Ben Ball
Yuriy Stejko

# Final Project Report

## Motivation

The photon mapping we implemented in class worked well under most circumstances, however there were some specific situations where the result of rendering did not look entirely realistic [Image 2]. One situation where the problem was apparent was the rendering of edges. When two edges with a sharp change in photon quantity were rendered, photons from the bright edge would sometimes fall within the spherical region of photons for determining the color and brightness of a given point increase the brightness near that edge. The effect would also make the edge of the brighter polygon look darker then is should as it received less photons per unit area. Although one way to solve the brightening problem on the darker edge would be to simply ignore all photons that had an angle of approach of 90 degrees or more away from the normal of the side, this would not solve the problem of darkening bright edges and could cause the edge of the darker side to become even darker than it should be.

Another issue apparent in basic photon mapping is the bleeding of light through walls. When a pixel on one side of the wall collects nearby photons it can collect photons on the other side of the wall. This problem can also be somewhat alleviated by checking the photon direction against the normal of the surface where the eye ray hit, eliminating photons which would have hit the surface from behind. But if two parallel surfaces are separated by a thin wall, checking normals will not be sufficient.

## Related work

Jensen's implementation of global illumination via photon maps (Jensen 1996) includes several optimizations and improvements. He constructed two separate photon maps, one for general illumination and one with higher detail specifically for caustics. For general illumination, he used traditional distributed ray tracing where the contribution to the final image was large, only referencing the photon map on subsequent diffuse bounces where the exact radiance value would not have a significant impact. To limit the number of shadow rays cast during ray tracing, he stored shadow photons which are photons which have passed through an object and continued on their original path. By observing the nearby shadow and normal photons, it is often possible to determine if a point is in shadow without casting a ray to the light source. When there are both shadow and direct illumination photons nearby the result is unclear, and a shadow ray is cast. While these techniques can provide significant improvement in image quality, they will do nothing to directly address the issues identified above, and are not implemented here.

In the paper "Advanced radiance estimation for photon map global illumination" (Hay and Purgathofer 2002) present a modification to the method of gathering the photons to determine the illumination value at the point hit by an eye ray. The method takes a cubic, rather than spherical, region of space and distributes the energy of a photon over all the area facing towards that photon. The distributed energy of all photons is then totaled and returned as the illumination value of the eye ray's target. One of the reasons that this can work better than taking the total of the energies of all the photons and dividing by the area of circle is that if the area being searched is on the edge of a polygon the photons will only be distributed over the actual area which they hit. Another feature which was mentioned in the paper's implementation is the

use of multiple KD-trees, however they only seemed to use this to take advantage of the parallel rendering cluster they had set up and not as a technical innovation.

Another extension to global photon maps was implemented in "Optimizing photon mapping using multiple photon maps for irradiance estimates" (Larsen and Christensen 2003). They split the photons into multiple tree structures, with two mesh faces only sharing a tree if they shared an edge, and the angle across that edge was less than a specified threshold. By using multiple smaller kd-trees, they were able to reduce the photon search time, increasing render speed. Because each eye ray only searches in a single tree, this method also has the benefit of eliminating light bleeding around sharp corners or through walls.

## Algorithms and Data structures

The basic photon mapping and ray tracing algorithms used in this project were those created for Homework 3, based roughly on the techniques of (Jensen 1996) but without many of the more advanced features. When calculating the radiance at a location, this basic algorithm simply totaled the energy of the nearest N photons, and divided this by the area of a circle with the same radius as the sphere needed to capture these photons. No calculations using the photon direction were made.

The addition of threaded ray tracing, while not directly related to the goals of the project, significantly increased the rendering speed, allowing a faster iteration process. Instead of simply looping through a specified number of pixels to render in the animation loop, we started a user specified number of threads, each of which looped through a number of pixels to render. The primary difficulty was in correctly implementing thread safety for the the pixel rendering operation. This primarily consisted of adding locks on key portions of the code, such as updating

the current position in the image being rendered, and access to the random number generator. The actual ray trace operation is complete parallel, since it modifies no shared data. Other sections of the code, such as the generation of photons in the scene, involve modification of a shared data structure which would make parallelization much more complex. Since the majority of time is spent on the ray tracing, only this section was made parallel.

