SCREEN-SPACE FAR-FIELD AMBIENT OCCLUSION

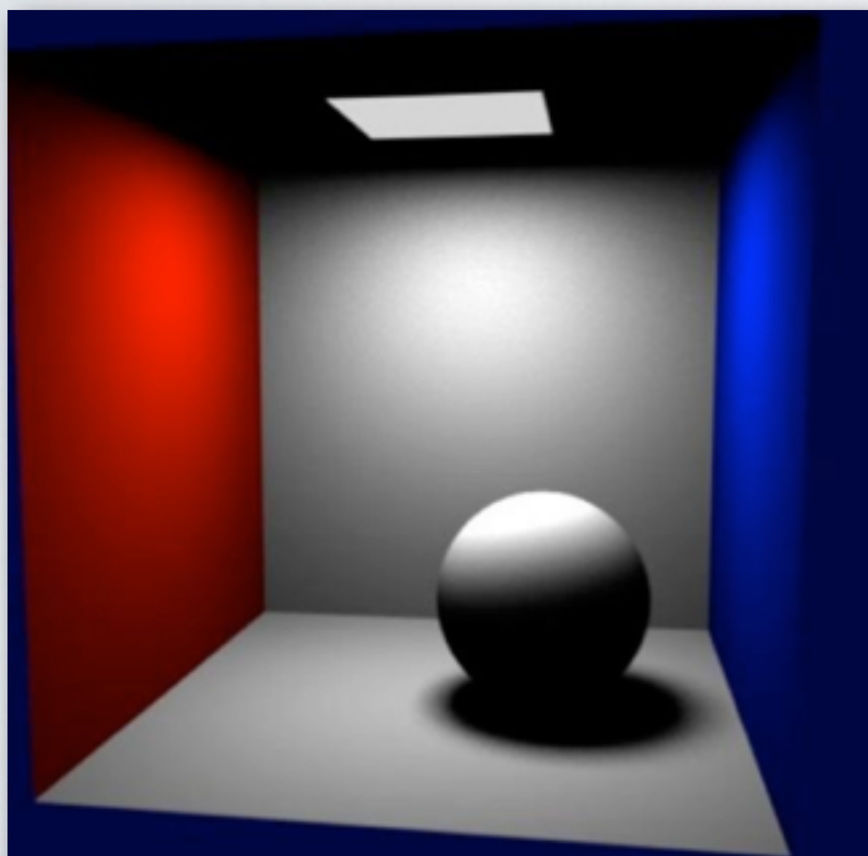
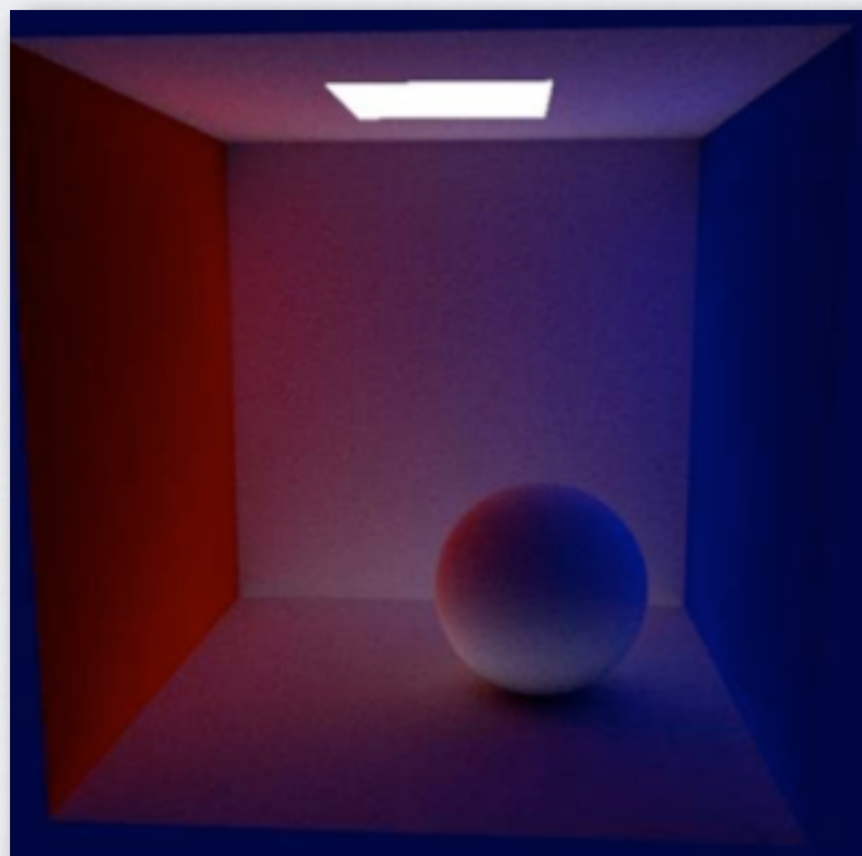
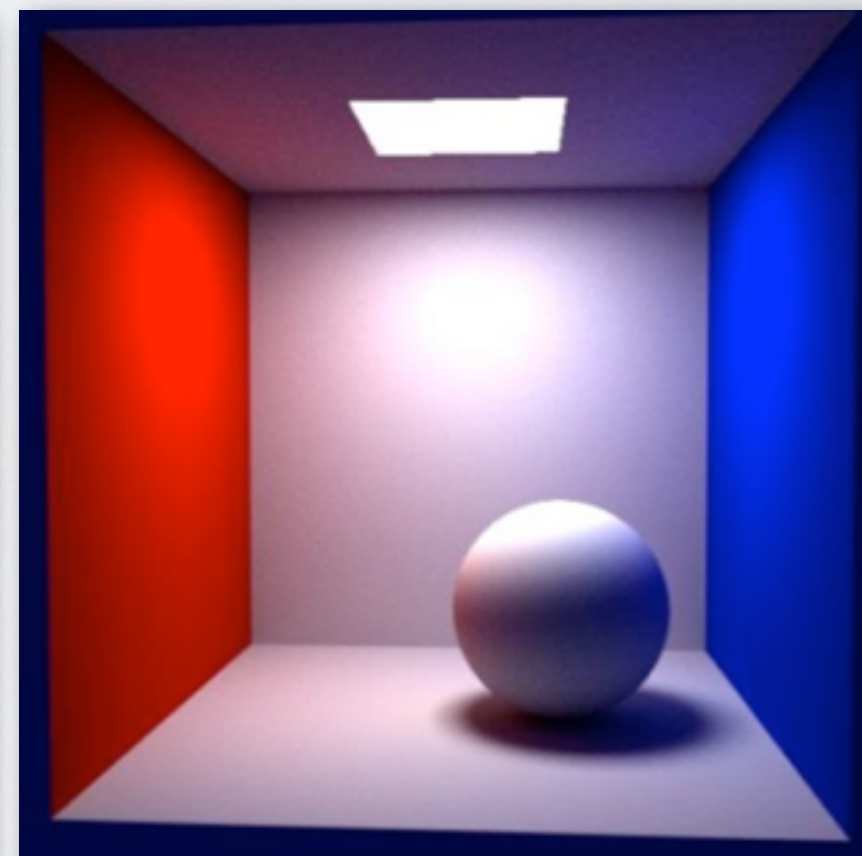
Ville Timonen

CONTENTS

1 AMBIENT OCCLUSION

Components of Light

Direct light + Indirect light = Global lighting

1 AMBIENT OCCLUSION

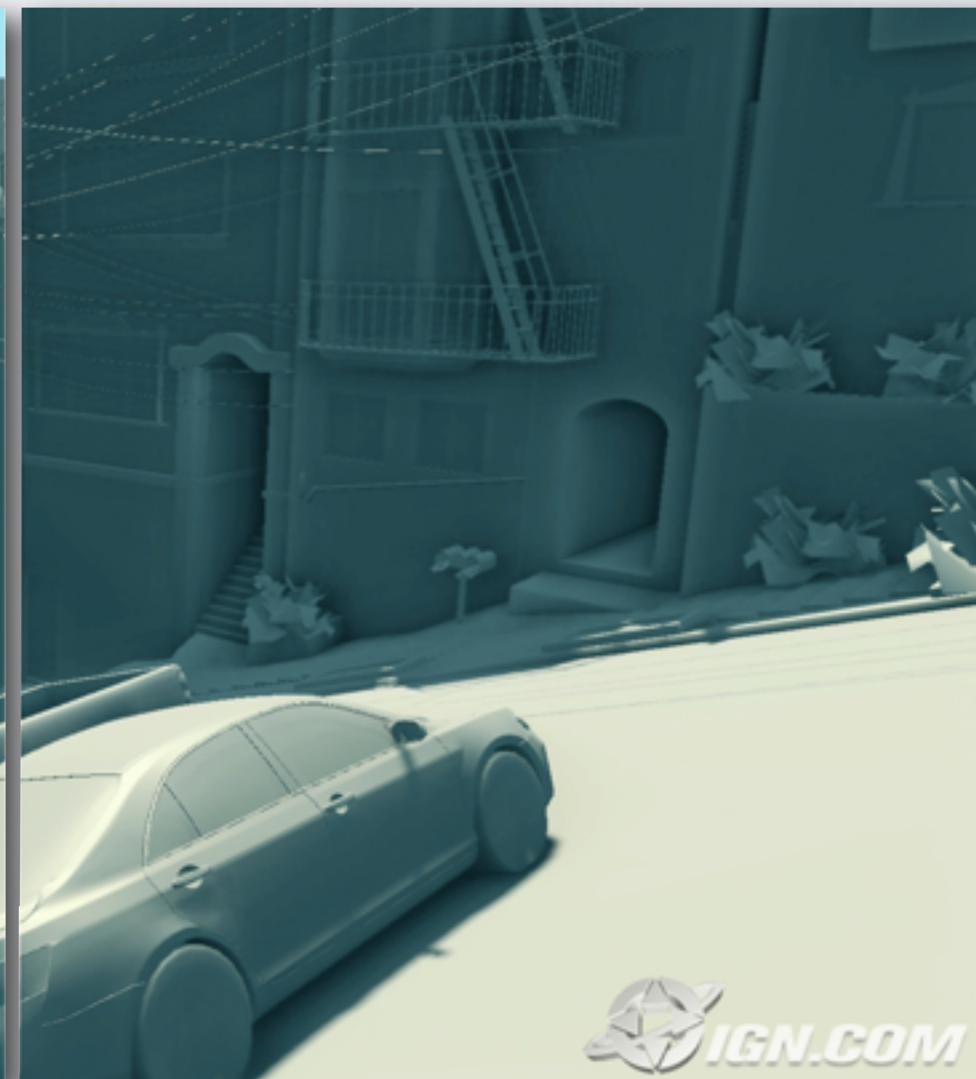
Components of Light

- Physically no need to treat different light sources differently
- However different approximations/algorithms suitable for each type
- Ambient occlusion (AO) approximates lighting from uniformly lit surroundings
- Complements direct lighting from local light sources (lamps, etc)

1 AMBIENT OCCLUSION

Components of Light

Direct (sun)

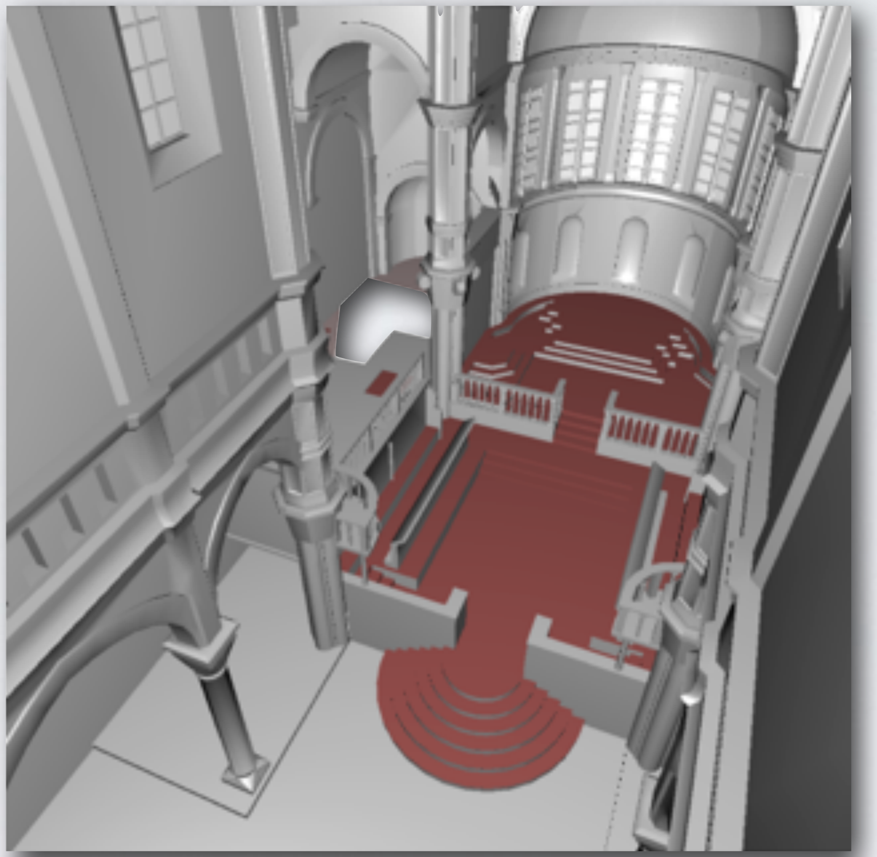


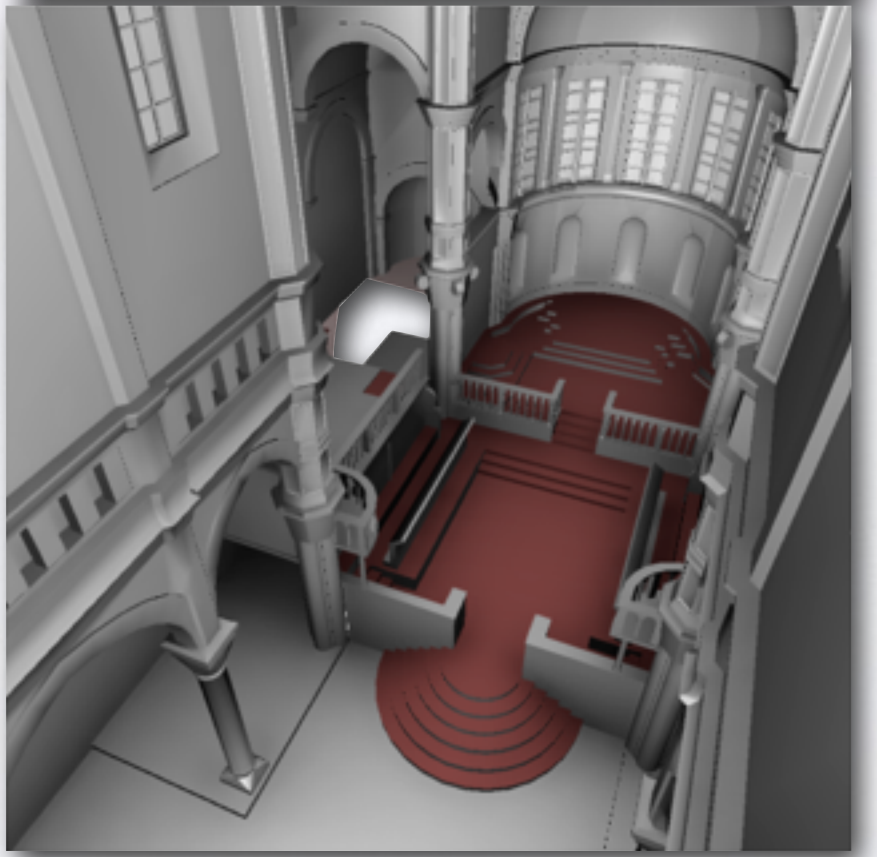
+AO (indirect)

