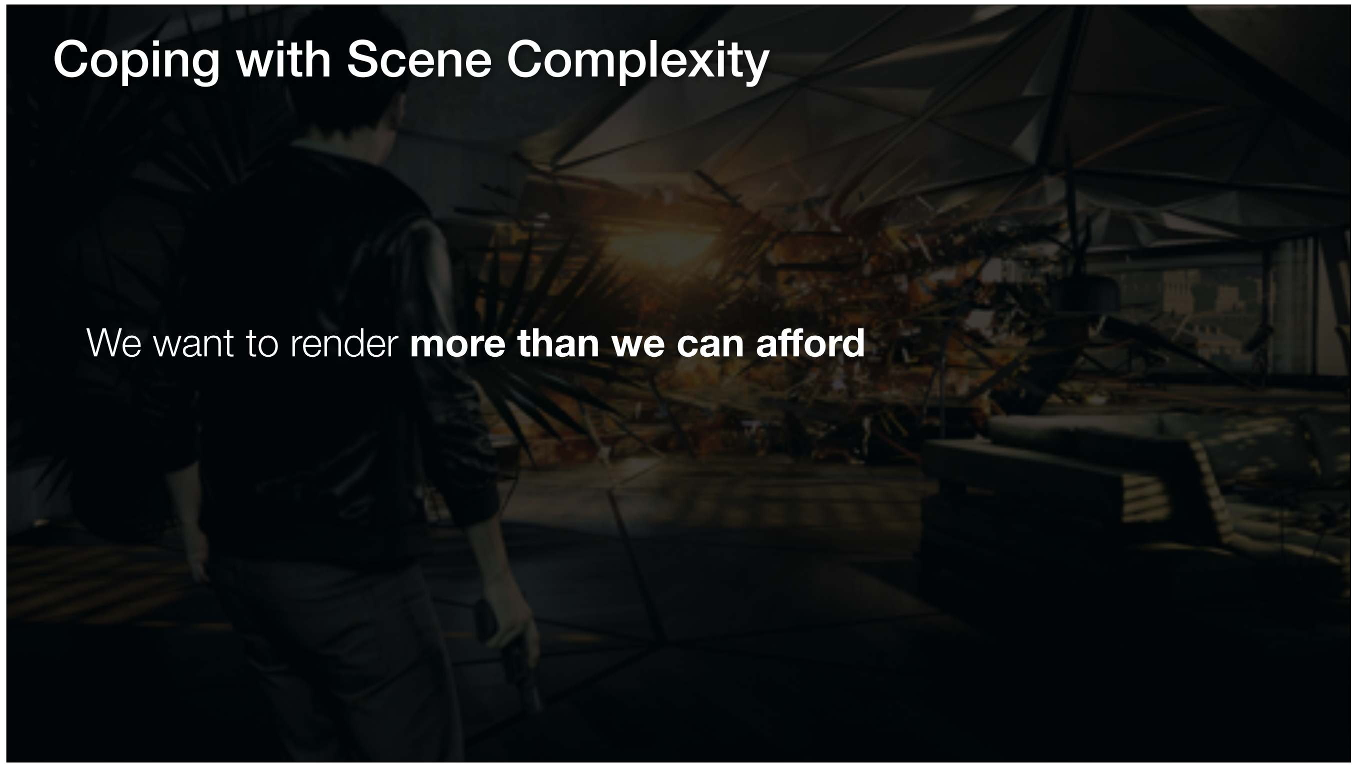
Occluder Simplification using Planar Sections

Ari Silvennoinen
Remedy Entertainment
Aalto University

Hannu Saransaari
Umbra Software

Samuli Laine
NVIDIA

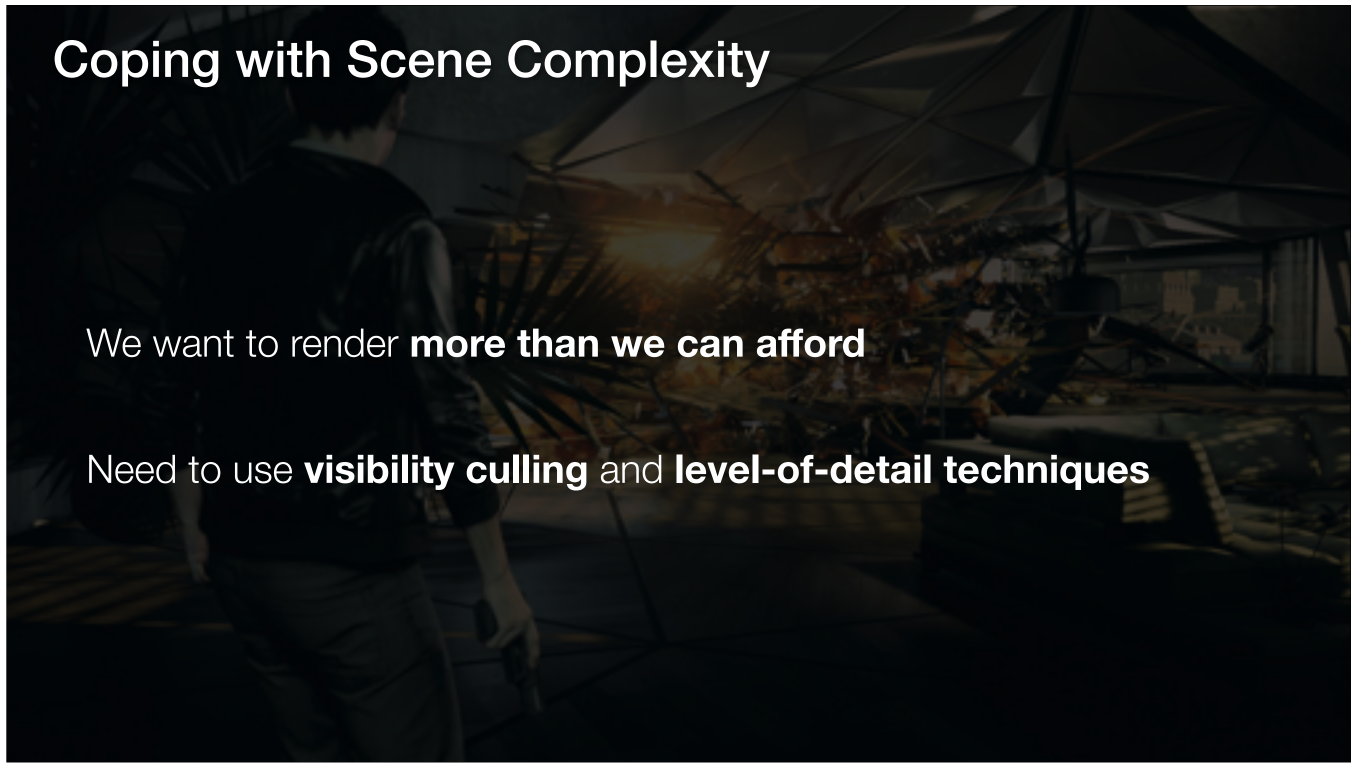
Jaakko Lehtinen
NVIDIA
Aalto University



Coping with Scene Complexity

We want to render **more than we can afford**

In real-time graphics, we often want to render more than we can afford. For example, in Quantum Break, a Xbox One title I’m currently working on, we typically have 10 or 20 million triangles in a scene and a budget of just a few million triangles per frame.



Coping with Scene Complexity

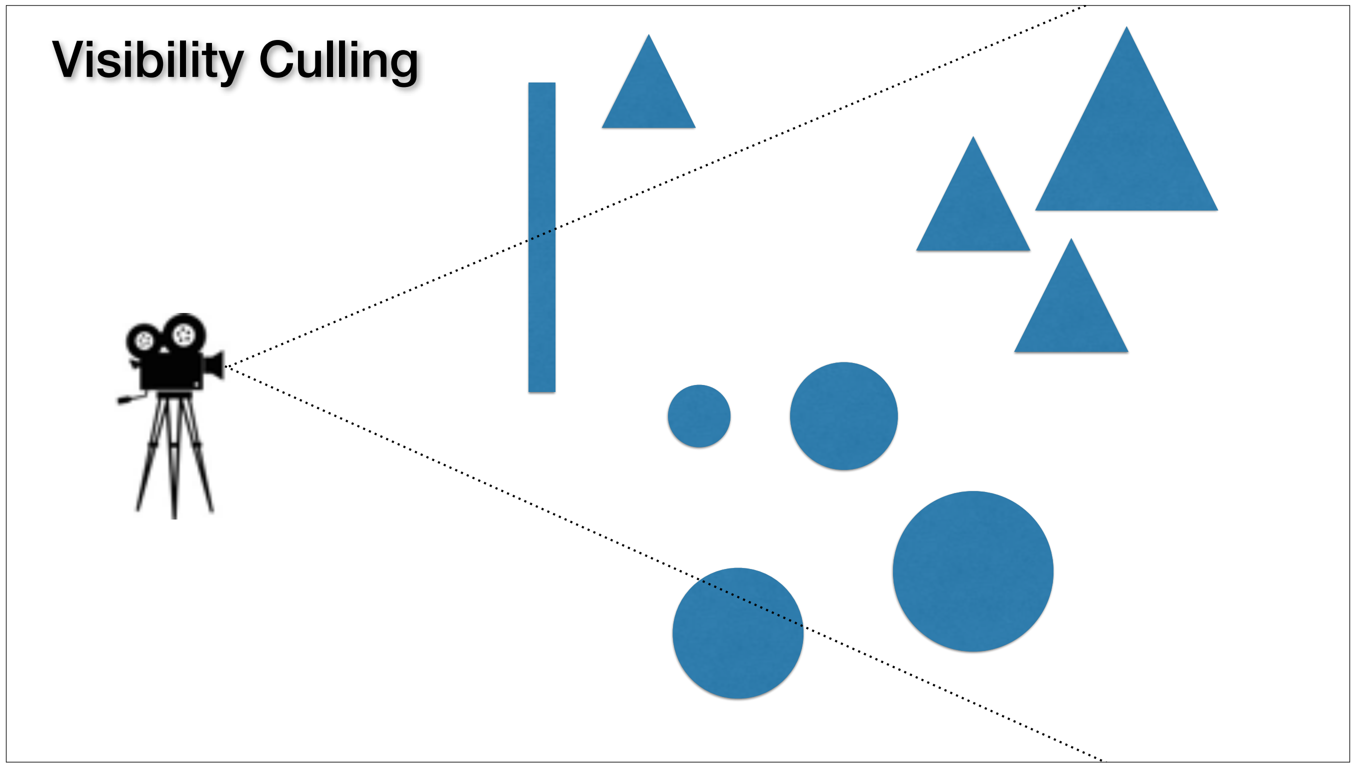
We want to render **more than we can afford**

Need to use **visibility culling** and **level-of-detail techniques**

To cope with this amount of geometry, we need to use visibility culling and level-of-detail techniques, which are orthogonal methods for reducing the number of triangles rendered per frame.

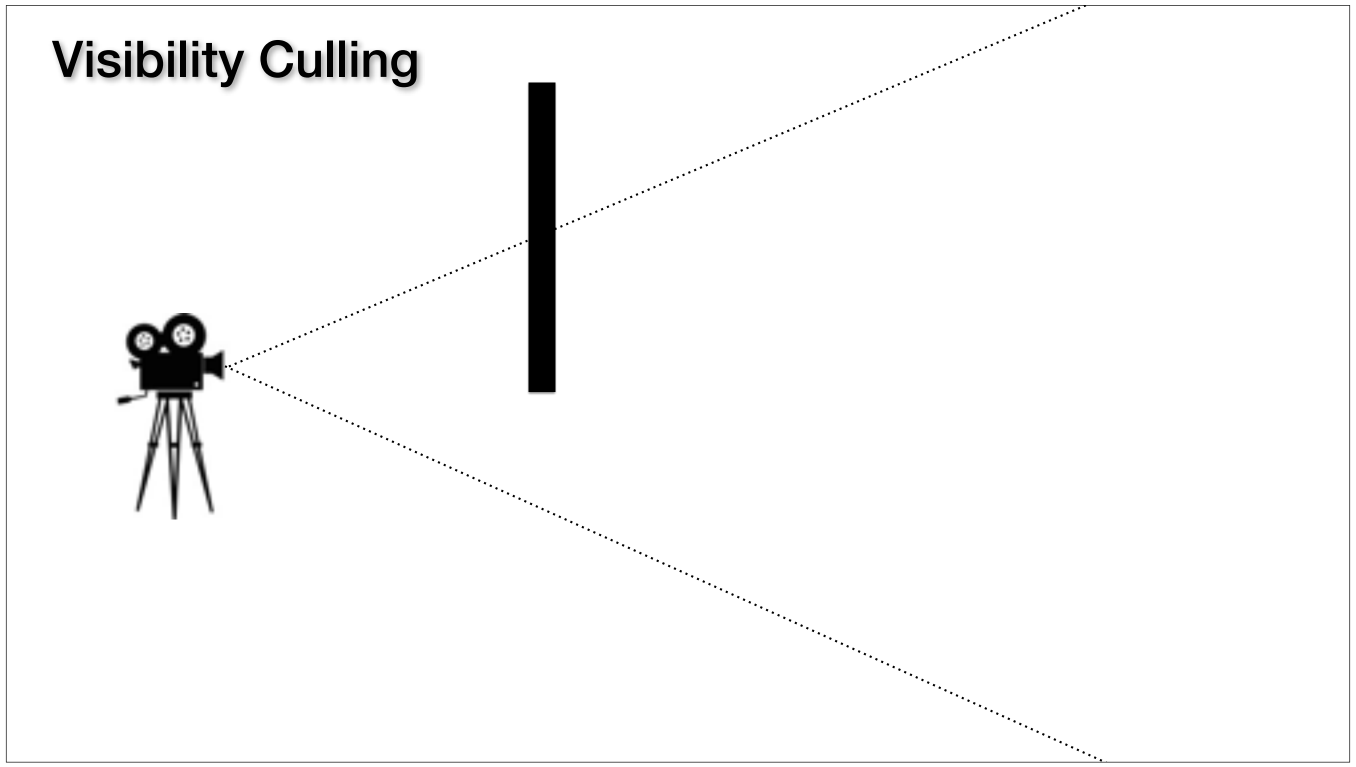
In this talk, we'll consider only visibility culling.

Visibility Culling



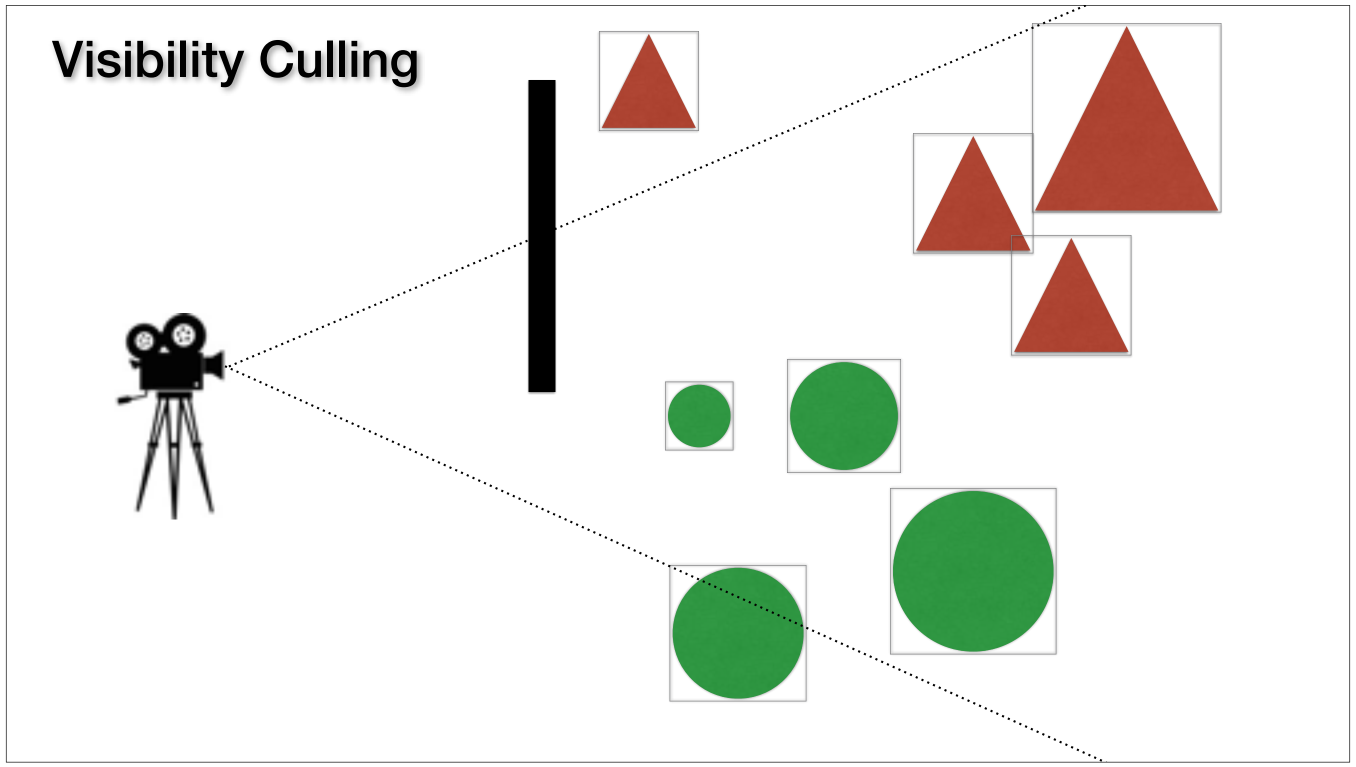
Given a scene and a camera, visibility culling is the process of identifying and removing the hidden parts of the scene from the renderer.

Visibility Culling



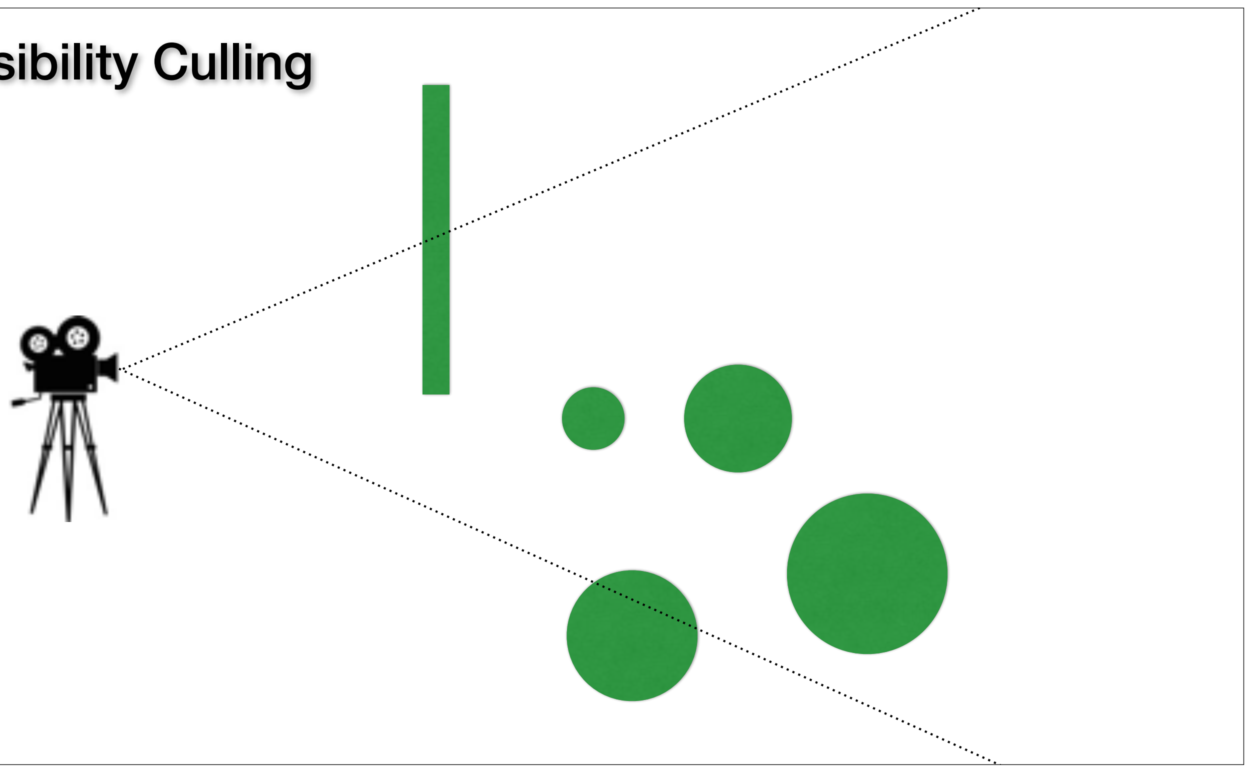
There are many ways to accomplish this but we typically start by rendering the occluders to a depth buffer.

