A Point-based Global Illumination Rendering Pipeline for Environment Lighting

Name*
Advanced Micro Devices, Inc.

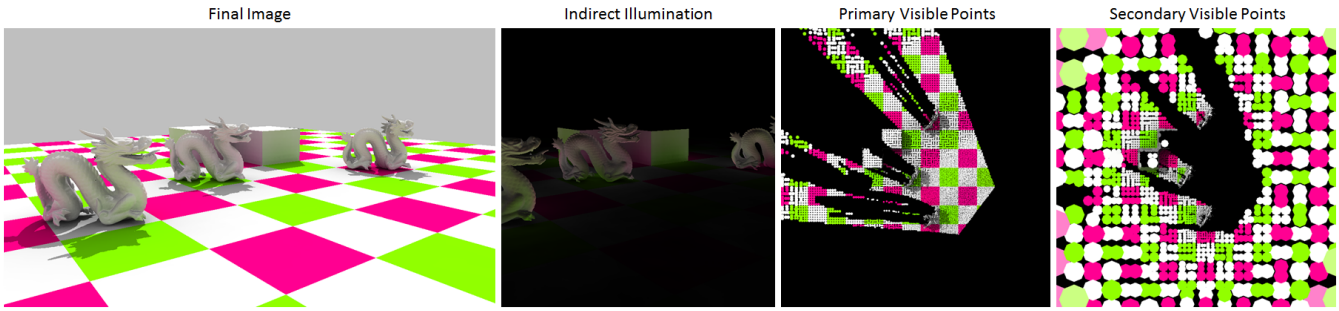


Figure 1: *Indirect illumination under environment lighting is calculated by detecting primary visible points, then secondary visible points. Direct illumination on secondary visible points are used to render indirect illumination.*

Abstract

This paper proposes a point-based rendering pipeline for indirect illumination for environment lighting. The method improves the efficiency of the algorithm used in the previous studies that calculate direct illumination on all the points in the scene which is prohibitively expensive for environment lighting because it requires a hemispherical integration for each point to compute direct illumination. The proposed rendering pipeline reduces the number of direct illumination computations by introducing approximations and careful selection of points on which indirect illumination is calculated. More specifically, the rendering pipeline first selects primary visible points on which indirect illumination is calculated from the point hierarchy. A micro-buffer is rasterized for each primary visible point to identify secondary visible points whose direct illumination affects indirect illumination of a primary visible point. Dependency of those points is analyzed and approximations are introduced to reduce the number of points on which micro-buffer rasterization is executed to calculate direct illumination. After direct illumination is obtained for those points, direct illumination on all of the primary and secondary visible points is calculated by illumination propagation without rasterizing any micro-buffer. As a result, the proposed rendering pipeline makes it possible to compute one-bounce indirect illumination for environment lighting at interactive speed.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: radiosity, global illumination, constant time

*e-mail:rsmith@gmail.com

Links: DL PDF

1 Introduction

Point-based global illumination is a technique to compute global illumination widely used in VFX [Christensen 2008]. Recent work studied ways to accelerate it using the computational power of the GPU [Ritschel et al. 2009]. Most of this research focused on a closed environment under a few analytic lights. To compute indirect illumination under such conditions, existing works compute direct illumination on all the points in the scene first; then, those points are used to calculate indirect illumination on the screen. We can use such an approach for this type of scene because direct illumination computation is computationally inexpensive and direct illumination of most of the points can affect indirect illumination of the visible surface in the scene. However, for an open outdoor scene under environment lighting, the situation is different; direct illumination computation is computationally expensive because it requires hemispherical integration, and it is likely that direct illumination on many points will not affect indirect illumination on the visible surface of the scene. Therefore, it is not practical to use the existing approach.

This paper proposes a point-based global illumination rendering pipeline for such a scene that computes at interactive speed one-bounce indirect illumination for environment lighting as shown in Fig. 1. The entire rendering pipeline is designed to reduces as much as possible hemispherical integration via micro-buffer rasterization. The method first selects primary visible points on which indirect

illumination is calculated from the point hierarchy. Secondary visible points whose direct illumination affects indirect illumination of a primary visible points are gathered by rasterizing a micro-buffer for each primary visible point. Then dependency among secondary visible points is analyzed to identify points on which direct illumination computation using hemispherical integration is inevitable. Approximations are also introduced to reduce the number of micro-buffer rasterizations.

We apply the proposed rendering pipeline to several scenes and show that the proposed method can reduce the number of micro-buffer rasterizations drastically, which is essential to render indirect illumination for environment lighting at interactive speed.