1 AMBIENT OCCLUSION

Components of Light

Direct





+AO

1 AMBIENT OCCLUSION

Want to produce this:

1 AMBIENT OCCLUSION

Want to produce this:

I AMBIENT OCCLUSION

Want to produce this:

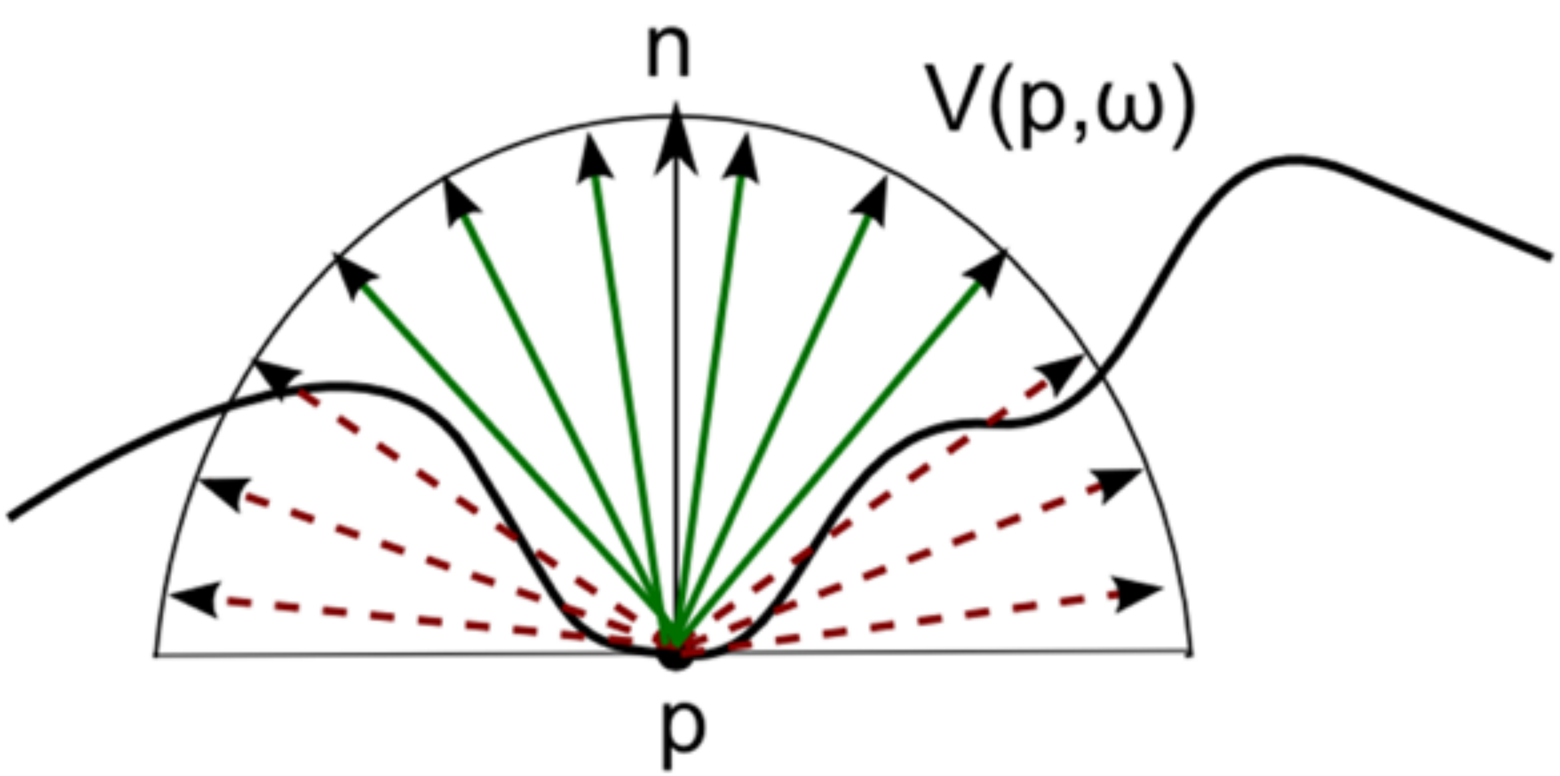
1 AMBIENT OCCLUSION

Want to produce this:

1 AMBIENT OCCLUSION

Definition of AO

- Screen consists of 1-2M points (e.g. 1920x1080, FullHD)
- For each point, determine whether there is surrounding occluders



$$AO(\mathbf{p},\mathbf{n}) = \frac{1}{\pi}\int_{\Omega} V(\mathbf{p},\omega)\,\mathbf{n}\cdot\omega\, d\omega,$$

"Skylight"

1 AMBIENT OCCLUSION

Definition of AO

- In addition to on/off visibility (skylight), the surrounding geometry also reflects light (otherwise indoor scenes would be pitch black)
- Add a falloff term F that tapers off as a function of distance
    - F(0) = 1, F(inf) = 0

$$A(\mathbf{p}, \vec{n}) = \frac{1}{\pi} \int_\Omega F(D(\mathbf{p}, \vec{\omega}))\, \vec{n} \cdot \vec{\omega} d\vec{\omega}$$

- Now need to know distance to occluder, D

CONTENTS

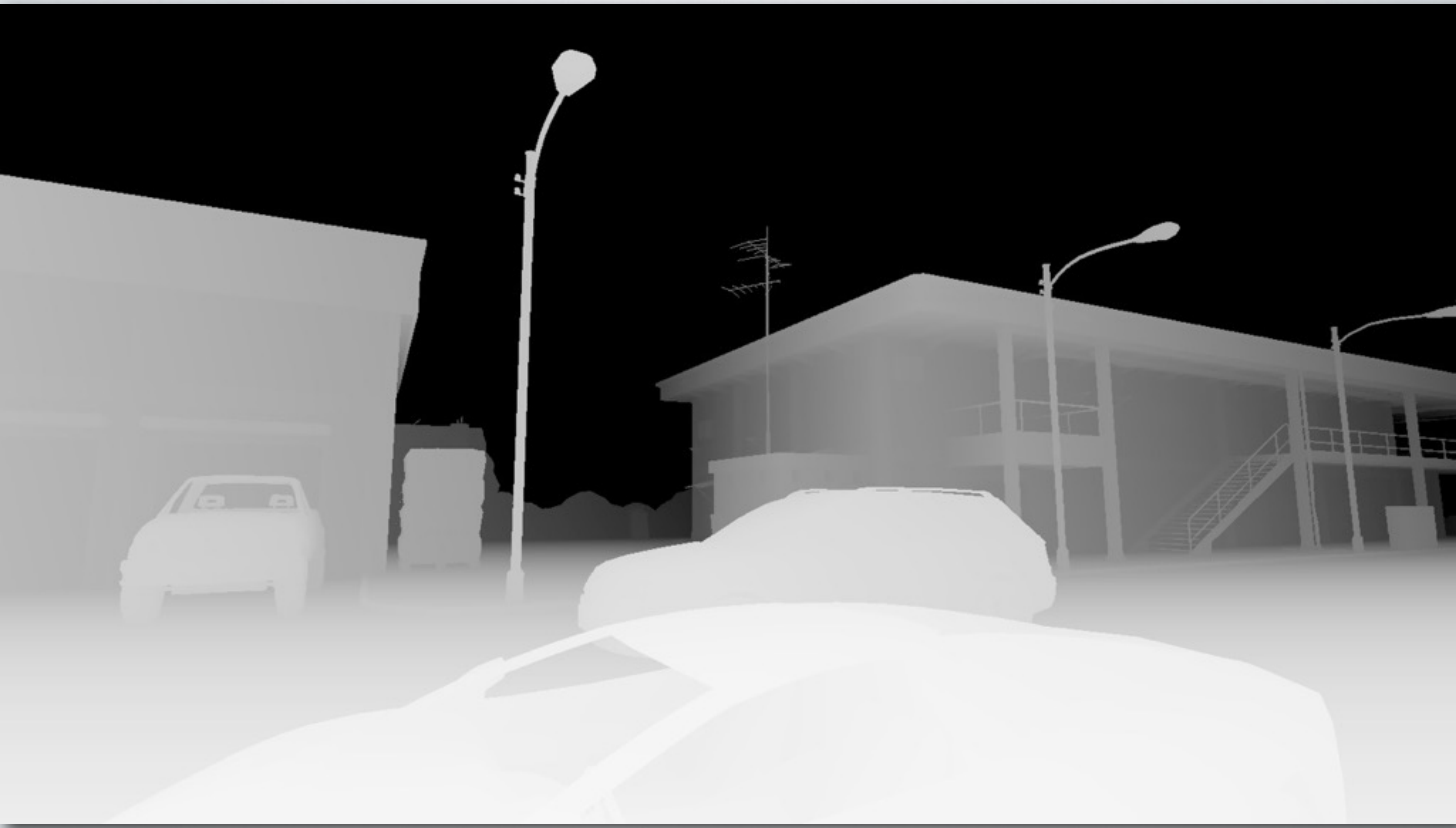
1. Ambient Occlusion
2. **Screen-Space Ambient Occlusion**
3. Previous methods
4. Our method
5. Results

2 SCREEN-SPACE AMBIENT OCCLUSION

- Used in real-time graphics (computer games)
- Fast but uses incomplete geometry of the scene: a depth buffer
- Depth buffer (aka Z buffer) is a free by-product of a rendering pipeline
  - Used to determine visibility
  - Distance value (camera -> geometry) for each screen pixel

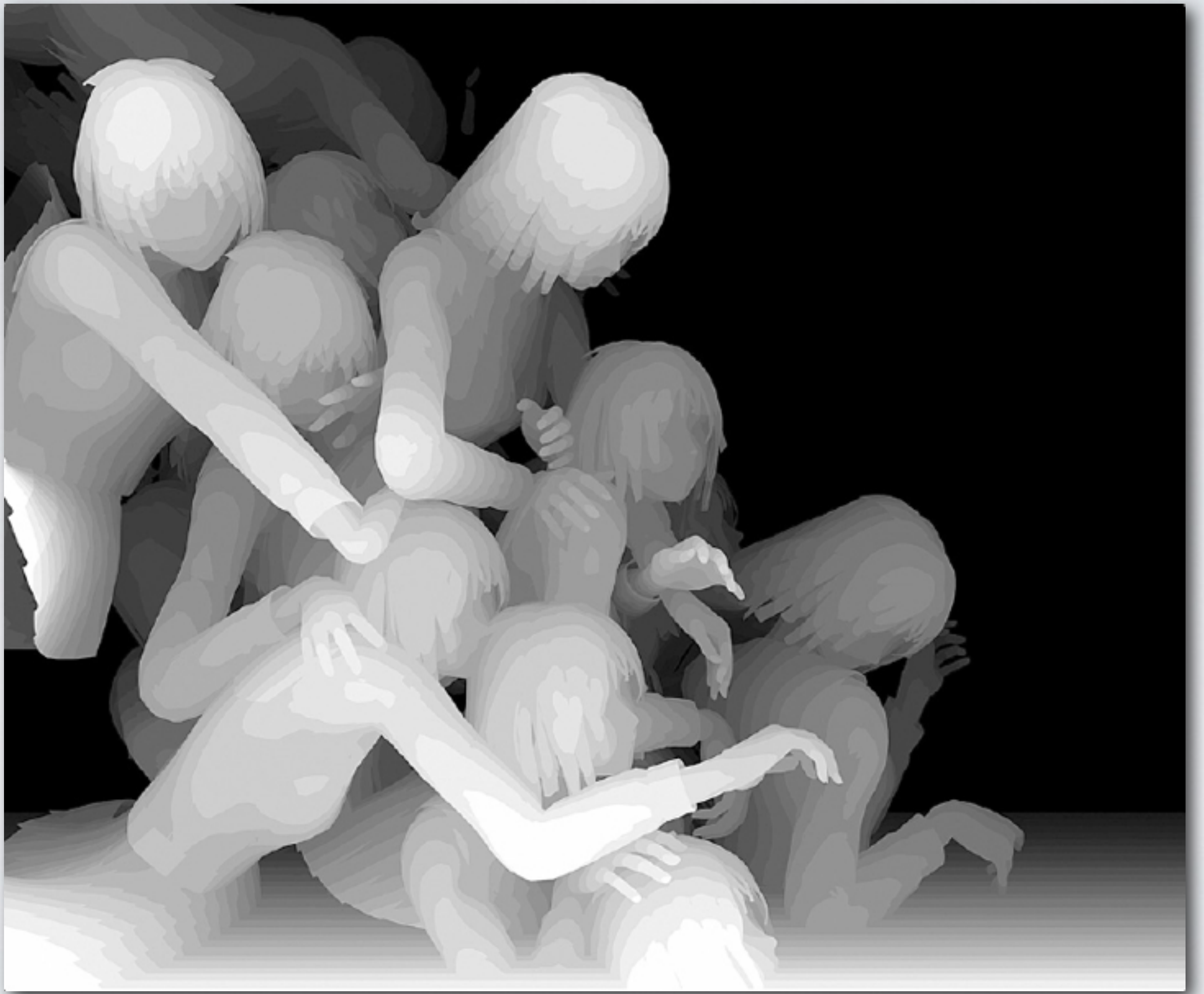
2 SCREEN-SPACE AMBIENT OCCLUSION

Example depth buffer (dark = far, light = near)



