3d curvilinear skeletonization algorithm

with application to path tracing

a story by

John Chaussard - Laurent Noël
Venceslas Biri - Michel Couprie

produced by

DGCI 2013 sevilla

A3SI

ESIEE PARIS

UNIVERSITÉ PARIS 13 NORD

UNIVERSITÉ PARIS-EST

cnrs

Once upon a time...

Once upon a time...

Once upon a time...

Once upon a time...

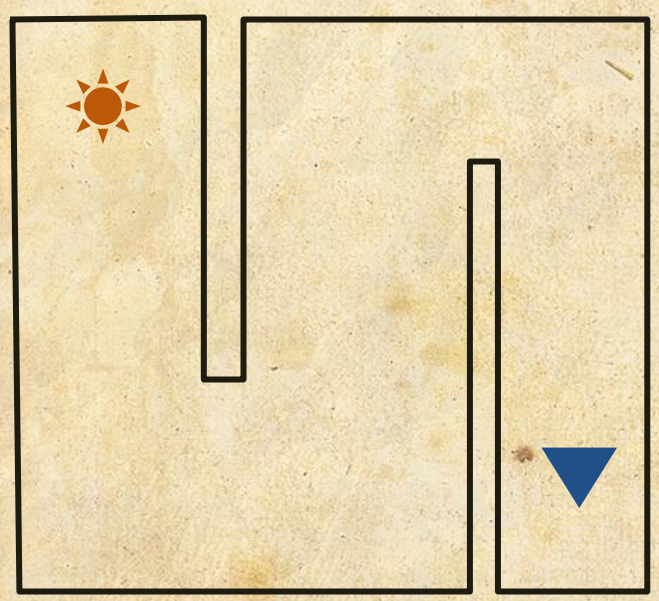
Once upon a time...

Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.

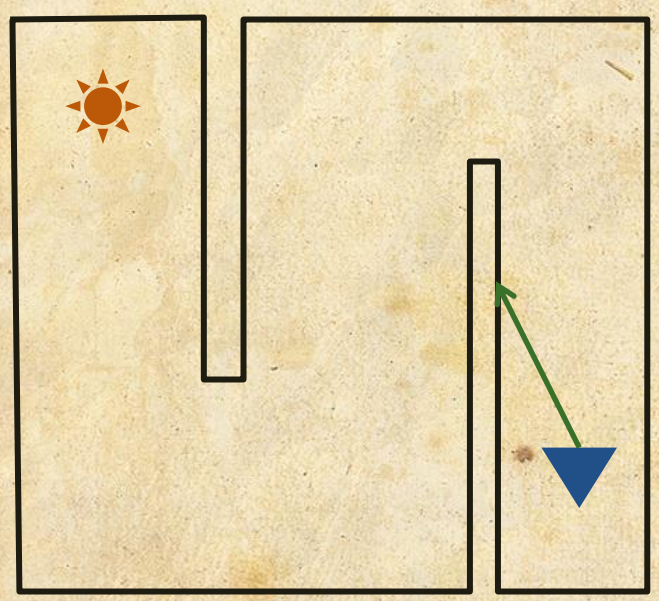
Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.



Path-tracing

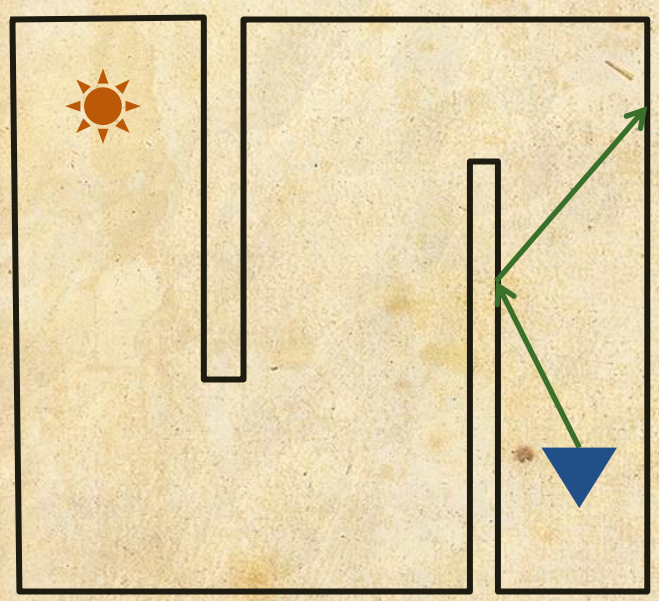
Path-tracing is an algorithm to create **photo realistic images** from a scene.



Virtual rays of light are sent from the camera and propagate through the scene.

Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.

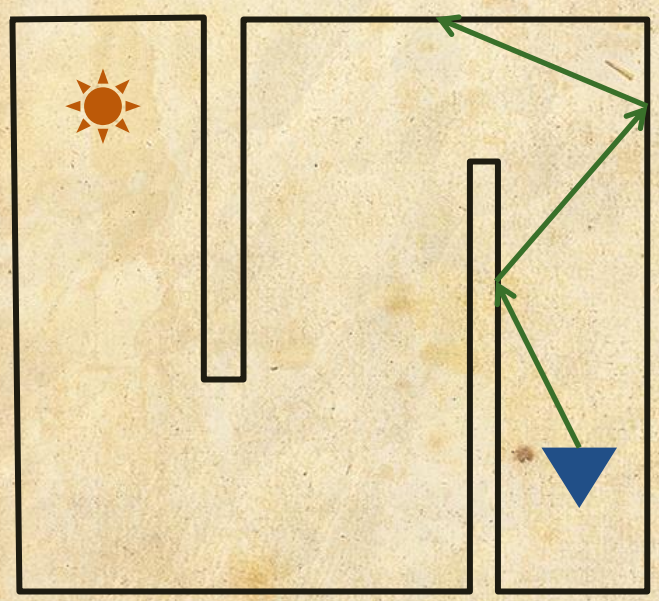


Virtual rays of light are sent from the camera and propagate through the scene.

Each ray of light collects, during its journey through the scene, information on the environment. **The journey ends after a fixed amount of bounces.**

Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.

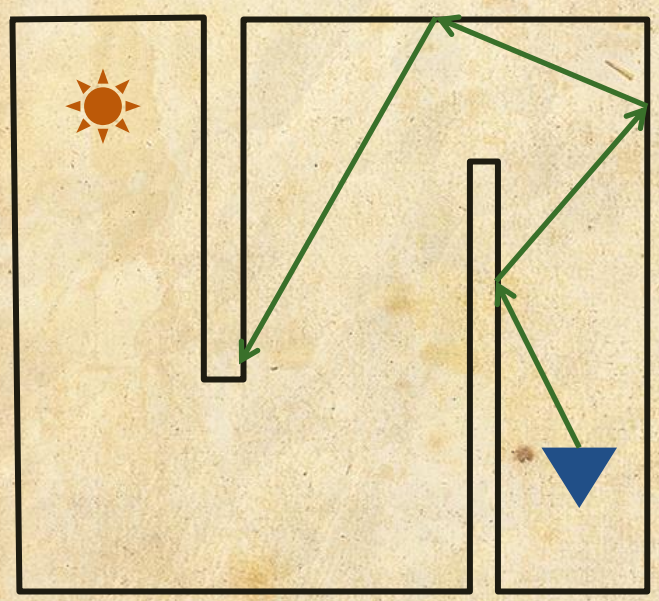


Virtual rays of light are sent from the camera and propagate through the scene.

Each ray of light collects, during its journey through the scene, information on the environment. **The journey ends after a fixed amount of bounces.**

Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.