2 Related Works

Interactive global illumination computation is an active area of research. A complete review can be found in [Ritschel et al. 2012]. Recently, using a hierarchical grid were studied [Kaplanyan and Dachsbacher 2010] [Crassin et al. 2011]. One advantage of using a grid for global illumination computation is the decoupling of the scene mesh complexity from the computational cost because it does not use scene polygons directly. This is important because the number of polygons used for a scene increases as the computation power of the GPU increases. Those indirect illumination computation methods rely on the computation of direct illumination on reflective shadow maps [Dachsbacher and Stamminger 2005]. Therefore, most of the existing methods compute indirect illumination only from analytic lights but not from an environment light.

Another hierarchical data structure to represent a scene is the point hierarchy. [Bunnell 2005] started using point-based representation for global illumination effects. They showed that it is possible to compute a high-quality ambient occlusion without any precomputation except for the hierarchy generation. [Dong et al. 2007] extended the work to consider occlusion of points. They rasterized the point hierarchy to a cube map storing the closest point for each direction. However, because of restriction on the GPU at the time, part of the rendering pipeline was executed on the CPU.

[Ritschel et al. 2009] developed a method to run point-based global illumination entirely on the GPU by implementing point-hierarchy traversal and rasterization on the GPU that was performed on the CPU in the previous work [Dong et al. 2007]. They also proposed to creating a mapping function for each BRDF that makes it possible to perform an importance sampling, which maximizes the usage of a limited number of texels in a small micro-buffer. All the works discussed so far traverses the point hierarchy for detection of the visible point in a direction; [Maletz and Wang 2011] introduced a stochastic sampling of points for micro-buffer rasterization. Because this method does not require traversal of the point hierarchy, a micro-buffer rasterization can be parallelized.

All previous research on the point-based method was for a small closed scene under analytic lights such as point or directional lights. For such a scene, we can first calculate direct illumination on all the points in the scene and then use it to compute indirect illumination. We can use this method because direct illumination computation is computationally inexpensive and direct illumination on most of the points affects indirect illumination on another point; therefore, there is not much unnecessary computation. However, this is not the case for an open outdoor scene in which there can be many points whose direct illumination does not affect illumination on any point. Also, an environment light is often used as well as analytic lights for an open scene. Direct illumination computation from an environment light is not as computationally cheap as computing illumination from analytic lights because it requires hemispherical integration. Therefore, it is not practical to compute direct illumination on all the points in the scene, as most of the previous work did.

This paper proposes a solution to compute at interactive speeds the indirect illumination for an open scene under an environment light.

3 Method

3.1 Discretized One-bounce Global Illumination

The rendering equation we want to solve for each pixel on the screen is

$$L_o(x^0,\omega_o) = \int_\omega f(x^0)\left(L_i^D(x^0,\omega) + L_i^I(x^0,\omega)\right)\left(n(x^0)\cdot\omega\right)d\omega \tag{1}$$

where x^0, f, L_i^D, L_i^I are surface position at the pixel, BRDF, and direct and indirect illumination terms.

If the hemisphere is split into n non-overlapping regions, Eqn. 1 can be approximated with the following discretized equation.

$$\begin{aligned} L_o(x^0,\omega_o) \quad \approx \quad & \sum_{i\in n} f(x^0)L_i^D(x^0,\omega_i)\left(n(x^0)\cdot\omega_i\right)\Delta\omega \\ & + \sum_{i\in n} f(x)L_i^I(x^0,\omega_i)\left(n(x^0)\cdot\omega_i\right)\Delta\omega \end{aligned} \tag{2}$$

If $\Delta\omega$ is small enough, we can find point x_i^1, which is visible from x^0 in the direction of ω_i. Therefore, once the set of points $x^1 = \{x_0^1, x_1^1, \cdots, x_{n-1}^1\}$ is found, we can solve the direct illumination term in Eqn. 2 because $L_i^D(x^0,\omega_i) = L_o^D(x_i^1,\omega_i)$ can be sampled from an environment light and mask it ($L_o^D(x_i^1,\omega_i) = 0$) for the direction where x_i^1 is found.

For the indirect illumination term in Eqn. 2, we need to calculate incoming radiance from x_i^1 in the direction of ω_i. This can be obtained by solving Eqn. 2 at x_i^1. If only one-bounce indirect illumination is considered, the indirect illumination term of Eqn. 2 can be dropped. Thus, the equation we need to evaluate is

$$L_o(x_i^1,\omega_i) \approx \sum_{j\in n} f(x)L_i^D(x_j^2,\omega_j)\left(n(x_i^1)\cdot\omega_j\right)\Delta\omega. \tag{3}$$

In the same way as finding x^0, we find point x_j^2, which is visible from x_i^1 in the direction of ω_j. Once the set of points x^2 is found, we can evaluate this equation in the same way we did for the direct illumination term in Eqn. 2.

