Real-Time Global Illumination for Games using Topological Information

Laurent Noël, Venceslas Biri

Real-Time Global Illumination for Games using Topological Information

Laurent Noël
Université Paris-Est
LIGM
Marne-la-Vallée, France
lnoel@univ-mlv.fr

Venceslas Biri
Université Paris-Est
LIGM
Marne-la-Vallée, France
biri@univ-mlv.fr

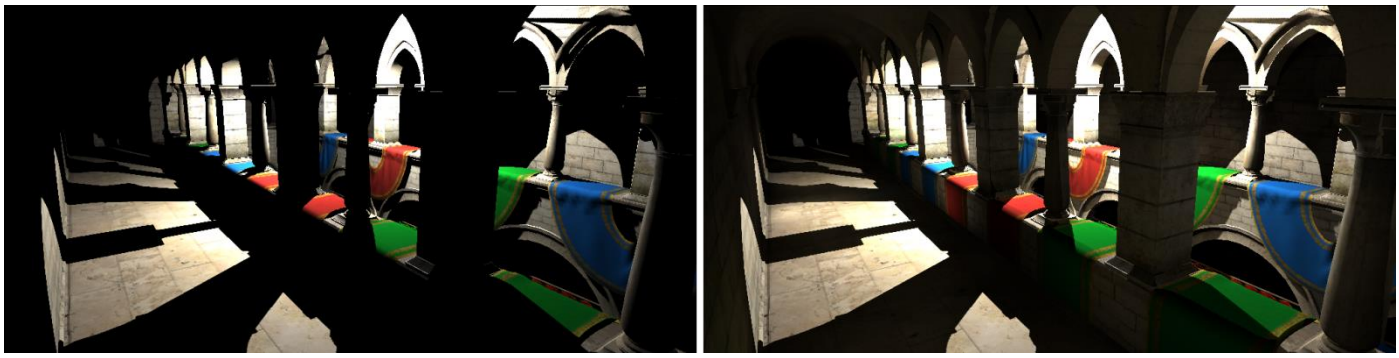

Figure 1. The left image is a rendering showing only direct illumination. The right image adds indirect illumination computed with our method. We can see new features that appear, only illuminated by indirect light like the left wall or the back of the pillars.

***Abstract*— Indirect Illumination is a key element to achieve realistic rendering. Unfortunately, since computing this effect is costly, there are few methods that render it with real-time frame rates. In this paper we present a new method based on virtual point lights and topological information about the scene to render indirect illumination in real-time.**

Keywords- real time rendering, global illumination, topology, interactive simulations

I. INTRODUCTION

Interactive 3D applications such as video games have become a lot more realistic last years due to the introduction of programmable hardware and the possibility to implement new kinds of algorithms based on global illumination techniques. Global illumination consists in the simulation of all light exchanges in a virtual scene, allowing the rendering of indirect illumination. It adds a lot of realism by simulating effects such as color bleeding or caustics. In some configurations, like indoor scenes, indirect illumination becomes the only way to light some parts of the scene (Fig. 1).

Unfortunately, such indirect, global, illumination remains a difficult challenge to compute for two reasons. First, the indirect luminance on any visible point has to be correctly evaluated. Secondly, and in order to compute correctly this luminance, visibility information between each couple of points (i.e. knowing if they are mutually visible) of the scene must be available.

Many methods have attempted to achieve real time global illumination, but recently a regain of interest has been noticed for methods based on Virtual Point Lights [24]. Such techniques send virtual rays from primary light sources. These rays intersect the virtual 3D scene and, on intersection points, create secondary lights called Virtual Points Light (VPLs). The higher the number of secondary lights, the better the global illumination is. Unfortunately, interactive evaluation of the illumination coming from a large set of VPLs remains challenging, especially when the application needs to include shadow computations (i.e. to evaluate the visibility between VPLs and points viewed by the camera).

In this paper we present a new method to render a scene illuminated by a large set of VPLs in real-time. We approximate visibility using a clustering of the VPLs obtained thanks to a segmentation of the scene in approximately convex areas. The segmentation is built using a curvilinear skeleton of the empty space of the scene that must be pre-computed. A set of disk that separate these convex areas are also computed and used to associate pixels with far away VPLs that potentially light them. The shadows are therefore rendered implicitly using this association.

Our contributions in this paper are:

- A new 3D segmentation of a 3D virtual scene based on quasi-convex area

- The definition of a new topological data structure ("*Visibility Gates*") based on the previous 3D segmentation that can handle visibility between any pair of points.
- The use of previous data structures to achieve the rendering, in real-time, of global illumination in a rendering engine.

We first present previous work related to our method, then give a global overview of our method before detailing each step. We finish by discussing our results and giving insights about future work.

II. PREVIOUS WORK

Global illumination in games is still a real challenge. Indeed, global illumination aims to compute all the light exchanges between surfaces of a 3D virtual scene. Of course, offline techniques exist able to render photorealistic image as pure classical Path-tracing using Monte Carlo [6, 7], Photon Mapping [3, 4, 2] or Metropolis Light Transport [1]. But none of these algorithms is able to produce images in real-time i.e. at 30 frames per seconds.

Therefore, strategies exist to approximate this global illumination in order to achieve real-time rendering. For example, parts of the global illumination can be pre-computed using any offline method and introduced at rendering thanks to classical light mapping or more recently using pre-computed radiance transfer [8]. For example the graphic engine of Halo [Halo3] is based on the pre-computation of the global illumination using photon mapping [3] and a decomposition of the incident light into spherical harmonics [10]. Unfortunately these techniques are limited to static scene or static lighting and are therefore not suitable for games with dynamic lights. Other methods focus on the first bounce of the light source. This lead to efficient algorithms like Reflective Shadow Maps (RSM) [11] that can produce, in real-time, all illumination with two bounces of light on surfaces. Since efficient, this technique use a lot of memory per primary light which prevent its use for massive lighting. Reflective Shadow Maps has been use in the graphic engine of Crytek, the CryEngine 3, with Light Propagation Volume.