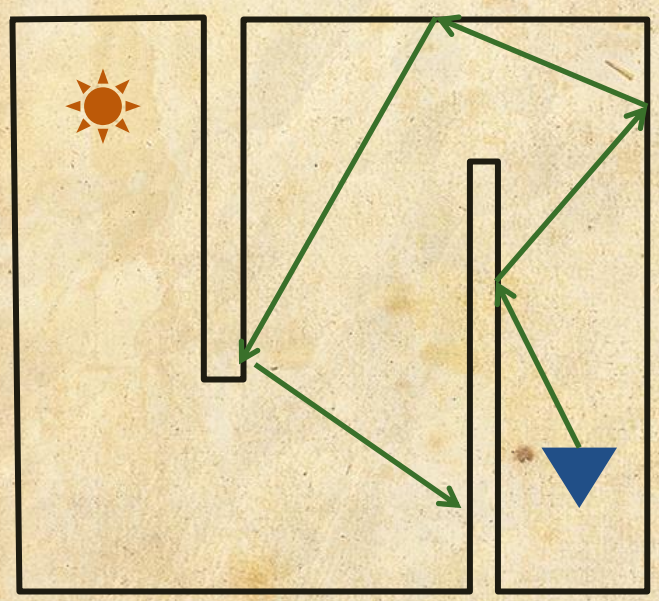


Virtual rays of light are sent from the camera and propagate through the scene.

Each ray of light collects, during its journey through the scene, information on the environment. **The journey ends after a fixed amount of bounces.**

Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.

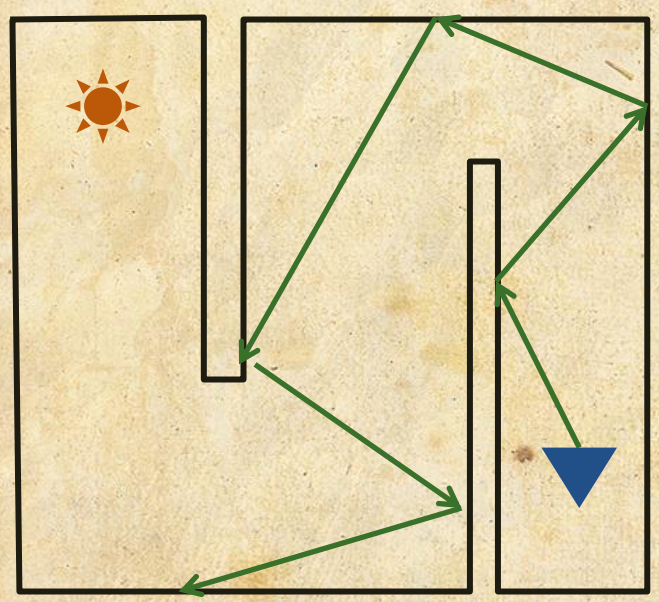


Virtual rays of light are sent from the camera and propagate through the scene.

Each ray of light collects, during its journey through the scene, information on the environment. **The journey ends after a fixed amount of bounces.**

Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.

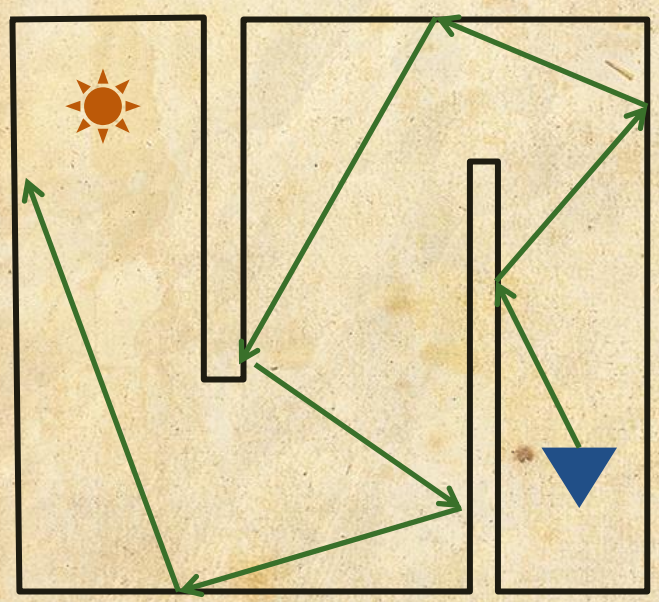


Virtual rays of light are sent from the camera and propagate through the scene.

Each ray of light collects, during its journey through the scene, information on the environment. **The journey ends after a fixed amount of bounces.**

Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.

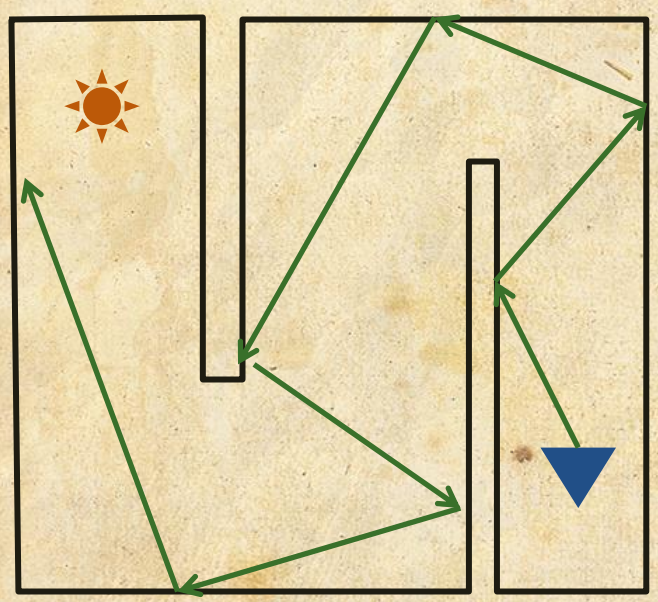


Virtual rays of light are sent from the camera and propagate through the scene.

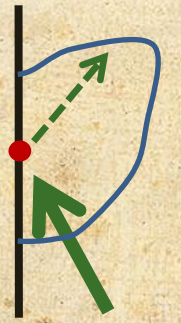
Each ray of light collects, during its journey through the scene, information on the environment. **The journey ends after a fixed amount of bounces.**

Path-tracing

Path-tracing is an algorithm to create **photo realistic images** from a scene.



Virtual rays of light are sent from the camera and propagate through the scene.



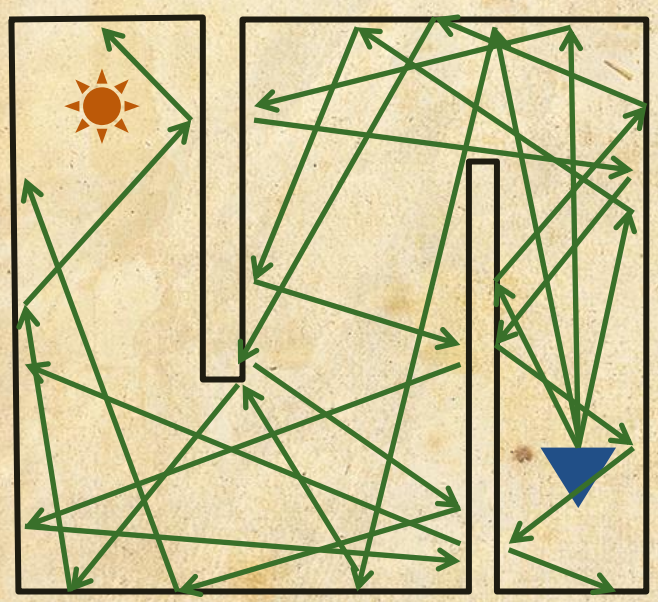
BRDF

The BRDF function of an object tells how light tends to reflect on this object.

