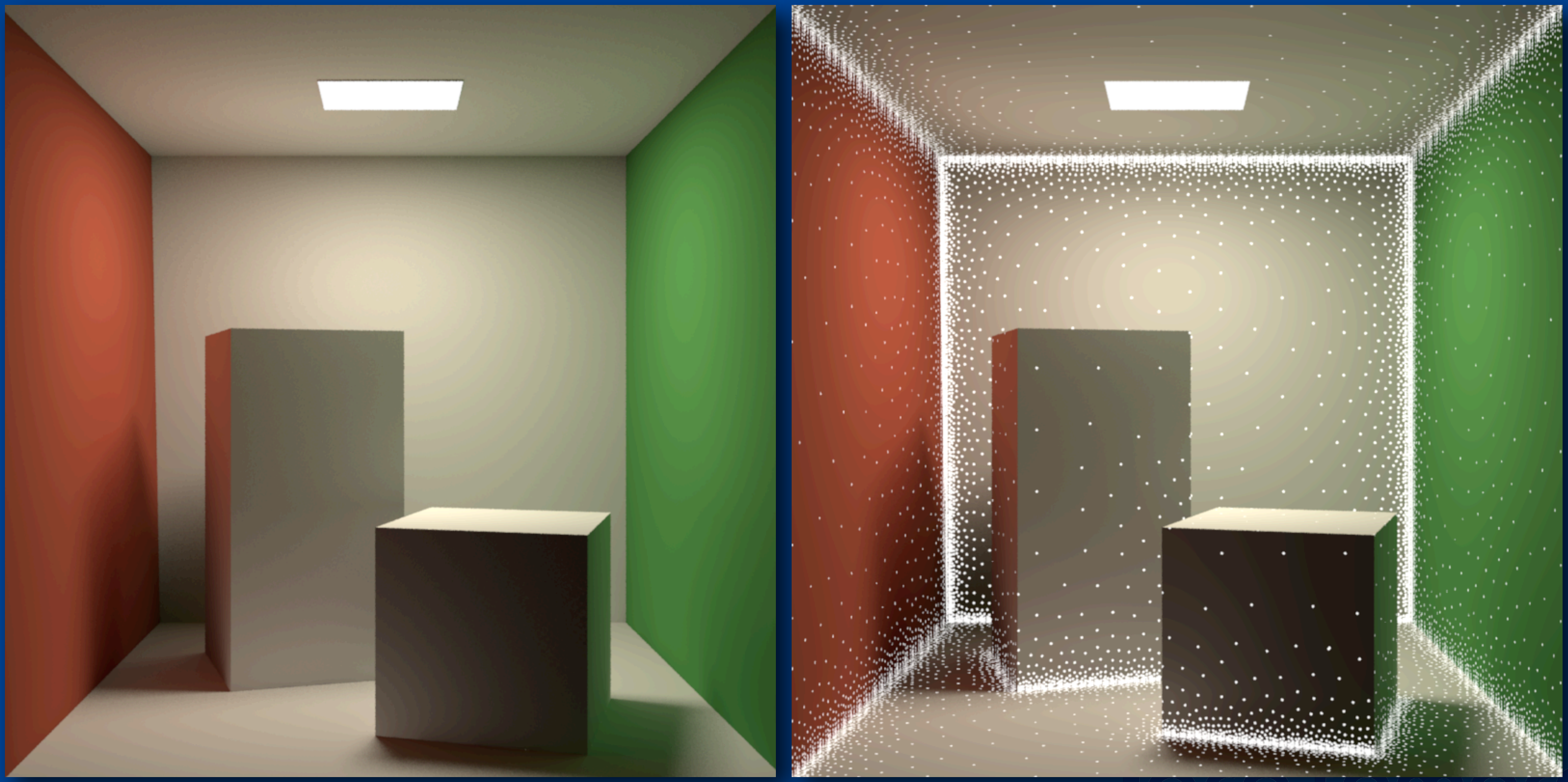
SIGGRAPH2007

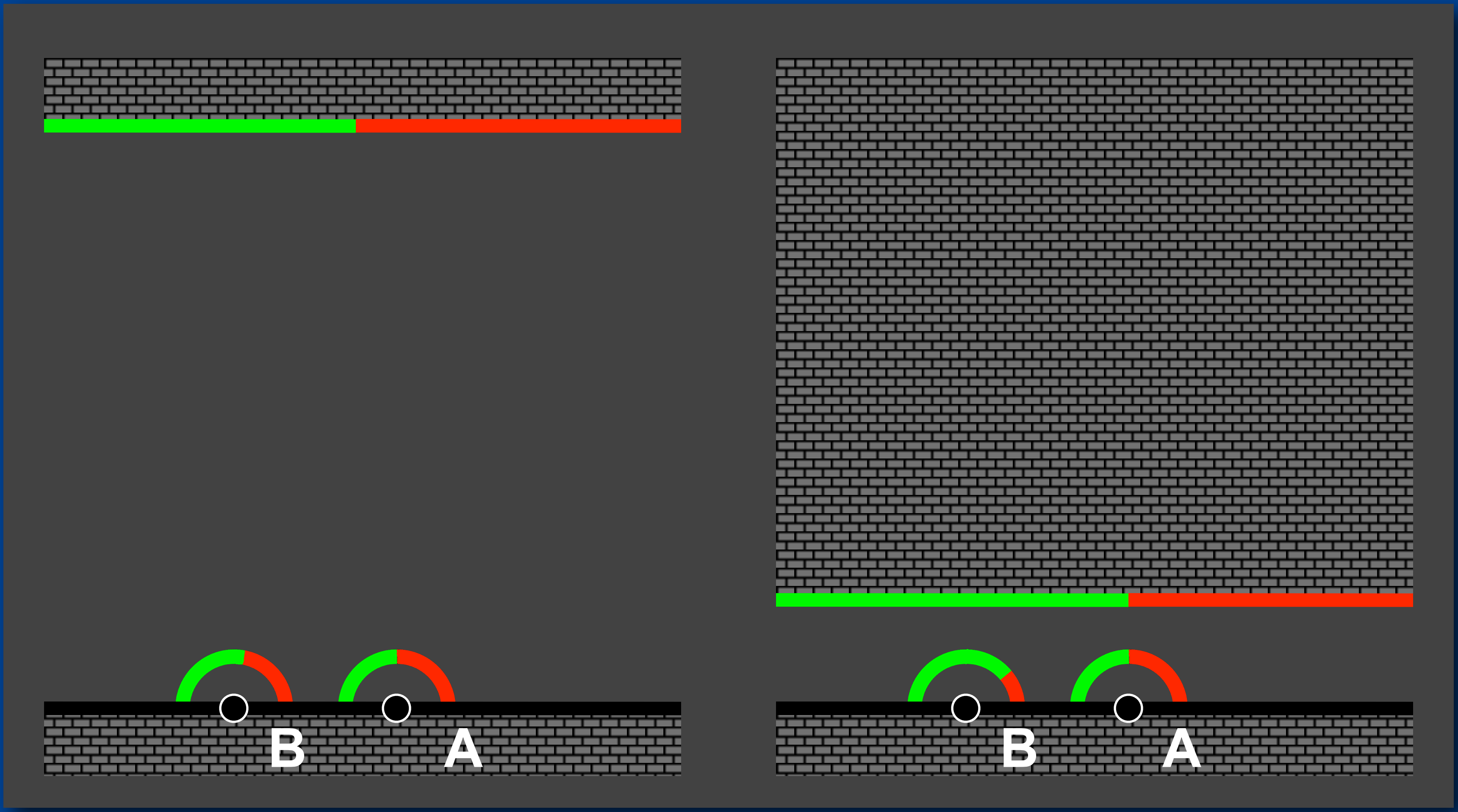
Previous Work

- Irradiance caching
  - Samples capture high frequency content



•Here’s an example of irradiance caching
•The picture on the right shows the sample locations
•Notice that samples are concentrated around the corners because the indirect illumination can change quickly around these regions
•The reason we sample densely around corners, or high geometric detail is to capture rapid changes in global illumination

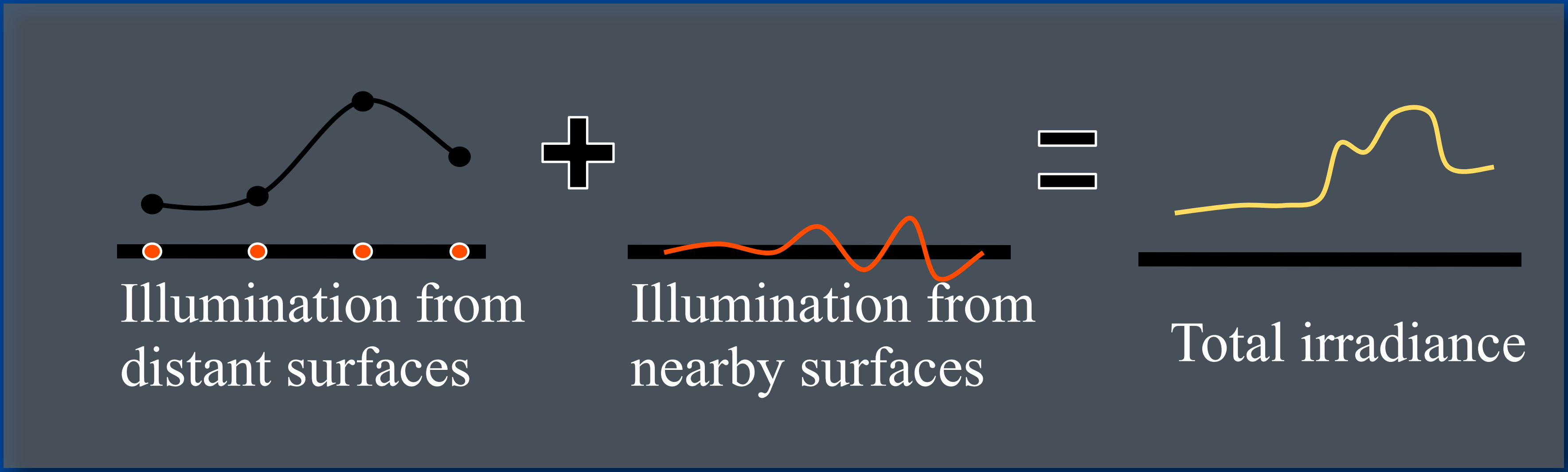
Observation



•You can see why we need denser sampling around corners and high geometric detail by visualizing this scene
•As the ceiling is far away, when we go from point B to point A, their incident hemispheres do not change much
•If the ceiling was lower, then what B sees can be very different from what A sees
•This is why irradiance caches must sample densely around high geometric detail

Our Approach

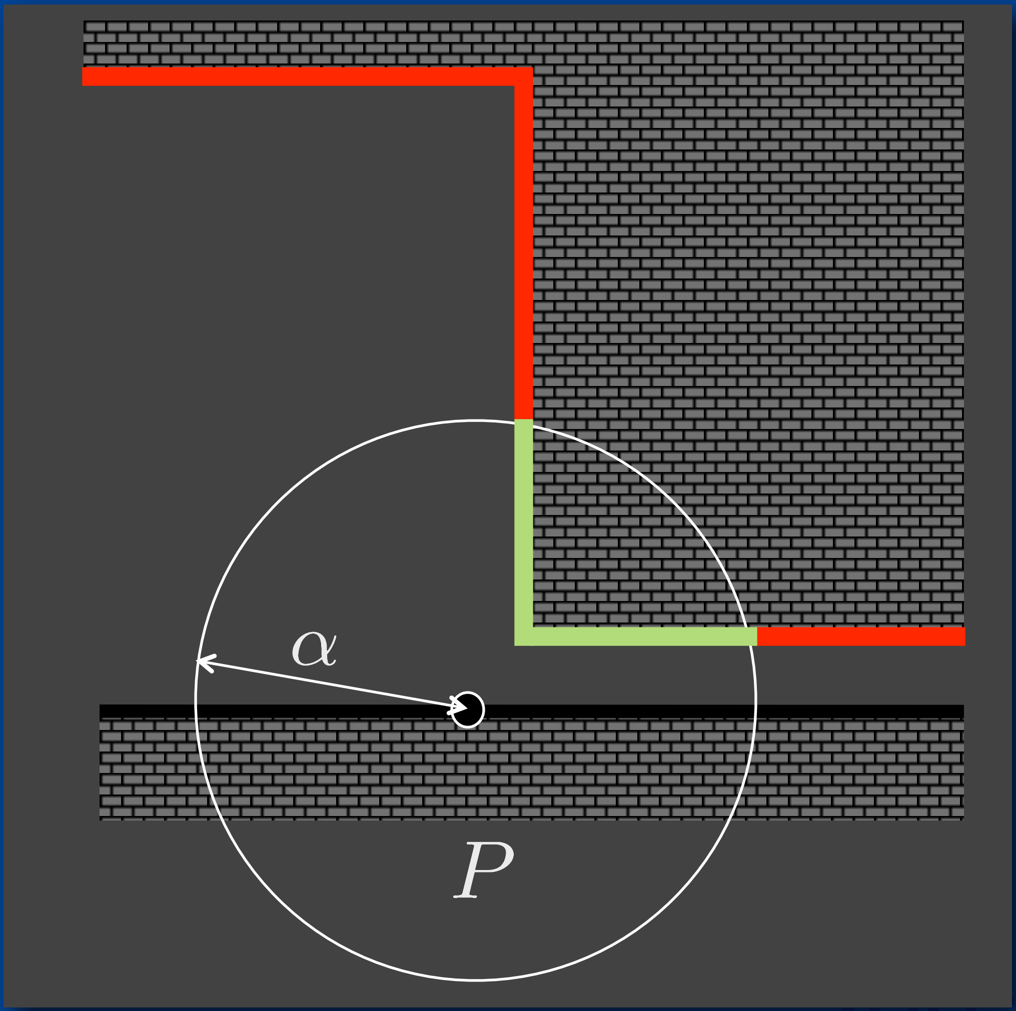
- Decompose Irradiance:
  - Slowly changing term
  - Rapidly changing term



Illumination from distant surfaces
Illumination from nearby surfaces
Total irradiance

•In this part of the course, I propose decomposing irradiance into two terms
•Irradiance coming from **distant** surfaces
•And irradiance coming from **nearby** surfaces
•Because of our observation, irradiance coming from distant surfaces is smooth and can be interpolated using scattered data interpolation
•The irradiance coming from nearby surfaces has high frequencies
•Fortunately because this is mainly a local phenomena, there is a way to approximate this high frequency component efficiently
•I will focus on diffuse surfaces, although I imagine this method can be incorporated into Jarda's radiance caching mechanism for more complicated BRDFs

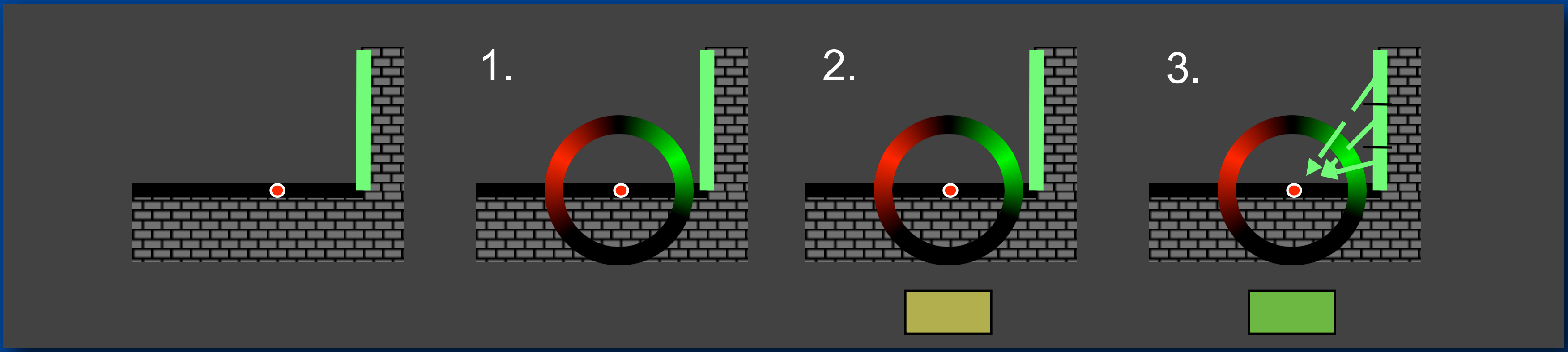
Near – Far Distinction



- •Before we go further, we need a definition for what's near and what's far
- •Given a point P, I will assume all the surfaces within the sphere centered at P with radius alpha are near
- •In this figure, green surfaces are near and red ones are far
- •Although this may seem like an arbitrary distinction, it has an intuitive meaning which will become clear in a few slides