Indeed, Crytek presented a new technique [12] based on the RSM and on light propagation volume that use the GPU to compute diffuse indirect lighting from a bunch of secondary light inside a cascade of volume. Once again, this technique is impressive but has several limitation especially with secondary shadows (shadows casted by secondary light sources) and also by the fact that the indirect light scatter very quickly. For its part, the Unreal graphic engine uses a technique called voxel cone tracing in a sparse voxel octree [13]. This technique which is difficult to implement works in real-time and offer secondary bounces of light. But it produces coarse indirect illumination especially when dealing with secondary shadows and is limited to two bounces of lights.

Finally, several techniques use the scalability of VPLs [5]. Indeed, this method allows the use of massive lighting, and is not limited in the number of bounce used to create secondary point lights. The number of VPLs usable is configurable what gives its scalability. Some methods focus on photorealistic rendering [16, 17, 18] whereas some deal with real-time rendering like [14, 15]. We focus on this last technique which uses deferred shading, a technique that store geometric information viewed from the camera, and perform a per-pixel lighting with clusters of VPLs. Secondary shadows are computed using a special algorithm called imperfect shadow maps. Compared to this approach, our algorithm doesn't use any kind of shadow maps for indirect illumination. Instead we choose to only use topological information about the scene to approximate the rendering of shadows, avoiding complex computation per VPL.

III. OVERVIEW

We start by a global overview of our method before detailing each step. Some pre-processing, depending only on the static part of the scene, are only done during initialization and allow the construction of topological data structure. These are then used during rendering.

The first part occurs during initialization and computes the topological data structure. First we voxelize the scene and we apply a thinning process [19] to compute a centered curvilinear skeleton from the voxels filling the empty space (where light travels, see Fig. 2). This thinning step is done only on the static part of the virtual scene, consisting, for example, on the walls. Therefore this step can be done once during initialization.

The skeleton is a graph that has the same topology as the empty space. Each node of this graph stores its position and the radius of the maximal ball contained in the empty space centered in this node. This radius gives us geometrical information about the scene (see Fig. 6). By using this information for all nodes, we are able to build clusters of nodes of the skeleton such that each cluster represents a portion of the scene approximately convex. We call these clusters *Visibility Clusters*. In the rendering pass we avoid visibility test between points of the same clusters by relying on the quasi-convexity property of the clusters.

Finally, based on the skeleton and the visibility clusters we compute a set of disks that separate the clusters. Those disks are later used in the rendering step to approximate the lighting that occurs between different neighbor clusters. We call them *Visibility Gates.*

The second part of our technique occurs during rendering and exposes the use of previous computed data structure to achieve real-time global illumination. Our rendering step implements a deferred shading pipeline to separate the processing of geometry and lights.

We approximate indirect illumination by a finite set of Virtual Point Lights. The segmentation of the scene into

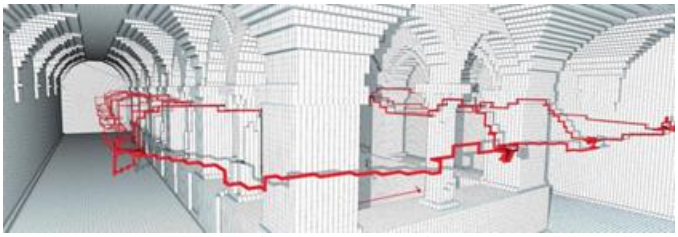

Figure 2. An illustration of the voxelization of the scene Sponza and the curvilinear skeleton of its empty space.

visibility clusters allows us to efficiently assign subsets of VPLs to pixels such that there is a high probability that the view sample (i.e. a point visible from the camera) of a pixel is visible from each VPL assigned to that pixel. To improve performances for this assignment, we use a recent method called *Clustered Shading* [20]. This technique groups view samples with similar properties (3D-positions and visibility cluster in our case). A list of VPLs is then assigned to each of these groups, based on the chosen properties. This saves us the need to do the association for each couple pixel-VPL, which is too slow.

After this assignment pass, we compute final gathering using, for each pixel, the VPLs that have been assigned to its group. To get a complete rendering, we sum our result with direct illumination. For this step we use shadow maps computed at primary light source positions.

IV. TOPOLOGICAL CONSTRUCTIONS

Our method is based on topological data built from a voxelized representation of the scene. In this section we detail the pre-processing steps to compute these data.

A. *Voxelization and Thinning*

As said previously, the scene is first voxelized. This can be done with any state-of-the-art real-time method on the GPU by exploiting the rasterization pipeline [21]. For our algorithm we only need a binary voxelization.

A curvilinear skeleton is then computed from the empty space of the voxelization (voxels that does not cover surfaces) with a thinning process. We guaranty that the skeleton is centered and has the same topology that the empty space by using an algorithm that process in the cubical complex framework [19]. The thinning process outputs a graph embedded in 3D space and a 3D grid that gives for each voxel the index of the nearest node of the skeleton (with respect to the geometry of the empty space). This grid allows us to obtain the nearest node of the skeleton for any 3D position of the scene in constant time. For each node we also get the radius of the maximal ball contained in the empty space (see the left image of Fig. 6). This radius will be used later to build visibility clusters mentioned in the overview.