Each ray of light collects, during its journey through the scene, information on the environment. **The journey ends after a fixed amount of bounces.**

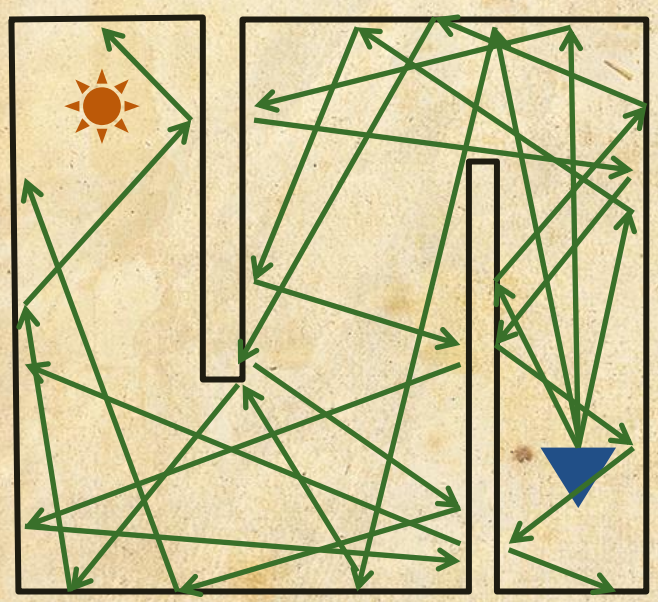
Path-tracing

Many rays must be simulated in order to obtain a photorealistic result.



Path-tracing

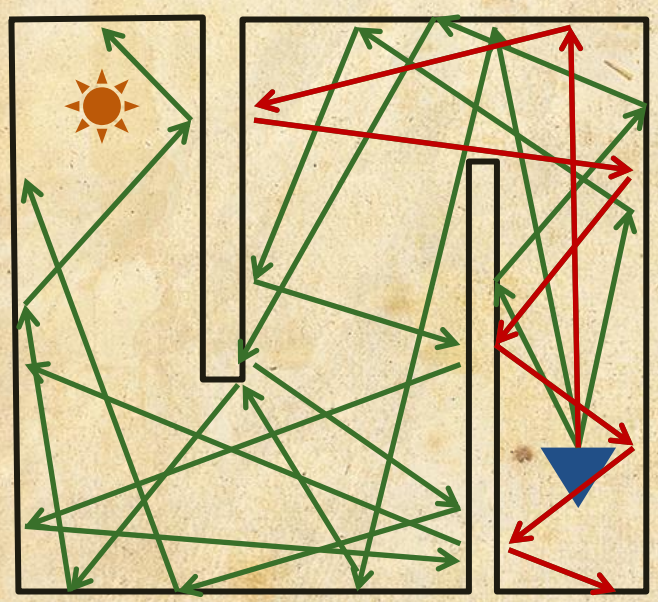
Many rays must be simulated in order to obtain a photorealistic result.



The process is **time–consuming.**

Path-tracing

Many rays must be simulated in order to obtain a photorealistic result.

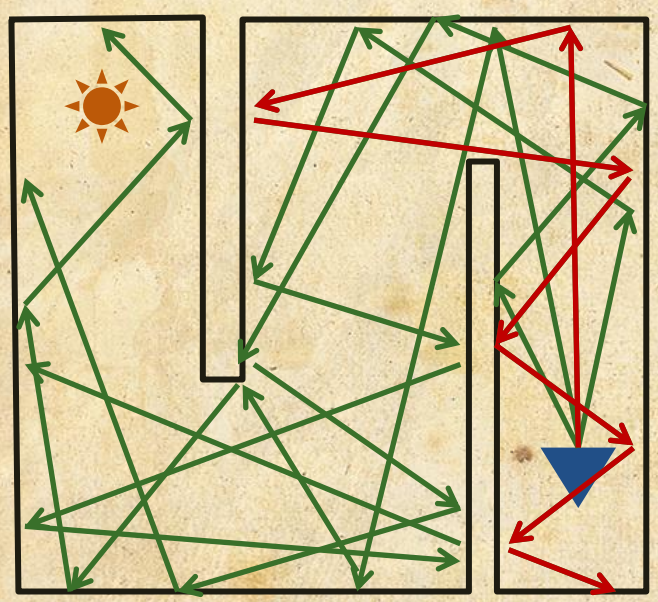


The process is **time–consuming.**

Some rays never reach any bright area of the scene : these rays collect little information on the scene and don't contribute much to the final result.

Path-tracing

Many rays must be simulated in order to obtain a photorealistic result.



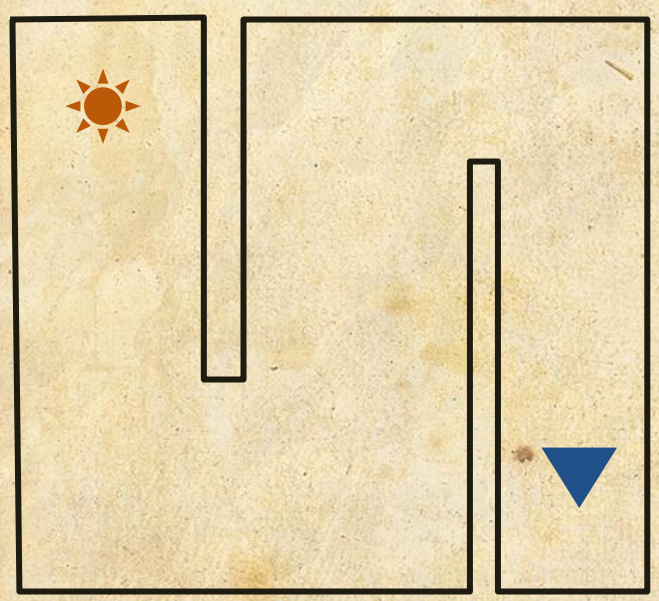
The process is **time–consuming.**

Some rays never reach any bright area of the scene : these rays collect little information on the scene and don't contribute much to the final result.

Reducing the number of these non contributing rays is a way to make the computation faster without changing the output quality.

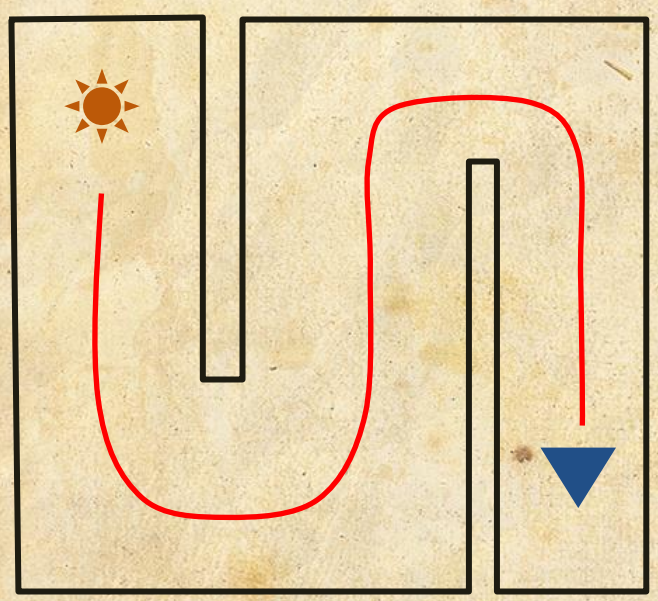
Path-tracing

Our idea is to use a **skeleton of the voids of the scene** in order to **guide the rays towards the bright areas** of the scene.



