HAL
open science

A 3D curvilinear skeletonization algorithm with application to path tracing

John Chaussard, Laurent Noël, Venceslas Biri, Michel Couprie

To cite this version:

John Chaussard, Laurent Noël, Venceslas Biri, Michel Couprie. A 3D curvilinear skeletonization algorithm with application to path tracing. Discrete Geometry for Computer Imagery, Mar 2013, Spain. pp.1–12. hal-00796545

HAL Id: hal-00796545

https://hal.science/hal-00796545

Submitted on 4 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A 3D curvilinear skeletonization algorithm with application to path tracing

John Chaussard[1], Laurent Noël[2], Venceslas Biri[2], and Michel Couprie[3]*

[1] Université Paris 13, Sorbonne Paris Cité, LAGA, CNRS(UMR 7539),
F-93430, Villetaneuse, France
chaussard@math.univ-paris13.fr
[2] Université Paris Est, LIGM, CNRS(UMR 8049)
Cité Descartes, 5 boulevard Descartes, 77454 Marne-la-Valle CEDEX 22, France
laurent.noel@esiee.fr, v.biri@esiee.fr
[3] Université Paris Est, LIGM, A3SI-ESIEE, CNRS(UMR 8049)
2, boulevard Blaise Pascal, 93162 Noisy le Grand CEDEX, France
michel.couprie@esiee.fr

Abstract. We present a novel 3D curvilinear skeletonization algorithm which produces filtered skeletons without needing any user input, thanks to a new parallel algorithm based on the cubical complex framework. These skeletons are used in a modified path tracing algorithm in order to produce less noisy images in less time than the classical approach.

1 Introduction

Path tracing algorithms [8, 9] are able to render photorealistic images from a scene description by simulating light propagation. The photorealistic aspect is achieved by simulating diffusion of light: elements of the scene which are illuminated by a light source can cast part of the light they receive. Consequently, parts of the scene which are not directly illuminated by any light source actually receive light, thus giving a realistic effect of light diffusion to the resulting image.

A path tracing algorithm sends rays from the camera, and let them bounce around the scene until they encounter a directly illuminated zone. When a ray hits a point of any surface, it is reemitted in a random direction. Once it reaches a directly illuminated area, all luminance gathered through its journey in the scene is averaged in order to set one of the image pixels, a process repeated for each pixel of the output image. To reduce the noise resulting from this stochastic process, the algorithm actually sends many rays for each pixel of the image: the more rays, the longer the algorithm will take to produce an image, but the more realistic the result will look. A path tracing can be seen as a random walk process through the scene.

In order to avoid infinite bouncing of rays in the scene, a limit is set for the number of bounces a ray can do. If a ray does not reach any directly illuminated

* This work has been partially supported by the "ANR-2010-BLAN-0205 KIDICO" project

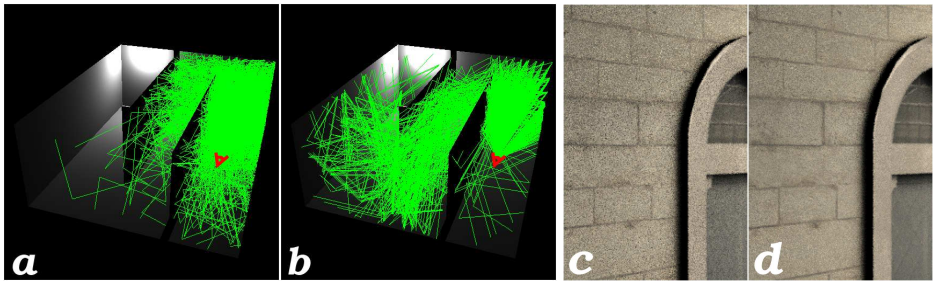

Fig. 1. Note how rays are guided toward the illuminated area thanks to our skeleton based algorithm (image *b*), compared to classical path tracing (image *a*). Images *c* and *d* point out the noise reduction between the classic path tracing (*c*) and our method (*d*).

area after a given number of bounces, its contribution is considered null. Such lost ray is a waste of computation time, and minimizing this loss would speed up the algorithm (see Fig 1.a).

Our proposition is to reduce both the time taken by a path tracing algorithms and the noise in the output images by guiding rays towards the scene's light sources (see Fig 1.b), thus reducing the number of rays "lost" during the computation. To do this, we perform some precomputation on the void space of the scene (the medium where the rays of light propagate) in order to have, later in the algorithm, a clue on the direction a ray should bounce in order to meet an illuminated area. Unfortunately, such precomputation may take time and become more expensive than the original algorithm.

