A Low Complexity Discrete Radiosity Method

Pierre Y. Chatelier and Rémy Malgouyres

LLAIC, Clermont-Ferrand
{chatelier, remy.malgouyres}@llaic3.u-clermont1.fr

Abstract. Radiosity in 3D scenes is usually computed using a discretization of the surfaces into patches. A discretization into voxels is possible, coupled with the use of discrete geometry. An algorithm for radiosity solving with voxels is introduced, lowering the theoretical complexity to an $O(N \log N) + O(N)$, where the $O(N)$ is largely dominant in practice, so that the apparent complexity is linear for time and space, with respect to the number of voxels in the scene. The method also fits in RAM and does not need disk storage. Instead of 3D discrete line traversal, a new algorithm is described to perform visibility computation. The voxel-based radiosity equation assumes the ideal diffuse case and uses solid angles similarly to the hemicube.

Keywords: Radiosity, voxels, discrete geometry, linear complexity, visibility, ideal diffuse case.

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. But compared to simple ray-*casting*, the complexity remains easily controllable. Mixing radiosity and ray-tracing usually gives the best results.

Radiosity has been largely studied, but has also been limited to a discretization of the surfaces into polygonal patches. 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 do not suit very well. The goal of this paper is to bring improvements to a previous voxel-based discrete radiosity method introduced in [5]. 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 in 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 emtpy spaces.

E. Andres et al. (Eds.): DGCI 2005, LNCS 3429, pp. 392–403, 2005.

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* [2] 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 [5] for instance.

2 The 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. We introduce the equation at first, then we explain how it can be interpreted. This equation comes from simplifications of a more general case, which uses the notion of *radiance* [4]. Its general continuous form is the following:

$$B(x) = E(x) + \rho_d(x) \int_{y \in scene} B(y) \frac{\cos\theta_x \cos\theta_y}{\pi \parallel x - y \parallel^2} V(x,y) dy \tag{1}$$

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

- $B(x)$ and $B(y)$ makes $B(\cdot)$ present in both sides of the equality. It reflects the interdependence between a point and its environment. It also supposes that each point emits light uniformly in every direction.
- $V(x,y)$ is a visibility function, equal to 1 if x and y are mutually visible, and 0 otherwise. This function makes it possible to write an integral over the whole scene.
- A point re-emits only a fraction $\rho_d(x)$ of the light that it receives. Assuming that this factor depends neither on the outgoing direction, nor on the direction of the incoming light, is known as the *ideal diffuse hypothesis*.
- To quantify how much of an object is seen from a point, the term $\frac{\cos\theta_y dy}{\|x-y\|^2}$, is a direct transcription of the definition of a solid angle. We denote by θ_y the angle between the direction (xy) and the normal vector at point y.
- Incident light is more or less important depending on whether it is received from right ahead or tangently. The more tangent a ray is, the less participation it has in the re-emitted light. This is represented by $\cos\theta_x$, where θ_x is the angle between the direction (xy) and the normal vector at point x. Additionally, $\cos\theta_x$ is floored to 0 to prevent light coming from *below* to be re-emitted.
- The π factor is a normalization term deriving from radiance considerations.

3 A Discrete Radiosity Equation

3.1 Discretizing the Equation

This section is a reminder of how the radiosity equation has been discretized in [5]. First, let us denote by $S_R(x)$ a virtual sphere of radius R centered on x, by σ a point on this sphere, by $y(x,\sigma)$ the first real surface point met from x in the half-line $[x;\sigma)$. The visibility problem is, like in the hemicube [2], translated to a solid angle computation on a virtual surrounding shape: a sphere.

