once.

The ExactShooting method

The progressive refinement radiosity stores for each element its radiosity B_i. For patches, the classic method stores also their emission E_i and the amount of already shot radiosity S_i. Radiosity of each patch is a sum of radiosities of all elements the patch consists of. As our method solves explicitly also relationships between patches, it must remember more information about them. For each patch, the method stores also radiosity PB_i and already shot radiosity PS_i. These values are used only for computation at the level of patches and we don't store any additional information about elements.

The number of elements is usually several times bigger than the number of patches. Moreover, besides the radiosity-like values we must also remember for each plane some information about its position in the hierarchy of planes and points creating its corners. Therefore, we can suspect only a small increase in memory demands (usually less than 10%).

The basic loop of our method can be briefly described by following pseudocode. The detailed description of all important functions can be found later in this paper.

```
while (!Converged) {
       if (WorkAtTopLevel()) {
              SelectTopPatch();       // select the patch with $max(Est(PB_i) - PS_i)$
              ComputePatchFF();       // compute form factors from patch to patches
              ShootAtPatches();       // shoot radiosity at the other patches
       } else {
              SelectBottomPatch(); // select the patch with $max(Est(PB_i) - S_i)$
              ComputeElementFF();     // compute form factors from patch to elements
              ShootAtElements();      // shoot radiosity at elements
       }
       if (requested) DisplayResult(); // display result
}
```

At first, we'll briefly describe functions similar to their respective functions in the classic progressive refinement radiosity. Function `ComputeElementFF()` is exactly the same as the classic function for computation form factors. Function `ComputePatchFF()` is similar but it doesn't take into account the division of patches and it uses a smaller hemicube. Therefore, this function is much faster and this is the main reason why our method is faster than the classic one. For any reasonable method for computation form factors, the required time increases with the number of planes and requested accuracy. Therefore, we can use any such method here and we aren't restricted only to the hemicube.

We use the classical ambient overshooting when we compute at the level of patches. During the computation we use the total unshot energy $sum(PB_i\text{-}PS_i)$ and average reflection ratio to compute the ambient. This ambient is then used in the function `SelectTopPatch`() for selecting of the shooting patch. Furthermore, the ambient is also shot from this patch at another patches together with the actual unshot radiosity in the function `ShootAtPatches`().

When we compute at the level of elements, the function `SelectBottomPatch`() uses the results from the computation at the level of patches for selecting of the shooting patch. If the number of shots at the level of patches is always greater than the number of shots at the level of elements, this method gives more accurate estimation of the patch radiosity. We use the patch ambient to compute the estimated patch radiosity $Est(PB_i)$. Then the shooting patch is selected according to the difference $Est(PB_i)\text{-}S_i$ between estimated radiosity at the level of patches and already shot radiosity at the level of elements.

After the function `ShootAtElements`() have shot the radiosity from patch at elements, we can use the already computed form factors at the level of elements to compute form factors at the level of patches. This technique saves time as these form factors need not be computed and it also ensures that the number of shots at the level of patches is never less than the number of shots at the level of elements.

How to combine results from different levels for displaying

Until now, we used the results of our computation at the level of patches only for the exact overshooting. Nevertheless, one of the most important features of the progressive refinement radiosity is the ability to display the current approximation at the time of computing. The scene is dark at the beginning and the ambient is usually added to currently computed result. As the whole iteration processes, the ambient goes to zero and we can see increasingly accurate results.

Our method need not rely on the rough ambient. We have done a lot of iterations at the level of patches and we have a much more accurate knowledge about the radiosity of each patch. Therefore, we replace the ambient by the actual estimated radiosity at the level of patches. Note that we still use the patch ambient to compute the estimated patch radiosity. Nevertheless, as the number of shots is much greater, our estimation is more exact.

The question is how to combine the actual results at both levels. Our goal is to receive a rough estimation of radiosity within a short time and then improve the exact shape of shadows and semi-shadows. In other words, we should begin with displaying of the estimated radiosity computed at the level of patches. Later, we can add results computed at the level of elements. Finally, these exact values will totally replace the rough results from the level of patches.