Path-tracing

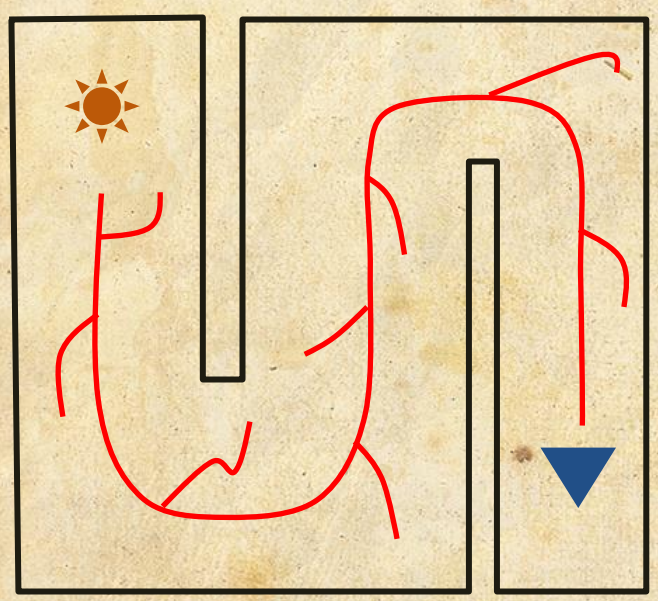
Our idea is to use a **skeleton of the voids of the scene** in order to **guide the rays towards the bright areas** of the scene.



We hope that this strategy will help reduce the number of non contributing rays and will **make the computation faster**.

Path-tracing

Our idea is to use a **skeleton of the voids of the scene** in order to **guide the rays towards the bright areas** of the scene.



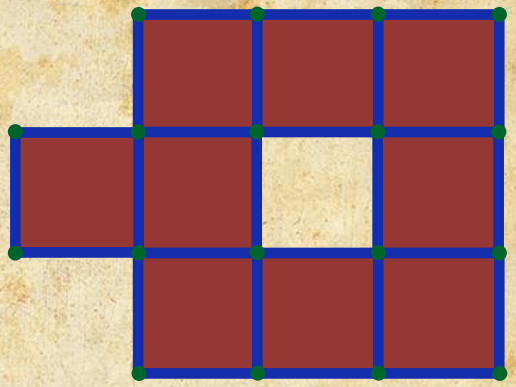
We hope that this strategy will help reduce the number of non contributing rays and will **make the computation faster**.

To be efficient, we need a **skeleton which « looks » like the original object and contain as few spurious branches as possible**.

Curvilinear skeletonization algorithm

Curvilinear skeletonization algorithm

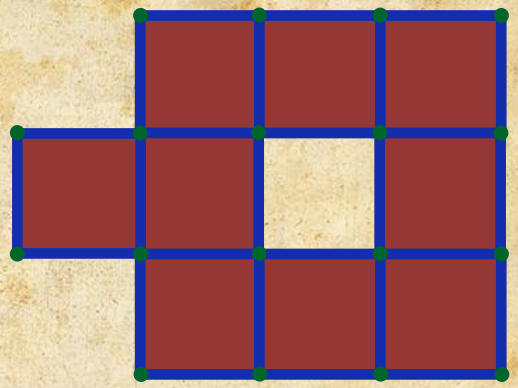
Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).



Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

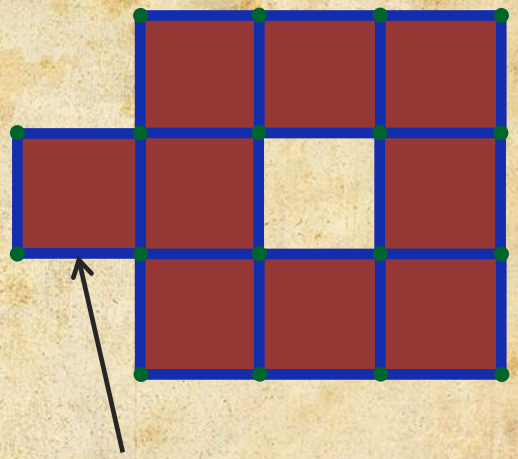
A face is free if it « touches » exactly one face of higher dimension.



Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

A face is free if it « touches » exactly one face of higher dimension.

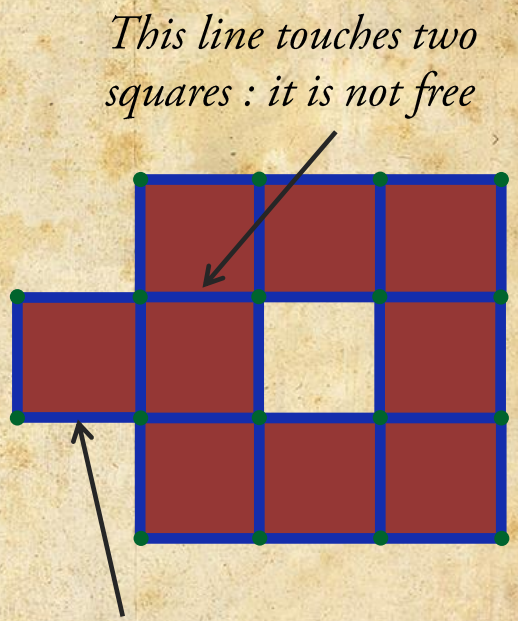


This line touches exactly one square : it is free

Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

A face is free if it « touches » exactly one face of higher dimension.

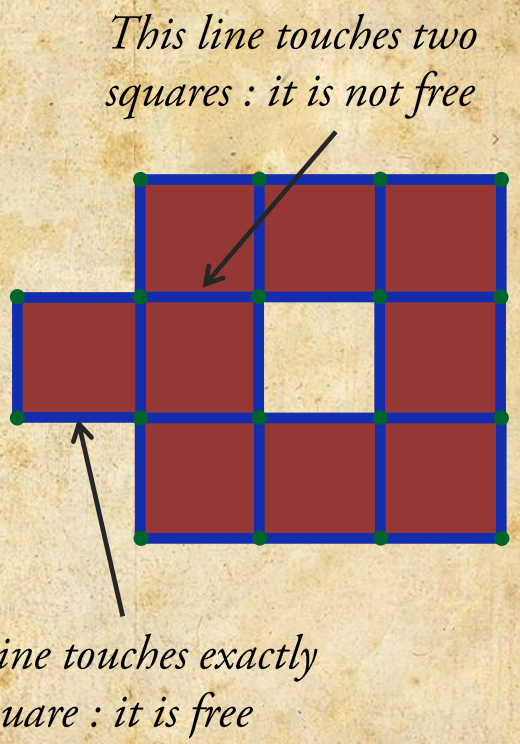


Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

A face is free if it « touches » exactly one face of higher dimension.

Thinning is achieved by removing a free face and the face of higher dimension it touches.