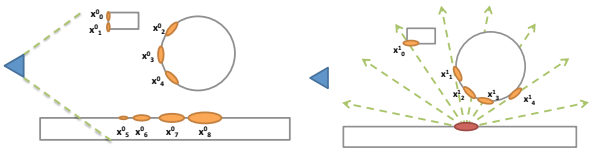
Therefore, the discretized rendering equation can be solved once we found x^1 for all the pixels and x^2 for all x^1.

3.2 Point-based Hemispherical Integration

Point-based global illumination uses the point hierarchy of a scene to evaluate the discretized hemispherical integration. It consists of two steps.

The first step is to find visible points from a surface point for all the discretized directions using micro-buffer rasterization. For point x^i, micro-buffer rasterization finds the set of points $x^{i+1} = \{x_0^{i+1}, x_1^{i+1}, \cdots, x_{n-1}^{i+1}\}$.

Once those points are found, they can be used to compute global illumination at x^i. For instance, indirect illumination at x^i can be



(a) (b)

Figure 2: *(a) Primary visible points. (b) Secondary visible points.*

calculated as follows once direct illumination on those points are calculated.

$$L_o^I(x^i, \omega_i) = \sum_{j \in n} f(x^i) L_i^D(x_j^{i+1}, \omega_j) \left(n(x^i) \cdot \omega_j \right) \Delta\omega \quad (4)$$

This process is called evaluation of a micro-buffer.

This equation is identical to the direct illumination term in the discretized rendering equation in Eqn. 2. Therefore, the point-based hemispherical integration can be used to calculate direct illumination.

To evaluate one-bounce indirect illumination, we can apply the brute-force method in which direct illumination is evaluated first for each leaf point and the values then are used to calculate direct illumination of internal points in the point hierarchy by propagating illumination from leaf points. Then a micro-buffer is rasterized for each pixel on the screen. This is the method employed by previous works to calculate indirect illumination using point-based global illumination. However, direct illumination computation is too expensive for an interactive application when hemispherical integration via micro-buffer rasterization is necessary for each point. One-bounce indirect illumination computation under environment lighting is such a case.

The proposed rendering pipeline keeps the number of micro-buffer rasterizations to a minimum by introducing following key ideas.

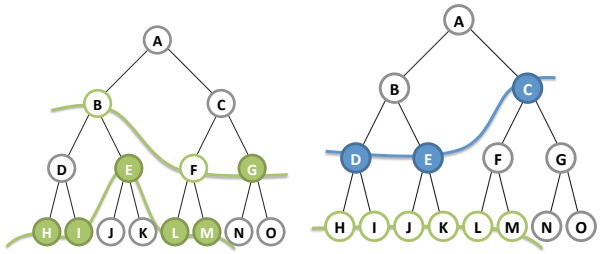
1. There is a dependency of direct illumination on x^1. Therefore, there is a subset of x^1 on which hemispherical integration is inevitable.
2. By selecting x^0, where points on which Eqn.2 is evaluated from the point hierarchy, there is an overlap between x^0 and x^1.

In the following, we call x^0 and x^1 primary and secondary visible points, respectively, because x^0 are points visible from the camera and x^1 are points visible from the camera after one bounce (Fig. 2).

3.3 Direct Illumination on Secondary Visible Points

If we assume that we have selected primary visible points x^0, we can obtain the list of secondary visible points by rasterizing a micro-buffer for each of those points. When indirect illumination is only calculated up to one-bounce effect, all we need is direct illumination on secondary visible points. Direct illumination on a secondary visible point can be calculated by selecting leaf points that have the point as an ancestor, rasterizing a micro-buffer for each of those leaf points, and averaging values by using areas of points as weights.

However, the amount of direct illumination computation is reduced by introducing an approximation. Secondary visible points are selected from the appropriate level of the point hierarchy so they occupy almost the same solid angle from a primary visible point. Therefore the contributions of illumination from secondary visible points to a primary visible point are almost the same even if the size of the points is different. This means that direct illumination calculation at the resolution of secondary visible points makes uniform the resolution of computation. Therefore, direct illumination is calculated once at the center of the point rather than on all the leaf points that are descendants of the secondary visible point.



