A Low Complexity Discrete Radiosity Method

Pierre Yves Chatelier, Rémy Malgouyres

A Low Complexity Discrete Radiosity Method

Pierre Y. Chatelier [a] Rémy Malgouyres [b]

[a] *chatelier@llaic3.u-clermont1.fr*

[b] *remy.malgouyres@llaic3.u-clermont1.fr*

LLAIC, Clermont-Ferrand, France

Abstract

Rather than using Monte Carlo sampling techniques or patch projections to compute radiosity, it is possible to use a discretization of a scene into voxels and perform some discrete geometry calculus to quickly compute visibility information. In such a framework , the radiosity method may be as precise as a patch-based radiosity using hemicube computation for form-factors, but it lowers the overall theoretical complexity to an $O(N \log N) + O(N)$, where the $O(N)$ is largely dominant in practice. Hence, the apparent complexity is linear for time and space, with respect to the number of voxels in the scene. This method does not require the storage of pre-computed form factors, since they are computed on the fly in an efficient way. The algorithm which is described does not use 3D discrete line traversal and is not similar to simple ray-tracing. In the present form, the voxel-based radiosity equation assumes the ideal diffuse case and uses solid angles similarly to the hemicube.

Key words: Radiosity; voxels; discrete geometry; linear complexity; visibility

1 Introduction

In computer graphics, radiosity is known to globally provide smoother results than simple ray-tracing, since it aims at diffusing light in a better way. The main drawback is the cost involved by the new physical properties to manage, that may still be simplified to be reasonably expensive. Mixing radiosity and ray-tracing usually gives excellent results, with Monte Carlo or stochastic approaches [11][6]. Apart from sampling methods, radiosity can also be handled in a more systematic manner by computing relationships between the scene elements [7]. However, it has usually been done on a discretization of the surfaces into polygonal patches [4] . A discretization into voxels (elementary

volume elements) coupled with discrete geometry can bring new possibilities of fast visibility computation, especially for complex scenes where the patches would form a very complex mesh. The goal of this paper is to detail a previous paper about the subject [3] that brought some improvements to a previous voxel-based discrete radiosity method introduced in [8]. We present a new representation of the visibility problem, and a new data flow in computing, which lead to a quasi-linear time and space complexity for radiosity solving. The new algorithm consists in handling the visibility problem globally for each direction in space. The space is partitioned into lists of voxels, where "neighbors in the list" means "neighbors in terms of visibility". Classical ray traversal becomes useless and no time is spent in empty spaces.
Then, the radiosity can be propagated between the voxels, using the computed visibility information. By iterating within a set of directions, the radiosity in the scene converges toward a solution of a discretized radiosity equation. A tool similar to the hemicube [4] is used, factorizing some computations and using the notion of solid angle.
This algorithm is designed to work on a given set of voxels. The discretization step, and the visualization with radiosity, do not belong to the algorithm, and can be derived from [8] for instance.
The first part of this paper presents our discrete radiosity equation, a second part introduces useful notions of discrete geometry, and a third part shows the radiosity algorithm that can be used with such tools.

2 The voxel-based radiosity equation

Radiosity is defined as the total power of light leaving a point. In practice, this notion is useful provided that some assumptions are made about the properties of the surfaces. In the continuous form of the radiosity equation, the computed radiosity $B(x)$ is given for each point x. For a discretization into patches, $B(x)$ is considered homogeneous along the patches, and for a discretization into voxels, it is simply considered homogeneous on the surface contained inside each voxel. The voxel-based radiosity equation initially found in [8] is written as follows:

$$B(x) = E(x) + \rho_d(x) \sum_{\overrightarrow{\sigma} \in D} B(V(x, \overrightarrow{\sigma})) \frac{\cos\theta(x, V(x, \overrightarrow{\sigma}))}{\pi} \hat{A}(\overrightarrow{\sigma}) \qquad (1)$$

An intuitive explanation of Equation (1) is: "the total power of light leaving a voxel x (the $B(x)$ term), is defined by two terms: the proper emittance of this voxel as a light source (the $E(x)$ term), and some re-emission of the light it receives from its environment (the sum)".