B. *Clustering of the skeleton*

The next step is to compute the visibility clusters. A visibility cluster is a set of nodes from the skeleton such that the union of the maximal balls centered in these nodes is quasi-convex. We have developed an ad-hoc method to compute such clusters based on the radius of maximal balls and curvature along the graph of the skeleton. This construction is divided in two steps: a segmentation of the skeleton to keep few nodes that contains the essential geometrical and topological information; then the clustering that we apply based on the segmented skeleton.

1) *Segmentation of the skeleton*

The goal of the segmentation is to keep few nodes from the original skeleton that contains most of the geometrical and topological information of the skeleton. To segment the skeleton, we classify each node based on the topology, on the curvature and on the coverage by maximal balls. At the end, nodes that have not been classified are removed (see Fig. 6). This section details the algorithm that classifies nodes.

First, each node of degree different than two is classified as a *topology node* (Fig. 3). The degree of a node is its number of neighbors. Such nodes encode connections in the graph and removing them can alter topology of the graph, so we classify them to ensure they are kept in the segmented skeleton.

Then we regroup nodes of degree two by *lines*. A line is a sequence $L = (n_1, n_2, \dots, n_k)$ of nodes such that n_i is a neighbor of n_{i+1} and $n_1$and n_k have both a neighbor of degree different than two respectively called extrema of L and denoted by L_1 and L_k (Fig. 5). These nodes are both topology nodes that have been identified in the first step. Such a sequence can be replaced by an edge between $L_1$and L_k without changing the topology of the graph. But doing so can remove too much geometric information: some edges will pass through walls and important variations in the radius of maximal balls can be lost. We want to identify these nodes.

Let n_i be a node of a line L. We define the curvature of n_i by $curv(n_i) = \frac{dist(n_i, L_1L_k)}{maxball(n_i)}$ where $dist(n_i, L_1L_k)$ is the shortest-distance from n_i to the straight-line L_1L_k and $maxball(n_i)$ is the radius of the maximal ball centered in $n_i$contained in the empty space. Intuitively, the higher $curv(n_i)$ is, the farthest n_i is from L_1L_k compared to its distance to the scene. If $curv(n_i) > 1$, then the straight-line L_1L_k does not even intersect the maximal ball of n_i and, in that case, replacing L by an edge means replacing a curve by a straight-line. It has a high probability to produce an edge passing through a wall. For any line that contains a node with curvature greater than one, we classify this node as a *curvature node* (Fig. 4). Then we build two new lines by setting this node as an extremum and apply recursively the process until no curvature node can be found on the two new lines.

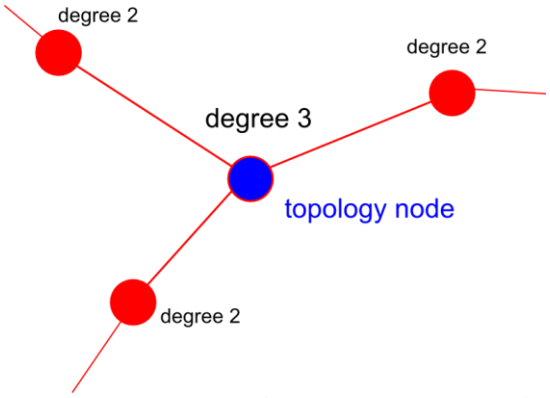

Figure 3. A topology node is a node of the skeleton with a number of neighbors that is not two.

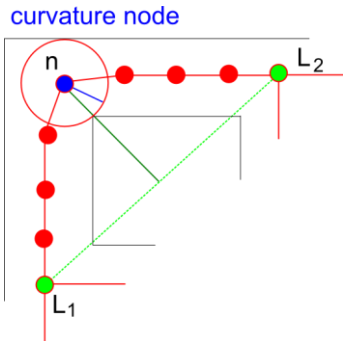

Figure 4. A curvature node n is such that its distance to the straight line L_1L_2 is shorter than the radius of its maximal ball

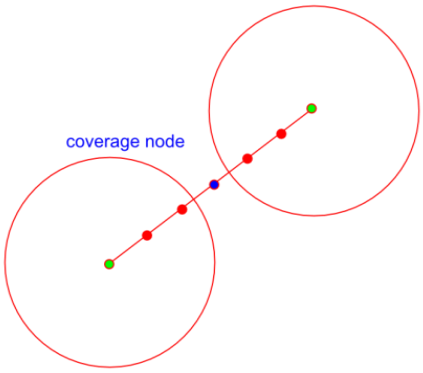

Figure 5. A coverage node is not contained in any of the maximal balls centered in extremum points of its line

At the end of the process we get a new set of lines such that each extremum is a curvature node or a topology node. At this point replacing lines with edges can still make us loose too much geometric information. Indeed, some lines are relatively straight but may contain nodes with a lot of little maximal balls. We want to keep a good coverage of edges of the segmented skeleton by maximal balls. To respect that criterion we check if the nodes of each line are contained in at least one maximal ball centered in the extrema of the line. If a node does not respect that criterion, we classify it as a *coverage node* (Fig. 5). Then we apply the same procedure than for curvature nodes to cut the line until there is no more coverage node in unclassified nodes.

Finally we get a new segmented skeleton by removing all unclassified nodes (Fig. 6). The segmented skeleton has a set of nodes that is a subset of the nodes of the original skeleton and each node of the original skeleton can be associated with a node of the segmented skeleton by taking the nearest extrema on the line containing that node.

2) Construction of Visibility Clusters

Giving the segmented skeleton, we build the visibility clusters above it using a greedy ad-hoc algorithm. We developed the algorithm such that the union of maximal balls represented by a cluster is quasi-convex.

