The Bidirectional Scattering Distribution Function as a First-class Citizen in *Radiance*

Greg Ward, Anyhere Software

Talk Overview

- Why aren't the existing BSDF types first-class citizens?
- What does the new *BSDF* primitive do?
- How does it work?
- How does it relate to other methods in *Radiance*?
- What are its limitations?
- Future outlook

What Does It Mean to Be a “First-class Citizen?”

- Material must be included in all relevant parts of the rendering equation:
  - Direct (including specular highlights)
  - Mirror & transmitted components
  - Indirect diffuse (irradiance cache)
  - Indirect scattering (non-Lambertian)
- Existing BSDF types do not incorporate this final component
  - Instead, they include this missing energy as part of the indirect diffuse

Existing *Radiance* BSDF Materials

- *plasfunc*, *metfunc*, *transfunc*
  - User provides functional description of specular lobe
- *plasdata*, *metdata*, *transdata*
  - User provides data array for specular lobe with functional lookup
- *BRTDfunc*
  - User provides separate functions for mirror, transmitted, and specular lobes
  - Can differentiate between front and back reflection

What’s So Hard about Indirect Scattering?

- We need a method to send out weighted ray samples
- We could send out uniformly distributed rays and weight them using the BSDF
  - This ends up being a stupendous waste for highly peaked functions
- A better approach is to tabulate the cumulative BSDF and invert it
  - We can then weight our samples uniformly -- much closer to optimal
  - But this turns out to be impossible for arbitrary procedural definitions
  - Even *plasdata* and friends have functional coordinate mappings, so fail again

How Do Other *Radiance* Materials Do It?

- *dielectric*, *glass*, *mirror*, etc. have only pure specular components
  - No Monte Carlo sampling is required by these types
- *plastic*, *metal*, *trans*, *plastic2*, etc. use Gaussian model for specular lobes
  - Well-behaved functions with known inversion formulas
  - David Geisler-Moroder and Arne Dür have spent considerable effort making the sampling more accurate (see past workshop presentations)

Introducing the *BSDF* Material Primitive in *Radiance* 4.1

- General, data-driven reflectance and transmittance distribution function
- Simple syntax relies on XML (eXtensible Markup Language) auxiliary file
- XML file may be imported from WINDOW 6 or created using **genBSDF**
- Proxy mode reveals detailed model underneath, similar to *illum* behavior

```
void BSDF m_bsdf110b
6 0 bsdf110b.xml 0 1 0 .
0
0
```

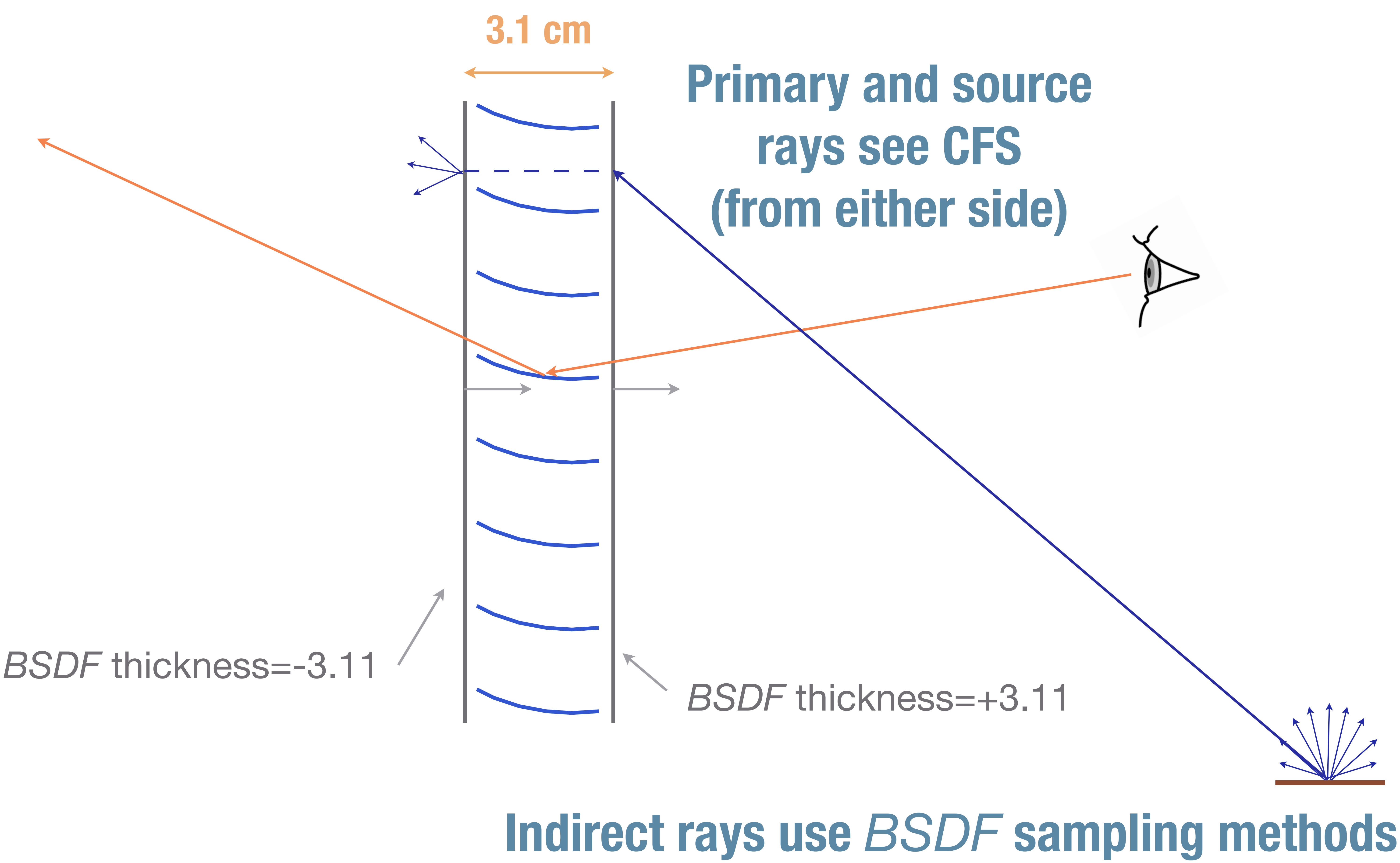
Placeholder for function file (if needed)

Thickness (non-zero for proxy)