- $B(\cdot)$ is present in both sides of the equality, reflecting interdependence between a point and its environment. It does not consider any outgoing direction, so it supposes that each point emits light uniformly in every direction.
- A point re-emits only a fraction $\rho_d(x)$ of the light that it receives. Assuming that this factor does not depend on incoming or outgoing direction is known as the *ideal diffuse hypothesis*.
- D is a set of discrete directions in space.
- $V(x, \overrightarrow{\sigma})$ is a visibility function, returning the first point y as seen from x in the direction of $\overrightarrow{\sigma}$.
- To quantify how much of an object is seen from a point, the term $\hat{A}(\overrightarrow{\sigma})$ is the fraction of a solid angle associated to the direction $\overrightarrow{\sigma}$. We call it a *direction factor*.
- $\cos\theta(x, V(x, \overrightarrow{\sigma}))$ expresses that incident light is more effective when it comes perpendicularly to the surface.
- The π factor is a normalization term deriving from radiance considerations.

In Equation (1), the expensive information to compute is the function $V(x, \overrightarrow{\sigma})$, (the first voxel encountered from x in the direction of $\overrightarrow{\sigma}$). In [8], it is precomputed, and thus is very similar to the *form factors* of the classical approach. The term $\hat{A}(\overrightarrow{\sigma})$ can also be easily precomputed. Note that $V(x, \overrightarrow{\sigma})$ is not well defined, since the discrete 3D line going through x and $\overrightarrow{\sigma}$ is sometimes defined up to a translation (see Fig. 1). But this is not a problem since each possible ray leads to an acceptable solution.

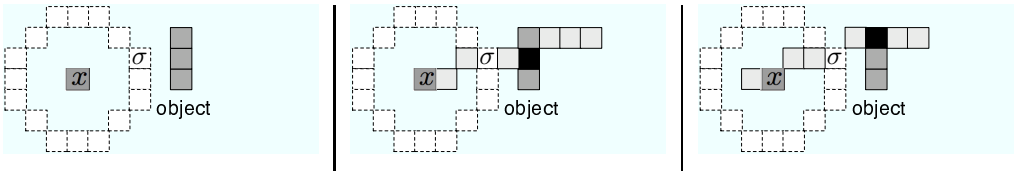


Figure 1. Given x and σ, several rays may be used to find a $V(x, \sigma)$. One of them can be arbitrarily chosen.

2.1 Solving the equation

To solve Equation (1), a converging iterative method similar to the one used with classical patch-based radiosity can be used: it usually relies on Gauss-Seidel relaxation [4]. If we consider the equation under its form $B = E + M.B$, where B is a vector of elements and M a matrix of factors, some properties of M ensure the sequence $B_{n+1} = E + M.B_n$ to converge toward a limit, which is a solution of the discrete equation. Roughly speaking, this is a transcription of light gathering, each iteration going a step further in light re-emission. The convergence is expected since light is progressively absorbed. Technically, each iteration consists in propagating packets of energy between mutually visible voxels.

3 Discrete geometry

3.1 Discrete lines

A 2D discrete line [9] whose directing vector is (a, b) can be represented by the set of points: $\{(x, y) \in \mathbb{Z}^2 / \mu \leq ay - bx < \mu + \omega\}$, where ω denotes the *arithmetical thickness* of the line, and μ sets the position of the line. The higher ω, the thicker the line. If ω is too small, the set of points becomes disconnected (see Fig. 4). ω is related to the *connectivity* of the line. If $\omega = \max(|a|, |b|)$, the line is 8-connected and it is called the *naïve* case. If $\omega = |a| + |b|$, the line is 4-connected and it is called the *standard* case.

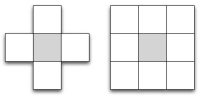


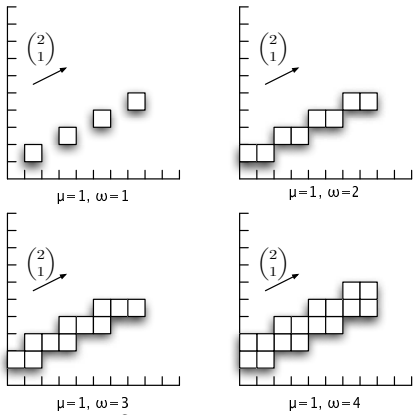
Figure 2. {4-8}-connectivity

Figure 3. {6-18-26}-connectivity

Given a direction vector $(a, b, c) \in \mathbb{Z}^3$ with $a \geq b \geq c$, a notion of 3D line has been defined [5], as the set of points $(x, y, z) \in \mathbb{Z}^3$ so that

$$\begin{cases} \mu \leq cx - az < \mu + \omega \\ \mu' \leq bx - ay < \mu' + \omega' \end{cases}$$

