Mapping obstructions and reflections for controlling electrochromic windows

Andy McNeil

Part 1 - Mapping Obstructions

Obstructions…

- Cast shadows on windows

- Typically include:
  - Neighboring buildings
  - Another wing of the same building
  - Static Shading devices, eg. overhangs, fins
  - Trees?

- We avoid tinting windows that are already shaded by an obstruction.

HALIO

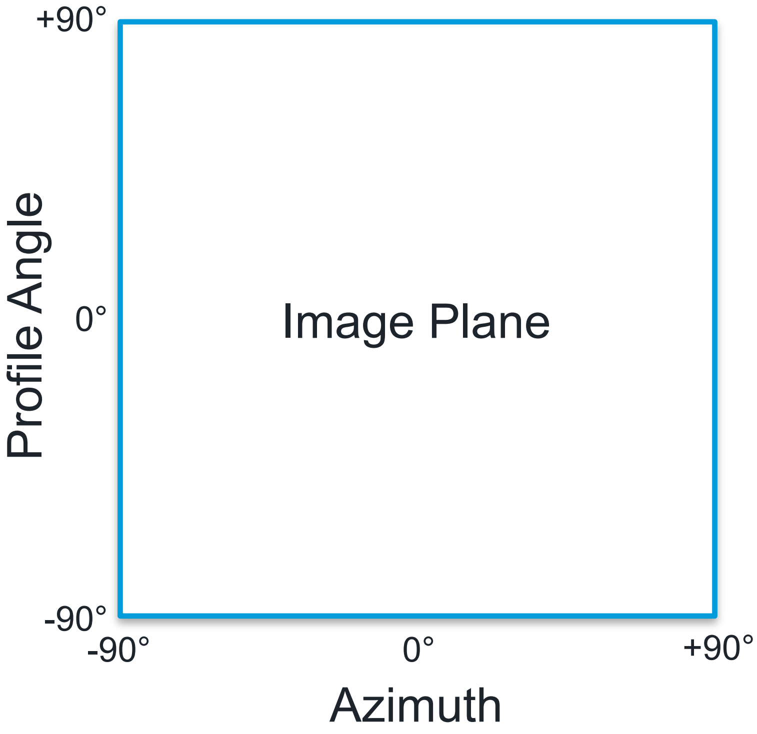
halioglass.com

Halio maps obstructions based on angular position, as viewed from the window.

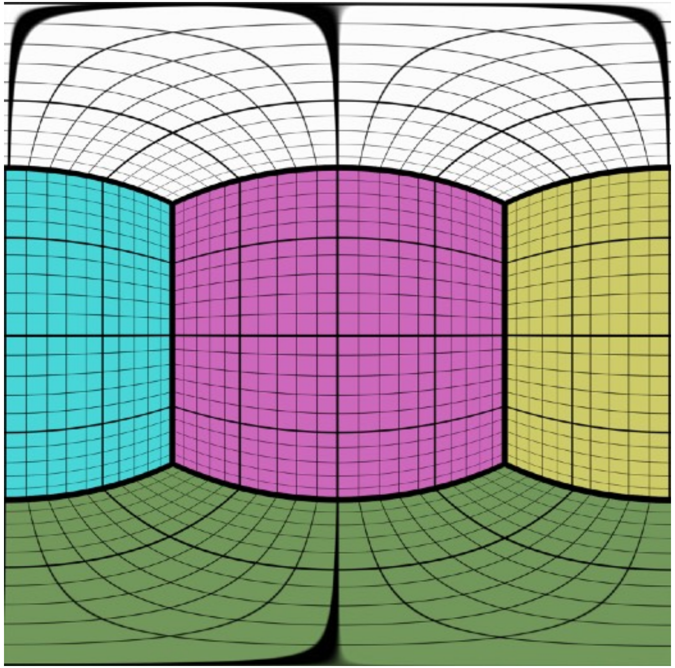
- Angle based mapping is
  - Independent of latitude, longitude, and facade orientation - Enter these things once in the control system, don't bake it into your data.
  - Visually understandable, and self documenting - you can look at the maps and understand what they contain.

Introducing ‘Orthonormal Pseudocylindrical’ Projection

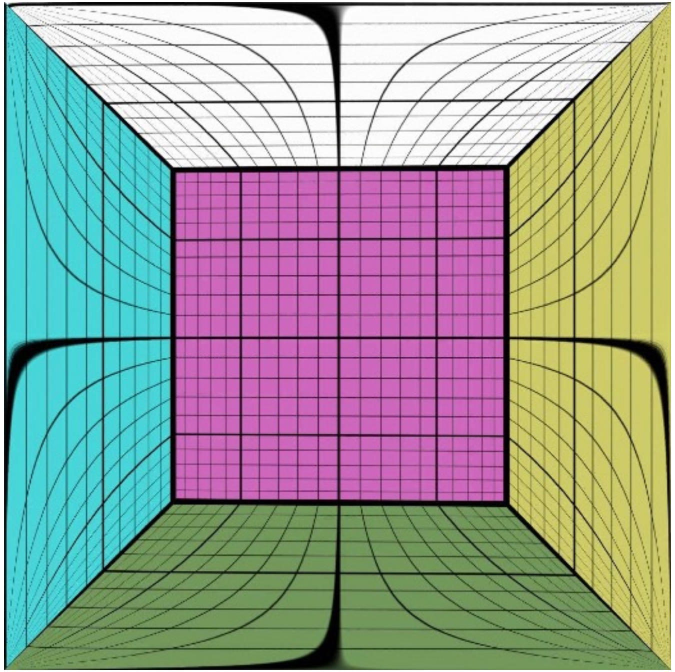
- X-axis:
  - azimuth angle
  - projected into horizontal plane
- Y-axis:
  - profile angle
  - projected into vertical plane
- Lines that are orthonormal to the direction of view are straight lines in the orthonormal projection.



Introducing Orthonormal Pseudocylindrical Projection



Equirectangular Projection



Orthonormal Projection

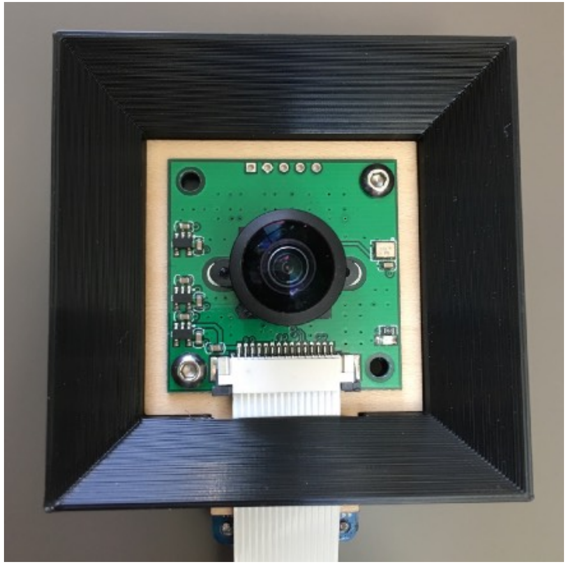
Three methods for generating obstruction maps

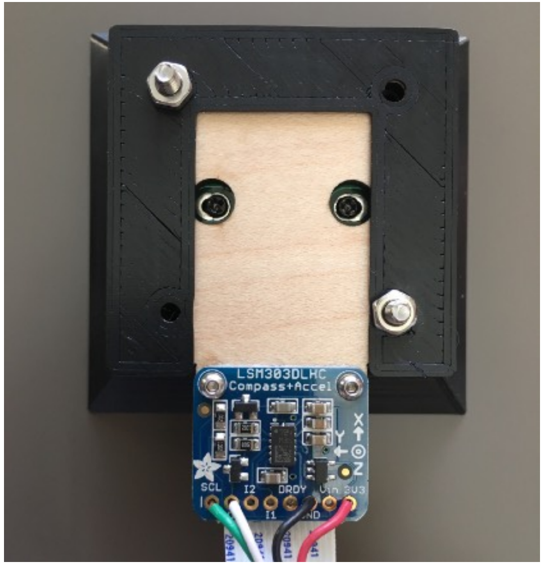
- **Geometric**: uses dimensional parameters for common obstruction types (overhangs, fins, cowls)
- **Ray Tracing**: Shoot rays through a 3D Cad model testing for sky or obstruction (Radiance!)
- **Photographic**: take a picture with a calibrated fisheye camera (real-time ray tracing)

Example Case - Hayward Office

HALIO
halioglass.com

Photographic method

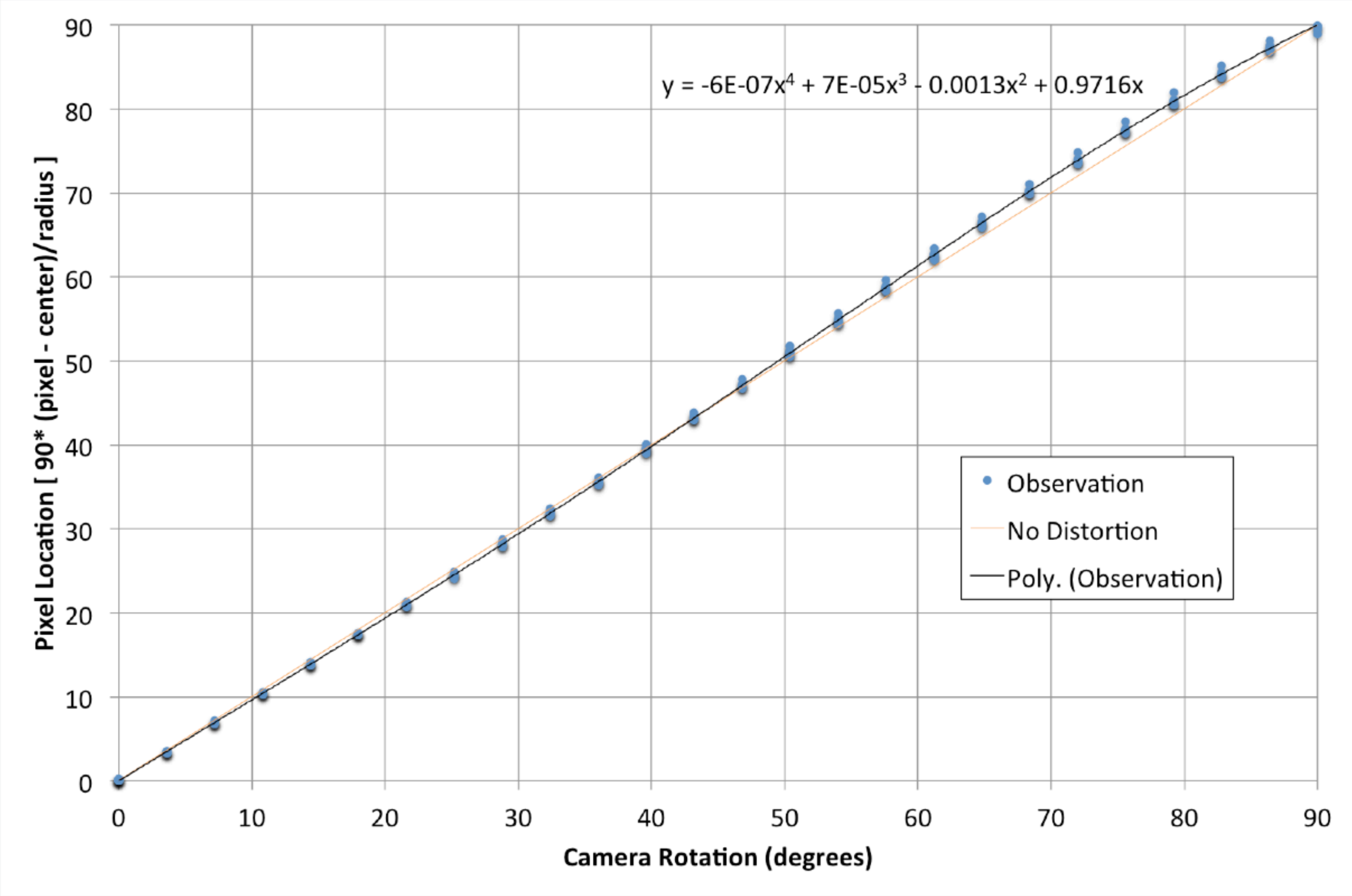




Prototype Camera

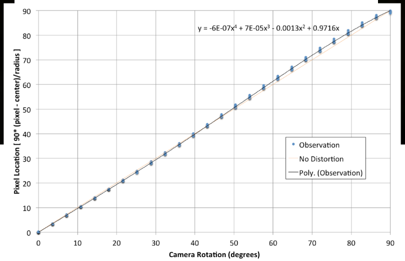
Accelerometer - Correct for Crooked Hand