Auxiliary XML with BSDF data

Up orientation vector

Proxy Example



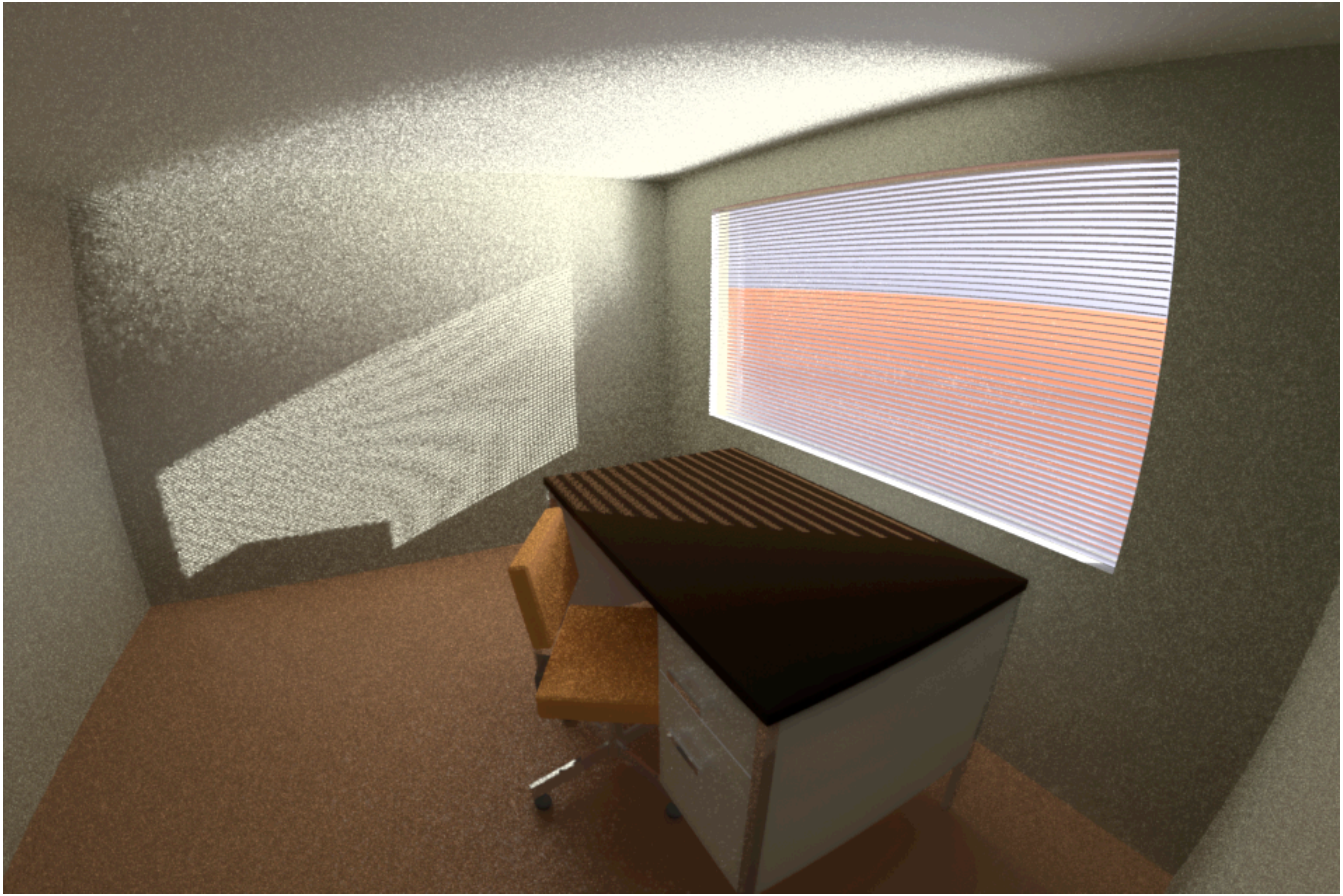
Using **genBSDF** to Create XML File

```
genBSDF +geom centimeter blinds.rad > blinds.xml
```

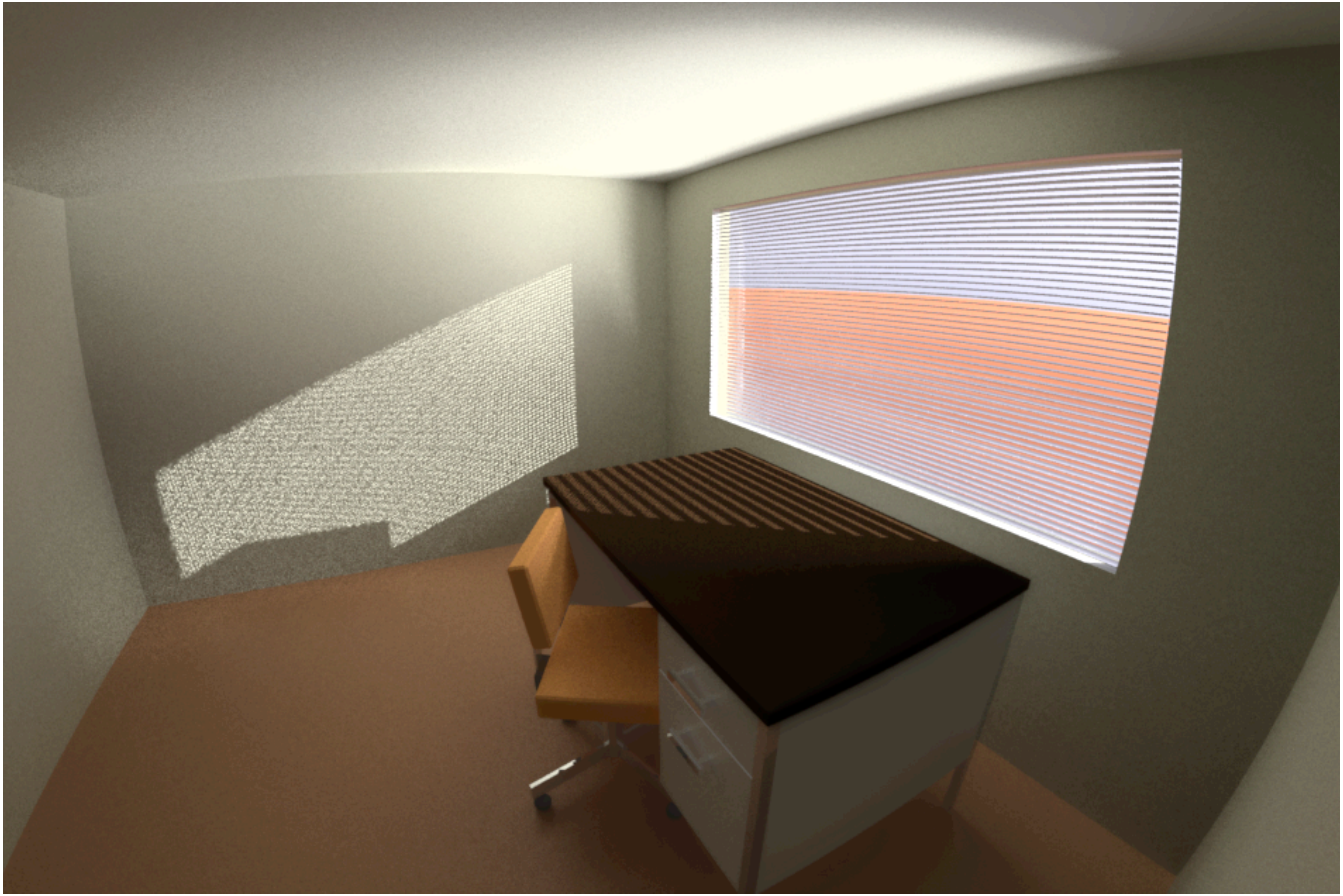
Using **pkgBSDF** to extract geometry:

```
!pkgBSDF -s blinds.xml
```