Other cases can be deduced by symmetry. It is noteworthy that with that definition, a 3D discrete line represents the intersection between two discrete *planes*, each one being the orthogonal extrusion of a 2D discrete line included in one of the coordinate planes. Alternatively, one can say that a 3D discrete line projects onto two 2D discrete lines that are sufficient to retrieve the 3D line (see Fig. 5). The connectivity is related to ω and ω'. If the two 2D projections are naïve (resp. standard), the 3D line is naïve (resp. standard) itself.



$(x, y) \in \mathbb{Z}^2 / \mu \leq ax - by < \mu + \omega$

Figure 4. Different 2D-thicknesses, with $(a, b) = (2, 1)$

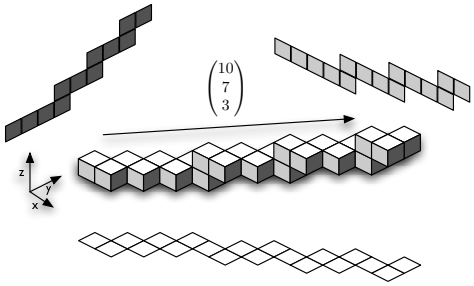


Figure 5. A 3D discrete line directed by (10, 7, 3) and its 2D projections

3.2 Partitioning the space into lines

In this section we set and prove that the space can be partitioned into parallel 3D discrete lines, along a given direction. This means that a voxel belongs to one and only one of these lines, which can be explicitly computed.

proposition: Let us denote by $\mathbb{Z}^3_*$ the set $\mathbb{Z}^3 \backslash \{(0,0,0)\}$. Given an integer vector $\overrightarrow{v} \in \mathbb{Z}^3_*$, a voxel space can be partitioned into a set of naïve, or a set of standard, 3D discrete lines, whose direction vector is $\overrightarrow{v}$.

proof: Let $\overrightarrow{v} = [a, b, c]$, with $(a, b, c) \in \mathbb{Z}^3_*$, (a, b, c) having no common divisor other than 1. We assume without loss of generality that $a \geq b \geq c \geq 0$.

A 3D discrete line with $\overrightarrow{v}$ as directing vector is defined by two 2D projections. The connectivity of the 3D line and of its projections are related, so that we can study separately the naïve case and the standard case. Let us denote the arithmetical thicknesses by:

$$\text{naïve case: } \begin{cases} \omega_{ab} = \max(|a|, |b|) \neq 0 \\ \omega_{ac} = \max(|a|, |c|) \neq 0 \\ \omega_{bc} = \max(|b|, |c|) \end{cases} \qquad \text{standard case: } \begin{cases} \omega_{ab} = |a| + |b| \neq 0 \\ \omega_{ac} = |a| + |c| \neq 0 \\ \omega_{bc} = |b| + |c| \end{cases}$$

Since $a \geq b \geq c \geq 0$, the relevant 2D projections are in the planes (Oxy) and (Oxz). This is why only ω_{bc} may be null. Thus, the 3D discrete line is equivalent to the set of all (x, y, z) such that:

$$\begin{cases} \mu_{ab} \leq -bx + ay < \mu_{ab} + \omega_{ab} \\ \mu_{ac} \leq -cx + az < \mu_{ac} + \omega_{ac} \end{cases}$$

Since $a, b, c, \omega_{ab}, \omega_{ac}$ are fixed, only the position of the line may be chosen and we denote such a set of voxels by $L(\mu_{ab}, \mu_{ac})$. Let us introduce the set of 3D discrete lines denoted by $\{L_{i,j}\}_{(i,j)\in\mathbb{Z}^2} = \{L(i*\omega_{ab}, j*\omega_{ac})\}_{(i,j)\in\mathbb{Z}^2}$. Given i and j, an $L_{i,j}$ 3D discrete line is defined by:

$$\begin{cases} i*\omega_{ab} \leq -bx + ay < i*\omega_{ab} + \omega_{ab} \\ j*\omega_{ac} \leq -cx + az < j*\omega_{ac} + \omega_{ac} \end{cases} \quad \text{or} \quad \begin{cases} i*\omega_{ab} \leq -bx + ay < (i+1)*\omega_{ab} \\ j*\omega_{ac} \leq -cx + az < (j+1)*\omega_{ac} \end{cases}$$