Angular distortion correction



HALIO

Correct for angular distortion



HALIO

halioglass.com

Correct for angular distortion
It’s subtly but important

Correct for angular distortion
It’s subtly but important

HALIO

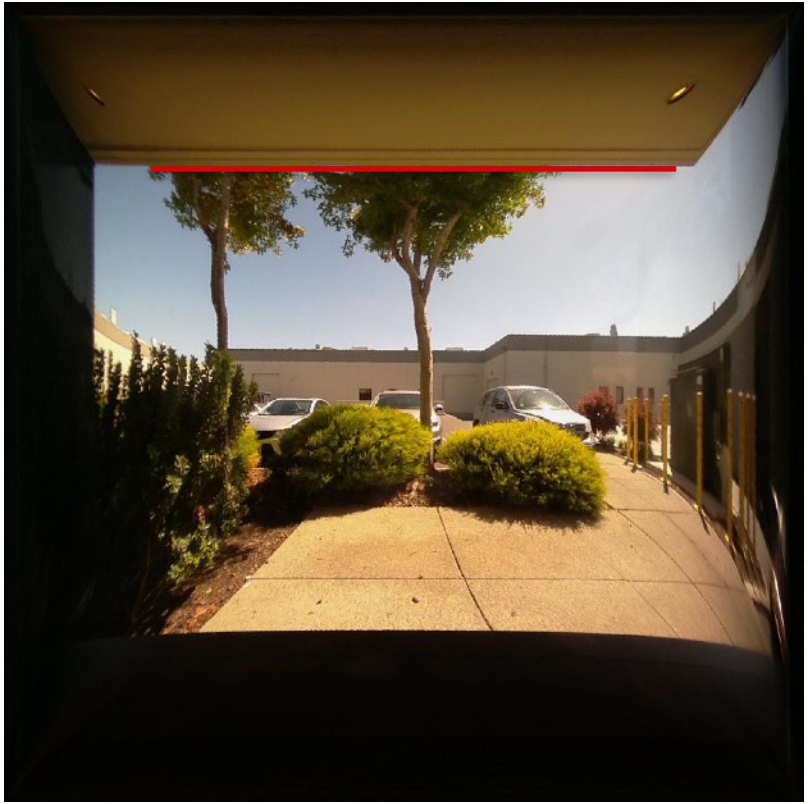
Reproject from angular to orthonormal

Angular Fisheye

Re-project from angular to orthonormal

Orthonormal

Re-project



Straight Line = Pleasing!

Orthonormal

HALIO

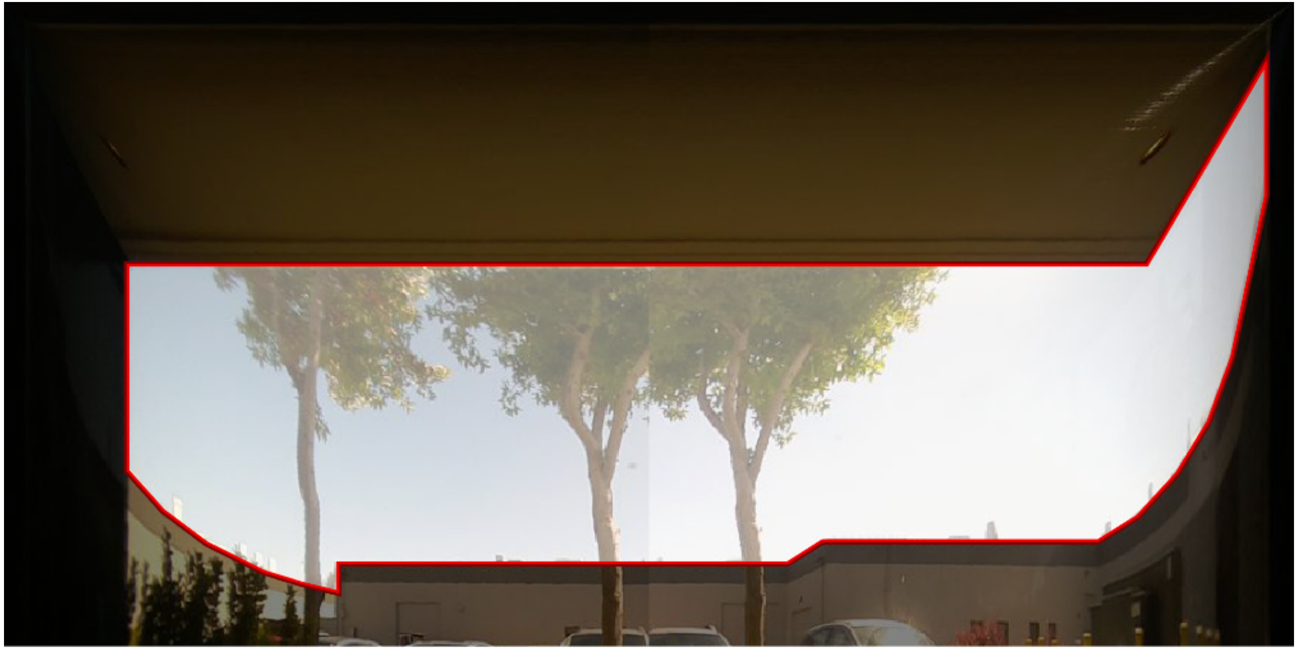
halioglass.com

Again, why we’re not using equirectangular…

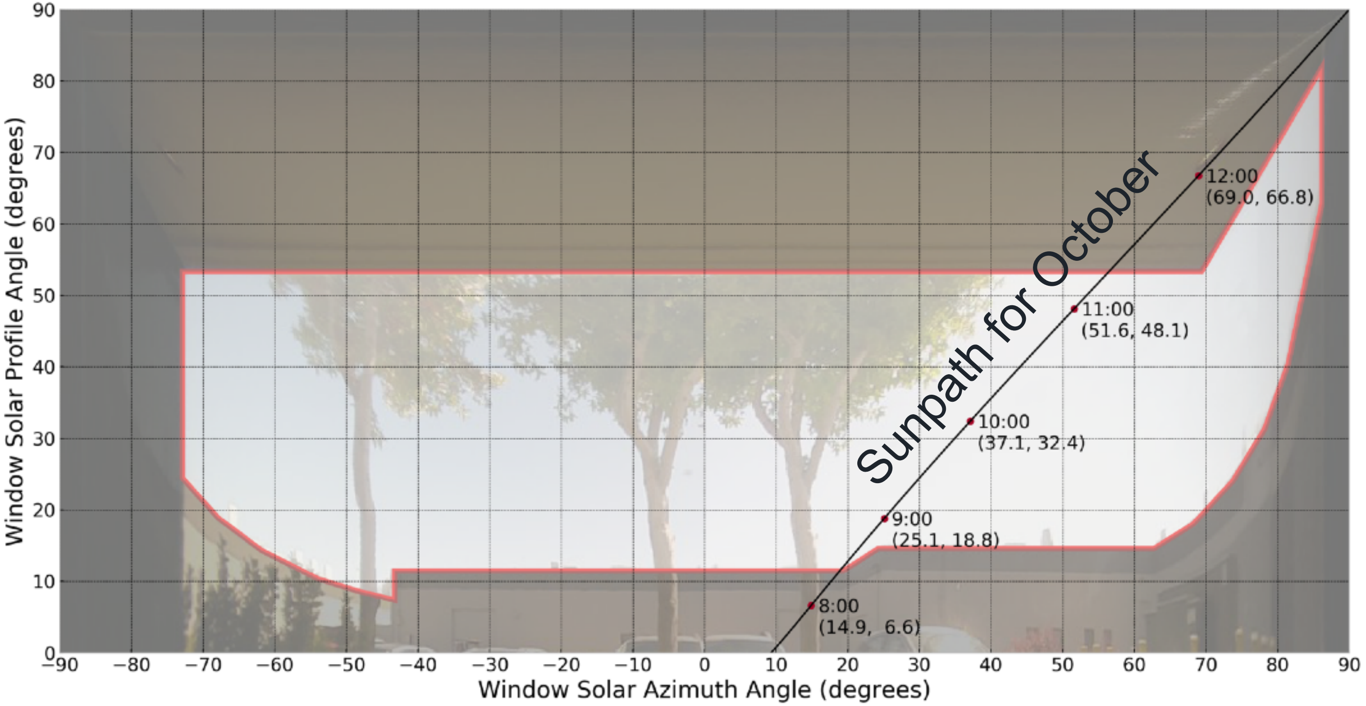
Not straight line = awkward

Equirectangular

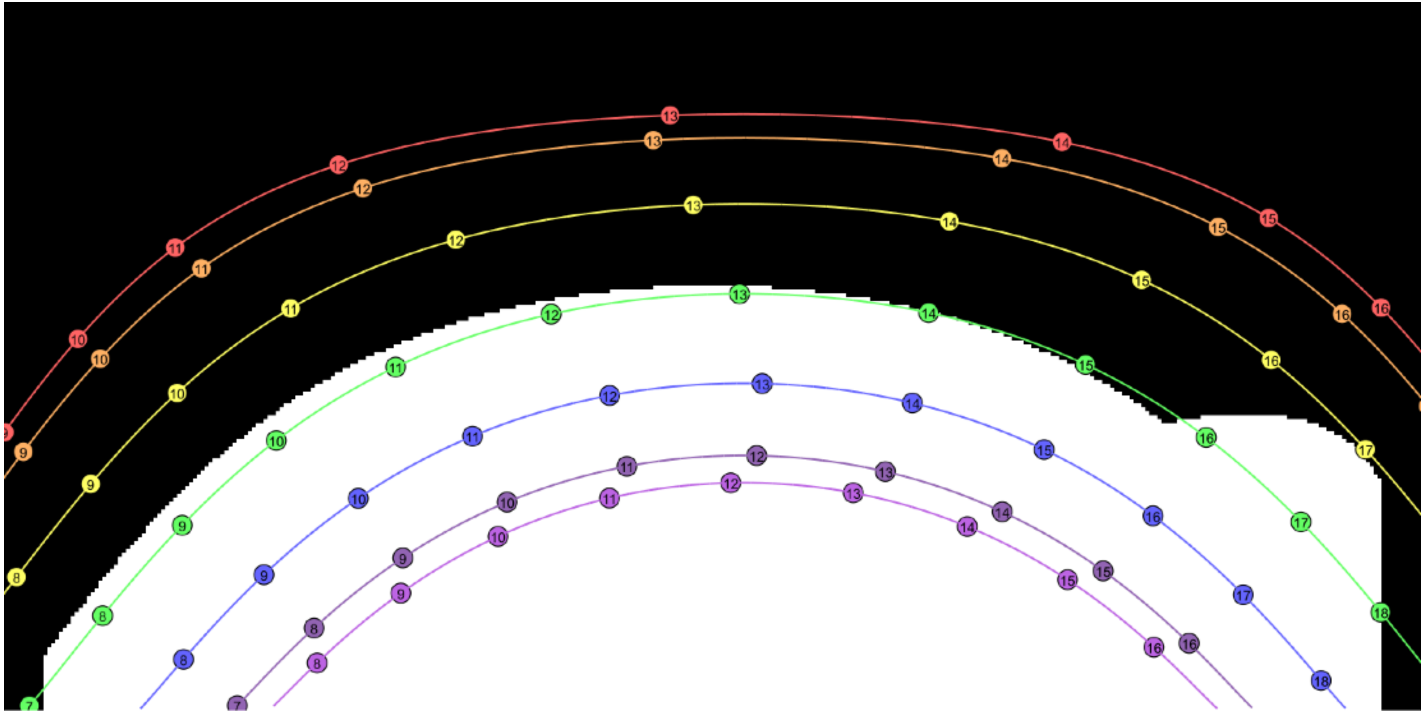
Trace Visible Sky



Now it’s an angular obstruction map!