For the portion of the project designed to prevent photons from being taken across edges, as described in motivations paragraph 1, we had to implement several new data structures and algorithms. The general algorithm implemented for this purpose, which was based on the work described in (Hay and Purgathofer 2002), involved finding a cube which enclosed a region containing a given number of photons surrounding an eye ray hit, spreading their energy to all enclosed surfaces, and totaling up the spread energy to get the intensity of the light at the given location. The high level description of the algorithm, although informative, does hide a large amount of complexity.

The most complex portion of implementing the algorithm as described was determining what portions of geometry were enclosed in the cube in the first place. In order to find the area of any partially or fully enclosed face which was inside the cube that geometry needed to be clipped. The algorithm we implemented for clipping to the inside of a cube was a 3d extension of the well-known Sutherland–Hodgman algorithm which involves clipping every part of the face beyond one of the 6 walls of the cube 6 times until either the face was completely removed or only portions of the face that were inside the cube remain.

To perform the clipping to each plane the points which represented the vertices of the face were placed into an ordered linked list, where the ordering was based on the forward

navigation of the provided half edge data structure. As each face could be assumed to be convex polygon any for any two vertices on a face which were behind the clipping plane all vertices between them must also be behind the clipping plane.

Taking advantage of the known property of convex polygons we could create a ray from the last unclipped vertex to the first clipped vertex, a ray from the first unclipped vertex to the last clipped vertex and create two new vertices from the interactions of those rays with the clipping plane. All of the removed vertices, which can range from a single vertex to all but one vertex in the face, could then be replaced by the two new vertices which would then be linked together. After performing the clipping the resulting polygons could take on any convex shape. The best way to determine the area of the shape was to break it into triangles. We did this by taking the first point in the list to be one vertex of each triangle within the shape and taking the next two points as the other two for the first triangle. The two other vertices were moved forward in the list by one each time to assure that the polygon was entirely covered.

After the enclosed area of each face was determined all non-zero areas were placed into a linked list data structure as their area would be needed once for each photon enclosed in the bounding cube. This structure also contained a reference to the original face as well as a cached value of that face's normal to avoid recalculating it during the next stage. After the faces are clipped and stored we simply divide the photon's energy; which is a 3-vector of floats to represent red, blue and green; by the sum of the projected areas of the geometry enclosed in the cube and summed up the results of all the photons in the bounding cube. Projected area was calculated as enclosed area of the polygon times the dot product of the face normal and the

incoming ray (Hay and Purgathofer 2002) multiplied by negative one to account for the opposite direction of the incoming ray of a photon to the face normal.

To implement multiple kd-trees, each surface must be linked to the kd-tree encompassing it. This is done by giving each mesh face a pointer to a kd-tree. Rather than assign each face its own kd-tree, we used a method similar to that in (Larsen and Christensen 2003) to connect multiple faces in the same kd-tree. This allows photons to be shared across parallel or nearly parallel connected faces, where the lighting conditions would be expected to be similar.

The algorithm to link face to trees begins by setting the kd-tree pointer on all faces to NULL. It then loops through all faces, and for any which still have a NULL pointer, creates a new kd-tree and assigns it to that face, then calls a recursive function. This function checks all faces adjacent to the current one, and if the angle between their normals is less than a threshold value, they are also assigned to the current kd-tree, and their neighbors are also recursively checked. In this fashion, all adjacent faces with a small enough crease angle are connected. The recursive function also tracks the minimum and maximum coordinates of every face which is added to the tree, so that the bounding box can be set appropriately. For primitive objects not composed of mesh faces, such as the sphere and ring from Homework 3, each primitive type is mapped to kd-trees manually. The entire sphere is assigned to a single kd-tree, while the ring is assigned four, one each for the top, bottom, inside and outside.