Given a voxel $(x, y, z) \in \mathbb{Z}^3$, this voxel belongs to $L_{k,l}$ with $k = \lfloor \frac{-bx+ay}{\omega_{ab}} \rfloor$ and $l = \lfloor \frac{-cx+az}{\omega_{ac}} \rfloor$ (where $\lfloor x \rfloor$ denotes the "floor" function). Moreover, the $L_{i,j}$'s are pairwise disjoint and thus they constitute a partition. Indeed, let us consider a voxel $v = (x, y, z)$ belonging to $L_{i,j}$ and to $L_{i',j'}$, with $(i, j) \neq (i', j')$. We assume for instance $i < i'$, because if $i = i'$, the following reasoning can still

be held, replacing i by j and say that $j < j'$.

$$
\begin{aligned}
v \in L_{i,j} \text{ and } v \in L_{i',j'} &\Rightarrow \begin{cases} i * \omega_{ab} \leq -bx + ay < (i+1) * \omega_{ab} \\ i' * \omega_{ab} \leq -bx + ay < (i'+1) * \omega_{ab} \end{cases} \\
&\Rightarrow \begin{cases} -bx + ay < (i+1) * \omega_{ab} \leq i' * \omega ab \\ i' * \omega_{ab} \leq -bx + ay \end{cases} \\
&\Rightarrow -bx + ay < -bx + ay \qquad \text{which is impossible}
\end{aligned}
$$

As a conclusion, given a direction, and a naïve or standard connectivity, the set of corresponding $L_{i,j}$'s is a partition of the voxel space into 3D discrete lines following this direction. Then, a simple operation using the $floor$ function allows to deduce, from the coordinates of a voxel, the $L_{i,j}$ line it belongs to.

4 Voxel-based radiosity

4.1 A new approach of discrete radiosity

The main problem of the approach presented in [8] lies in the required information about visibility, stored as a precomputed set of $V(x, \overrightarrow{\sigma})$. Such information is very expensive to store for each voxel of the scene, and usually does not fit in RAM. A secondary memory is necessary. A scene with 2×10^6 voxels would basically generate about 80 GB of data. With the help of discrete geometry, a new approach can be introduced, that deduces visibility on-the-fly. No pre-computations are needed, and no information has to be stored on disk. Both time and space complexity are improved by this method. This can be done by transforming the visibility problem.

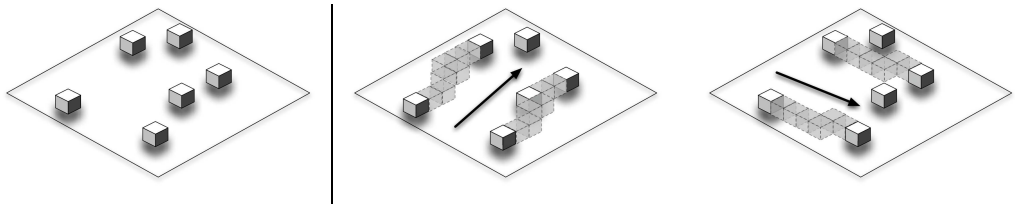
4.2 Transforming the visibility problem by considering lists

When querying $y = V(x, \overrightarrow{\sigma})$, we find out a relationship between the two voxels x and y. Then, when querying $z = V(y, \overrightarrow{\sigma})$, we do not only find out a relationship between y and z, but we also emphasize that x and z, even if they can't see each other because of y, lie on the same ray for the direction $\overrightarrow{\sigma}$. This could be represented by a list containing x, y and z in this order. Querying $V(x, \overrightarrow{\sigma})$ can be done using discrete ray-tracing [1] [10] [12].

Considering that, the visibility problem can be reversed. Instead of discovering the lists (representing rays) when querying visibility, it is rather possible to

deduce the visibility from such lists if they already exist. A list $L_{\overrightarrow{\sigma}}$ containing x, y and z encodes the fact that $V(x, \overrightarrow{\sigma}) = y$ and $V(y, \overrightarrow{\sigma}) = z$.