Visibility Culling

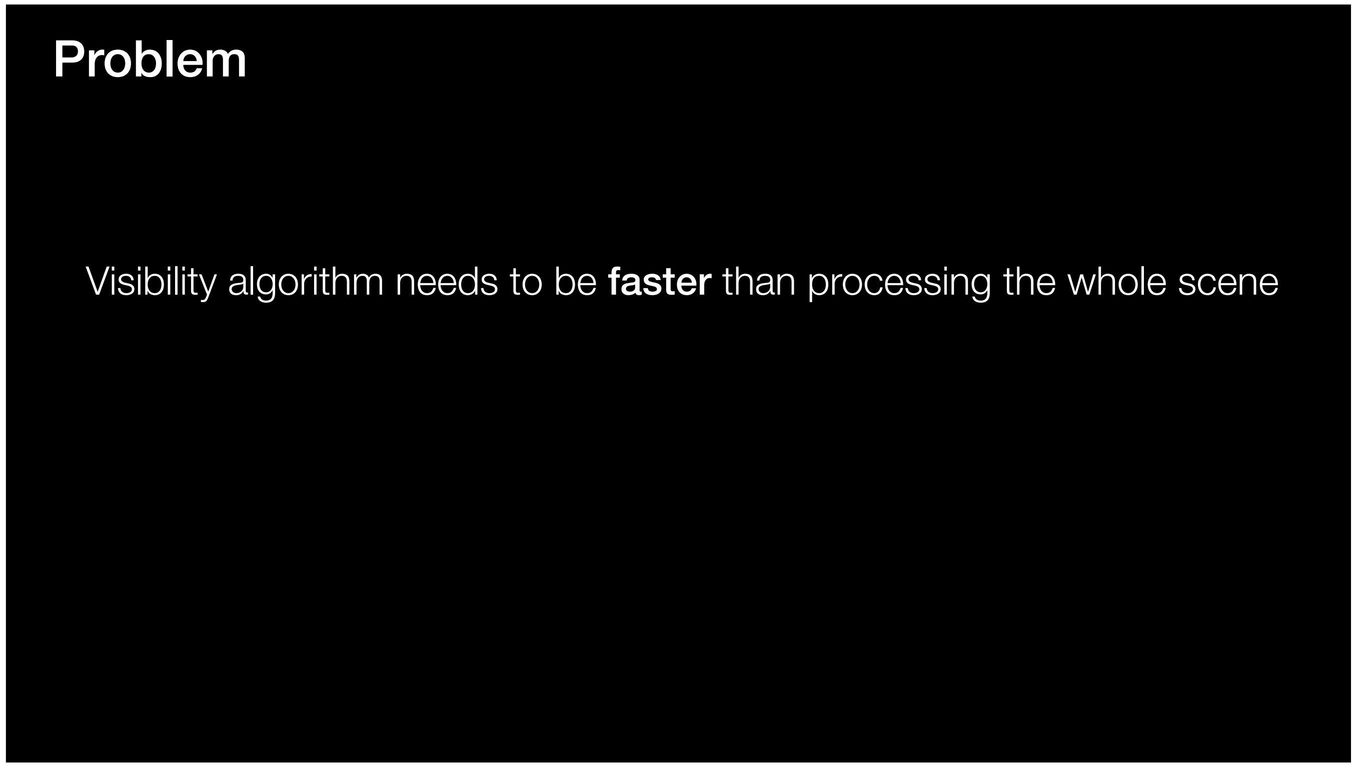


And then we test the bounding box of each scene object against the depth buffer.

Visibility Culling

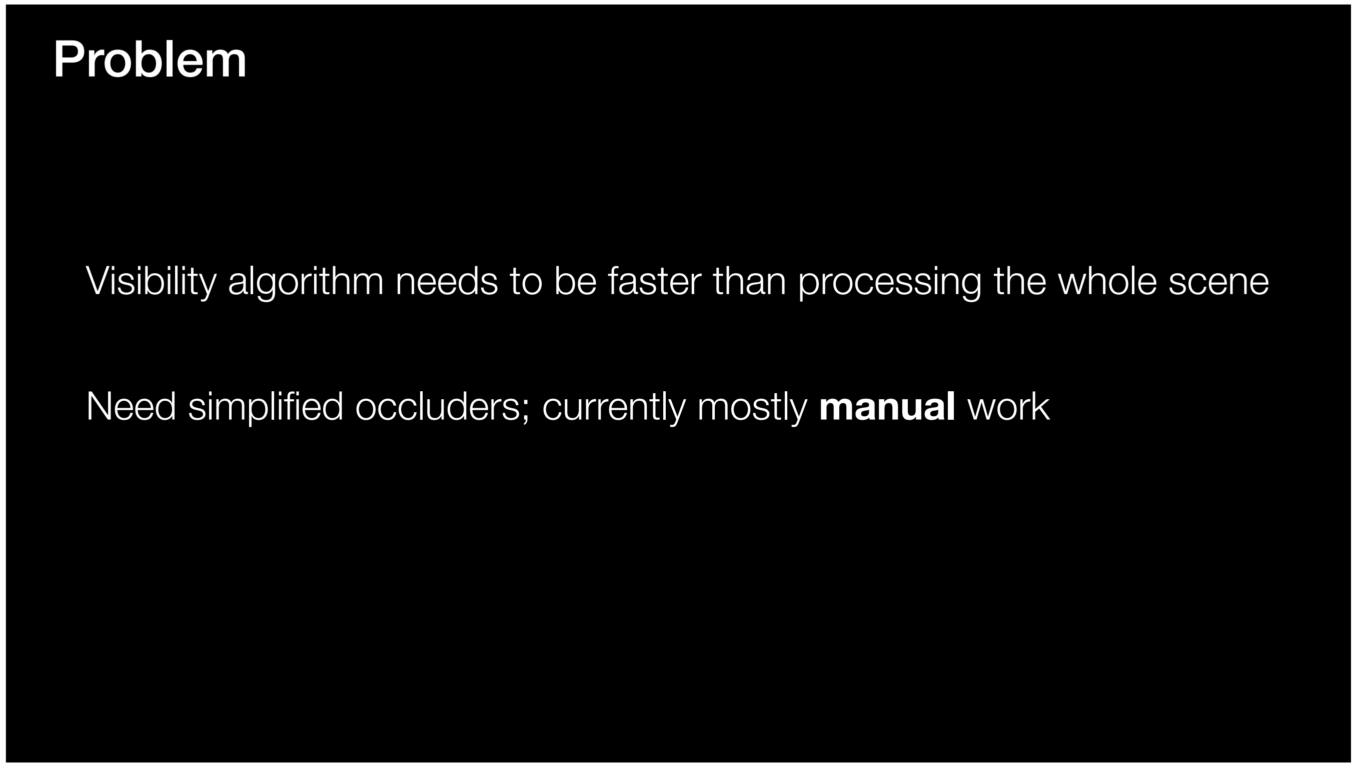


This way we find out which objects are potentially visible and need to be rendered.


Problem

Visibility algorithm needs to be **faster** than processing the whole scene

In order for the visibility culling algorithm to be useful, it must be faster than processing the whole scene.

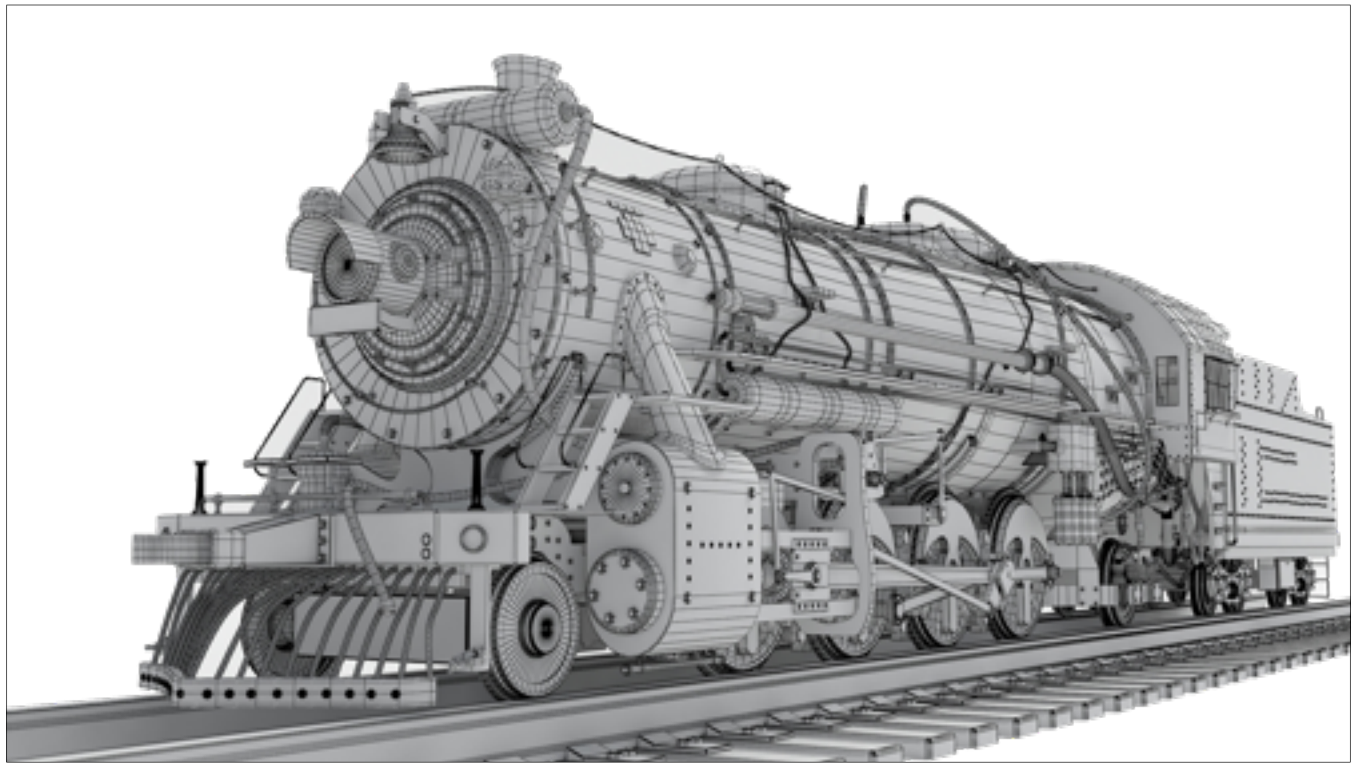

Problem

Visibility algorithm needs to be faster than processing the whole scene

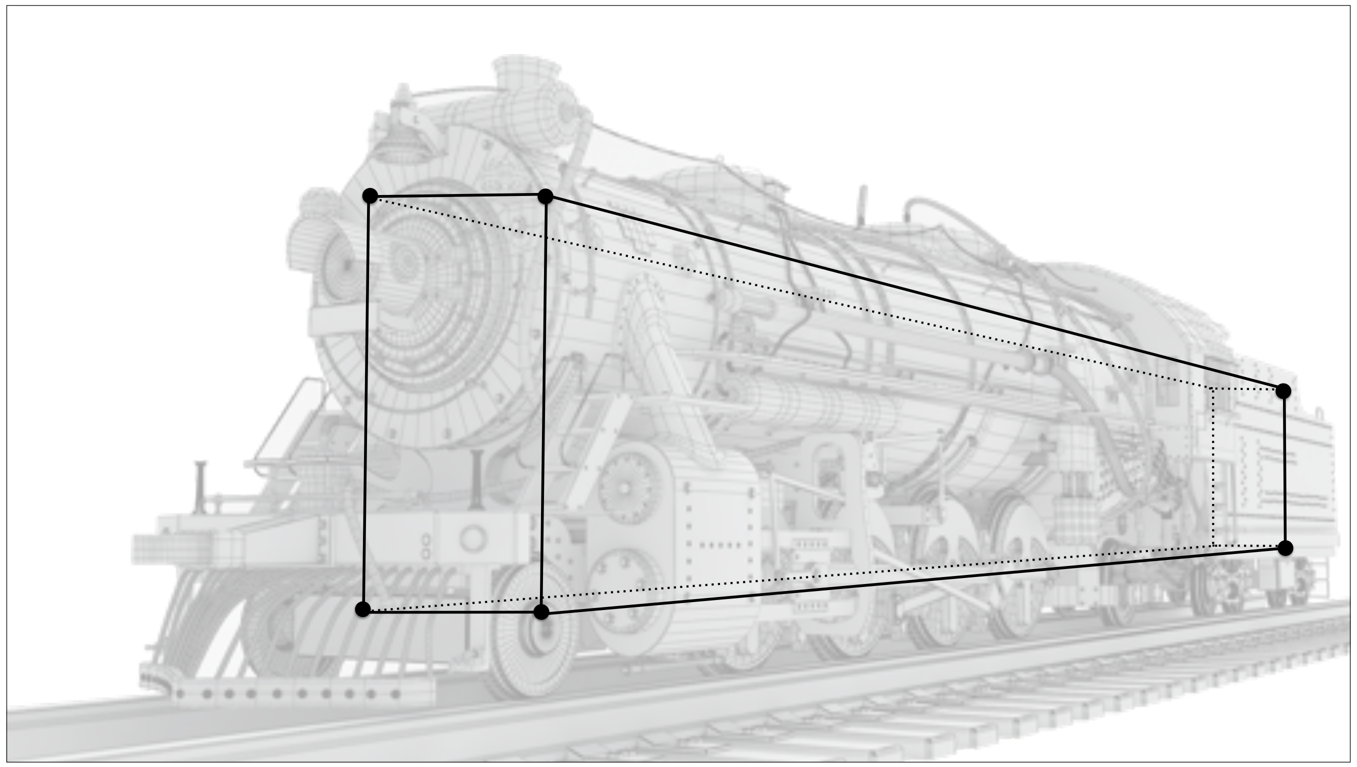
Need simplified occluders; currently mostly **manual** work

This had led to a situation, where artists need to manually create simplified occluders instead of working on the visuals, which is clearly not economical.

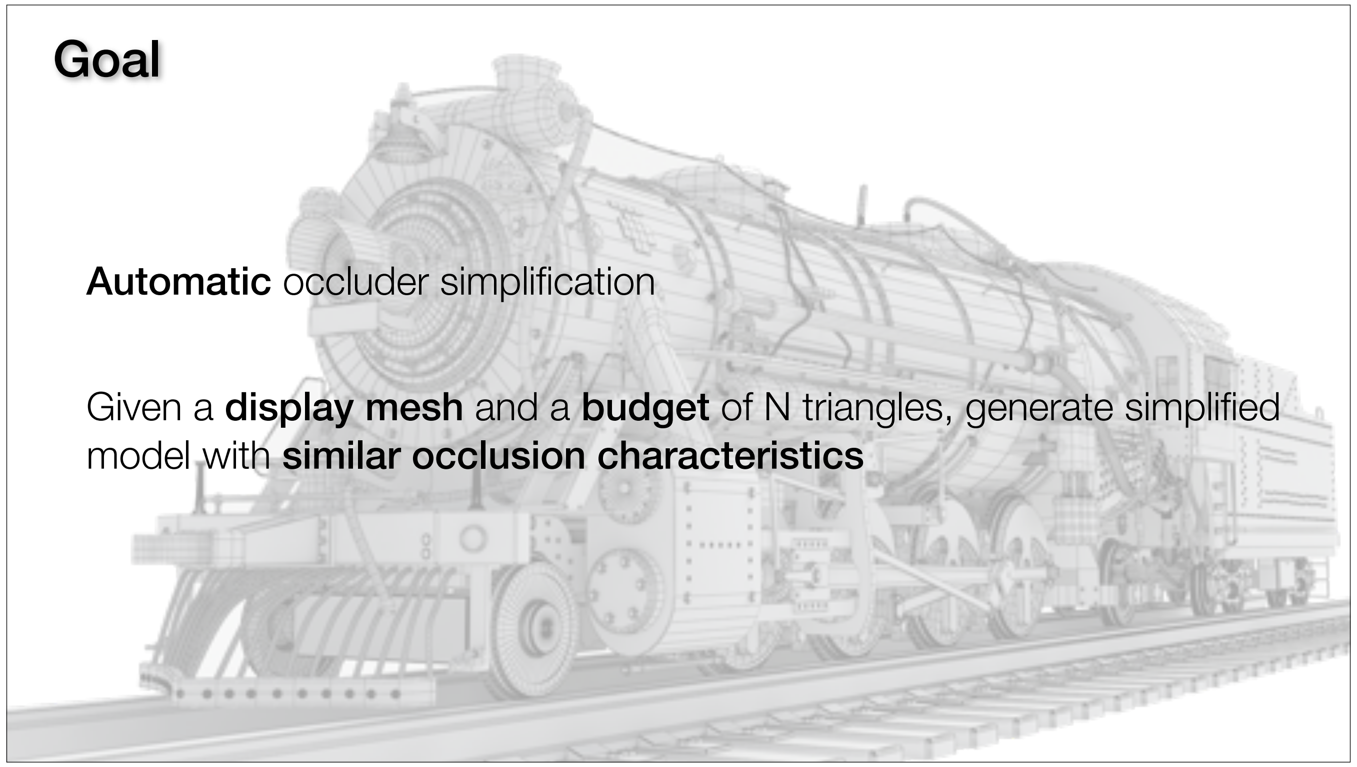
This has been a commonplace practice since the late nineties and hence the lack of research in the area of occluder simplification in general 3D environments, a known problem for more than 15 years, is somewhat surprising.



For example, this is a detailed model of a train that we want to simplify.



And this is the hand-crafted, extremely simplified occluder that you might get from an artist. Our goal should look something like this.



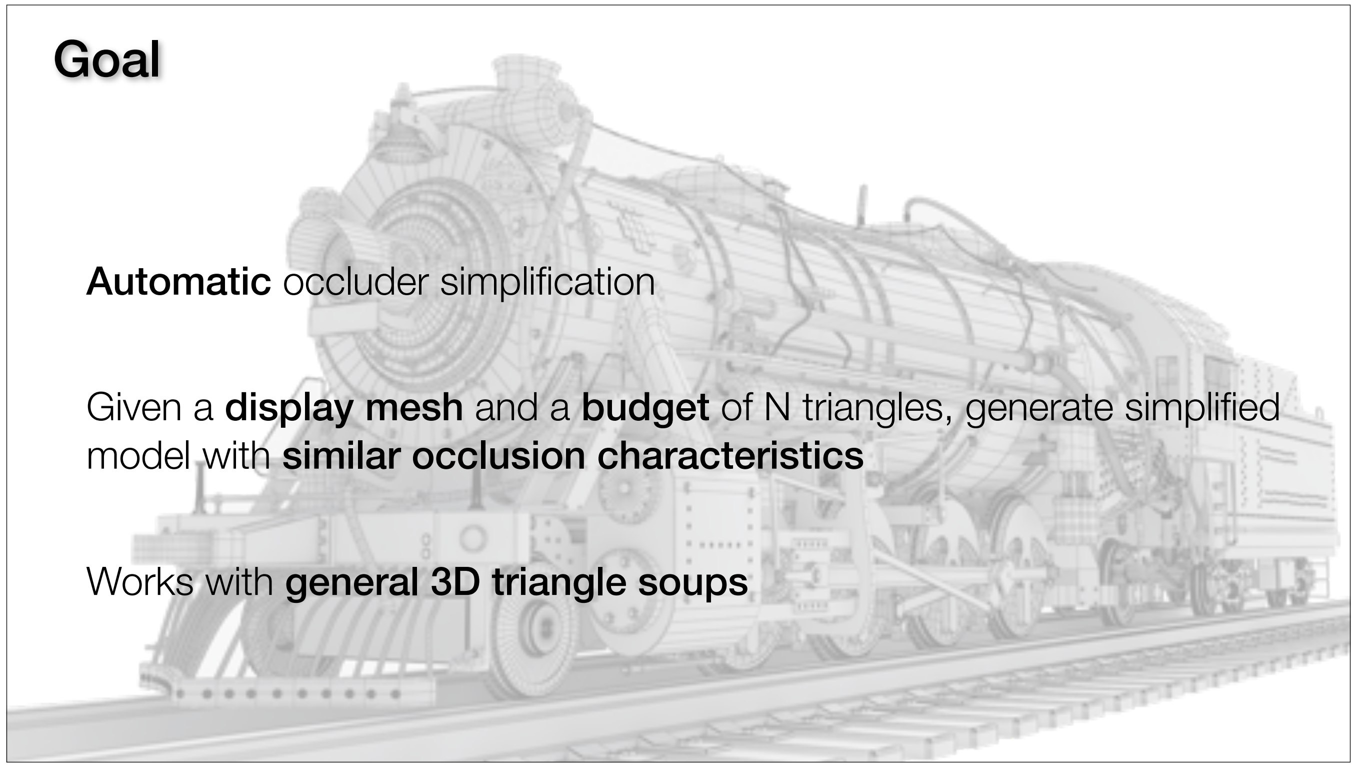
Goal

Automatic occluder simplification

Given a **display mesh** and a **budget** of N triangles, generate simplified model with **similar occlusion characteristics**

So, given all this, our goal is to take a crack at a fully automatic occluder simplification method.