First we sort the nodes of the segmented skeleton by radius of maximal balls (bigger first). Then for each node n in that order, if n is not already clustered, we build a new visibility cluster C and put n in a stack.

While this stack is not empty we pop a node m from it and add it to the cluster C. Then we add all neighbors of m that are not curvature nodes and for which radius of maximal balls does not differ more than α % of the maximal radius contained in the cluster. The parameter α controls how similar two maximal balls must be to cluster their corresponding nodes (we used $\alpha = 20$ for our tests). That way we cluster nodes with similar maximal balls but we avoid breaking too much the convexity by not taking curvature nodes.

We also add to the stack all nodes contained in the maximal ball of m that are accessible from it (by applying a traversal algorithm starting at m).

At the end of the process, all nodes must have been clustered. By construction, each visibility cluster is a connected component of the segmented skeleton.

C. Visibility Gates

Visibility clusters can be used to avoid doing visibility test: if a viewed surface point is in the same cluster than a VPL, it is very likely that they are mutually visible and therefore we can light it without any visibility test. But VPL of a cluster can also illuminate points from other clusters. To simulate this and avoid losing too much illumination, we developed the concept of visibility gates (Fig. 7). For each edge (n_1, n_2) of the segmented skeleton such that n_1 and n_2 are in different visibility clusters, we build a disk called *visibility gate* separating the two clusters. We place that disk at the middle of the edge (n_1, n_2), orthogonal to it, with radius $min(maxball(n_1), maxball(n_2))$. A subset of light rays coming from VPLs of one cluster pass through that disk to illuminate points of the second cluster. We will use these gates

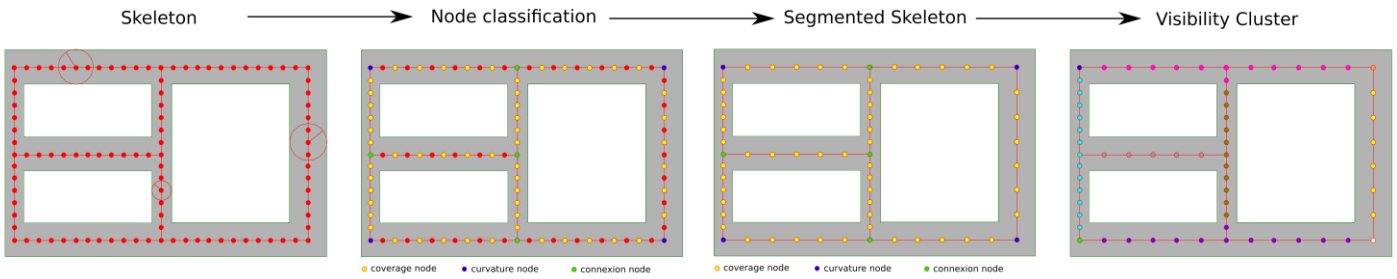

Figure 4. Sequence of steps to compute the Visibility Clusters from the detailed skeleton. The left image is the original skeleton (we also show the maximal ball centered in some nodes). Then nodes are classified using geometrical and topological information. We only keep classified nodes to get a segmented skeleton that contains most of the information of the detailed skeleton. Finally we compute Visibility Clusters from the segmented skeleton such that each cluster represents a part approximately convex of the scene.

during the rendering pass to build light cones coming from VPLs to illuminate from one cluster to the other (see Fig. 11).

V. RENDERING

The rendering step uses the pre-computed topological and geometrical information and a set of VPLs to illuminate the scene at real-time frame rate for a reasonable number of VPLs (see section VI for rendering times).

As mentioned previously, indirect illumination can be approximated by a finite set of Virtual Point Lights. The VPL concept was first introduced in the Instant Radiosity method [5]. To compute VPLs, we trace paths from the light sources and create a VPL at each intersection point (see Fig. 8). The intensity of a VPL depends on the intensity of the primary light source and all reflections that have occurred before reaching the intersection.

VPLs can be computed using monte carlo path-tracing ([5, 22, 23]) or with rasterization ([11]). This last method can achieve real-time frame rates for the generation step but limits the number of bounces that can easily be done.

To illuminate a view sample point P with n VPLs P_1, …, P_n, we apply the following equation:

$$L(P \to V) = \sum_{i=1}^{n} f_r(P_i \to P \to V) G(P_i, P) V(P_i, P) L(P_i)$$

where $L(P \to V)$ is the light reflected by P towards the view point V, $f_r(P_i \to P \to V)$ represents the bidirectional reflectance distribution function (BRDF) at the point P, $G(P_i, P)$ is the geometric term between, P_i and P, $V(P_i, P)$ is 1 if P_i is visible from P (0 if not) and $L(P_i)$ is the intensity of the VPL P_i.

In the equation, the term that is the most costly to compute is the visibility term $V(P_i, P)$. With a path-tracing approach it can be computed by tracing a shadow ray. When using a rasterization pipeline, a shadow map can be used for each VPL but it would require to render the scene multiple times.

A common approximation used to achieve real-time rendering is to simply ignore the visibility term for indirect lighting since direct lighting is generally the most important part of illumination. Where light is mostly indirect this strategy overestimates the contribution of VPLs and the result differs greatly from reality (see Fig. 12).