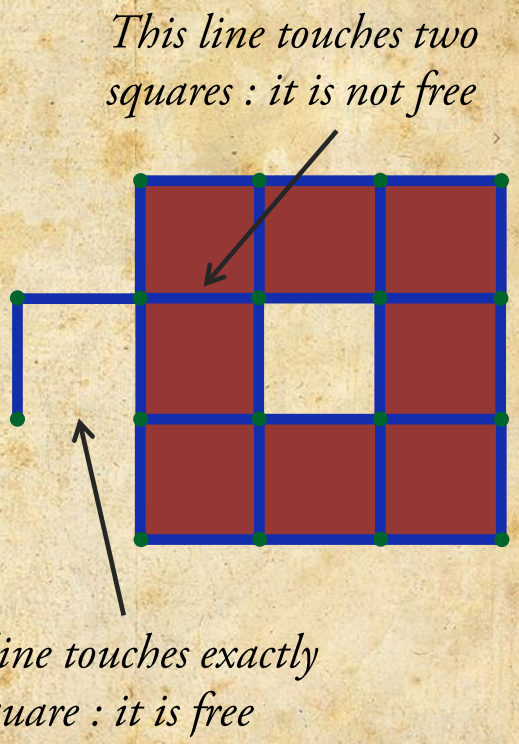


Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

A face is free if it « touches » exactly one face of higher dimension.

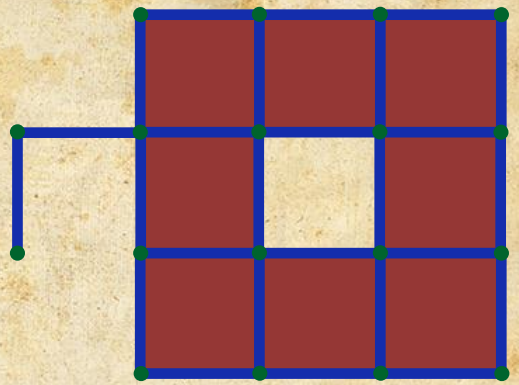
Thinning is achieved by removing a free face and the face of higher dimension it touches.



Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

A face is free if it « touches » exactly one face of higher dimension.



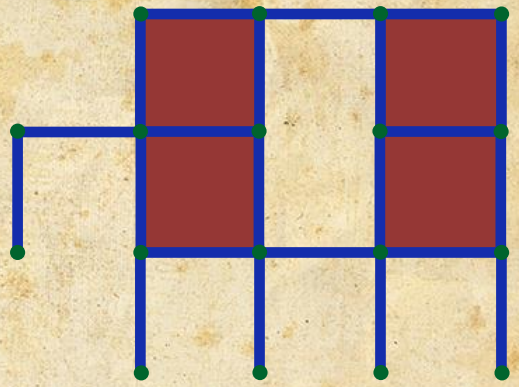
Thinning is achieved by removing a free face and the face of higher dimension it touches.

Our algorithm performs thinning in parallel: all free faces have a given direction and dimension are removed simultaneously (the algorithm iterates the directions and dimensions).

Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

A face is free if it « touches » exactly one face of higher dimension.



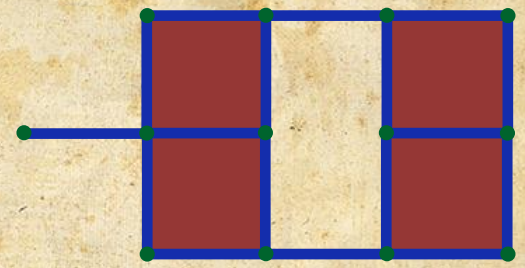
Thinning is achieved by removing a free face and the face of higher dimension it touches.

Our algorithm performs thinning in parallel: all free faces have a given direction and dimension are removed simultaneously (the algorithm iterates the directions and dimensions).

Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

A face is free if it « touches » exactly one face of higher dimension.



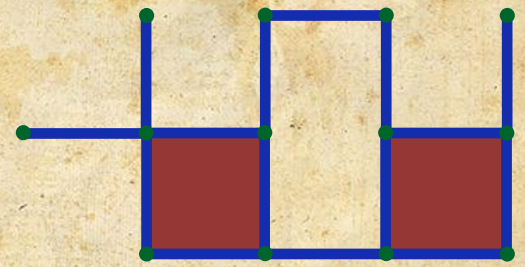
Thinning is achieved by removing a free face and the face of higher dimension it touches.

Our algorithm performs thinning in parallel: all free faces have a given direction and dimension are removed simultaneously (the algorithm iterates the directions and dimensions).

Curvilinear skeletonization algorithm

Our skeletonzation algorithm works on **cubical complexes framework**: objects are not made of voxels, but of cubes, squares, lines and points (called faces).

A face is free if it « touches » exactly one face of higher dimension.



Thinning is achieved by removing a free face and the face of higher dimension it touches.

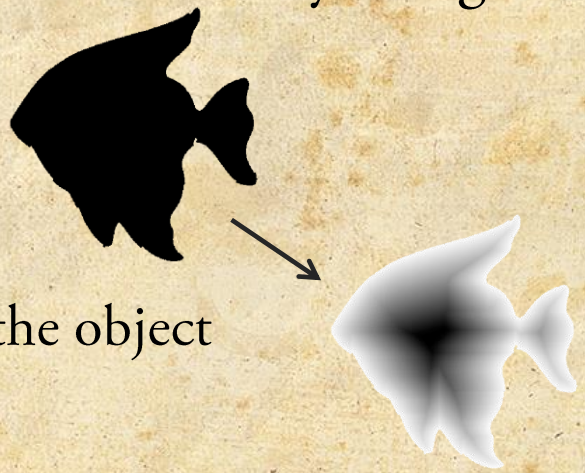
Our algorithm performs thinning in parallel: all free faces have a given direction and dimension are removed simultaneously (the algorithm iterates the directions and dimensions).

Curvilinear skeletonization algorithm

Visual aspect preservation is achieved by keeping some face untouched during the thinning. This is decided by using three elements:

Curvilinear skeletonization algorithm

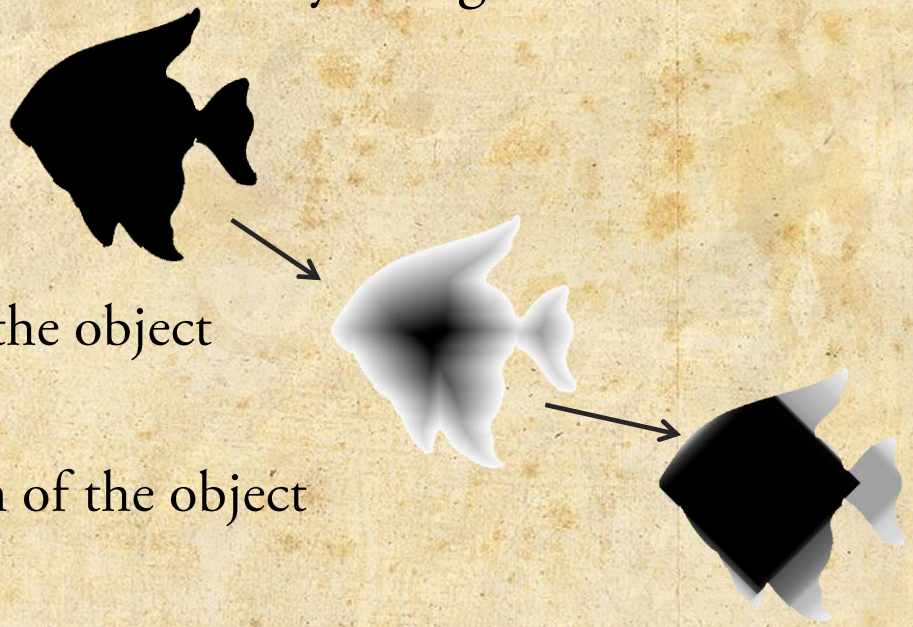
Visual aspect preservation is achieved by keeping some face untouched during the thinning. This is decided by using three elements:



The 6-distance map of the object

Curvilinear skeletonization algorithm

Visual aspect preservation is achieved by keeping some face untouched during the thinning. This is decided by using three elements:

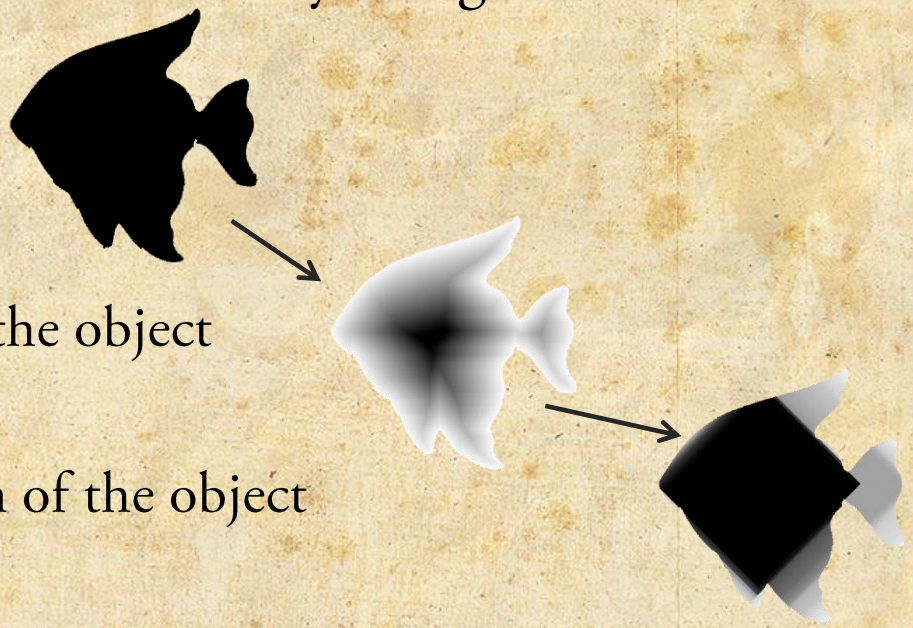


The 6-distance map of the object

The 6-opening function of the object

Curvilinear skeletonization algorithm

Visual aspect preservation is achieved by keeping some face untouched during the thinning. This is decided by using three elements:



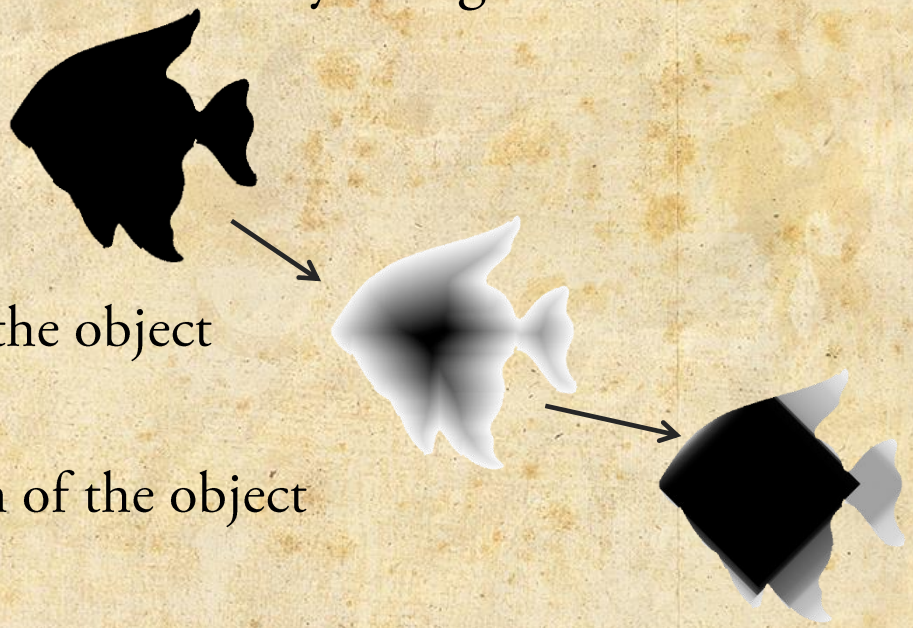
The 6-distance map of the object

The 6-opening function of the object

The time our algorithm needs to « reach » a face and delete it.

Curvilinear skeletonization algorithm

Visual aspect preservation is achieved by keeping some face untouched during the thinning. This is decided by using three elements:



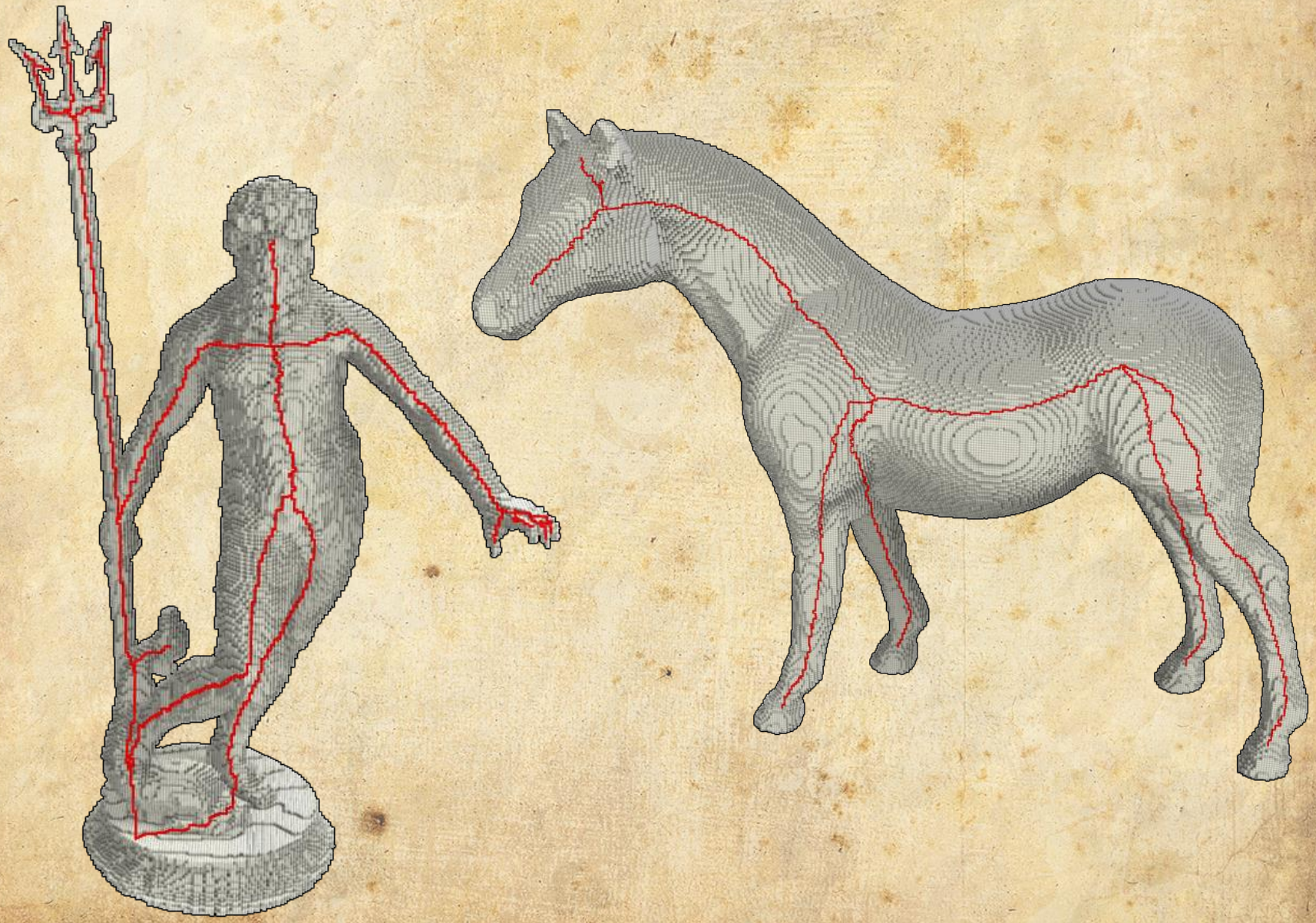
The 6-distance map of the object

The 6-opening function of the object

The time our algorithm needs to « reach » a face and delete it.