Recall that the actual estimated radiosity of the *i*-th patch is denoted as $Est(PB_i)$ and the actual radiosity of the *j*-th element (based on exact shots from patches at elements) is B_j. For displaying of the actual result we won't use the estimated patch radiosity $Est(PB_i)$. Instead, we define a new „display patch radiosity" DPB_i as a part of $Est(PB_i)$

$$DPB_i = Est(PB_i) * \left(1 - \frac{Est(PB_i) * A_i}{\sum_k B_k * A_k}\right),$$

where B_k are radiosities of all elements the *i*-th patch consists of. Radiosity of each element can be obtained by adding the actual result at the level of elements (denoted as B_j) and just described display patch radiosity (DPB_i). We define a „display element radiosity" DB_j as

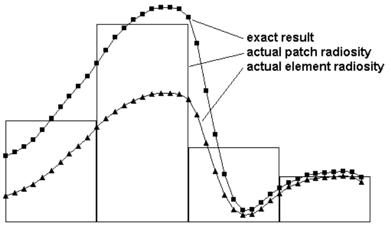
$$DB_j = B_j + DPB_i .$$

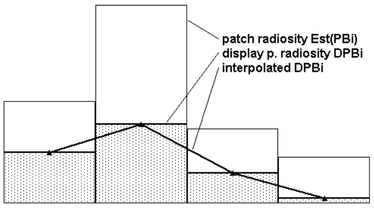
In other words, if the sum of emitting power computed at the level of elements is 80% of estimated power of the whole patch, we add 20% of the estimated patch radiosity to these element radiosities. Note that during the computation, the display patch radiosity goes to zero and sometimes it can also be negative.

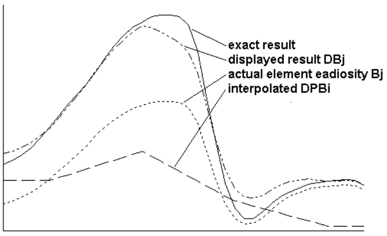
As we have added a constant radiosity over the whole patch, the element radiosity computed by this way would mark the borders between patches. Better results can be achieved if we replace the constant radiosity over the whole patch by its linear interpolation (or higher order interpolation). We define $IDB(DPB_i|B_j)$ as the interpolated display radiosity of the *i*-th patch at the *j*-th element. This estimated radiosity depends also on the neighbouring patches and it must contain, for example, the adaptive subdivision of patches. The new adjusted expression for computation of the element radiosity for displaying is

$$DB_j = B_j + IDB\left(DPB_i \middle| B_j\right).$$

The whole process of the combining of actual results at both levels is on the next pictures.







Scenes we used for our testing

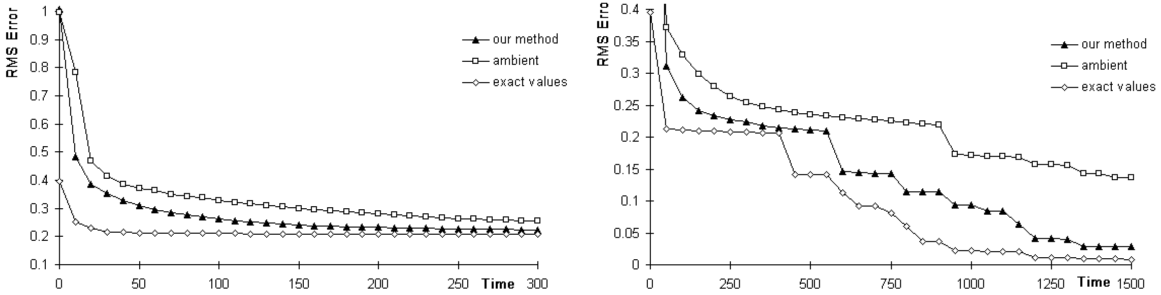
This method was tested on two scenes of different kind. The first scene was a room with several pierces of furniture inside. There was only one primary light source positioned near the ceiling. Most of planes pairs could „see“ each other and form factors matrix was relatively dense. The second scene was a maze. The maze consisted of a several corridors and there were four primary light sources there. As the structure of the maze was more complicated, the form factors matrix was sparser. We wanted to know how our method depends on the average reflection factor. Therefore, we created also the third scene as a maze with higher average reflection.

scene	# patches	# elements	hemicubes	average reflection	non zero FF
maze	2448	37994	200 / 1050	<0.58,0.69,0.51>	8%
room	1792	20394	150 / 800	<0.52,0.58,0.43>	19%
lit maze	2448	37994	200 / 1050	<0.69,0.87,0.61>	8%

Table 1: Scenes used for testing of our method.