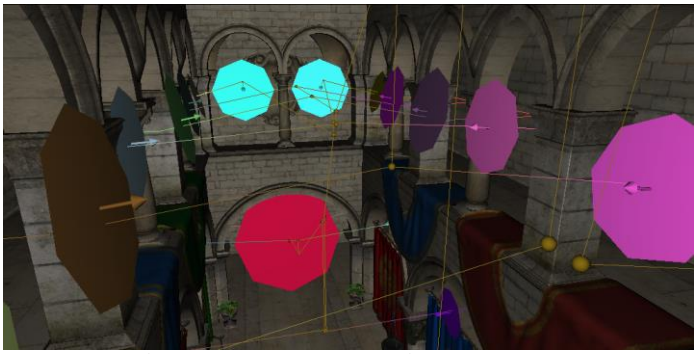

Figure 5. This figure illustrates the position of some gates in the Sponza scene

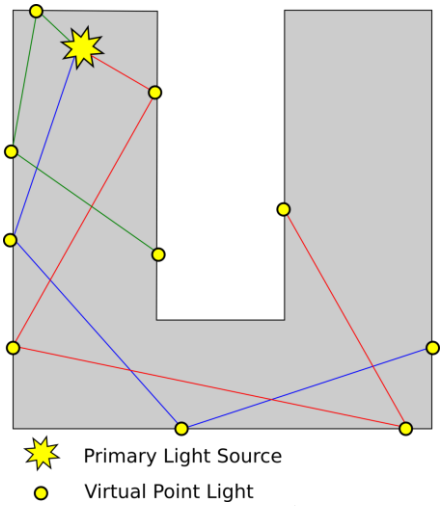

Figure 8. Here three random paths are traced from the primary light source. Each intersection with the scene is used to create a Virtual Point Light.

The goal of our method is to use the pre-computed topological information to quickly decide if a view sample can see a VPL. This approximation can be large but produces a coherent indirect illumination.

Two types of indirect illumination will be applied using the VPLs. Let P be a VPL and C its visibility cluster. All view samples in C will be directly illuminated by P. For others, we will build a set of cones by combining P with visibility gates separating C from neighbor clusters (Fig. 11). These cones will be tested against view samples contained in other clusters to decide if the VPL illuminates them.

Since doing this test for each couple view sample-VPL is too costly, it can't be done each frame without optimization. We can note that a few number of view samples can be affected by a cone in regard to the total number of them. To reduce the number of tests we implemented the recent clustered shading technique [20] that allows us to reject quickly large group of view samples that does not intersect the cone created from a VPL and a gate. To make the distinction with our visibility clusters, we will call *geometry clusters* the clusters computed by the clustered shading method.

In this section we detail each step of the rendering step. It follows a deferred shading algorithm but can also be implemented with forward rendering and early depth test.

A. *Geometry Pass (GP)*

The geometry pass of deferred rendering classically build a Geometry Buffer (GBuffer). A GBuffer is composed of multiple textures containing geometrical information for each pixel of the scene. Our GBuffer contains the following information:

- Normal in view space
- Depth, to reconstruct position

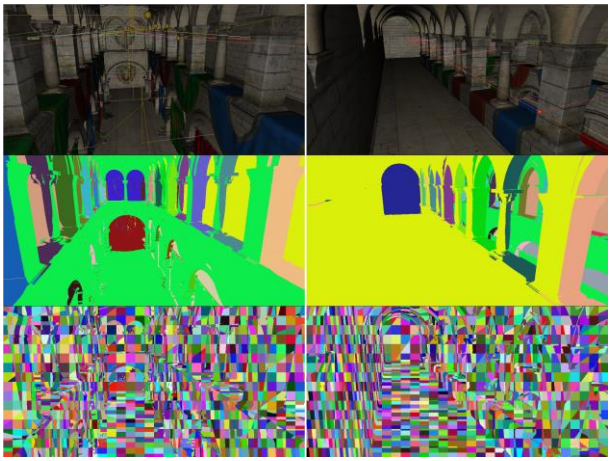

Figure 9. This figure illustrates two points of view. The second row presents the segmentation in visibility clusters. The third row shows the segmentation in geometry clusters.

- Diffuse color
- Glossy color and exponent
- Index of the visibility cluster
- Index of the geometry cluster needed for clustered shading

To be able to compute the index of visibility cluster of each pixel, we store our topological data in GPU memory. More specifically the data sent to the GPU is the segmented skeleton (position, radius of maxball and list of neighbors for each node), the 3D grid that gives for each voxel the nearest node of the segmented skeleton and an array giving for each node the index of its visibility cluster (see Fig. 9 for an illustration of the last two textures of the GBuffer).

B. Identification of unique Geometry Clusters (IUGC)

You can refer to [20] to get more detail about this step. By exploiting compute ability of modern GPU, we identify unique geometry clusters that can be seen by the camera and re-index them from 0 to the total number of them. We choose to implement the local-sort strategy for its simplicity and the possibility to compute the bounding box of each geometry cluster at the same time. We also add on top of it a new constraint: all fragments of a geometry cluster must have the same visibility cluster. This will be used in the next step to assign VPLs to geometry clusters.

C. Light assignement pass (LA)

For each geometry cluster this step identifies the VPLs that potentially affects its fragments and build a list of those VPLs. There is two cases: if the VPL is contained in the same visibility cluster that the one associated to the geometry cluster, we trivially add the VPL to the list. Else, we compute a bounding sphere for the geometry cluster (enclosing the bounding box of this one). This sphere is used to test for intersection against all cones created from the combination of the VPL and a visibility gate. If we find a cone that intersects the bounding sphere, we store both the VPL and the index of the gate for final test in the fragment shader. All this pass is again implemented using a compute shader.