To keep the precomputation fast, we perform it only on a subset of points of the voids of the scene. Since this precomputation relies on paths and visibility tests, we want this set of points to be representative of both the topology and the geometry of the original scene. For example, it should possess branches propagating in the elongated parts of the scene. A filtered curvilinear skeleton of the scene's void meets all the required properties for our subset of points: it is a thin subset of points, with the same topology and retaining the main geometrical information of the scene. Note that in this application, the filtering of the skeleton (elimination of spurious branches) need to be both efficient and completely automatic, for one cannot ask a user to set parameters for each scene.

The purpose of this paper is twofold: we first introduce a new curvilinear skeletonization algorithm (Sec. 3) in the cubical complex framework (Sec. 2), producing filtered skeletons without needing any user input (Sec. 4). Then, we expose a new method, based on such skeletons, to enhance the results (see Fig 1.c and d) and the performance of the path tracing algorithm (Sec. 5).

2 The cubical complex framework

In the 3D voxel framework, objects are made of voxels. In the 3D cubical complex framework, objects are made of cubes, squares, lines and vertices. Let $\mathbb{Z}$ be the set of integers, we consider the family of sets $\mathbb{F}_0^1$ and $\mathbb{F}_1^1$, such that $\mathbb{F}_0^1 = \{\{a\} \mid a \in \mathbb{Z}\}$ and $\mathbb{F}_1^1 = \{\{a, a+1\} \mid a \in \mathbb{Z}\}$. Any subset f of $\mathbb{Z}^n$ such that f is the Cartesian product of m elements of $\mathbb{F}_1^1$ and $(n-m)$ elements of $\mathbb{F}_0^1$ is called a face or an *m-face* of $\mathbb{Z}^n$, m is the dimension of f, we write $dim(f) = m$. A 0-face is called a *vertex*, a 1-face is an *edge*, a 2-face is a *square*, and a 3-face is a *cube*.

Given $m \in \{0, \ldots, n\}$, we denote by $\mathbb{F}_m^n$ the set composed of all m-faces in $\mathbb{Z}^n$. We denote by $\mathbb{F}^n$ the set composed of all m-faces in $\mathbb{Z}^n$: $\mathbb{F}^n = \bigcup_{m \in [0;n]} \mathbb{F}_m^n$. Let $f \in \mathbb{F}^n$. We set $\hat{f} = \{g \in \mathbb{F}^n | g \subseteq f\}$, and $\hat{f}^* = \hat{f} \setminus \{f\}$. Any element of $\hat{f}$ (resp. $\hat{f}^*$) is *a face of f* (resp. *a proper face of f*). The *closure* of a set of faces X is the set $\widehat{X} = \cup\{\hat{f} | f \in X\}$.

Definition 1. *A finite set X of faces in $\mathbb{F}^n$ is a* complex *if $X = \widehat{X}$, and we write $X \preceq \mathbb{F}^n$.*

Any subset Y of X which is also a complex is a subcomplex of X*, and we write $Y \preceq X$.*

A face $f \in X$ is *a facet of X* if f is not a proper face of any face of X. The *dimension of X* is $\dim(X) = \max\{\dim(f) \mid f \in X\}$. If $\dim(X) = d$, then we say that X is a d-complex.

Traditionally, a binary image is a finite subset of $\mathbb{Z}^n$ (called voxel image when $n = 3$). To transpose such an image S to the cubical complex framework, we associate to each element of $S \subseteq \mathbb{Z}^n$ an n-face of $\mathbb{F}^n$. Let $x = (x_1, ..., x_n) \in S$, we define the n-face $\Psi(x) = \{x_1, x_1+1\} \times \ldots \times \{x_n, x_n+1\}$. We can extend the map Ψ to sets: $\Psi(S) = \{\Psi(x) | x \in S\}$. Given a set $S \subset \mathbb{Z}^n$, we associate to it the cubical complex $\widehat{\Psi(S)}$.

The collapse operation is the basic operation for performing homotopic thinning of a complex, and consists of removing free pairs of faces:

Definition 2. *Let $X \preceq \mathbb{F}^n$, and let f, g be two faces of X. The face g is* free for X*, and the pair (f, g) is* a free pair for X *if f is the only face of X which strictly contains g.*

It can be easily seen that if (f, g) is a free pair for a complex X, then f is a facet of X and $\dim(g) = \dim(f) - 1$.