The problem we are solving can be stated as follows: given a display mesh and budget of N triangles, generate a simplified model with similar occlusion characteristics which satisfies the given budget constraints.



Goal

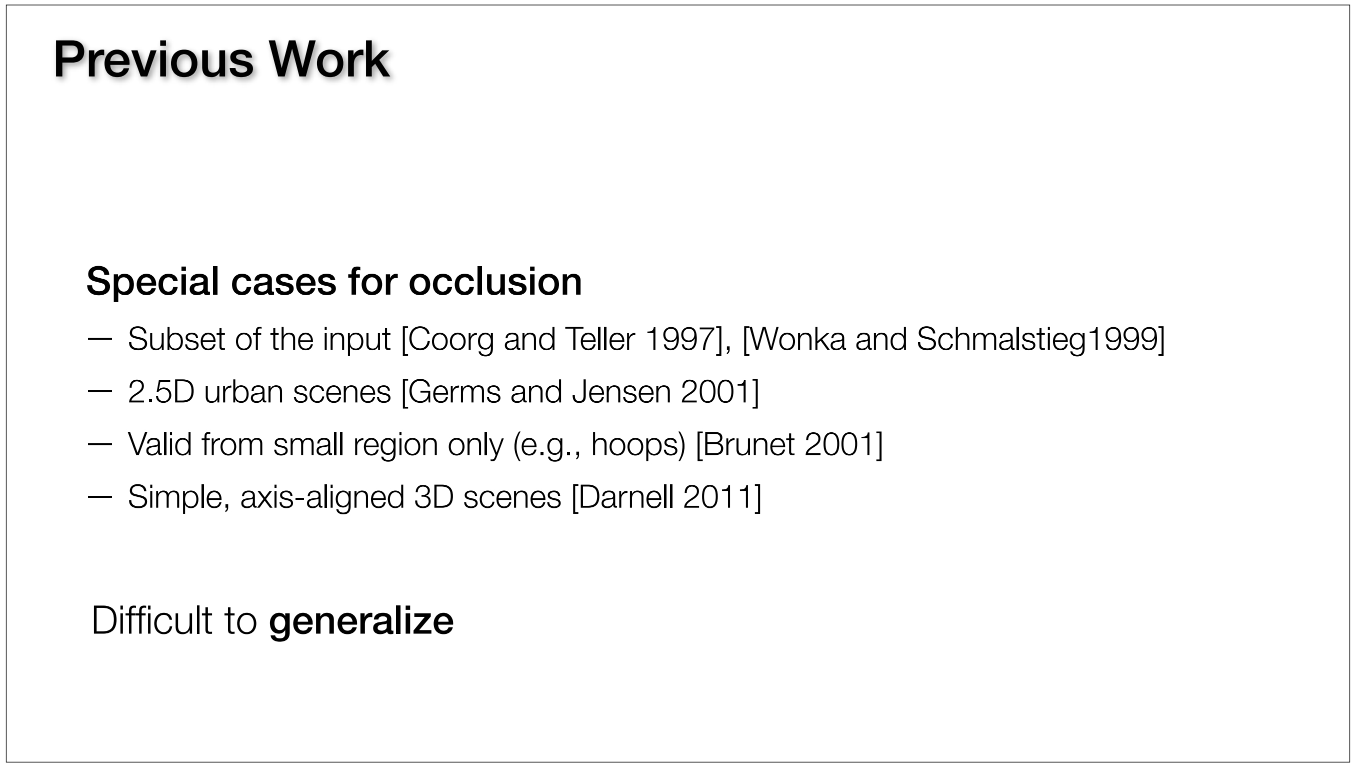
Automatic occluder simplification

Given a **display mesh** and a **budget** of N triangles, generate simplified model with **similar occlusion characteristics**

Works with **general 3D triangle soups**

It should be noted that the problem has a trivial solution: we could just pick a subset of the original triangles, and, in the late nineties, there actually were a few papers on how to do this in a principled manner.

However, we don't assume any special structure of the input geometry. We take a general triangle soup as input, like this train for example, and don't really care about the size, orientation, or even the degeneracy of the input triangles.



Previous Work

Special cases for occlusion

— Subset of the input [Coorg and Teller 1997], [Wonka and Schmalstieg1999]

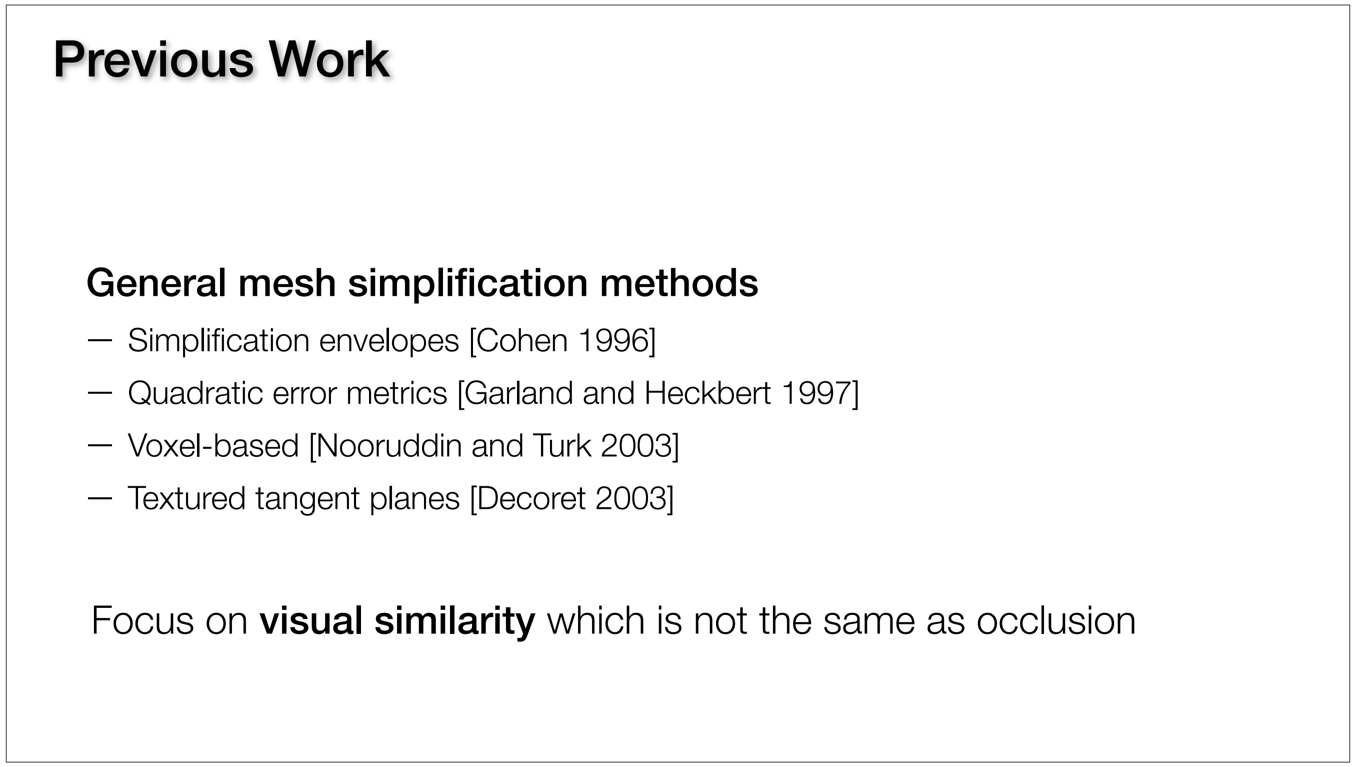
— 2.5D urban scenes [Germs and Jensen 2001]

— Valid from small region only (e.g., hoops) [Brunet 2001]

— Simple, axis-aligned 3D scenes [Darnell 2011]

Difficult to **generalize**

As mentioned, most of the direct previous work has been focused on solving special cases, all of which assume some special structure in the input geometry and therefore they are difficult to extend to handle the general 3D case.



Previous Work

General mesh simplification methods

— Simplification envelopes [Cohen 1996]

— Quadratic error metrics [Garland and Heckbert 1997]

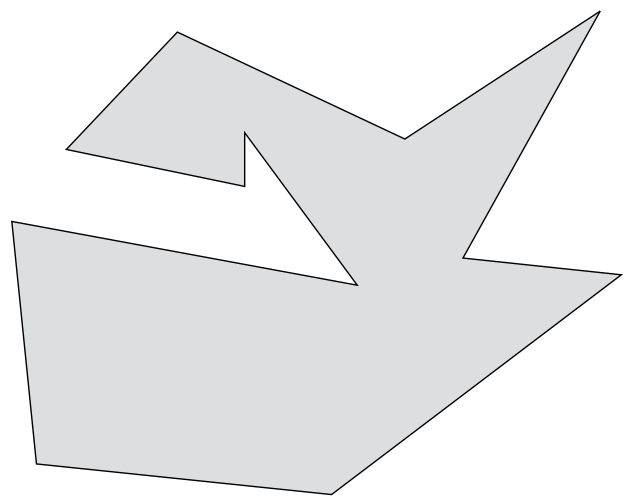
— Voxel-based [Nooruddin and Turk 2003]

— Textured tangent planes [Decoret 2003]

Focus on **visual similarity** which is not the same as occlusion

There exists a large base of work related to general mesh simplification. Methods like Garlands's quadratic error metrics and the tangent plane approximation by Decoret have focused extensively on visual similarity.

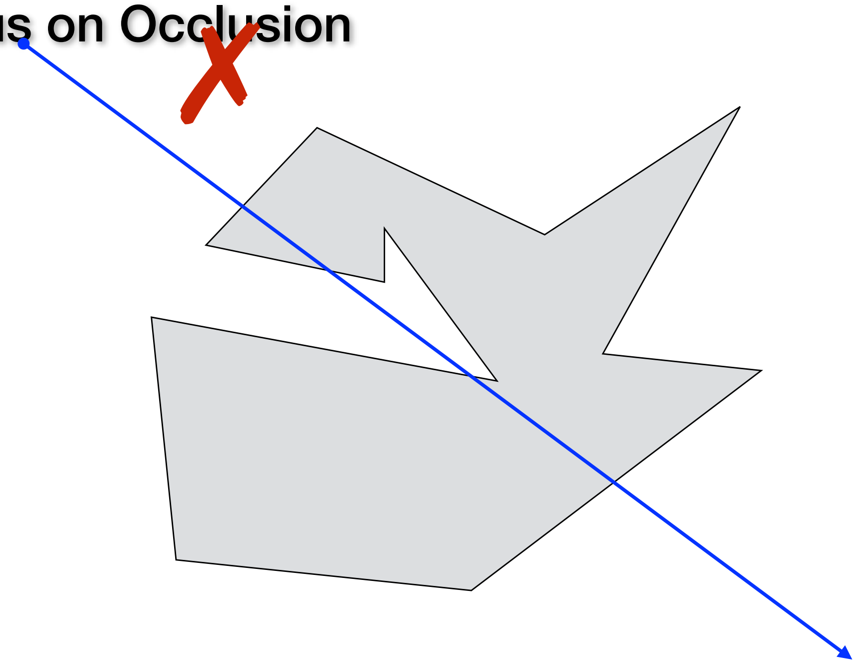
Main Idea: Focus on Occlusion



Instead of trying to match visual similarity, we focus directly on the occlusion properties.

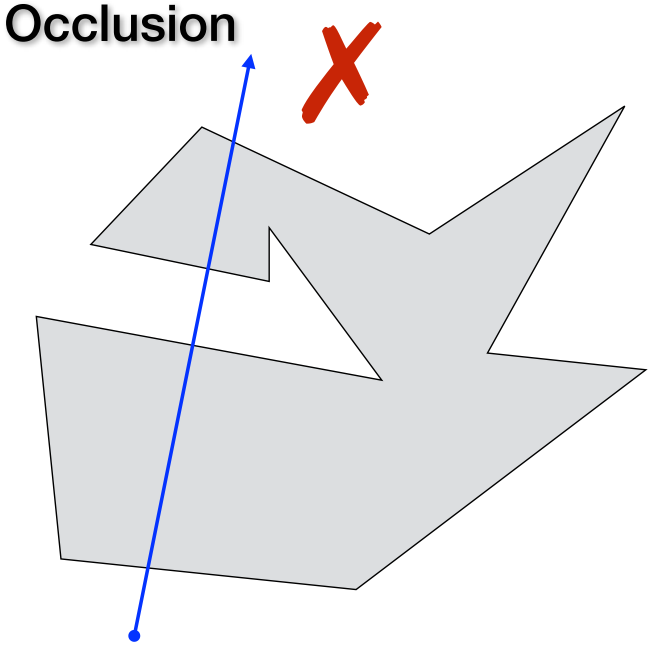
Let’s consider this 2D, closed polygon. We want to find a simplified shape which has the same occlusion characteristics.

Main Idea: Focus on Occlusion

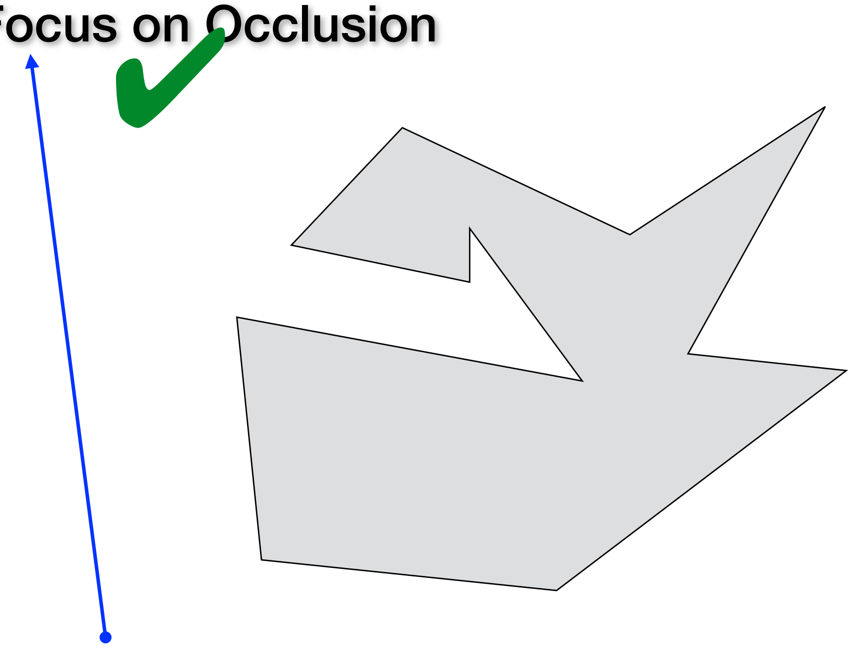


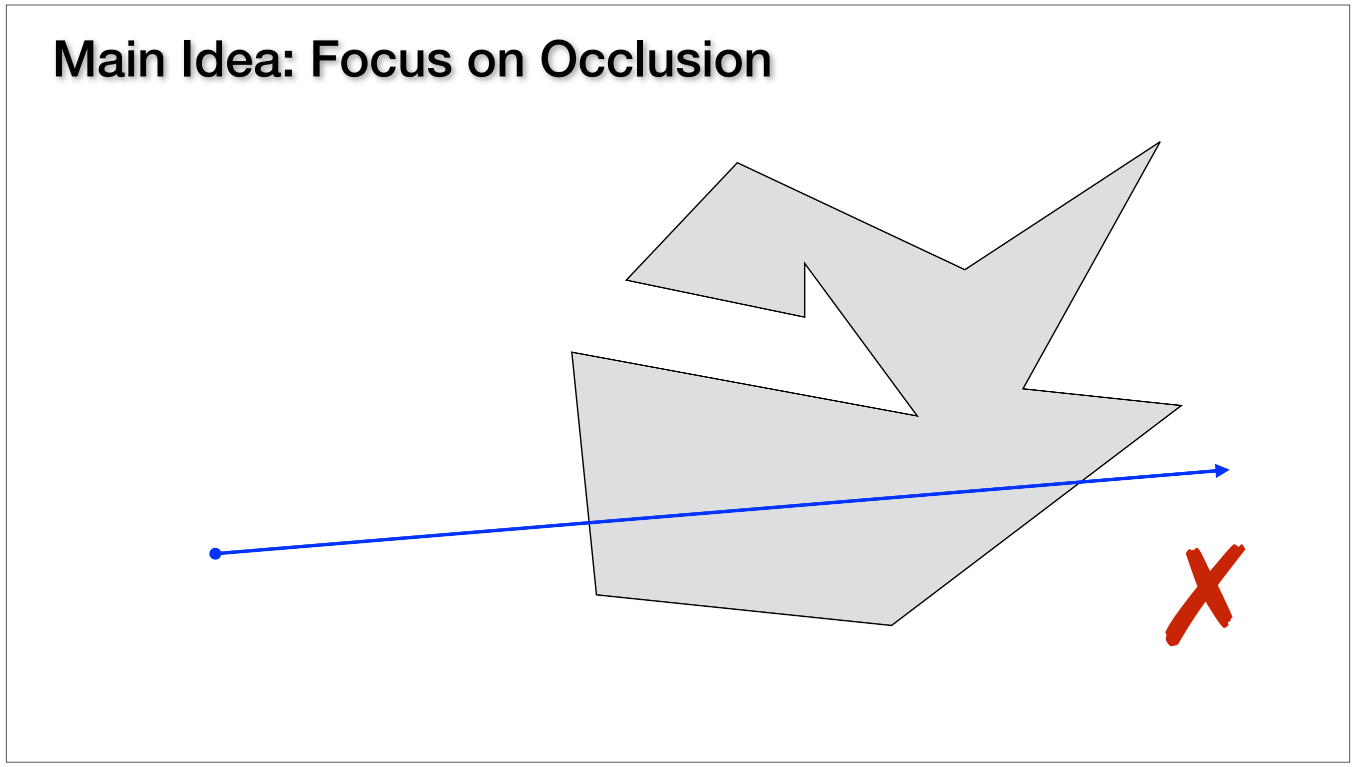
Let’s take a look at a few rays.