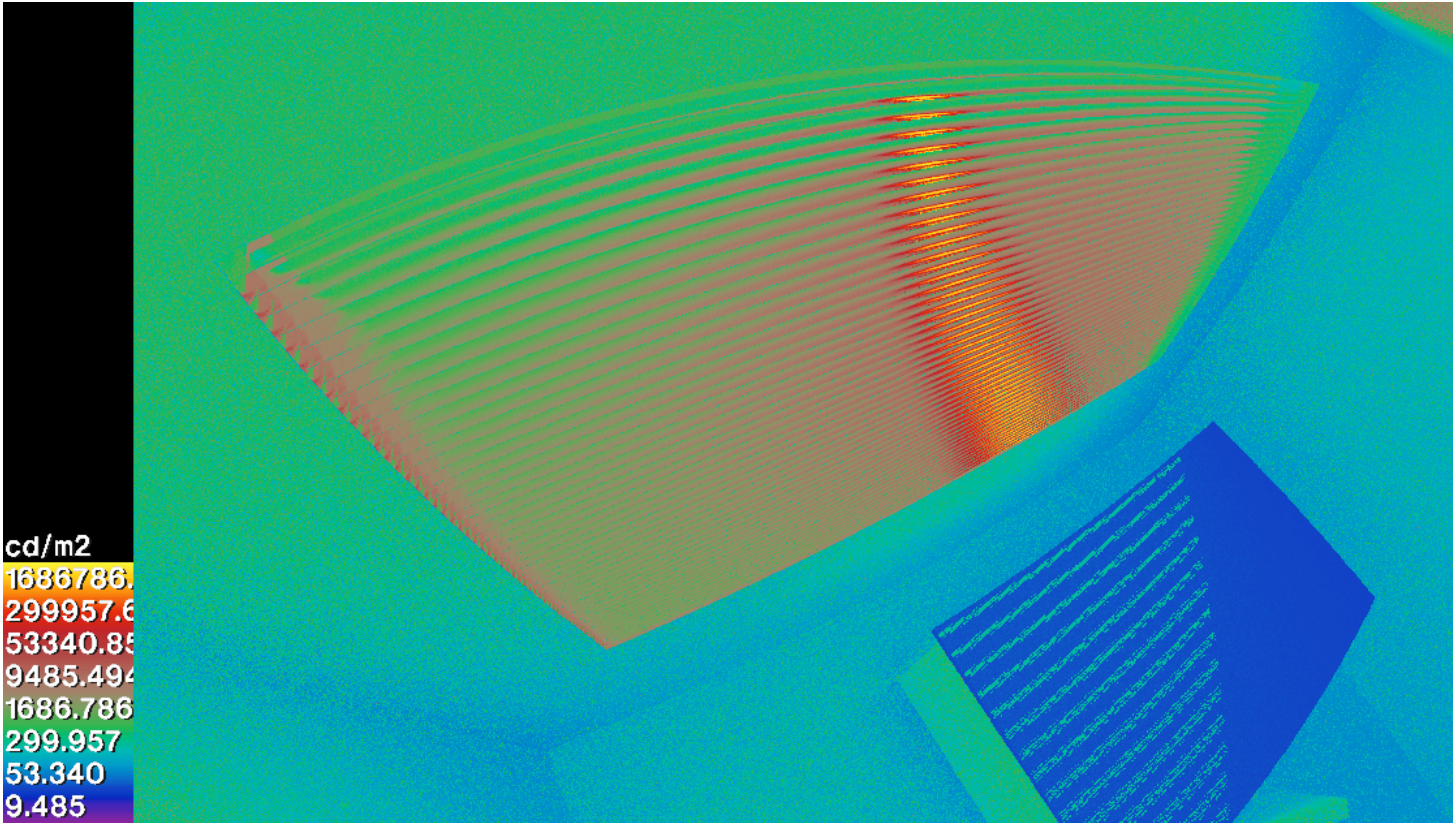
Converts MGF to *Radiance* and places proxy surfaces in front and behind (if appropriate)



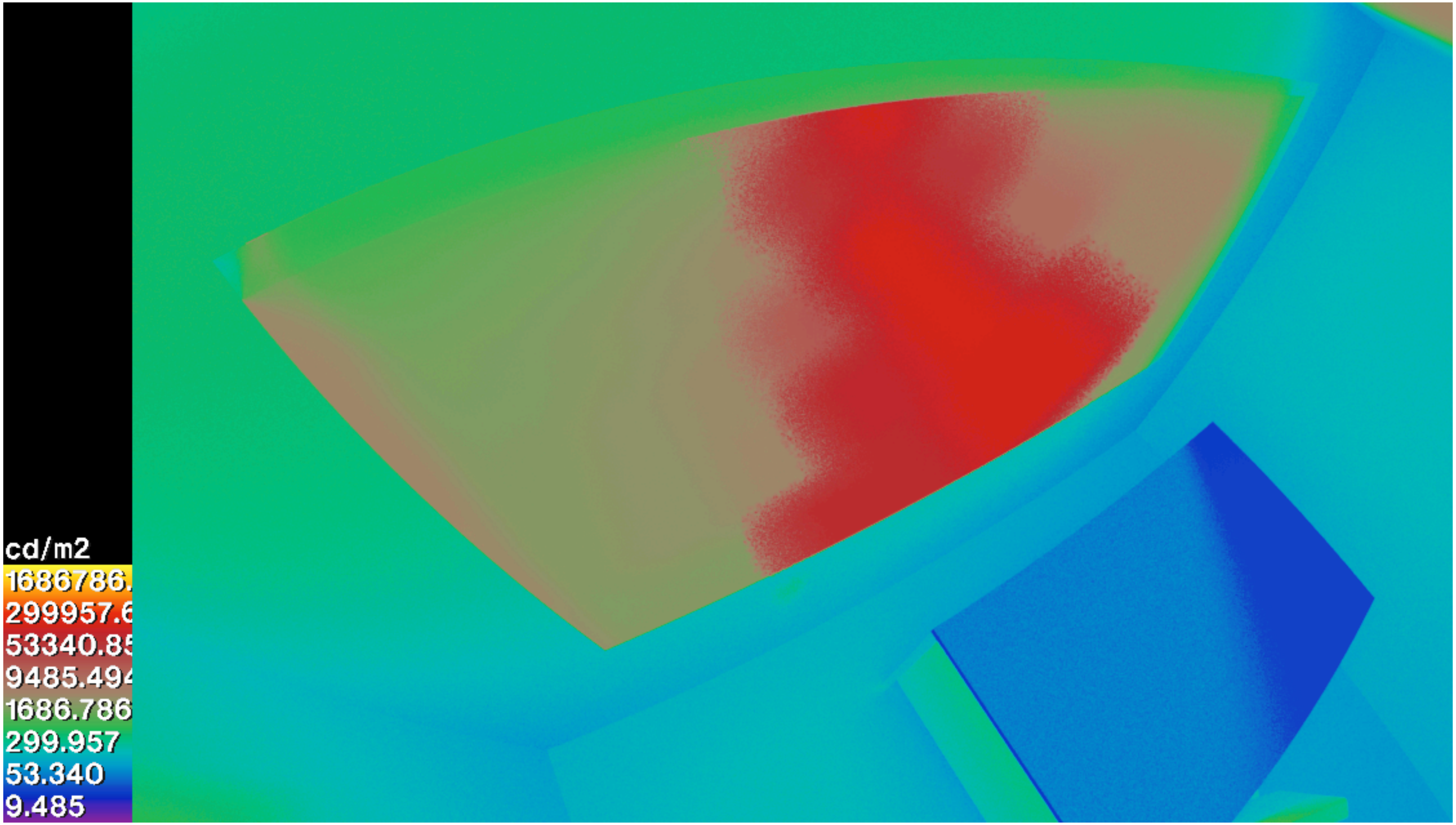
Rendering using blind geometry directly
Render time: 2.7 CPU hours