Definition 3. *Let $X \preceq \mathbb{F}^n$, and let (f, g) be a free pair for X. The complex $X \setminus \{f, g\}$ is an* elementary collapse of X.

Let $Y \preceq \mathbb{F}^n$. The complex X collapses onto Y *if there exists a sequence of complexes $(X_0, ..., X_\ell)$ of $\mathbb{F}^n$ such that $X = X_0$, $Y = X_\ell$ and for all $i \in \{1, \ldots, \ell\}, X_i$ is an elementary collapse of X_{i-1}. We also say, in this case, that* Y is a collapse of X.

Recent works in the cubical complex framework brought new parallel thinning algorithms in the voxel framework ([2], [1]).

3 A parallel directional thinning based on cubical complex

In the cubical complex framework, parallel removal of free pairs can be easily achieved when following simple rules that we give now. First, we need to define the *direction* and the *orientation* of a free face. Let (f, g) be a free pair for $X \preceq \mathbb{F}^n$: we have $\dim(g) = \dim(f) - 1$, and it can be seen that $g = f \cap f'$, where f' is the translate of f by one of the $2n$ vectors of $\mathbb{Z}^n$ which have all their coordinates equal to 0 except one, which is either equal to +1 or -1. Let v be this vector, and c its non-null coordinate. We define $Dir(f, g)$, called the *direction* of the free pair (f, g), as the index of c in v. The *orientation* of the free pair (f, g) is defined as $Orient(f, g) = 1$ if $c = 1$, and $Orient(f, g) = 0$ else.

Now, we give a property of collapse (previously proven in [4]) which brings a necessary and sufficient condition for removing two free pairs of faces in parallel from a complex, while preserving topology.

Proposition 4. *Let $X \preceq \mathbb{F}^n$, and let (f, g) and (k, ℓ) be two distinct free pairs for X. The complex X collapses onto $X \setminus \{f, g, k, \ell\}$ if and only if $f \neq k$.*

From Prop. 4, the following corollary is immediate.

Corollary 5. *Let $X \preceq \mathbb{F}^n$, and let $(f_1, g_1) \dots (f_m, g_m)$ be m distinct free pairs for X such that, for all $a, b \in \{1, \dots, m\}$ (with $a \neq b$), $f_a \neq f_b$. The complex X collapses onto $X \setminus \{f_1, g_1 \dots f_m, g_m\}$.*

Considering two distinct free pairs (f, g) and (i, j) for $X \preceq \mathbb{F}^n$ such that $Dir(f, g) = Dir(i, j)$ and $Orient(f, g) = Orient(i, j)$, we have $f \neq i$. From this observation and Cor. 5, we deduce the following property.

Corollary 6. *Let $X \preceq \mathbb{F}^n$, and let $(f_1, g_1) \dots (f_m, g_m)$ be m distinct free pairs for X having all the same direction and the same orientation. The complex X collapses onto $X \setminus \{f_1, g_1 \dots f_m, g_m\}$.*

Intuitively, we want our thinning algorithm to remove free faces of a complex "layer by layer" and to avoid having unequal thinning of the input complex. Therefore, we want each execution of the algorithm to remove free faces located on the border of the input complex. We define $Border(X)$ as the set all faces belonging to a free pair for X. We now introduce Alg. 1, a directional parallel thinning algorithm.

On a single execution of the main loop of Alg. 1, only faces located on the border of the complex are removed (l. 7). Thanks to corollary 6, we can remove faces with same direction and orientation in parallel (l. 8), while guaranteeing topology preservation. Figure 2 depicts the first steps of the algorithm.

Different definitions of orientation and direction can be given, each corresponding to a different order of free faces removal in the complex and leading to different results. Algorithm 1 can be implemented to run in linear time complexity (proportionally to the number of faces in the complex). Indeed, checking if a face is free or not may be easily done in constant time and when a free pair (f, g) is removed from the input complex, it is sufficient to scan the faces contained in f in order to find new free faces.