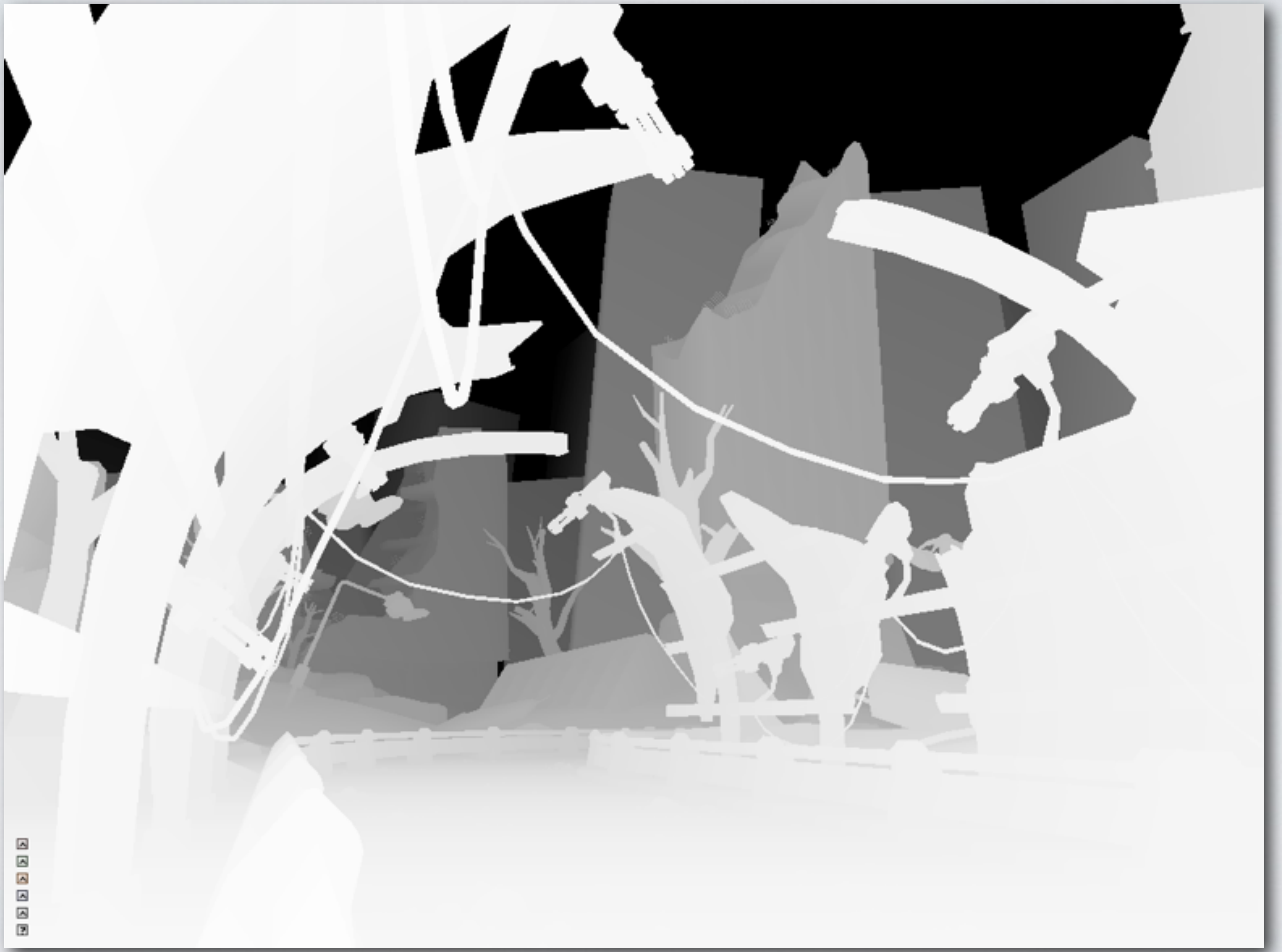
2 SCREEN-SPACE AMBIENT OCCLUSION

Example depth buffer (dark = far, light = near)



2 SCREEN-SPACE AMBIENT OCCLUSION

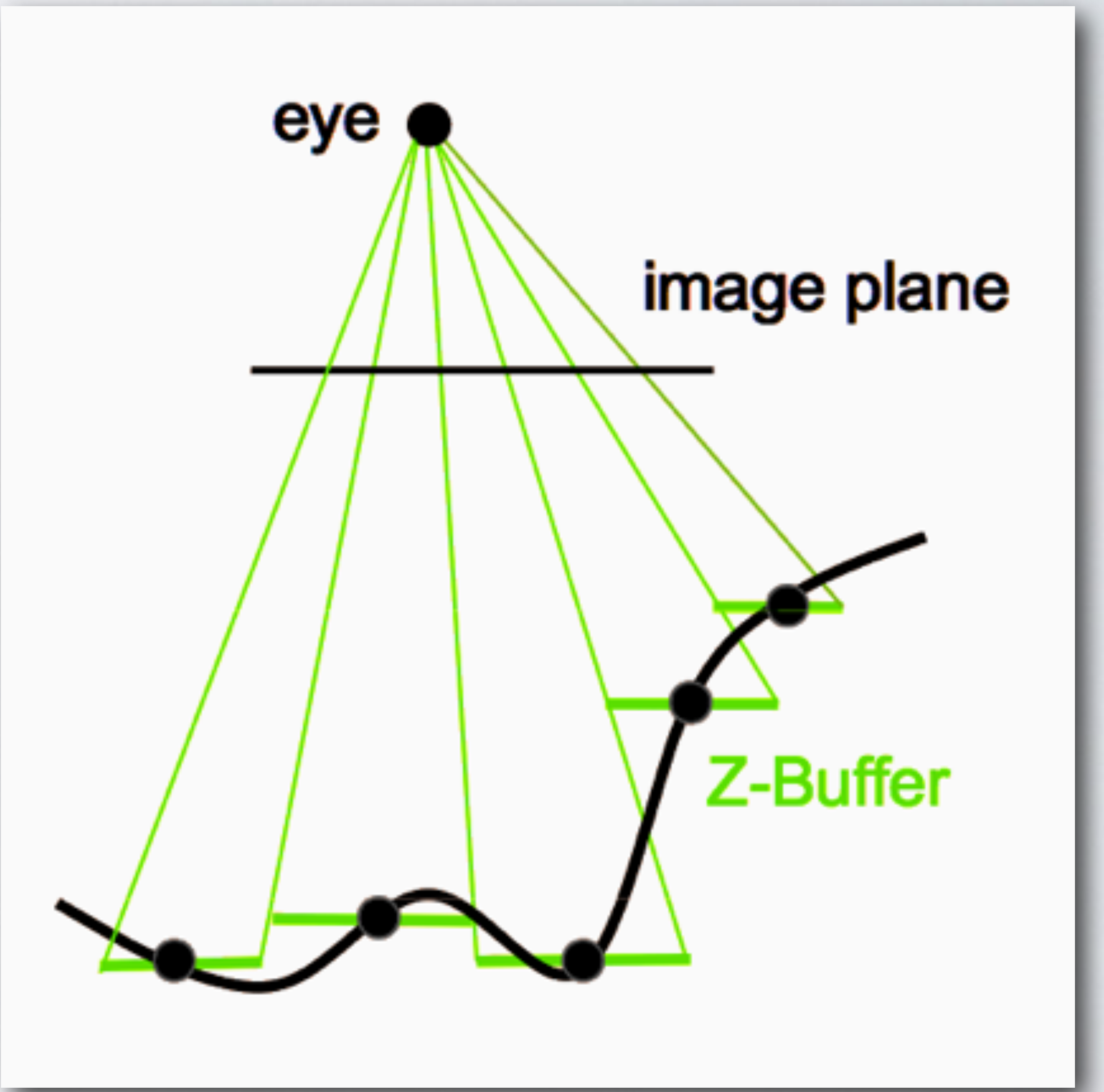
Example depth buffer (dark = far, light = near)



2 SCREEN-SPACE AMBIENT OCCLUSION

Depth to Height

- Flip the depth buffer around and it becomes a height field
- We don't know what's behind the first layer
  - For now, let's assume it represents solid geometry
  - Can be linearly interpolated



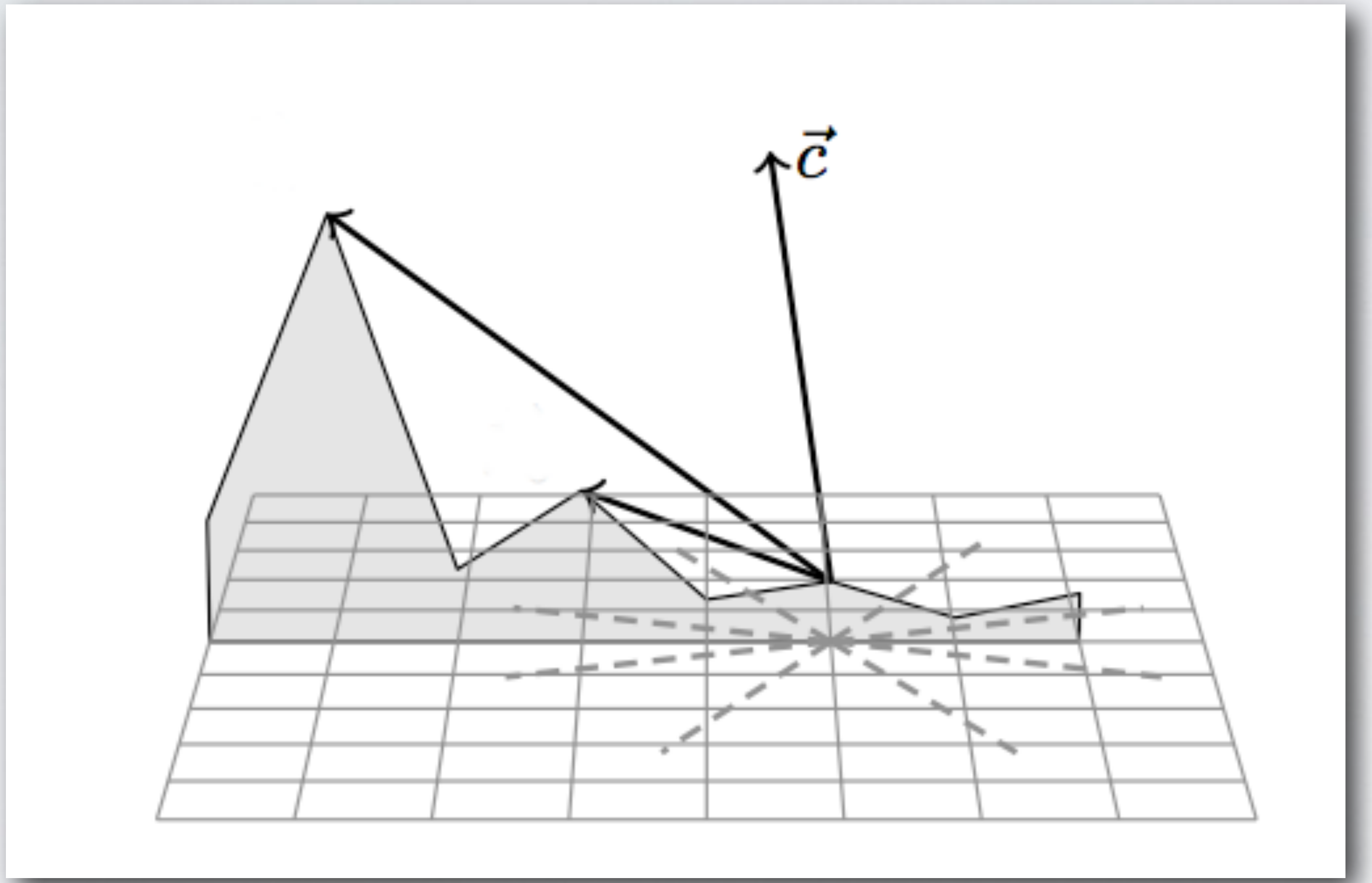
CONTENTS

1. Ambient Occlusion
2. Screen-Space Ambient Occlusion
3. **Previous methods**
4. Our method
5. Results

3 PREVIOUS METHODS

2D integral to K x 1D

- Instead of evaluating the 2D integral, decompose it into multiple (K) 1D integrals



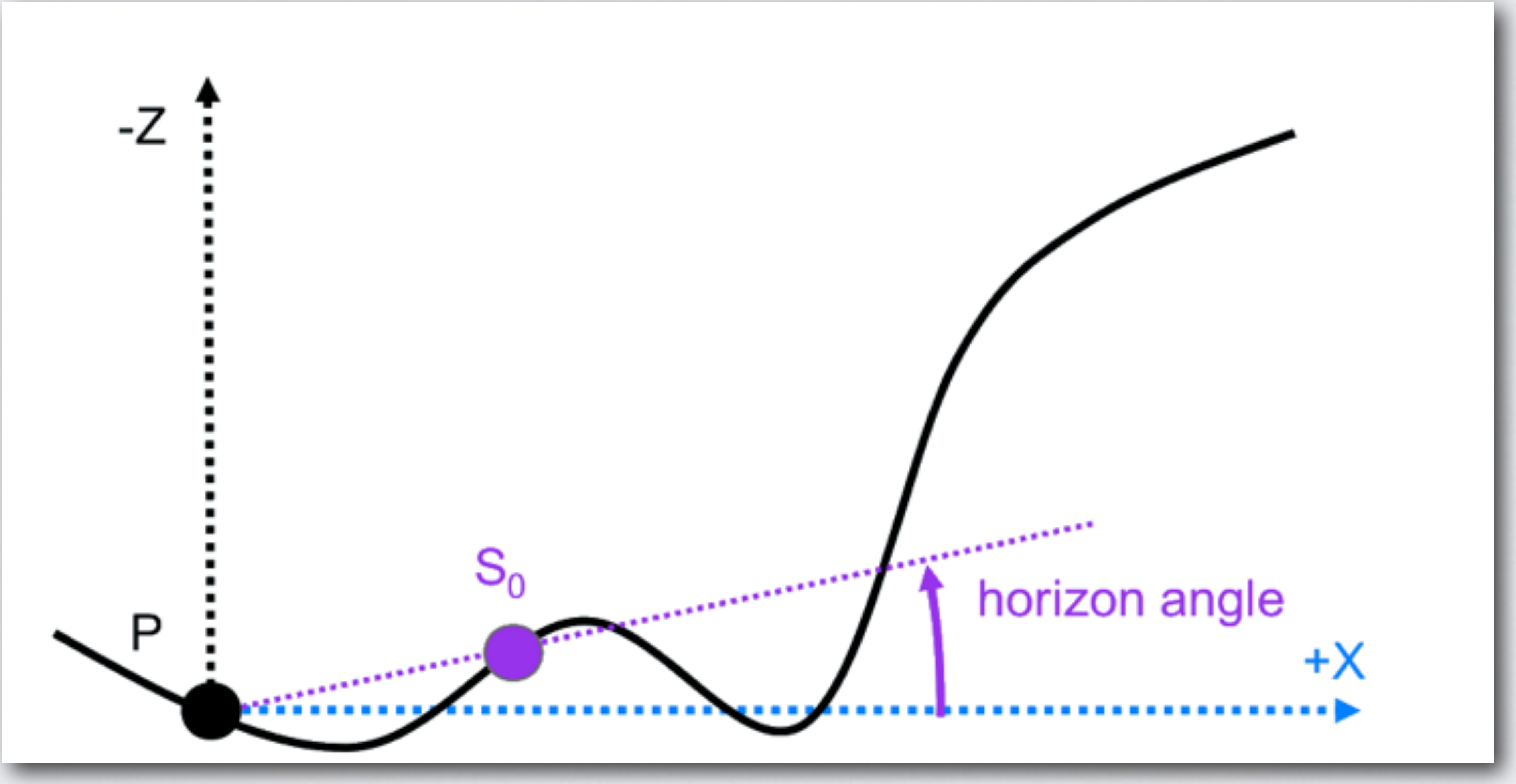
Here K=8

- I.e. from each receiver point (screen pixel), occluders are searched in K azimuthal directions