Both when shooting photons into the scene to create the photon map and when searching for nearby photons to make a radiance calculation, the code needs to know which kd-tree to use. For this purpose, a kd-tree pointer is added to the information returned when a ray hits an object. If the object hit is a mesh face, that face's kd-tree pointer is returned. If the object is a primitive,

the code which determines if the object was hit also determines the location, and assigns it to the correct kd-tree. The only changes required in the photon tracing and rendering code are to use this returned kd-tree rather than a global one.

## Challenges

The biggest challenge in implementing the geometry based photon mapping was implementing a way to clip arbitrary polygons and retrieving the area of the remaining region. As clipping for arbitrary polygons was difficult to figure out we decided to limit our work to convex polygons. We consider this acceptable as convex polygons are more commonly used in modeling and any polygon can be broken into triangles in order to make it convex. The algorithm I ultimately implemented for the purpose was a 3d extension of the Sutherland–Hodgman algorithm. Because all of the clipping planes were axis aligned all I had to do was clip against the x,y, or z coordinate.

A portion of the implementation that I did not get to complete was adding geometry to the KD-tree. Although this is not necessary for smaller scenes, it would become more necessary as the number of faces in a scene increases. Part of the problem with adding the geometry to the KD tree was determining if a piece of geometry intersected a portion of the KD tree. Ultimately the clipping algorithm could have been used to determine this, however after implementing the required code for the project there was not sufficient time to implement the acceleration data structure.

The algorithm did not create any problems in the translation between theory and practice, however the linked list data structure required several cases to be accounted for when clipping. The three cases were clipping between two points where all the points linked to one another,

clipping that overflowed from the end of the list to the beginning, and clipping which removed the first value in the list. Originally a problem with the implementation of the second case caused multiple points to be given the same value causing some enclosed regions to be treated as having no area.

When combining the two algorithms into one, our implementation originally resulted in some surfaces being completely black. This took significant time to identify, but turned out to be caused by the case where a photon map contains fewer total photons than the number of photons to gather for a single point. This obviously never occurred with only a single kd-tree. When this happened, the photon inclusion area expanded to its maximum size, including all surfaces in the scene. The geometry based area calculation then distributed the limited number of photons captured on the actual surface of interest over all the surfaces captured, resulting in very little radiance. The solution was to only allow the inclusion of geometry in the area calculation if it used the kd-tree from which we were pulling photons.

## Testing

We constructed three test scenes as well as using the cornell box with reflective sphere scene from homework 3. The first scene consists of two fully enclosed cubic rooms next to each other, one of which has a bright light source on one side while the other is completely dark. The primary purpose of this scene was to highlight the effect of light bleeding through the thin wall due to collection of photons from the other side.

The second scene is similar to the first, but uses different colored lights in each room, and also includes a second smaller cube hovering in the center of each room. This scene also shows the mixing of the two colors along the shared wall when using basic photon maps. On the center

cubes, it also shows the lightening and darkening effects along the edges of a surface due to either light bleeding from adjacent surfaces, or the incorrect area calculation.

The final scene shows an accordion surface lit from an angle, and is based on the test scenes in (Hay and Purgathofer 2002). It is designed to show the edge effects due to light bleeding and the assumption of a circular area. The effect can best be seen in the dark strips on the darker sides of [Image 3a] and the light strip in on the dark side of [Image 3b].

Time spent and work distribution

Getting geometry based photon gathering was implemented by Yuriy Stejko. Getting this portion of the project working took approximately 20 hours of coding and debugging in addition to a couple hours which were spent finding a clipping algorithm which would work for our purpose. In addition to the time spent working on the project, the time to render a given test scene, before Ben Ball implemented parallel ray tracing, was slow enough that it interfered with testing making correct implementation difficult.

The extension to multiple photon maps was implemented by Ben Ball. This was mostly straight forward but required modifications throughout the existing code to allow for the additional storage and transfer of kd-tree pointers, and took a total of about 12 hours. The implementation of parallel ray tracing took an additional 4 hours, spent primarily on debugging simultaneous access issues to ensure the code was thread safe.