Main Idea: Focus on Occlusion

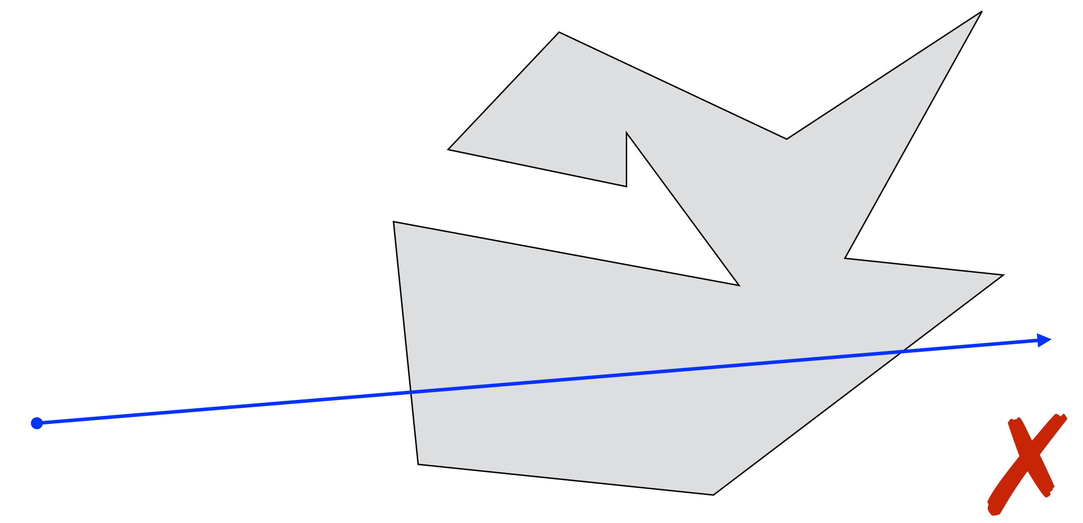


Main Idea: Focus on Occlusion





Main Idea: Focus on Occlusion

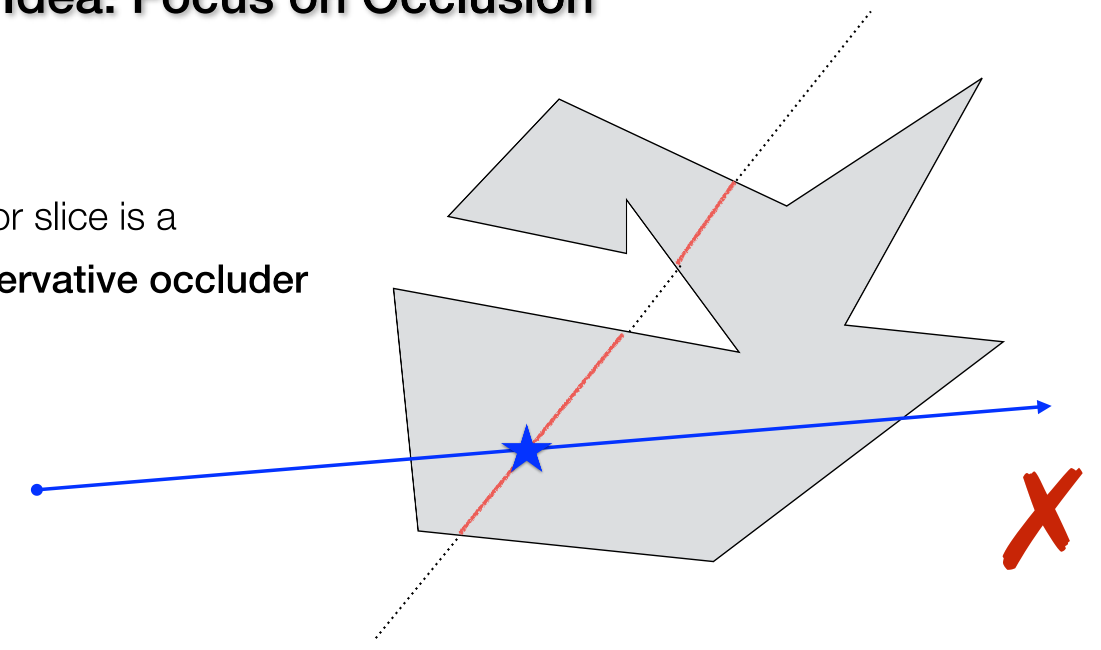


We’d like to have something simpler than the polygon which blocks the same rays.

Main Idea: Focus on Occlusion

Interior slice is a

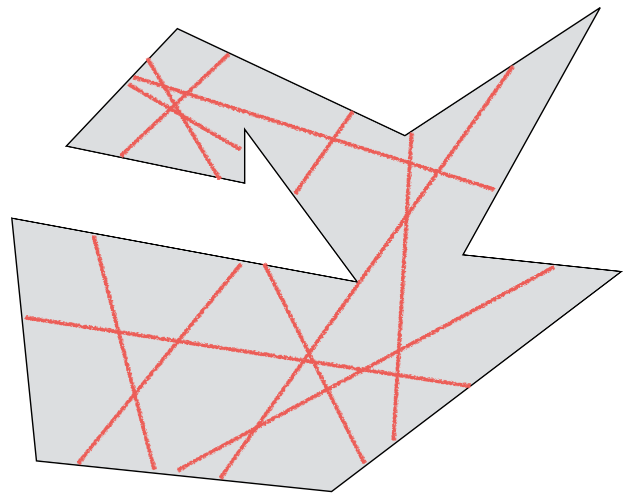
conservative occluder



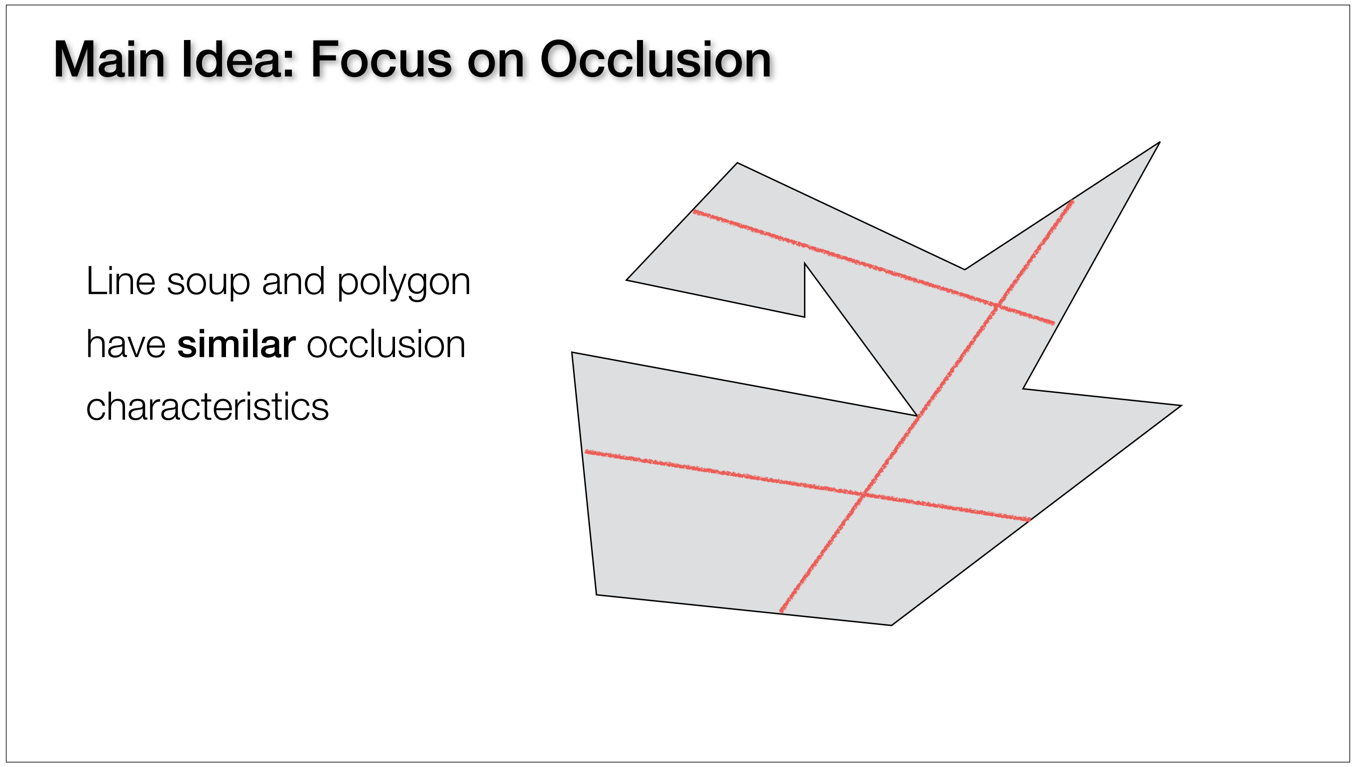
If we cut the polygon with a line we observe that the same ray also intersects the interior line segment.

Main Idea: Focus on Occlusion

Line soup and polygon have **similar** occlusion characteristics

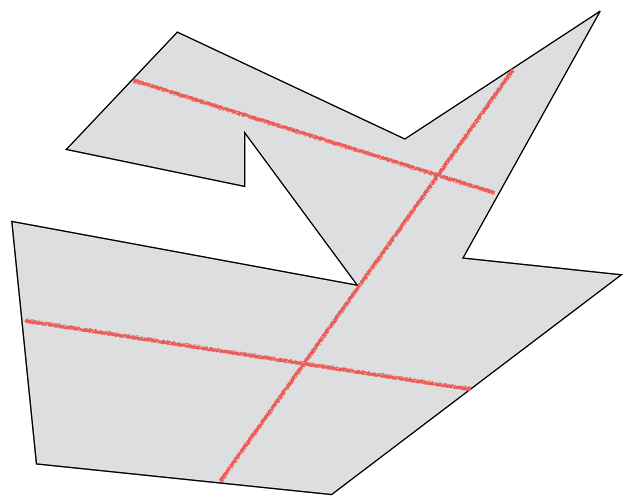


If we cut the polygon with more lines, we can block almost the same set of rays which are blocked by the original polygon: the line soup and the polygon have similar occlusion characteristics.



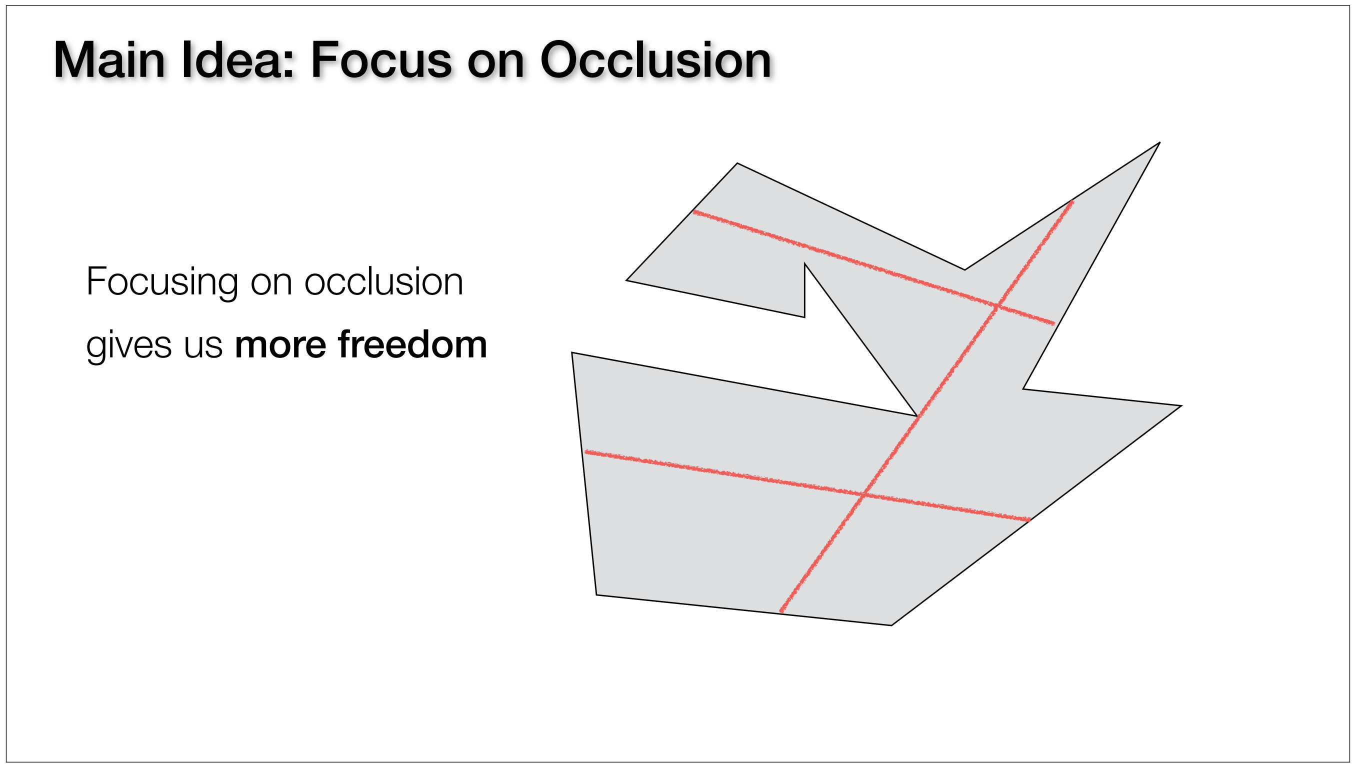
Main Idea: Focus on Occlusion

Line soup and polygon have **similar** occlusion characteristics



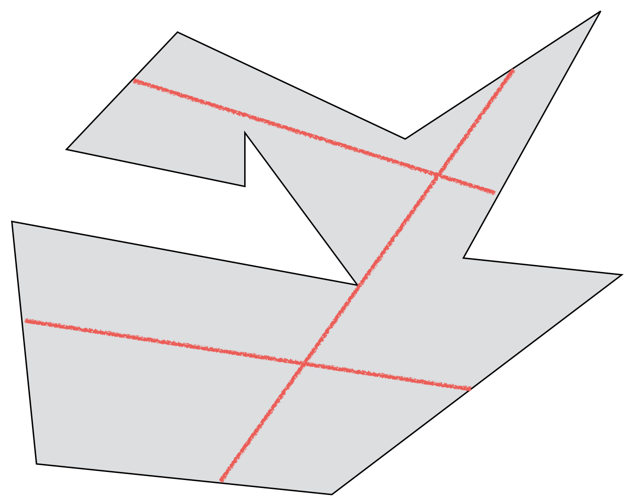
However, most of the lines are redundant, in the sense that they block the same rays, and by carefully choosing which cuts to use, we can obtain a simplified occluder.

Note that this would be impossible with a general mesh simplification method.



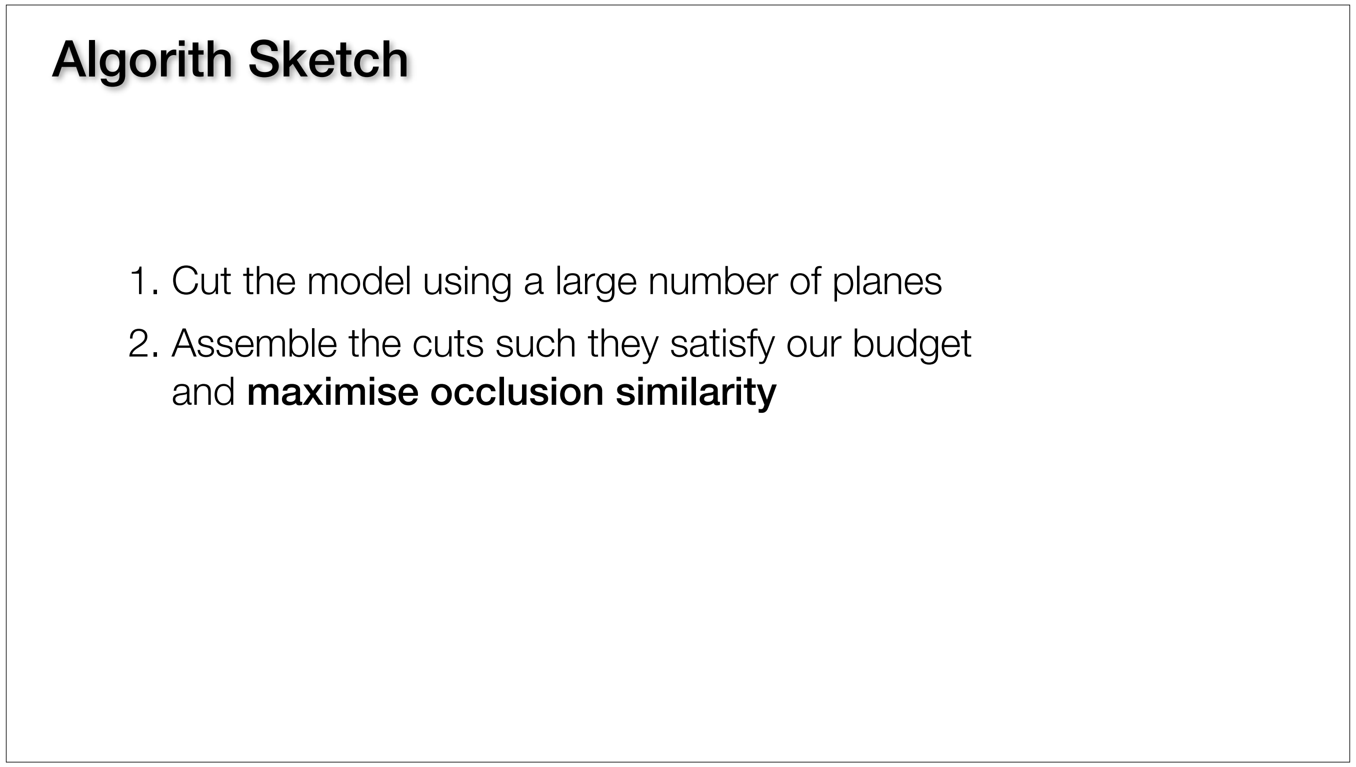
Main Idea: Focus on Occlusion

Focusing on occlusion gives us **more freedom**



By focusing on occlusion instead of visual similarity, we have relaxed the problem of surface based simplification by allowing plane cuts, which gives us more degrees of freedom to exploit.

And the same 2D principles generalise directly into 3D.



Algorith Sketch

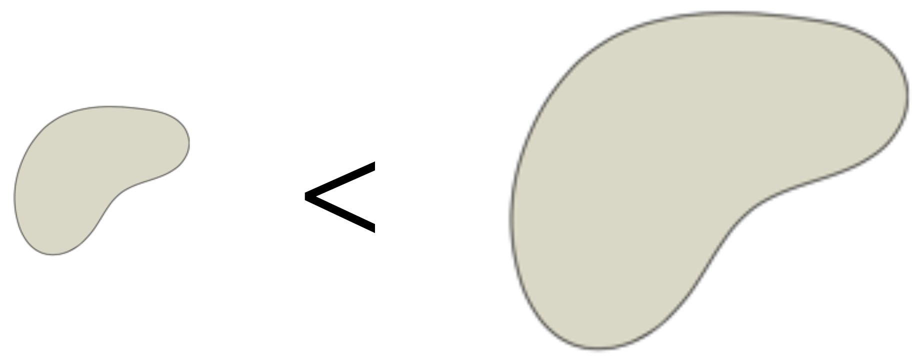
1. Cut the model using a large number of planes
2. Assemble the cuts such they satisfy our budget and **maximise occlusion similarity**

This can be turned into an algorithm as follows. First we cut the model with a large number of planes and then we assemble the plane cuts such that they satisfy the budget constraint and maximise the occlusion similarity.

How to Quantify Occlusion?

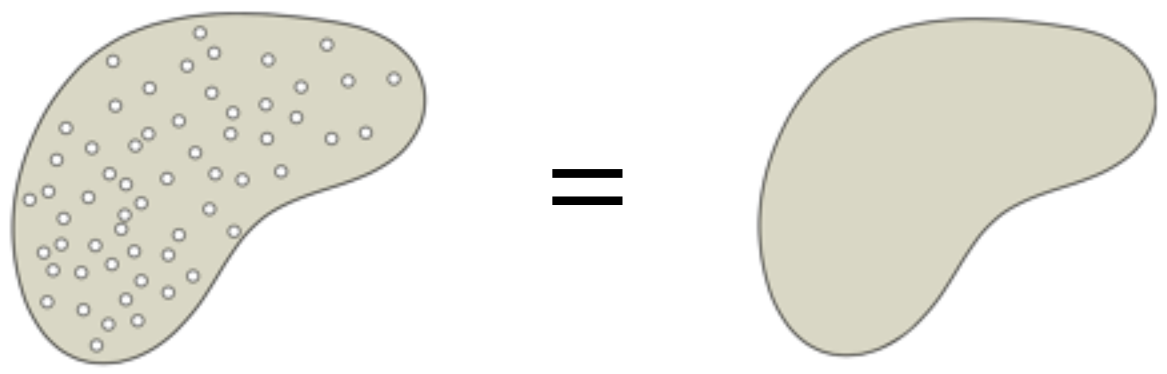
But there is a big question: how do we measure occlusion?

Surface Area?