$$\begin{aligned}
&\int\limits_{\text{y } \in \text{ scene}} B(y)\frac{\cos\theta(x,y)\cos\theta(y,x)}{\pi \parallel x-y \parallel^2}V(x,y)dy \\
&= \int\limits_{\sigma\in S_R(x)} B(y(x,\sigma))\frac{\cos\theta(x,y(x,\sigma))}{\pi}\underbrace{\left(\cos\theta(y(x,\sigma),x)\frac{d(y(x,\sigma))}{\parallel x-y(x,\sigma)\parallel^2}\right)}_{\text{solid angle as seen from } x} \\
&= \int\limits_{\sigma\in S_R(x)} B(y(x,\sigma))\frac{\cos\theta(x,y(x,\sigma))}{\pi}d\sigma
\end{aligned}$$

Now, denoting by x a voxel, by $\Sigma_R(x)$ a discrete sphere of radius R centered on x, by σ a voxel of $\Sigma_R(x)$, by $V(x,\sigma)$ the voxel of $y(x,\sigma)$, by $\hat{A}(\overrightarrow{x\sigma})$ an approximation of $d\sigma$, we can write:

$$\int\limits_{\sigma\in S_R(x)} B(y(x,\sigma))\frac{\cos\theta(x,y)}{\pi}d\sigma \quad \approx \sum_{\sigma\in\Sigma_R(x)} B(V(x,\sigma))\frac{\cos\theta(x,V(x,\sigma))}{\pi}\hat{A}(\overrightarrow{x\sigma})$$

We approximate Equation (1) by the following linear system:

$$B(x) = E(x) + \rho_d(x)\sum_{\sigma\in\Sigma_R(x)} B(V(x,\sigma))\frac{\cos\theta(x,V(x,\sigma))}{\pi}\hat{A}(\overrightarrow{x\sigma}) \tag{2}$$

In Equation (2), the expensive information is the function $V(x,\sigma)$, (the first voxel encountered from x in the direction of σ). In [5], it is precomputed, and thus is very similar to the *form factors* of the classical approach. The term $\hat{A}(\overrightarrow{x\sigma})$ can also be easily precomputed. Note that $V(x,\sigma)$ is not well defined, since several discrete rays may cross x and σ (see Fig. 1). This is not a problem since each possible ray leads to an acceptable solution.

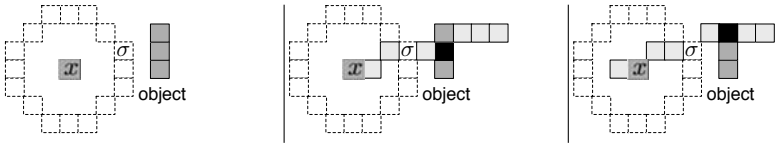

Fig. 1. Given x and σ, several rays may be used to find a $V(x,\sigma)$

3.2 Solving the Equation

To solve Equation (2), a converging iterative method similar to the one used with patch-based radiosity can be used. It usually relies on Gauss-Seidel relaxation. 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 radiosity between mutually visible voxels.

3.3 Computations Made in [5]

The goal of this paper is to introduce a new way to handle $V(x, \sigma)$. In [5], it was precomputed by a kind of discrete ray-tracing method. Since the voxelization that we use produces an octree, it was easy and efficient. This precomputation is however very expensive in terms of time and storage.

The $\hat{A}(\overrightarrow{x\sigma})$ term has not been modified since [5]. As with the hemicube, each exterior voxel face of the surrounding sphere is a kind of screen cell. Thus, it is associated to a solid angle and each voxel is given a value summing the solid angles of its exterior faces. Moreover, the discrete sphere is considered to be global, there is not one sphere per voxel. Therefore, $\hat{A}(\overrightarrow{x\sigma})$ can rather be written $\hat{A}(\overrightarrow{\sigma})$, and we call it a *direction factor*. A radius of 38 for the sphere produces about $15,000$ direction factors. This is a small amount of data, and requires only a few seconds to be pre-computed. It is also a good parameter for the quality of the voxel-based radiosity algorithm.

4 Discrete Geometry

The algorithm that we introduce uses some notions of discrete geometry. Section 4.1 is a reminder, and Section 4.2 is the proof of an interesting partitioning property.

4.1 Discrete Lines

A 2D discrete line [6] 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. ω is related to the *connectivity* of the line (see Fig. 2). 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.