So, instead of iterating on each voxel x, and querying $V(x, \overrightarrow{\sigma})$, we can rather iterate on $\overrightarrow{\sigma}$ and compute a whole bunch of $V(x, \overrightarrow{\sigma})$ at a time for a fixed $\overrightarrow{\sigma}$. This second approach is interesting, since we show in the next two sections that the cost of building and storing these sorted lists can be lower than discrete ray-tracing.



a set of voxels relationships in the same set under two directions

Figure 6. Visibility solving by finding voxels on the same 3D discrete lines

4.3 A simple approach to build the lists

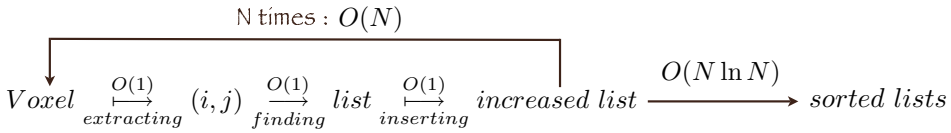
A ray of light is represented by a sorted list of the voxels that would be intersected if not considering occlusion. Therefore, for a given direction, each voxel belongs to one and only one ray, the set of all rays constituting a partition of the space. Then, using the mathematical properties presented in Section 3.2 about discrete space partitioning, we can handle such structures rather easily.

In the partition, each ray is characterized by a couple of integers (i, j). First, since the voxels are considered available in memory, it is easy to iterate over them, whatever the order, and compute their (i, j) and put them in the matching ray. Since i and j are bounded by the geometry of the scene, the most efficient structure to use is a 2D array of lists, indexed by i and j. It is not very efficient in term of space, since sparse scene would generate a sparse array, but one can find a compromise by considering hash maps. To simplify, we assume our structure to be a 2D array of lists, indexed by i and j. The lists are incrementally filled when iterating the voxels and computing their (i, j); here again it is technically tunable, by considering a mean length for the lists and pre-allocating some memory to make it work faster.

That first step only builds unsorted lists, which hold voxels which are in relation, but need sorting to reflect visibility: if the x, y and z voxels used above have been encountered in the y, z, x order during that traversal, the list should sort them anyway to x, y and z. This can easily be done by considering a sorting criterion relative to a scalar product between the direction and the position of the voxel.

Finally, to save space, there is only one big 2D array of lists at a time in memory, and it is reset for each direction $\overrightarrow{\sigma}$.

The overall complexity of the dispatching process, for a given direction, for one voxel, is a constant time. For N voxels, the time complexity of this dispatching process is obviously $O(N)$. Then adding sorting for the built lists results in an $O(N \log N)$.



4.4 An efficient approach to build the lists

Sorting the lists is not required if they can be filled in the right order. To ensure that property, an appropriate traversing order must be found for the data structure supplying the voxels to the list-filling algorithm (see Figure 7).

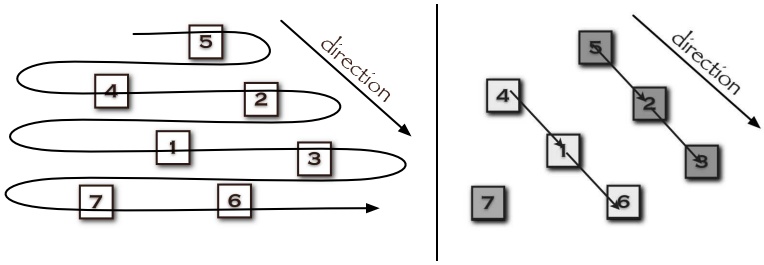


Figure 7. Given a direction and a set of voxel, traverse them in a right order allows to build the lists incrementally without extra sort

A kind of wavefront, perpendicular to the current direction, and evolving in that direction, encounters the voxels of the scene in their natural order for the given direction. In the discrete case, the wavefront can be aligned along a coordinate axis. It is worth noting that the voxels in a 3D discrete line are ordered with respect to a lexicographic order on their coordinates x, y and z, depending on the directing vector. Hence, the wavefront itself can be implemented by a lexicographic order (see Fig. 8).