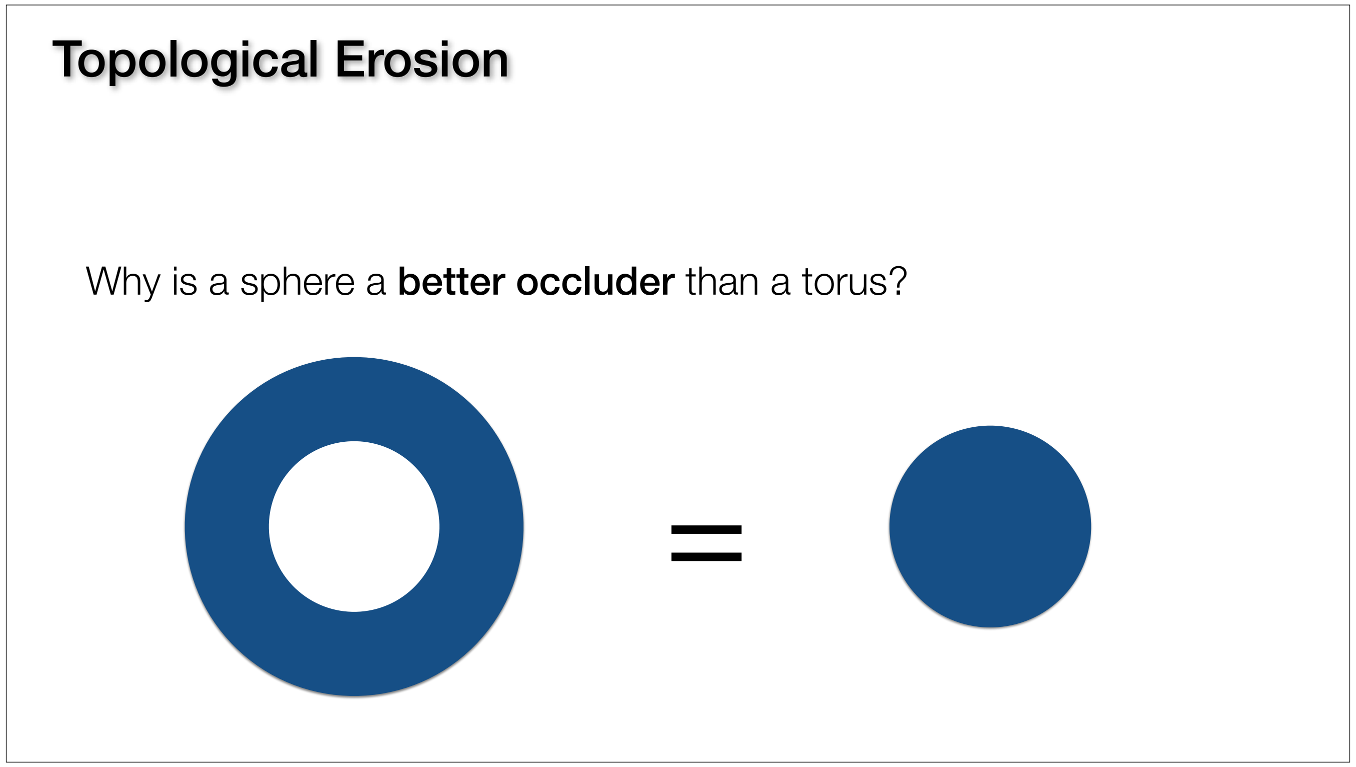
If we are looking at individual rays, then surface area is the right answer: bigger surface area means that more rays are blocked.

Surface Area?



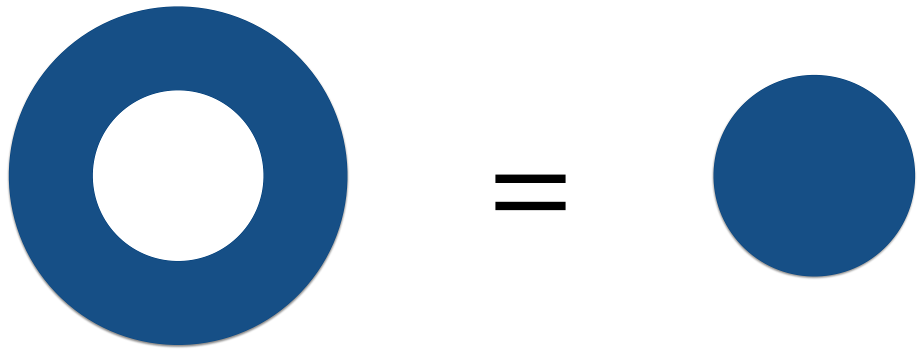
However, in this case both objects have equal surface area, even thought the left one contains a lot of holes and is useless as an occluder because we can see through it.

It’s seems clear that surface area is not enough.



Topological Erosion

Why is a sphere a **better occluder** than a torus?

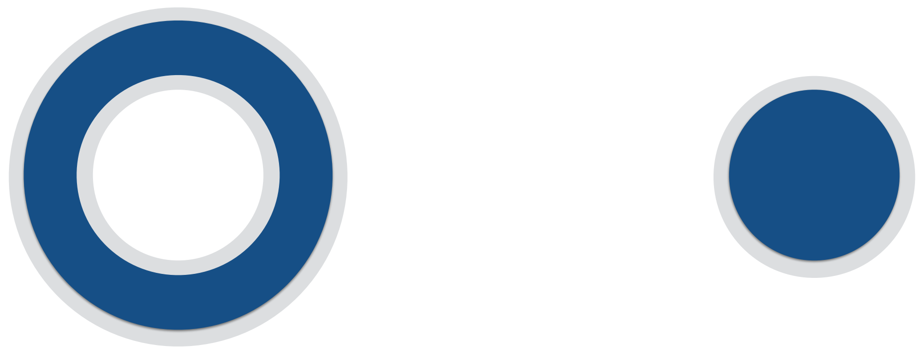


To understand what's going on with all these holes, let's consider a 2D projection of a torus which has just a one hole and a sphere which has no holes at all. Note that the shapes have equal surface area just like in the previous example.

If we perform a topological erosion over both shapes,

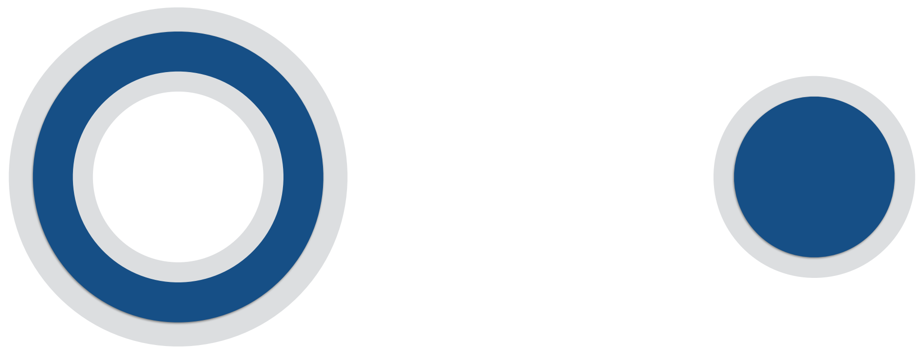
Topological Erosion

Why is a sphere a **better occluder** than a torus?



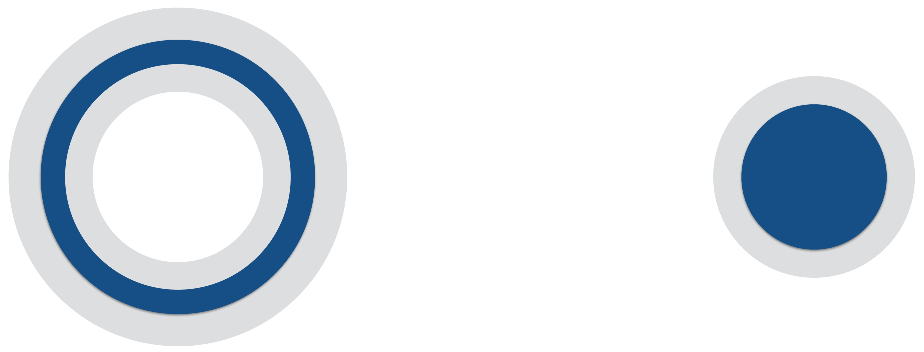
Topological Erosion

Why is a sphere a **better occluder** than a torus?



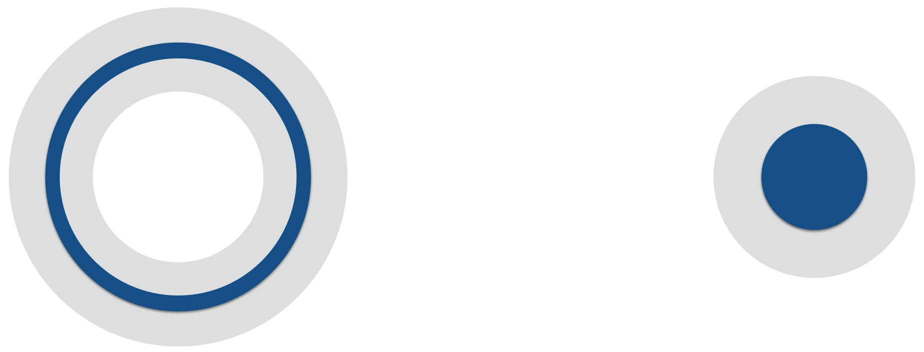
Topological Erosion

Why is a sphere a **better occluder** than a torus?



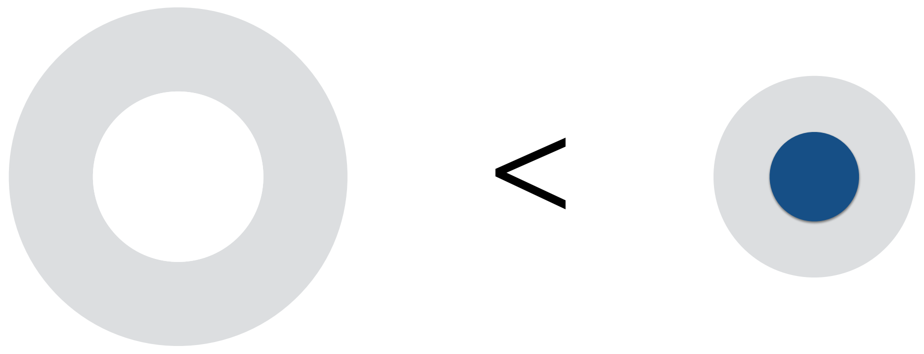
Topological Erosion

Why is a sphere a **better occluder** than a torus?



Topological Erosion

Why is a sphere a **better occluder** than a torus?



We see that the erosion process eats the donut but just shrinks the sphere.

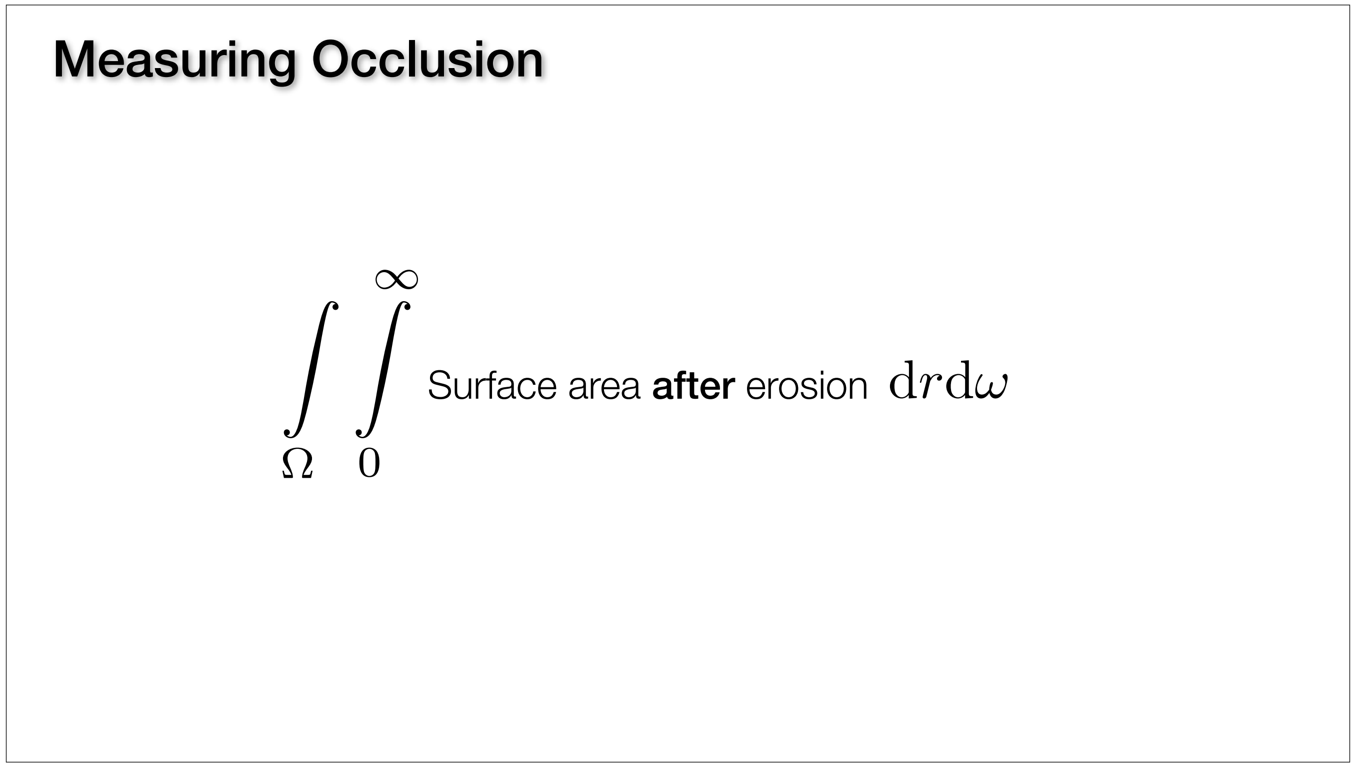
This is a clue that we should look at the surface area after the erosion.

Measuring Occlusion

Surface area **after** erosion

And that’s the basic idea of how we measure occlusion.

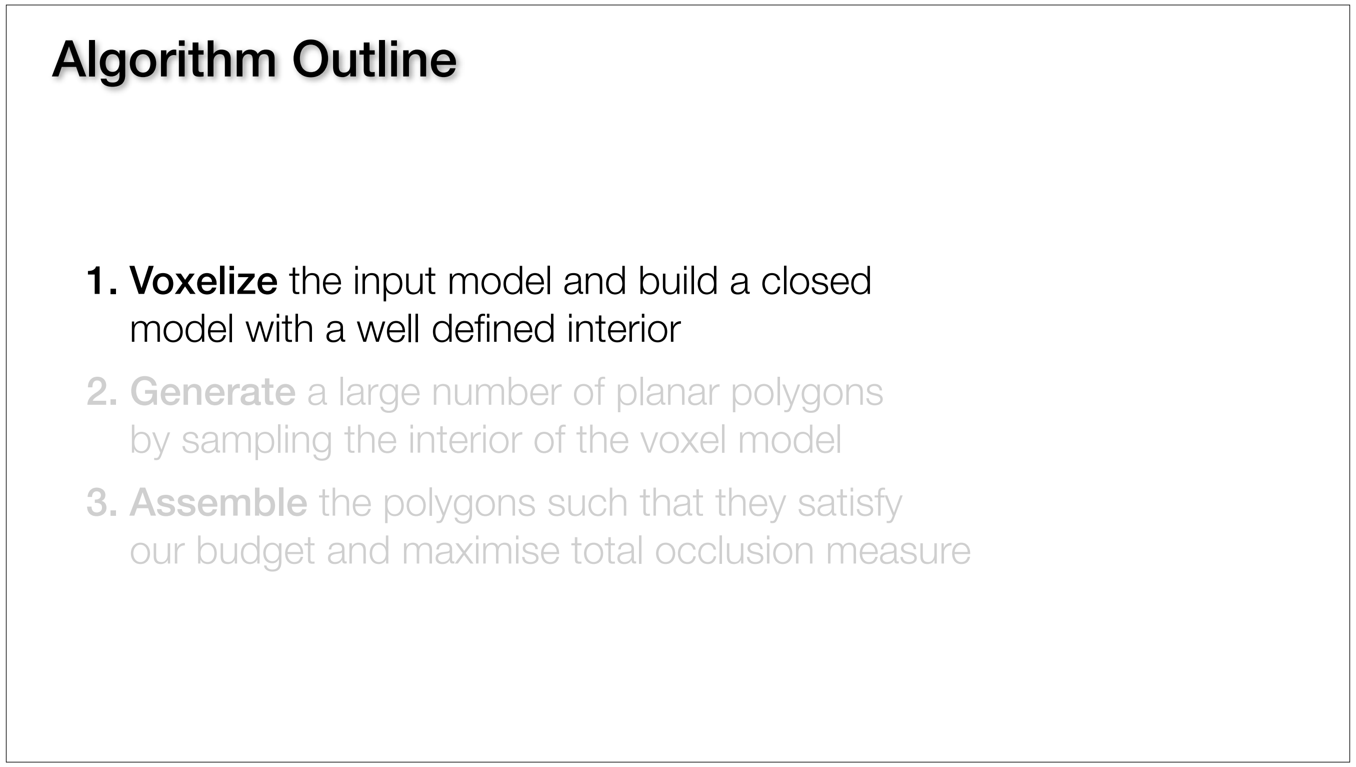
For a fixed direction and erosion radius, we do an orthogonal projection into 2D, just like a shadow map, and perform topological erosion and compute the surface area after the erosion.



Measuring Occlusion

$$\int_{\Omega} \int_{0}^{\infty} \text{Surface area } \textbf{after} \text{ erosion } \mathrm{d}r\,\mathrm{d}\omega$$

And then we integrate this over all radii and directions to obtain a the total occlusion measure.

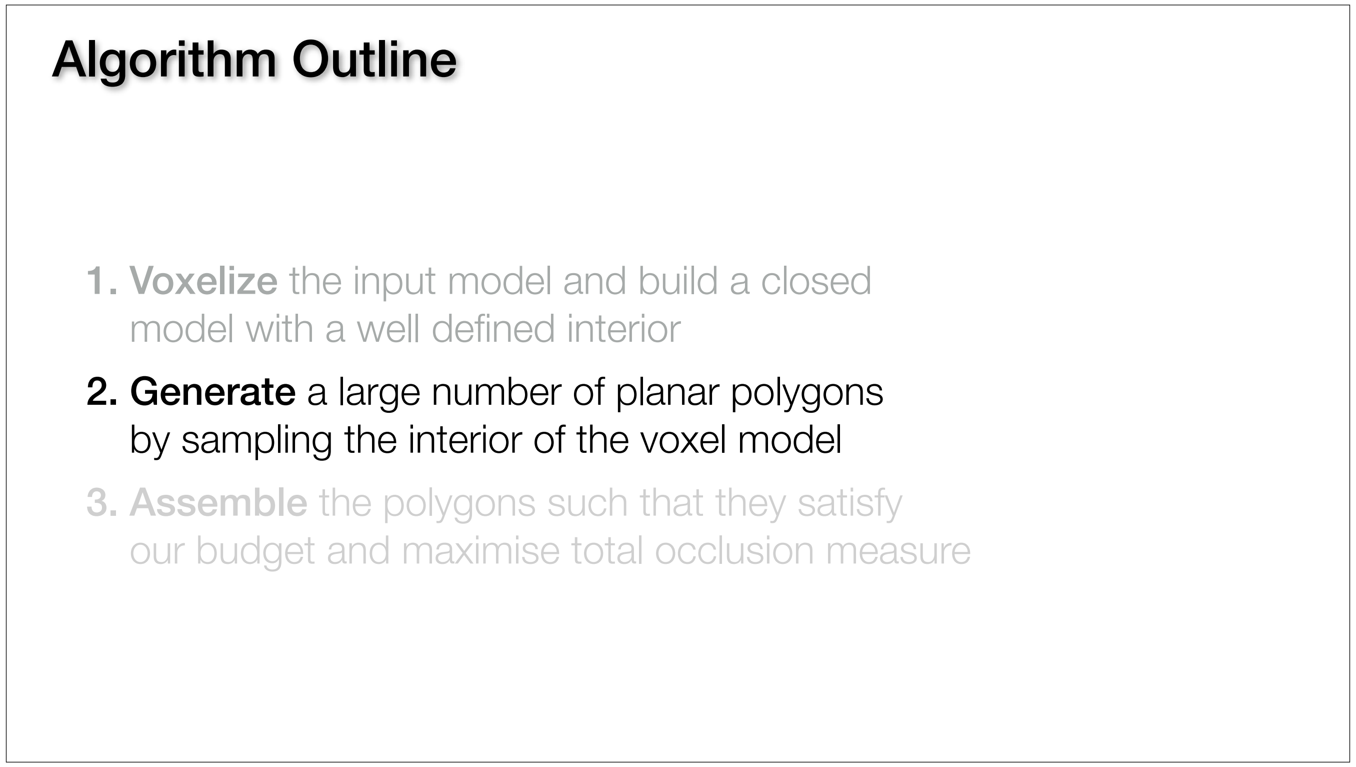


Algorithm Outline

1. **Voxelize** the input model and build a closed model with a well defined interior
2. **Generate** a large number of planar polygons by sampling the interior of the voxel model
3. **Assemble** the polygons such that they satisfy our budget and maximise total occlusion measure

So now that we have a practical and a principled way to measure occlusion, we can finish fleshing out the algorithm sketch we had before.