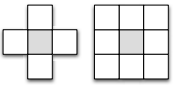

Fig. 2. (2D) {4-8}-neighborhood

Fig. 3. (3D) {6-18-26}-neighborhood

A notion of 3D discrete line has also been defined [3], where the naïve (26-connected) and the standard (6-connected) cases are also defined, as:

$$(x,y,z) \in \mathbb{Z}^3 / \begin{cases} \mu \leq cx - az < \mu + \omega \\ \mu' \leq bx - ay < \mu' + \omega' \end{cases}$$

It is noteworthy that 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 retreive the 3D line. The connectivity (see Fig. 2 and Fig. 3) is related to ω and ω'. If the two 2D projections are naïve (resp. standard), the 3D line is naïve (resp. standard) itself.

4.2 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 1. *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. The projections are defined by:

$$\begin{cases} \{(x,y,z) \in \mathbb{Z}^3 / z = 0 \text{ and } \mu_{ab} \leq -bx + ay < \mu_{ab} + \omega_{ab}\} & \textit{projection in (Oxy)} \\ \{(x,y,z) \in \mathbb{Z}^3 / y = 0 \text{ and } \mu_{ac} \leq -cx + az < \mu_{ac} + \omega_{ac}\} & \textit{projection in (Oxz)} \end{cases}$$

Thus, the 3D discrete line is equivalent to the set of all $(x, y, z) \in \mathbb{Z}^3$ 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.

5 Voxel-Based Radiosity

5.1 A New Approach of Discrete Radiosity

The main problem of the approach presented in [5] 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. Giving full sense to discrete geometry, a new approach can be introduced, that deduces visibility on-the-fly. No precomputations 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.

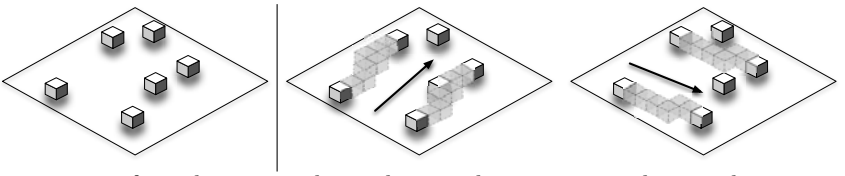

Fig. 4. Visibility solving by finding voxels on the same 3D discrete lines

Transforming the Visibility Problem: Instead of iterating on each voxel x, and querying $V(x, \overrightarrow{\sigma})$, (*i.e.* the first voxel visible from x in the direction of $\overrightarrow{\sigma}$), we can rather iterate on $\overrightarrow{\sigma}$ and compute a whole bunch of $V(x, \overrightarrow{\sigma})$ at a time for a fixed $\overrightarrow{\sigma}$. In this case, we represent a ray of light by the list of every intersected voxels, as if there was no occlusion. Thus, one list encodes several information about visibility in the given direction of the ray. If x and y are two consecutive voxels (not necessarily neighbors in space) in a ray (list) of direction $\overrightarrow{\sigma}$, then we can assume that $y = V(x, \overrightarrow{\sigma})$ (see Fig. 4).

Building the lists could still done by discrete ray-tracing, which has been largely studied [1] [9] [8], but we propose another approach, more adapted to the current context, where we have to compute at the same time the visibility of every couple of voxels. Our approach does not use any traversal ; in our model, the voxels of empty spaces are not represented nor traversed : they are implicitly ignored. To perform this approach, we have addressed two difficulties: how to quickly find the ray (or 3D discrete line) each voxel belongs to, and how to keep at a low cost a sorted representation of this ray to handle the analogy between consecutiveness and visibility.

5.2 Efficient Solving of the Visibility Problem

Finding Which Line a Voxel Belongs to: We have proved in Section 4.2 that the space could be partitioned into 3D discrete lines, each one characterized by a couple of integers (i, j). Such a couple is extractable in constant time from the coordinates of a voxel, for a particular direction. With appropriate data structures, a linear time complexity can then be ensured to link each voxel to its line.