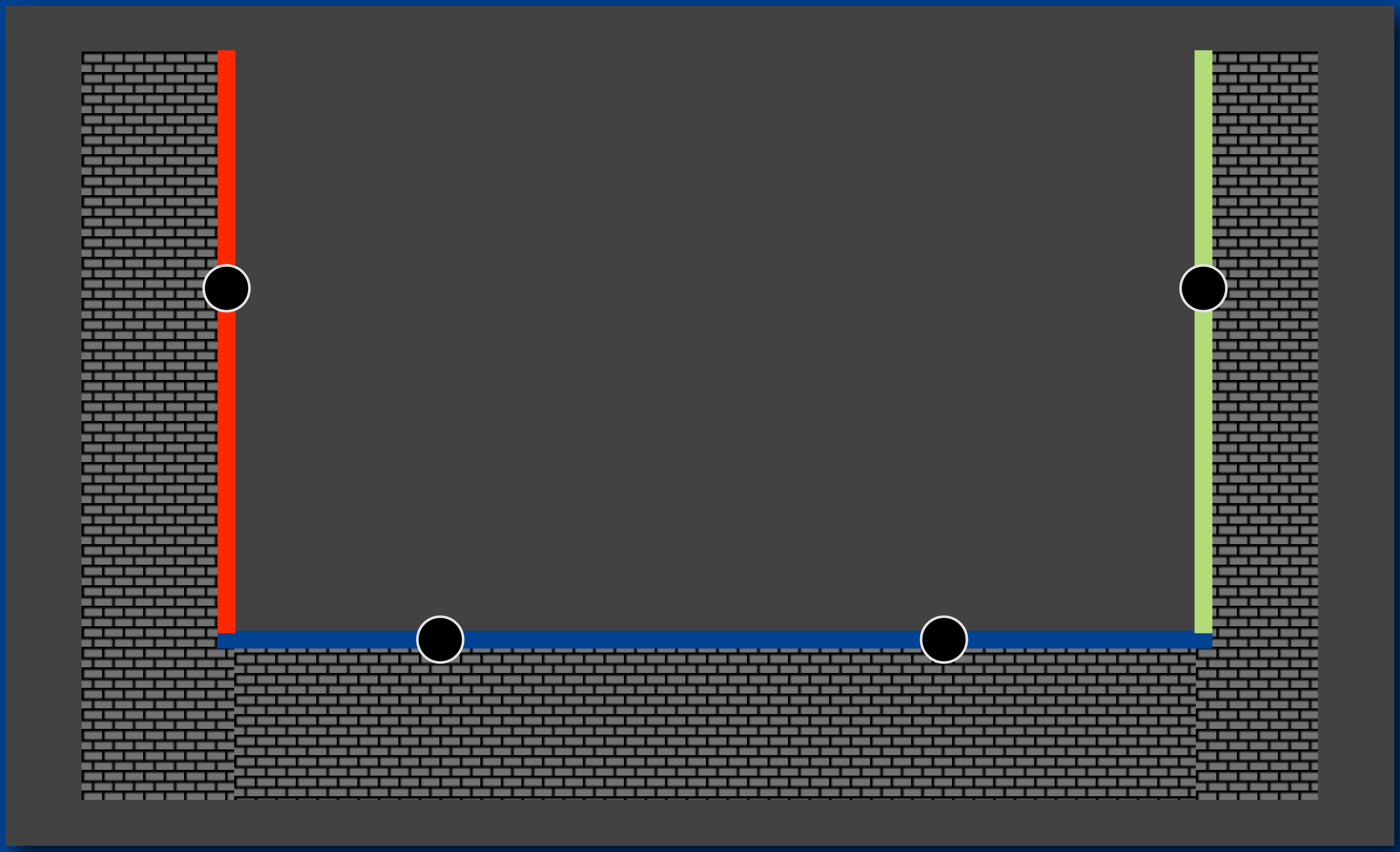
Overview

1. Compute distant incident radiance
2. Estimate the total irradiance
3. Update the irradiance using nearby triangles:
   - Add the power coming from triangle
   - Subtract the power we thought was coming from the triangle



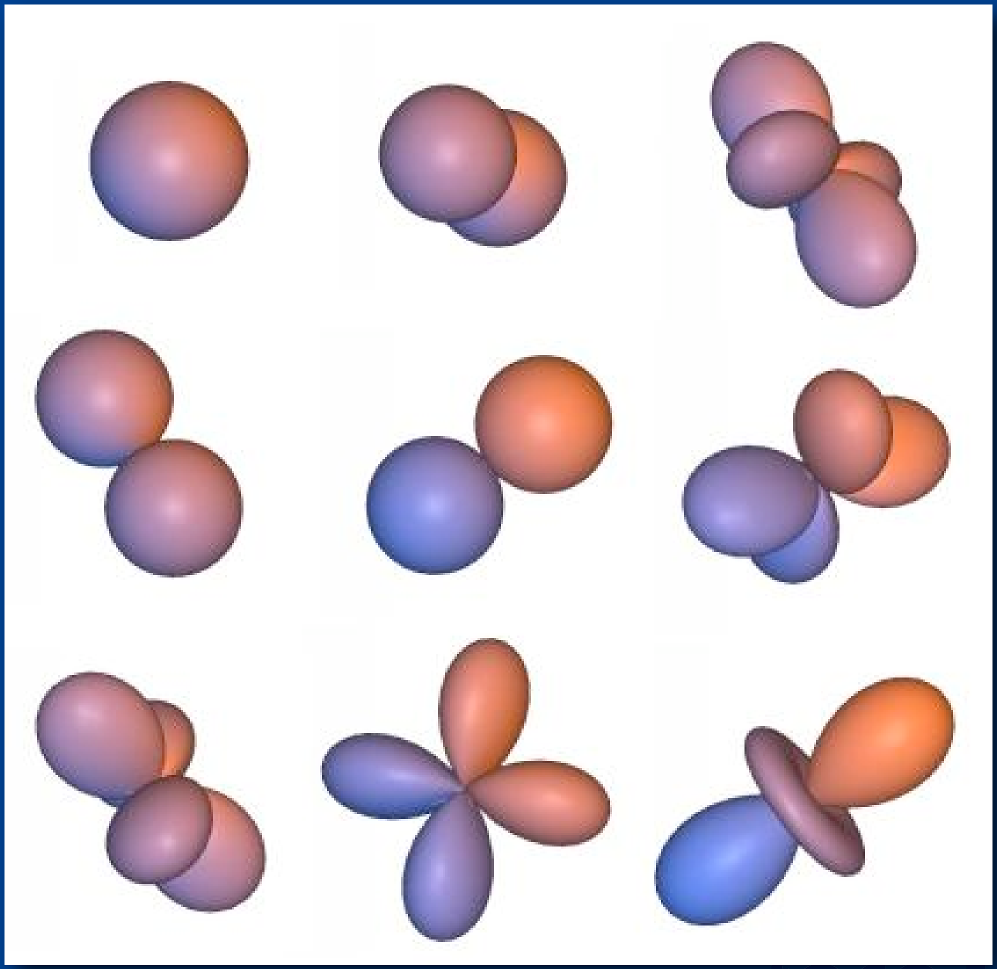
•To compute the irradiance integral at a point, our algorithm follows three steps
•1- We compute a representation of the incident radiance from only distant surfaces, so nearby surfaces are not accounted for
•2- We compute the irradiance integral over this representation
•3- We go over nearby scene triangles and update the irradiance integral by taking the power that they reflect or occlude into account
•Let's go over these steps in detail

Incident Distant Radiance



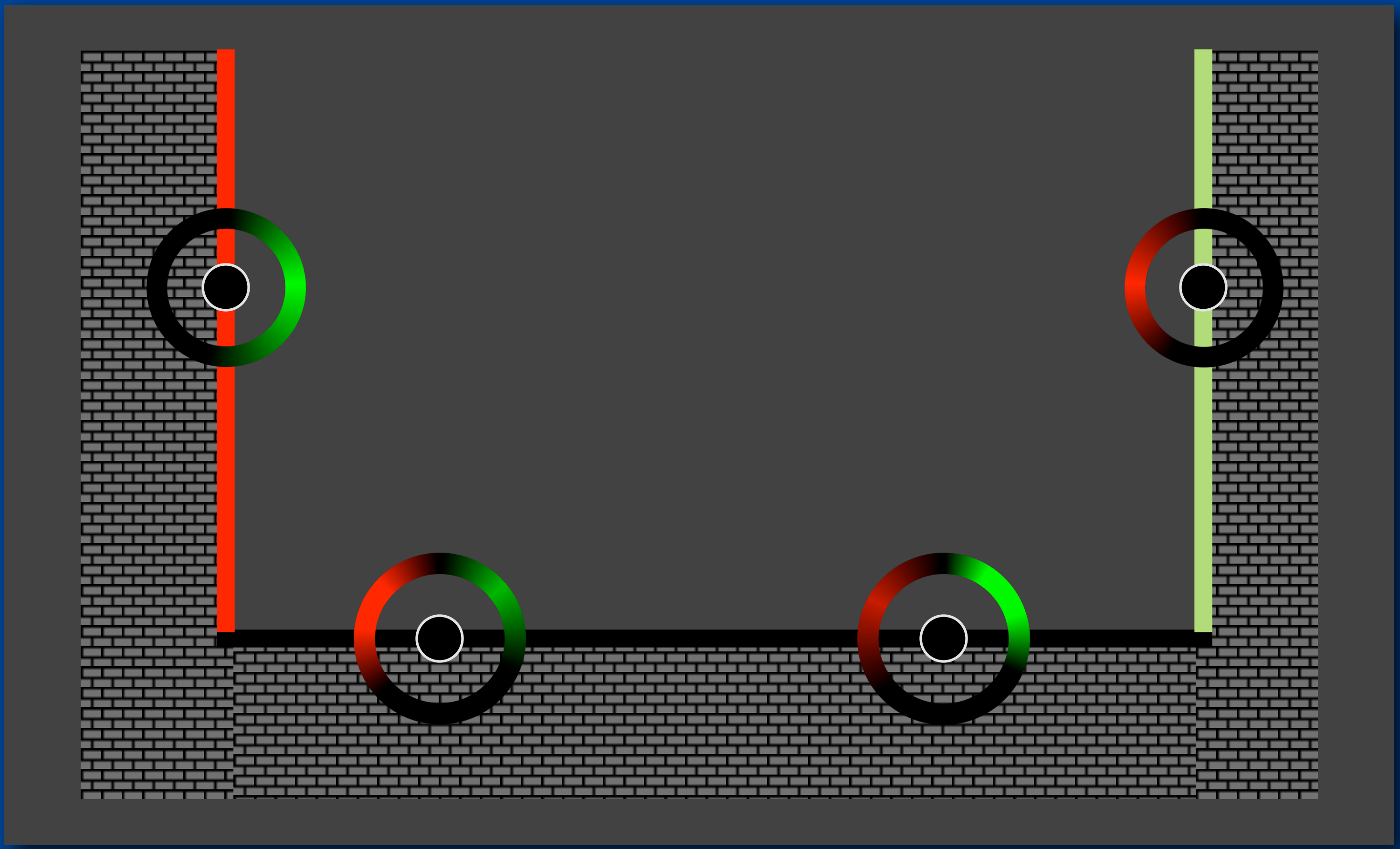
•To compute the incident radiance from distant surfaces we use scattered data interpolation
•This is essentially irradiance caching
•We compute the incident radiance at some points, similar to irradiance caching
•Irradiance is an integral quantity, so it is only a scalar value (actually a spectral quantity or a color)
•Whereas incident radiance is a function defined over the incident directions and it gives the amount of power coming from that direction

Spherical Harmonics



•To represent incident radiance, we use spherical harmonics (SH) which is a very natural way to represent functions defined over sphere of directions
•The advantages of using this representation have been shown previously in these excellent papers
•Notice that this is very similar to the representation used in the Jarda's talk
•We should emphasize also that this is not the only way to represent functions defined over incident directions
•One could easily use hemispherical harmonics or wavelets.

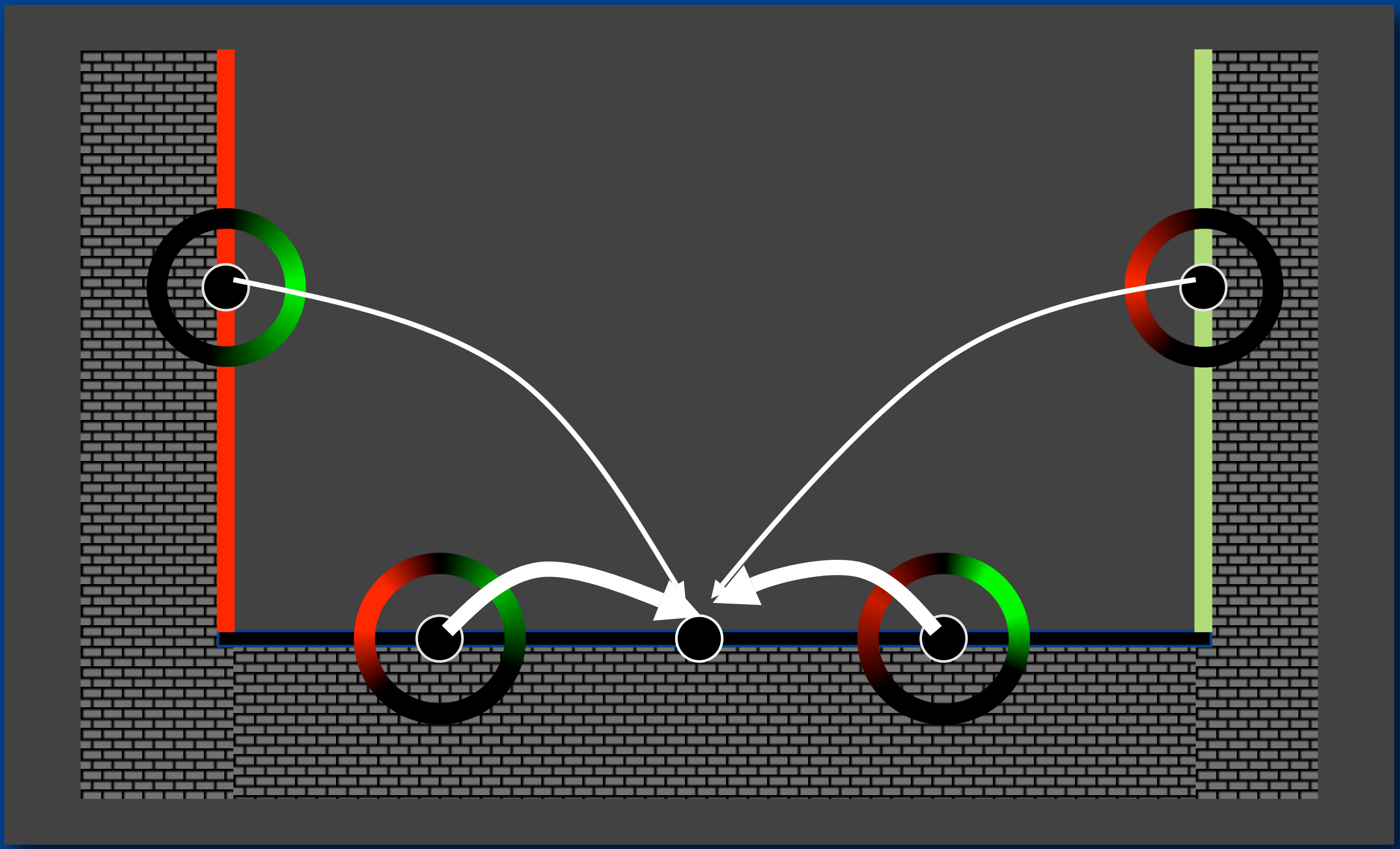
Incident Distant Radiance



face tomorrow SIGGRAPH2007

•SH samples representing the incident radiance are computed at sparse set of points
•To compute these samples, we sample the incident radiance by shooting rays
•Wherever these rays hit, we lookup the proxy global illumination (such as a photon map)
•We then find the least squares spherical harmonic representation for these rays (the incident radiance)