3 PREVIOUS METHODS

Marching along one of the K directions

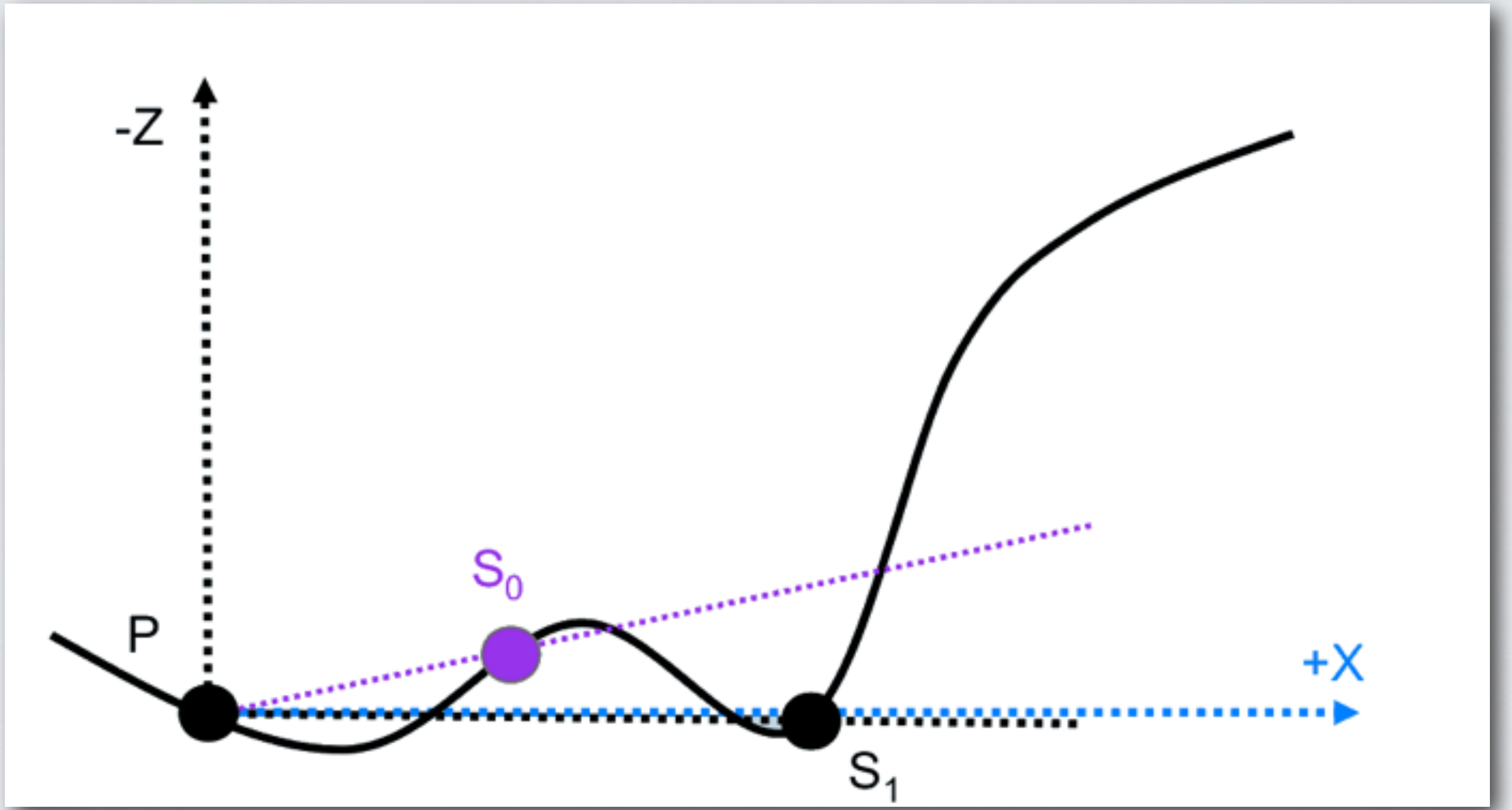
- Take steps (S_n) of constant length along the direction
- Keep track of horizon angle



3 PREVIOUS METHODS

Marching along one of the K directions

- When the horizon (from P to S_n) exceeds the previous max, the new point (S_n) is visible to P

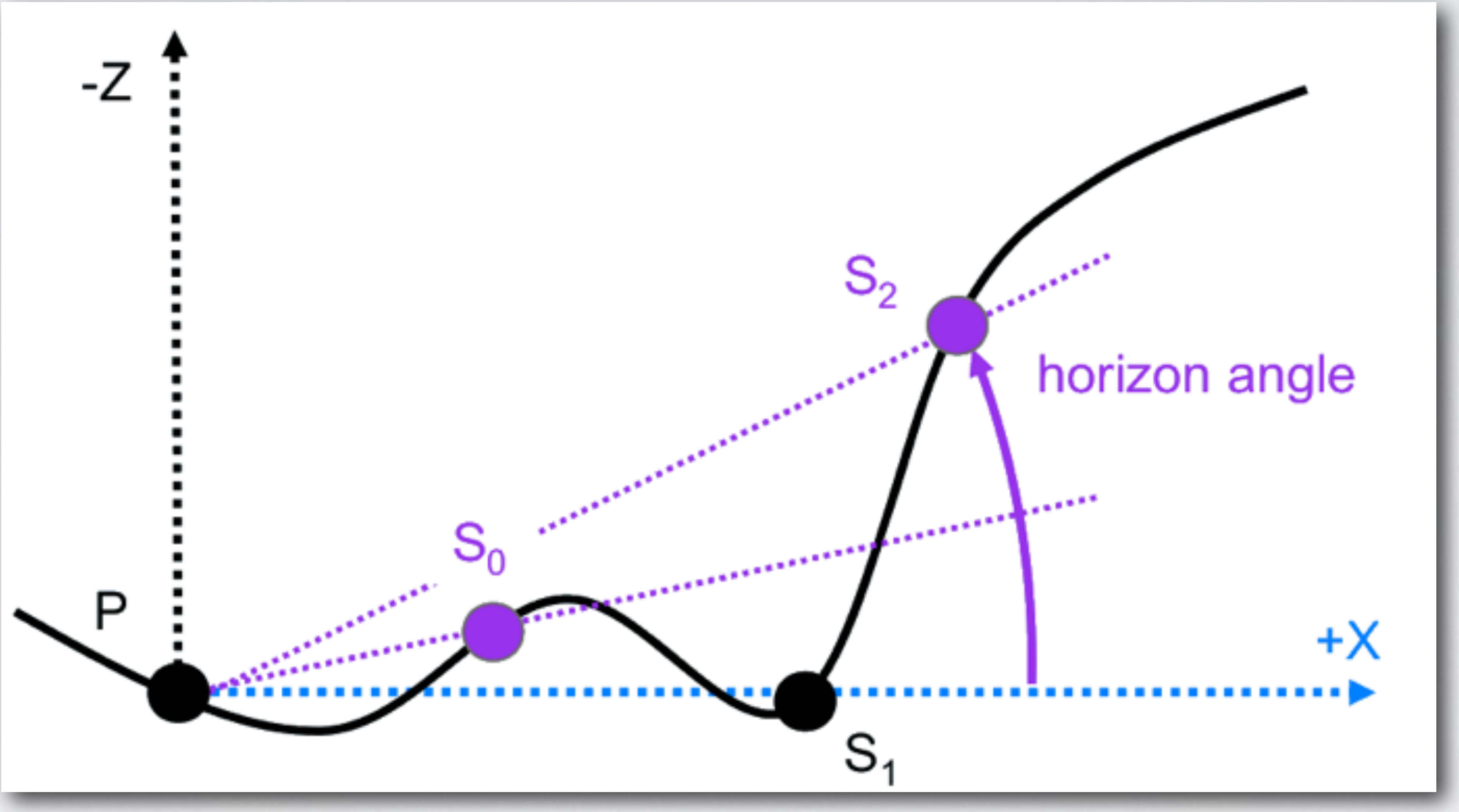


S1 not visible

3 PREVIOUS METHODS

Marching along one of the K directions

- When the horizon (from P to S_n) exceeds the previous max, the new point (S_n) is visible to P

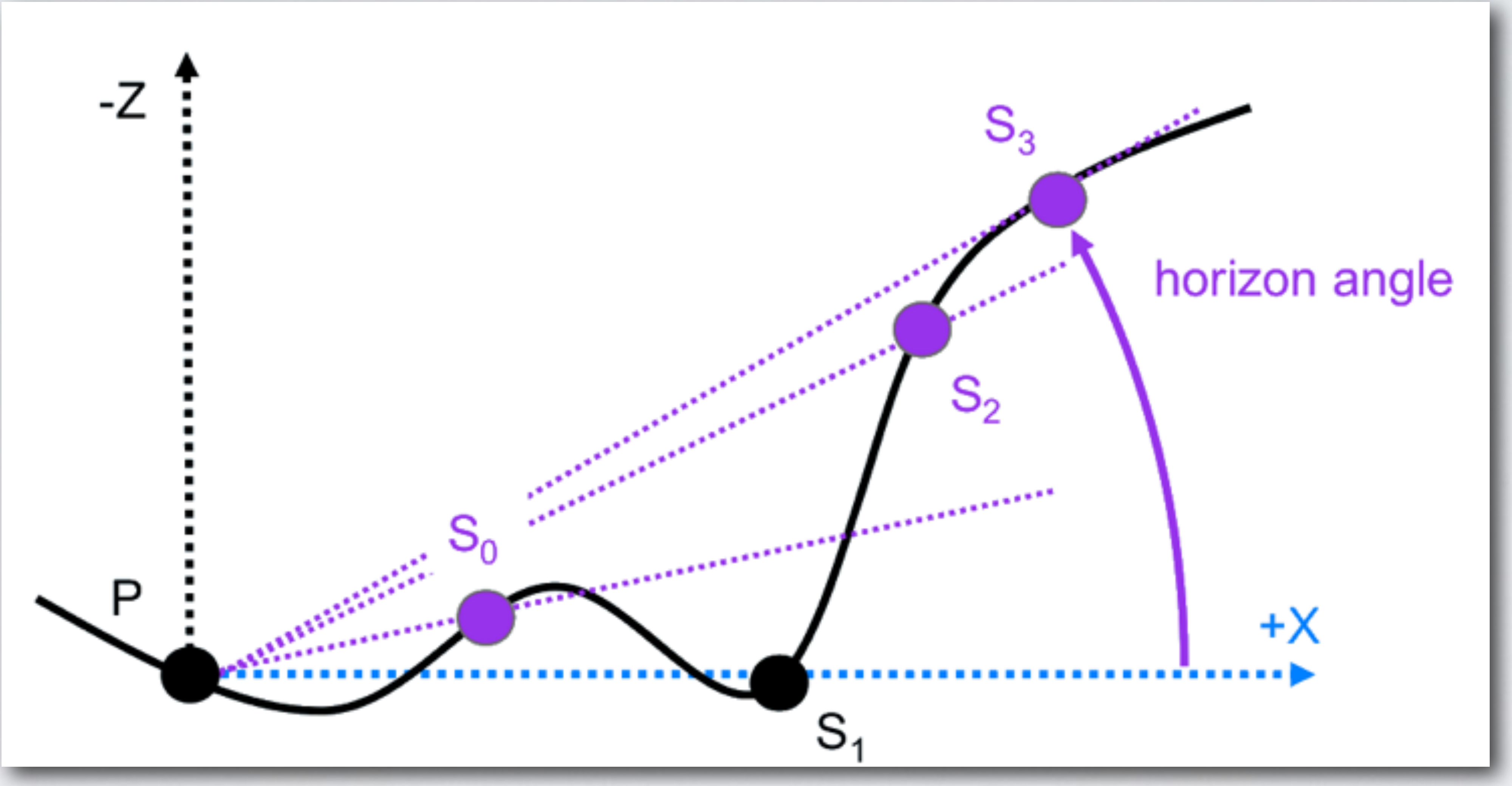


S2 visible

3 PREVIOUS METHODS

Marching along one of the K directions

- When the horizon (from P to S_n) exceeds the previous max, the new point (S_n) is visible to P