The fusion between the two pieces of code was done as a group and took roughly 4 hours to complete. Although the code fusion was largely easy because very few of the edits touched the same parts of the code some incompatibilities in representing corner cases, such as when too few photons existed for the photon gathering, caused code conflicts that had to be fixed.

## Results

We ran all four test scenes with basic photon maps, the multiple photon map improvement, the geometry based area calculation improvement, and both improvements, for a total of 16 test cases. The table below shows timing results.

| Scene | Basic Photon Maps | Multiple Photon Maps Only | Geometry Based Area Only | Both Techniques |
|---|---|---|---|---|
| Cornell Box | 622.1 | 555.0 | 549.5 | 473.6 |
| Accordion Surface | 53.4 | 34.7 | 46.9 | 29.2 |
| Lit and Unlit Rooms | 1412.8 | 137.3 | 1343.5 | 119.7 |
| Different Colored Rooms | 384.3 | 256.1 | 325.9 | 218.7 |

Table 1: Timing results for all combinations of test scenes and rendering techniques. All tests were run at 500 by 500 resolution with 100,000 photons on an Intel i7-4770k with 8 threads.

The timing results show that both improvements give increased performance in addition to better image quality. In the case of the multiple photon maps this is due to the photon search taking place in a smaller map while for the geometry based area calculation it is due to the simplification of using a cube to find nearby photons instead of a sphere. The test scene with two rooms, one lit and one unlit, shows a drastic performance increase in the cases which use multiple photon maps. When the entire scene is contained in one map, each ray shot into the dark room will search through the kd-tree many times until finally reaching the maximum size and giving up. Each of these searches is in the full sized kd-tree. In contrast, when the scene is broken into multiple maps the searches take place in a nearly empty kd-tree.

Figures X-Y show the four scenes rendered with each of the techniques (top left is basic photon mapping, top right is multiple photon maps, bottom left is geometry based area calculation, and bottom right is both. One can observe the effects of light bleeding both through walls and around edges, and the darkening at edges due to incorrect area calculation, and how our methods reduce these effects.

## Conclusion

Our work demonstrated a combination of methods which together remove the edge shading artifacts and the effects of photons bleeding through walls. Although the complete algorithm which uses only photon mapping does result in a significant amount of noise in the resulting image when geometry based area is used to distribute the photons we consider the effect achieved to be a successful rendering. In addition if shadow rays were used in the rendering rather than using only the photon map for rendering the result would appear much smoother as in the following [Image 1].

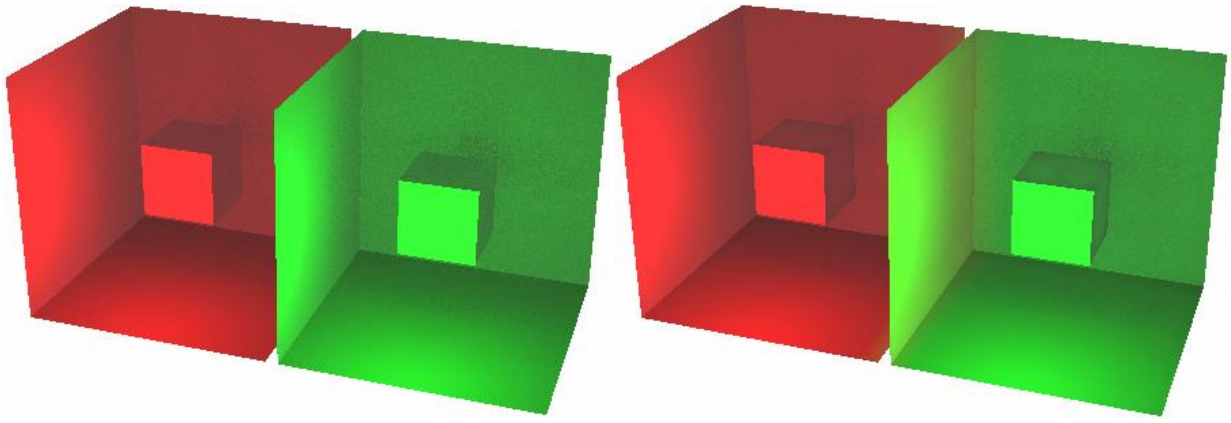

Image 1 (a and b): These two images show our method (left) and the basic photon mapping (right) with ray traced shadows. The same effects from our other results are still visible but less prominent because the photon map contributes to less of the total light. Thus, we felt it better to use direct visualization of the photon map in our results, to highlight the differences between methods.