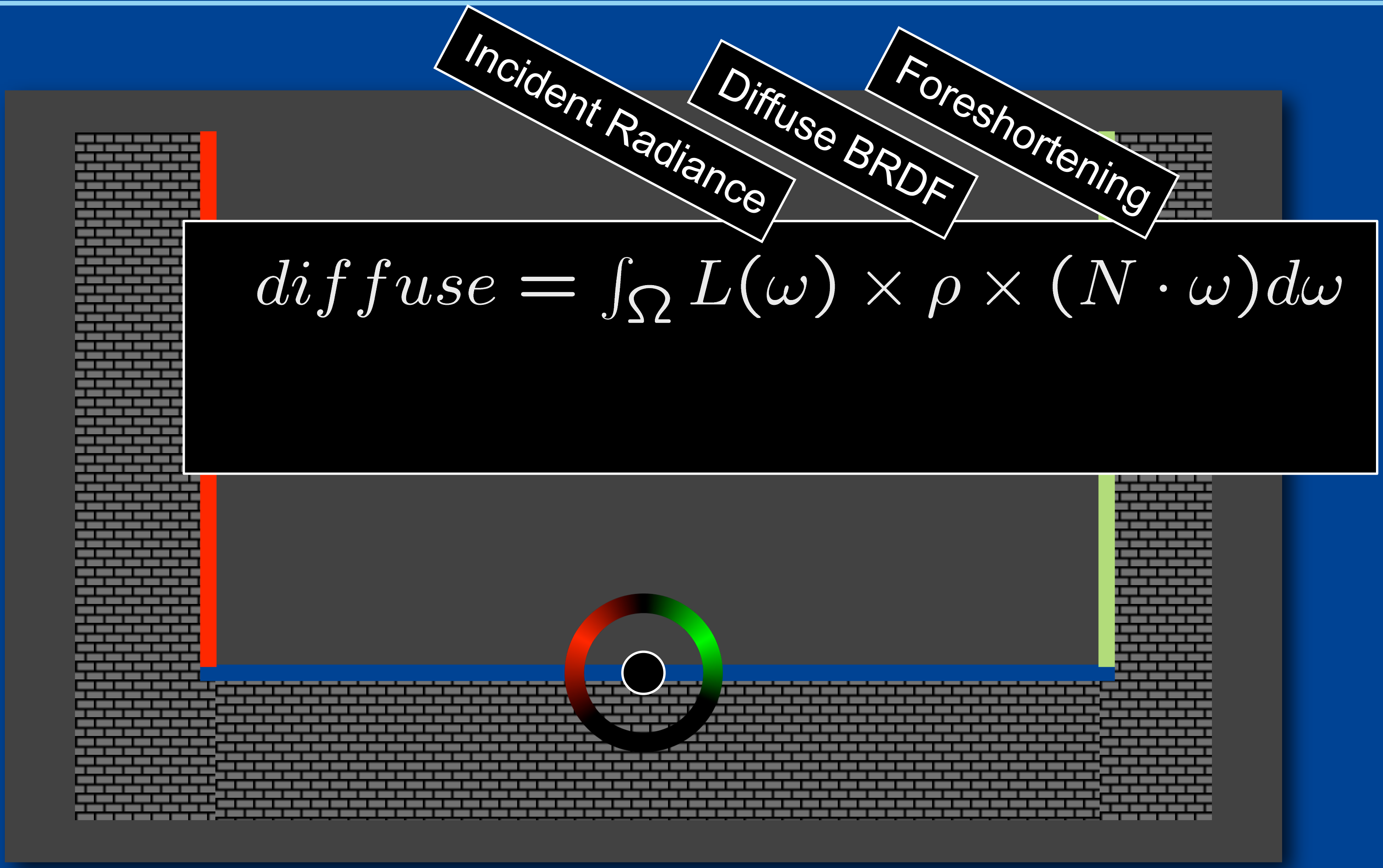
Incident Distant Radiance



•Given a query location, we interpolate these samples (similar to irradiance caching) to obtain an approximation to the incident radiance

•This interpolation is very similar to the one presented in the irradiance caching section

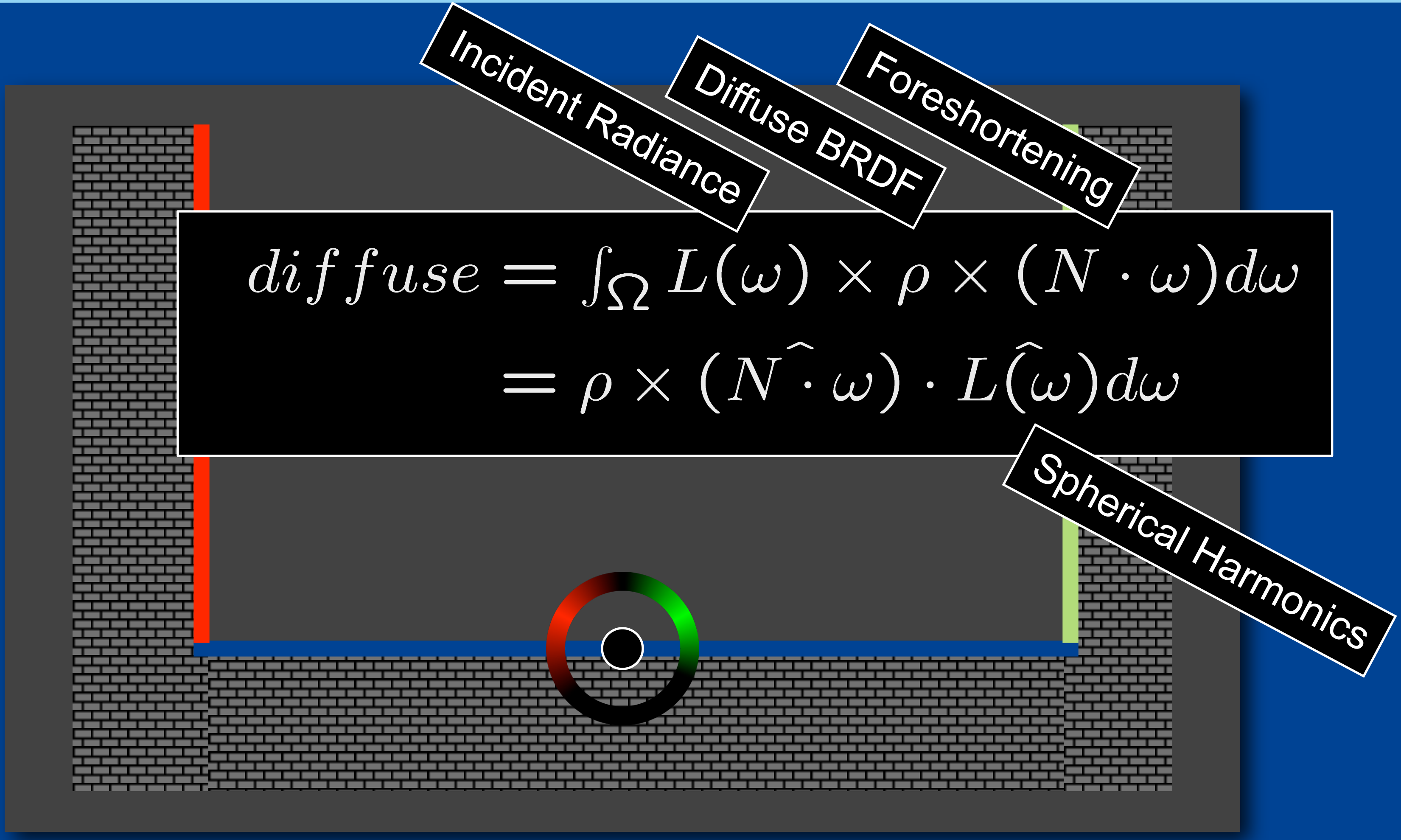
Incident Distant Radiance



$$diffuse = \int_{\Omega} L(\omega) \times \rho \times (N \cdot \omega) d\omega$$

•Because of orthanormality of SH, the distant component becomes a dot product (easy)
•In particular, the diffuse illumination at this point is the integral of the incident radiance times the diffuse BRDF times a foreshortening term
•Spherical harmonics is a frequency space representation that shares the same features as Fourier representation for functions
•Therefore we can write this convolution as a dot product in the spherical harmonic representation
•We already have the SH representation for the incident radiance
•There are analytical forms for the SH representation of the foreshortening term (see [Ramamoorthi and Hanrahan 2001])

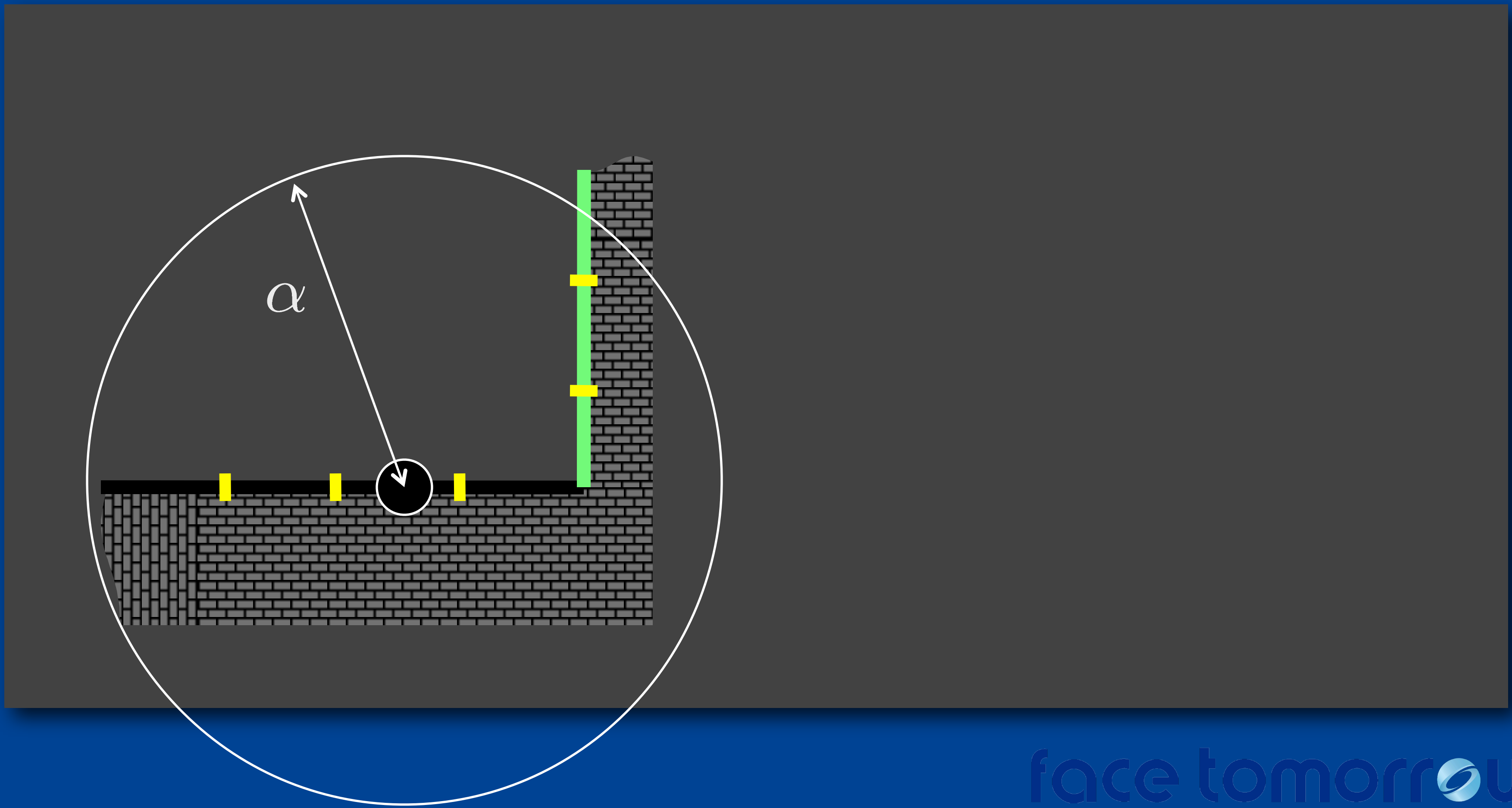
Incident Distant Radiance



$$
\begin{aligned}
diffuse &= \int_{\Omega} L(\omega) \times \rho \times (N \cdot \omega) d\omega \\
&= \rho \times (N\hat{\ }\cdot \omega) \cdot L(\hat{\omega}) d\omega
\end{aligned}
$$

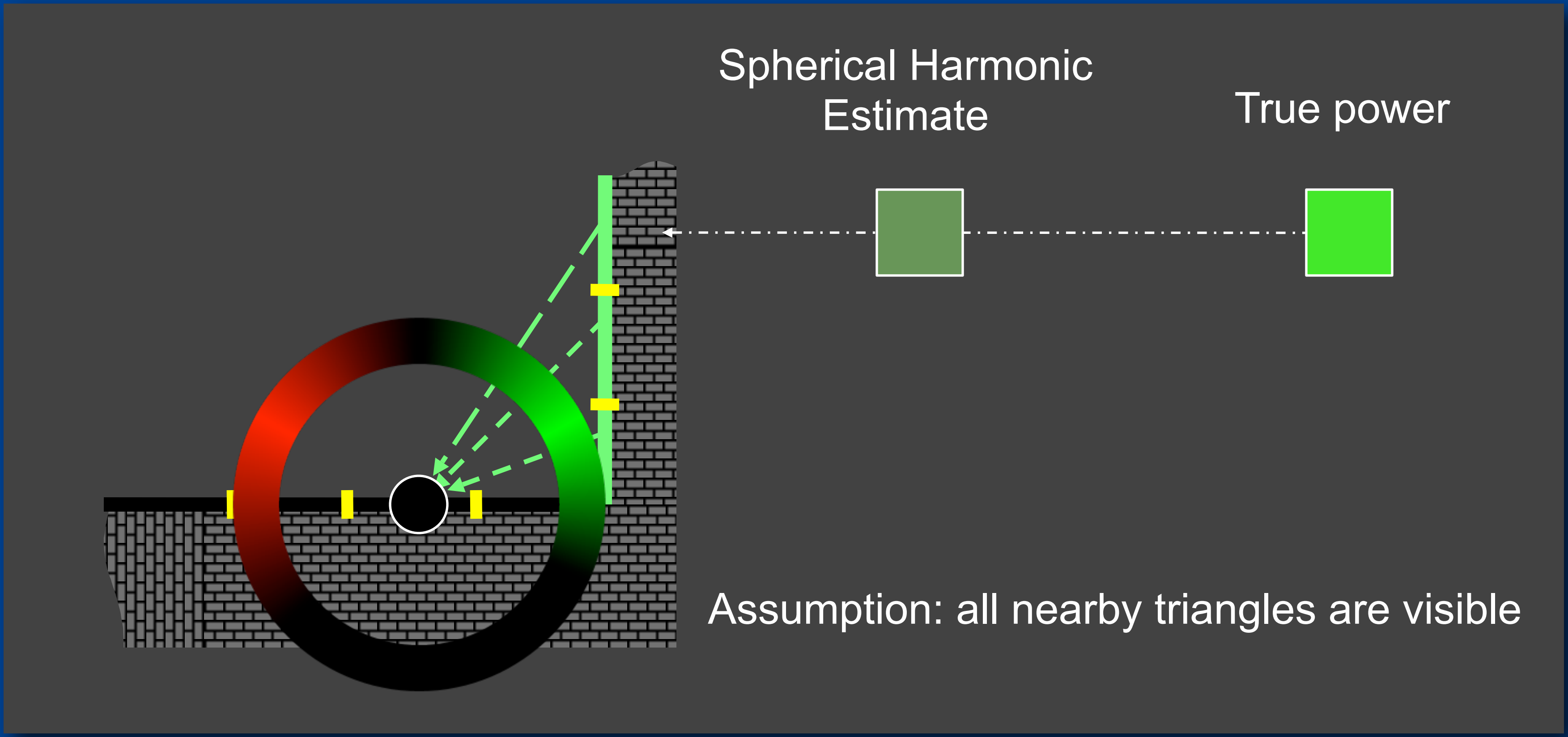
•Because of orthanormality of SH, the distant component becomes a dot product (easy)
•In particular, the diffuse illumination at this point is the integral of the incident radiance times the diffuse BRDF times a foreshortening term
•Spherical harmonics is a frequency space representation that shares the same features as Fourier representation for functions
•Therefore we can write this convolution as a dot product in the spherical harmonic representation
•We already have the SH representation for the incident radiance
•There are analytical forms for the SH representation of the foreshortening term (see [Ramamoorthi and Hanrahan 2001])