So far, no conditions were required on the data structure supplying the voxels to the radiosity algorithm. But we now need the ability to be given the voxels with respect to one of the 48 possible lexicographic orders. In practice, without assumptions on the original data structure, it is possible to use 48 additional arrays, one for each lexicographic order, containing sorted pointers to the N voxels. This is affordable as a precomputation, in $O(N \log N)$ time, and in practice only needs a few seconds. Its practical time cost is totally negligible aside the radiosity solving step, since it is done once for all and is not systematically repeated for each $\overrightarrow{\sigma}$.

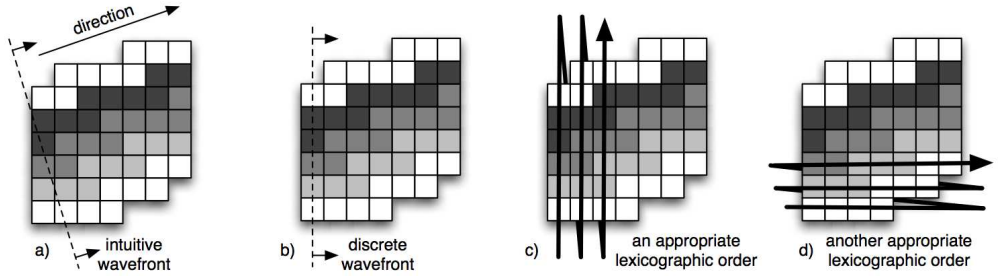


Figure 8. An appropriate traversing order of voxels respects their natural order in the discrete lines. a) A wavefront encounters the voxels in the good order. b) In the discrete case, the wavefront may be oriented along a coordinate axis. c) and d) This wavefront can be represented by a lexicographic order on the coordinates.

4.5 Reducing the number of lexicographic orders

An obvious method to reduce the number of lexicographic orders to use is to take in account the symmetry of opposite directions. We can keep only a half of the directions, the other half being the opposites, and instead of 48 sorted arrays, we can use only 24 of them, with the ability to parse them forwards and backwards.

A second method to reduce this number is to consider that the respective order of the x, y and z coordinates does not matter. As it can be seen in 2D on Figure 8, considering x before y or y before x ends by building the same lists. However, in 3D, this is not true when encountering "bubbles" (see Figure 9). Those bubbles may appear in the standard case because we use only two out of three projections to characterize the 3D discrete lines. This has been done on purpose as a basic simplification of our algorithm, because we consider that bubbles can safely be ignored. Not only the case is rare, but also adjacent voxels usually do not exchange light in the radiosity kernel. Therefore, neglecting the bubbles won't perturb the final results.

Finally, assuming that the respective order of x, y and z does not matter, the number of different lexicographic orders to consider drops down from 48 to 8, and it can still be cut to 4 thanks to the symmetry property. In this context, for a direction $\overrightarrow{v} = (a, b, c)$, the lexicographic order must only fulfill the following conditions: *if a (resp. b, resp. c) ≥ 0, then x (resp. y, resp. z) is considered in ascending order, otherwise in descending order.*

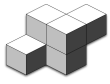
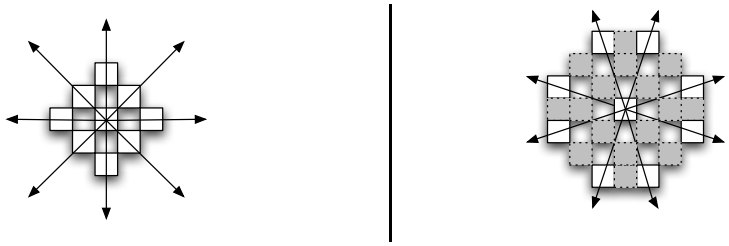


Figure 9. A bubble

4.6 Computing the set of discrete directions

Our algorithm relies on a set of discrete directions, allowing fast voxel ordering. Computing such a set is easy. We can consider a discrete sphere, each voxel of its surface giving a direction with respect to its center. However, the order in which these directions will be considered may be tuned a little. Taking the directions randomly may be a good idea since it will certainly be very "isotropic" and ensure a fast convergence of the algorithm. Another idea is to consider successively different discrete sphere, with increasing radius. This methods guarantees that even if the algorithm is stopped before it has completed, it has been very "isotropic" so far (see Figure 10).