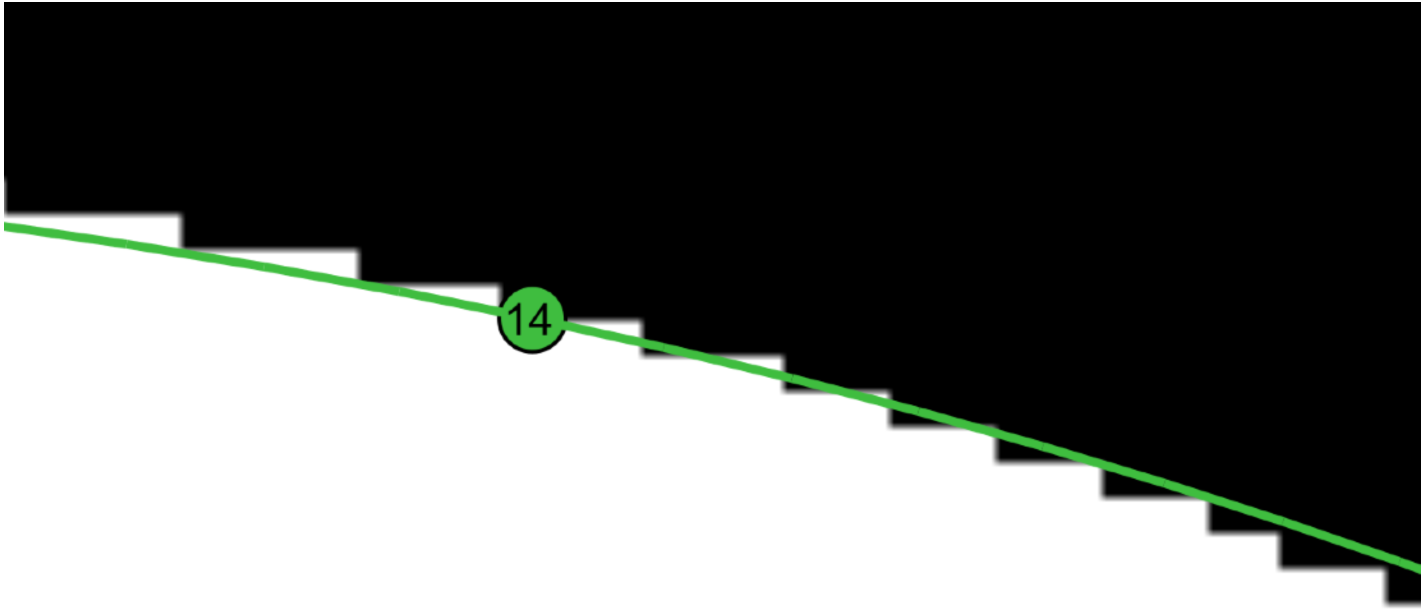


Straight lines are more than just pleasing…



Equirectangular

Straight lines are more than just pleasing…



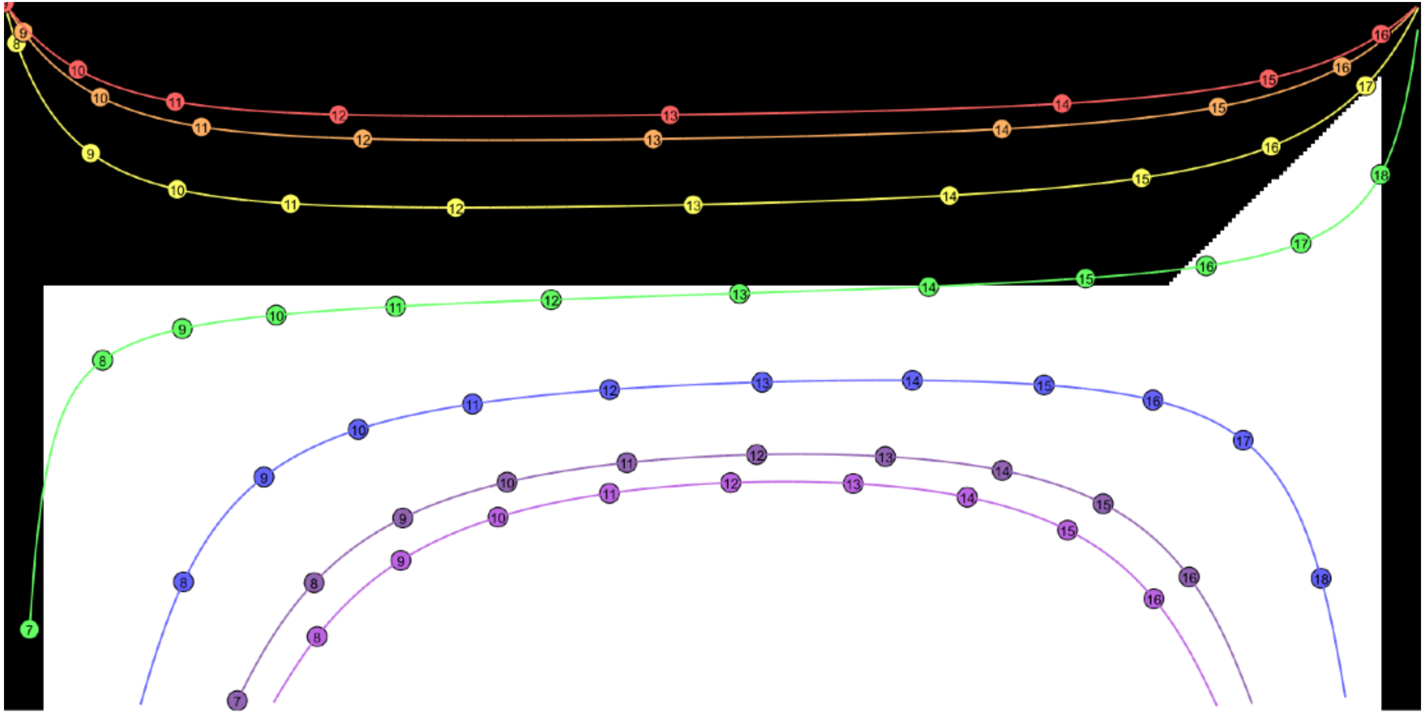
(count the number of crossings between black and white)

Equirectangular

HALIO

halioglass.com

Straight lines are more than just pleasing…

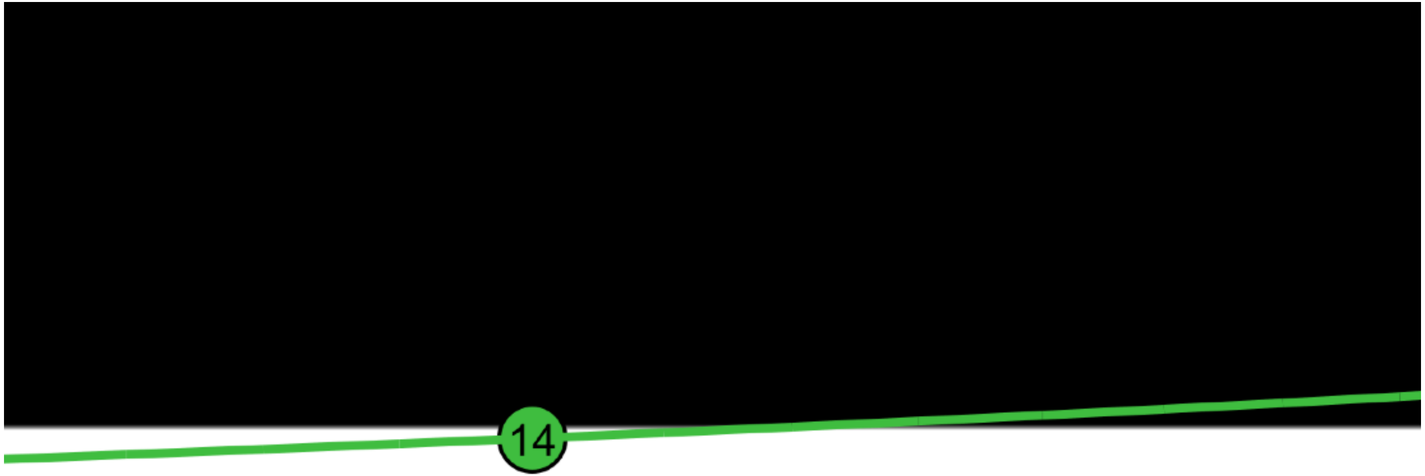


Orthonormal

HALIO

halioglass.com

Straight lines are more than just pleasing…



(count the number of crossings between black and white)

Orthonormal

Straight lines are more than pleasing…

- Straight lines are practical!
  - Sun path can cross a pixelated curve many times, requires filtering to prevent the window from cycling.
  - Sun path crosses a pixelated straight line once, less need for filtering.

  - Straight lines are easier to trace.

Geometric Method Example - Overhang

Geometry Parameter	Description
Overhang Depth	Distance from the façade to the
Overhang Height above Window	Distance from the top of the window to the bottom of the overhang
Window Height	Distance from the window sill to the window head
Window Width	Distance from the left window jamb to the right window jamb
Overhang Extension Left	Distance from the left edge of the window to the left edge of the overhang (looking from inside out)
Overhang Extension Right	Distance from the right edge of the window to the right edge of the overhang (looking from inside out)
Jamb Thickness	Distance from the inside edge of the window jamb to the outside edge of the window jamb

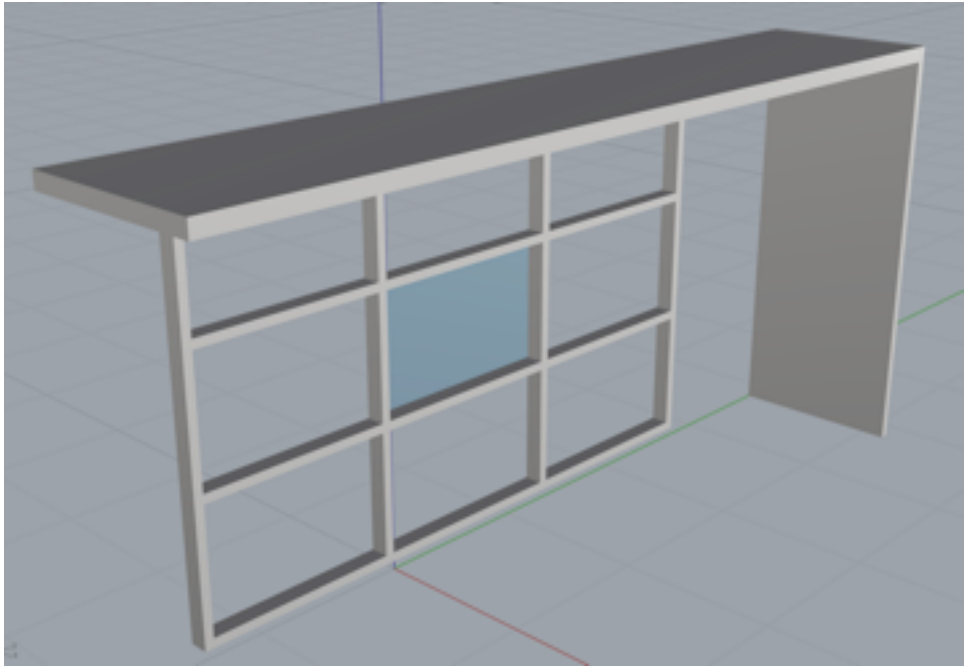
Geometric Method Example - Overhang

Geometry Parameters	Obstruction Map Image	Geometry Parameters	Obstruction Map Image
OD = 1.2 m OH = 0.1 m WH = 2.5 m WW = 1.0 m OEL = 1.5 m OER = 1.5 m JD = 0.1 m		OD = 1.2 m OH = 0.1 m **WH = 1.5 m** WW = 1.0 m OEL = 1.5 m OER = 1.5 m JD = 0.1 m	
OD = 2.0 m OH = 0.1 m WH = 2.5 m WW = 1.0 m OEL = 1.5 m OER = 1.5 m JD = 0.1 m		OD = 1.2 m OH = 0.1 m WH = 2.5 m WW = 1.0 m **OEL = 0.5 m** **OER = 12.0 m** JD = 0.1 m	