We represent each discrete ray by a list of the voxels it contains, not necessarily sorted at first. The set of all lists is stored in a 2D array, indexed by i and j, which are bounded by the scene geometry. Given a direction, finding the (i, j) list of a given voxel is a constant-time operation. Then, finding this list in the array of lists is also a constant-time operation. Then, adding the voxel to this list is a constant-time operation, if no sorting is done. Therefore, 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 $O(N)$.

$$Voxel \underset{extracting}{\overset{O(1)}{\longmapsto}} (i,j) \underset{finding}{\overset{O(1)}{\longrightarrow}} list \underset{inserting}{\overset{O(1)}{\longmapsto}} increased\ list$$

Finding Consecutive Voxels in the Visibility Lists: The lists holding the voxels are not sorted at first: they do not reflect the *visibility* between voxels. Hence, at first sight, the list filling step should be followed by, or mixed up with a sorting step. However, the complexity would increase up to worst case $O(N \log N)$. We show how appropriate data structures can lead to avoid this sorting step.

Building Sorted Lists Without Extra Cost: Sorting the lists is not required if they can be filled in a way such that the voxels are already sorted in each of them. To ensure that property, an appropriate traversing order must be found for the data structure supplying the voxels to the list-filling algorithm.

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 (see Fig. 5).

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. The relative order of x, y and z is not relevant for the discrete lines we use, since their "thickness" is at most 1, due to our choice of ω. In this case, no "bubble" can appear, which would require attention (see Fig. 6).

In the 3D case, the lexicographic order is applied on x, y, and z depending on the considered direction. 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.* For instance, with intuitive notations, the lexicographic order for $\overrightarrow{v} = (-1, 5, 2)$ can be defined as $\prec_{x\downarrow y\uparrow z\uparrow}$, $\prec_{x\downarrow z\uparrow y\uparrow}$, $\prec_{y\uparrow x\downarrow z\uparrow}$, $\prec_{y\uparrow z\uparrow x\downarrow}$, $\prec_{z\uparrow x\downarrow y\uparrow}$, or $\prec_{z\uparrow y\uparrow x\downarrow}$.

Moreover, since $\prec_{x\downarrow y\uparrow z\uparrow}$ gives the reverse order of $\prec_{x\uparrow y\downarrow z\downarrow}$ (for instance), only 4 out of 8 different lexicographic orders are to be coded to handle any direction.

So far, no conditions were required on the data structure supplying the voxels to the radiosity algorithm. But we need the ability to be given the voxels with respect to one of the four lexicographic order previously defined. In practice, without assumptions on the original data structure, it is possible to use four additional arrays, one for each lexicographic order, containing sorted pointers to the N voxels. This is affordable as a precomputation, at worst 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.

5.3 The Final Algorithm

We have shown in the previous section 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 [5]. The final algorithm is described on page 401.

Time Complexity: The algorithm requires two parameters, which are the radius R of the discrete sphere used to compute the direction factors, and a num-

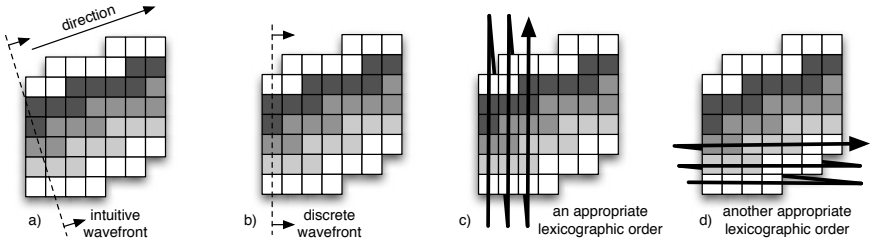

Fig. 5. An appropriate traversing order of voxels respects their natural order in the discrete lines. a) A wavefront encounters the voxels in the good order. b) In the discrete case, the wavefront may be oriented along a coordinate axis. c) and d) This wavefront can be represented by a lexicographic order on the coordinates