The only difficulty is to prevent a direction from being used twice, and to assign to each of them a tuned direction factor (see Section 2): it should be proportional to the importance of the direction in the *biggest* sphere. This is done easily, using appropriate data structures to store the computed directions on each sphere. A more extensive background on discrete spheres can be found in [2].



small radius for the surrounding sphere greater radii give new directions

Figure 10. The set of discrete directions can be computed by increasing the radius of a discrete sphere. In this case, the directions are computed in an order such that each step further is a refinment in light spreading, and can be stopped without generating visible artifact due to privileged directions.

4.7 The final algorithm

We have shown in the previous sections how to handle the visibility problem with a low complexity. To solve the voxel-based radiosity equation, a converging iterative method is used, as described in [8]. The final algorithm is described on the next page.

4.7.1 Time complexity

The algorithm requires two parameters, which are the radius R of the discrete sphere used to compute the direction factors, and a number I of iterations to

```
Prepare the needed four lexicographic orders;
Compute a set of discrete directions;
for each direction do
    compute and store the associated direction factor;
//the number of iterations is a small constant
for iterations = 1 to MaxIterations do
    //the number of directions is a big constant
    for each direction do
        //dispatching step: O(N)
        Select the appropriate lexicographic order;
        for each voxel (with respect to the lexicographic order) do
            //The voxels are naturally sorted at insertion
            Add it to the back of the list (3D discrete line) it belongs to;
        //propagation (solving) step: O(N)
        for each list do
            propagate radiosity between contiguous voxels by updating B_{n+1};
        reset the lists;
```

Algorithm 1: The final radiosity algorithm

converge to a radiosity solution (usually less than 6). The number of directions is an $O(R^2)$ (usually a few thousands). For N voxels in the scene, the time complexity is:

$$\underbrace{4 \times O(N \log N)}_{\text{preparing lexicographic orders}} + \underbrace{I \times O(R^2) \times (O(N) + O(N))}_{\text{radiosity solving}} = \underbrace{O(N \log N)}_{\text{negligible in practice}} + O(I \times R^2 \times N)$$

4.7.2 Space complexity

We assume that the N voxels modeling the 3D scene are already encoded in an $O(N)$ data structure. Given a direction, the set of lists we use for partitioning contains exactly one reference to each voxel, so that $O(N)$ is expected for the total space complexity. Four precomputed lexicographic arrays of voxels are needed, this is an $O(N)$. We also need an array to store the lists. Since the lists form a partition of the 3D space, the number of lists needed for a scene is related to the square of the width of the 3D scene. Only the surface of objects are discretized, so that the width of the scene is usually an $O(\sqrt{N})$. Thus, the number of lists is at most an $O(N)$, since in the worst case, where each list contains a single voxel, there are exactly N lists. The lists are reset for each direction, so that the hidden constant depends solely on the geometry of the scene, not on the number of directions. Thus, the hidden constant remains small, and the space complexity needed by our algorithm is an $O(N)$ which merges with the $O(N)$ already needed by the data structure used for modeling.

5 Experimental results

5.1 Improvements over previous voxel-based method

The scene of Fig. 11, presented in [8], is made of 310,000 patches, and has been discretized into about 2×10^6 voxels. It has been computed with the new algorithm in the same conditions as with the previous one: 6 iterations, about 15,000 directions, on an Athlon 900 MHz with 1.5 GB of RAM. It has shown a 60% time improvement, for identical result quality (27 hours instead of 72). A main advantage is also that no hard disk space is needed since the whole computation can be done in RAM.

5.2 Storing some results on disk to optimize

Each iteration uses the same set of directions and leads to the same computations to retrieve the visibility of the voxels along the rays of a given direction. This is not a real problem because there are very few iterations. However, if disk space is available, the lists computed during the *first* iteration can be dumped and recalled in the *following* iterations. Tests have shown that it results in a 20% time regression for the first iteration, and a 30% time improvement for the following ones. On the one hand, more resources help optimizing, on the other hand, it does not deeply outperform the RAM-only approach, especially if we consider the huge required amount of hard disk: basically, it is about 60 GB for 2×10^6 voxels and 15,000 directions.

6 Parallelization of the method

There are at least two easy levels of parallelization of the presented algorithm.