Nearby Geometry



•OK, so we can compute the diffuse irradiance of a point by a simple dot product
•But we ignored the nearby geometry and because we sampled the incident radiance, the result is a smooth approximation to the global illumination
•We need to take the power reflected or occluded by the nearby geometry into account
•To do this, for each query point, we iterate over the nearby scene triangles

Nearby Geometry



Assumption: all nearby triangles are visible

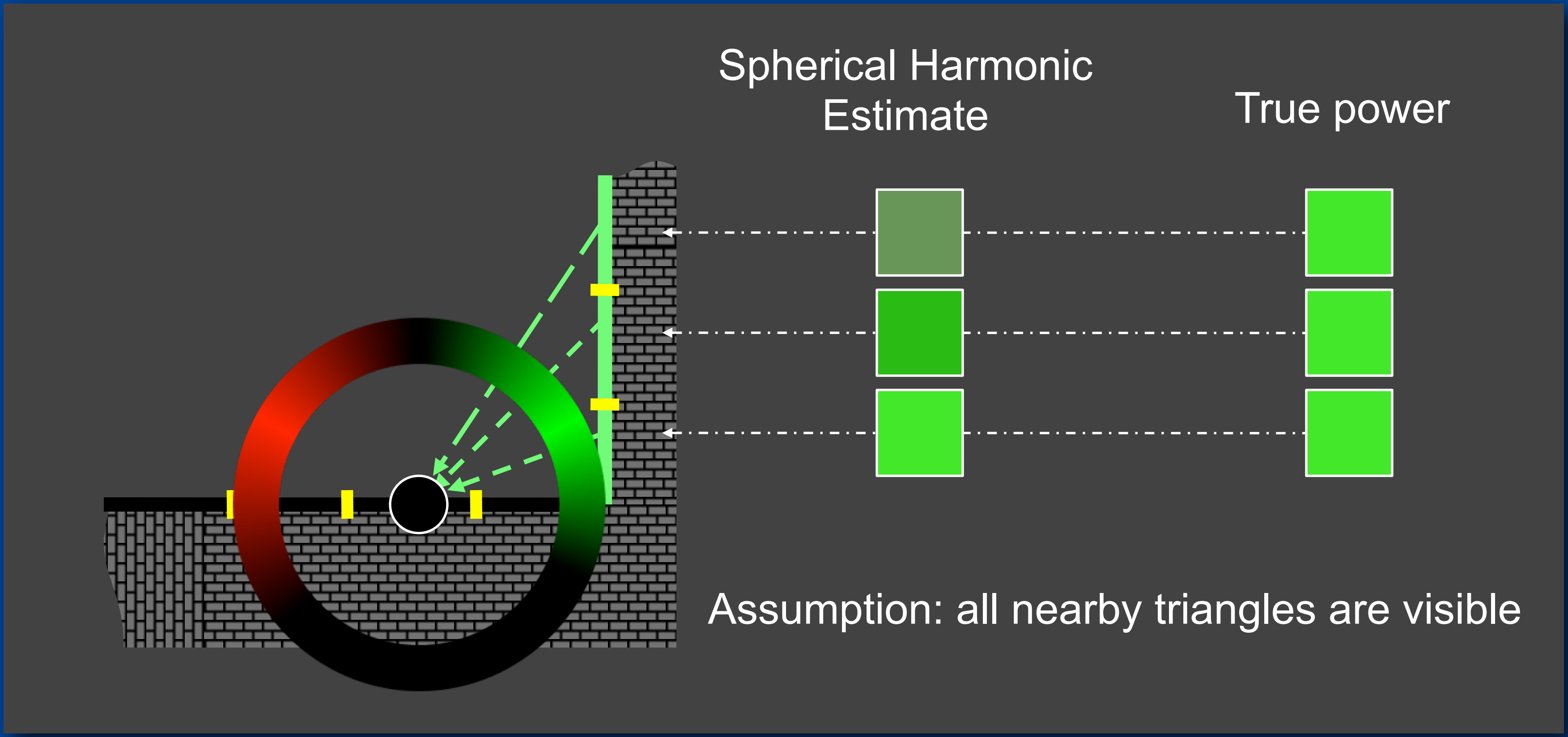
•For each nearby triangle,

- •Subtract the energy from the SH representation that the triangle covers
- •We do this by Monte Carlo (MC) Integration. In particular since we only use low order harmonics, the SH function we're sampling is smooth and MC integration is efficient
- •Add the energy from the triangle to the point
- •We do this by computing the form factor of the triangle and looking up the radiosity of the triangle by looking into our proxy global illumination representation (photon map)

•If we assume all nearby scene triangles are visible (barring those that are backfacing to the query point), this can be computed efficiently using analytical triangle to point form factors

•This step restores the high frequency content of the global illumination due to geometric detail

Nearby Geometry



Assumption: all nearby triangles are visible

•For each nearby triangle,

•Subtract the energy from the SH representation that the triangle covers

•We do this by Monte Carlo (MC) Integration. In particular since we only use low order harmonics, the SH function we're sampling is smooth and MC integration is efficient

•Add the energy from the triangle to the point

•We do this by computing the form factor of the triangle and looking up the radiosity of the triangle by looking into our proxy global illumination representation (photon map)

•If we assume all nearby scene triangles are visible (barring those that are backfacing to the query point), this can be computed efficiently using analytical triangle to point form factors

•This step restores the high frequency content of the global illumination due to geometric detail

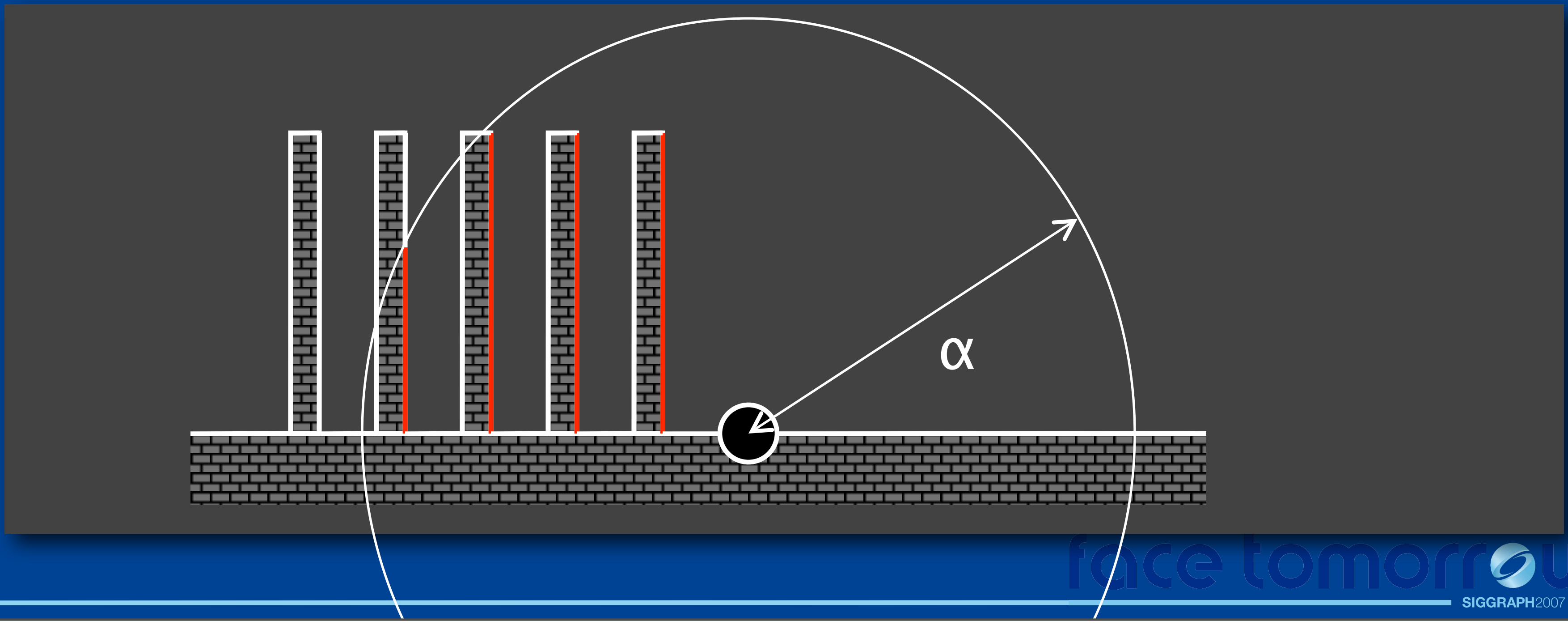
α Parameter

- Effect of changing α
  - α = 0 : Irradiance caching
  - α = ∞: Everything is nearby and visible
- Visibility computations are expensive

•Crucial parameter is alpha (the nearby-distant threshold)
•When alpha is zero, the method is similar to irradiance caching. We will collect denser samples of incident radiance around corners or high geometric detail
•When alpha is large, the method will assume everything is visible and the results will be inaccurate
•Alpha allows us to ignore expensive visibility computations between nearby surfaces
•This is where we make our money, nearby surfaces are usually visible to each other. By ignoring these visibility computations, we obtain good speedup without creating too much error

Limitations

- Polygonal geometry
- Range search
- Single α for the entire scene



•This method has several limitations:

•We assume the scene is polygonal so that we can compute analytical form factors for nearby geometry

•For each query point, we need to locate nearby triangles which involves a range search similar to the one in photon mapping

•Having a single alpha value for the entire scene may not be too optimal

•Also, for scenes such as this one, our assumption about the nearby surfaces being visible may fail.

Implementation Details

1/3

- Pass 1: Save the surface locations that will need global illumination
  - Can be coarser than the actual beauty rendering

•To recap, this is a 3 pass algorithm
•In the first pass, we record the surface locations that will need the indirect illumination computation
•Therefore this is a lazy algorithm, we do not need to sample all surfaces, only the visible ones

Implementation Details 2/3

- Pass 2: Cluster these points so that no cluster is bigger than alpha
  - For each cluster sample incident radiance by shooting random rays from random surface points in the cluster
  - Fit Spherical Harmonics to these samples for each cluster
  - Also tesselate the surfaces into triangles here
  - Split big triangles (bigger than alpha) --- they may create discontinuities in the computation

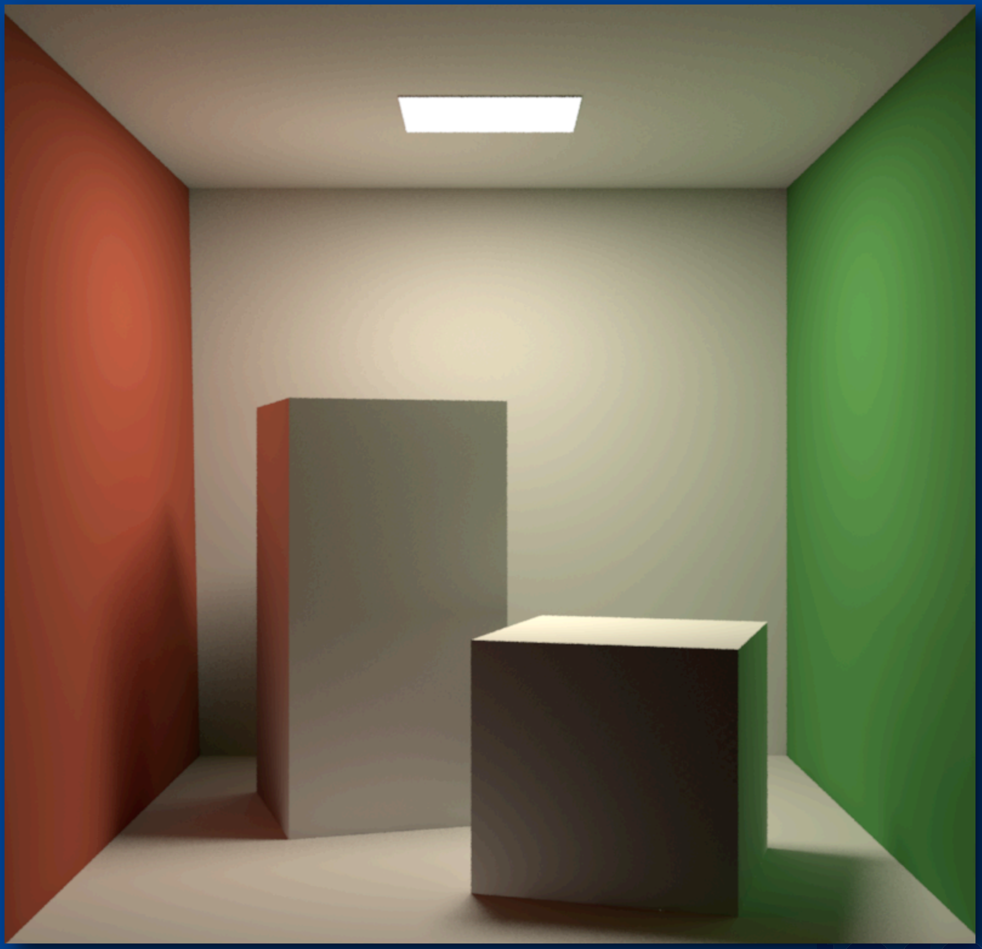
•The second pass involves hierarchical vector quantization to cluster these surface locations so that each cluster is no bigger than alpha
•For each cluster we can now go ahead and sample the spherical harmonics
•We do that by shooting random rays from random surface locations within each cluster
•At this step we do not care about intersections within the origin clusters, so we ignore intersections closer than alpha
•In this step, we also convert the scene geometry into triangles
•We must avoid big triangles because they may create discontinuities (imagine a big triangle which was not near suddenly becoming near)
•We split these triangles so that they are no bigger than alpha in scale

Implementation Details 3/3

- Pass 3: For each query point,
  - Interpolate SH samples
  - Locate the nearby triangles (centers near to the query point)
  - Perform the computation

•Third pass is the beauty pass
•Here for every query point, we interpolate the SH samples and locate nearby triangles (whose centers are near to the query point)
•We then compute the indirect illumination
•We iterate over the scene triangles closer than alpha to the gather point and update the irradiance estimate by subtracting the power they occlude from the SH representation and add the power they reflect towards the gather point