The algorithm is entirely automatic: it does not need any user input to perform the filtering.

Curvilinear skeletonization algorithm



Skeleton path-tracing

When a ray hits an object, the bouncing direction depends on two elements:

Skeleton path-tracing

When a ray hits an object, the bouncing direction depends on two elements:

The object's BRDF *(realistic result)*.

Skeleton path-tracing

When a ray hits an object, the bouncing direction depends on two elements:

The object's BRDF *(realistic result)*.

The skeleton node which is closest to a light source and still visible from the hit spot *(help the rays go towards bright areas)*.

Skeleton path-tracing

When a ray hits an object, the bouncing direction depends on two elements:

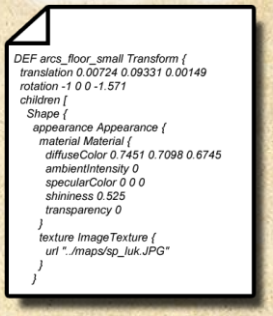
The object's BRDF *(realistic result)*.

The skeleton node which is closest to a light source and still visible from the hit spot *(help the rays go towards bright areas)*.

These two elements must be carefully averaged in order to still obtain photorealistic images and reduce the number of non contributing rays.

Skeleton path-tracing

Workflow



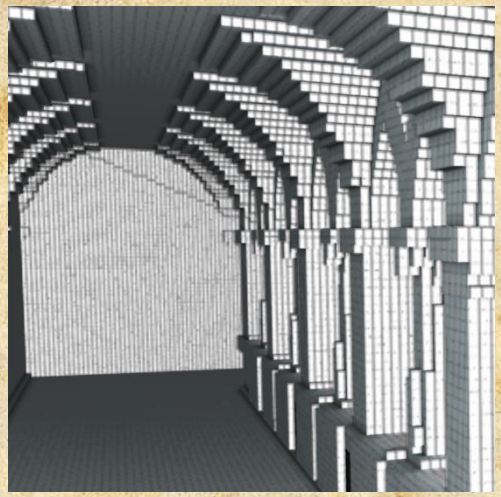
```
DEF arcs_floor_small Transform {
  translation 0.00724 0.09331 0.00149
  rotation -1 0 0 -1.571
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0.7451 0.7098 0.6745
          ambientIntensity 0
          specularColor 0 0 0
          shininess 0.525
          transparency 0
        }
        texture ImageTexture {
          url "../maps/sp_luk.JPG"
        }
      }
```

Skeleton path-tracing

Workflow

```
DEF arcs_floor_small Transform {
  translation 0.00724 0.09331 0.00149
  rotation -1 0 0 -1.571
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0.7451 0.7098 0.6745
          ambientIntensity 0
          specularColor 0 0 0
          shininess 0.525
          transparency 0
        }
        texture ImageTexture {
          url "../maps/sp_luk.JPG"
        }
      }
```

voxelization

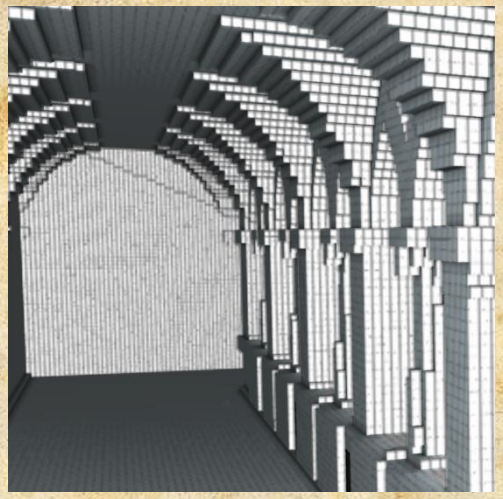


Skeleton path-tracing

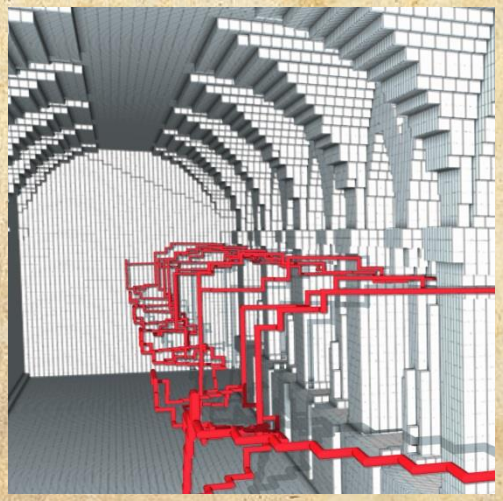
Workflow

```
DEF arcs_floor_small Transform {
  translation 0.00724 0.09331 0.00149
  rotation -1 0 0 -1.571
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0.7451 0.7098 0.6745
          ambientIntensity 0
          specularColor 0 0 0
          shininess 0.525
          transparency 0
        }
        texture ImageTexture {
          url "../maps/sp_luk.JPG"
        }
      }
```

voxelization



thinning

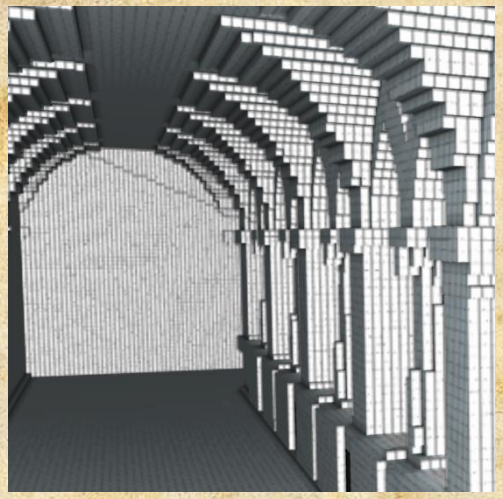


Skeleton path-tracing

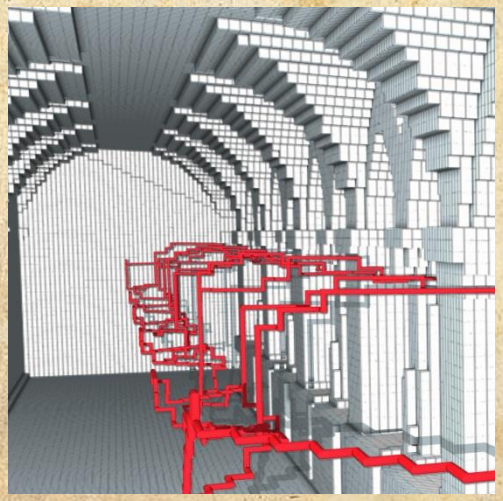
Workflow

```
DEF arcs_floor_small Transform {
  translation 0.00724 0.09331 0.00149
  rotation -1 0 0 -1.571
  children [
    Shape {
      appearance Appearance {
        material Material {
          diffuseColor 0.7451 0.7098 0.6745
          ambientIntensity 0
          specularColor 0 0 0
          shininess 0.525
          transparency 0
        }
        texture ImageTexture {
          url "../maps/sp_luk.JPG"
        }
      }
```