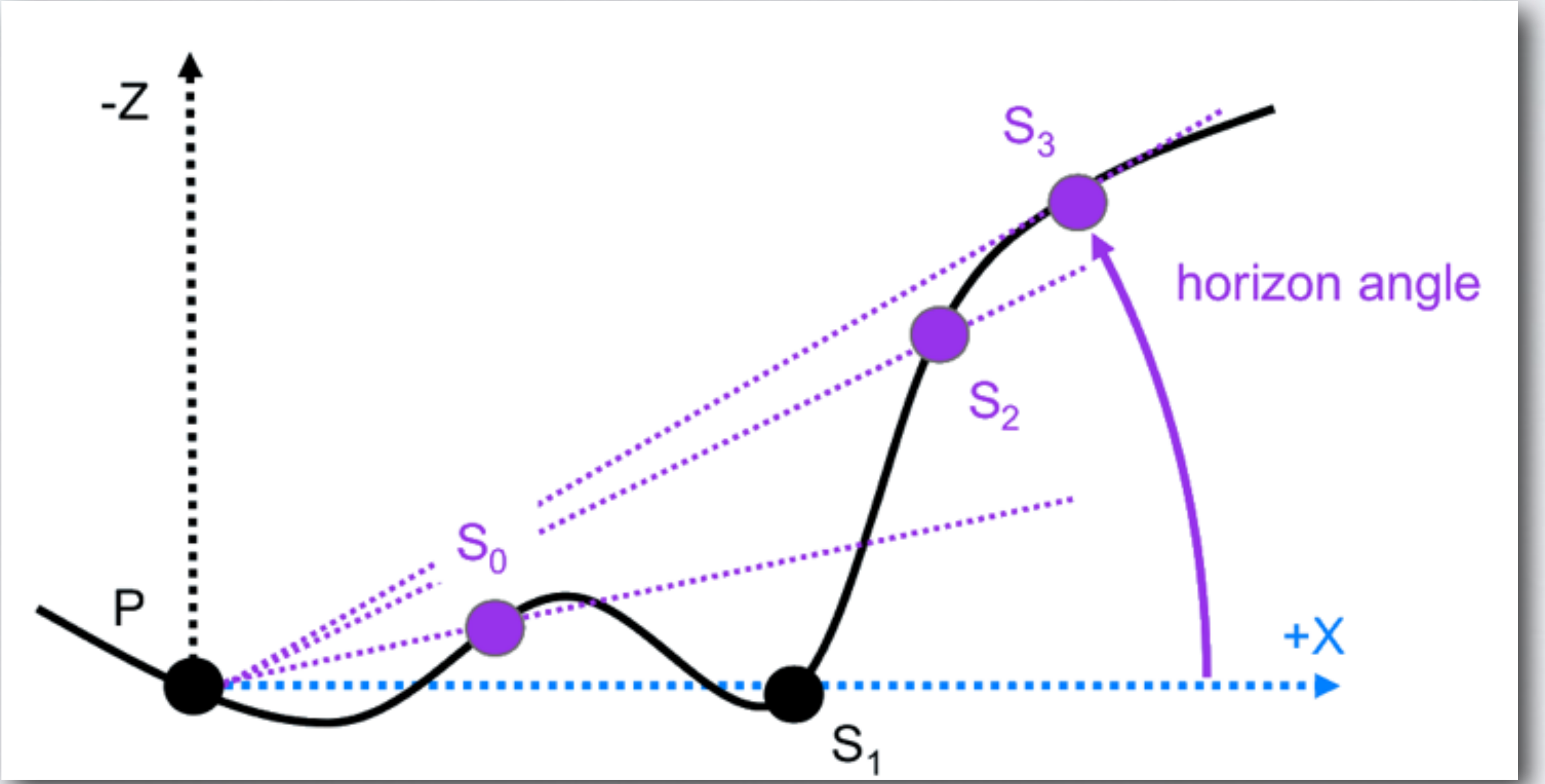


S3 visible

3 PREVIOUS METHODS

Marching along one of the K directions

- Integrate occlusion along the horizon angle piece-wise
- From a visible point to the next (tangent at P -> S_0 -> S_2 -> S_3)



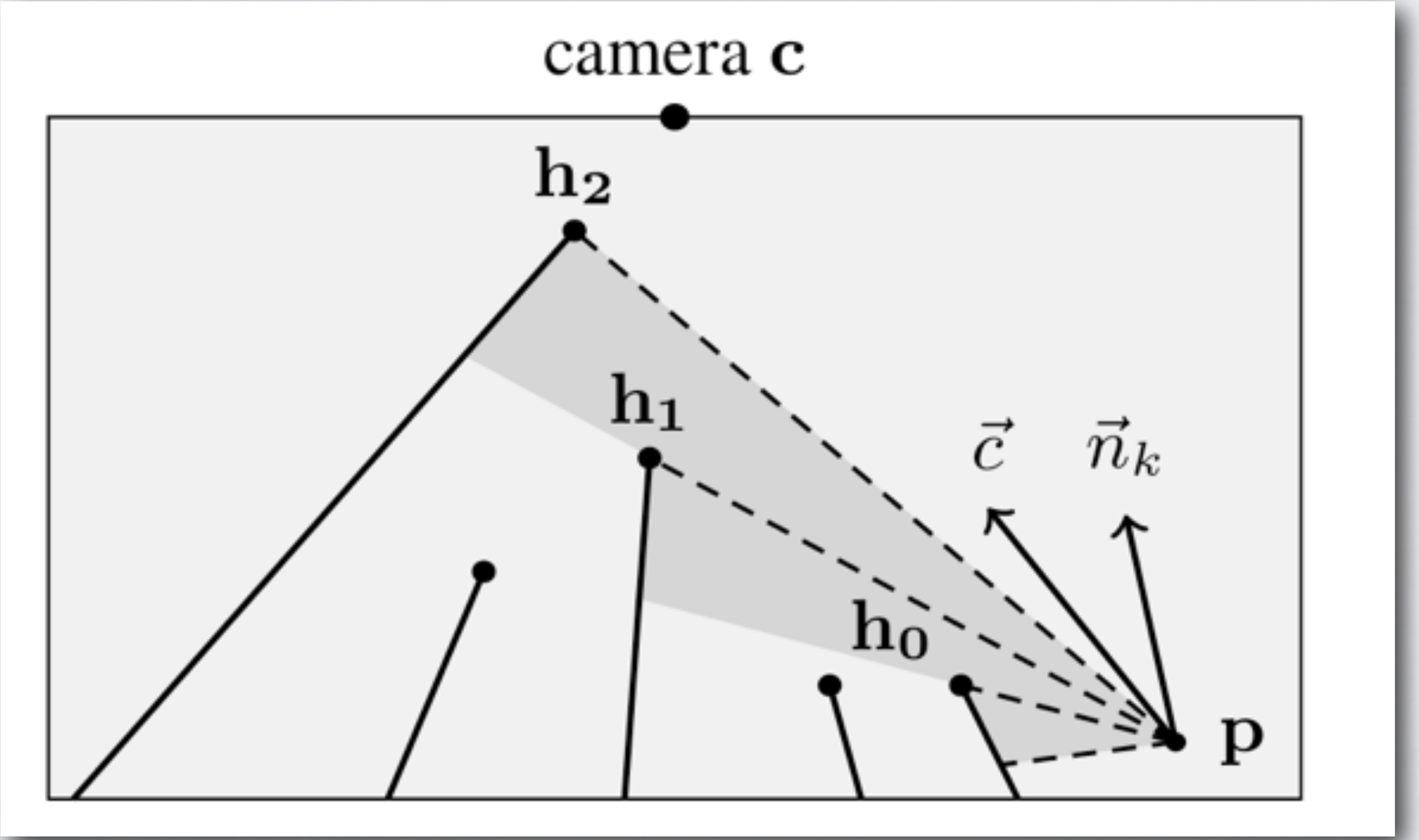
- Rinse and repeat for each K and for each pixel

3 PREVIOUS METHODS

Marching along one of the K directions

- Let h_n be a vector from P to (a visible) S_n (we call this a *horizon vector*)

- Integrate occlusion along the horizon vectors piece-wise

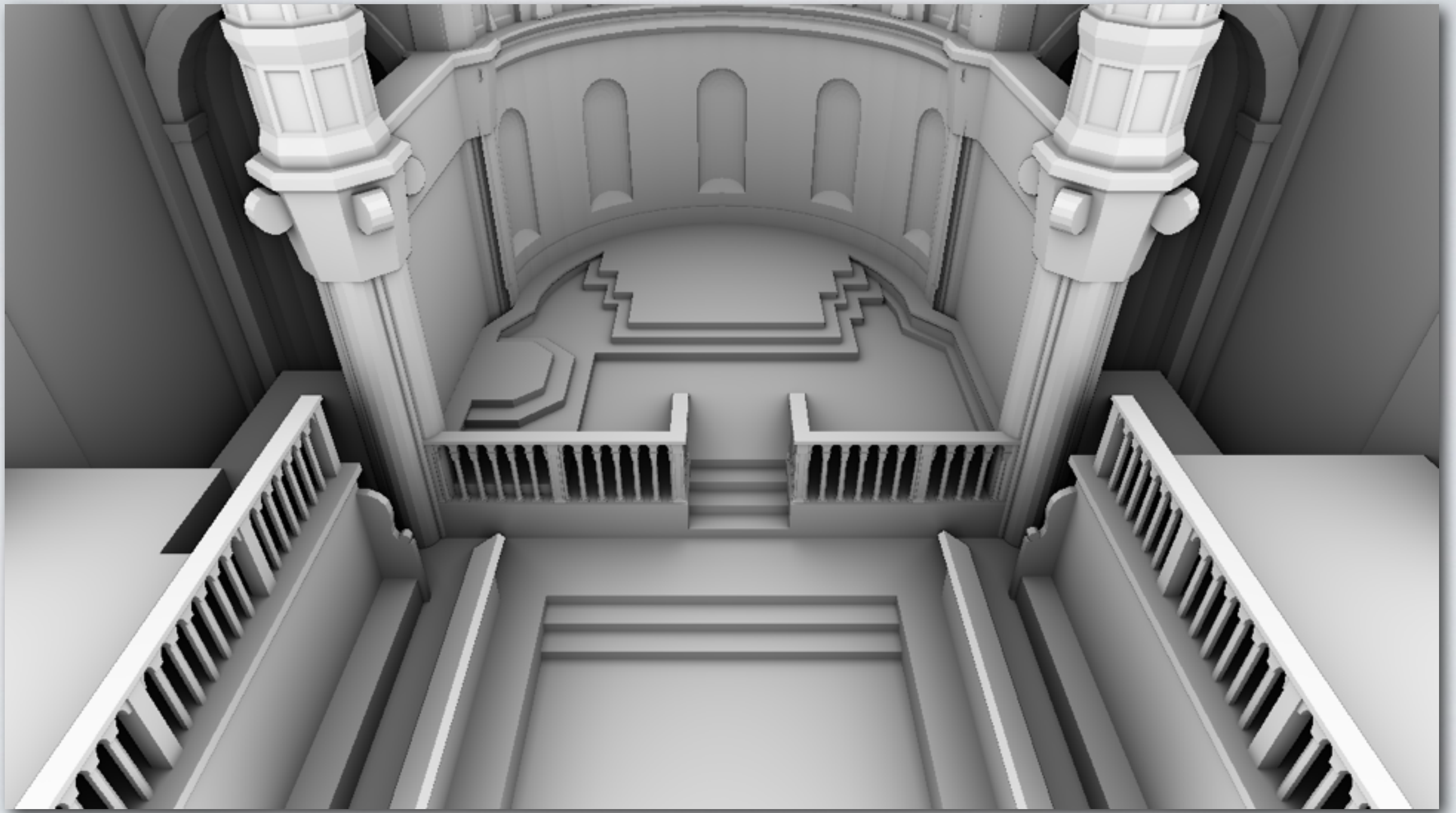


The problem

- Falloff defined in world space: AO effect can span arbitrary lengths on screen
- Often need many steps per direction before contribution has fallen enough to be cut
- Cannot afford to take hundreds of samples per direction
- *What to do?*