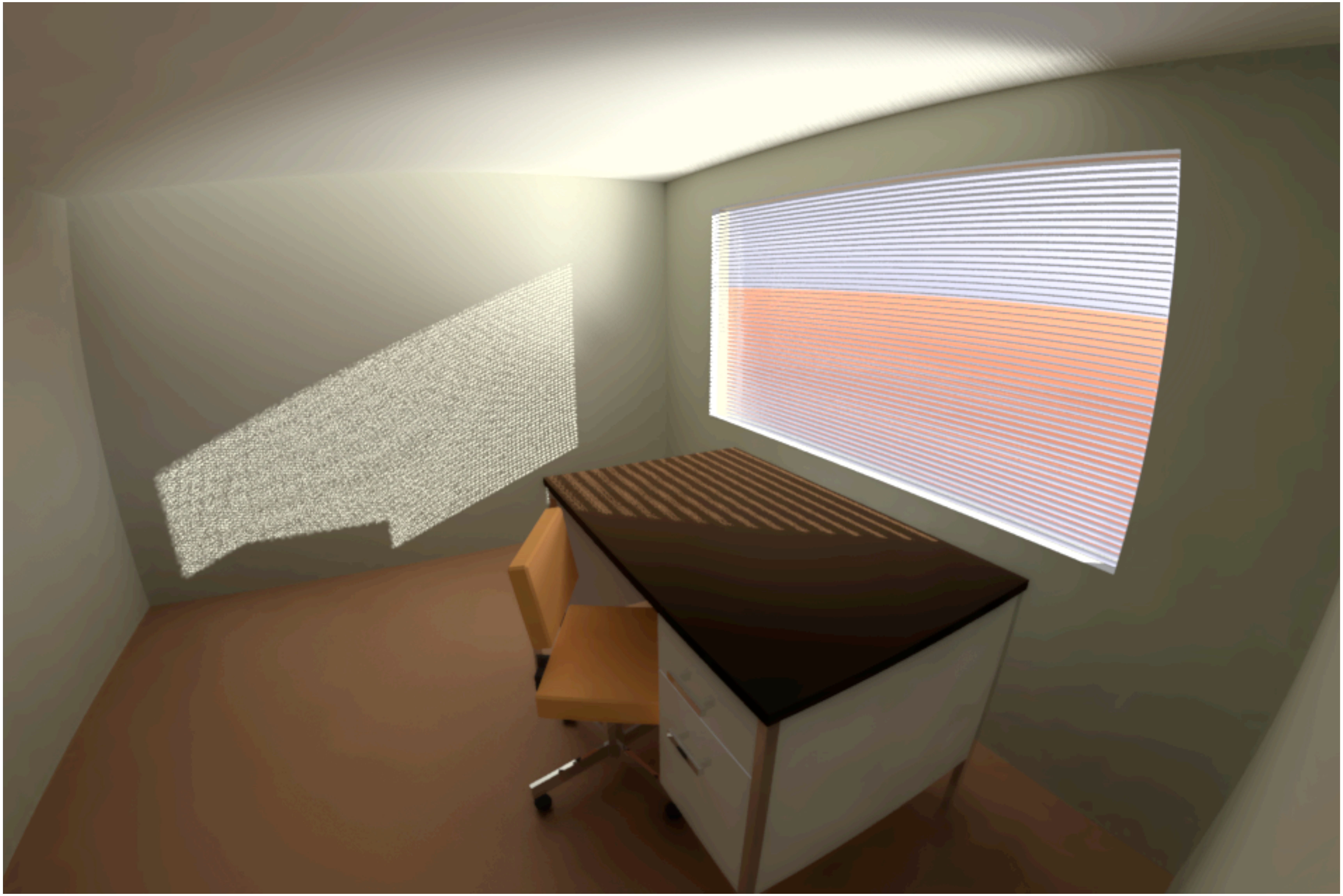
Rendering result using *BSDF* and proxied geometry
Render time: 3.7 CPU hours



The straight geometry rendering so much noisier -- why?
Because this is the view of the window from the ceiling



Using a *BSDF* surface, indirect rays see a simplified picture
This is still not very nice to have in the indirect calculation, however



With or without a *BSDF*, **mkillum** can improve the results considerably
Putting an *illum* on the window removes this from the indirect portion

So, What Does a *BSDF* Do, and Why Do We Care?

- So far, we haven't made a very compelling case for the *BSDF* type
- In truth, there are three instances when you really need it:
    1. When the *Radiance* model is complex and/or highly specular
    2. When you wish to perform annual simulations
    3. When you are handed a BSDF instead of a *Radiance* model
- We hope the last case will become more common as time goes on....

How Does the *BSDF* Type Work?

- There are currently two implementations underlying the *BSDF* primitive
  - One is for matrix BSDF data
  - The other is for variable-resolution BSDF data
- These are integrated under a common Application Programming Interface
- We are sharing this API with other application developers wishing to use BSDFs
- New data subtypes may be incorporated into the XML spec and our library

Operations supported by the BSDF API

- Load and/or cache BSDF data from a XML input file
- Separate diffuse portion and extracts geometry (MGF) if any
- Evaluate a BSDF for a given vector pair (incident and exitant directions)
- Query the resolution (solid angle) of a BSDF given a vector or vector pair
- Integrate the hemispherical reflectance or transmittance for a vector
- Generate a uniformly distributed Monte Carlo sample for an incident vector
- All operations commute between incident and exitant directions (reciprocity)

Additional API Features

- Computes global ↔ local coordinate transformations
- Separate access to BSDF components (transmitted & reflected, directional & diffuse, front & back)
- Caches cumulative sampling tables for efficiency
- Permits access to underlying data structions (i.e., BSDF matrix, Tensor Tree)
- Convenient error diagnostics & reporting
- Permits multiple simultaneous representations & spectral data (unused for now)

How Is the BSDF Library Used in *Radiance*?

- *BSDF* rendering routine loads and caches XML auxiliary files
  - Evaluates BSDF function at multiple sample points for light sources
  - Uses API's stratified sampling method to choose specular directions
  - Uses indirect irradiance cache for diffuse components
  - Adds special checks for proxy behavior when thickness is non-zero
- Other programs, such as **dctimestep**, access BSDF matrix directly