Results

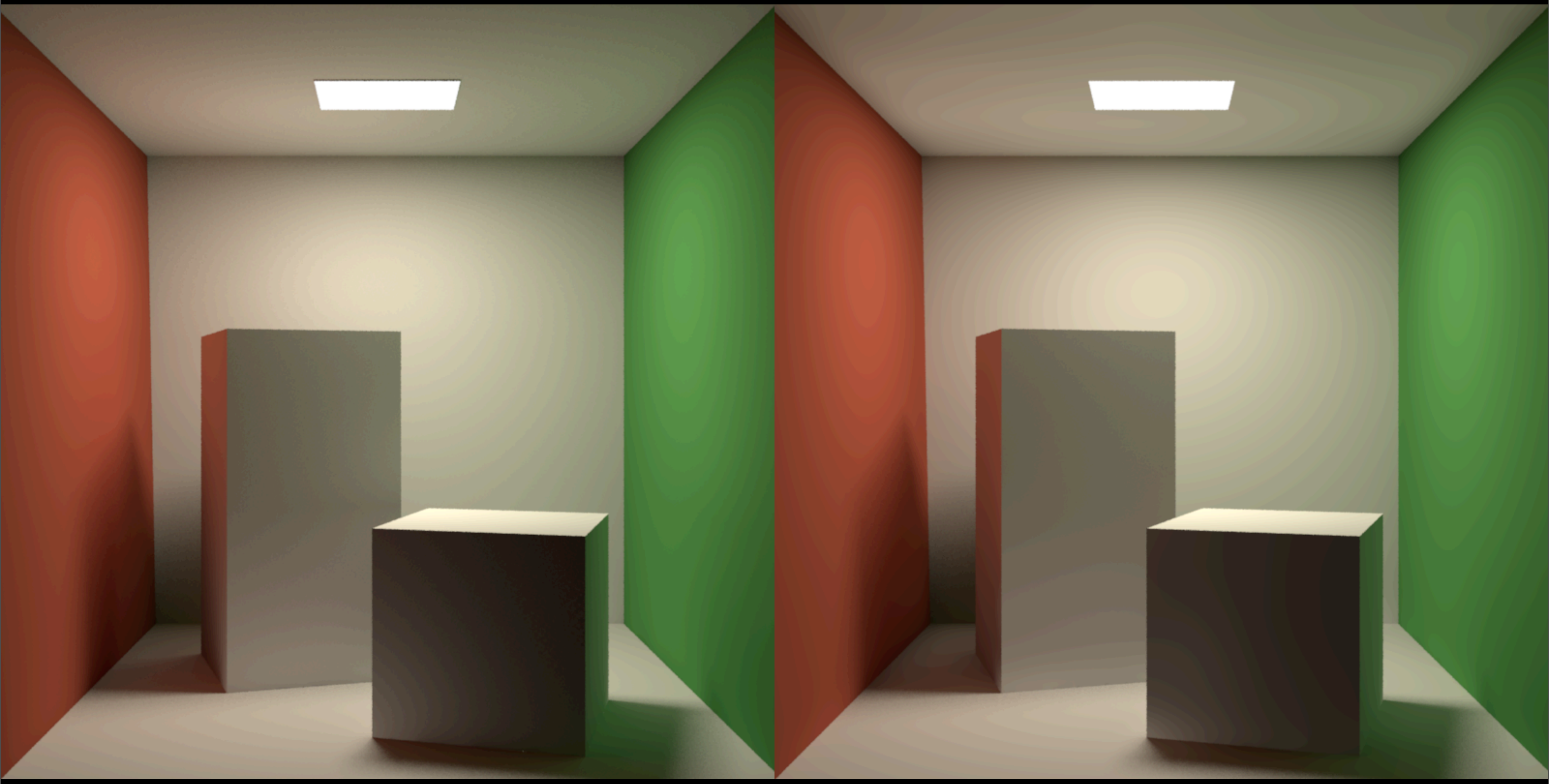


face tomorrow SIGGRAPH2007

•Let’s move onto the results and talk about what we can do with this method
•First, the good old Cornell Box

Irradiance Caching

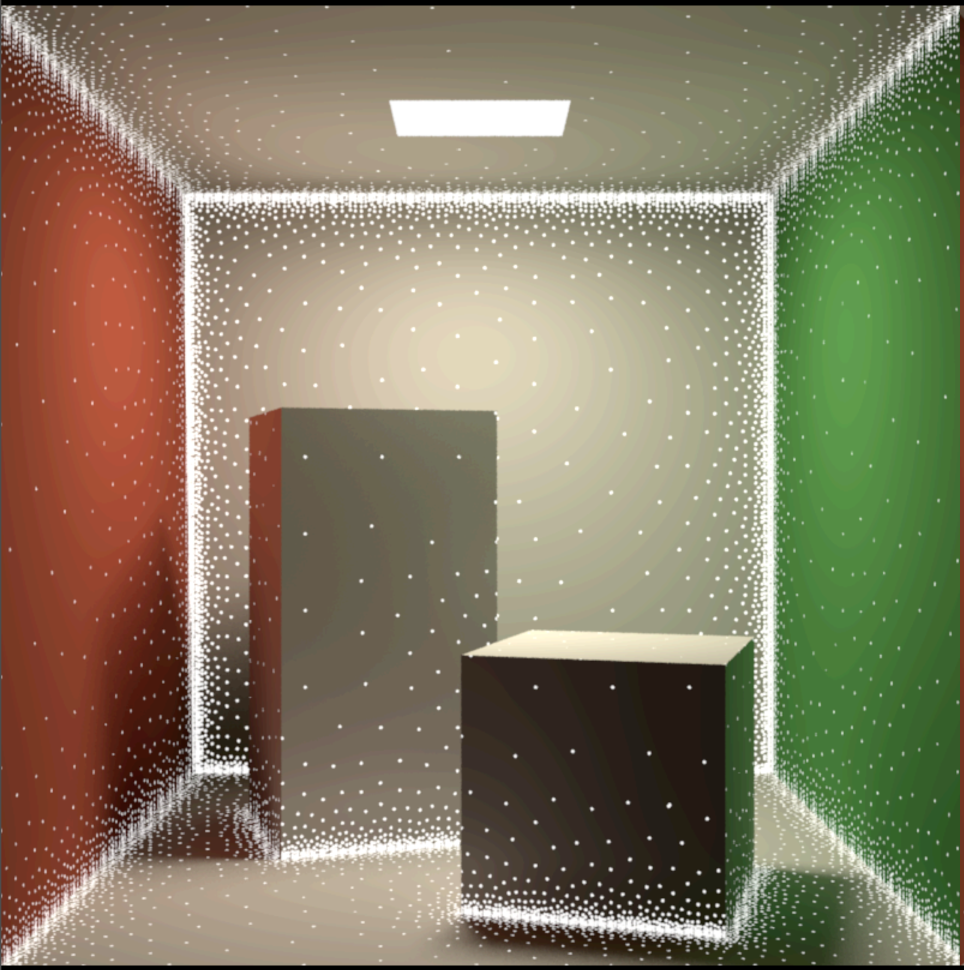
Our Method

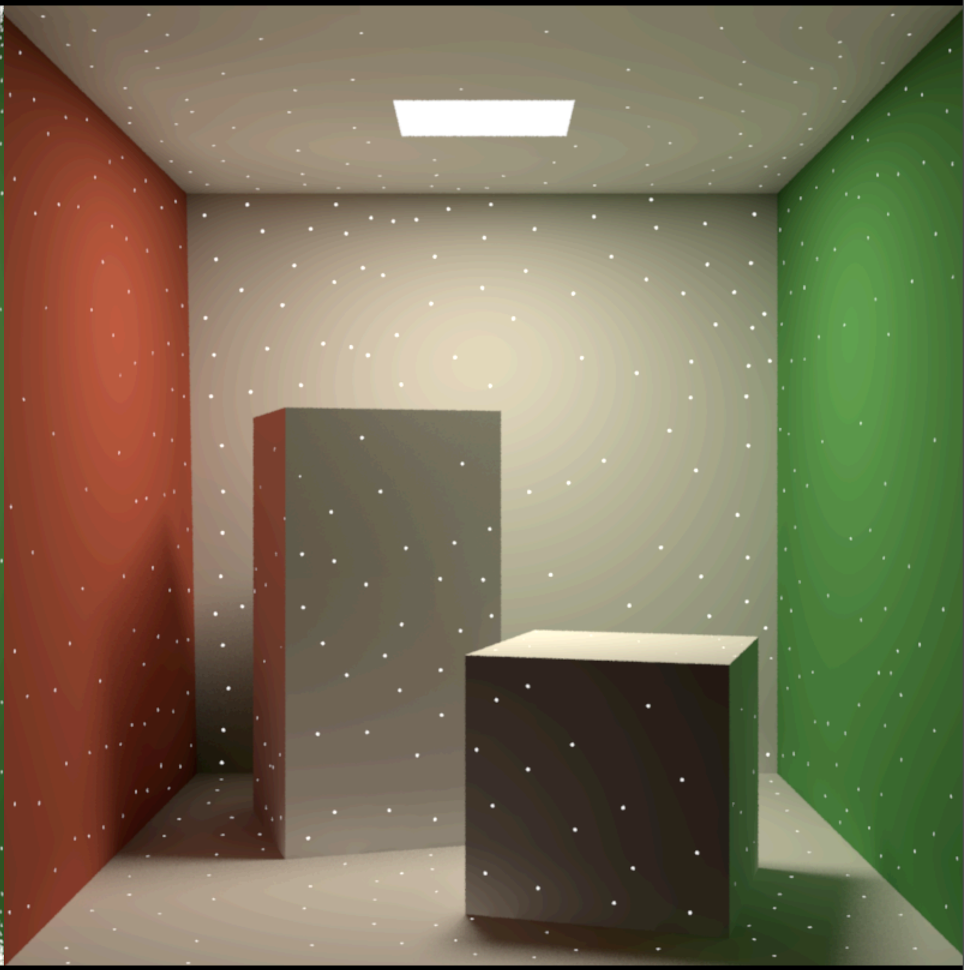


•Here’s a comparison with irradiance caching

Irradiance Caching

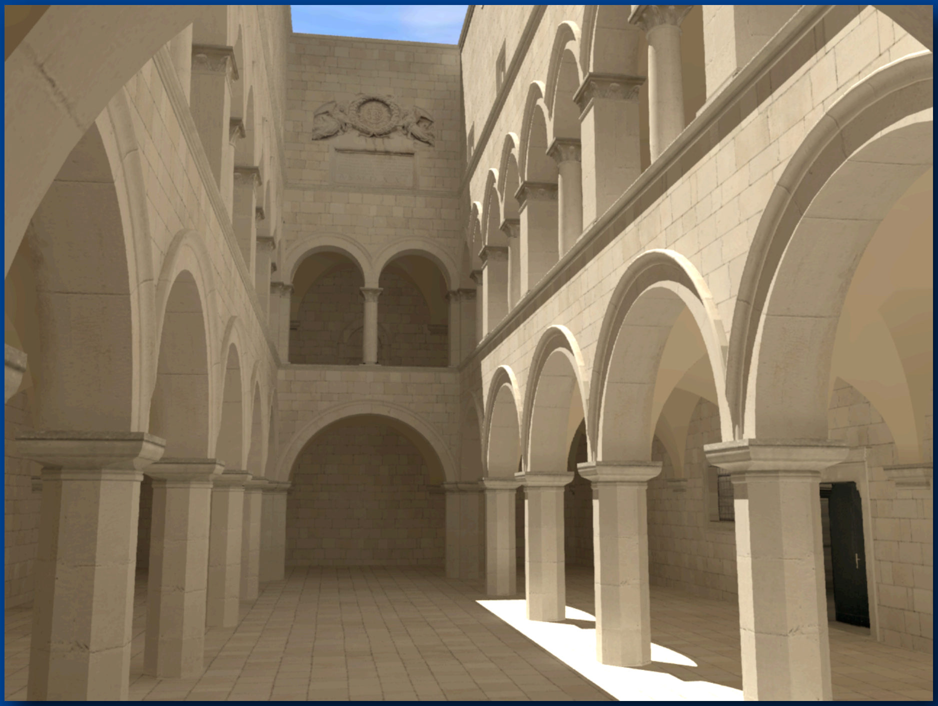
Our Method

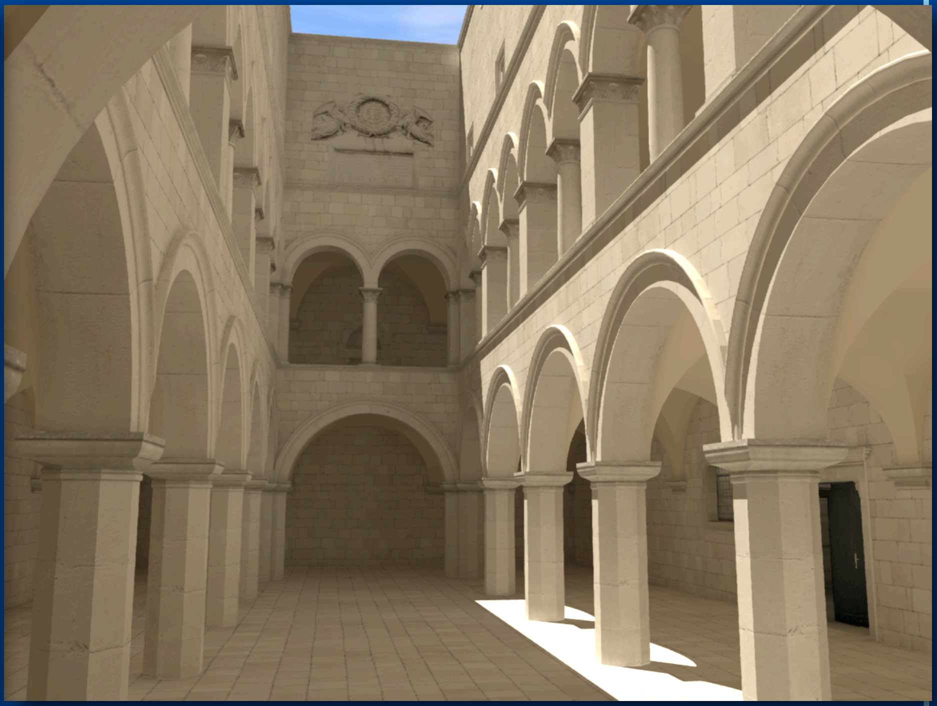




•Here're the corresponding sample locations
•The cost of computing one of these samples for irradiance caching and our method is the same
•(the cost of the dots is the same)

Visual Quality

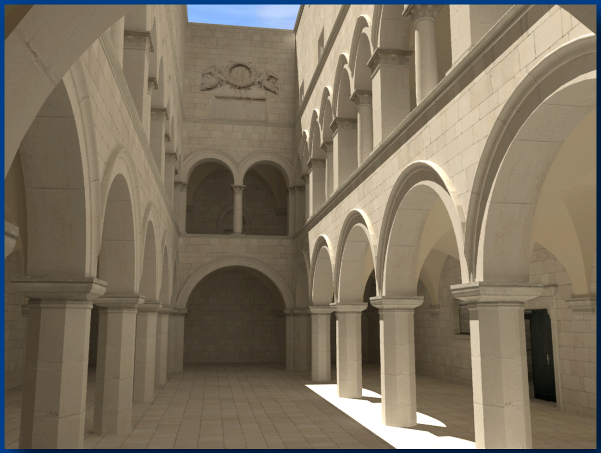

Irradiance Caching (1 hr)

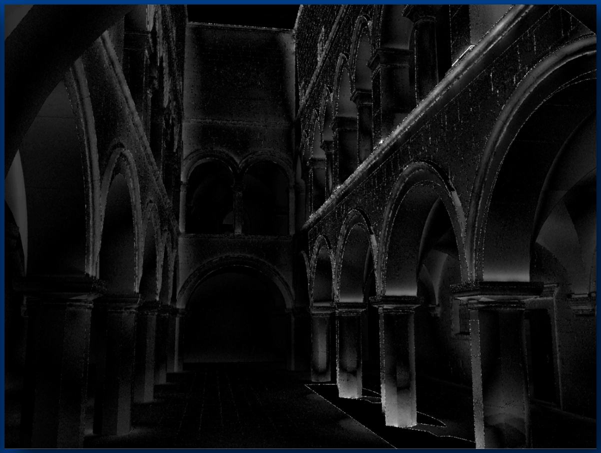

Our Method (5 min)