## Future work

To further improve the method we would be interested in determining a way to reduce the noise produced without resorting to using shadow rays from ray tracing. One thing we noticed when working on the original code is that using a cube to bound the points captured resulted in a significant noise increase on its own. The reason for using a cube is the difficulty of clipping a region to the inside of a sphere however this should not be impossible. In order to clip a geometric shape to a sphere one can figure out how far the plane of the face is away from the center of the sphere and use that to overlay a bounding circle on the plane. After doing this we would just have to determine the area covered by the intersection of the circle and the plane. This might be able to eliminate the noise artifacts which may have been created when the sphere for gathering photons was changed to a cube.

Currently, the multiple kd-trees and geometric area calculation are based purely on the mesh geometry. This means that bleeding effects can still occur in the case of non-manifold geometries. For example, if in the box scene out mesh consisted of a single floor rectangle across both boxes with the center wall touching it but not sharing an edge, the light bleeding effect would still occur. It may be possible to extend the technique to detect such situations and either modify the geometry, or leave the geometry intact but make use of the additional knowledge to improve the rendering.

## Bibliography

Hay, Heinrich, and Werner Purgathofer. "Advanced radiance estimation for photon map global illumination." *Computer Graphics Forum*, 2002: 541-545.

Jensen, Henrik W. "Global illumination using photon maps." *Rendering Techniques*, 1996: 21-30.

Larsen, Bent D., and Niels J. Christensen. "Optimizing photon mapping using multiple photon maps for irradiance estimates." *Winter School of Computer Graphics.* Plzen, Czezh Republic: Union Agency-Science Press, 2003.