Algorithm 1: $ParDirCollapse(X, W, \ell)$

Data: A cubical complex $X \preceq \mathbb{F}^n$, a subcomplex $W \preceq X$ which represents faces of X which should not be removed, and $\ell \in \mathbb{N}$, the number of layers of free faces which should be removed from X

Result: A cubical complex

```
while there exists free faces in X \ W and ℓ > 0 do
    L = Border^(X);
    for t = 1 → n do
        for s = 0 → 1 do
            for d = n → 1 do
                E = {(f, g) free for X | g ∉ W, Dir(f, g) = t, Orient(f, g) = s,
                dim(f) = d};
                G = {(f, g) ∈ E | f ∈ L and g ∈ L};
                X = X \ G;
    l = l − 1;
return X;
```

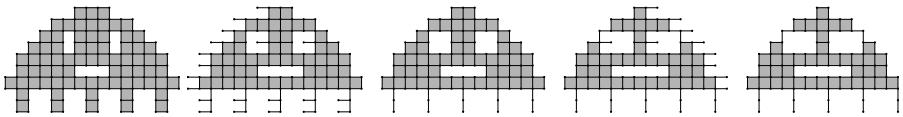

Fig. 2. Four first iterations of Alg. 1 running on the left-most shape.

4 Aspect preservation during thinning

As previously said, our goal is to use the skeleton of the voids of a scene in order to guide light rays from the camera of the scene towards the light. Moreover, this skeleton needs to capture the main geometrical features of the original scene. For example, if the input is a corridor (the void of the corridor), then the skeleton should be a line following the main direction of the corridor.

Generally, two strategies are possible to achieve this goal: find, during the skeletonization process, points whose neighbourhood configuration seems interesting and keep them in the result [6] [7] [5], or choose, before skeletonization, interesting points of the object which should remain untouched, based on a function on these points and a filtering parameter [2] [12].

Algorithm 1 does not necessarily preserve geometrical features of the input object in the resulting skeleton (for example, the skeleton of a corridor could be reduced to a single vertex). In the following, we introduce a new method in the cubical complex framework, requiring no user input, for obtaining a curvilinear skeleton yielding satisfactory geometrical properties. Our method is based on the two previously listed strategies: it finds, during thinning, elements with a specific neighbourhood configuration, and uses a function on these elements to decide whether to preserve them, or not, in the result.

4.1 The lifespan of a face

In the following, we define additional functions in the cubical complex, related to the thinning process (Alg. 1), which are essential for the filtering step of the skeletonization. The first one we present is the *death date* of a face.

Definition 7. *Let $f \in X \preceq \mathbb{F}^n$. The* death date of f in X*, denoted by $Death_X(f)$, is the smallest integer δ such that $f \notin ParDirCollapse(X, \emptyset, \delta)$.*

Intuitively, the death date of a face indicates how many layers of free faces should be removed from a complex X, using Alg. 1, before removing completely the face from X. We now define the *birth date* of a face:

Definition 8. *Let $f \in X \preceq \mathbb{F}^n$. The* birth date of f in X*, denoted by $Birth_X(f)$, is the smallest integer b such that either f is a facet of $ParDirCollapse(X, \emptyset, b)$, or $f \notin ParDirCollapse(X, \emptyset, b)$.*

The birth date indicates how many layers of free faces must be removed from X with Alg.1 before transforming f into a facet of X (we consider that a face "lives" when it is a facet). Finally, we define the *lifespan* of a face :

Definition 9. *Let $f \in X \preceq \mathbb{F}^n$. The lifespan of f in X is the integer*

$$Lifespan_X(f) = \begin{cases} +\infty & \text{if } Death_X(f) = +\infty \\ Death_X(f) - Birth_X(f) & \text{otherwise} \end{cases}$$

These three values depend on the order of direction and orientation chosen for Alg. 1.

The lifespan of a face f of X indicates how many "rounds" this face "survives" as a facet in X, when removing free pairs with Alg. 1, and is a good indicator of how important a face can be in an object. Typically, the higher the lifespan is, and the more representative of an object's geometrical feature the face is. The lifespan, sometimes called *saliency*, was used in [10] (with the name "medial persistence") in order to propose a thinning algorithm in cubical complexes based on two parameters.

4.2 Distance map, opening function and decenterness map

In addition to the lifespan of a face, the proposed homotopic thinning method uses information on distance between faces in order to decide if a face should be kept safe from deletion. We define hereafter various notions based on distances in the voxel framework.

We set $d_1(x, y)$ as the L1 distance between x and y (Manhattan distance). Let $S \subset \mathbb{Z}^n$, we set $S^c = \mathbb{Z}^n \setminus S$, and for all $x \in \mathbb{Z}^n$, the map $D_1(S) : \mathbb{Z}^n \to \mathbb{N}$ is such that $D_1(S)(x) = \min_{y \in S^c} d_1(x, y)$.