Geometric Method Example - Overhang

Overhang Depth	1.19
Overhang Height Above Window	0.97
Window Height	0.80
Window Width	1.32
Overhang Extension Left	15.0
Overhang Extension Right	1.82
Jamb Depth	0.12

Raytracing Method - Example



Script to generate ray samples

```
import math
import numpy

# Utility function to convert between angle and unit vector. Accepts azimuth and altitude angle and returns a unit
# vector +y is 0 degree orientation; +x is 90 degree orientation; +z is 90 degree altitude
def ang2vec(azi, alt):
    z=math.sin(math.radians(alt))
    y=math.cos(math.radians(alt))*math.cos(math.radians(azi))
    x=math.cos(math.radians(alt))*math.sin(math.radians(azi))
    return([x,y,z])

class Window:
    def __init__(self, orientation, inclination ):
        self.orientation = orientation
        self.inclination = inclination
        self.normal_xyz = ang2vec(self.orientation, 90-self.inclination)
        self.facade_up_xyz = ang2vec(self.orientation, 180-self.inclination)
        # Define translation to facade coordinate system, v=outward facing normal to facade,
        # w=up(projected into facade if sloped), u=along facade to the right facing out (perpendicular to v&w)
        v = self.normal_xyz
        w = self.facade_up_xyz
        u = ang2vec(self.orientation+90,0)
        # Basis conversion matrices between site (xyz) and facade (uvw) coordinates.
        self.xyz2uvw = numpy.array([u,v,w]).transpose()
        self.uvw2xyz = numpy.linalg.inv(self.xyz2uvw)
```

```
def WinAngle2GlobalVector(self, winAzimuth, winProfile):
    winvector_uvw = ang2vec(winAzimuth, winProfile)
    site_xyz = numpy.linalg.solve(self.uvw2xyz, numpy.array(winvector_uvw))
    return( site_xyz )

def generateRays(self, rayOrigin, rayOriginR = None):
    if len(rayOrigin) != 3 or ( rayOriginR != None and len(rayOriginR) != 3 ) :
        print('Ray Origin must be a list of length 3.')
        return -1
    if self.inclination <= 90:     # Half Map
        winAzimuthRange = list(numpy.arange(-89.75, 90, 0.5))
        winProRange = list(numpy.arange(89.75, 0, -0.5))
    else:                          # Full Map
        winAzimuthRange = list(numpy.arange(-89.75, 90, 0.5))
        winProRange = list(numpy.arange(89.75, -90, -0.5))
    for winProfile in winProRange:
        for winAzimuth in winAzimuthRange:
            if rayOriginR != None and winAzimuth > 0 : origin = rayOriginR
            else: origin = rayOrigin
            direction = self.WinAngle2GlobalVector(winAzimuth,winProfile)
            ray = '{}\t{}\t{}\t{}\t{}\t{}'.format(origin[0], origin[1], origin[2],
            direction[0], direction[1], direction[2])
            print (ray)
```

HALIO

There are slight differences between maps generated with different methods

Photographic Map	Ray Traced Map	Difference
		

The ray traced model didn’t include neighboring buildings.

There are slight differences between maps generated with different methods

Photographic	Geometric	Difference
		

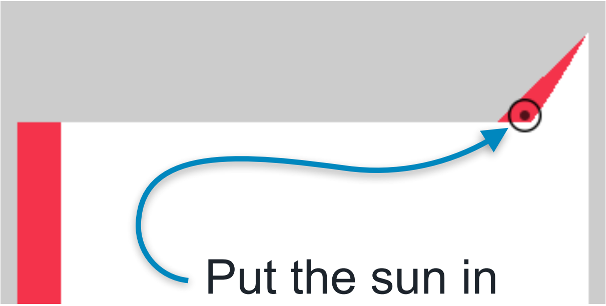
The geometric description didn't include neighboring buildings, or the perpendicular facade to the left (could have been added as a fin).

There are slight differences between maps generated with different methods

Ray Traced	Geometric	Difference
		

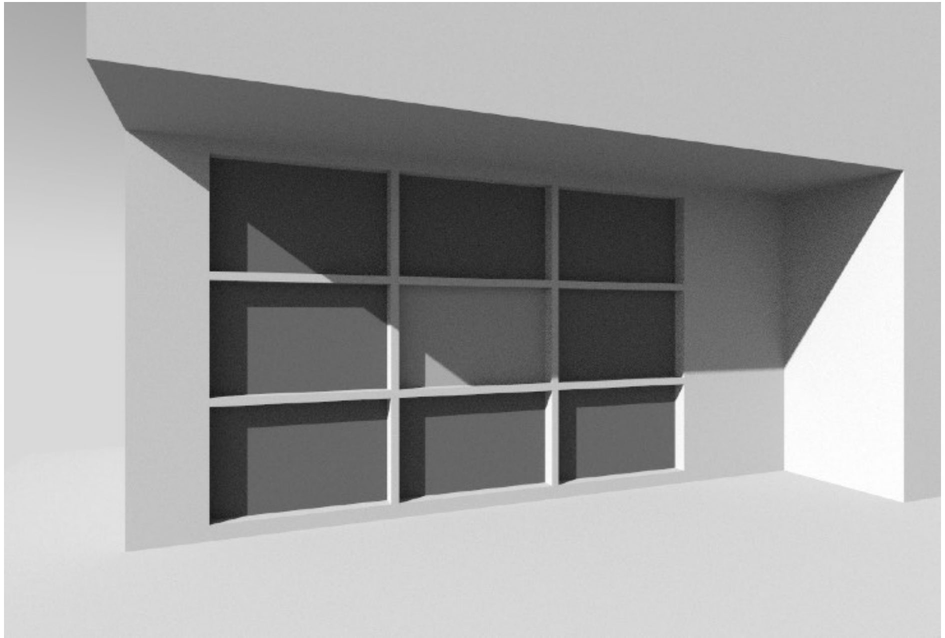
The geometric description didn't include the perpendicular facade to the left (could have been added as a fin).

The geometric method accounts for the whole window, while the retraced and photometric only account for the bottom corners.



Put the sun in this position

And render



Neither of the bottom corners are in the sun, but a small triangle of the window does receive sunshine.

When to use which methods

- Geometric is best for building attached geometry. It can be logically combined with photographic or ray-traced for site based obstructions.

- Photographic is suited for a small number of window
  - No need to generate/merge CAD model
  - Can handle any type of exterior obstruction (trees, billboards, etc.)
  - Manual or computer aided tracing becomes cumbersome with more windows

- Raytracing is suited for large numbers of windows
  - Model setup time is amortized over all windows, and with a large number becomes insignificant per window.

Bonus - Allowable Sun Map!

- Turn the camera (or ray generator) around and face into the space
- Outline areas where direct sun is allowed

HALIO
halioglass.com

Obstruction Map - Examples from NYC

- Hayward is great but…

Obstruction Map - Examples from NYC

- Hayward is great but…
- New York is better.

Obstruction Map - Examples from NYC

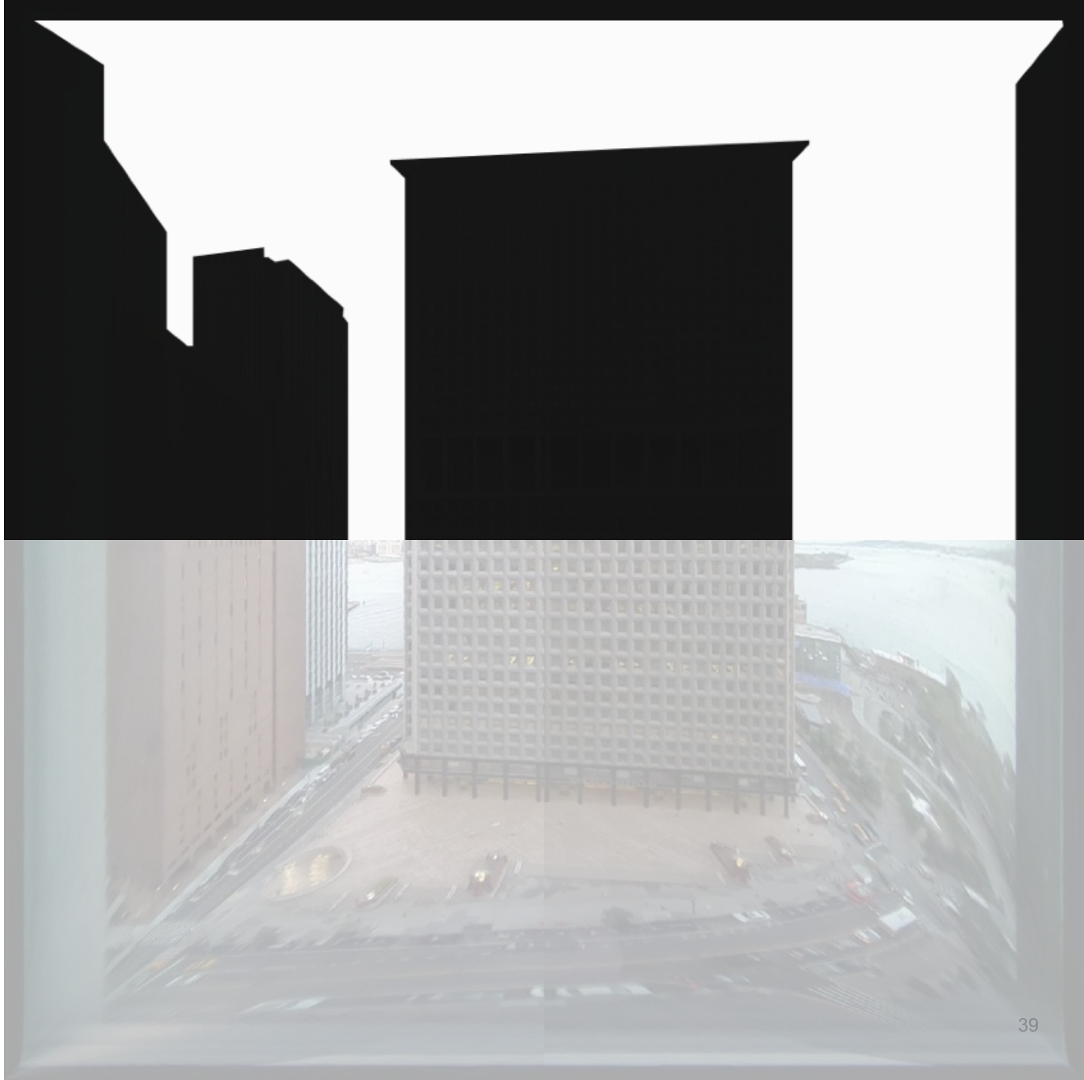
- Hayward is great but…
- New York is better.
- Actually, Hayward isn’t great.