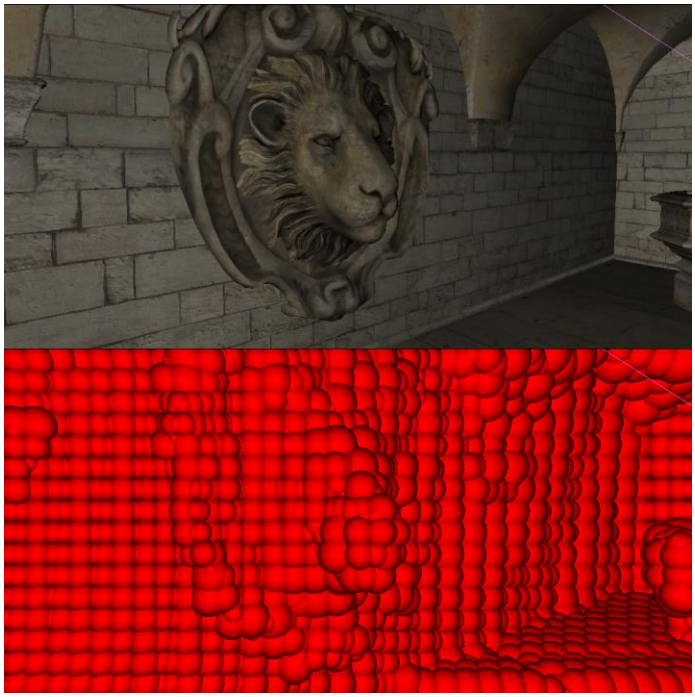

Figure 10. This image exposes the bounding spheres of each geometry cluster. These spheres are used to quickly reject group of view samples that are not affected by a cone.

D. Indirect Lighting (IL)

This step consists of a final gathering of VPL contributions using VPLs assigned to the geometry cluster of each fragment during the previous step. For each fragment we loop through the list of VPLs stored in the light assignment pass for the geometry cluster of the fragment. If a visibility gate has also been stored (which occurs when the view sample belongs to a different visibility cluster than the VPL), we test that the view

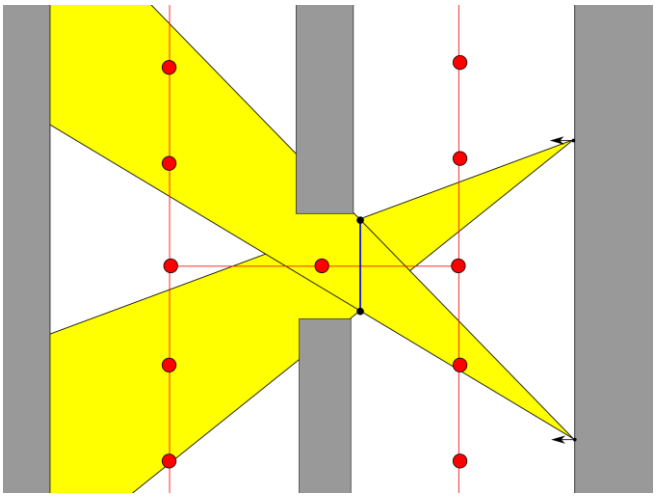

Figure 11. In this figure, two VPLs (on the right wall) are combined with a visibility gate (blue line at the center of the image) to produce a cone (in 3D). These cones are used to illuminate the neighbor visibility cluster that does not contain the two VPLs.

sample is contained in the cone resulting from the combination of the VPL and the gate.

E. *Direct Lighting*

This classical last step uses primary light sources with shadow maps to add direct light in the final rendered image.

VI. RESULTS AND DISCUSSION

We evaluated the performance of our method on the scene Sponza (Crytek version, 262,267 triangles). All measurements were performed on an NVIDIA GTX 670 Ti GPU. We also show a result computed from the scene Sibenik (75,284 triangles) to discuss the rendering of glossy reflections.

As mentioned before, the scene must be pre-computed to get topological data. In our current implementation this step is done on the CPU but we plan for getting it done on the GPU and eventually reach real-time rates for this pre-computation step. All views are rendered with a 1024x512 pixels frame-buffer.

A. *Precomputation*

The pre-computation of topological data is actually a weakness of our method because it can't be done each frame. This constraints the method to compute the indirect illumination only on the static part of the scene. Dynamic meshes can still be illuminated but doesn't produce indirect illumination. Dynamic lights can be used as long as a real-time method is implemented to generate the VPLs [11].

In our implementation all this pre-computation is done on CPU. We plan to implement our thinning algorithm on GPU, expecting strong improvement in performance due to the parallel nature of the algorithm [19].

For our tests we used a 256x256x256 voxel grid. The voxelization can be computed in real-time using recent GPU methods [21]. In the case of Sponza, the thinning of the empty space takes 109 seconds with an Intel Xeon 1,2Ghz CPU, using a non-optimized implementation.

B. *Rendering*

Fig. 1, Fig. 12, Fig. 13 and Fig. 14 show results of our method on the scene Sponza from multiple points of view and for different light configurations. All reference images are computed on the CPU using the same set of VPLs using shadow rays for visibility tests.

Figure 13. Here is a result computed with our method. We can see color bleeding in red frames. This effect can only be simulated using indirect illumination.

For each configuration we compare the result of our method with two kind of rendering:

- direct illumination only, to show how indirect illumination improves the quality of the result
- direct illumination + indirect illumination but without taking into account shadows for the VPLs.

Not taking into account shadows for indirect illumination is a well-known solution to approximate global illumination. But as it can be seen in Fig. 12, it strongly overestimates illumination compared to the reference, especially when illumination is mostly indirect.

Our method produces results closer to the reference in term of illumination. While producing some artifacts (due to the discrete association from surface points to visibility clusters,

Figure 12. We compare the result of our method with a rendering that ignore shadows for the virtual point lights. When illumination is mostly indirect, like in this case, doing such an approximation can drastically affects the final image. Our method gives a result close to the reference by using a subset of VPLs well-chosen to illuminate each view sample.

see Fig. 14 on the round pillar for example), these one are strongly attenuated by adding direct illumination.