(a) (b)

Figure 3: *The green curve is a cut of visible points from a primary visible point. The blue curve is a cut of visible points from the camera (i.e., primary visible points). Points filled with a solid colors are points on which direct illumination needs to be calculated by micro-buffer rasterization. (a) Direct illumination on F can be obtained from values on L and M; thus F does not require its own micro-buffer. B has a child D on which direct illumination is not calculated. However, the value on D can be obtained from H and I; therefore, illumination on B can be deduced from its children. (b) Direct illumination on all the secondary visible points in this figure can be calculated from values on primary visible points.*

By using the dependency among secondary visible points, we can reduce the number of micro-buffer rasterization further, while increasing the accuracy of direct illumination computation. If two children of a secondary visible point are included with the secondary visible point, direct illumination is calculated for those child points by hemispherical integration. Then, direct illumination on the secondary visible point parent to those child nodes can be calculated by the weighted average of direct illumination on the children. For example, F in Fig. 3 (a) corresponds to the parent in this case where it has children L and M both of which are included with the secondary visible point. the parent is F and children are L and M. By re-using direct illumination on children, we execute only two micro-buffer rasterizations and the weighted average of illumination values, whereas three micro-buffer rasterizations were necessary when the dependency among them was ignored.

Even if both its direct children are not included with the secondary visible point, direct illumination of the node can be calculated by values on descendants if secondary visible points form a cut of the hierarchy under the point. A secondary visible point on which hemispherical integration is not necessary can be detected by traversing the hierarchy upward from all the secondary visible points. This is illustrated in Fig. 3 (a).

3.4 Selection of Primary Visible Points

An obvious choice for primary visible points is points at which primary rays for pixels hit a scene. However, it requires an additional micro-buffer rasterization for each pixel to identify secondary visible points in the hierarchy. Instead, we select primary visible points from the point hierarchy too. This choice allows us to reduce the number of micro-buffer rasterizations. To find secondary visible points for each primary visible point, micro-buffer rasterization is unavoidable.

If a primary visible point is included with a secondary visible point, we do not have to rasterize a micro-buffer for the point to calculate direct illumination because it is already executed to detect secondary visible points. Thus, we have micro-buffer for the point already.

We also introduce an approximation in which primary visible points are small enough that direct illumination on the area around the point is uniform. More specifically, we assume that direct illumination on all of its descendants is equal to the illumination of the point. By using the assumption, direct illumination of a secondary visible point that has a primary visible point as an ancestor does not need to be calculated by micro-buffer rasterization; instead, the value is copied from the value of the ancestor. Those secondary visible points whose direct illumination can be obtained from primary visible points are detected by downward traversal of the point hierarchy as illustrated in Fig. 3 (b).

Micro-buffer rasterizations are executed for secondary visible points whose direct illumination is not computed from direct illumination on other secondary visible points and primary visible points. Then direct illumination on those points and primary visible points are propagated in the hierarchy to obtain direct illumination on all the secondary visible points.

3.5 Illumination Calculation on Frame Buffer

Once direct illumination on all the secondary visible points is calculated, indirect illumination can be evaluated at primary visible points by Eqn. 2. However, the resolution of indirect illumination calculation is different from screen pixel resolution, i.e., we need to calculate indirect illumination for those primary visible points as well as all the pixels on the screen. Therefore, calculating indirect illumination on primary visible points is not enough.

Instead, primary visible points close to the surface at a pixel are searched for in the object space. Once m nearest primary visible points are found, outgoing radiance at the pixel is calculated by taking a weighted average.

$$L_o(x,\omega) = \sum_{k\in m} w_k \sum_{j\in n} f(x) L_i^i(x_k^0, \omega_j)\,(n(x)\cdot\omega_j)\,\Delta\omega \quad (5)$$

This is essentially the same as a radiance cache in which primary visible points are used as cached location [Gautron et al. 2005].

Direct illumination has to be calculated for each pixel on the screen. However, direct illumination from environment lighting is expensive and low-frequency. Therefore, instead of calculating on a per-pixel basis, it is also calculated at each radiance cached point or primary visible point at this stage of the pipeline. In this way, direct illumination can be calculated almost for free on primary visible points because micro-buffers are already rasterized when secondary visible points are searched for.

4 Implementation Details

The proposed method is implemented on the GPU using OpenCL and OpenGL. Therefore, we are going to use OpenCL terminology for explanation of a parallel implementation.

Data structure Points are generated uniformly on the surfaces of the scene by using a low-discrepancy sequence. Then a binary bounding volume hierarchy is built on those points. Each point in the hierarchy stores position, radius, normal vector, and cone angle of the point [Rusinkiewicz and Levoy 2000]. Those data are stored without quantization (i.e., occupying 32 bytes). In addition to those, we allocate two child indices, a skip index that is used for stackless traversal, and a 32bit field that is used for selection of a subset of secondary visible points on which micro-buffer rasterization is executed. Therefore, the size of a point is 48 bytes. Point generation and hierarchy construction are done before the rendering loop starts.