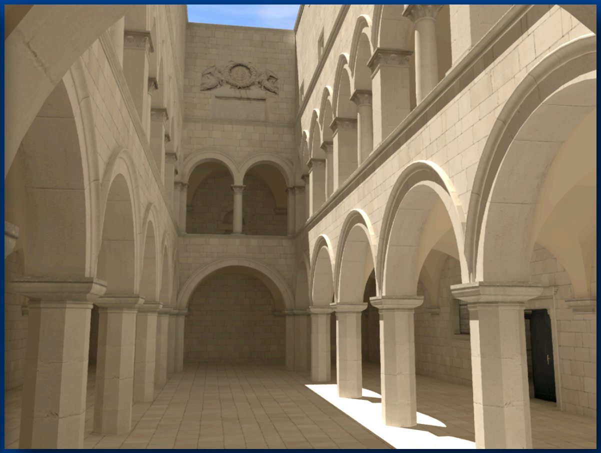
face tomorrow SIGGRAPH2007

•Here’s a comparison on a more complicated scene

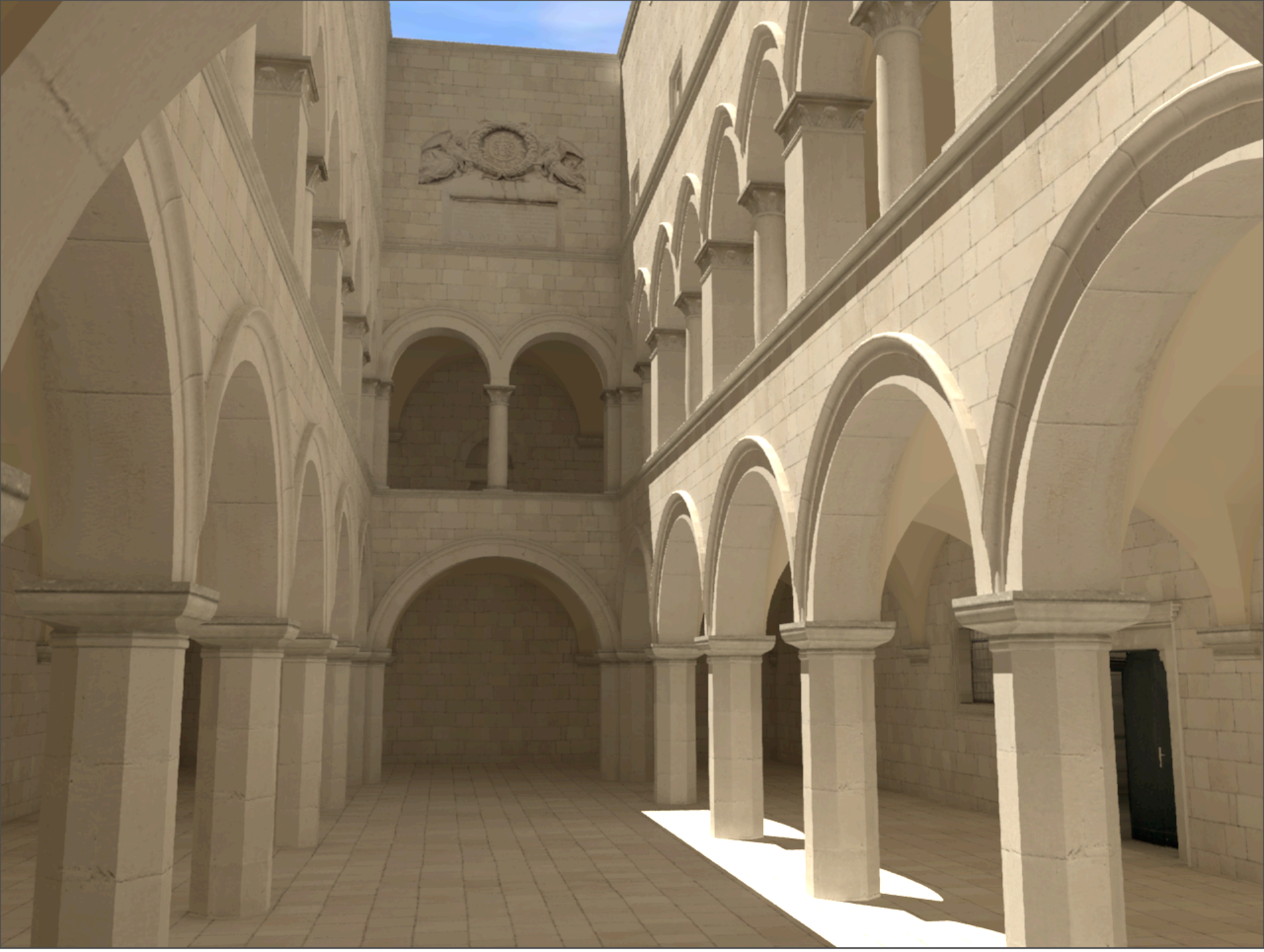
Visual Quality


Ground Truth
(Monte Carlo)
24 Hrs.


4X Difference

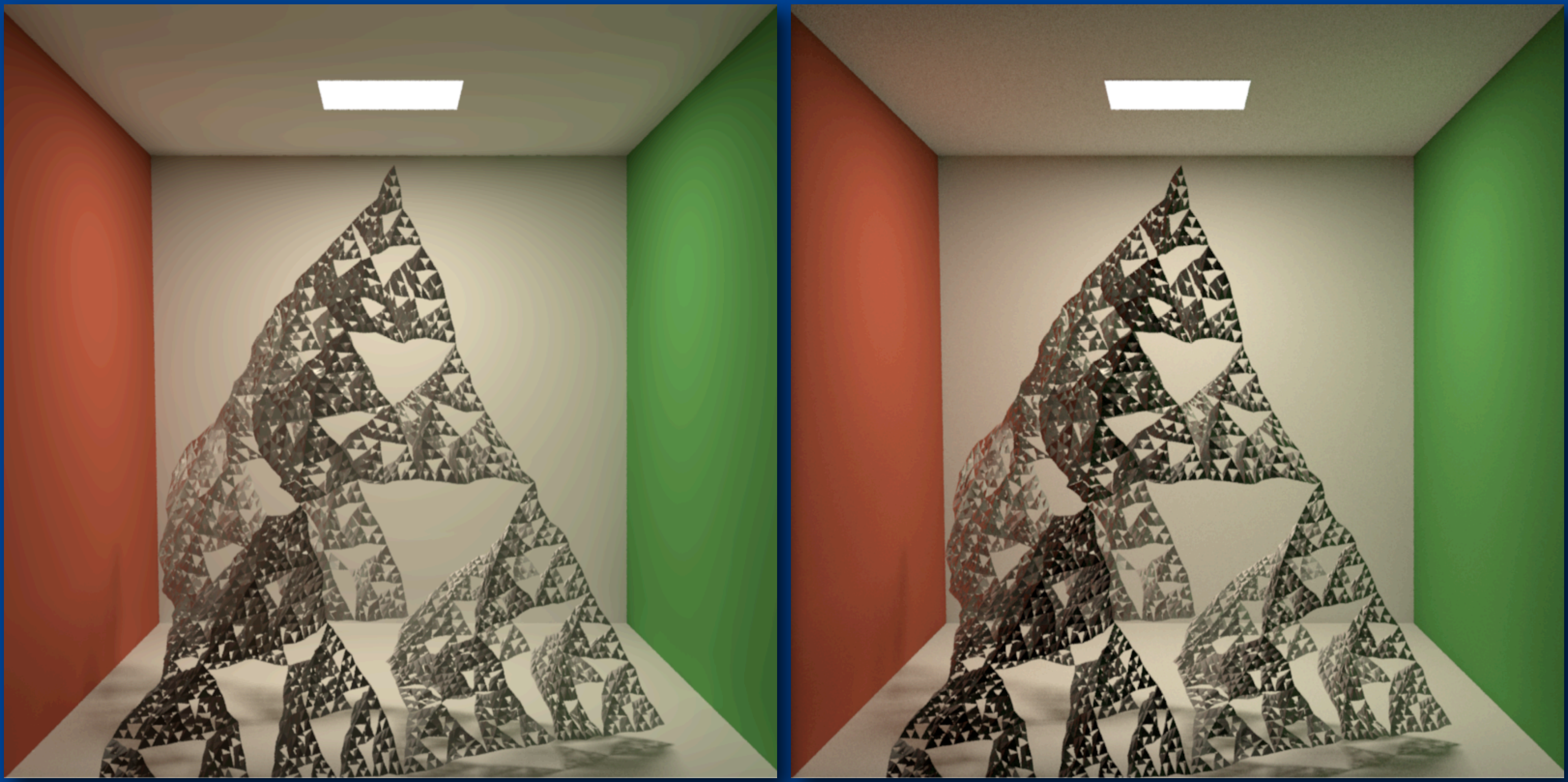

Our Method
5 Minutes

•The middle image shows the 4 times magnified difference between the ground truth and this method