API Handles the Tricky Part

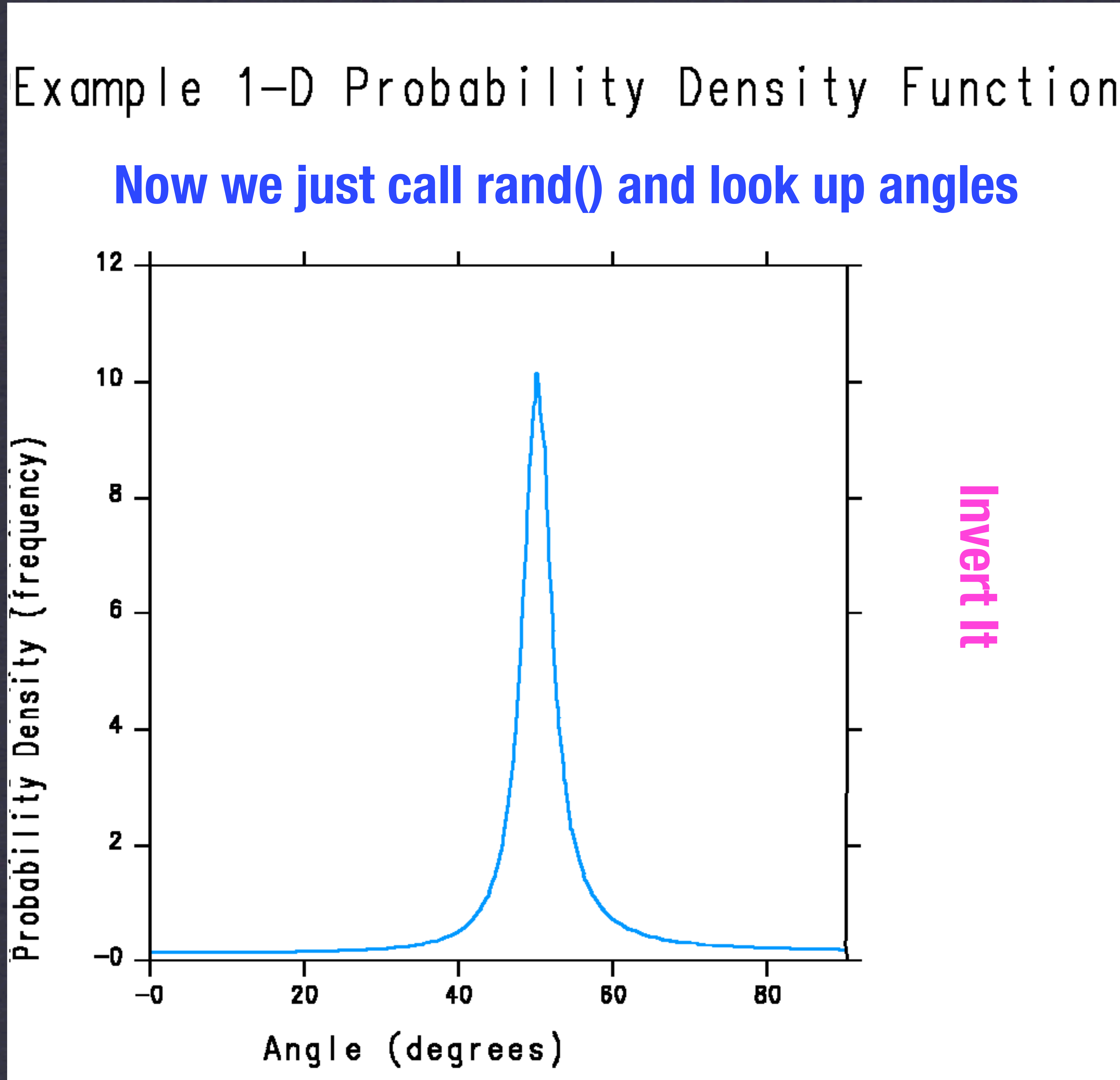
- Main challenge to general BSDF sampling is Monte Carlo inversion
  - Given a particular incident angle, where do we want to send our samples?
  - Sending multiple samples, how do we insure they are well-separated?
- This gets even trickier for variable-resolution data
- BSDF library caches cumulative tables based on query directions
- Caller may release the cache at any time and tables will be rebuilt as needed

Review of Monte Carlo Inversion

Convert a uniform random variable, X ∈ [0,1) into a properly distributed value on the sample domain

Start with a probability density function, which we can think of as a 1-dimensional BRDF

Accumulate densities and normalize to arrive at an invertible distribution

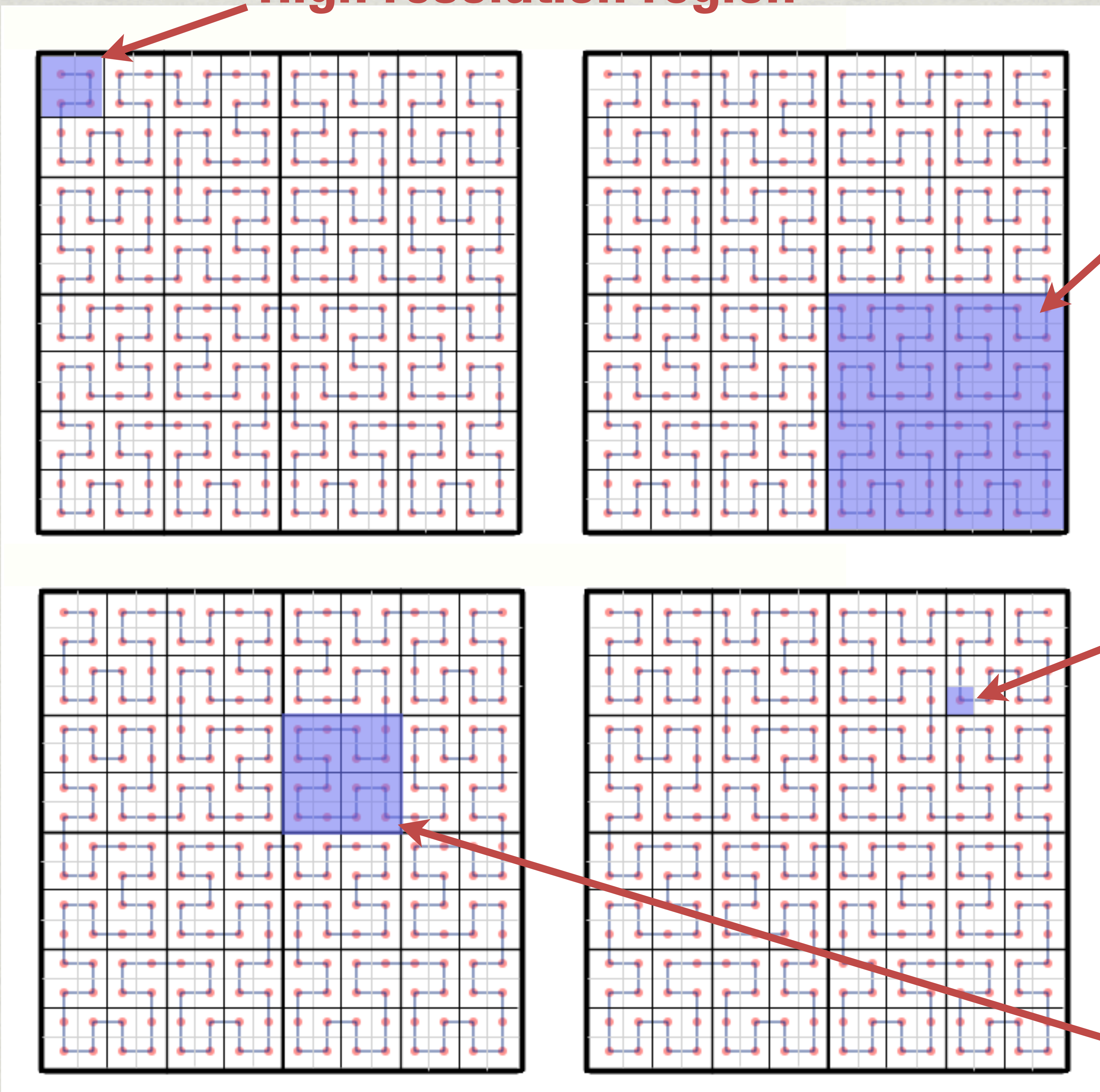


MC Inversion in Two Dimensions

- Direction is two-dimensional (e.g., altitude & azimuth), so how do we extend this?
- In the case of a matrix BSDF, we base 1-D cumulative table on the Klems index
- For variable-resolution BSDF, we use a Hilbert curve that maximizes neighbor proximity and thereby improves stratified sampling
- We can do all this because we have a finite number of BSDF values
    - I.e., we know where those values are and their sizes