Fig. 6. A 3D discrete line whose thickness is limited cannot hold such a bubble, for which the relative order of x, y and z is discriminative

ber I of iterations to 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:

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

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.

```
Prepare the needed four lexicographic orders;

Compute a set of discrete directions;
Pour each direction faire
    compute and store the associated direction factor;
Fin Pour
[the number of iterations is a small constant]
Pour iterations = 1 to MaxIterations faire
    [the number of directions is a big constant]
    Pour each direction faire
        [dispatching step: O(N)]
        Select the appropriate lexicographic order;
        Pour each voxel (with respect to the lexicographic order) faire
            [The voxels are naturally sorted at insertion]
            Add it to the back of the list (3D discrete line) it belongs to;
        Fin Pour
        [propagation (solving) step: O(N)]
        Pour each list faire
            propagate radiosity between contiguous voxels;
        Fin Pour
        reset the lists;
    Fin Pour
Fin Pour
```

Algorithm 1: Quasi-linear radiosity algorithm

6 Experimental Results

Improvements over Previous Voxel-Based Method: The scene presented in [5] 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.

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 dramatical 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 one hand, more resources help optimizing, on the other hand, it does not dramatically outperform the

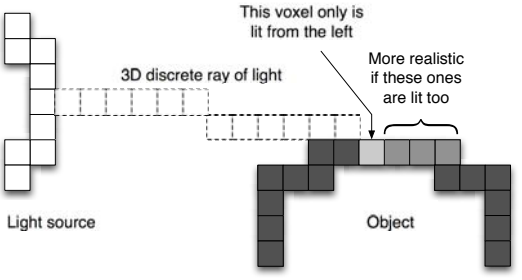

Fig. 7. Small correction on propagation for tangent rays

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.

Visual Artifacts: A roughly discretized scene may contain classical artifacts, when rendering with radiosity information supplied by the present form of the algorithm: light leaks, or sharp shadows might be observed. A low number of directions in the parametrization may as well generate irregularities on surfaces that should render smooth in reality. However, an artifact specific to our algorithm has a specific workaround. Let us consider a ray almost tangent to a surface. Chances are high that many contiguous voxels of the surface belong to the same list representing the ray. With basic visibility, only the first voxel of the surface encountered from the light source would receive light, the others being occluded. With a limited number of directions, this could prevent some voxels in the middle of large flat surfaces to ever receive light directly from the light sources that are low in their horizon. To address this issue, a small correction can be done, that spread the light received by a voxel to its immediate neighbors (see Fig. 7).

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 may add to our radiosity the management of complex Bidirectional Reflectance Distribution Functions (BRDF). We are also studying another voxelization method found in [7], a density approach showing nice results coupled with discrete ray-tracing. We are actually investigating to see if radiosity plugs well into this approach. At last, the algorithm suits well for clustering, so that our implementation let us hope for a linear improvement with respect to the number of nodes. A basic parallelization would dispatch the directions, so that each node handles a subset of directions, but we are also studying a way to dispatch the voxels on the nodes.

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. 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.
3. Isabelle Debled-Rennesson. *Étude et reconnaissance des droites et plans discrets*. PhD thesis, Université Louis Pasteur, Strasbourg, 1995.
4. François X. Sillion and Claude Puech. *Radiosity & Global Illumination*. Morgan Kaufmann Publishers, Inc., 1994.
5. 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.
6. Jean-Pierre Reveillès. *Géometrie discrète, calcul en nombres entiers et algorithmique*. PhD thesis, Université Louis Pasteur, Strasbourg, 1991.
7. M. Sramek and A. Kaufman. Vxt: a c++ class library for object voxelization. *Volume Graphics*, pages 119–134, 2000.
8. Nilo Stolte and René Caubet. Discrete ray-tracing of huge voxel spaces. *Comput. Graph. Forum*, 14(3):383–394, 1995.
9. R. Yagel, D. Cohen, and A. Kaufman. Discrete ray tracing. *IEEE Computer Graphics & Applications*, 12(9):19–28, 1992.