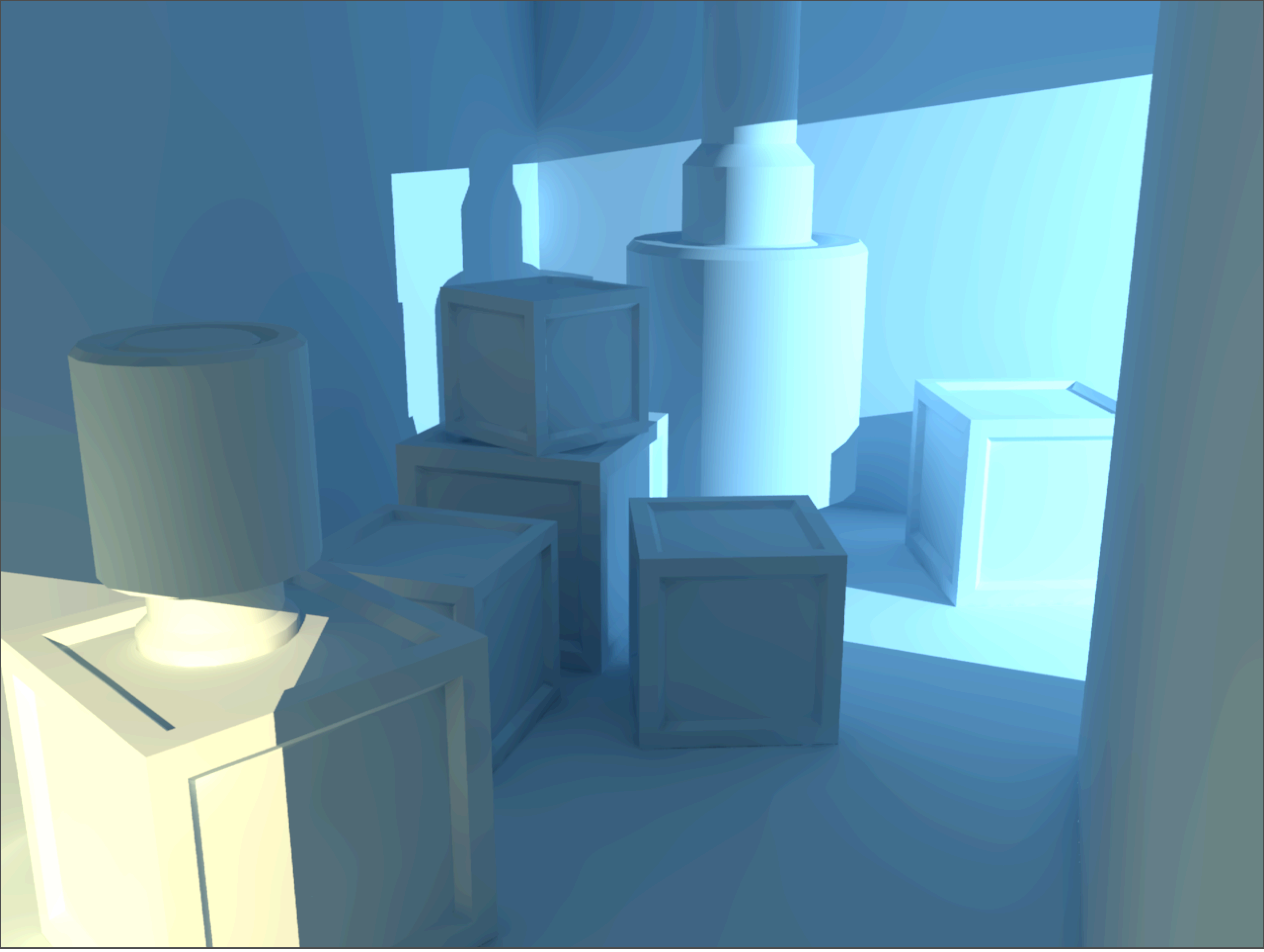


•Here’s a bigger version of the atrium scene

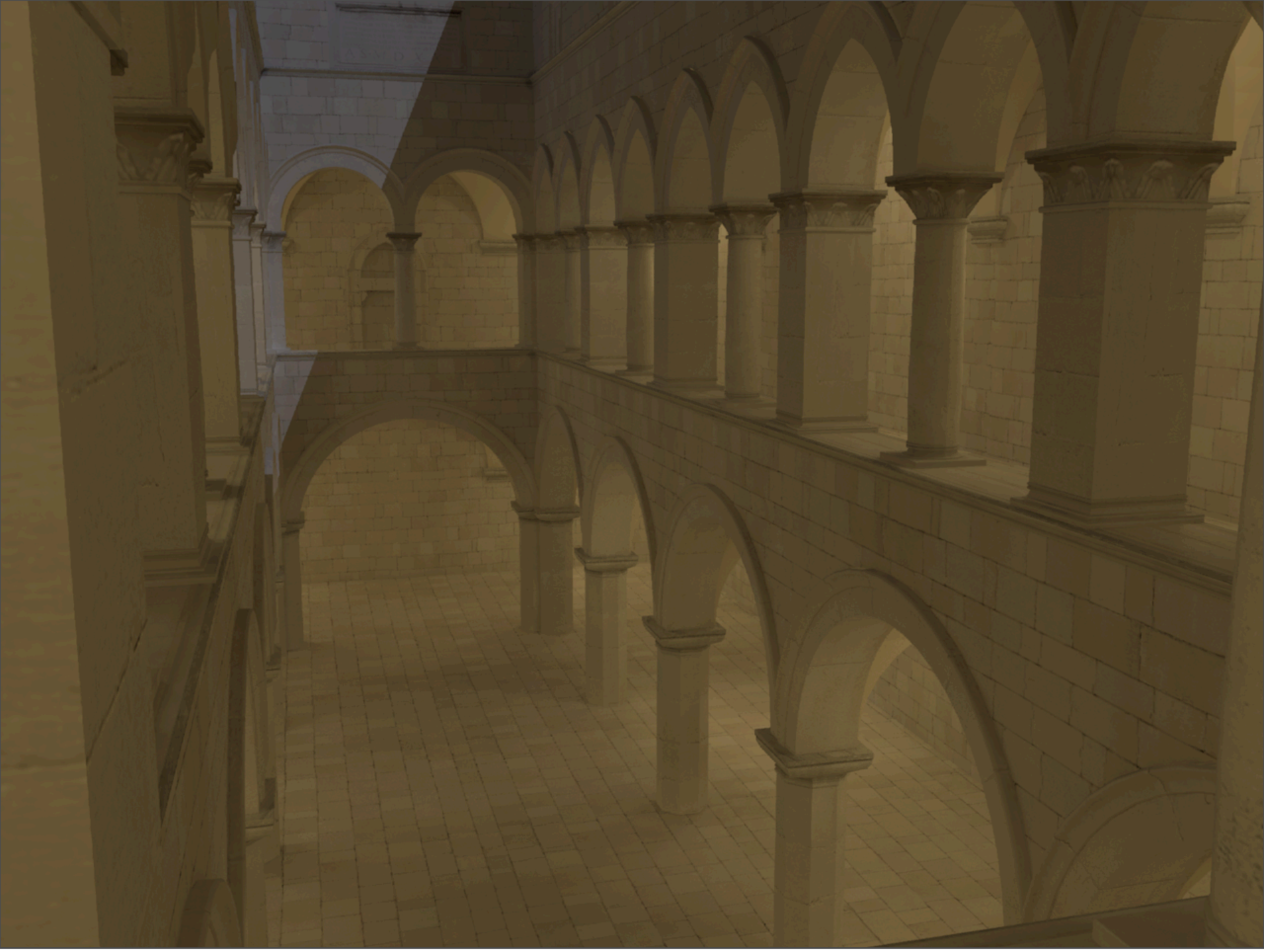
Failure Case


Ground Truth Our Method

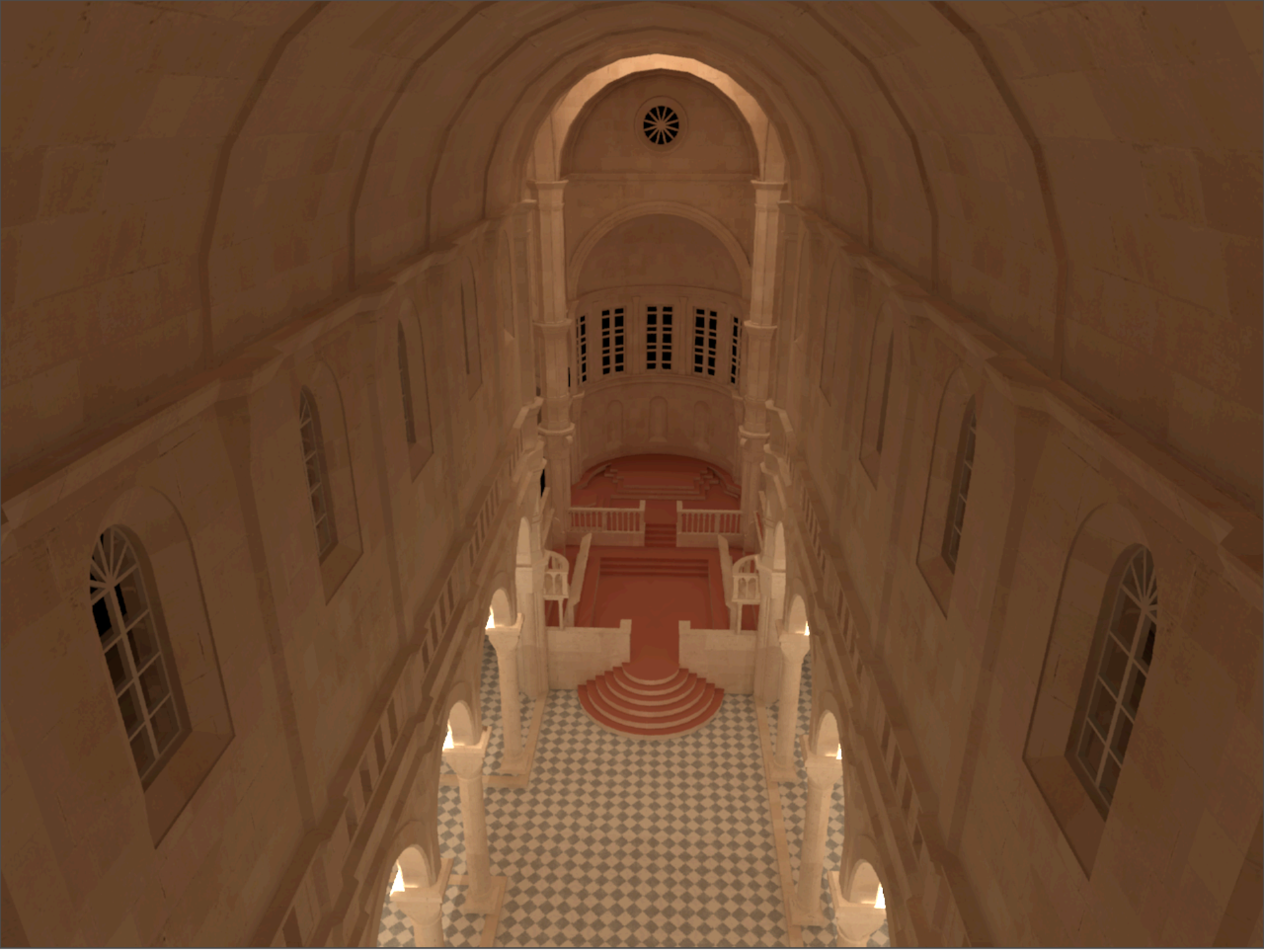
•Here's a failure case
•Notice that our method is generating a darker appearance because the visibility assumption is being violated in this convoluted example
•Fortunately, such examples are rare



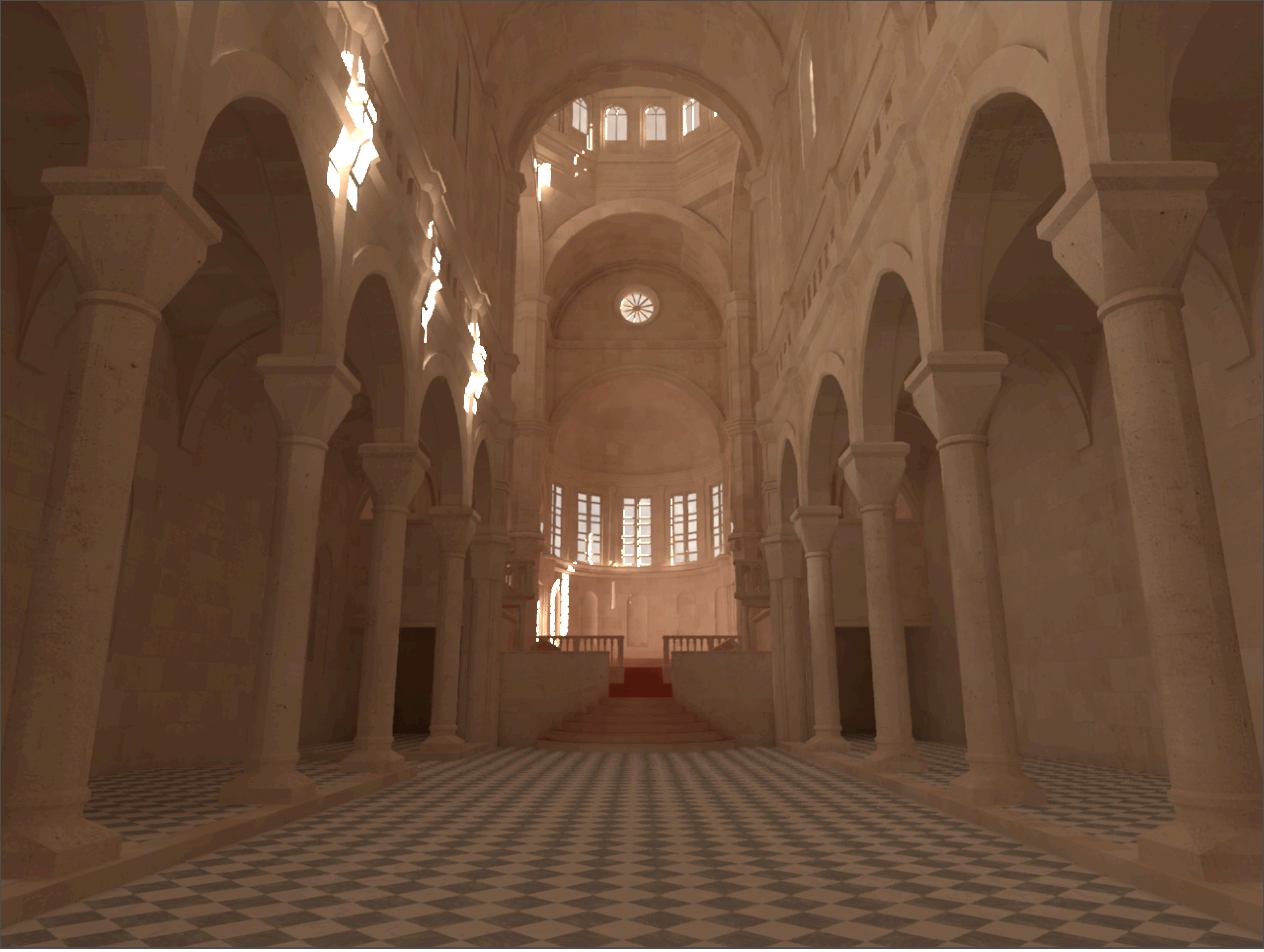
•Let’s look at some more examples



•A night time
scene

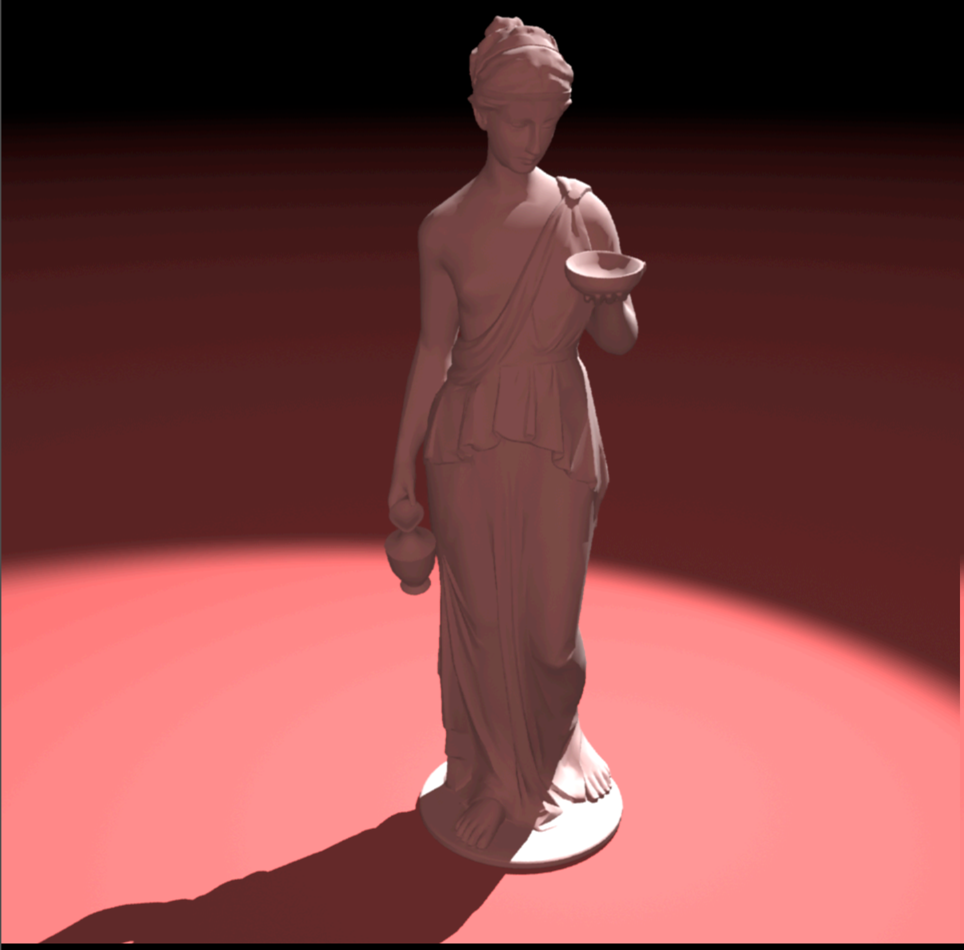


•A cathedral during night

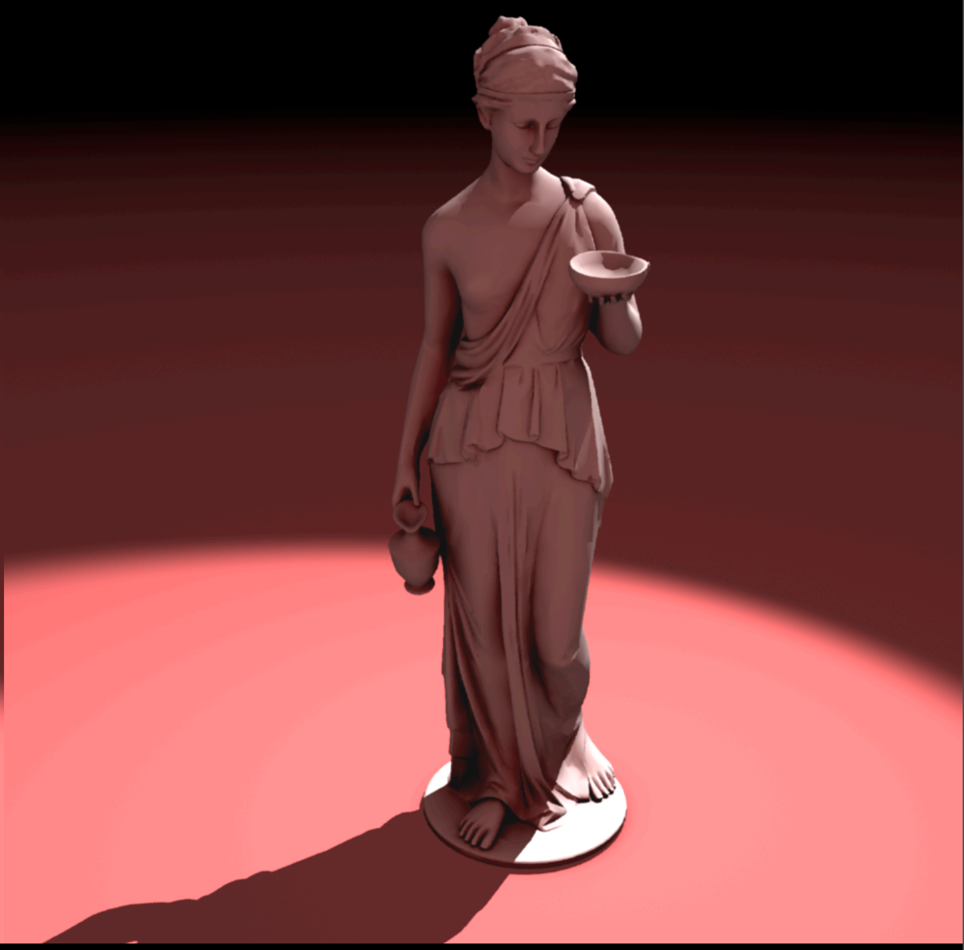


•The cathedral during the day

Manually lit scene



Manually lit scene
with local correction



- By using only the nearby scene triangles and subtracting the energy that they cover, we can simulate ambient occlusion as well
- The image on the right is generated without firing a single ray

Overall

- Cute and fast, but difficult to implement
  - 3 passes !!!
  - Need to tesselate surfaces

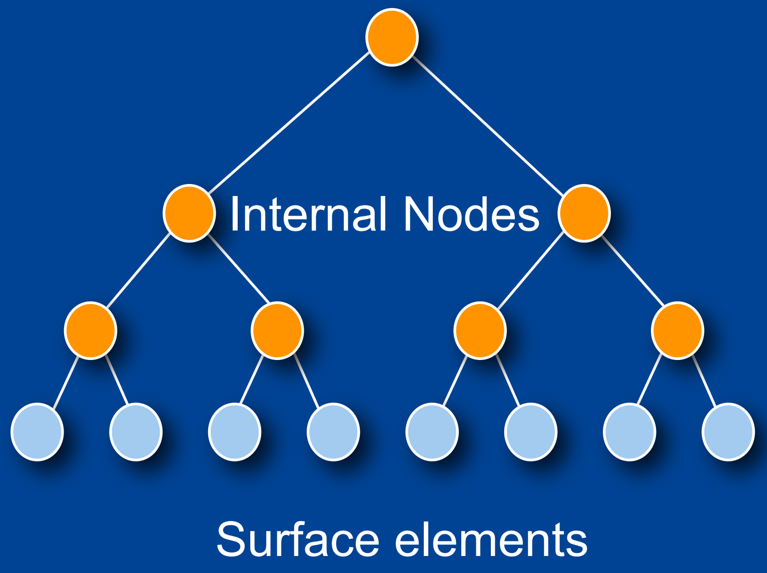
- Although you can generate very high quality images very fast, the method I described is somewhat difficult to implement
- It involves three passes, we need to tesselate the geometry which is typically undesirable
- How can we simplify the method?
- One source of difficulty is this near-far distinction

Simplify

- Set alpha to infinity
  - No distant term, everything is near
- Convert triangles into area elements (discs)
  - No costly triangle to point form factor
- Create a hierarchy of surface elements
  - For faster summation