3 PREVIOUS METHODS

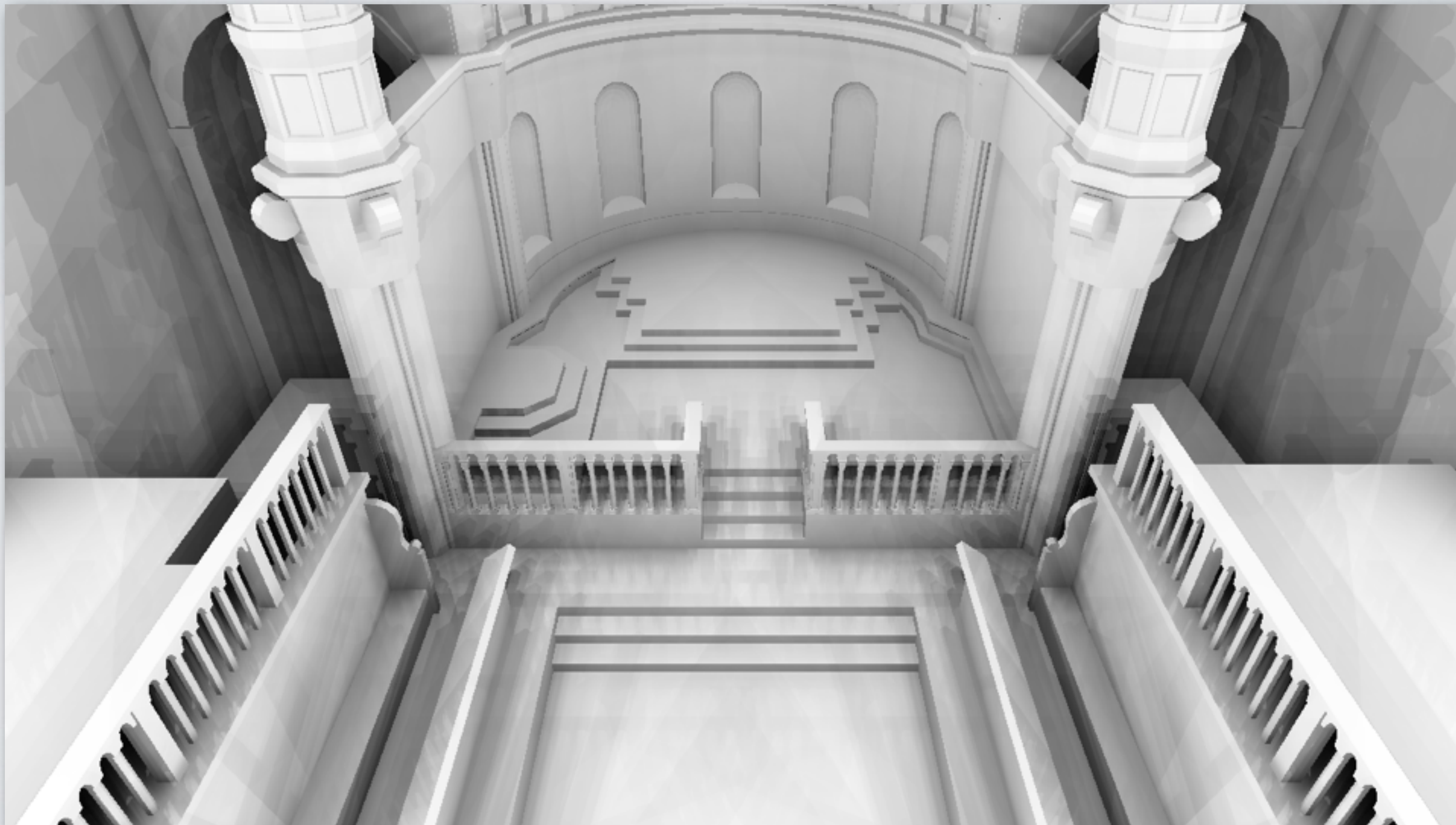
Want this



- Takes 2 seconds/frame, way too slow

3 PREVIOUS METHODS

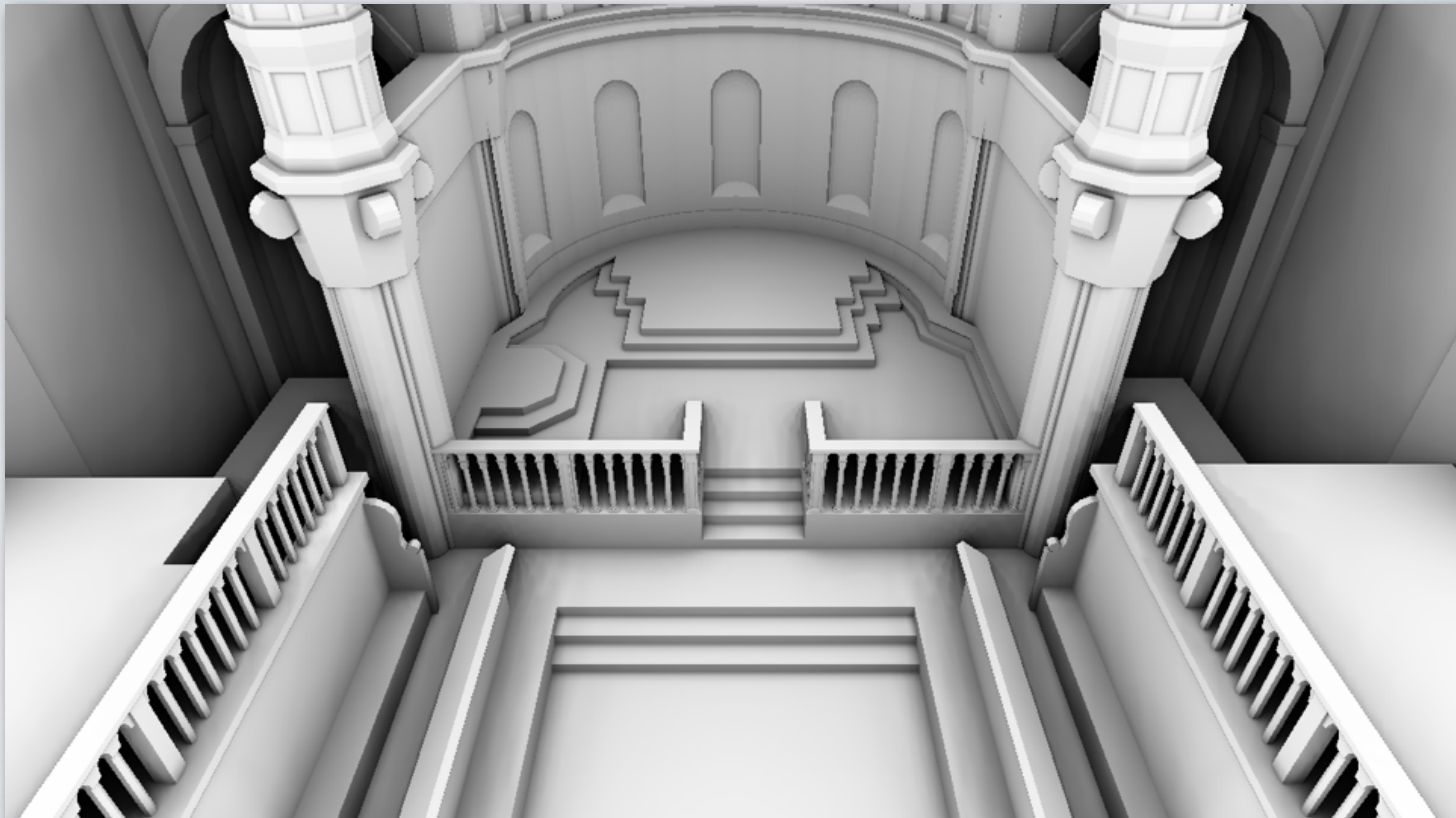
Sparser sampling farther from receiver



- Now it's fast enough, but some pixels hit occluders, some miss..

3 PREVIOUS METHODS

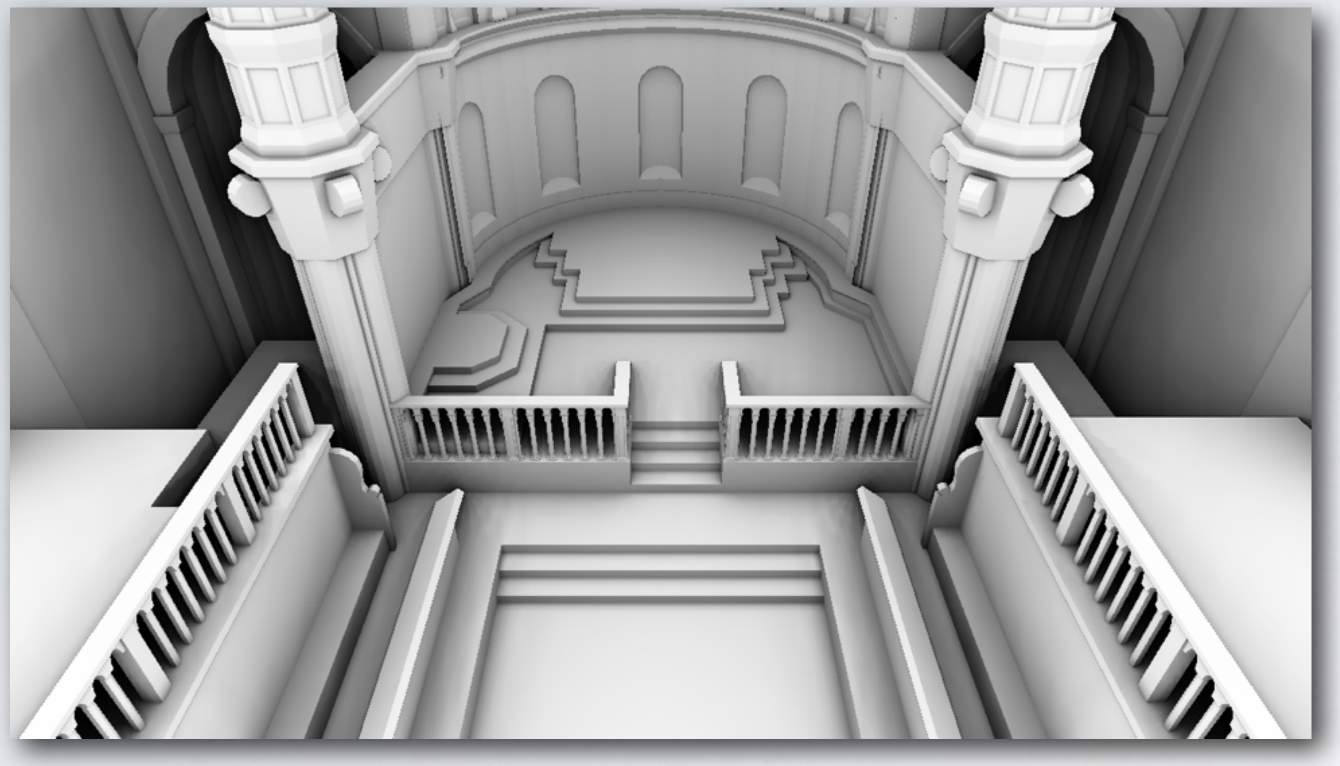
MIPMAP



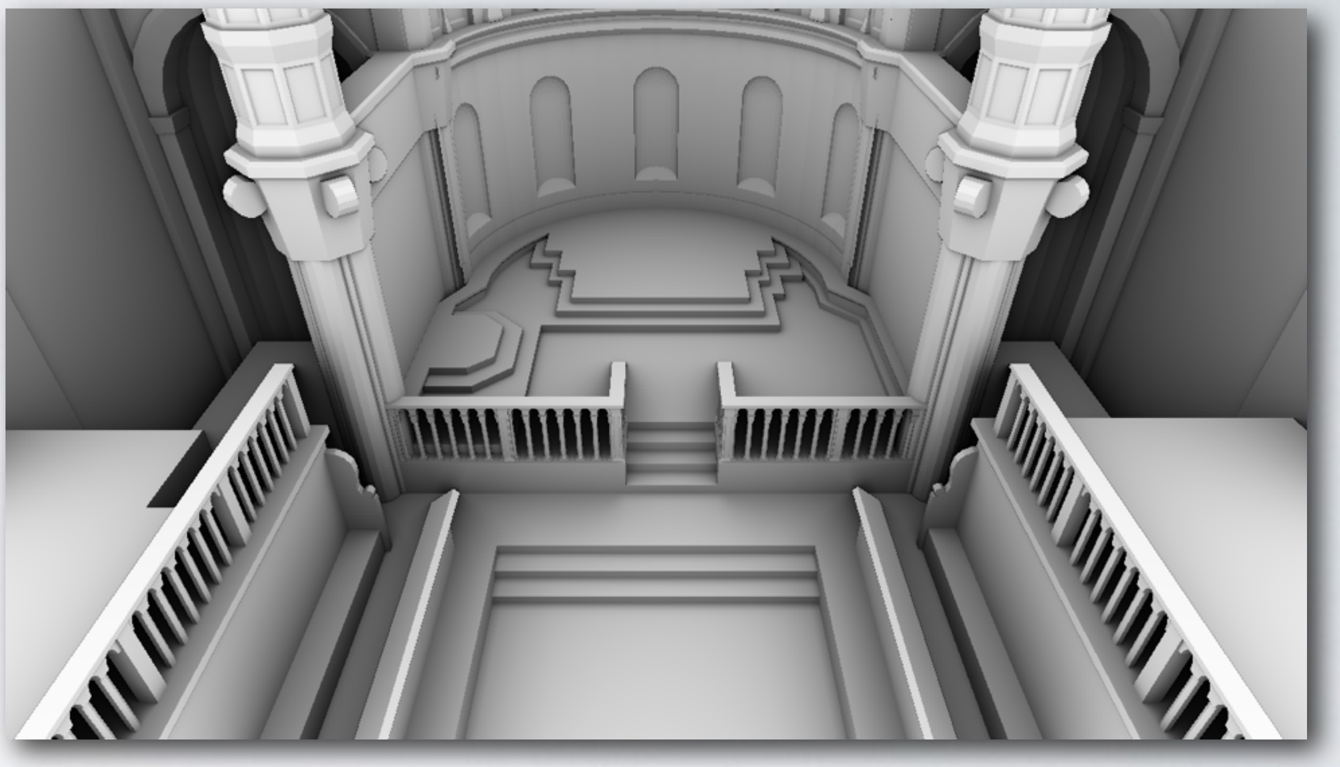
- Got rid of the blockiness, but...

3 PREVIOUS METHODS

MIPMAP



MIPMAP



reference

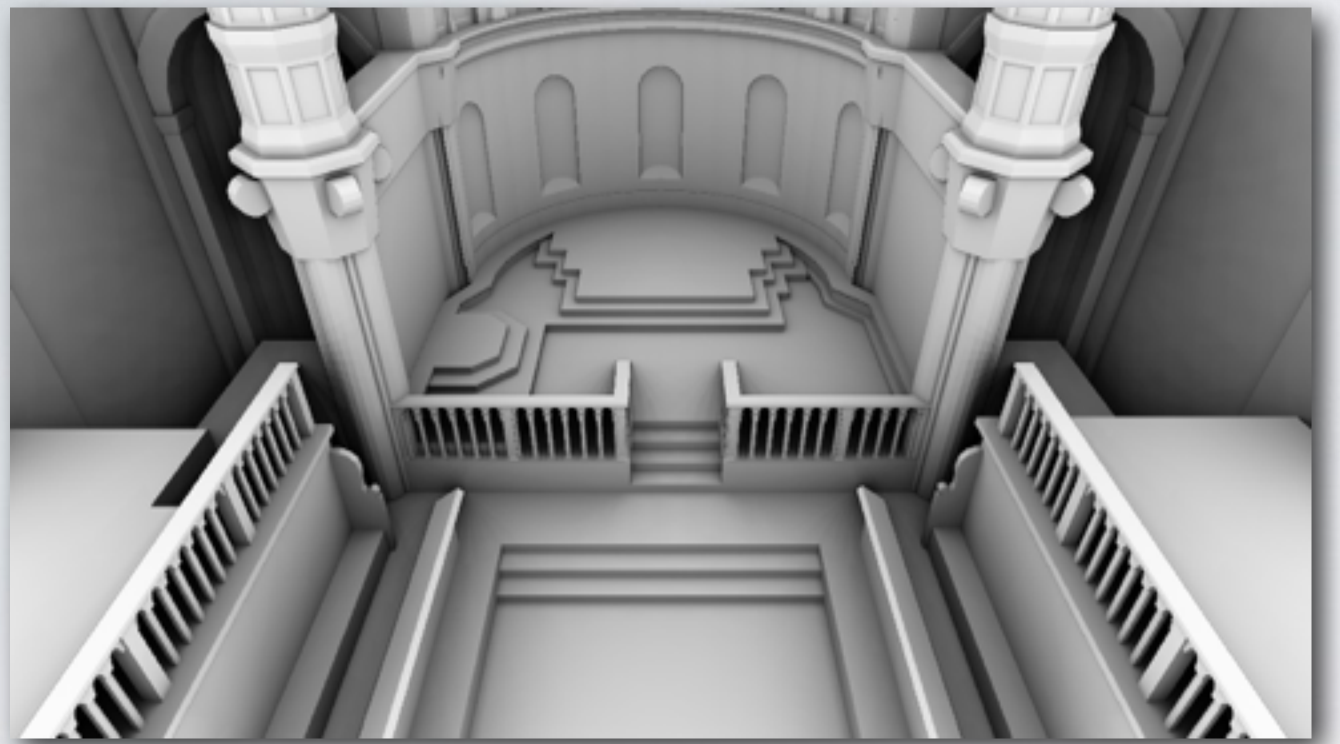
- No longer artefacts, but systematic underocclusion