Hilbert Curve in 2-D

High resolution region



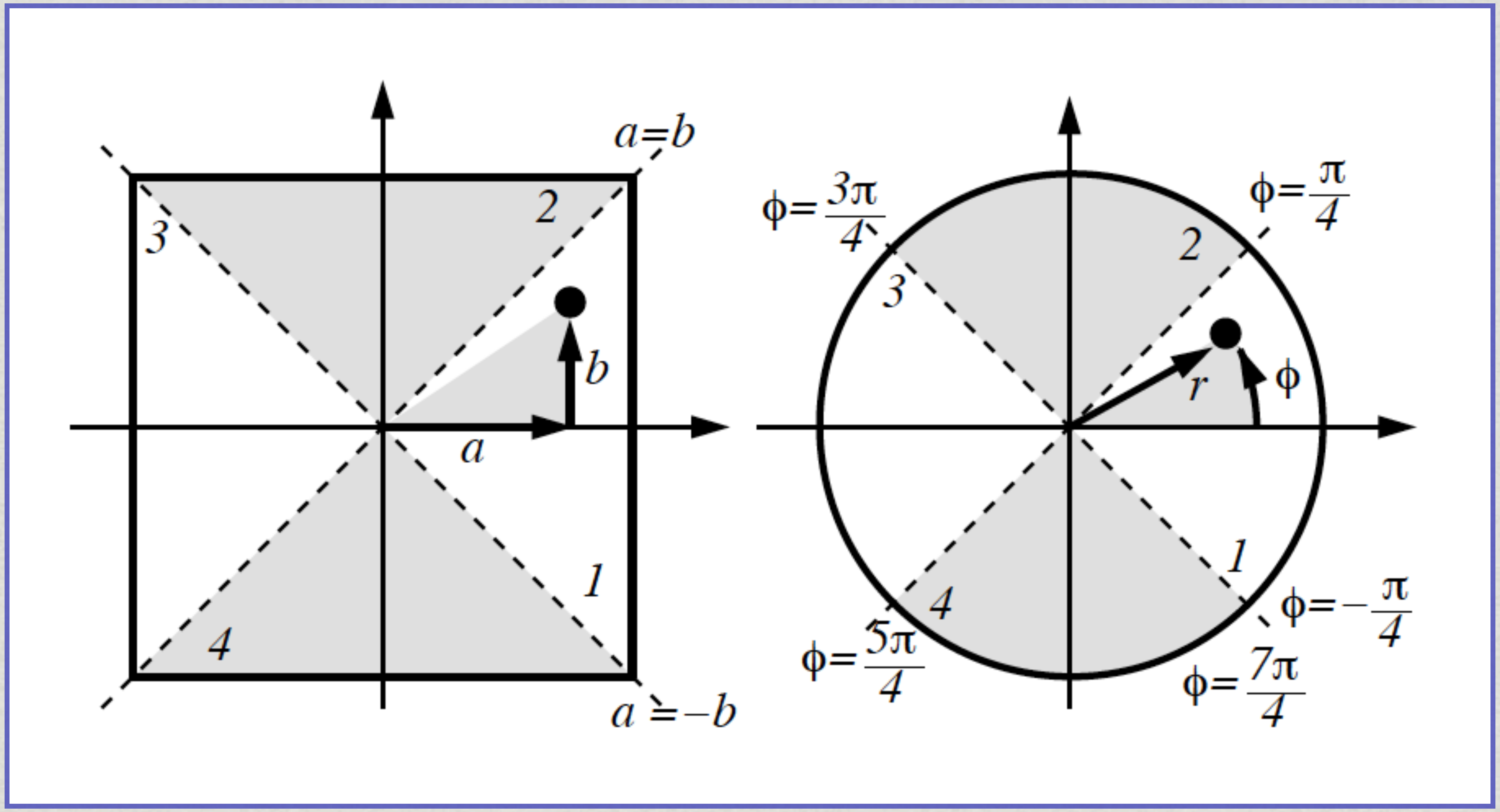
Low resolution region
(nearly diffuse)

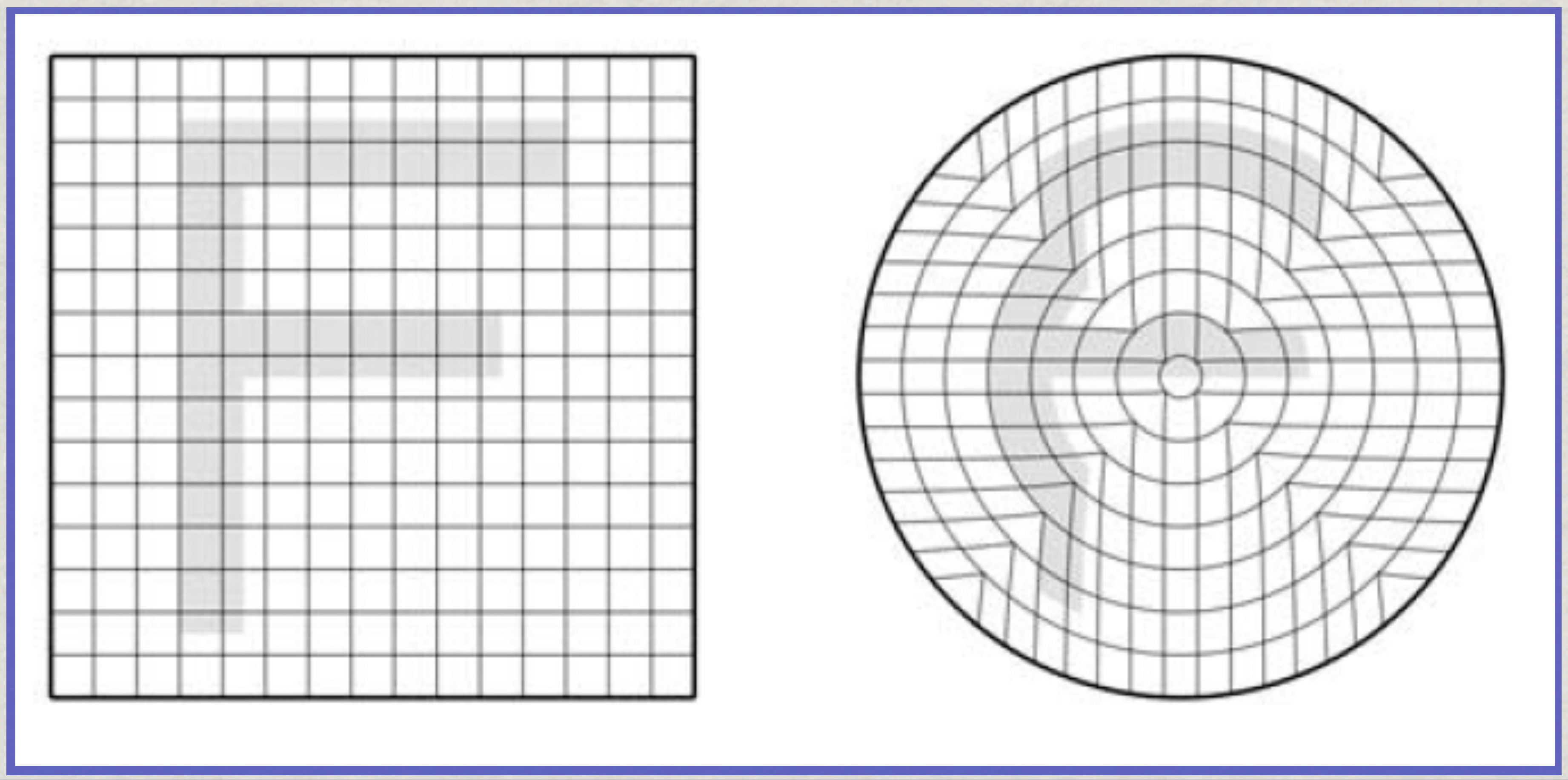
Spike in BSDF

Hilbert curve winds through
our 2-D direction space
& subdivides each region

Medium resolution region

Shirley-Chiu Mapping





Maintains relative areas, important for hemispherical sampling

Peter Shirley and Kenneth Chiu, "A Low Distortion Map Between Disk and Square," JGT 2(3), 1997