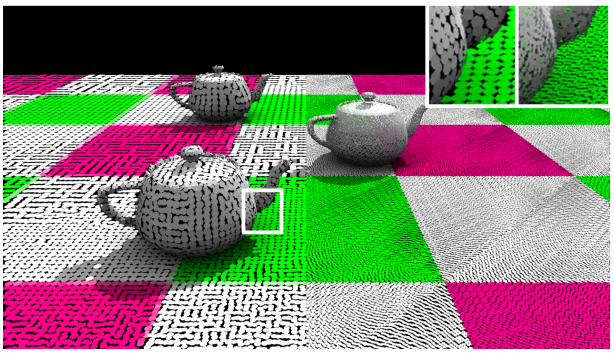


Figure 4: *Left and right halves show primary visible points and points stored at leaf level, respectively.*

Select Primary Visible Points To select primary visible points, the point hierarchy is traversed and a cut is defined at the points whose solid angle is below the threshold. Note that we do not collect all the points meeting the criteria which includes most of the leaf points; instead, we collect just a set of points belonging to the cut.

If we select points by checking only the solid angle, it can contain a point that has a large cone angle [Rusinkiewicz and Levoy 2000]. However, using such a point for a radiance cache increases the error of radiance interpolation. Thus, the cone angle of a point is also compared to the threshold angle. This results in gathering higher-resolution points at a corner of a geometry, which is similar to the strategy of placing cached points for irradiance caching [Ward et al. 1988].

For a single-threaded implementation, we can descend the point hierarchy from the root until we hit a point that satisfies the criteria. For a parallel implementation, a work-item is executed for each point of the hierarchy. It exits if the node does not meet the two criteria. It then checks its two child points. A child point satisfying the conditions is projected into screen space and the depth of the point is compared to the depth stored in the depth buffer at the screen-space position. A point passing the test is appended to the list of primary visible points. The reason we need to check two levels of the hierarchy is because we need to collect points at the cut. In other words, we need to detect the discontinuity of the conditions in the hierarchy.

The left half of Fig. 4 shows the primary visible points selected for a scene. They are different from points stored at the leaf level in the hierarchy shown on the right half of Fig. 4.

Select Secondary Visible Points Micro-buffer rasterization for primary visible points is implemented in the same way as [Ritschel et al. 2009]. A work-item is executed for a primary visible point and it rasterizes a micro-buffer storing depth and point index. Depth and point index are compressed into 8 and 24 bits and packed into a 32-bit value. Micro-buffers are stored in global memory. For all the results shown in Section. 5, the resolution of micro-buffers was set to 16×16 and 8×8 for rasterization at primary and secondary visible points, respectively. We used the paraboloid projection for pro-

jection of points [Heidrich and Seidel 1999]. Ray-casting is used to calculate accurate occlusion from leaf points.

After storing point indices in micro-buffers, the indices are used to build a list of secondary visible points.

Select Direct Illumination Computation Points To select secondary visible points on which micro-buffer rasterization is inevitable, the hierarchy is traversed upward and downward. For upward traversal, a work-item is executed for each leaf point. Each work-item ascends the tree by reading the parent pointer for a point that is computed in advance. Because we use a binary tree, an internal point is always touched twice by a work-items processing its children. To compute visibility of the point correctly, visibility on both children must be computed before the point is processed. Therefore, the first work item to visit the point exits the computation and the second work-item processes the point. This guarantees that visibility is propagated up to both children when the point is processed. The order of arrival at a point is checked by using an atomic operation on a counter prepared for each point.

Downward propagation is a parallel computation for each primary visible point because they belong to a cut of the point hierarchy that guarantees that a primary visible point does not have any primary visible point as a descendant. Therefore a work-item executed for a primary visible point descends the tree from the point.

Calculate Direct Illumination Direct illumination from environment lighting can be calculated by using a micro-buffer storing a visible node index for each texel to calculate occlusion. For secondary visible points, we do not have micro-buffers. Therefore, it must be rasterized and then evaluated. Occlusion from analytic lights such as directional light is calculated by shadow map. At this point, we have direct illumination only on a portion of the secondary visible points. Direct illumination on other points is computed by the following illumination propagation.

Propagate Illumination Propagation of direct illumination on points is performed in the same way as the selection of direct-illumination computation points. A work-item is executed for a leaf point and a primary visible point on upward and downward propagations, respectively.

Build Radiance Cache A workgroup is executed for a visible point to convert point index to radiance for each texel in a micro-buffer by looking up illumination on points. After obtaining incoming radiance from all the directions, it is convolved to build a pre-convolved radiance cache [Scherzer et al. 2012]. In this work, we created a pre-convolved radiance cache for diffuse BRDF at 5×5 resolution.