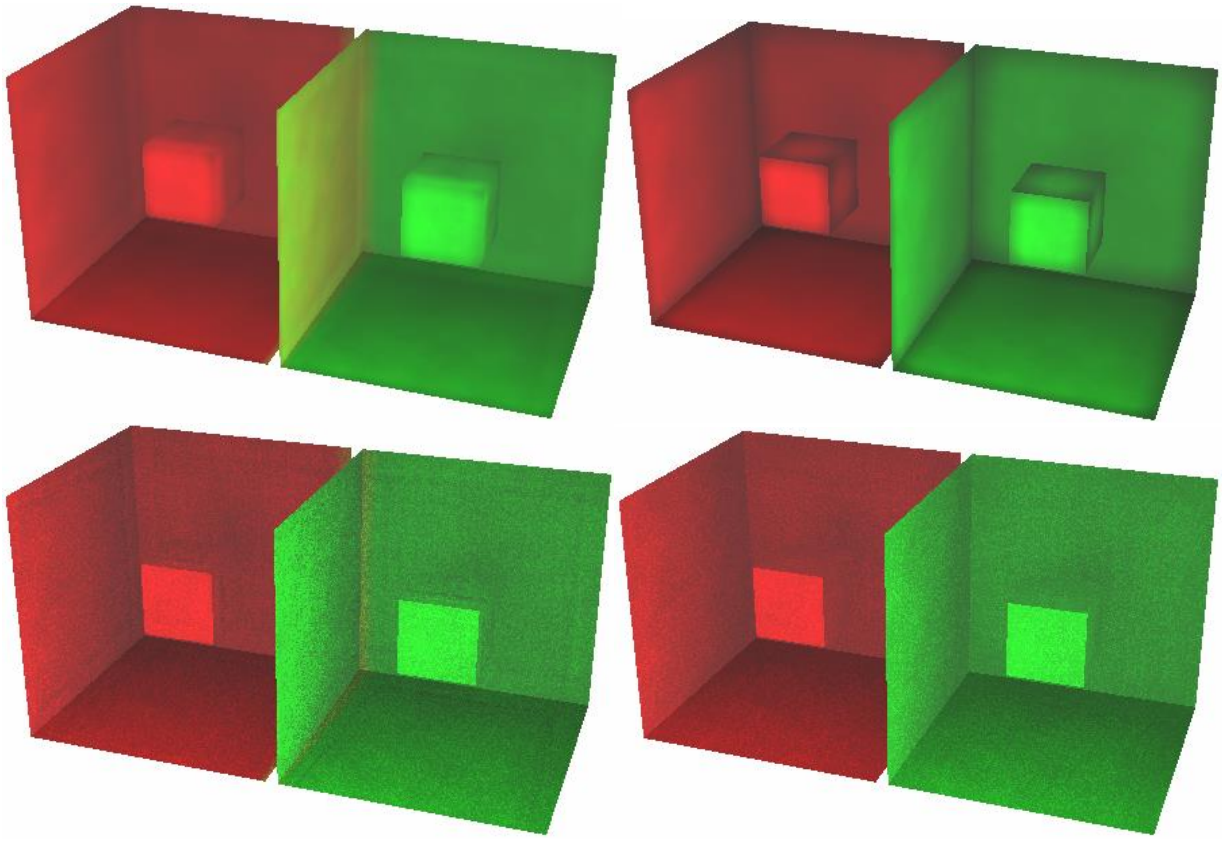

Image 2 (a, b, c, and d): The first image in the set represents lighting using only the original algorithm. The second uses only multiple photon maps, The third only geometric area calculation, and the fourth image uses both methods. The rendering is of two rooms next to each other with cubes floating inside.

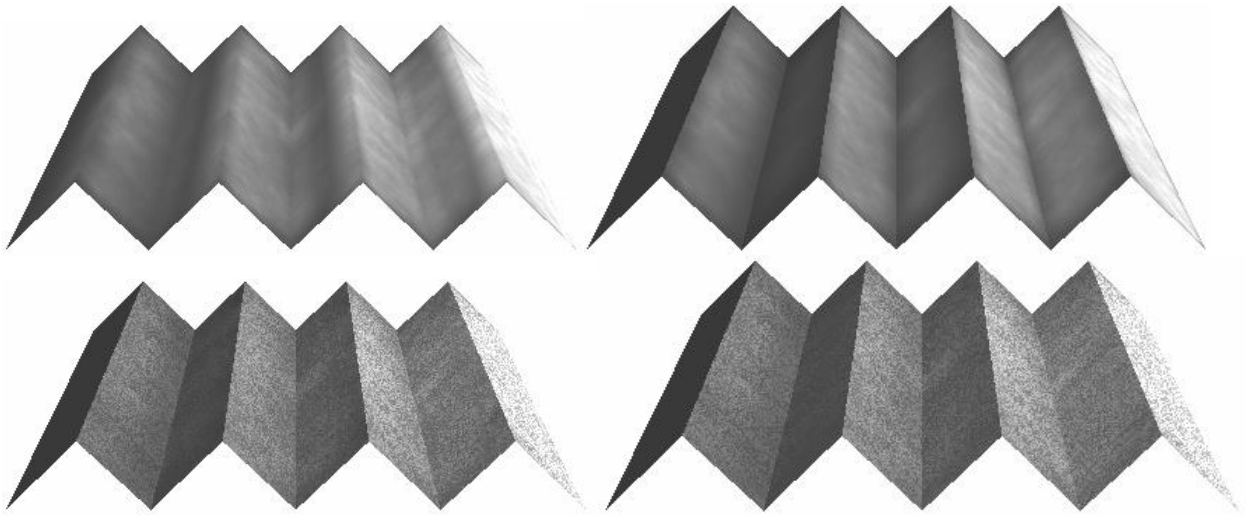

Image 3 (a, b, c, and d): The first image in the set represents lighting using only the original algorithm. The second uses only multiple photon maps, The third only geometric area calculation, and the fourth image uses both methods. The rendering is of the accordion surface and clearly demonstrates the effect of color bleeding over sharply differently lit edges.