Color bleeding is an effect resulting from the reflection of light coming from close colored textures. You can see it in Fig. 13 and in the indirect illumination snapshot of Fig. 14.

Methods based on VPLs are not good to simulate glossy reflections. Indeed, using a discrete set of points to approximate incident indirect illumination does not allow the evaluation of incident luminance in every direction (which is required to get good approximation of glossy reflections). Fig. 15 shows a view of the Sibenik scene which exposes glossy reflections. We used 4192 indirect VPLs to get that result. With that number of VPLs, our method is no longer real-time but stays interactive (3.7 frames per second in that case). A more appropriate method to simulate glossy reflections in real-time is [13] but is limited to two bounces of light.

Our results are slightly darker than the reference images. This is due to the nature of gates which only accounts for a subset of the illumination that occurs between visibility clusters. A good improvement to the method would be to use polygon instead of disks to better match the shape of separation between clusters. Unfortunately these polygons can be tricky to compute and to use efficiently during rendering. Oversizing the disks could also be a solution but can produce the opposite behavior for some configurations (i.e. produce results lighter than the references).

Table 1 gives times in milliseconds of each step of the rendering pass for different numbers of VPLs. We see that we reach real-time frame rate for a number of VPLs less than 512. We stay interactive for few thousands of VPLs.

Table 2 presents mean square errors (MSE) computed for two different configurations. The table shows a lower error for our method than for direct lighting or indirect lighting without indirect shadows. Configuration 2 is the one of Fig. 14. Since there is few indirect illumination on this configuration (compared to direct lighting), the error produced by computing direct lighting only is really closed to the one produced by our result.

TABLE I. RENDERING TIMES (MS)

VPLs	GP	IUGC	LA	IL	Total
512	6.1	4.3	12.3	10.9	33.6
1024	6.1	4.3	24.5	22.1	57
2048	6.1	4.3	49.0	44.2	103.6

TABLE II. MEAN SQUARE ERROR

Config.	Direct light	No shadows on indirect	Our method
1	111.96	69.1091	33.1041
2	39.7544	170.121	39.6128

We implemented the light assignment pass using a brute force algorithm on GPU (each geometry cluster is tested in parallel against each light). This pass can be optimized using acceleration data structures on lights.

The performance of our algorithm depends on light and view configuration: if all VPLs are in the same cluster, then rendering take much longer if the camera look at a lot of points from this cluster (due to the association). The rendering times given in the table are obtained with the view point and primary light source position of Fig. 11.

Finally, it can also be a good solution to remove lighting using visibility gates. That way we just keep indirect illumination that occur inside visibility clusters. This solution loses some long range lighting but also improve the performances of the algorithm by removing small contributions.

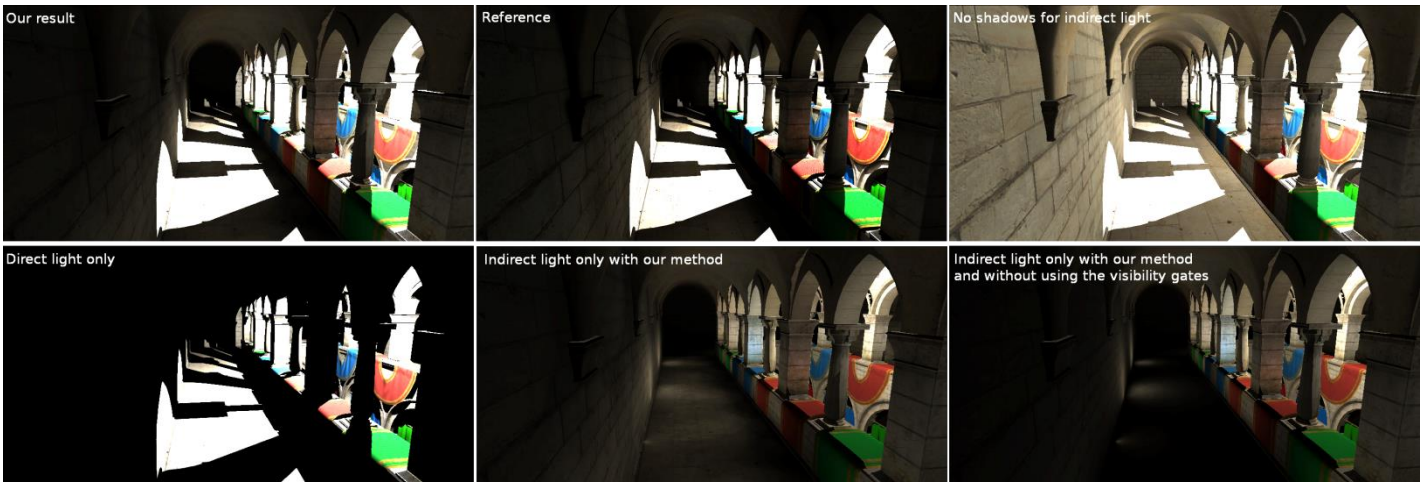

Figure 14. Here we show how visibility gates can affect the indirect illumination. Most of the light on the floor is produced by virtual point lights that are not in the same cluster. By using visibility gates, we can find for each view sample a subset of these virtual points lights that certainly illuminate it. Since gates are only disks, we still miss a lot of VPLs, producing darker results than the reference in some part of the result. Other parts are over evaluated due to shadow tests that are ignored inside visibility clusters. Still, our result is a better match of the reference image than when we totally ignore shadows for virtual point lights.