My Hotel Room

My Hotel Room

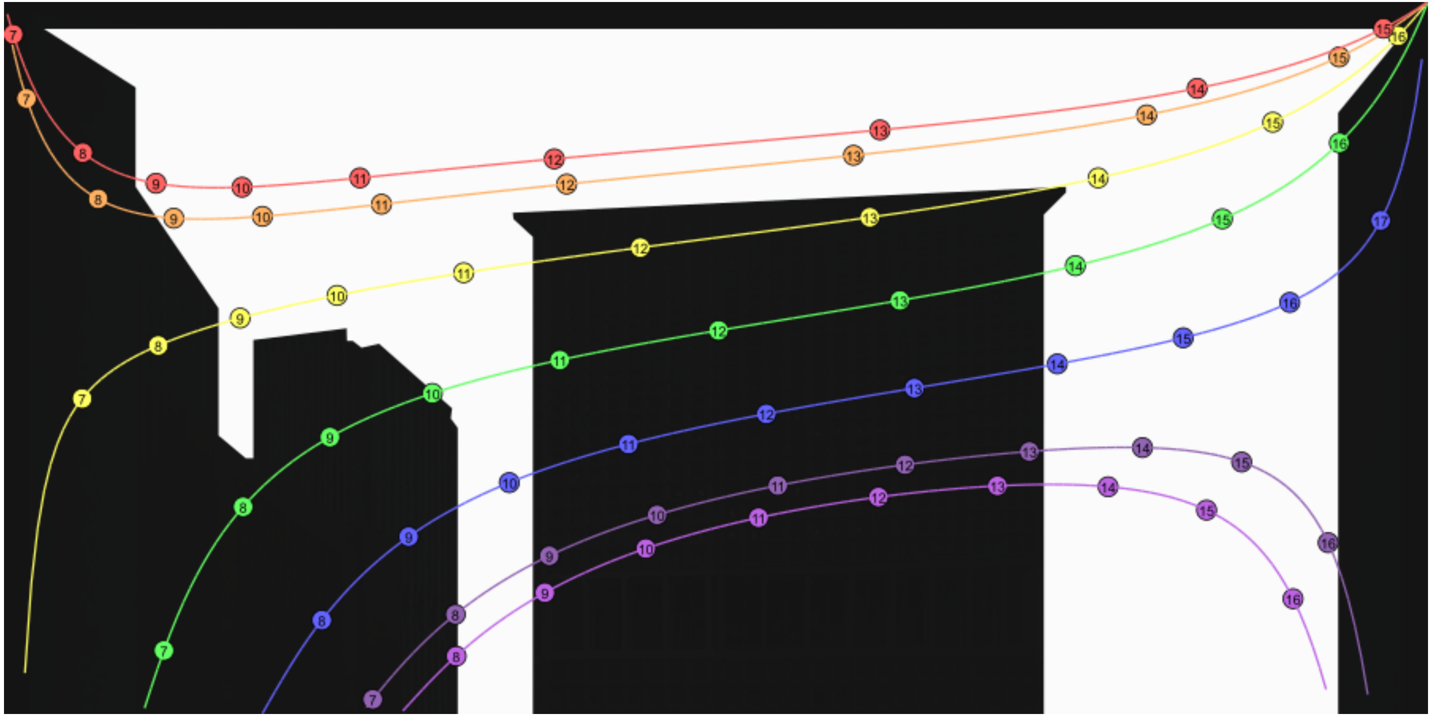
HALIO
halioglass.com

My Hotel Room



halioglass.com

My Hotel Room



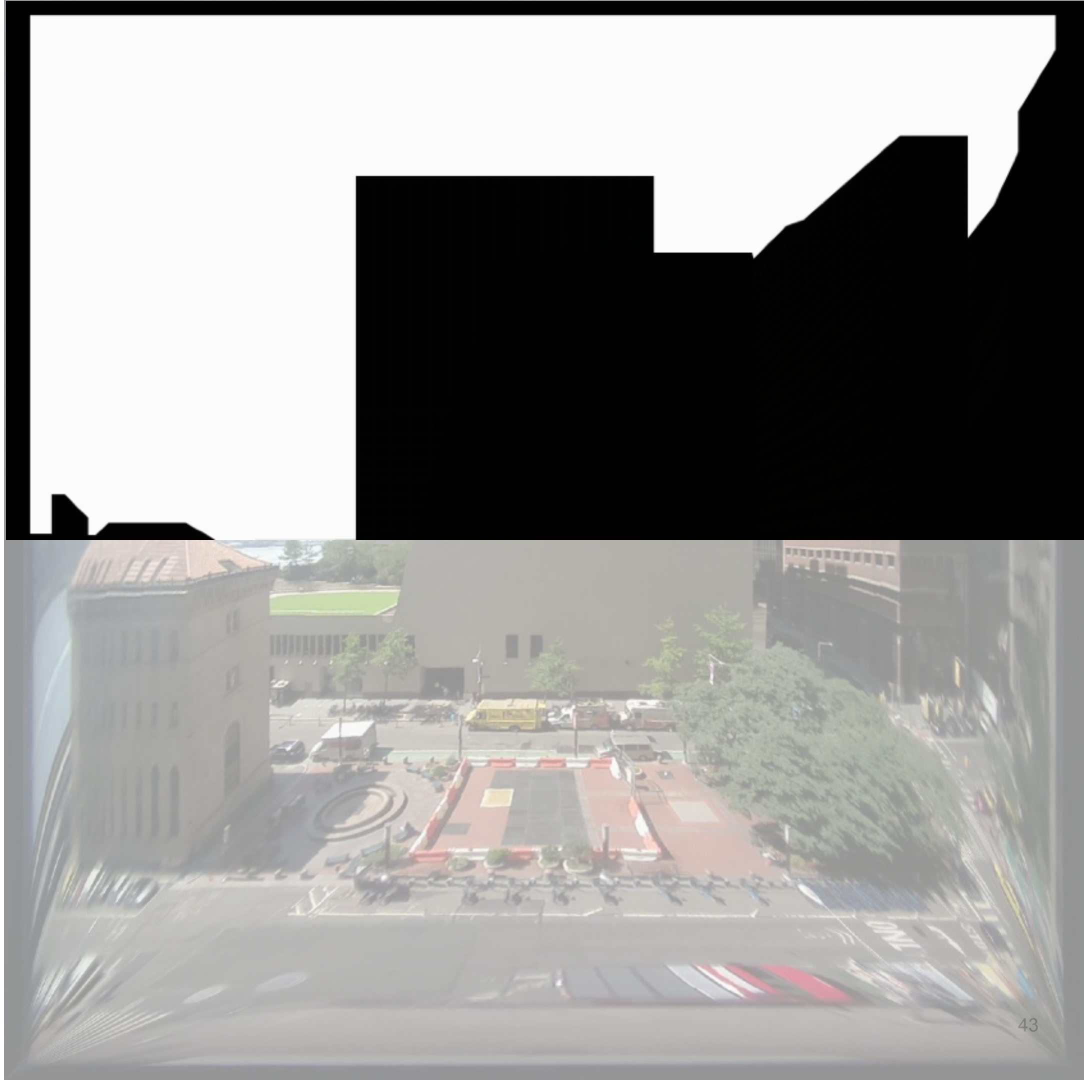
HALIO

halioglass.com

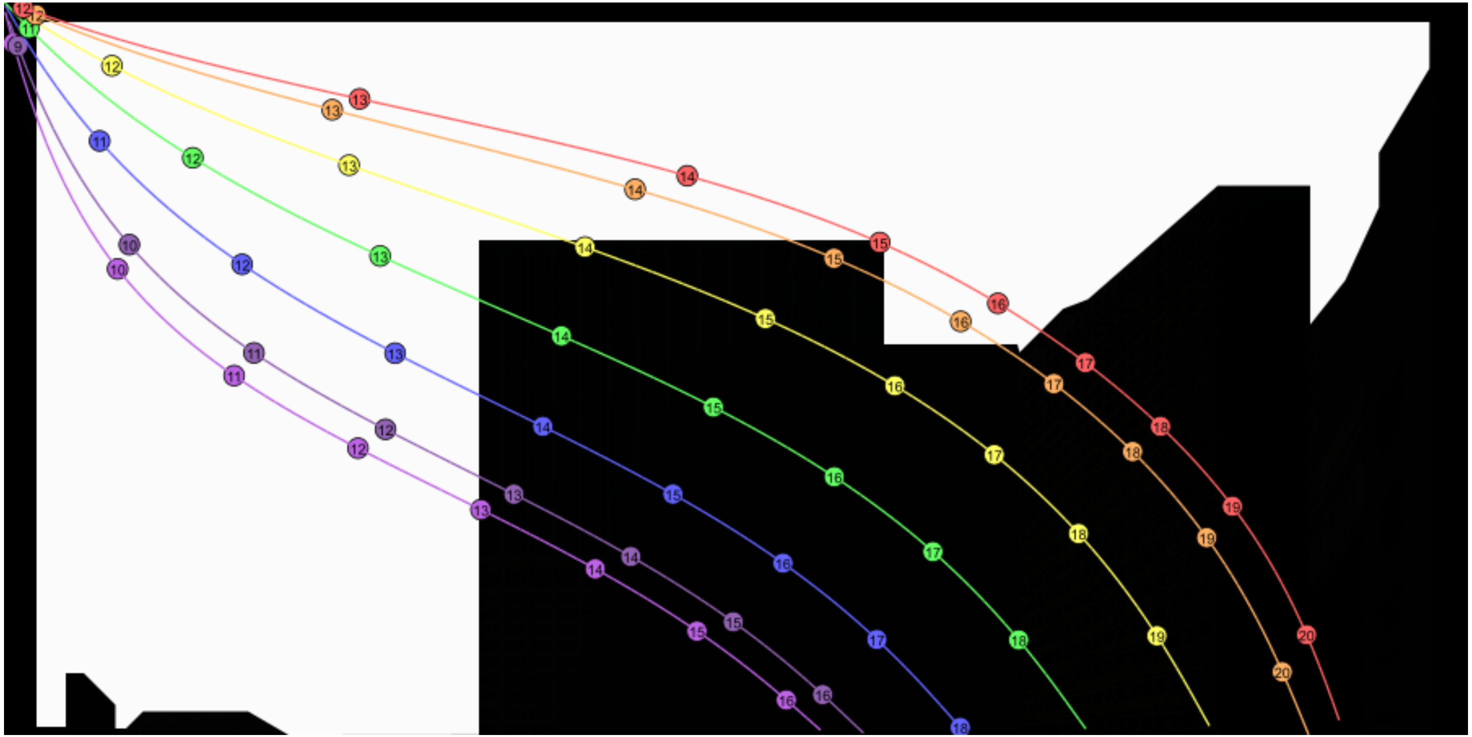
Arup’s Southwest Facade

Arup's Southwest Facade

Arup’s Southwest Facade



Arup’s Southwest Facade



Arup’s Southeast Facade

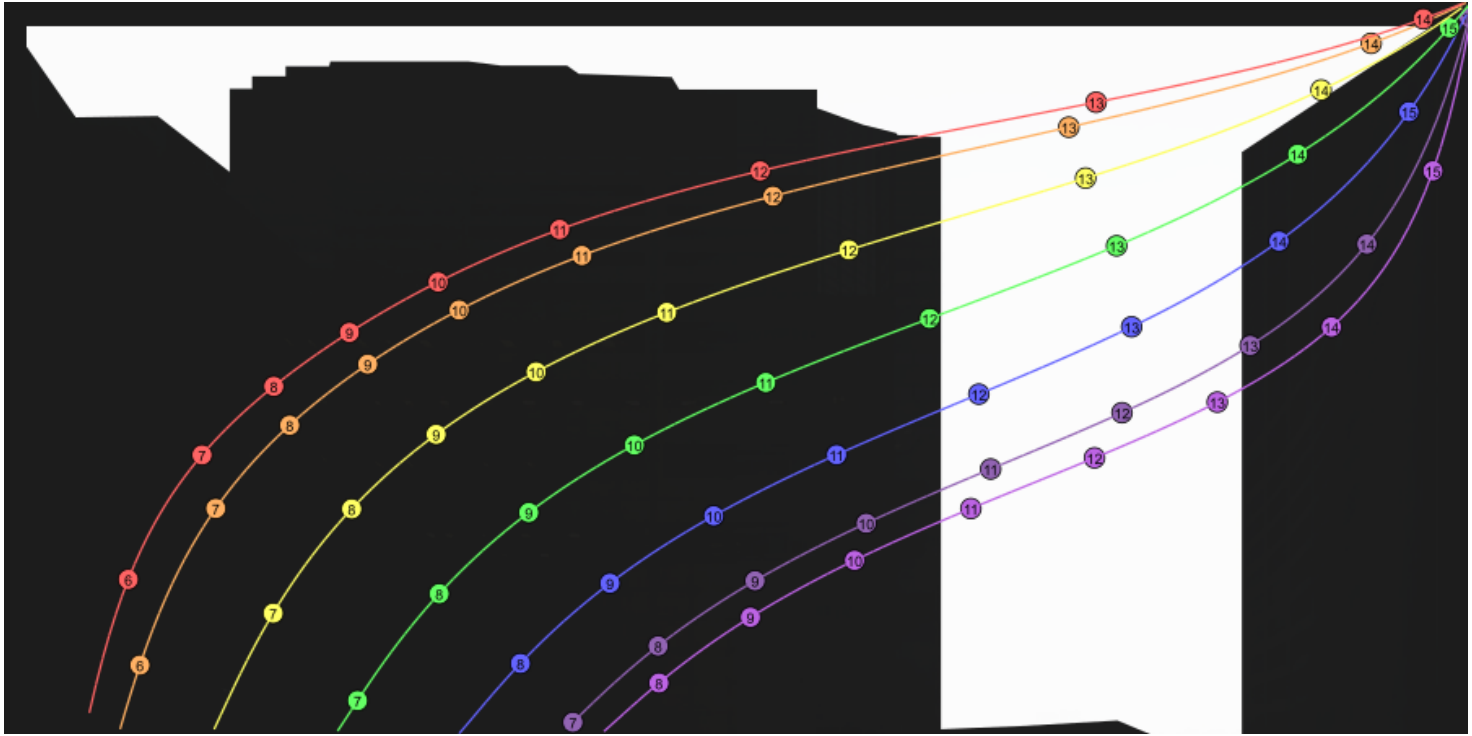
Arup's Southeast Facade

halioglass.com

Arup’s Southeast Facade

halioglass.com

Arup’s Southeast Facade

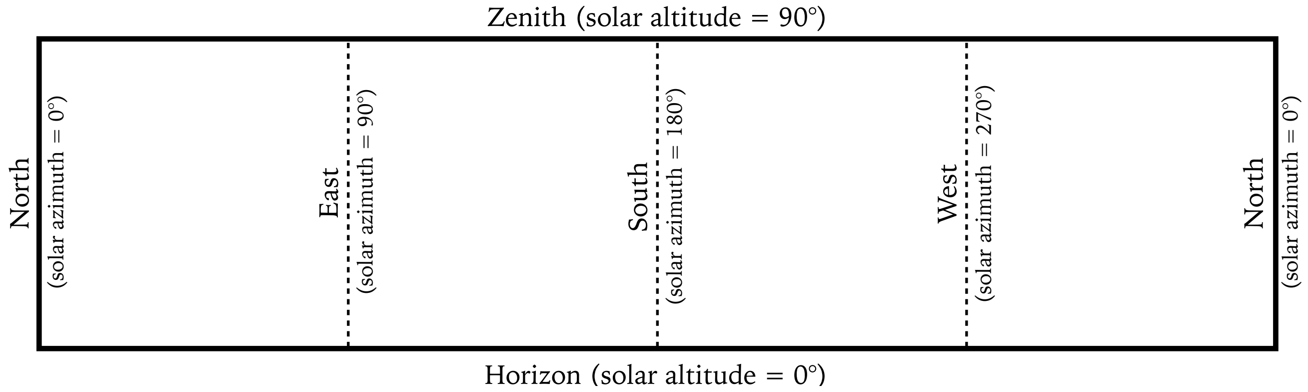


Reflected Sun Maps

Reflection map goals

- Only one method: raytracing
- Identify sun positions that cause reflected glare as viewed from the window.
  - Need 180° by 90° format - The sun can be anywhere in the sky and cause a reflection onto the window
  - Equirectangular
- Halio has continuous tint range - Need to know how much to tint for reflected sun
  - Surface reflectance
  - Incident angle on window
  - Position in field of view

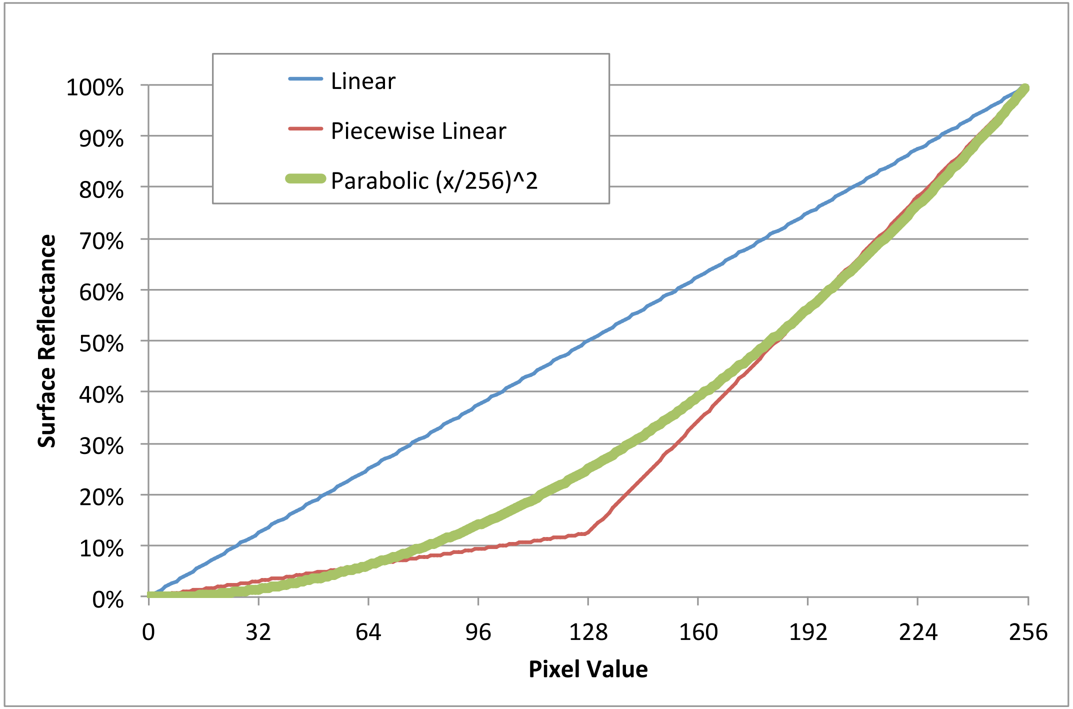
Reflection Map Format - Equirectangular



Reflection map format - Four Channels