voxelization



thinning

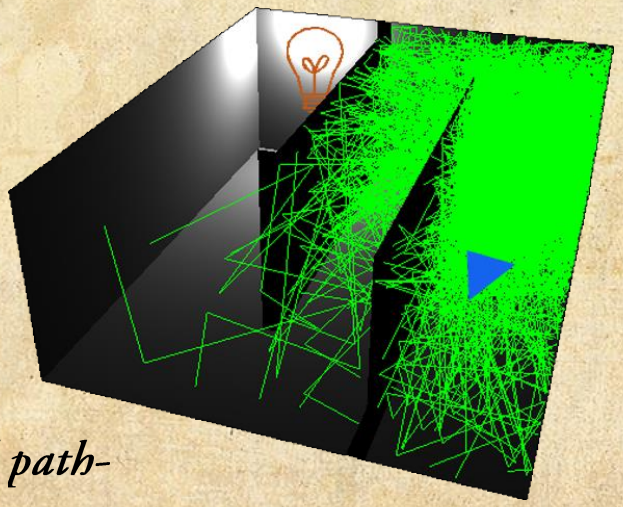


skeleton path-tracing

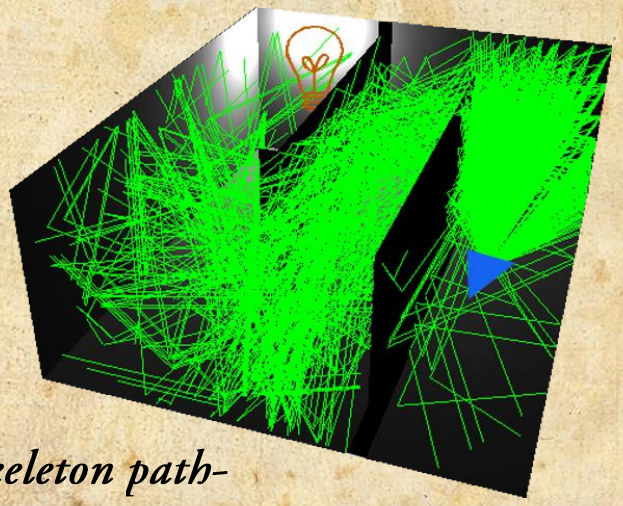
Skeleton path-tracing

Results:

For a same amount of rays sent in the scene, **our method produces more rays reaching bright areas** (useful rays).



classical path-tracing

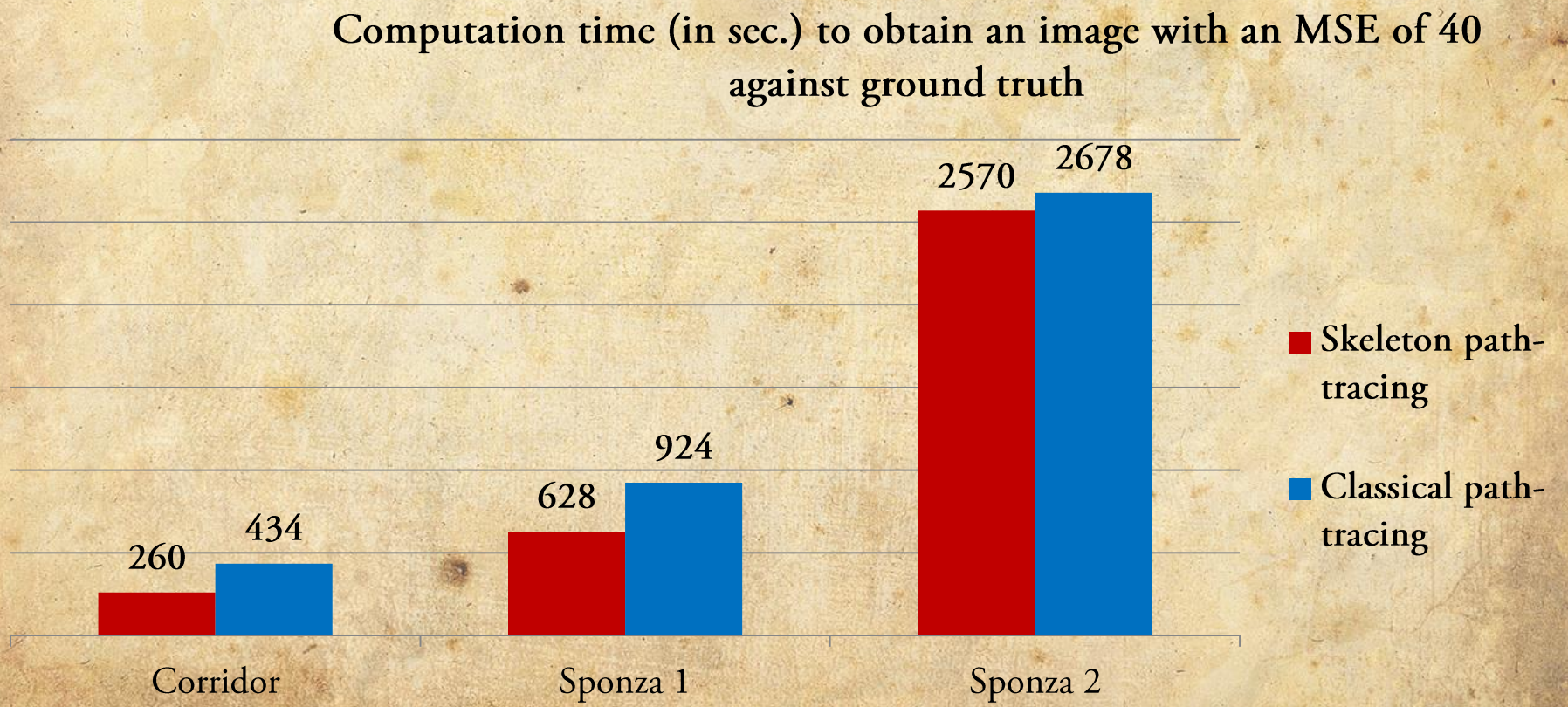


skeleton path-tracing

Skeleton path-tracing

Results:

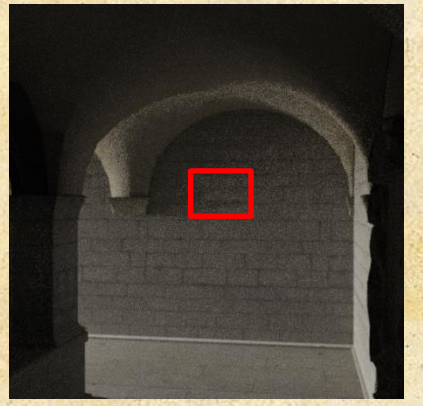
Our method is faster than classical path-tracing to produce same quality images.



Skeleton path-tracing

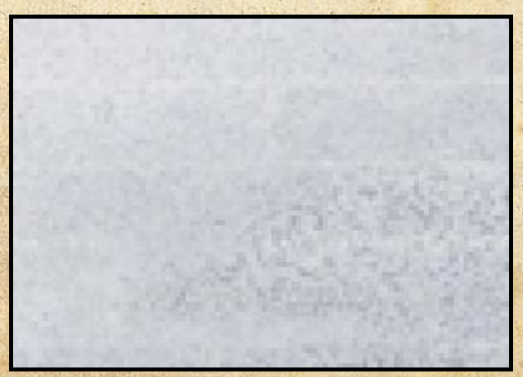
Results:

Our method produces less noisy images.



classical path-tracing

skeleton path-tracing



Thank you for your attention,

Please, come visit our poster !