Error
black = 20%
white = 0%

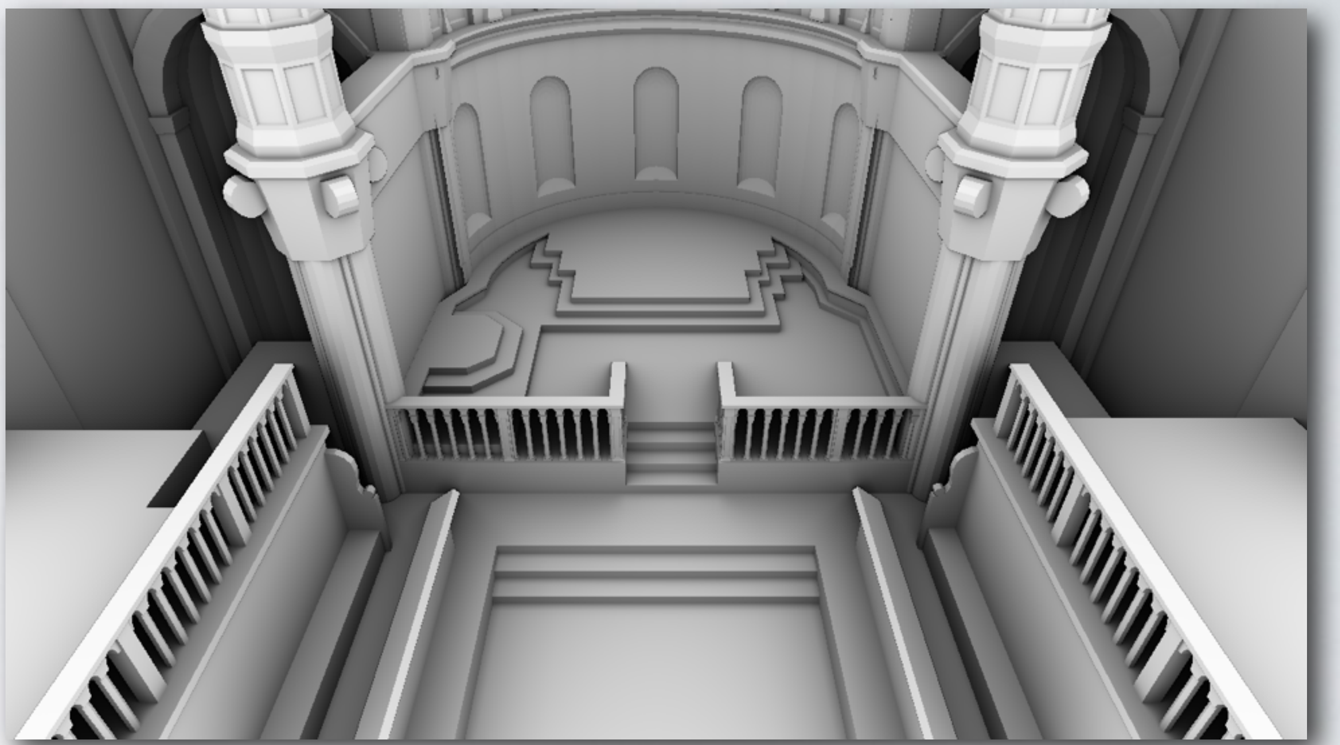
CONTENTS

4 OUR METHOD

Teaser



our method



reference

- Same number of samples as in MIPMAP, but significantly closer to truth

Error
black = 20%
white = 0%

4 OUR METHOD

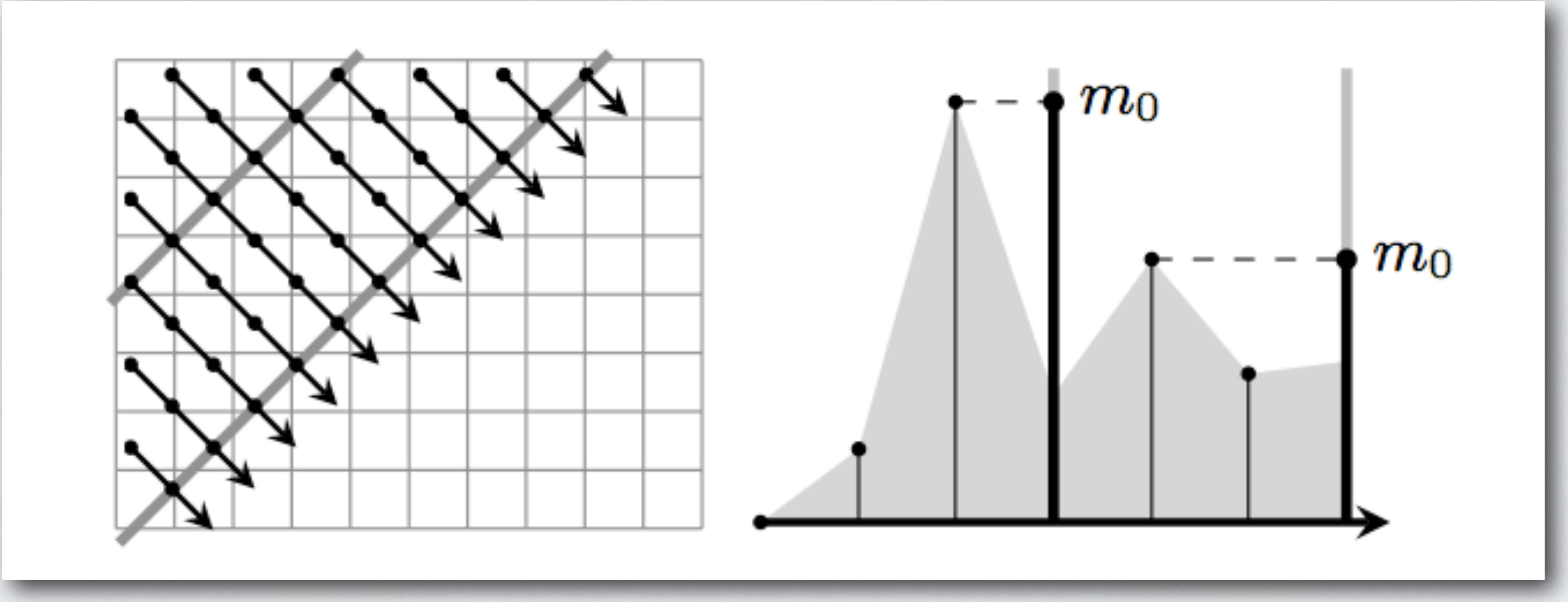
Otherwise the same except the data used for sampling...

- MIPMAPs flatten the geometry
- Instead, we would like to retain is the *silhouette* of the geometry as seen from any receiver point
- Silhouette is formed by *local maxima* of the height field
- Let's start with that...

4 OUR METHOD

We generate an intermediate geometry proxy

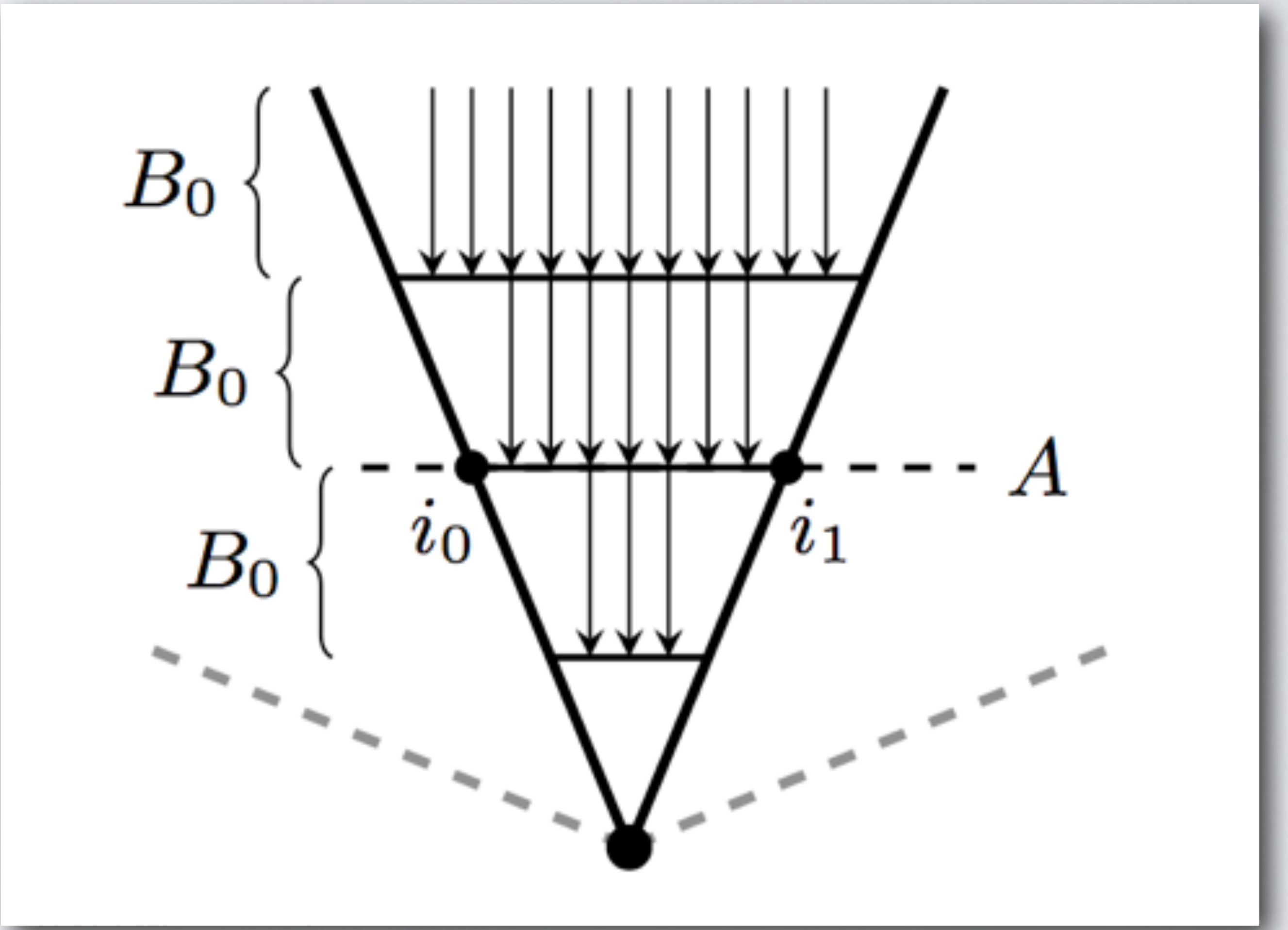
- Traverse the height field in parallel lines, a step at a time
- Every B_0 steps (here B_0=3) we write out the highest value on the line



4 OUR METHOD

We generate an intermediate geometry proxy

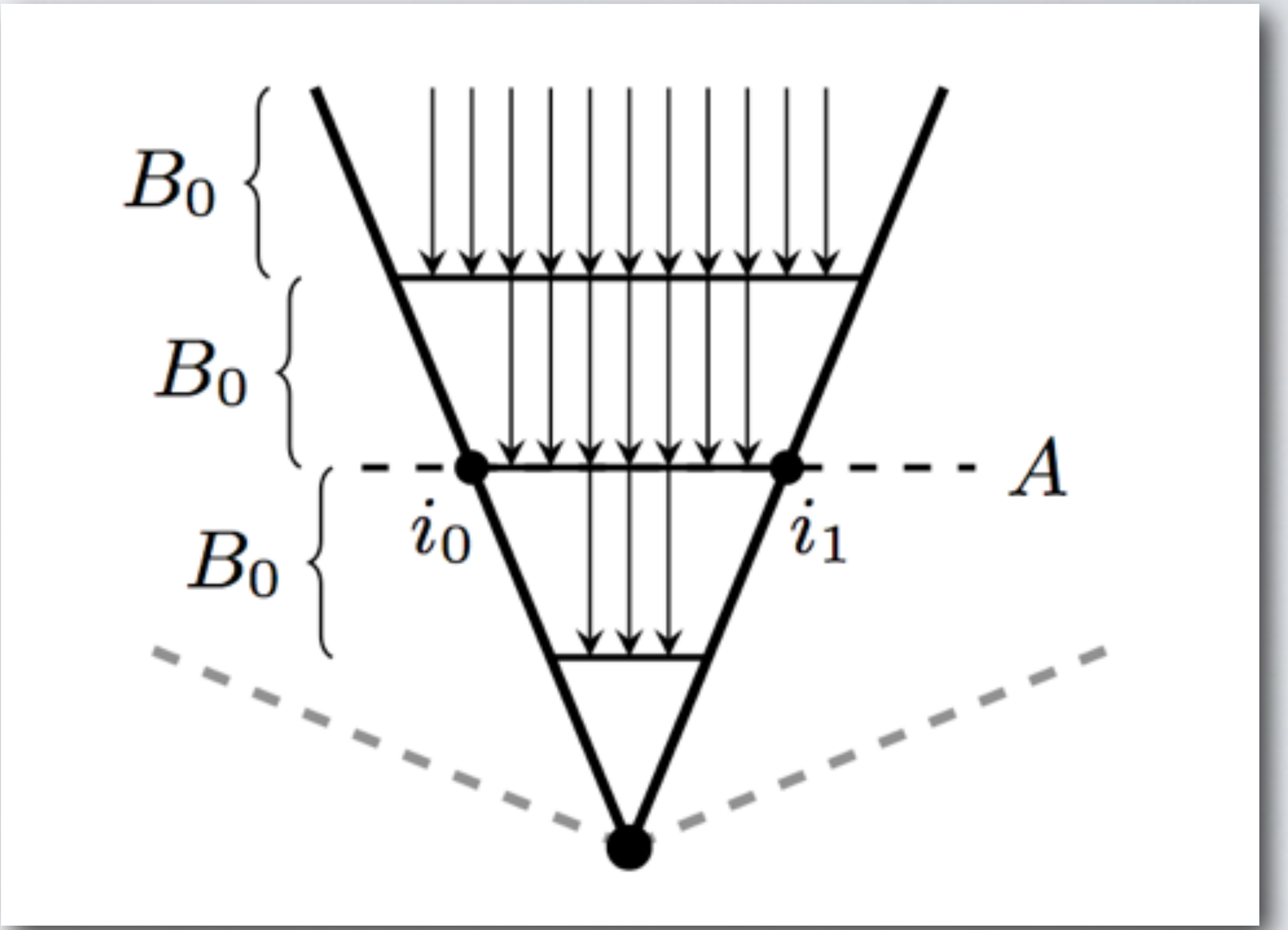
- Recall that we split the 2D integral into K 1D integrals
- Each of these should represent the entire sector (2*PI/K)
- So instead of sampling the maximum heights along one line, want to take average maximum height along the sector's width