- Channel 1 (R): surface reflectance
  - Parabolic encoding: reflectance= $(R/256)^2$
  - Includes angular effects included in model
- Channel 2 (G): elevation angle of the reflection
  - Linear encoding from -90° (G=0) to 90° (G=255)
- Channel 3 (B): reflection incident angle on window
  - Linear encoding from 0° (perpendicular) to 90° (glancing)
- Channel 4 (A): Does this sun position cause reflection to window?
  - Boolean value:
    True (255) causes reflection
    False (0) does not cause reflection

Channel 1 (reflectance) encoding



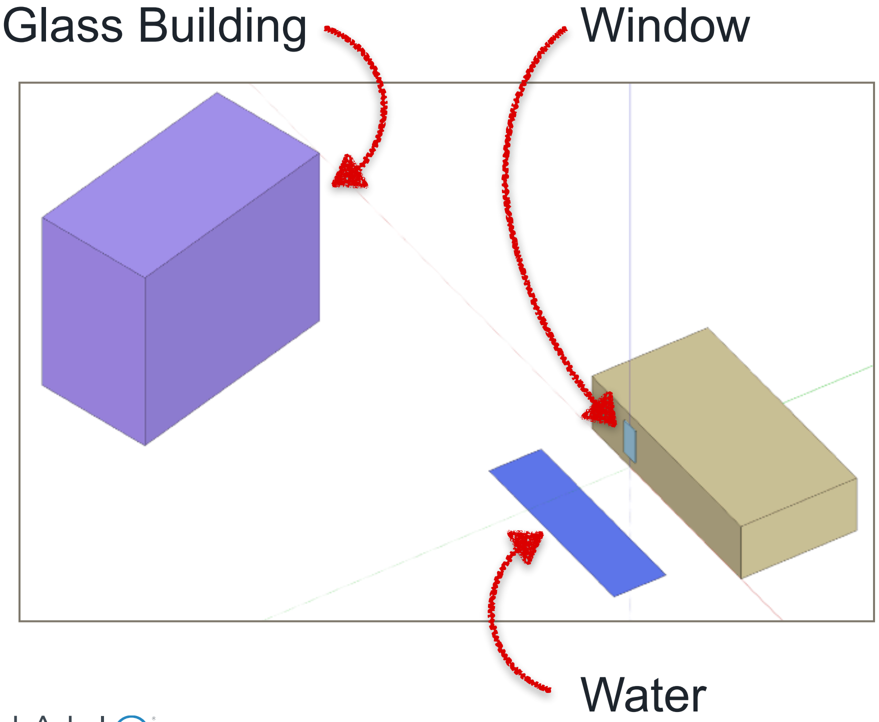
Increments:

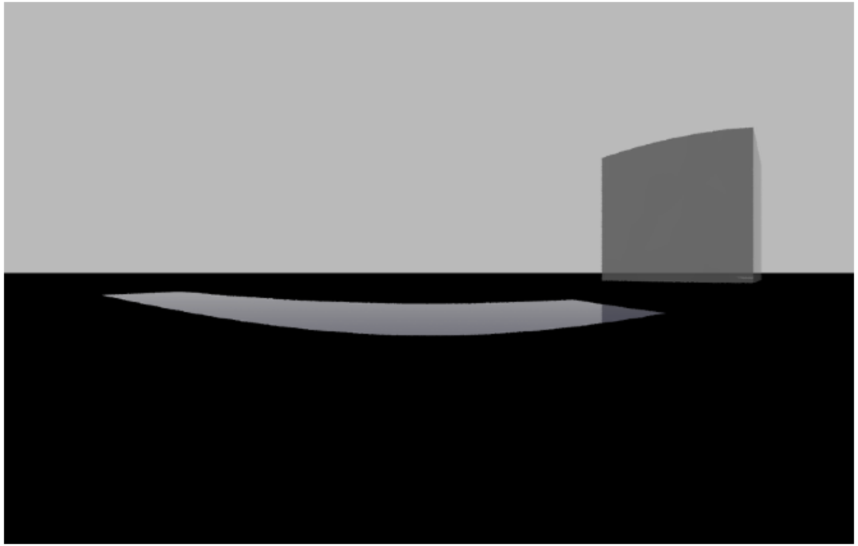
Linear: 0.4%

Piecewise:
<12%: 0.1% increment
>12%: 0.7% increment

Parabolic:
@1%: 0.08% increment
@10%: 0.2% increment
@90%: 0.7% increment

Test Scene





Rendering from window

Trace ray from window out to scene

- Generate sample rays with something like this:

```
res = 0.25

for alt in frange(-90 + res/2, 90, res):
        for in frange(-90 + res/2, 90, res):
                direction = angle2vector(alt,azi)
                sample.append([*window_origin,*direction])
```