Sampling Summary

- Sampling method in BSDF library is fast and memory-efficient
- There are always issues with sampling, since it creates noise in renderings
- The **-ss** option can be used to increase sampling rate and reduce noise
- **mkillum** is still valuable as a means to improve rendering performance
- **mkillum** access to BSDF data will be removed in upcoming release
  - BSDF sampling is more general in rendering code
  - Incorporates reflection and variable-resolution data

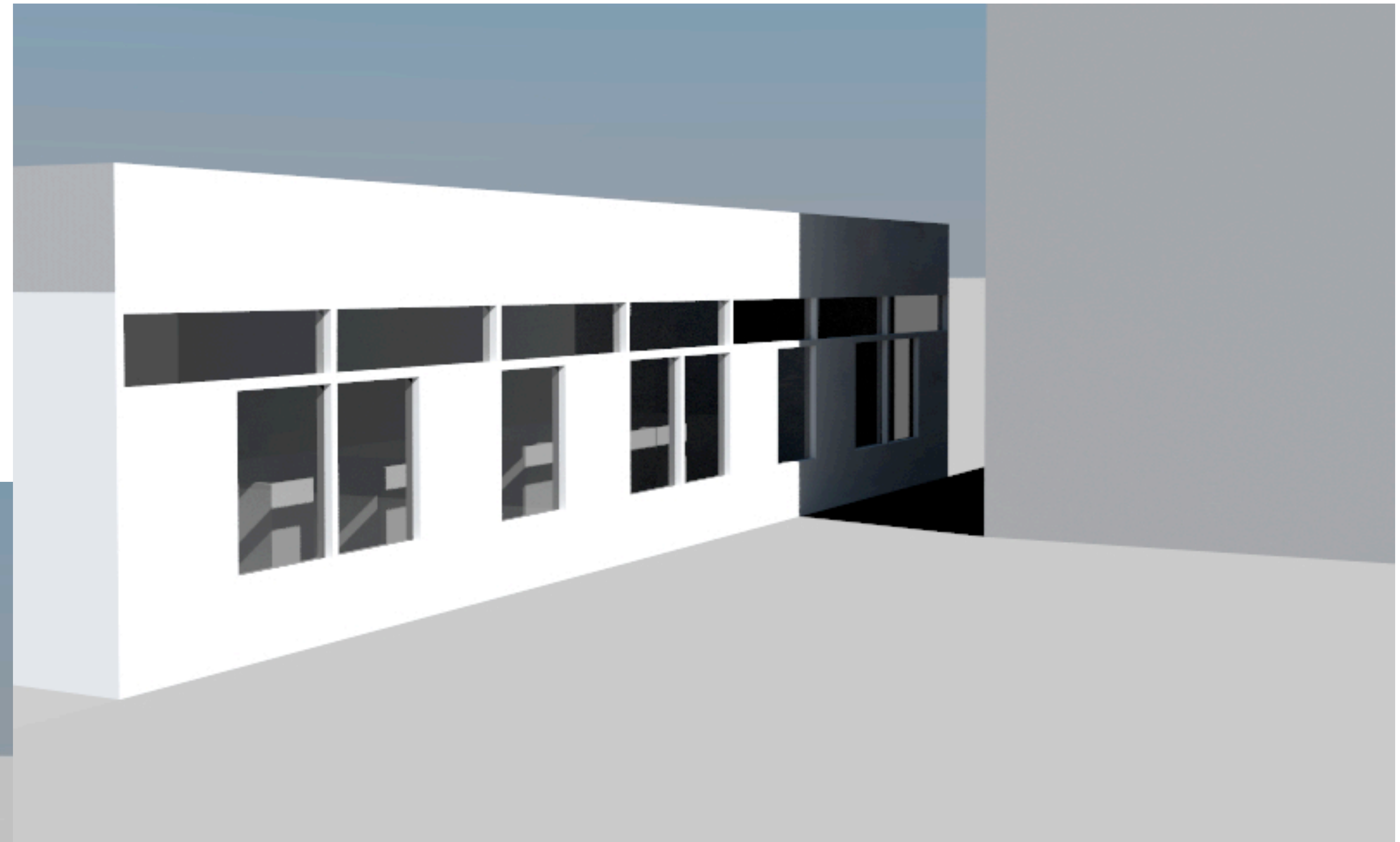
How Does New *BSDF* Material Fit into *Radiance* Ecosystem?

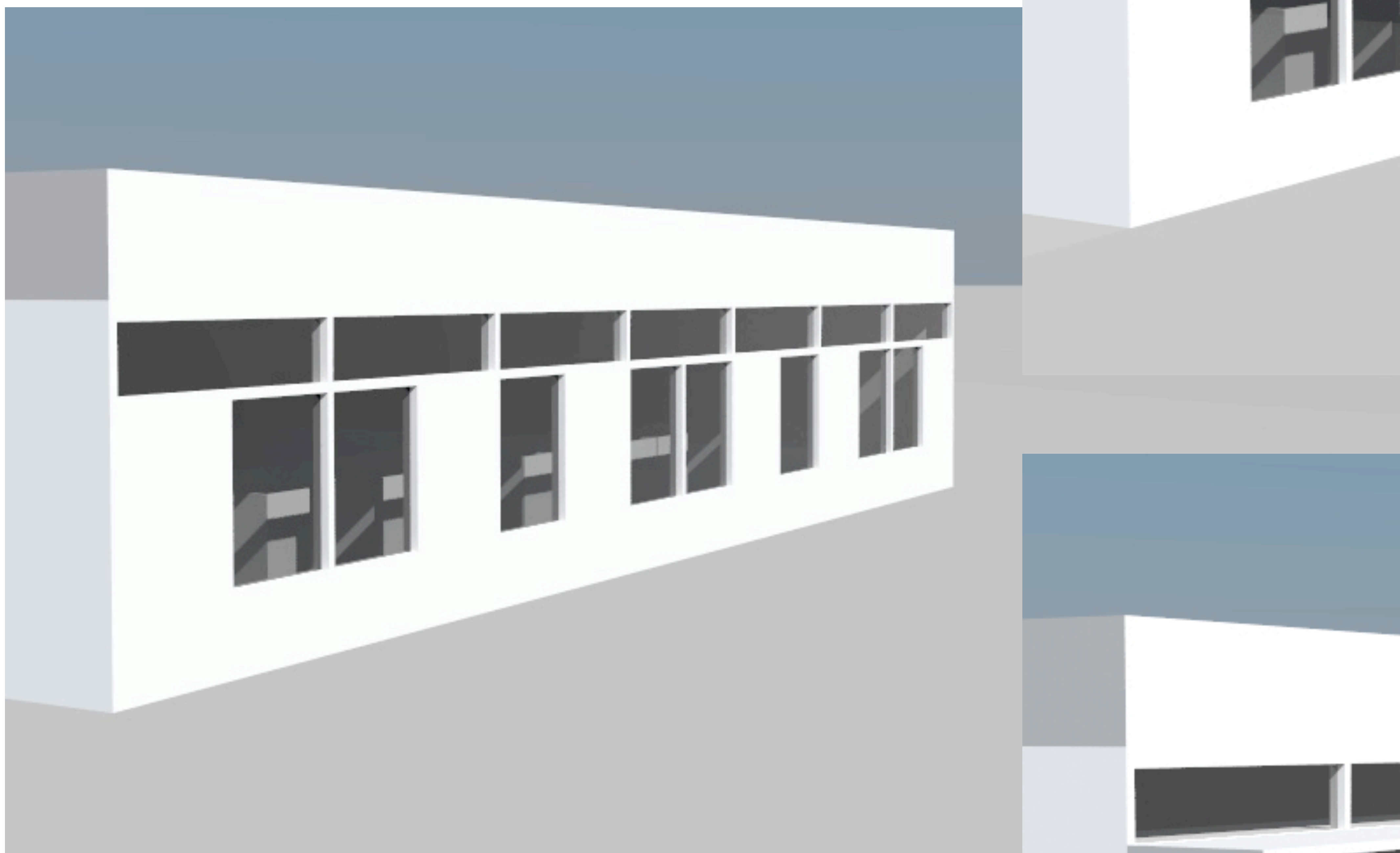
- As a first-class citizen, it participates like most materials applied to surfaces
- In proxy mode, we can use it as a stand-in for detailed appearance geometry
    - Detail geometry is also used in shadow-testing for complex fenestration
- We can use this new material in a completely general way, in *any* calculation
- In particular, we can use it in a daylight coefficient approach for annual simulation