The *maximal* 1*-ball in S centered on x* is the set $\mathbb{MB}^1_S(x) = \{y \in \mathbb{Z}^n | d_1(x, y) < D_1(S)(x)\}$. We set, for all $x \in S$, the map $\Omega_1(S) : \mathbb{Z}^n \to \mathbb{N}$ such that $\Omega_1(S)(x) =$

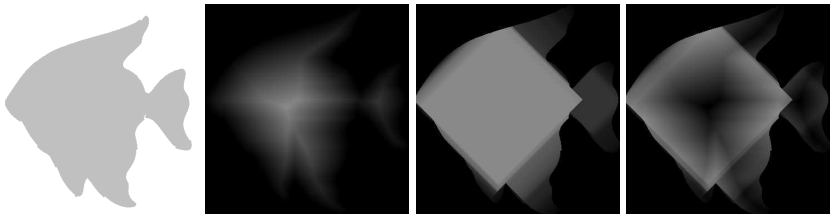

Fig. 3. Examples of opening and decenterness map - From left to right: a shape $S \subset \mathbb{Z}^2$ (in gray), $D_1(S)$, $\Omega_1(S)$ and $\mathcal{DC}_1(S)$ (low values have dark colour).

$\max_{x \in \mathbb{MB}^1_S(y)} D_1(S)(y)$: this value indicates the radius of a largest maximal 1-ball contained in S and containing x. If $x \in S^c$, we set $\Omega_1(S)(x) = 0$. The map $\Omega_1(S)$ is known as the opening function of S based on the 1-distance (also called the granulometry function) [11]: it allows to compute efficiently results of morphological openings by balls of various radius, and gives information on the local thickness of an object.

Given $S \subset \mathbb{Z}^n$, the value of $\Omega_1(S)(x)$ of every $x \in S$ can be naively computed by performing successive morphological dilations of values of the map $D_1(S)$. A linear algorithm for computing the map $\Omega_1(S)$ (with regard to the size of the input image) was proposed in [3], and will be explored further in details in a future paper.

Finally, we define the *decenterness map*:

Definition 10. *Given $S \subset \mathbb{Z}^n$, the* decenterness map of S *is the map $\mathcal{DC}_1(S) = \Omega_1(S) - D_1(S)$.*

An example of these maps is shown on Fig. 3.

In order to extend all these previous maps defined in $\mathbb{Z}^n$ to the cubical complex framework, we use the map Ψ^{-1}, inverse of the bijective map $\Psi : \mathbb{Z}^n \to \mathbb{F}^n_n$ defined in Sec. 2. It is used to project any n-face of $\mathbb{F}^n$ into $\mathbb{Z}^n$. This map induces a map from $\mathcal{P}(\mathbb{F}^n_n)$ to $\mathcal{P}(\mathbb{Z}^n)$, that we also denote by Ψ^{-1}.

Given $Y \subset \mathbb{F}^n$, we set $S = \Psi^{-1}(Y \cap \mathbb{F}^n_n)$. We define the map $D^{cc}_1(Y) : \mathbb{F}^n \to \mathbb{N}$ as follows: for all $f \in \mathbb{F}^n$,

$$D^{cc}_1(Y)(f) = \begin{cases} D_1(S)(\Psi^{-1}(f)) & \text{if } f \text{ is an } n\text{-face} \\ \max\limits_{f \in (\hat{g}^* \cap \mathbb{F}^n_n)} D^{cc}_1(Y)(g) & \text{otherwise} \end{cases}$$

Informally, if f is a 3-face, then $D^{cc}_1(Y)(f)$ is the length of the shortest 1-path between the voxel "corresponding" to f and the set of voxels corresponding to Y. In the same way, we define $\Omega^{cc}_1(Y)$ and $\mathcal{DC}^{cc}_1(Y)$.

4.3 Parameter-free filtered thinning

As previously said, we add edges to the set W of Alg. 1 in order to retain, in the resulting curvilinear skeleton, important edges from the original object. Given

Algorithm 2: $CurvilinearSkeleton(X)$

Data: A cubical complex $X \preceq \mathbb{F}^3$
Result: A cubical complex $Y \preceq \mathbb{F}^3$
1 $W = \{f \in X | Lifespan_X(f) > \mathcal{DC}_1^{cc}(X)(f) + Birth_X(f) - D_1^{cc}(X)(f)$ and $\dim(f) = 1\}$;
2 **return** $ParDirCollapse(X, W, +\infty)$;

a cubical complex X, if an edge of X has a high decenterness value for X, then it is probably located too close to the border of X and does not represent an interesting geometrical feature to preserve. On the other hand, if an edge has a high lifespan for X, then it means it was not removed quickly, after becoming a facet, by the thinning algorithm and might represent some precious geometrical information on the original object. An idea would be to keep, during thinning, all edges whose lifespan is superior to the decenterness value. Unfortunately, this strategy produces skeletons with many spurious branches in surfacic areas of the original object.