HALIO

halioglass.com

Trace rays like this

rtrace -ab 0 -st 0 -lr 1 -h -otwdv model.oct

Parameter	**Name**	**Description**
-ab 0	ambient bounces	turns off diffuse reflections
-st 0	specular threshold	a threshold of zero ensures that all specular reflections are traced.
-lr 1	limit reflections	limits specular reflections to a single bounce (we're not controlling for secondary or higher order reflections in our reflection map)
-h	header	turns off the header in the output
-otwdv	output specification	outputs the following: t - whole ray tree w - ray weight d - ray direction v - ray value

Output

red = sample rays
blue = child rays

1.000000e+00	6.221329e-01	-7.381519e-01	-2.609260e-01	0.000000e+00	0.000000e+00	0.000000e+00	
1.000000e+00	6.189058e-01	-7.408598e-01	-2.609259e-01	0.000000e+00	0.000000e+00	0.000000e+00	
1.000000e+00	6.156672e-01	-7.435533e-01	-2.609261e-01	0.000000e+00	0.000000e+00	0.000000e+00	
1.000000e+00	6.124173e-01	-7.462323e-01	-2.609261e-01	0.000000e+00	0.000000e+00	0.000000e+00	
1.000000e+00	6.091549e-01	-7.488978e-01	-2.609259e-01	0.000000e+00	0.000000e+00	0.000000e+00	
1.000000e+00	6.058821e-01	-7.515481e-01	-2.609260e-01	0.000000e+00	0.000000e+00	0.000000e+00	
1.000000e+00	6.025968e-01	-7.541848e-01	-2.609259e-01	0.000000e+00	0.000000e+00	0.000000e+00	
	2.615670e-01	5.993001e-01	-7.568071e-01	2.609260e-01	1.000000e+00	1.000000e+00	1.000000e+00
1.000000e+00	5.993001e-01	-7.568071e-01	-2.609260e-01	2.615670e-01	2.615670e-01	2.615670e-01	
	2.615670e-01	5.959920e-01	-7.594150e-01	2.609260e-01	1.000000e+00	1.000000e+00	1.000000e+00
1.000000e+00	5.959920e-01	-7.594150e-01	-2.609260e-01	2.615670e-01	2.615670e-01	2.615670e-01	
	2.615670e-01	5.926730e-01	-7.620081e-01	2.609260e-01	1.000000e+00	1.000000e+00	1.000000e+00
1.000000e+00	5.926730e-01	-7.620081e-01	-2.609260e-01	2.615670e-01	2.615670e-01	2.615670e-01	
	2.615671e-01	5.893428e-01	-7.645867e-01	2.609259e-01	1.000000e+00	1.000000e+00	1.000000e+00
1.000000e+00	5.893428e-01	-7.645867e-01	-2.609259e-01	2.615671e-01	2.615671e-01	2.615671e-01	
	2.615670e-01	5.860009e-01	-7.671509e-01	2.609260e-01	1.000000e+00	1.000000e+00	1.000000e+00