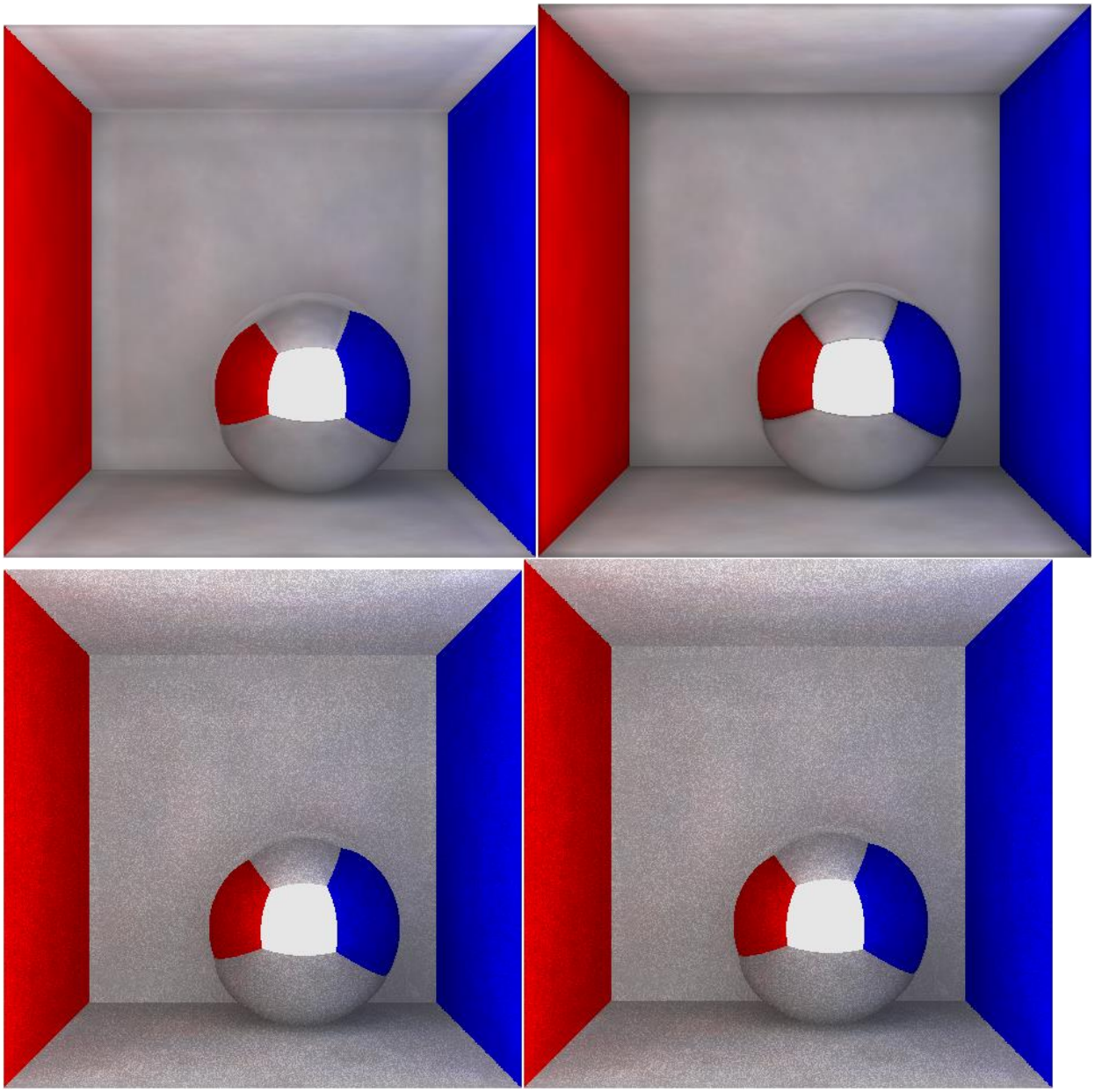

Image 4 (a, b, c, and d): The first image in the set represents lighting using only the original algorithm. The second uses only multiple photon maps, The third only geometric area calculation, and the fourth image uses both methods. This image of the reflected ball also demonstrates the sharp changes in brightness on the corner of the room however in this case either one of the methods alone s sufficient to reduce the problem however the problem is only completely eliminated in c and d.