The best possible overshooting

As we design an overshooting technique, we should know what is the maximum possible speedup of any such technique. There is a trivial bound. Nothing can be better than the method that knows the exact radiosity of each patch already at the beginning and shoots from each patch exactly once. Of course, as we don’t know the radiosity in any real method, we cannot be as efficient as this method. Another bound for our method is the average ratio of time required for shots at the patch and element levels. This bound could be broken only if we remembered the already computed form factors at the level of patches. The next two graphs show three methods - classical ambient overshooting, overshooting with apriori known patch radiosities and an example of our method. Our method carry out 5 shots at the patch level per one shot at the element level here. As we can suspect, our method lies somewhere between the ambient and exact overshooting.



Graph 1. Convergence of ambient overshooting, the best possible overshooting and our method

How to measure the speedup

Similarly as in the ambient overshooting, the speedup of our method depends also on the displaying scene. Generally, our method beats the ambient overshooting when we compute a scene where the ambient is too rough, e.g. a scene with light and dark regions or a scene with highly varying reflection factors. As in the ambient overshooting, the efficiency of our method increases with the average reflection. The efficiency of our method also strongly depends on the ratio of the number of elements and patches.

To be able to talk about efficiency, we must find out a way how to measure it. The first problem is how to choose an independent variable. As our method contains steps that take different time, we cannot use the number of shots as the independent variable. Instead, we must use the time. In order to be comparable with the results of other radiosity methods, we at first had to find out the average time of a radiosity shot. For our scenes, one shot took usually 5-10 seconds on PC with Pentium processor running at 150 MHz. When we created our graphs, we divided the obtained time by this average value. Therefore, our independent variable has similar values as usually but it is computed according to the consumed time.

As the dependent variable we used the RMS Error which measures the distance between actual and converged result (converged radiosity is denoted by *B’* here)

$$RMSError = \sqrt{\frac{\sum A_i * \left(B_i - B_i^{'}\right)^2}{\sum A_i}}\ .$$

Unfortunately, this measure doesn’t represent the real visual quality of the actual result. For example, this measure prefers the ambient overshooting. If we compare two pictures with the same RMS error, one created by ambient overshooting while the second without any overshooting, the second image has a better visual quality [6] because this one was created after a higher number of radiosity shots. The relationship between our method and the RMS error is much more complicated. We carry out a lot of patch to patch iterations but much less patch to element iterations. Therefore, for the same RMS error, our method prefers the color blending and discriminates the exact shapes of shadows.

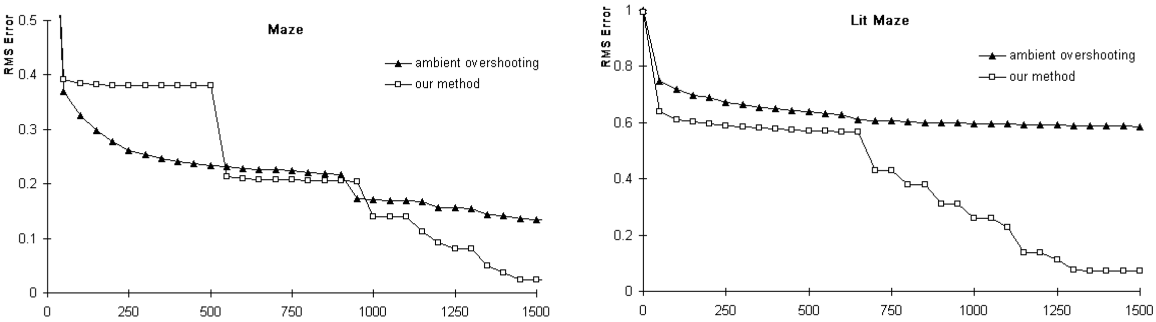
When to shoot at the element and patch level

Probably the most important and the most interesting part of our method is an implementation of the function `WorkAtTopLevel()`. This function says whether the next iteration should be performed at the level of elements or patches, and the right behavior of this function is crucial for the overall efficiency. Its result depends on the computed scene, the actual solution and the requested accuracy.

If our primary goal is the final result, the best idea is to divide the whole computation into two parts. At first, we compute at the level of patches and we find out the accurate radiosity for all patches.

Then we use these already computed results for exact overshooting at the level of elements. In optimal case, it is enough to shoot exactly once from each patch. As we use less accurate form factors in the first phase, the outcome need not be exact. This is especially true for small patches where the relative error is biggest. Nevertheless, as the small patches have usually only a small influence to the whole result, we need not shoot more times neither from these patches.

The next graphs show the convergence of our method when our primary goal was the final result. At first, we computed the starting value of ambient and then our method worked at the level of patches. When the patch ambient had dropped under 0.01% of its starting value, we switched to the level of elements.