- first, the propagation step, after the lists have been built, acts iteratively on each list, and the lists do not share data, since it is a partition of voxels. So, there may be several threads updating the lists without conflicts;
- second, our discrete radiosity equation is linear, so that it is possible to divide the sum into several sums. $\sum_{D} = \sum_{D_1} + \sum_{D_2} + \ldots + \sum_{D_n}$ where $\bigcup_{D_i} = D$. Thus, several processors may be used to divide the set of directions. However, it requires the voxel data to be duplicated on each node, and the results must be regularly gathered and updated. But the first tests about this parallelization, that can be seen on Figure 12, show an effective speed improvement: the cost of communications is not a problem until 4 cluster nodes.

These two parallelizations are quite easy, compared to the problem of dispatching the data on a cluster of machines. Indeed, the later involves complex

communication to be efficient. The overall parallelization is the subject of another related work.

Figure 11. This scene is only lit by radiosity

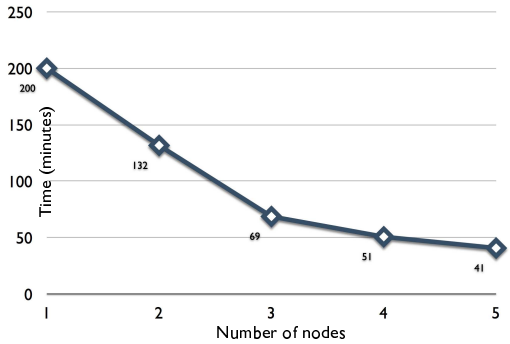


Figure 12. Parallelization results on a given test scene (multi-thread disabled)

7 Conclusion and perspectives

A voxel-based radiosity algorithm has been presented, with a quasi-linear complexity in time and a linear complexity in space, with respect to the number of voxels encoding the scene. Many experiments remain to be done in order to improve this approach of radiosity. First, instead of limiting ourselves to the ideal diffuse case, we are investigating the efficient introduction of Bidirectional Reflectance Distribution Functions (BRDF), to enhance the physical properties taken in account. Then, we are also working on the effective parallelization of the method, sharing work and data on a cluster of machines.

Acknowledgments:

Thanks to Rita Zrour for providing us the first results about parallelization.

References

[1] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, 1987.

[2] Eric Andres and Marie-Andrée Jacob. The discrete analytical hyperspheres. *IEEE Transactions on Visualization and Computer Graphics*, 3(1):75–86, 1997.

[3] Pierre Y. Chatelier and Rémy Malgouyres. A low complexity discrete radiosity method. In Eric Andres, Guillaume Damiand, and Pascal Lienhardt, editors, *DGCI*, volume 3429 of *Lecture Notes in Computer Science*, pages 392–403. Springer, 2005.

[4] Michael F. Cohen and Donald P. Greenberg. The hemi-cube: a radiosity solution for complex environments. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 31–40. ACM Press, 1985.

[5] Isabelle Debled-Rennesson. *Étude et reconnaissance des droites et plans discrets*. PhD thesis, Université Louis Pasteur, Strasbourg, 1995.

[6] Philip Dutre, Kavita Bala, and Philippe Bekaert. *Advanced Global Illumination*. A. K. Peters, Ltd., Natick, MA, USA, 2002.

[7] Paul S. Heckbert. Finite Element Methods for Radiosity. In *ACM SIGGRAPH '93 Course Notes - Global Illumination*, pages 1–7. 1993.

[8] Rémy Malgouyres. A discrete radiosity method. In Achille Braquelaire, Jacques-Olivier Lachaud, and Anne Vialard, editors, *Discrete Geometry for Computer Imagery, 10th International Conference, DGCI 2002, Bordeaux, France*, pages 428–438. Springer, April 2002.

[9] Jean-Pierre Reveillès. *Géometrie discrète, calcul en nombres entiers et algorithmique*. PhD thesis, Université Louis Pasteur, Strasbourg, 1991.

[10] Nilo Stolte and René Caubet. Discrete ray-tracing of huge voxel spaces. *Comput. Graph. Forum*, 14(3):383–394, 1995.

[11] Ingo Wald, Timothy J. Purcell, Joerg Schmittler, Carsten Benthin, and Philipp Slusallek. Realtime Ray Tracing and its use for Interactive Global Illumination. In *Eurographics State of the Art Reports*, 2003.

[12] R. Yagel, D. Cohen, and A. Kaufman. Discrete ray tracing. *IEEE Computer Graphics & Applications*, 12(9):19–28, 1992.