Calculate Indirect Illumination For an efficient neighboring search for radiance caches for a pixel, we build a screen-space two-dimensional grid in which a cell of the grid corresponds to a tile in the screen, as in [Harada et al. 2012]. The effective radius for each point is defined as a linear function of the radius of a point. Because we chose points with small radius on a surface close to the camera and points with large radius on surface far from the camera, a point close to the camera has a smaller cache footprint while a point far from the camera has a larger cache footprint. A bounding volume for each radiance-cached point constructed in the screen space is used to identify the overlapping cells to which a reference of the point is stored. We store multiple references per point which increases the memory usage but it simplifies the neighbor search, which is usually more expensive than building a grid.

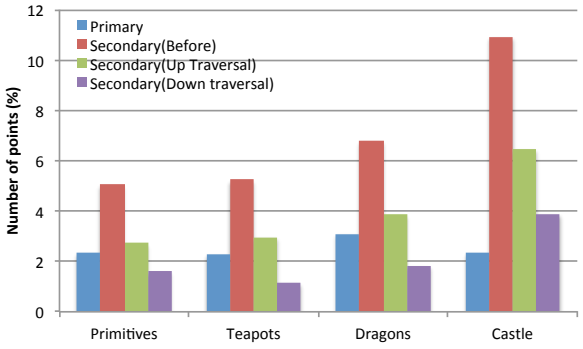


Figure 5: *Ratio of the number of points detected as primary and secondary visible points to the number of points generated for each scene.*

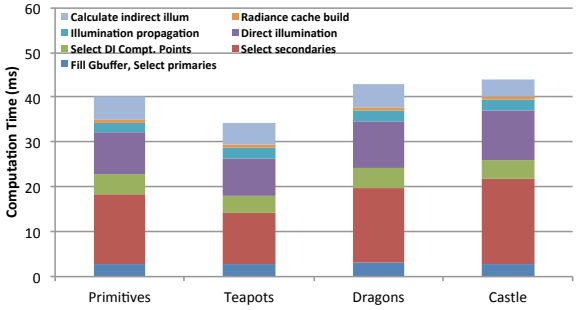


Figure 6: *Break down of computation time.*

After the 2D grid is built, we execute a work-item for a pixel in the screen. A work-item fetches the world position and normal vector of the pixel first and iterates through the list of radiance caches stored in the cell to which the pixel belongs. The weight function proposed in [Ward et al. 1988], with clamping by maximum angle is used to calculate a weight from a cache. 2D workgroups are executed in which a workgroup processes pixels in a cell of the 2D grid to increase the coherency of memory access to the radiance cache in a grid cell.

5 Results and Discussion

We test the proposed method on a system with an AMD Radeon™ R9 290X GPU and an AMD FX-8150 CPU. Test scenes are converted to 500,000 points and rendered under a uniform environment light and a directional light. Therefore, memory footprint for the point hierarchy is 48MB. The scenes are rendered at 1280×720 screen resolution and screenshots of final rendering are shown in the first column of Fig. 1 and Fig. 7. The third column of those figures shows primary visible points rendered from the top of the scenes. We can see that it captures points that are in the view frustum but not occluded by other geometries. We can also see that the size of points increases as the distance from the camera increases. The number of primary visible points for all the test scenes is less than 5% of the number of points generated for the scene (Fig. 5).

We also counted the number of secondary visible points before and after upward and downward hierarchy traversals (Fig. 5). For the scene "Primitives", the number of secondary visible points is reduced from 25.28k to 13.59k after upward traversal. After downward traversal, it is reduced further to 8.04k which is only less than

Table 1: *Test scene data*

	Primitives	Teapots	Dragons	Castle
Triangle count	3.67k	3.1k	180.91k	57.13k
Rendering time (ms)	40.00	34.4	42.97	44.09
Primary vis. points	11.72k	11.45k	15.52k	11.72k
Secondary vis. points	8.04k	5.73k	9.14k	19.32k

Figure 8: *Comparison to the brute-force implementation. The difference is magnified by four.*

2% of all the generated points. Also, we can see that the proposed rendering pipeline culled about 70% of the secondary visible points for this scene. The secondary visible points on which direct illumination are computed using micro-buffer rasterizations are visualized at the right-most column of Fig. 1 and Fig. 7. We can see that it does not include points that are in the view frustum except for those which were occluded by a geometry; points in the view frustum are culled because a primary visible point are found as an ancestor. Also, this method does not compute direct illumination on fine resolution at the location far from the camera frustum. From Fig. 5, we can see that the scene "Castle" has more secondary visible points than other scenes. This comes from the nature of the point-based method, which captures more points at a fine level than at a coarse level at a concave edge of a geometry. Because the proposed method culls secondary visible points by traversing upward, it does not reduce points at lower level of the hierarchy. In other words, there is a room for improving the point-reduction algorithm.