Figure 15. This view of Sibenik church demonstrates glossy reflections on the stained glass. A high number of VPLs must be used to get correct glossy reflections (here 4192 VPLs were used). There exists other methods which are more appropriated to compute that kind of effect.

VII. CONCLUSION

We presented a new method to illuminate a scene at real-time rate with a set of VPLs. By using information about the topology and geometry of the scene we presented a new 3D segmentation of the scene in clusters such that each one is quasi-convex, making us able to avoid any visibility test between points of the same cluster. To compute illumination between points of different clusters, we also developed the concept of visibility gates, built from the visibility clusters and the skeleton of the empty space of the scene. These visibility gates allow us to compute a coarse approximation of the light emitted by one VPL toward other clusters.

Our method generates artifacts at the separation between clusters due to the potential change of illumination. These artifacts would disappear if the visibility gates were perfects manifolds separating clusters. One perspective for our future work is to attenuate these artifacts using filtering strategies.

Another perspective for future research is to use our concepts of visibility cluster and visibility gate for photorealistic rendering, for example in path-tracing or photon mapping. Indeed we can quickly compute coarse repartition of light in clusters and try to guide rays toward most illuminated clusters.

REFERENCES

[1] Veach, Eric and Guibas, Leonidas J., “Metropolis light transport” SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, 1997

[2] Hachisuka, Toshiya and Ogaki, Shinji and Jensen, Henrik Wann, “Progressive photon mapping”, ACM Trans. Graph., 2008

[3] Henrik Wann Jensen, “Global Illumination using Photon Maps”, 7th Eurographics Workshop on Rendering, Technical Rendering’96, 1996

[4] Henrik Wann Jensen, “Realistic Image Synthesis Using Photon Mapping”, 2001

[5] Keller, Alexander, “Instant radiosity”, Proceedings of the 24th annual conference on Computer graphics and interactive techniques, SIGGRAPH '97, 1997

[6] T.Kajiya,” The Rendering Equation”, Computer Graphics (ACM SIGGRAPH '86 Proceedings), 1986

[7] Veach, Eric and Guibas, Leonidas J., “Optimally combining sampling techniques for Monte Carlo rendering”, Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, SIGGRAPH '95, 1995

[8] Sloan, Peter-Pike and Kautz, Jan and Snyder, John, “Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments”, ACM Trans. Graph., 2002

[9] Chen, Hao and Liu, Xinguo, “Lighting and material of Halo 3”, ACM SIGGRAPH 2008 Games, SIGGRAPH '08, 2008

[10] Ramamoorthi, Ravi and Hanrahan, Pat, “An efficient representation for irradiance environment maps”, Proceedings of the 28th annual conference on Computer graphics and interactive techniques, SIGGRAPH '01, 2001

[11] Dachsbacher, Carsten and Stamminger, Marc, “Reflective shadow maps”, Proceedings of the 2005 symposium on Interactive 3D graphics and games, I3D '05, 2005

[12] Kaplanyan, Anton and Dachsbacher, Carsten, “Cascaded light propagation volumes for real-time indirect illumination”, Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games, I3D '10, 2010

[13] Crassin, Cyril and Neyret, Fabrice and Sainz, Miguel and Green, Simon and Eisemann, Elmar, “Interactive Indirect Illumination Using Voxel Cone Tracing”, Computer Graphics Forum (Proceedings of Pacific Graphics 2011), 2011

[14] Laine, Samuli and Saransaari, Hannu and Kontkanen, Janne and Lehtinen, Jaakko and Aila, Timo, “Incremental instant radiosity for real-time indirect illumination”, Proceedings of the 18th Eurographics conference on Rendering Techniques, EGSR'07, 2007

[15] Ritschel, T. and Grosch, T. and Kim, M. H. and Seidel, H.-P. and Dachsbacher, C. and Kautz, J., “Imperfect shadow maps for efficient computation of indirect illumination”, ACM Trans. Graph., 2008

[16] Walter, Bruce and Fernandez, Sebastian and Arbree, Adam and Bala, Kavita and Donikian, Michael and Greenberg, Donald P., “Lightcuts: a scalable approach to illumination”, ACM Trans. Graph., 2005

[17] Ou, Jiawei and Pellacini, Fabio, “LightSlice: matrix slice sampling for the many-lights problem”, Proceedings of the 2011 SIGGRAPH Asia Conference, SA '11, 2011

[18] Georgiev, Iliyan and Křivánek, Jaroslav and Popov, Stefan and Slusallek, Philipp, “Importance Caching for Complex Illumination”, Comp. Graph. Forum, 2012

[19] John Chaussard, Laurent Noël, Venceslas Biri, Michel Couprie, “A 3D Curvilinear Skeletonization Algorithm with Application to Path Tracing”, DGCI 2013

[20] Olsson, Ola and Billeter, Markus and Assarsson, Ulf, “Tiled and Clustered Forward Shading”, SIGGRAPH '12: ACM SIGGRAPH 2012 Talks, 2012

[21] Elmar Eisemann, Xavier Décoret, “Fast Scene Voxelization and Applications”, ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, 2006

[22] Benjamin Segovia and Jean-Claude Iehl and Bernard Péroche, “Metropolis Instant Radiosity”, Computer Graphics Forum, 2007

[23] Benjamin Segovia, Jean-Claude Iehl, Richard Mitanchey, and Bernard Péroche, “Bidirectional Instant Radiosity”, Proceedings of the 17th Eurographics Workshop on Rendering, 2006

[24] Carsten Dachsbacher, Jaroslav Křivánek, Miloš Hašan, Adam Arbree, Bruce Walter, and Jan Novák, “Scalable Realistic Rendering with Many-Light Methods”, EUROGRAPHICS State of the Art Reports, 2013