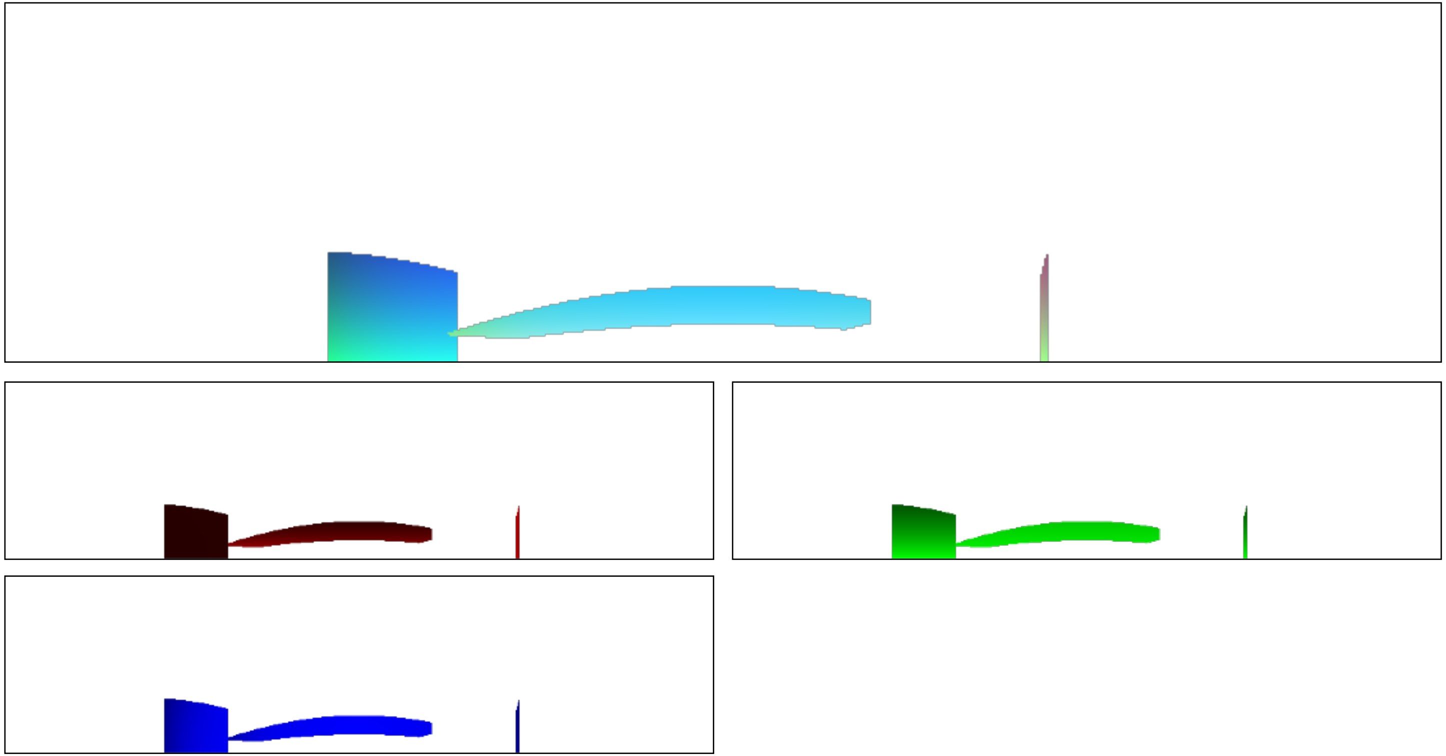
No Reflection

Reflection

Reflectance

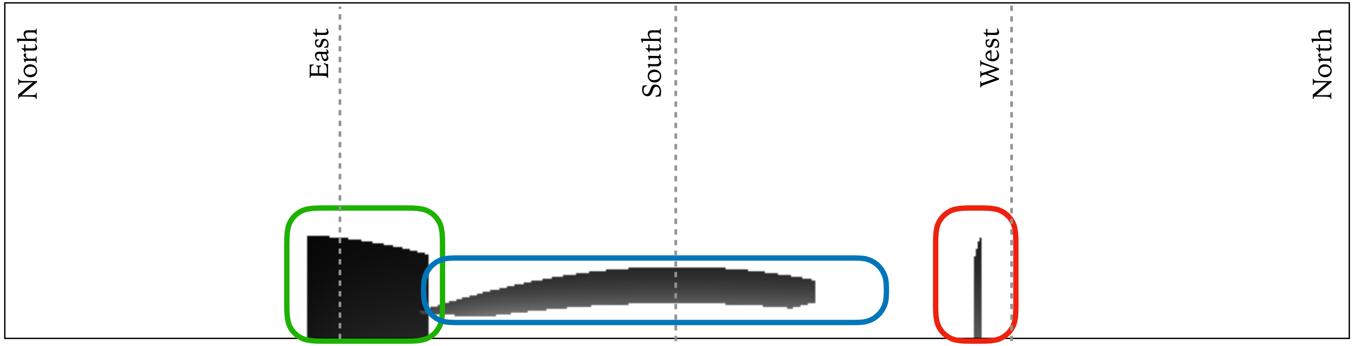
Direction of reflected ray

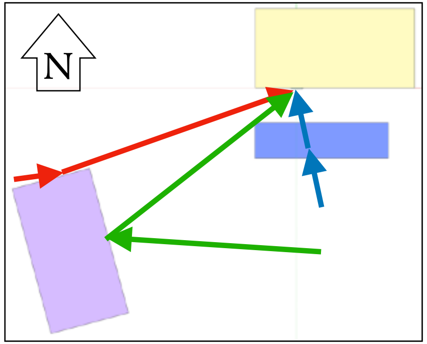
Reflection Map



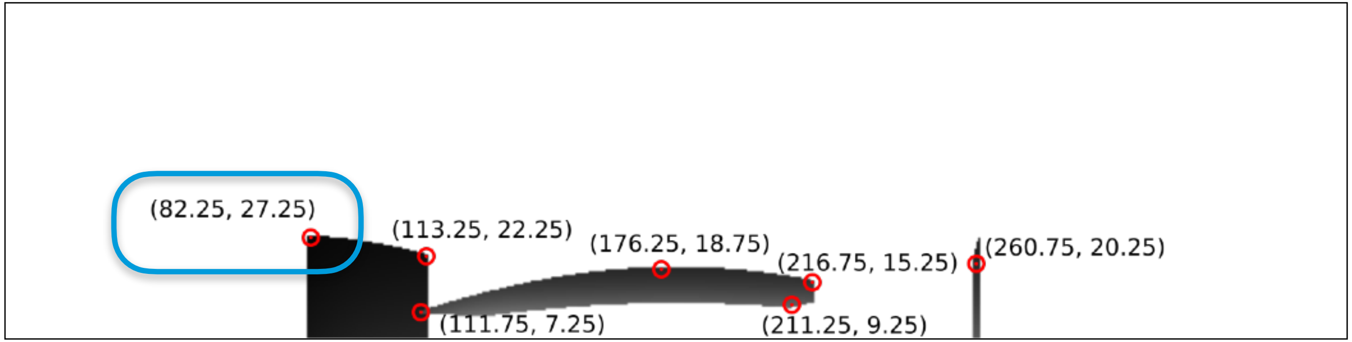
halioglass.com

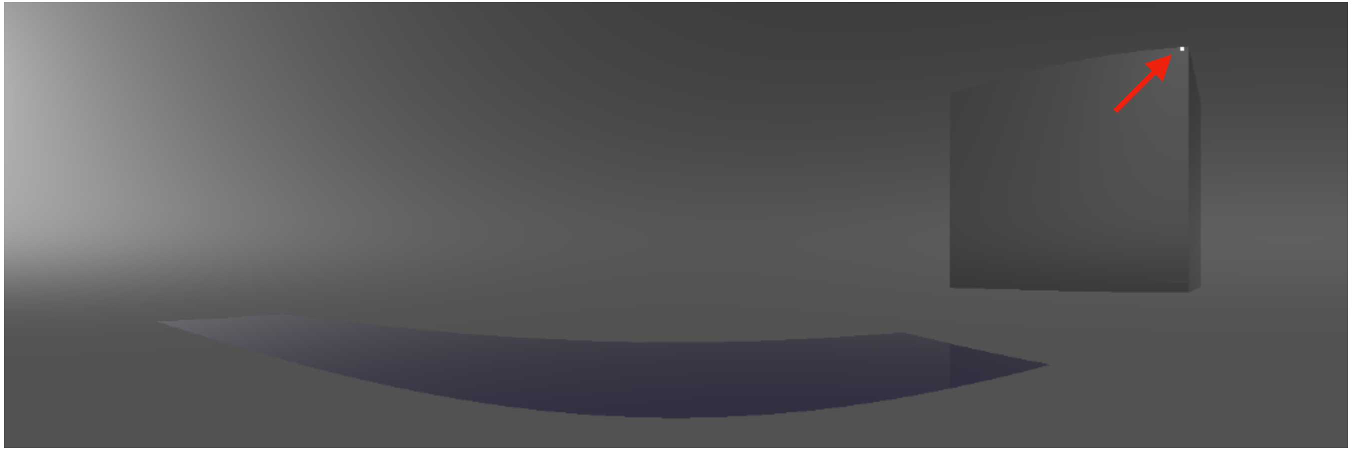
Understanding the map



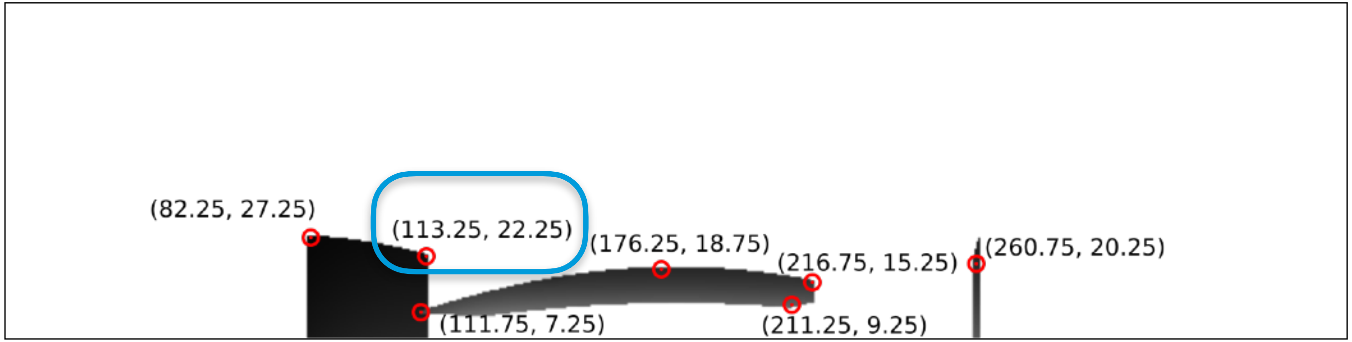


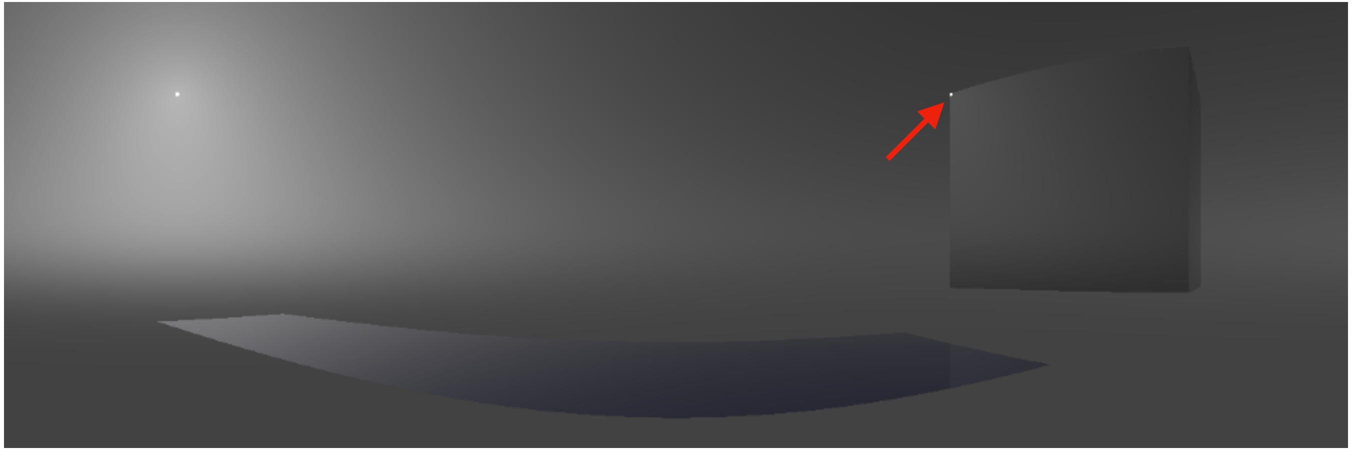
Verifying the map



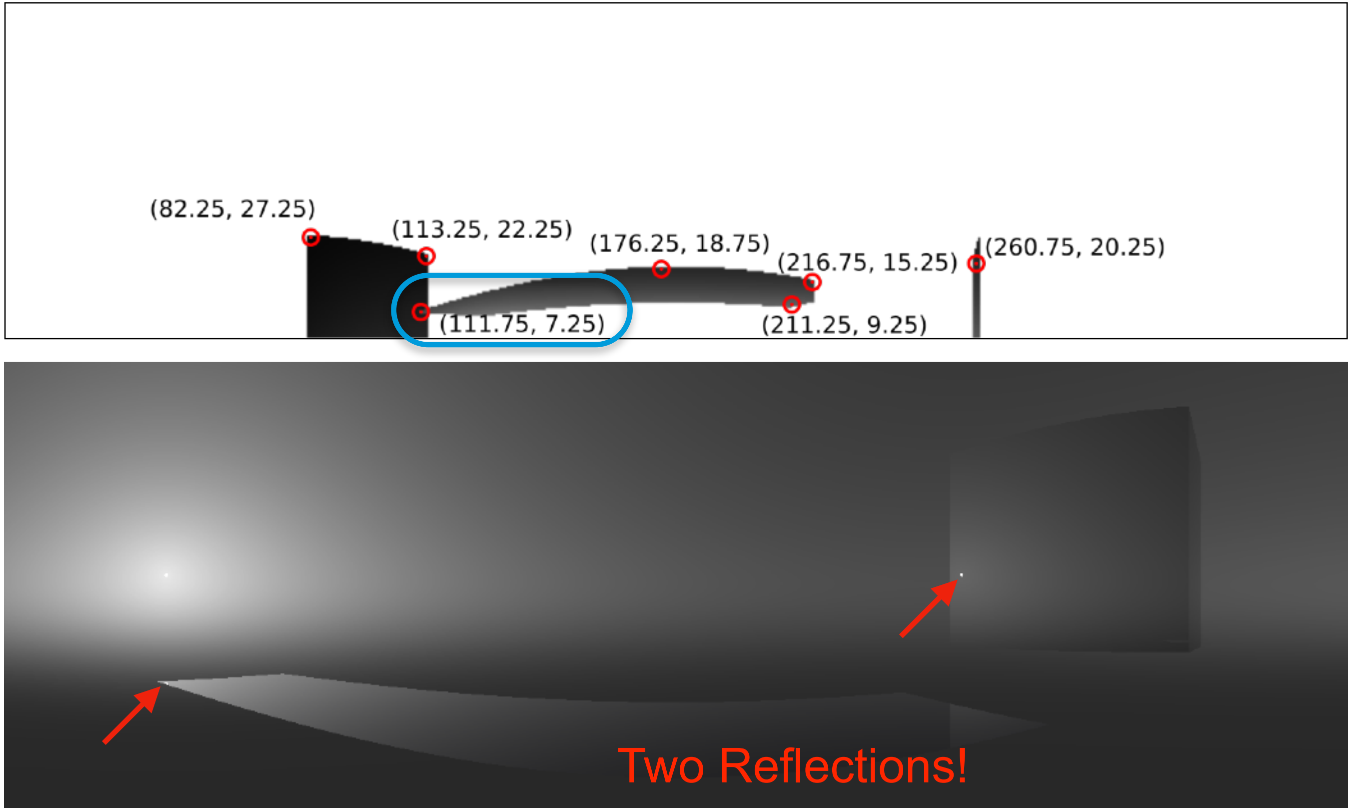


Verifying the map



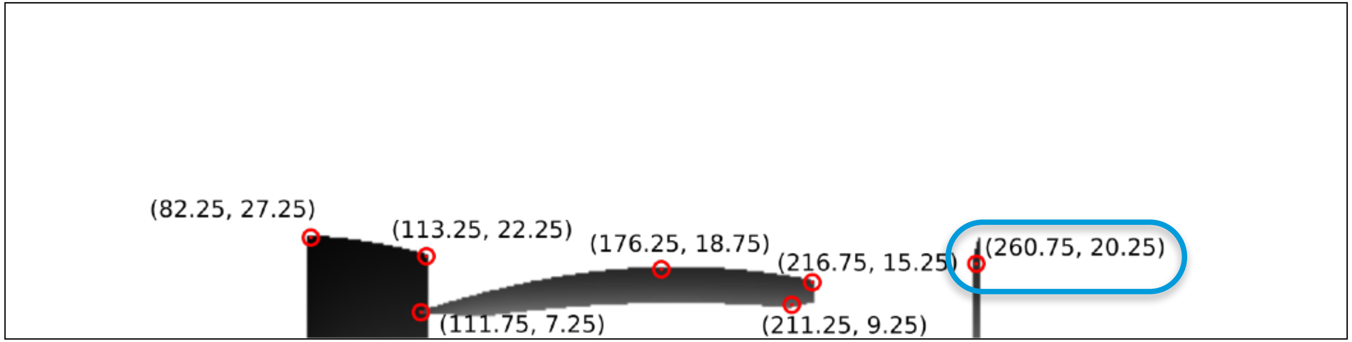


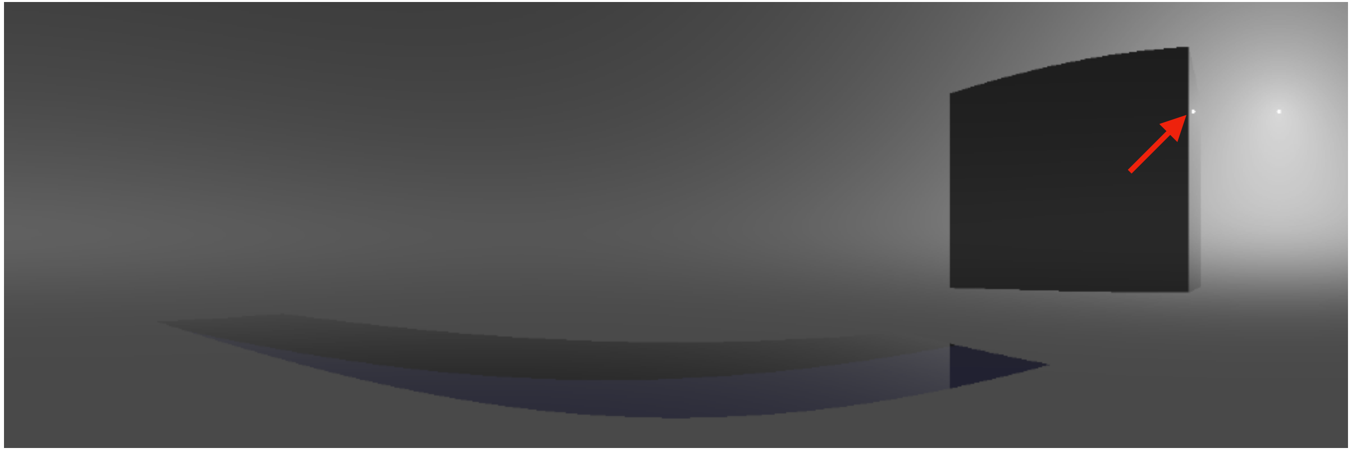
Verifying the map



halioglass.com

Verifying the map





HALIO®

halioglass.com