- We can get rid of this by making alpha infinity, meaning everything is nearby
- Therefore we do not need a distant term, no more spherical harmonics or scattered data interpolation
- If we do this, however, we will have to sum over every triangle we have in the scene for every shading point
- We can make this slightly faster by summing over area elements (dics) rather than triangles
- This is faster because the triangle to point form factor is somewhat difficult to compute and if we work with area elements rather than triangles, we do not need to tesselate surfaces
- Although area elements are faster, we still need to cum over all area elements in the scene which is slow
- We can make this faster by creating a hierarchy of area elements

Surface Element Hierarchy



face tomorrow SIGGRAPH2007

- We can create this hierarchy by hierarchically clustering the area elements
- In this hierarchy, he leaves are the original area elements
- and internal nodes are proxy area elements that represent the area underneath them
- They are simply dics whose area is the summer area of their children
- When summing over the elements, we start from the root and sum over the nodes of this tree
- If a node is far away, we can add it into our sum, otherwise, we sum its children

Originally Proposed by

Dynamic Ambient Occlusion and Indirect Lighting
Michael Bunnell, NVIDIA

● Actually, this method was described in the paper by Bunnell to approximate the ambient occlusion and global illumination on the graphics hardware in 2005

Implemented in Pixie

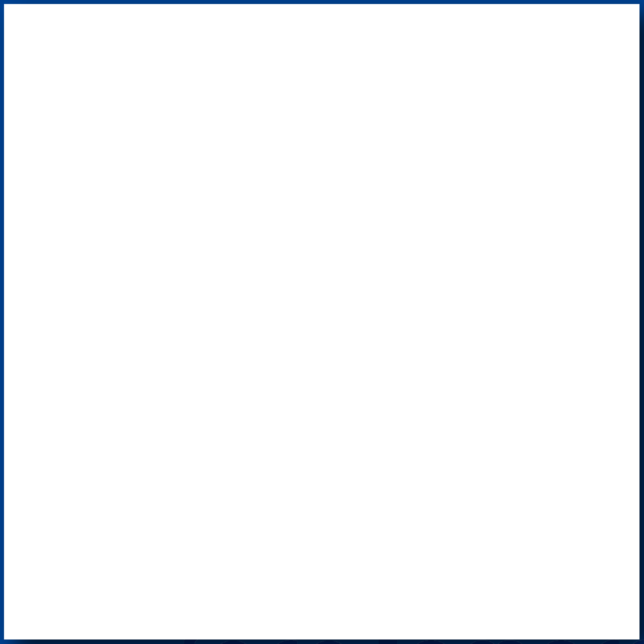
Open Source RenderMan

http://pixie.sourceforge.net

- This method is also incorporated into PrMan last year and in Pixie under the title of point based ambient occlusion and global illumination
- --- Begin advertisement
- Pixie is an open source RenderMan renderer
- You can download the entire source code that also includes this new method
- --- End advertisement
- I would like to finish my presentation by how you can use this new and much simpler point based ambient occlusion and global illumination in RenderMan

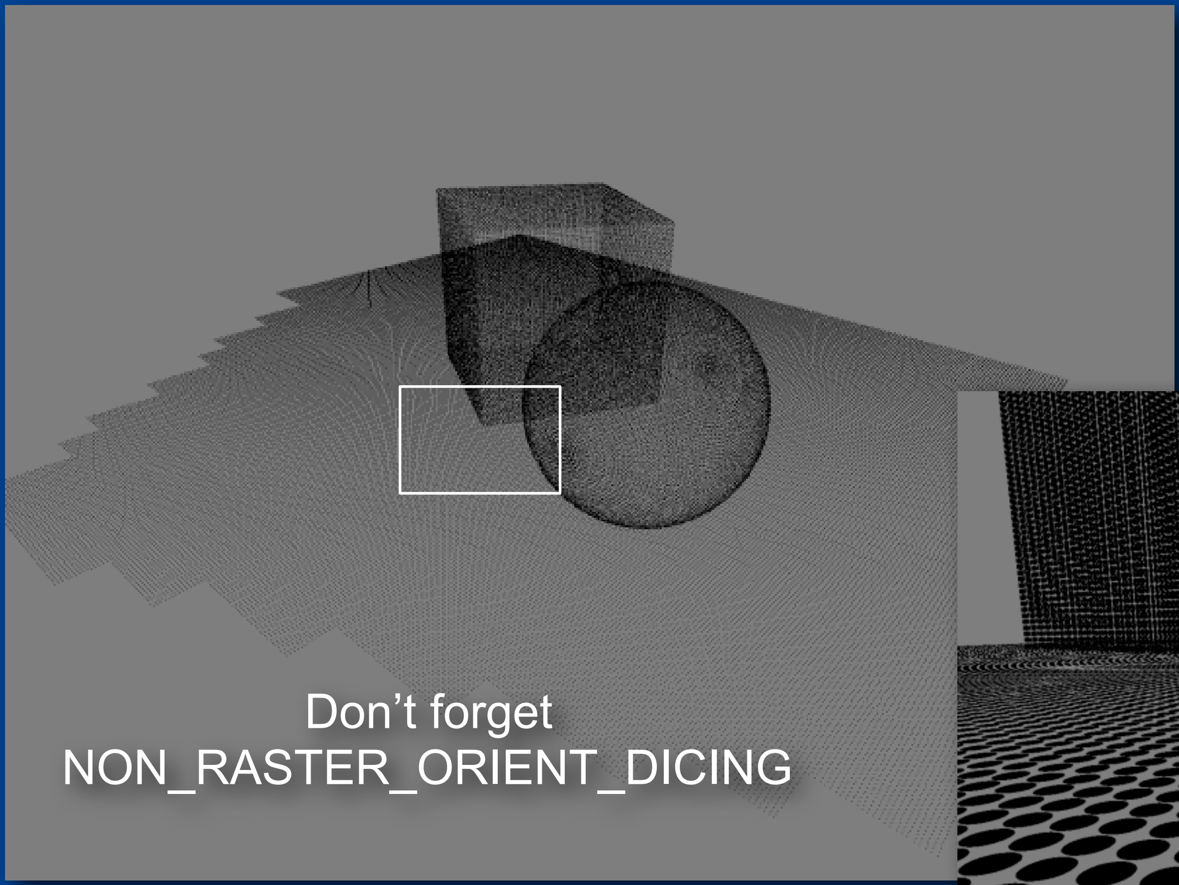
RenderMan - Point Based AO

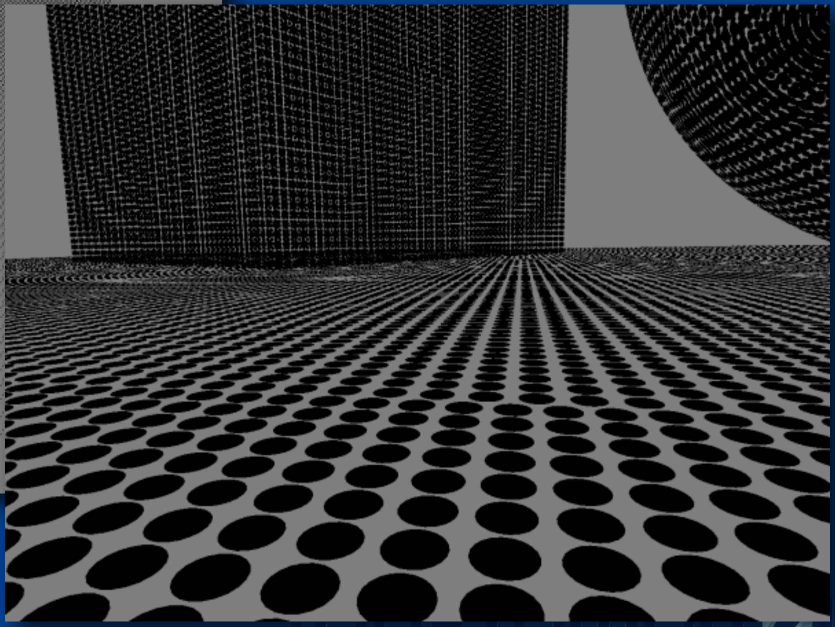
```
// Shader
surface   bake_areas( uniform string filename = "", displaychannels = "" ) {
    normal Nn = normalize(N);
    float  a  = area(P, "dicing"); // Shading element area

    bake3d(filename, displaychannels, P, Nn, "interpolate", 1, "_area", a);

    Ci = Cs * Os;
    Oi = Os;
}
```



- The first step involves “baking” the surface area elements
- RenderMan comes with a built-in function named bake for this purpose
- This function allows the user to save any quantity computed in a shader into a file
- Here is a surface shader example that computes the area of a shading element and saves it into a file
- If you attach this shader to your scene and render it, you get this entirely uninteresting image back

Surface Elements



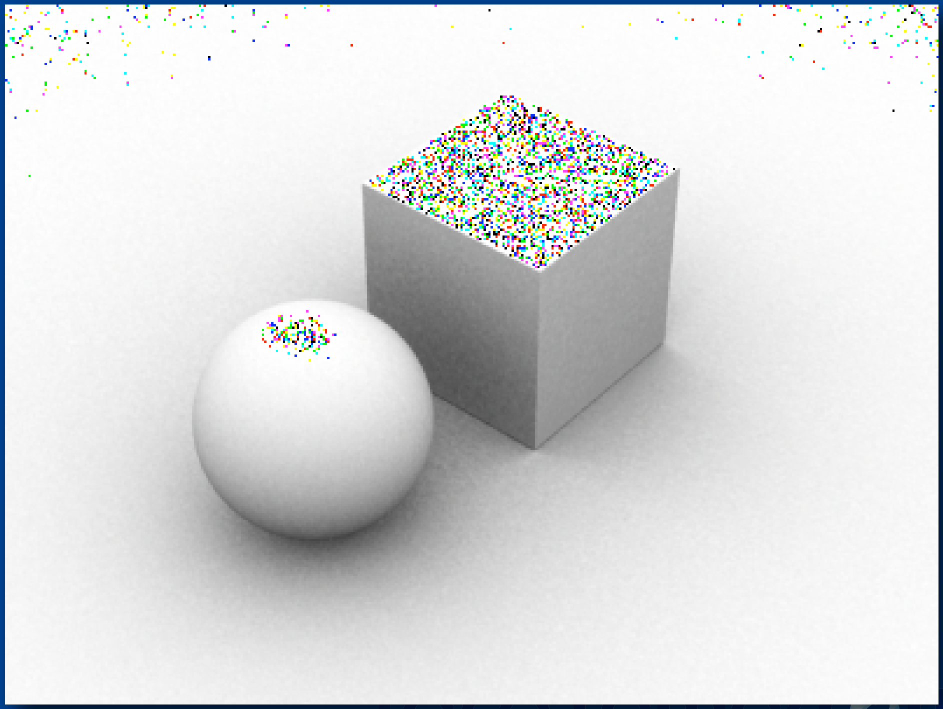


- However, the renderer also generates another file which contains the surface area elements as tiny little discs
- Pixie comes with a program that allows you to visualize these disc clouds
- Here's how it looks like
- Notice that each of these tiny discs correspond to a shading point and the size of the discs are proportional to their shading area
- One thing that you need to watch out for is turning on the non raster orient dicing to ensure that the renderer renders the back facing and occluded portions of the scene and creates a uniform sampling pattern on the surfaces

RenderMan - Point Based AO

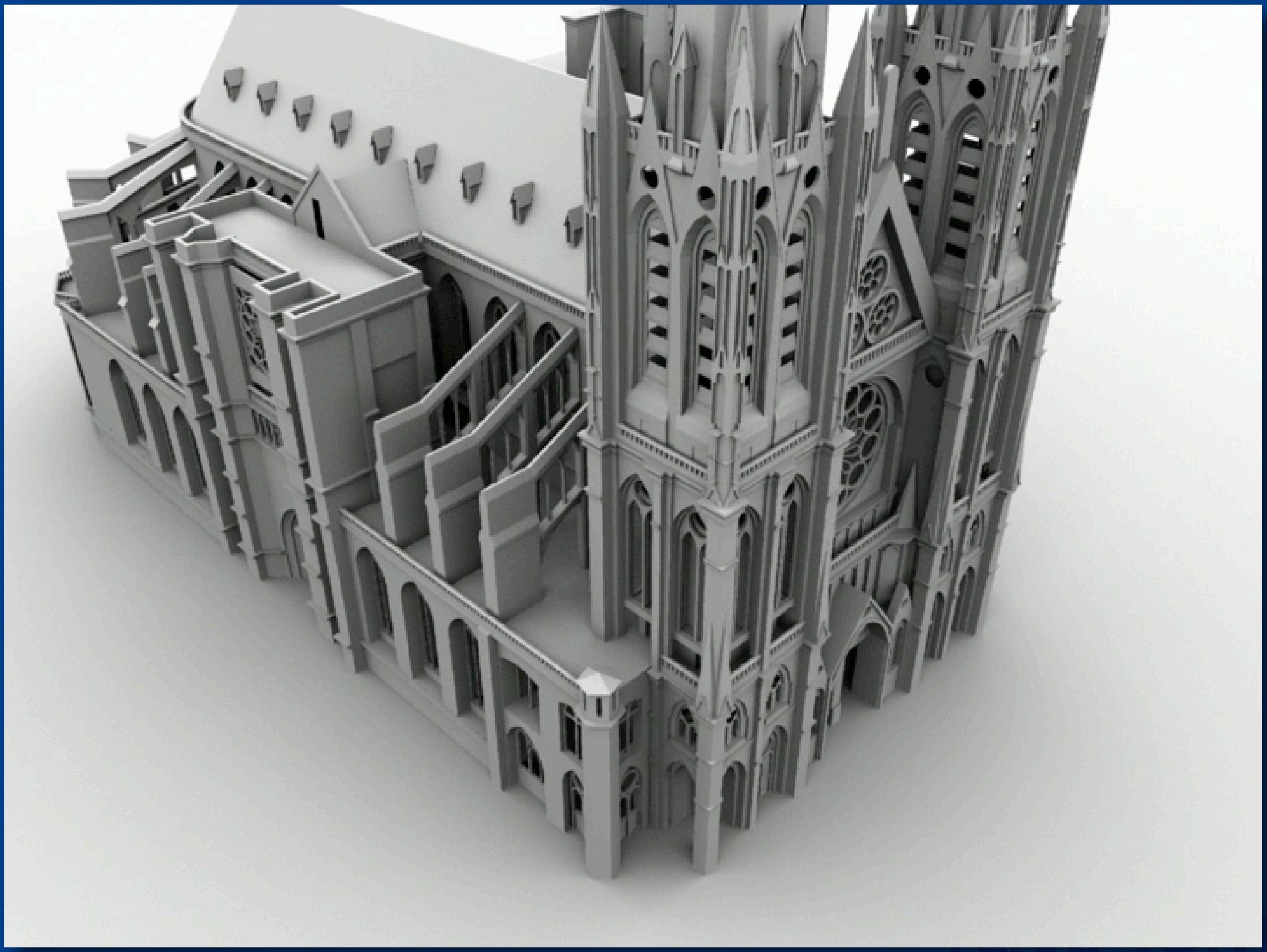
```
// Shader
surface use_areas (string filename = "") {
   normal Ns  = faceforward(normalize(N), I);
   float  occ = occlusion(P, Ns, 0, "pointbased", 1, "filename", filename);

   Ci = (1 - occ) * Os;
   Oi = Os;
}
```

Not a single ray is killed



- In a second pass, we can instruct the renderer to use the saved disc cloud to approximate the ambient occlusion or global illumination
- Here’s a surface shader that does that
- The built in function occlusion normally computes ambient occlusion by raytracing
- However we can give it the name of the disc cloud file and it will run this new algorithm and will gather occlusion from these surface area samples
- The cool thing about this image is that a) we did not shoot a single ray to compute it and b) we can do this on any type of surface geometry, from polygons to subdivision surfaces to hair and fur
- Although it is an approximation, it is quite a high quality one

Perfect for complicated scenes



- We can safely use this method to approximate global illumination and ambient occlusion on complex scenes such as this
- Because the method does not shoot rays or samples the incident hemispheres, it does not miss small but important surface detail

PIXIE
http://pixie.sourceforge.net

- The nice this is that the method comes with full source code
- You can download Pixie from http://pixie.sourceforge.net