The total number of micro-buffer rasterization executed is the number of primary visible points plus remaining secondary visible points after hierarchy traversals; for example, it is 19.76k for the scene "Primitives". For the brute-force implementation, we need to rasterize a micro-buffer for all the points generated on the surface at least (i.e., 500,000 points in those tests). Therefore, our rendering pipeline is rasterizing less than 5% of the brute-force implementation.

Furthermore, if final gathering is performed for each pixel of the screen, we need to execute on the order of another 1 million micro-buffer rasterizations. Thus, the proposed method realizes indirect illumination under an environment light with a fraction of micro-buffer rasterizations compared to the brute-force implementation.

A breakdown of the computation time is measured and shown in Fig. 6. Rendering a frame of those test scenes took from $34ms$ to $44ms$. Computation time is almost irrelevant to the polygon count for the scene although time for selection of secondary visible points that executes micro-buffer rasterization on primary visible points varies slightly. This is because the number of primary visible points is not the same for those scenes. We can also see that the scene "Castle" takes more time for direct illumination which includes micro-buffer rasterization for secondary visible points. This is because the number of points on which direct illumination has to be evaluated for this scene is more than the other scenes (Table 1).

To evaluate the approximation introduced in this work, a rendered image for scene "Primitives" is compared to an image rendered with the brute-force implementation that calculates direct illumination on all the leaf points and is propagated to the entire hierarchy. Although there is error around the small features of teapots, the difference is negligible, as we can see in Fig. 8.

6 Conclusion

In this paper, we presented a point-based global illumination rendering pipeline for environment lighting. The proposed rendering pipeline starts with selecting primary visible points which are used to collect secondary visible points from the point hierarchy. After secondary visible points are collected, points on which direct illumination computation using hemispherical integration is inevitable are selected by traversing the point hierarchy. Approximations are introduced to reduce the number of those points.

Experiments showed that the proposed rendering pipeline reduces the number of points on which direct illumination must be calculated by micro-buffer rasterization to 10% to 30% of the secondary visible points. Illumination propagation also is proposed to calculate direct illumination on those culled points from direct illumination on points for which micro-buffer rasterization is executed. The proposed rendering pipeline builds a radiance cache on the primary visible points, which then is used to compute indirect illumination on all the pixels on the screen.

There are several avenues for future work, including the further optimizations discussed in Section 5. We would like to apply the proposed rendering pipeline to a dynamic scene by combining it with a fast hierarchy generation implementation. We have studied one-bounce indirect illumination in this paper and we would like to extend the method to support more light bounces and study how the number of direct illumination computation points grows as the number of bounces increases. Also, we have applied the proposed method only to a scene with diffuse surfaces; we would like to extend it to support glossy surfaces for which we need to build only radiance caches.

Acknowledgement

We acknowledge XX for useful discussions and comments.

References

BUNNELL, M. 2005. Dynamic ambient occlusion and indirect lighting. In *GPU Gems 2*, M. Pharr, Ed. Addison-Wesley, 223–233.

CHRISTENSEN, P. H. 2008. Point-based approximate color bleeding. *Pixar Tech. Report*.

CRASSIN, C., NEYRET, F., SAINZ, M., GREEN, S., AND EISEMANN, E. 2011. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum (Proceedings of Pacific Graphics 2011) 30*, 7.

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, New York, NY, USA, I3D '05, 203–231.

DONG, Z., KAUTZ, J., THEOBALT, C., AND SEIDEL, H.-P. 2007. Interactive global illumination using implicit visibility. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, PG '07, 77–86.