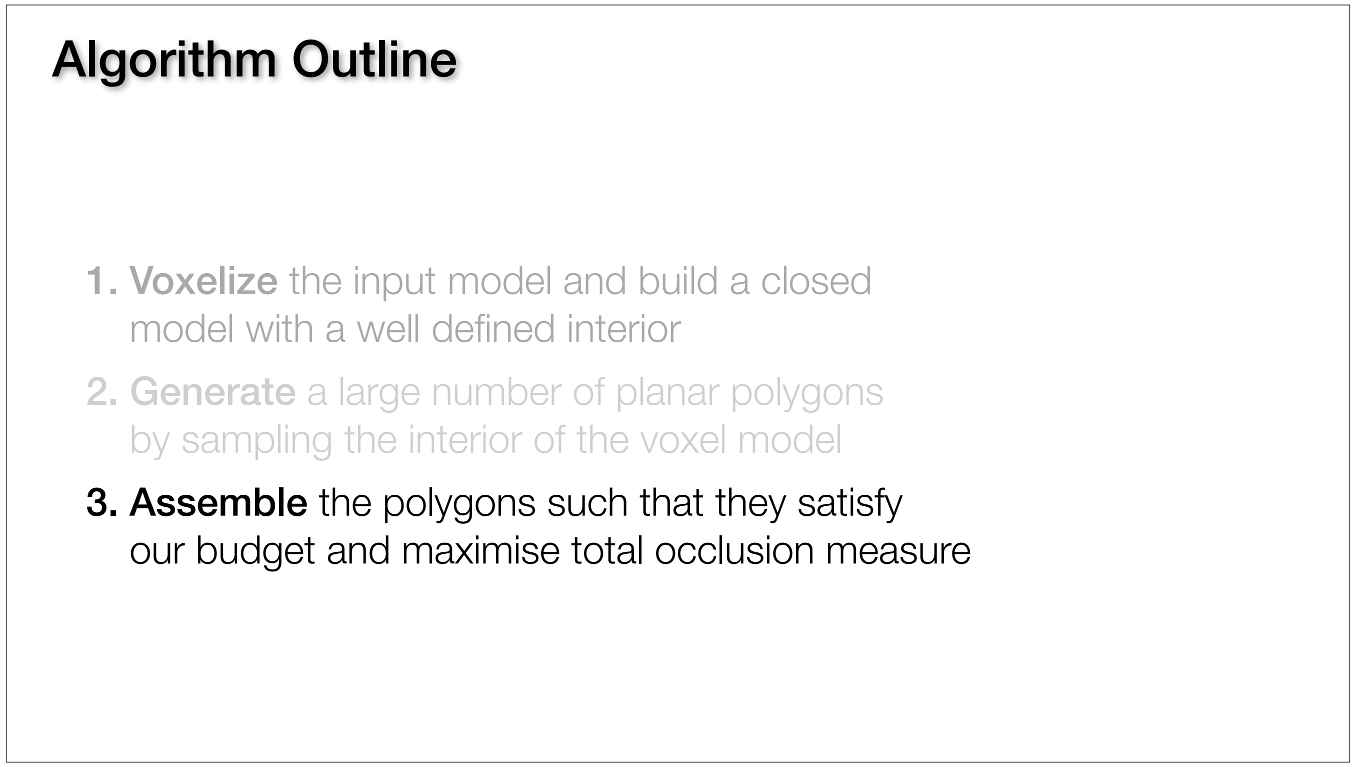
We start by voxelizing the input geometry, which guarantees that we have a well defined interior so we can form valid plane cuts.



Algorithm Outline

1. **Voxelize** the input model and build a closed model with a well defined interior
2. **Generate** a large number of planar polygons by sampling the interior of the voxel model
3. **Assemble** the polygons such that they satisfy our budget and maximise total occlusion measure

Then we generate a large number of candidate polygons by cutting the voxel mesh with random planes.



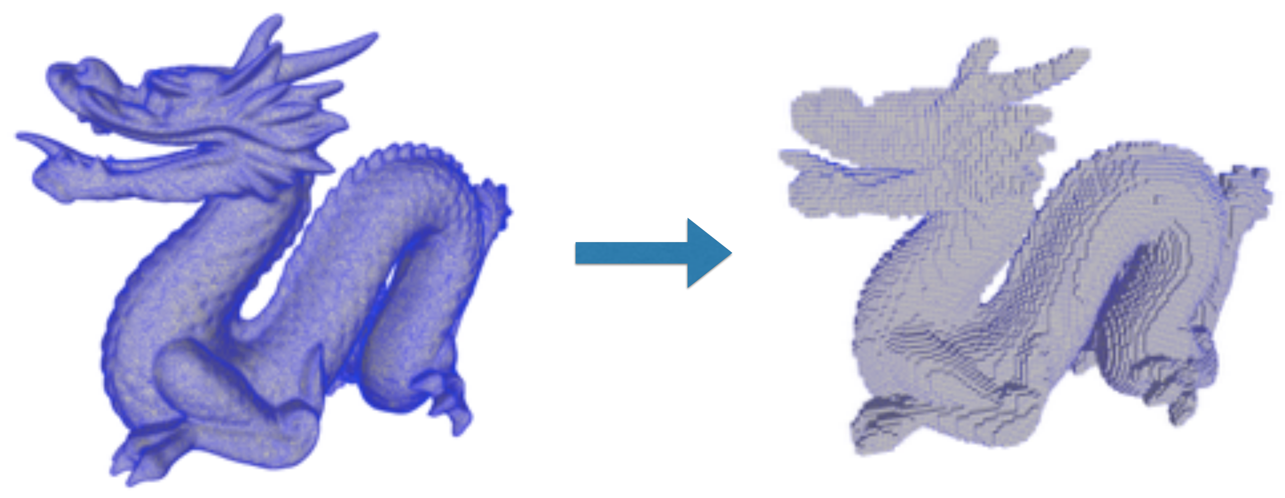
Algorithm Outline

1. **Voxelize** the input model and build a closed model with a well defined interior
2. **Generate** a large number of planar polygons by sampling the interior of the voxel model
3. **Assemble** the polygons such that they satisfy our budget and maximise total occlusion measure

Finally, we assemble the polygons together to obtain the simplified occluder.

Let’s walk through each of the three parts in more detail.

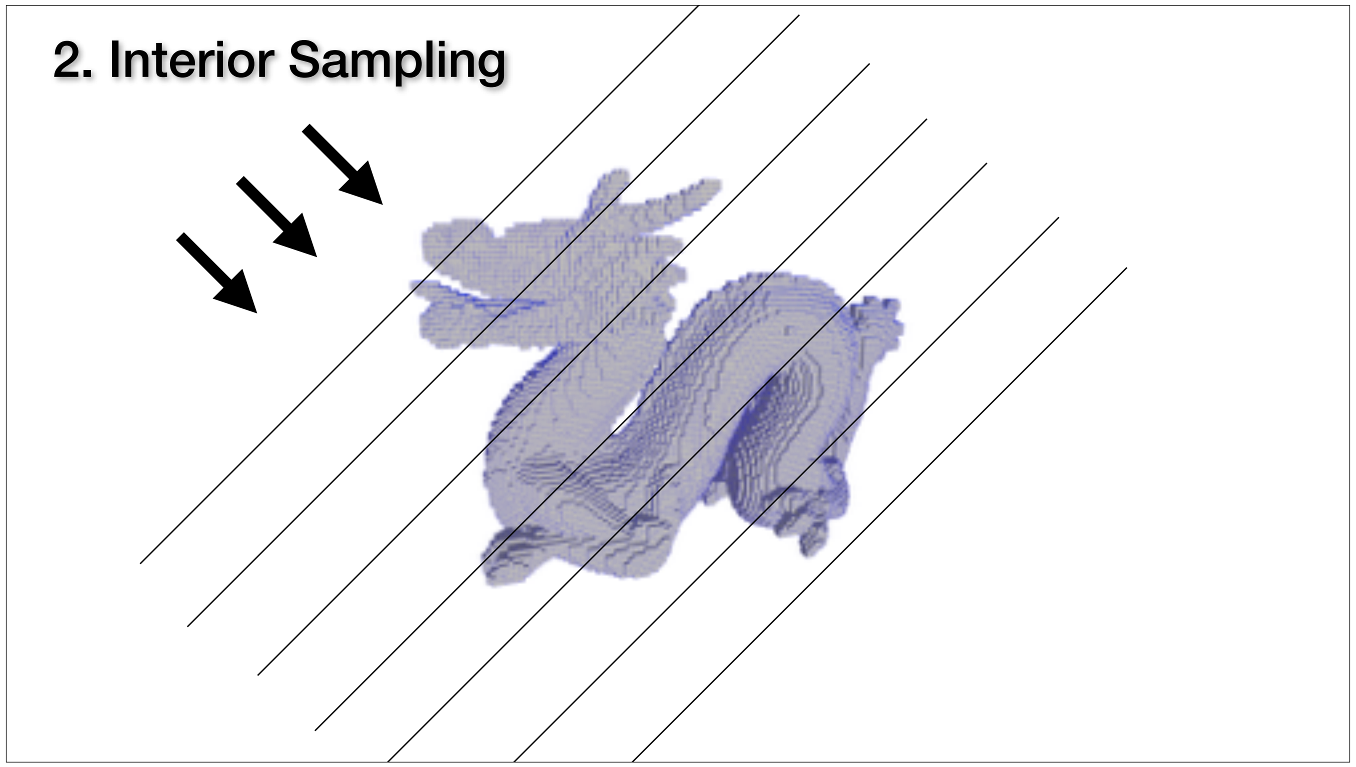
1. Voxelization



We begin by voxelizing the input geometry using accurate triangle-AABB intersection tests. This discretization step introduces some error but it is bounded by voxel resolution.

After the voxelization, we flood-fill the solid interior and finally generate a closed, 2-manifold boundary mesh from the exterior faces.

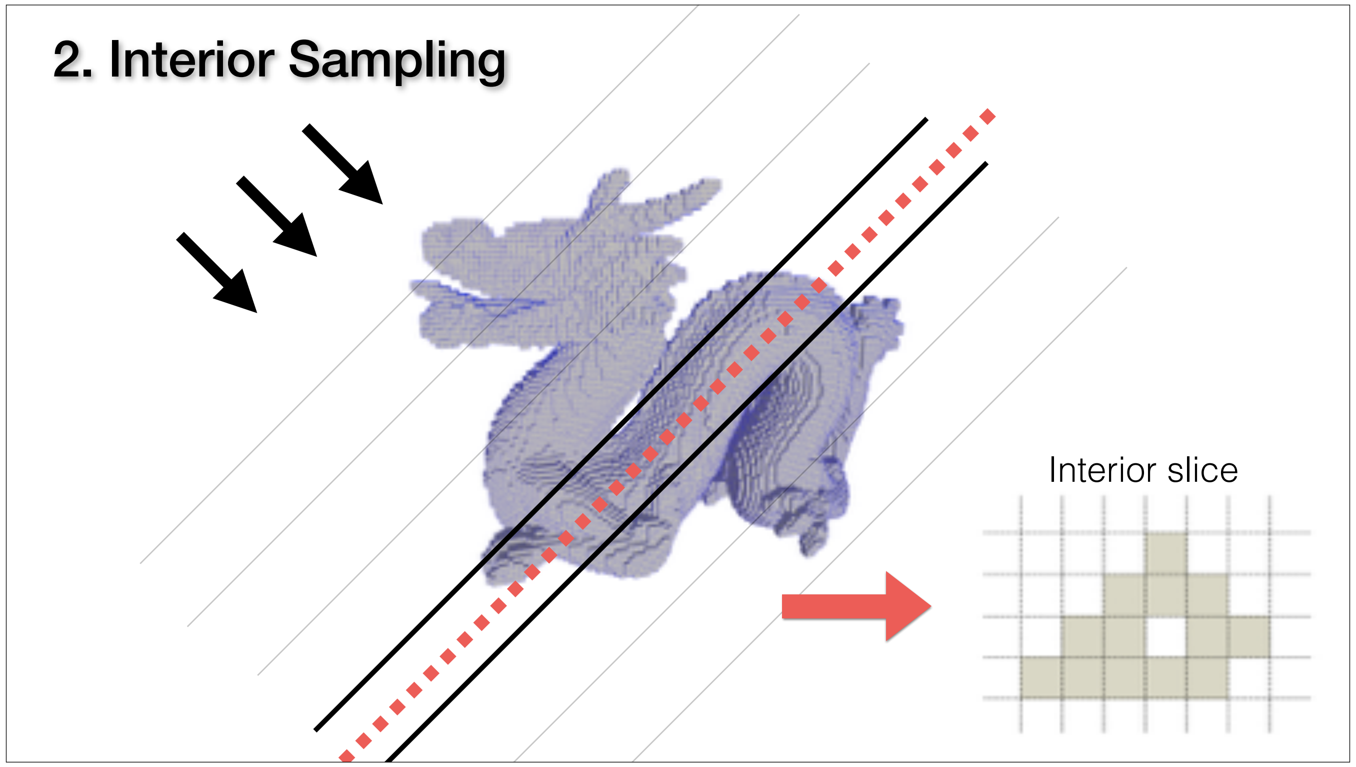
2. Interior Sampling



Next we generate a large number of plane cuts by sampling the space of directions and depths.

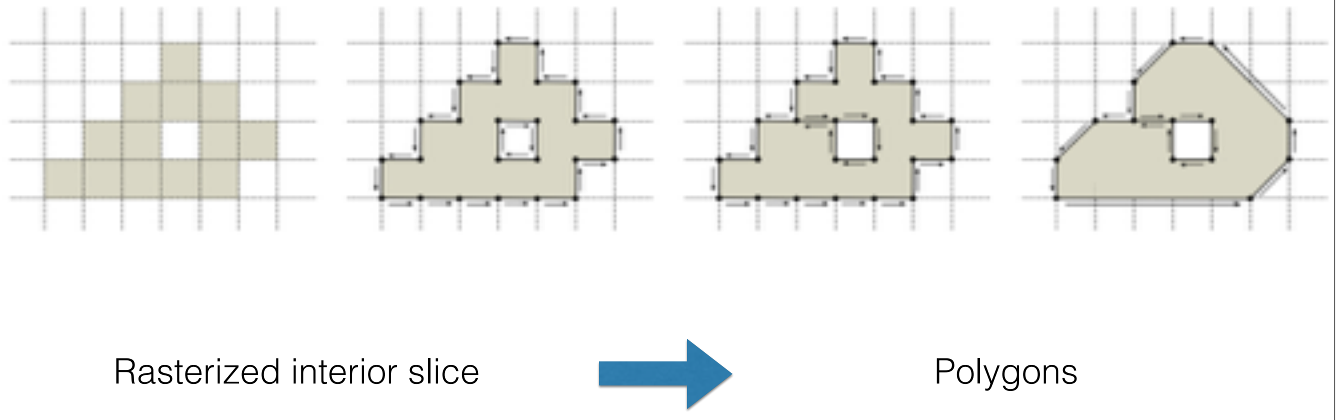
For each direction, we stratify the depth range into depth bins.

2. Interior Sampling



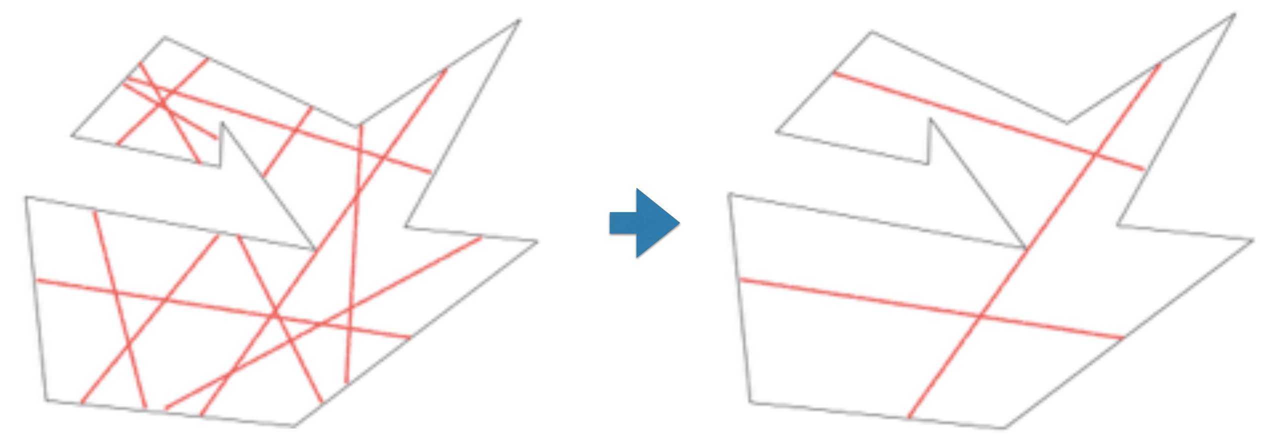
And for each bin we sample the depth range by rasterising the interior of the solid voxel mesh and pick the interior slice with maximal surface area.

2. Interior Sampling

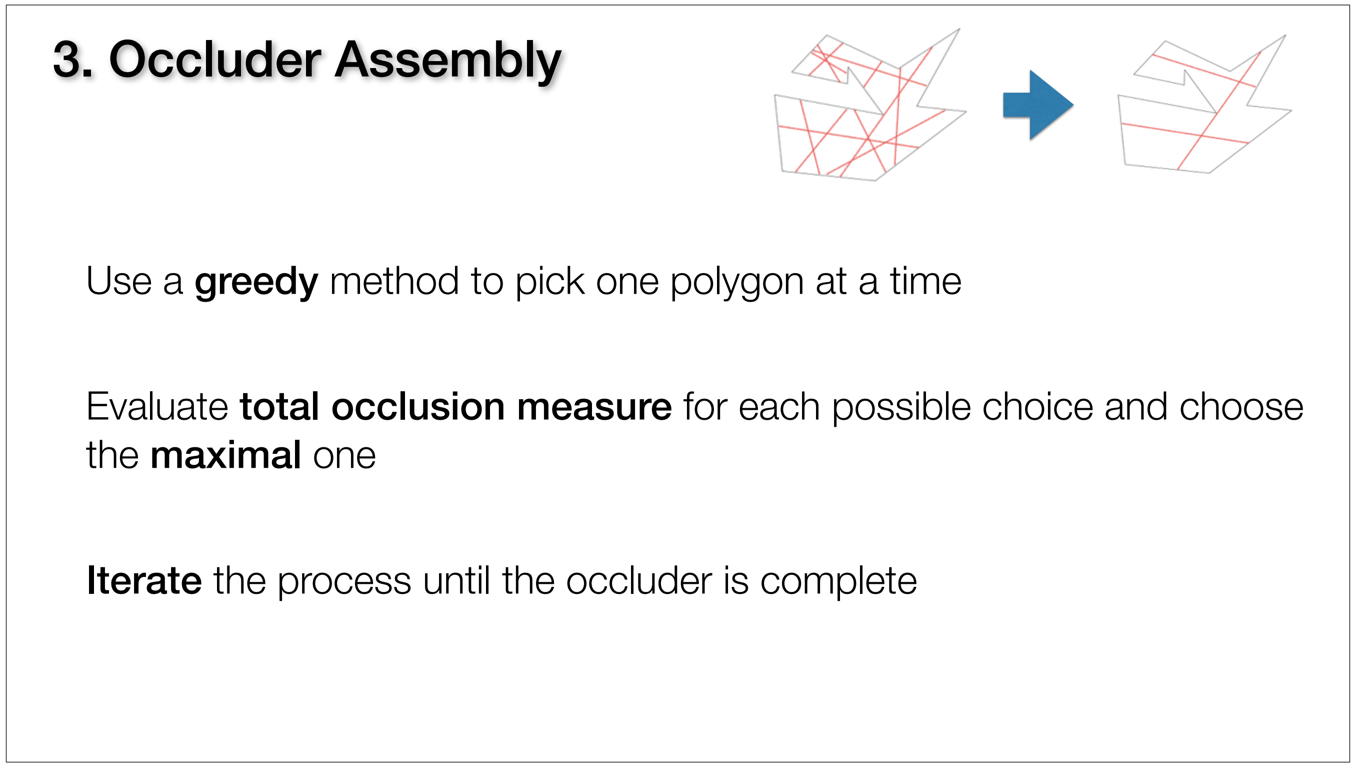


Next we convert the rasterised plane cuts into polygons by running this pipeline which converts the bitmap into edge loops and simplifies the topology in order to generate a set of polygons for each bitmap.