4 OUR METHOD

We generate an intermediate geometry proxy

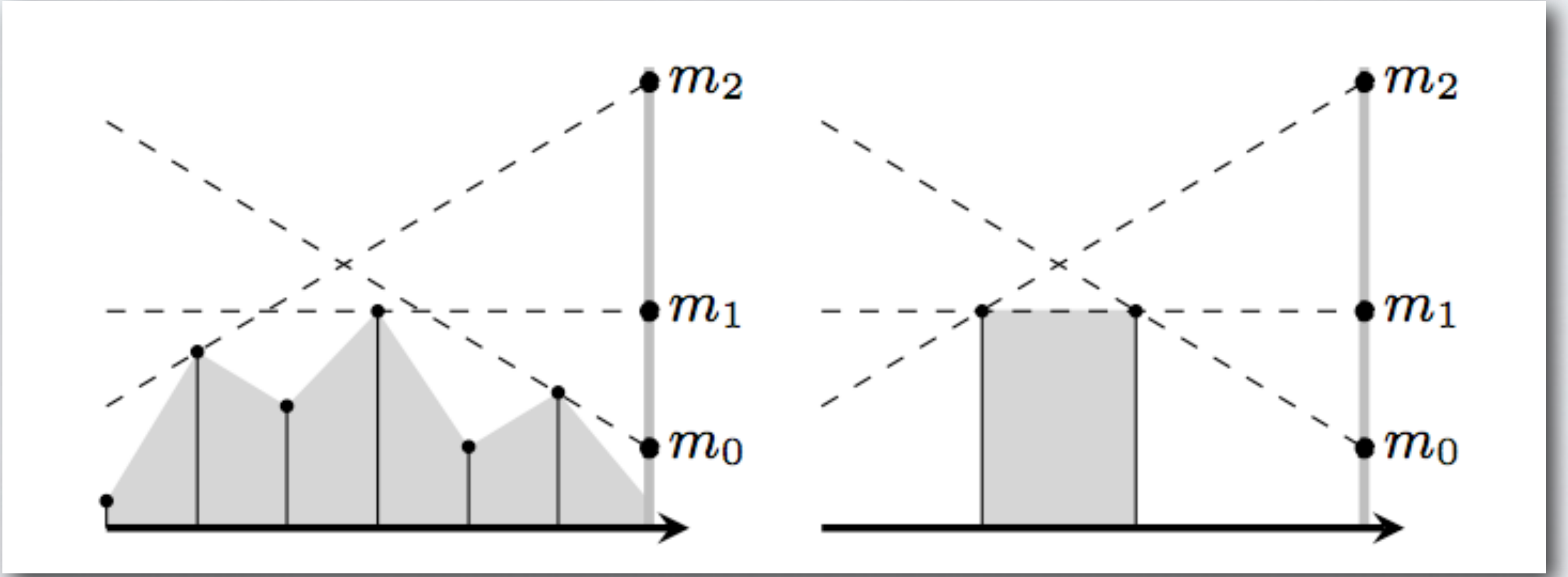
- Calculate running sums of the maximum heights
- Getting the average becomes $(A[i_1]-A[i_0])/(i_1-i_0)$



4 OUR METHOD

Multi-view

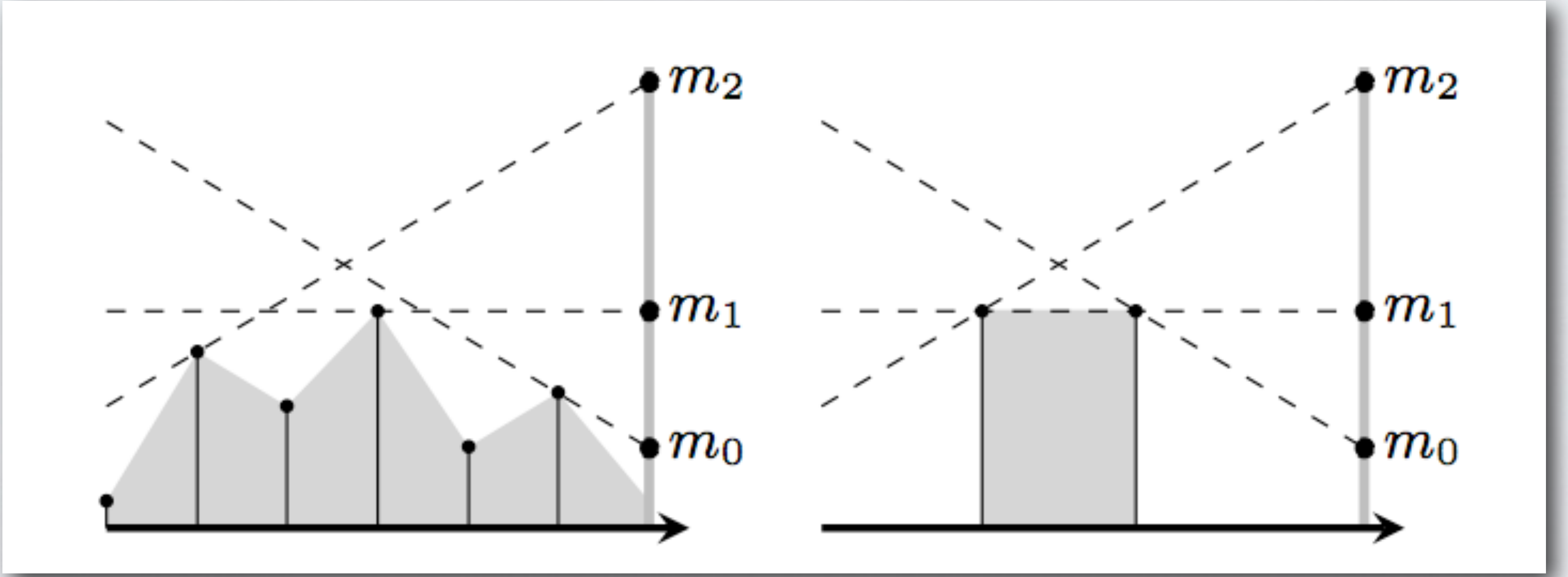
- Maximum heights represent silhouette only when the receiver is horizontal to the caster
- In addition, we can project maximum height along multiple viewing directions (left)



4 OUR METHOD

Multi-view

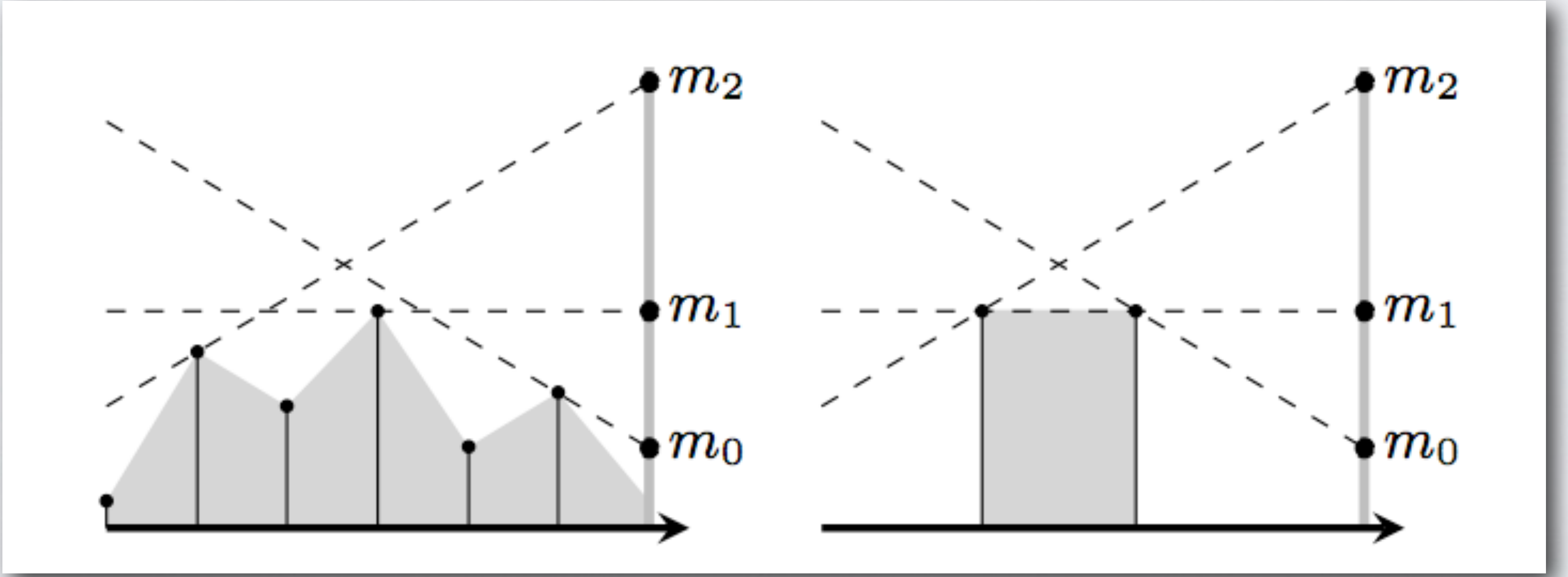
- This has an alternative interpretation: Intersections of the projections describe a convex hull of the geometry (right)
- Edges of the convex hull can be used as the endpoints (S_n) of the horizon vectors (h_n)



4 OUR METHOD

Multi-view

- Of particular interest is the case of 2 viewing directions
- The convex hull is reduced to a single point
- Can be used directly as the horizon vector end point S_n



4 OUR METHOD

Level of detail

- We generate multiple resolutions of the projections
- Differ in the range the max is taken over of
- Combined by maxing higher resolutions
- Used when sampling farther from the receiver

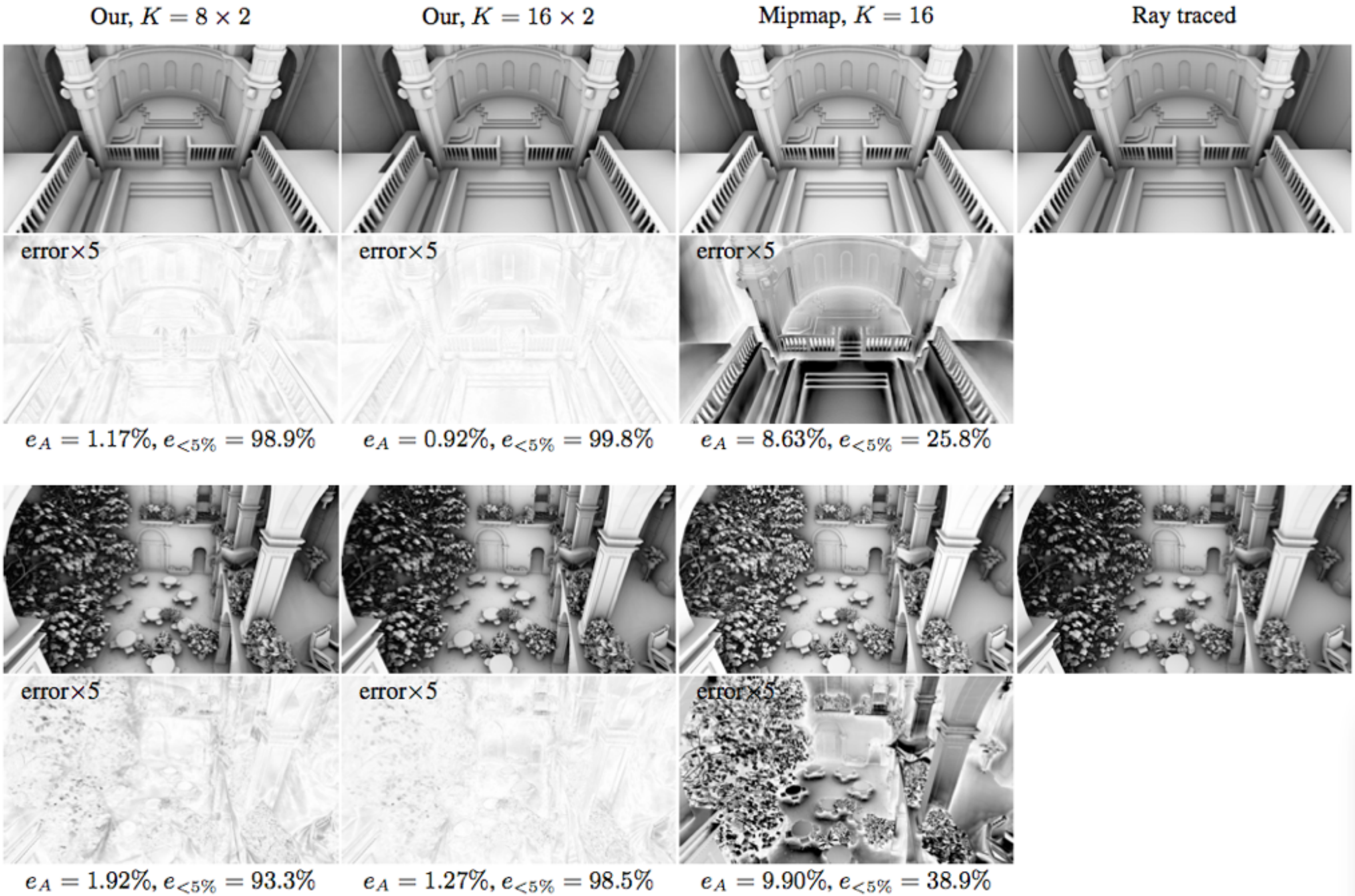
CONTENTS

5 RESULTS

Our method

Reference

5 RESULTS



Our, $K = 8 \times 2$
Our, $K = 16 \times 2$
Mipmap, $K = 16$
Ray traced
error×5
error×5
error×5
$e_A = 1.17\%, e_{<5\%} = 98.9\%$
$e_A = 0.92\%, e_{<5\%} = 99.8\%$
$e_A = 8.63\%, e_{<5\%} = 25.8\%$
error×5
error×5
error×5
$e_A = 1.92\%, e_{<5\%} = 93.3\%$
$e_A = 1.27\%, e_{<5\%} = 98.5\%$
$e_A = 9.90\%, e_{<5\%} = 38.9\%$

5 RESULTS

Table 1: *Total render times of the far-field occlusion component*

Method	7970 (OpenCL)	GTX 580 (CUDA)
$1280(+256) \times 720(+144)$, $B_0 = 10$:		
Our, $K = 8 \times 2$	7.26 ms	12.0 ms
Our, $K = 16 \times 2$	13.3 ms	23.6 ms
Mipmap, $K = 16$	19.2 ms	17.7 ms
$1920(+384) \times 1080(+216)$, $B_0 = 10$:		
Our, $K = 8 \times 2$	16.7 ms	29.4 ms
Our, $K = 16 \times 2$	31.6 ms	58.1 ms
Mipmap, $K = 16$	31.5 ms	37.9 ms

Roughly as fast as the MIPMAP method