We can identify surfacic areas of a complex as zones where squares have a high lifespan. Therefore, in order to avoid spurious branches in surfacic areas, we need to make it harder for edges to be preserved in these zones. It can be achieved by deciding that an edge will be kept safe from deletion by the thinning algorithm if its lifespan is superior to the decenterness value plus the lifespan of squares "around" this edge. This leads us to proposing Alg. 2.

In order to understand what was realised on line 1 of Alg. 2, we might point out that the birth date of an edge corresponds to the highest death date of the squares containing this edge. Moreover, the map $D_1^{cc}(X)$ gives, for all 3-faces of X, their death date (as the thinning algorithm naturally follows this map to eliminate cubes from a 3-complex). Therefore, for an edge f of X, $D_1^{cc}(X)(f)$ informs us on the highest death date of cubes containing f, also equal to the highest birth date of squares containing f. In conclusion, $Birth_X(f) - D_1^{cc}(X)(f)$ is an approximation of the lifespan of the squares containing f.

Although the output of Alg.2 may contain 2-faces, the algorithm is said to be a *curvilinear skeletonization* algorithm because it only adds 1-faces (edges) in the inhibitor set W (used for the visual aspect preservation step) (on line 1 of Alg. 2). For the same reason, we say that the output of Alg. 2 is a *curvilinear skeleton*. In the scenes studied in Sec. 5, the outputs of Alg. 2 were one dimensional complexes.

4.4 Results

Algorithm 2 allows to obtain a filtered curvilinear skeleton from a three dimensional complex. The results presented in Fig. 4 show that the skeletons contain the main geometrical information from the input shapes, and no spurious branch.

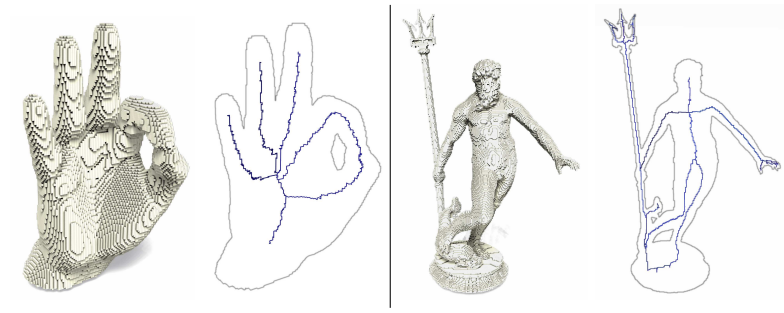

Fig. 4. Results of algorithm 2 for two shapes: a hand (left) and a statue (right). In each pair, the rightmost image represents the skeleton.

5 Application to path tracing

In this section, we explain how the curvilinear skeleton of the voids of the scene can enhance the path tracing algorithm. We start with some basic elements related to path tracing and then present our modified path tracing algorithm.

5.1 The path tracing

Let O be the origin of the camera in the scene. For each pixel P of the final image, let x be the nearest intersection point of the ray $(O, \overrightarrow{OP} = -\Theta)$ with an element of the scene. To obtain the luminosity, we must solve the rendering equation [8] which is a recursive integral equation, the integrand containing the radiance function that we must compute:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Phi \in \Omega_x} f_s(x, \Theta \leftrightarrow \Phi) L(r(x, \Phi) \to -\Phi) d\omega_\Phi$$

where L is the radiance (a radiometric quantity that represents luminosity) from point x toward direction Θ and L_e is the emitted radiance (non null only on light sources). The point $r(x, \Phi)$ is the nearest visible point from x in the direction Φ of the hemisphere Ω_x. The f_s function expresses how much luminosity is exchanged, on point x, between an incoming ray Φ and the outgoing ray Θ.