Graph 2. The convergence for *Maze* and *Lit Maze* when the primary goal was the final result

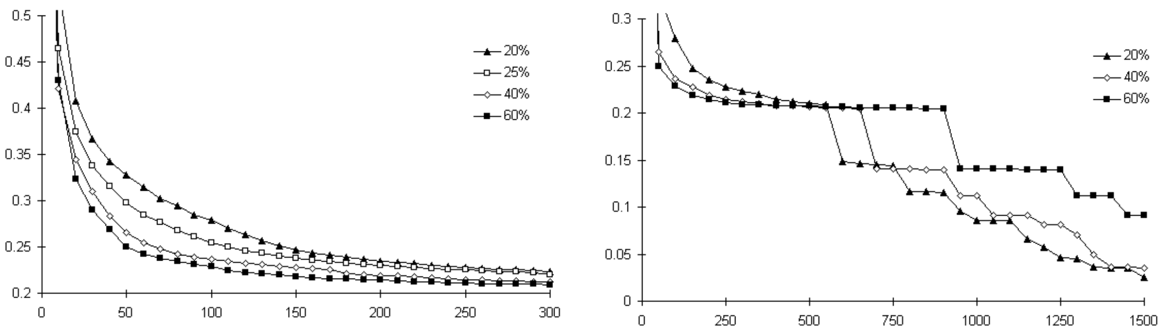
The next table compares the time required for computation of the final result by the ambient overshooting and our method for all three scenes.

scene	time of ambient overshooting	time of our method	achieved speedup
Maze	7070	1930	3,66
Lit Maze	22390	2970	7,53
Room	4780	2250	2,12

Table 2. The speedup of the computation with the final result as the main goal

The more difficult case becomes when we want to speedup not only the computation of the final result but also the computation of the current result. As we could see, the previous method may be even worse than the ambient overshooting for the current result. This is true especially at the beginning of the whole process. To improve the actual result, we must start working at the level of elements sooner than in the previous method. Nevertheless, if we start the shooting at the level of elements sooner, we'll have only inaccurate knowledge about patch radiosity and we'll have to shoot more times from some patches.

In the first method, the function `WorkAtTopLevel()` makes its decisions according to the consumed time. At first, we must find out how many times the shooting at the level of patches is faster than the shooting at the level of elements. This can be done by apriori estimation or experimentally according to several first shots. Then we can use this outcome and declare how much time we'll spend by computation at each level. In other words, to declare how many patch shots we carry out after one shot at elements. Recall that in order to ensure correctness of our method, the number of shots at patches cannot be less than the number of shots at elements. The next graphs show the convergence for different amount of the time spent by computation at the level of patches.



Graph 3. Speedup for different amount of time spent by shooting onto patches in the scene *Maze*

As we can see from these graphs, the constant ratio of time doesn't give the best possible result. The better idea is to start our work at the level of patches and quickly compute the rough estimation of their radiosity. Later, as the work at this level continues, it receives less and less time. Finally, we shoot only at elements. This strategy is partially similar to the case when our primary goal was the final result. Nevertheless, in this case, we start the shooting at elements immediately (although at the beginning we don't shoot at elements as often as at patches).

The essential problem is to find out a function that gives the optimal ratio of the time spent at both levels. This function depends on the scene and the actual solution. As we have not available the final result during the computation, we cannot use the RMS error as the measure of the convergence. Therefore, we measure the convergence according to the estimated relative unshot radiosity RB_{unshot}

$$RB_{unshot} = \frac{\sum_i A_i * \left(Est(PB_i) - S_i\right)}{\sum_i A_i * E_i} .$$

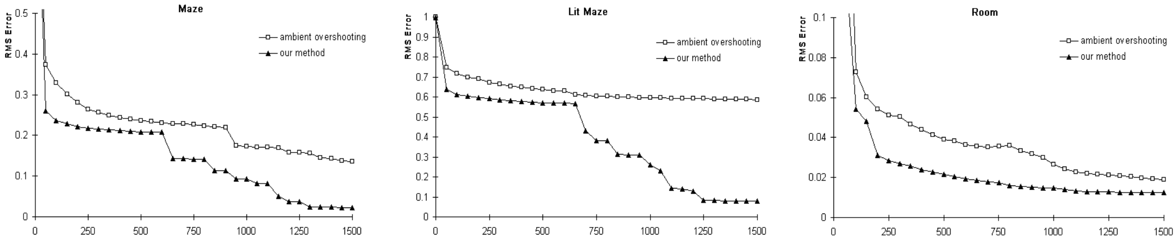
Finally, we define the ratio of the time spent by computation at both levels according to the relative unshot radiosity as

$$T_{patch} = 1 + \frac{\log_{10}\left(RB_{unshot}\right)}{4} .$$

Note that this function takes into account also the average reflectivity in the scene. This dependency is hidden in the estimated patch radiosity $Est(PB_i)$. We use the ambient overshooting for computation of this estimated radiosity, and the ambient depends on the average reflection factor.