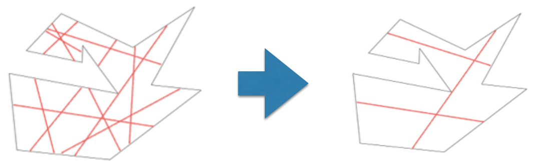
3. Occluder Assembly



The interior sampling produced a large number of candidate polygons, each of which is a valid occluder. But as we saw previously, most of the polygons are redundant and now we need to choose the best ones based on the occlusion measure.



3. Occluder Assembly



Use a **greedy** method to pick one polygon at a time

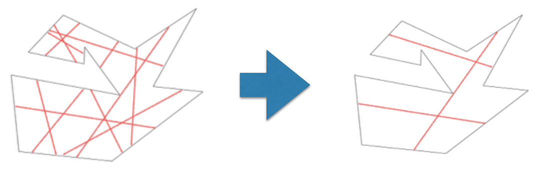
Evaluate **total occlusion measure** for each possible choice and choose the **maximal** one

Iterate the process until the occluder is complete

We use a greedy method to pick one polygon at a time. We evaluate the occlusion measure for each possible choice and choose the maximal one.

This process is iterated until the occlusion measure does not change much when adding new polygons to the occluder.

3. Occluder Assembly

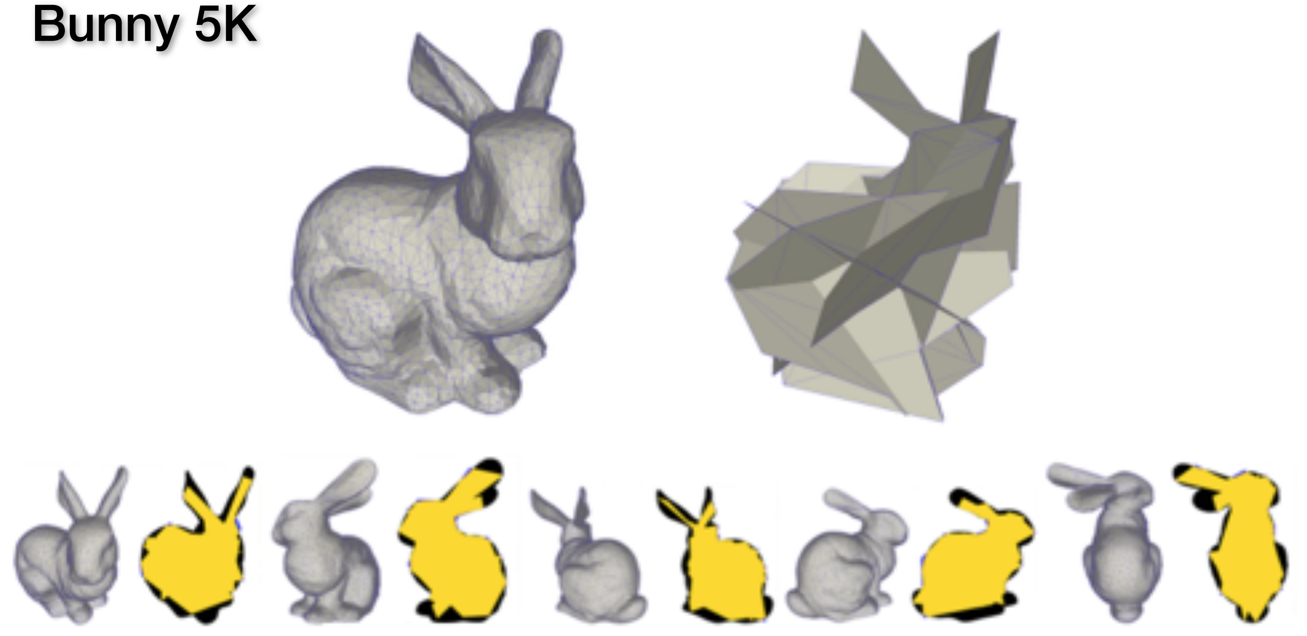


The resulting occluder might still exceed our triangle budget

Remove triangles one-by-one until we reach our budget

The resulting occluder might still exceed our budget, so the final step is to progressively remove triangles from the polygons until we have met the budget.

Bunny 5K

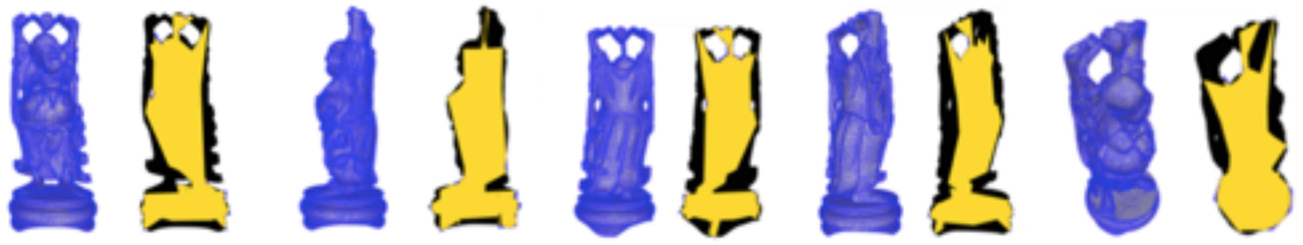


Let’s take a look at some results.

Here is the bunny with 5 thousand triangles and the simplified occluder with 64 triangles.

The bottom row shows the occluder superimposed on the model in various view angles in yellow.

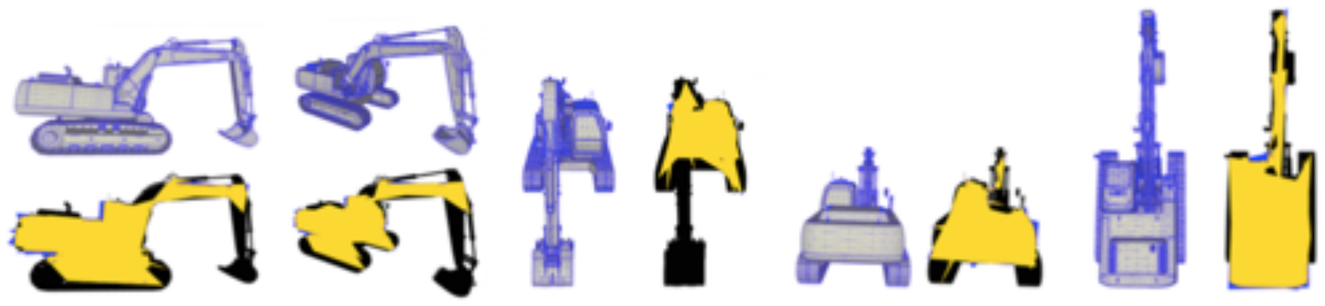
Buddha



Input 1087474 tris **Output** 64 tris

And here is the Buddha. You can see that the resulting occluders respects the holes around the arms in the original model.

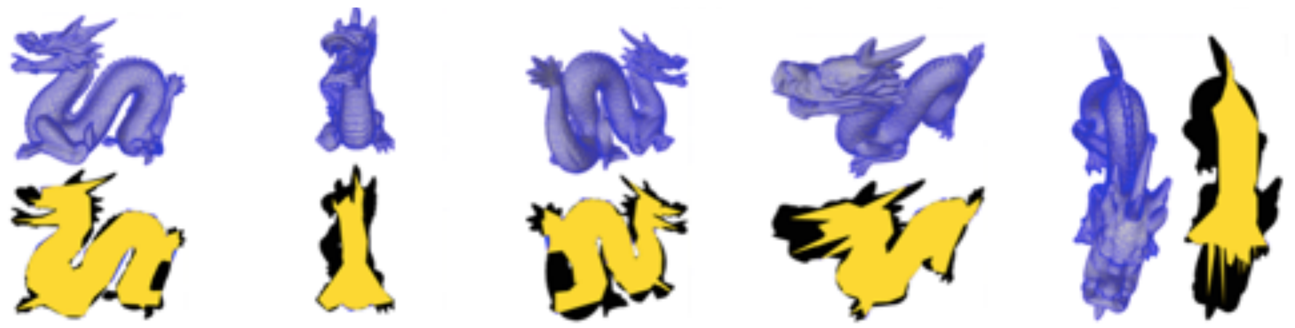
Machine



Input 394452 tris **Output** 64 tris

And here is the machine.

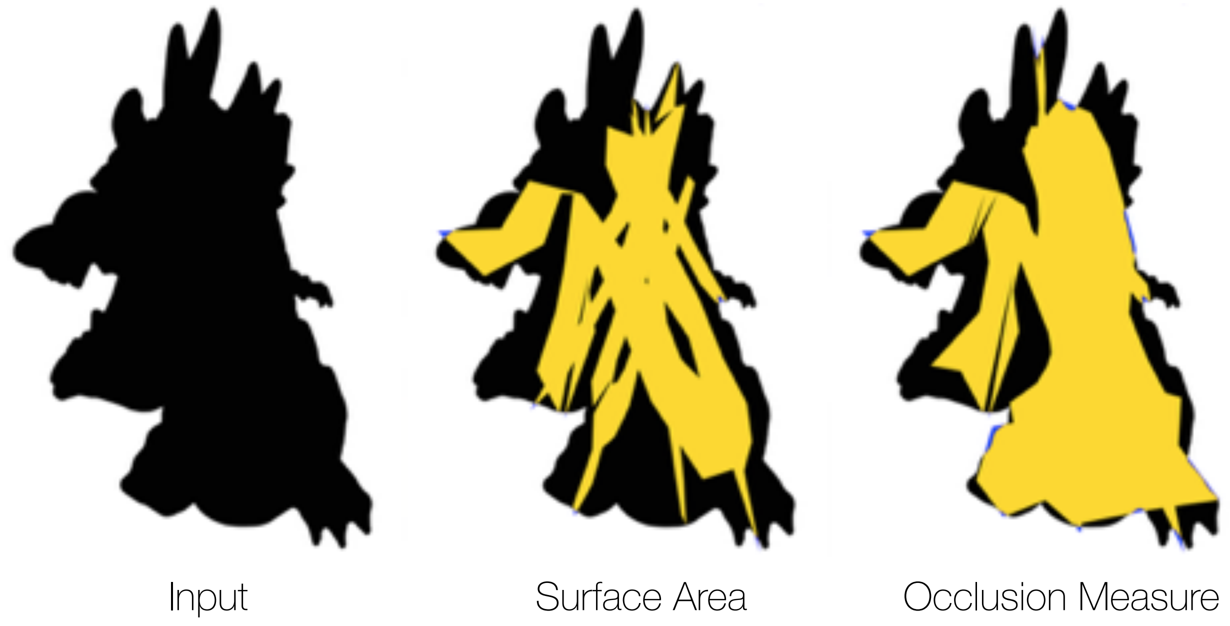
Dragon



Input 871306 tris **Output** 64 tris

And here is the dragon.

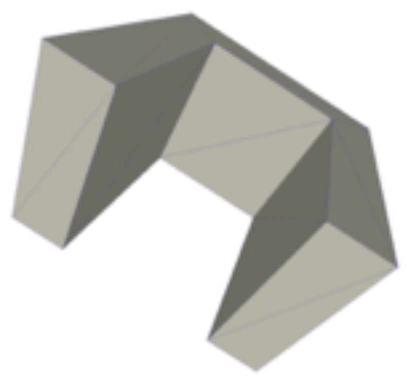
Surface Area vs. Occlusion Measure



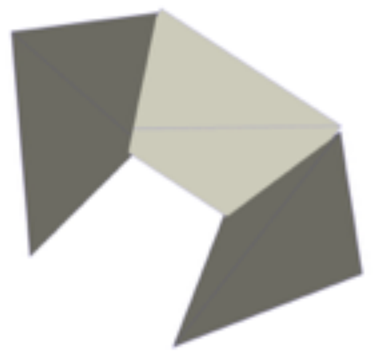
Here is a comparison of using surface area instead of the occlusion measure as we described. The occluder built by choosing polygons based on their surface area instead of occlusion measure has several holes in it from which we can see through.

We also did a statistical precision-recall analysis of the occlusion properties between surface area and the occlusion measure and it showed that occlusion measure is consistently better than surface area. You can find the details in the paper.

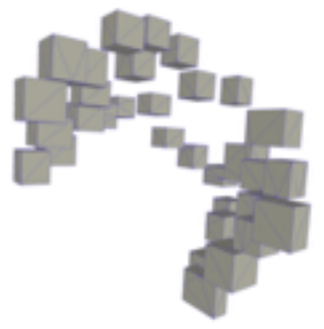
Comparison against Oxel



Input 28 tris



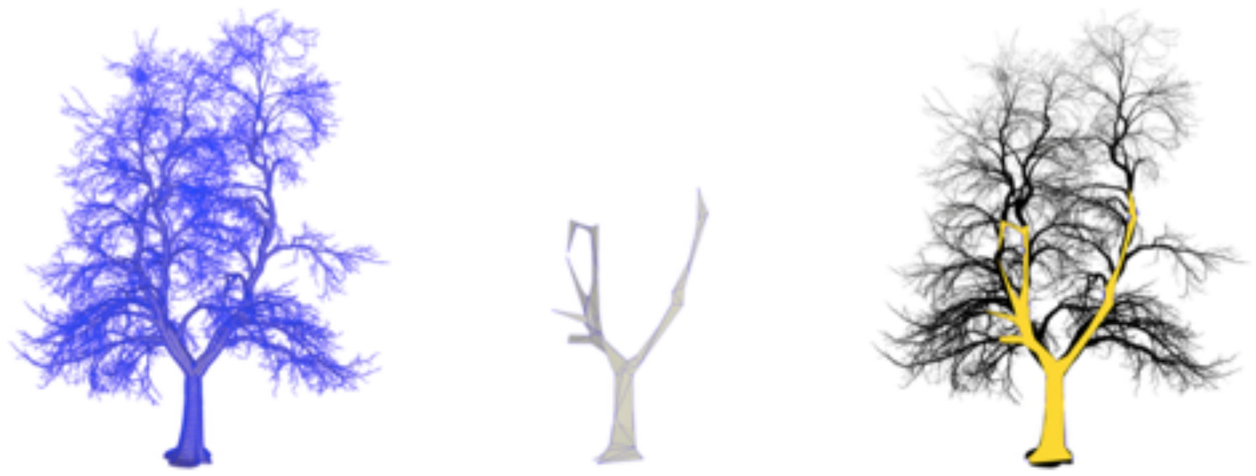
Ours 6 tris



Oxel 509 tris
[Darnell 2011]

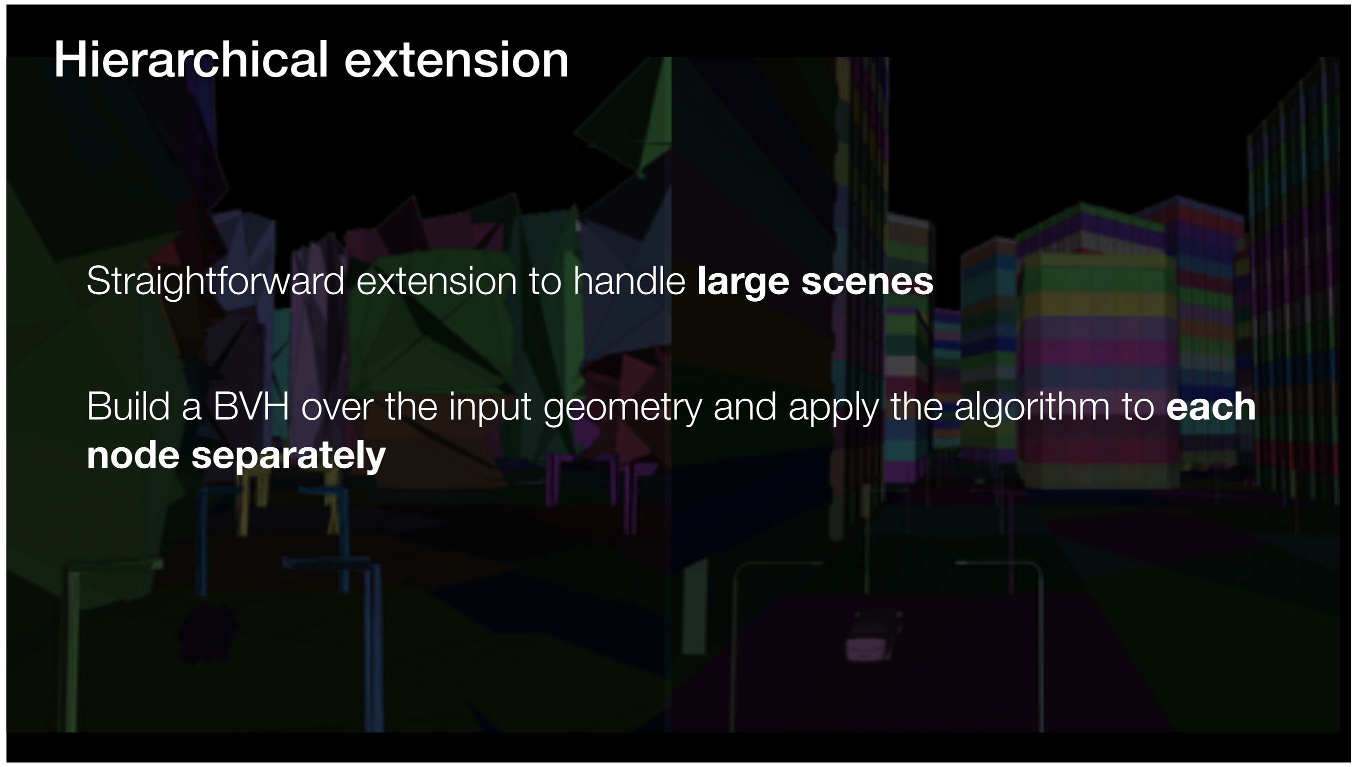
Here is a comparison against Oxel which shows that our method is not sensitive to the orientation of the input geometry.

Difficult Input



Input 100K tris **Ours** 64 tris **Comparison**

And here is an example which shows that our method is robust and can produce reasonable results even for difficult input models.