The previous equation expresses an intuitive idea: the reflected radiance from x towards the camera is the result of computing all incoming luminosity on x scaled by a factor which depends on the material on x and the angle of the ray going from the camera to x. The most common method used to estimate the integral, denoted by $L_r(x \to \Theta)$, is the Monte-Carlo integration which provides the following:

$$\langle L_r(x \to \Theta) \rangle = \frac{f_s(x, \Theta \leftrightarrow \Phi) L(r(x, \Phi) \to -\Phi)}{p(\Phi)}$$

The function p is a probability density function (pdf) that is used to sample Φ. This estimator is unbiased, meaning that the expected value $E[\langle L_r(x \to \Theta) \rangle]$

is equal to $L_r(x \to \Theta)$. The variance of the estimator expresses its quality and depends on the chosen pdf p. The best choice is a pdf that matches the shape of the function to integrate (ie. gives high density to samples that have high values for the function and low density to samples that have low values). The strategy of choosing an adapted pdf is called *importance sampling* [8] and is used in global illumination to improve the convergence speed of the algorithms.

The algorithm computes L as follow: it chooses one random direction Φ based on the pdf p and applies the estimator by calling L recursively to compute L_r (shoots a new ray in the direction Φ, computes the new hit point with the scene, and computes the new radiance at this point). Finally, it returns the sum of the emitted radiance L_e and the result for L_r. The recursion stops when either a maximal number of bounces or a directly illuminated area have been reached.

This computation is done multiple times for each pixel. We average the results and get an estimation of the mean radiance passing through the pixel and heading towards the camera. A bad pdf would lead in picking directions that do not reach the light before the end of the recursion, and produce results with a lot of noise in the final image. In the next part, we explain how to produce a pdf based on the curvilinear skeleton.

5.2 Skeleton based importance sampling

As stated in the introduction, a curvilinear skeleton of the void (with some preprocessing performed on it) gives us information on which directions the light comes from. Given these directions, we can build a efficient pdf p_{skel} and guide our rays by sampling the hemispheres with p_{skel}. The integrand of L_r is:

$$f_s(x, \Theta \leftrightarrow \Phi) L(r(x, \Phi) \to -\Phi).$$

The most common strategy used to sample Ω_x is to use the function f_s because it is an input of the algorithm. The term L, representing the distribution of light in the scene, is unknown. Our method gives a way to sample L.

Construction of the importance points The skeleton of the voids of the scene is computed using Alg. 2 (as shown on Fig. 5 on the right) and is converted into a graph (the nodes are the 0-faces and the edges are the 1-faces). We then compute a set of *importance points*, which will be used to sample Ω_x in the path tracing algorithm. To each node n of the skeleton, one importance point imp_n is computed. Intuitively, the importance point associated to n is the direction to follow in order to find a light source.

Let L be the light source of the scene and n_L the nearest node of the skeleton that is visible by L. For each node, we compute the shortest path to n_L along the skeleton. To do so, we use the Dijkstra algorithm and weight the edges of the skeleton depending on a visibility criteria: the weight of an edge e is 1 if e is visible from n_L and 10 else. It results that illuminated paths will be shorter than paths located in dark areas.

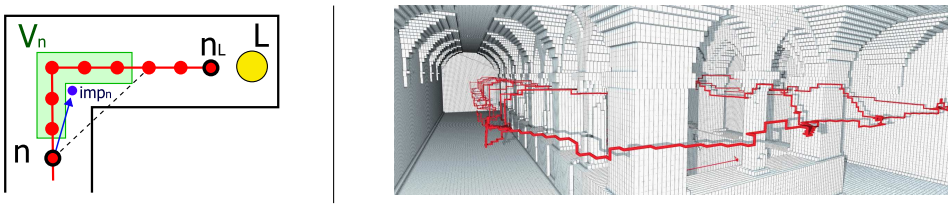

Fig. 5. On the left, we illustrate the importance point imp_n associated to a node n. The dotted black line shows the closest node to n not included in V_n. **On the right**, an example of the curvilinear skeleton (in red) obtained in the Sponza scene.

Let n be a node of the skeleton and V_n the set of visible nodes from n along the shortest path toward n_L. The importance point imp_n associated to n is the barycenter of V_n. An example is shown on Fig. 5 on the left. The algorithm can be extended to multiple light sources by taking into account, when computing the importance point of a skeleton node, only the closest light source.