This function estimates how far we are in the whole iteration process. At the beginning, the shooting at the patches receives 75% of time. As the whole process continues, the computation at the level of patches receives less and less time. When the relative unshot radiosity drops under 0.0001, then the computation at the level of patches is supposed to be fully converged and we continue only at the level of elements.

The behaviour of this function for all scenes is presented in the next graphs.



Graph 4. Our optimal method when the primary goal was the current result and ambient overshooting

Conclusions and future works

An efficient overshooting technique for progressive refinement radiosity has been presented here. The whole computation is divided into two steps. In the first step we use a relatively cheap rough computation to estimate the radiosity for each patch. This estimated radiosity is shot at elements in the second step. As the estimation from the first step is relatively accurate, we need not shoot from patches more times.

Another advantage of our method is an ability to choose our primary goal. The method allows us to select whether we want the final result in the best time or we can wait longer for the final result but have sooner a good approximation of the result. If the final result is our main goal, then we firstly compute the fully converged result at the level of patches and then we shoot this computed radiosity at elements. This technique ensures that we can shoot from each patch exactly once. On the other side, if we want to see a good approximation as soon as possible, we can combine the shooting at both levels during the whole computation. We'll have to shoot more times from some patches but we'll soon get a good description of the shadow boundaries. The most important shadows are created by the most luminous planes and this method combines these most important shadows together with a good knowledge about radiosity of each patch in the scene.

If we compare our method to the classic ambient overshooting, our method is several times faster. The total speedup depends on the character of the scene. Generally, if the scene is less suitable for ambient overshooting we can reach a better speedup.

We can also compare our method with the hierarchical radiosity HR [5]. Both methods work at more levels of the plane hierarchy, and memory requirements of both methods are linear. On the other side, the HR can better exploit the estimated importance and estimated error of links between planes. The importance and error of a link are usually estimated by inaccurate heuristics such as visibility testing or distance of planes. Our method works only at two levels of hierarchy and therefore it cannot exploit the importance of each link. Nevertheless, as our method can use straightforward and fast algorithms for computation of form factors (such as hemicube), this disadvantage is partially compensated. The advantage of our method is a continuous improving of the actual result. In the HR, we must firstly choose a maximal error bound and the whole computation is based on this error. Our method starts with a less accurate solution but the solution is improved over the time and it depends only on our decision when we say „stop, the result is enough accurate".

In the future, we would like to develop a parallel version of this method. As the method soon obtains a good approximation of the radiosity of each patch, we could probably achieve a good efficiency even on massive parallel computers. This would be important especially because the hierarchical radiosity is difficult to parallelize. For example, Bohn [1] achieved speedup less than 9 on 64 processors.

Literature

[1] Christian A.Bohn, Robert Garmann: *A Parallel Approach to Hierarchical Radiosity*, Proceeding of The Third International Conference in Central Europe on Computer Graphics, Plzen February 1995, p.26-33

[2] Michael Cohen, Donald P. Greenberg: *The Hemicube: A radiosity solution for complex environment*, Computer Graphic (SIGGRAPH '85), Vol.19, No.3, p.31-40, August 1985

[3] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, Donald P. Greenberg: *A Progressive Refinement Approach to Fast Radiosity Image Generation*. Computer Graphics (SIGGRAPH '88), Vol. 22, No.4, p.75-84, August 1988

[4] Martin Feda: *Speeding Up Progressive Radiosity by Overshooting*, Proceeding of The Third International Conference in Central Europe on Computer Graphics and Visualization 95, Plzen February 1995, p.87-92

[5] Pat Hanrahan, David Salzman, Larry Aupperle: *A Rapid Hierarchical Radiosity Algorithm*, Computer Graphics (SIGGRAPH '91), Vol.25, No.4, p.197-206, July 1991

[6] Werner Purgathofer, Martin Feda: *Accelerating radiosity by Overshooting,* Third Eurographics Workshop on Rendering, p. 21-32, Bristol May 1992

[7] Rodney J. Recker, David W. George, Donald P. Greenberg: *Acceleration Techniques for Progressive Refinement Radiosity*. Symposium on Interactive 3D Graphics, Vol.24, No.2, p.59-64, March 1990