The *BSDF* Type in Annual Simulations

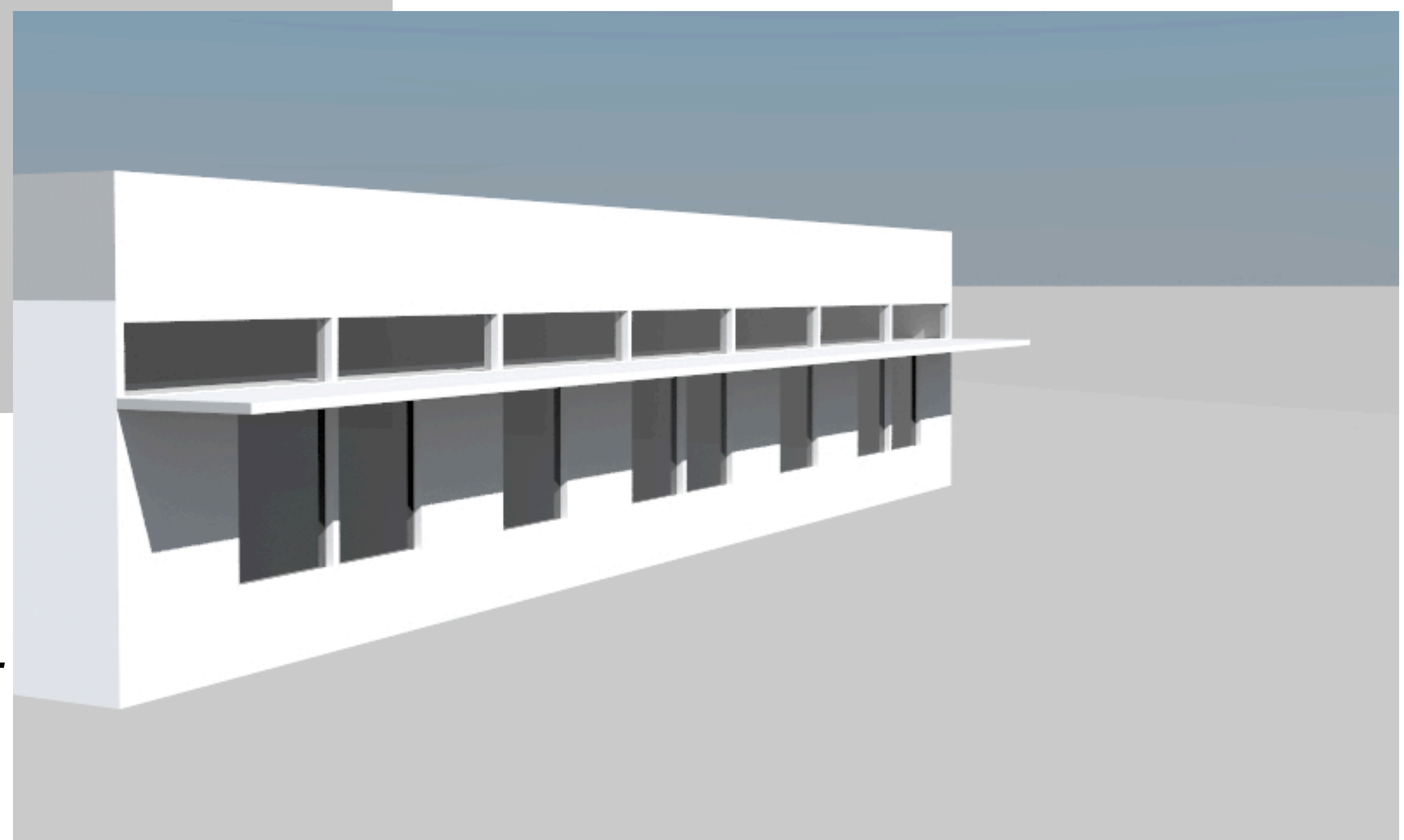
- The three-phase method relies on matrix BSDF data for annual simulation
  - Does not use the *BSDF* material type
  - Allows replacement of different CFS configurations in the hourly loop
  - Requires intelligent subdivision of windows where there is additional façade geometry (e.g., fins or overhangs) or nearby structures
- Using the new *BSDF* type and a more traditional DC approach, the calculation becomes simpler and handles nearby geometry naturally
  - May take longer to set up for multiple CFS configurations, however

Nearby geometry needs to subdivide window for 3-phase DC method





No special treatment needed for standard DC computation using *BSDF* material



Other Radiance Tools that Use or Produce BSDFs

- **genBSDF** creates XML from *Radiance* or MGF model
  - Now computes BRDF as well as BTDF
  - New output options produce variable-resolution data
- **pkgBSDF** places *BSDF* surface with geometry converted from XML file
- **dctimestep** loads matrix from XML file for annual simulation inner loop
- **mkillum** has the ability to load and utilize XML matrix data
  - No longer needed thanks to new *BSDF* type

What Are the Current Limitations of Our BSDF Implementation?

- BSDF XML data is completely uncolored
  - *BSDF* primitive accepts colored patterns and additional diffuse components
  - If and when future definitions in XML provide spectral data, we will support it
- Proxy mode only appropriate for “thin” systems
- Other practical problems:
  - Computing high-resolution BSDFs is very expensive using **genBSDF**
  - Sampling generates noise, so **mkillum** is still needed for clean CFS renderings

Future Outlook for *BSDF* Type

- First general, data-driven material to be added in *Radiance*
- Utilizes library/API that is extensible and designed to be shared and used by other simulation software
    - Ian Ashdown, author of Helios and AGI32, is on board
- Others may add new BSDF data types in the future, and provide data
    - Jérôme Kämpf of EPFL has tested rectangular matrix data
- Plan to convert measured BSDFs to variable-resolution XML data