Sampling according to L Given a point x on the scene and a direction Θ, we want to compute $L(x \to \Theta)$ and then sample the hemisphere Ω_x. We look for the nearest skeleton node n to x and its importance point imp_n. We sample the hemisphere with a power-cosine pdf centered on $\overrightarrow{x\,imp_n}$:

$$p_{skel}(\Phi) = \frac{s+1}{2\pi} * cos^s \alpha$$

with α the angle between $\overrightarrow{x\,imp_n}$ and Φ, s being a parameter called skeleton strength. The higher s is, the closer to $\overrightarrow{x\,imp_n}$ we sample.

5.3 Results and discussion

Some results are presented on Fig. 6 and Fig. 1 where we can see that our method produces less noisy images compared to the regular path tracing. We present, in Fig 7, the time taken by each method to produce an image of same quality (the quality is measured by the mean square error (MSE) with a reference image) in different scenes: our method is the fastest one.

6 Conclusion

We presented in this article a new skeletonization algorithm that both preserves geometrical features of objects and produces a pure curvilinear skeleton. These two properties allow us to improve the path tracing algorithm in guiding the rays towards the main illuminated area of the scene. Our algorithm is faster and produce less noise than the classical path tracing method.

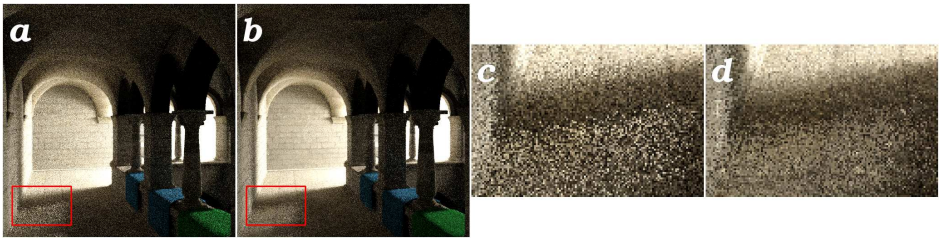

Fig. 6. Same scene rendered using the classical path tracing (image a) or our method (image b). Images c and d are both details of respectively images a and b

Scene	Corridor	Sponza 1	Sponza 2 (Fig. 6)
MSE 100	145 / 57	301 / 156	881 / 826
MSE 40	434 / 260	924 / 628	2678 / 2570

Fig. 7. Time (s) to reach an MSE of 100 and 40 against reference images. In each cell, the left number is for the standard path tracing, the right is for our method

References

1. Bertrand, G.: On critical kernels. Comptes Rendus de l'Académie des Sciences, Série Mathématiques I(345), 363–367 (2007)
2. Bertrand, G., Couprie, M.: A new 3D parallel thinning scheme based on critical kernels. In: Discrete Geometry for Computer Imagery. pp. 580–591. Springer (2006)
3. Chaussard, J.: Topological tools for discrete shape analysis. Ph.D. thesis, Université Paris-Est (December 2010)
4. Chaussard, J., Couprie, M.: Surface thinning in 3d cubical complexes. In: Proceedings of the 13th International Workshop on Combinatorial Image Analysis. pp. 135–148. IWCIA '09, Springer-Verlag, Berlin, Heidelberg (2009)
5. Chaussard, J., Couprie, M., Talbot, H.: Robust skeletonization using the discrete lambda-medial axis. Pattern Recognition Letters In Press, (2010)
6. Couprie, M., Coeurjolly, D., Zrour, R.: Discrete bisector function and euclidean skeleton in 2d and 3d. Image and Vision Computing 25(10), 1543–1556 (2007)
7. Hesselink, W.H., Roerdink, J.B.T.M.: Euclidean skeletons of digital image and volume data in linear time by the integer medial axis transform. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(12), 2204–2217 (2008)
8. Kajiya, T.: The Rendering Equation. In: Computer Graphics (ACM SIGGRAPH '86 Proceedings). vol. 20(4), pp. 143–150 (Aug 1986)
9. Lafortune, E.P., Willems, Y.D.: Bi-directional path tracing. In: Proceedings of the 3rd international conference on computational graphics and visualization techniques. pp. 145–153 (1993)
10. Liu, L., Chambers, E.W., Letscher, D., Ju, T.: A simple and robust thinning algorithm on cell complexes. In: Computer Graphics Forum (Proceedings of Pacific Graphics 2010) (2010)
11. Matheron, G.: Eléments pour une Théorie des Milieux Poreux (1967)
12. Palágyi, K.: A Subiteration-Based Surface-Thinning Algorithm with a Period of Three. In: Pattern Recognition, Lecture Notes in Computer Science, vol. 4713, pp. 294–303. Springer Berlin / Heidelberg (2007)