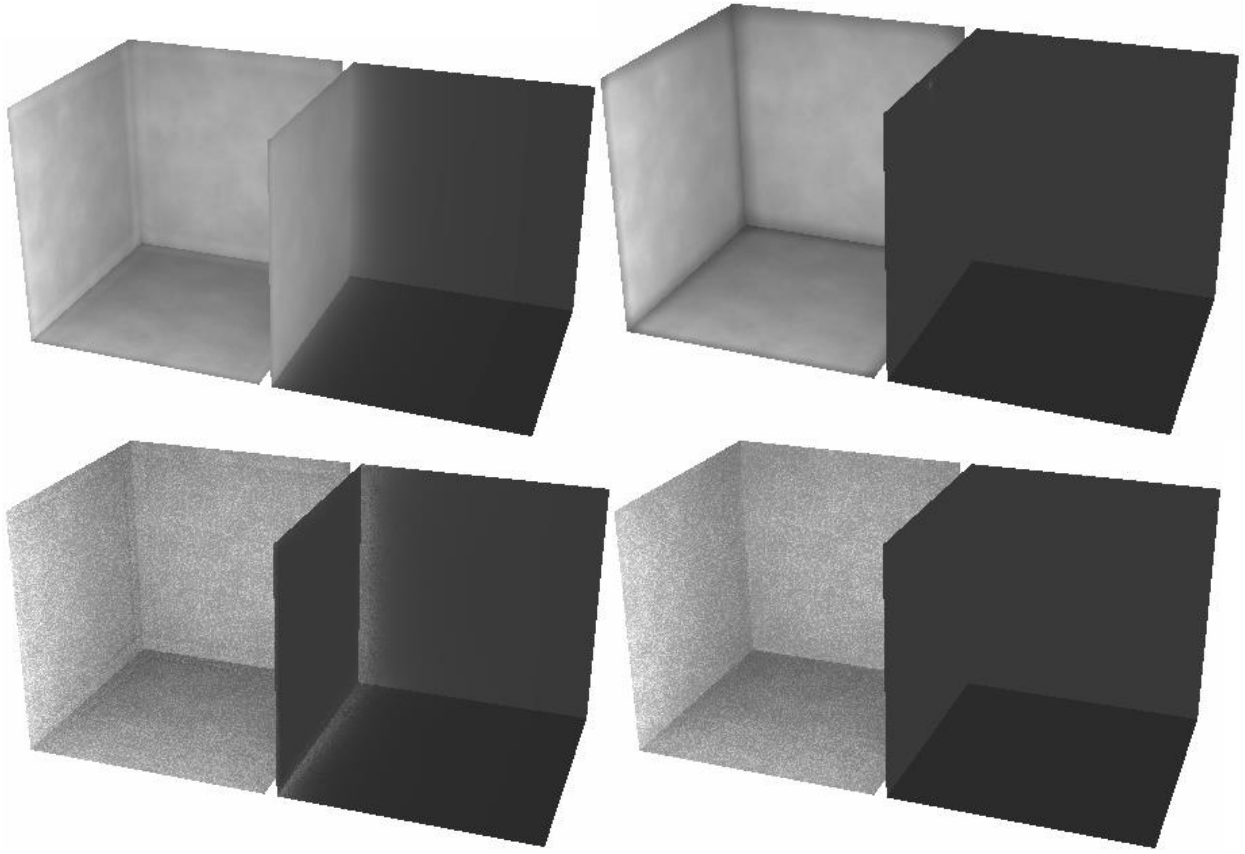

Image 5 (a, b, c, and d): The first image in the set represents lighting using only the original algorithm. The second uses only multiple photon maps, The third only geometric area calculation, and the fourth image uses both methods. The use of gray walls highlights the light bleeding through the walls and the effect of the problem of the incorrect edge lighting. Only the combination of both methods fully eliminates all the problems we were looking to explore.