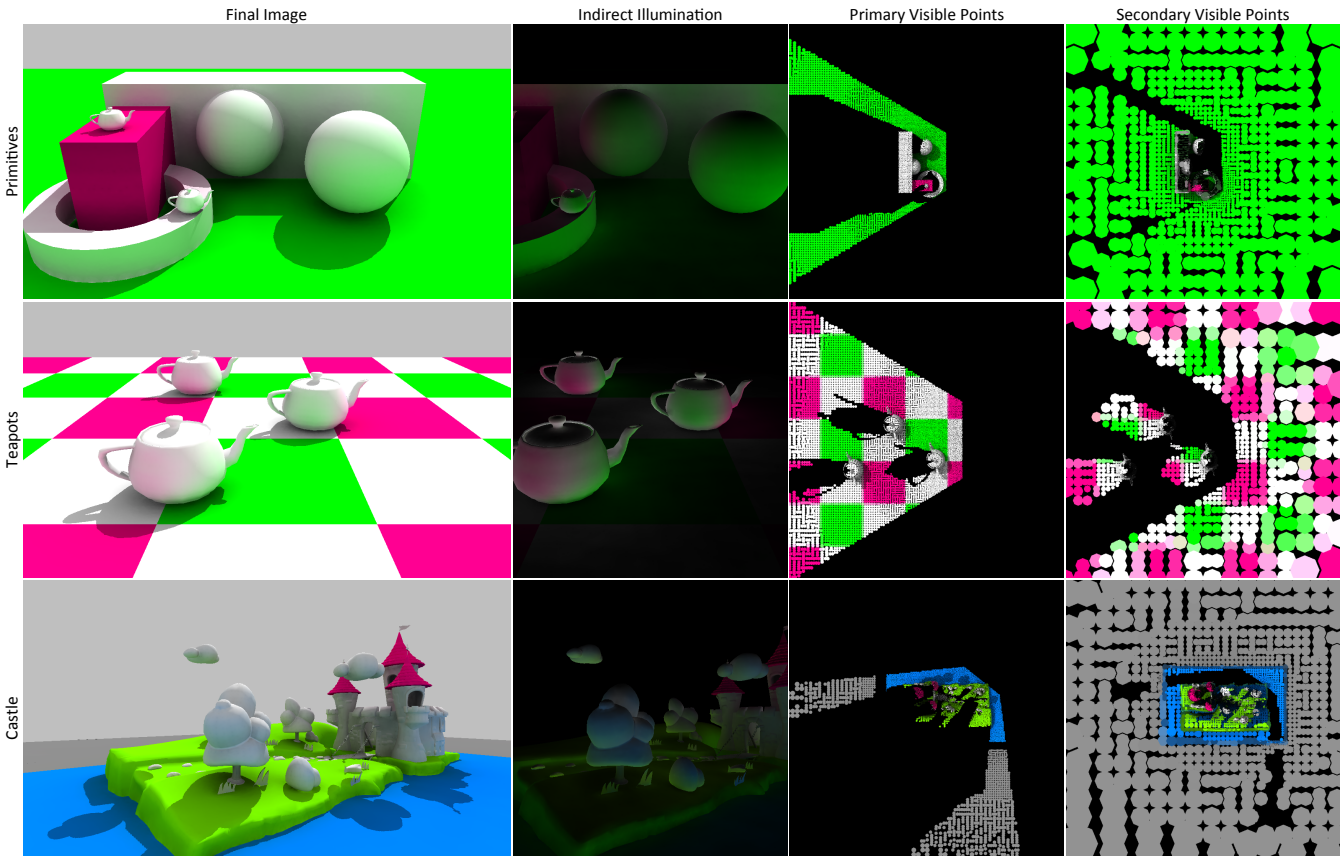


Figure 7: *Each column shows one-bounce global illumination under environment lighting, indirect illumination, visualization of primary visible points, and visualization of secondary visible points, respectively.*

GAUTRON, P., KŘIVÁNEK, J., BOUATOUCH, K., AND PATTANAIK, S. 2005. Radiance cache splatting: A gpu-friendly global illumination algorithm. In *Proceedings of Eurographics Symposium on Rendering*, 55–64.

HARADA, T., MCKEE, J., AND YANG, J. C. 2012. Forward+: Bringing deferred lighting to the next level. In *Eurographics (Short Papers)*, 5–8.

HEIDRICH, W., AND SEIDEL, H.-P. 1999. Realistic, hardware-accelerated shading and lighting. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '99, 171–178.

KAPLANYAN, A., AND DACHSBACHER, C. 2010. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D '10, 99–107.

MALETZ, D., AND WANG, R. 2011. Importance point projection for gpu-based final gathering. In *Proceedings of the Twenty-second Eurographics conference on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, EGSR'11, 1327–1336.

RITSCHEL, T., ENGELHARDT, T., GROSCH, T., SEIDEL, H.-P., KAUTZ, J., AND DACHSBACHER, C. 2009. Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2009) 28*, 5.

RITSCHEL, T., DACHSBACHER, C., GROSCH, T., AND KAUTZ, J. 2012. The state of the art in interactive global illumination. *Comput. Graph. Forum 31*, 1 (Feb.), 160–188.

RUSINKIEWICZ, S., AND LEVOY, M. 2000. Qsplat: a multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 343–352.

SCHERZER, D., NGUYEN, C. H., RITSCHEL, T., AND SEIDEL, H.-P. 2012. Pre-convolved Radiance Caching. *Computer Graphics Forum (Proc. EGSR 2012) 4*, 31.

WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. 1988. A ray tracing solution for diffuse interreflection. *SIGGRAPH Comput. Graph. 22*, 4 (June), 85–92.