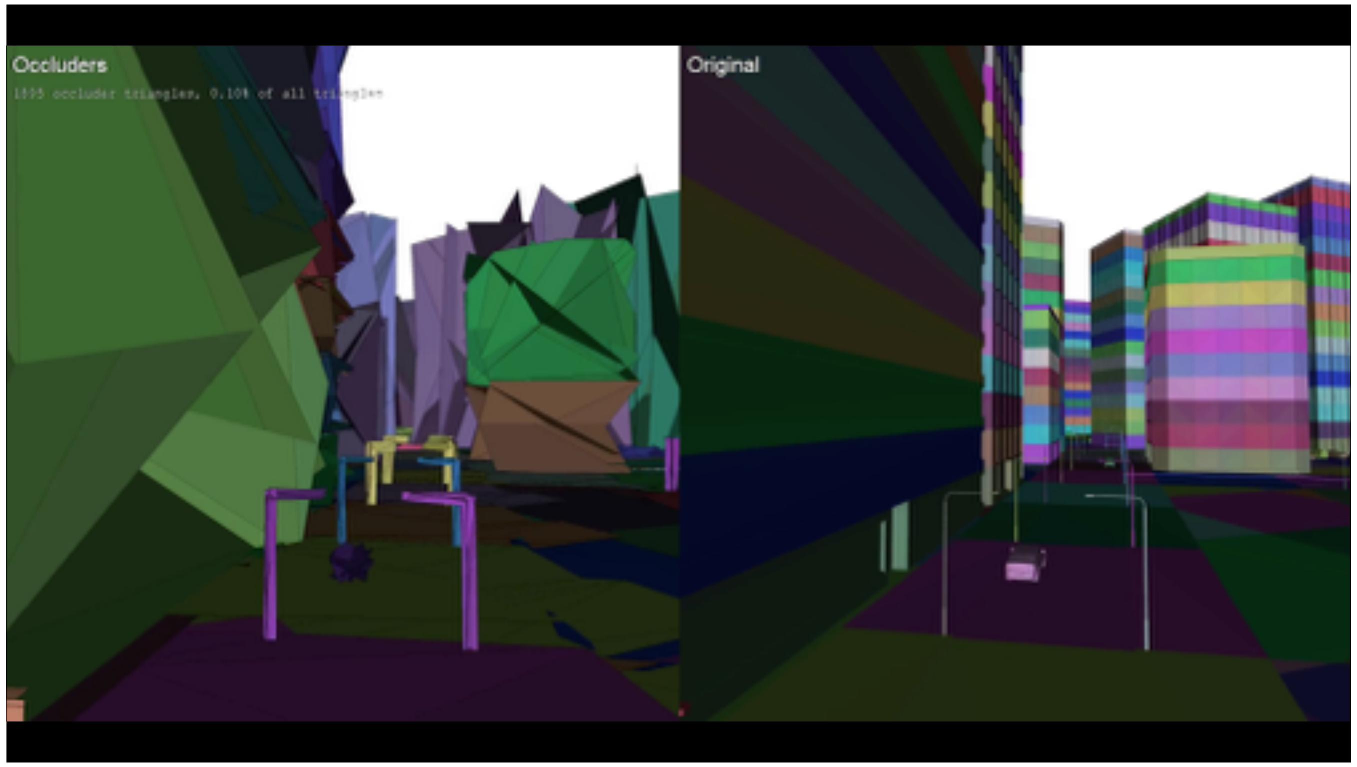


Hierarchical extension

Straightforward extension to handle **large scenes**

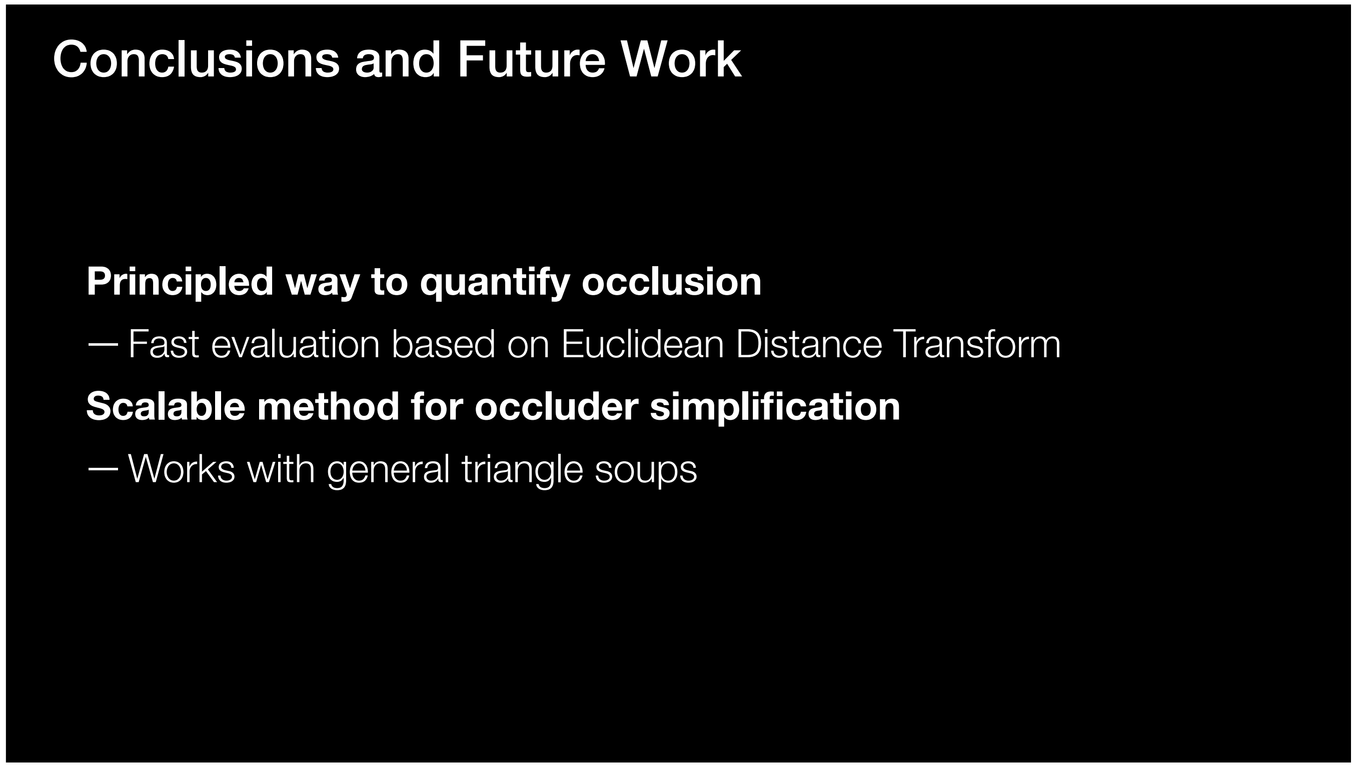
Build a BVH over the input geometry and apply the algorithm to **each node separately**

Finally, it is straightforward to extend the method to handle large scenes by first building a bounding volume hierarchy over the input geometry and then applying the basic method to each hierarchy node in isolation.



Let’s take a look at a video.

Here you can see the hierarchical occluders on the left and the original scene on the right.



Conclusions and Future Work

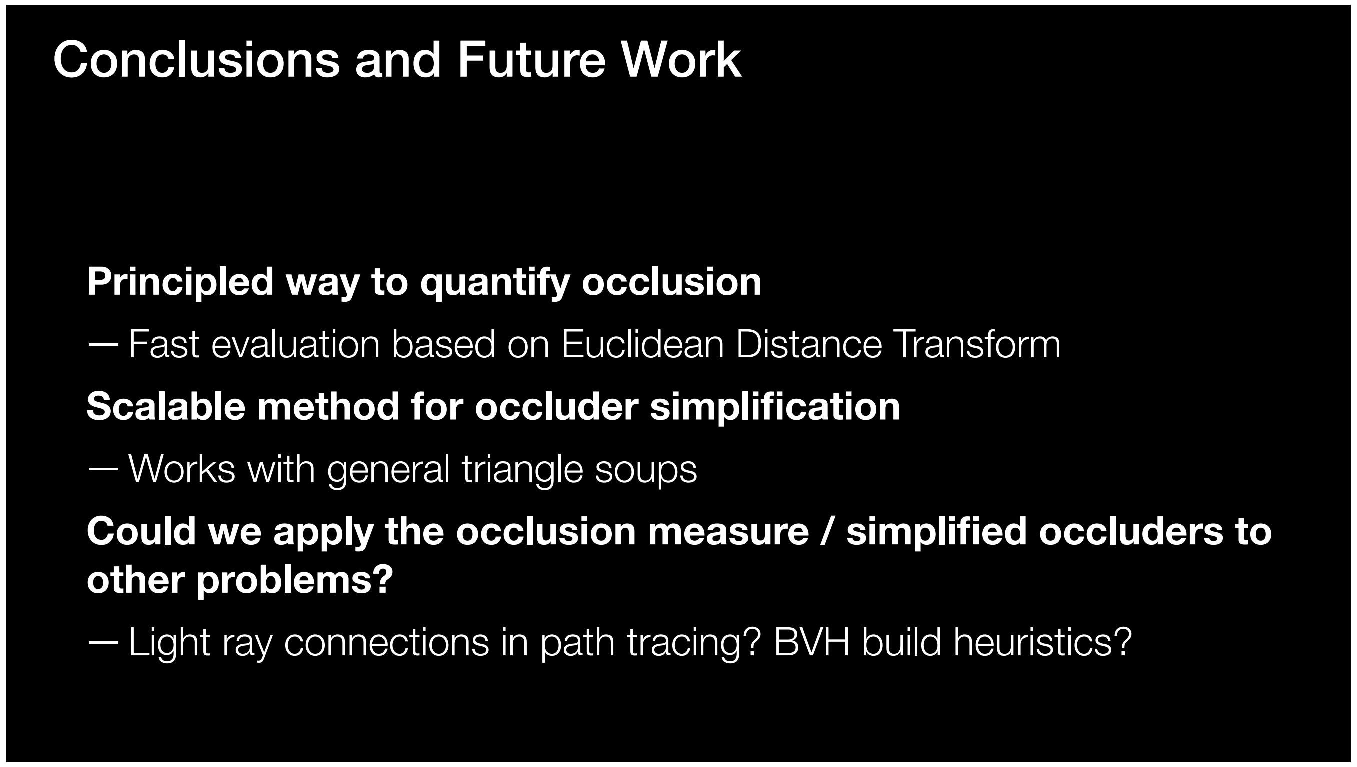
Principled way to quantify occlusion

— Fast evaluation based on Euclidean Distance Transform

Scalable method for occluder simplification

— Works with general triangle soups

To sum up, we presented a principled way to quantify and evaluate occlusion together with a scalable and general method for occluder simplification.



Conclusions and Future Work

Principled way to quantify occlusion

— Fast evaluation based on Euclidean Distance Transform

Scalable method for occluder simplification

— Works with general triangle soups

Could we apply the occlusion measure / simplified occluders to other problems?

— Light ray connections in path tracing? BVH build heuristics?

Finally, we may ask if we could apply the occlusion measure or the simplified occluders in other problem domains, and with that, I'll conclude my talk.

Thank You!

Acknowledgments

Anonymous reviewers for constructive and thorough feedback

umbra A! Aalto University School of Science NVIDIA REMEDY

Bounded Approximation Error

Sources of error

— Voxelization error (bounded by voxel size)

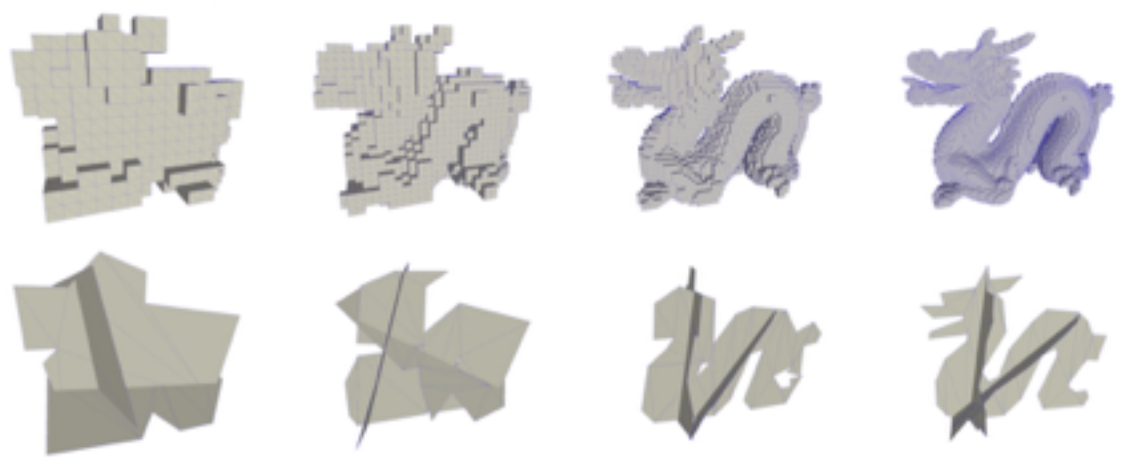
— Rasterization error (bounded by raster resolution)

— Edge loop simplification error (bounded by tolerance)

Single user parameter: Voxel resolution

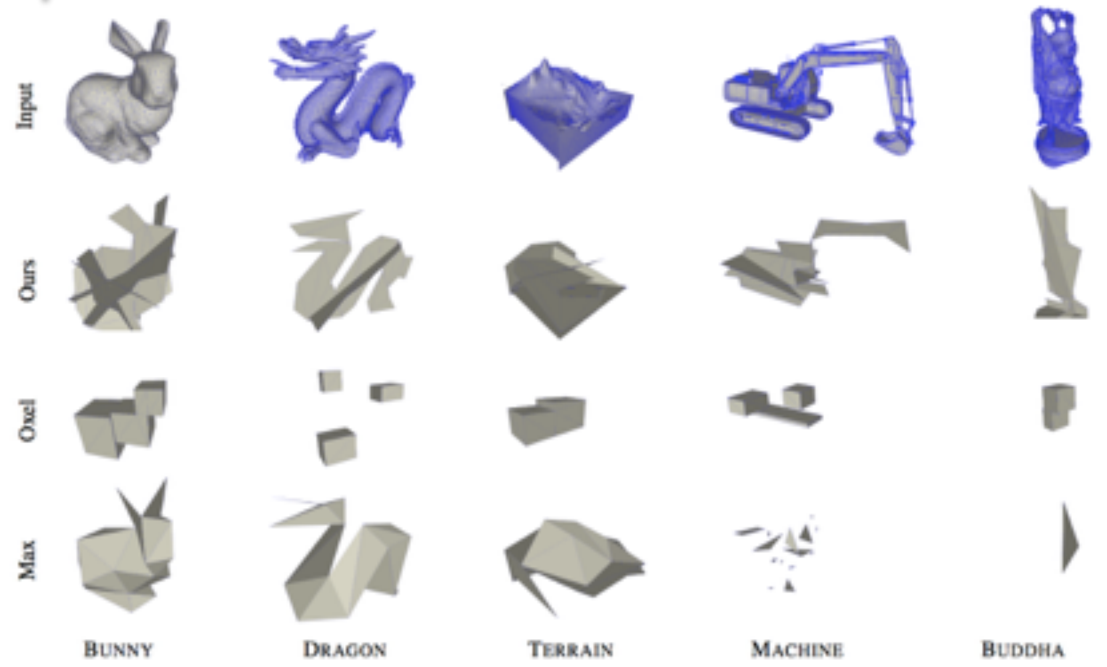
— Raster resolution and edge loop simplification set accordingly

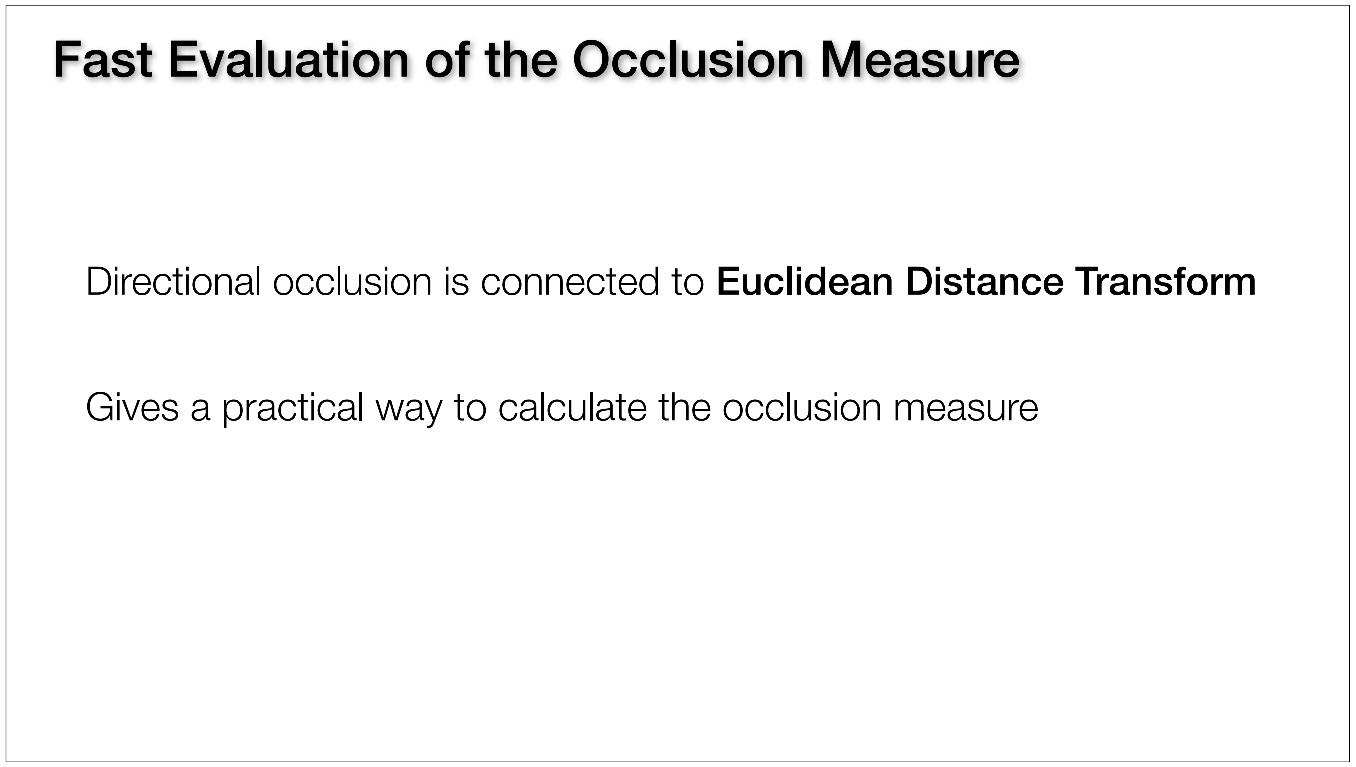
Effect of Scale-Sensitive Discretization



16x16x16 32x32x32 64x64x64 128x128x128

Comparison to Other Methods





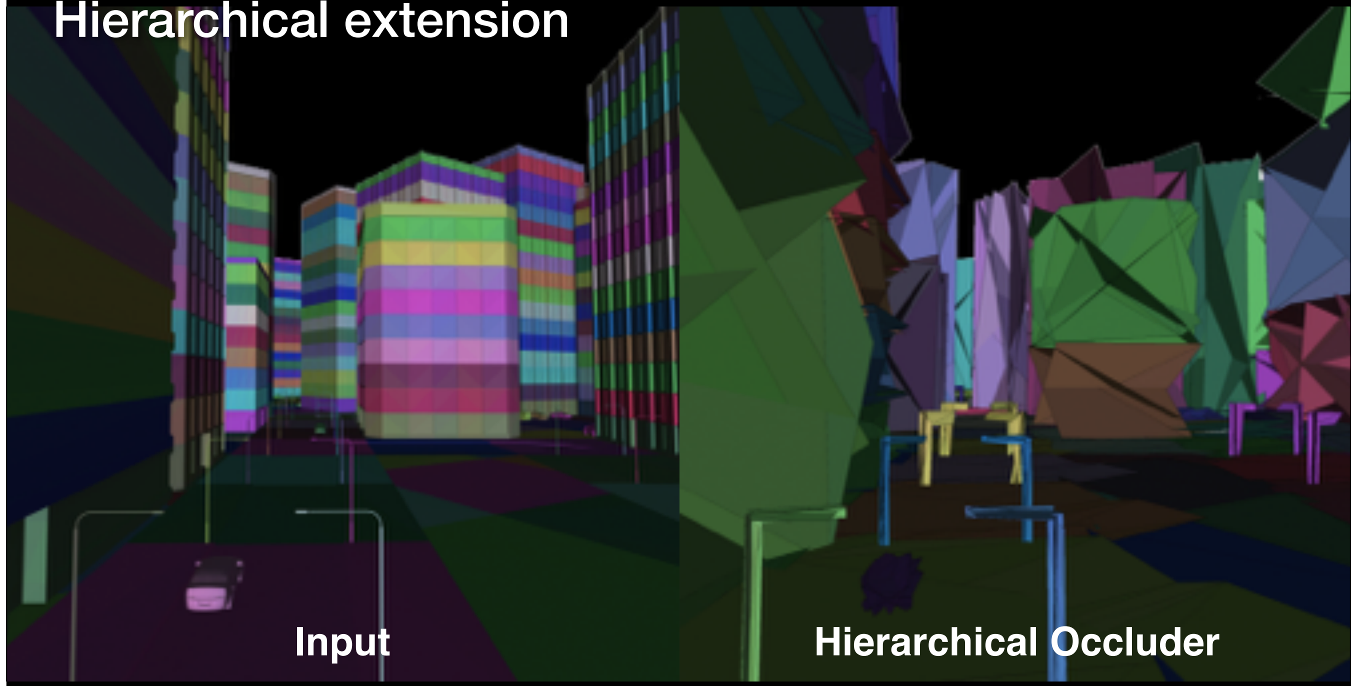
Fast Evaluation of the Occlusion Measure

Directional occlusion is connected to **Euclidean Distance Transform**

Gives a practical way to calculate the occlusion measure

The directional occlusion measure is directly connected to Euclidean Distance Transform. And this is great news, because it gives us a practical and efficient way of evaluating the occlusion measure. The details can be found in the paper.

Hierarchical extension



And here you can see a side-by-side image showing a large scene and a the hierarchical occluder.