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 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
Photographic method

Prototype Camera
Accelerometer - Correct for Crooked Hand
Angular distortion correction

\[ y = -6\times10^{-7}x^4 + 7\times10^{-5}x^3 - 0.0013x^2 + 0.9716x \]
Correct for angular distortion
Correct for angular distortion. It's subtly but important.
Correct for angular distortion
It’s subtly but important
Reproject from angular to orthonormal
Re-project from angular to orthonormal
Re-project

Straight Line = Pleasing!

Orthonormal
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
Straight lines are more than just pleasing…

Orthonormal
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

<table>
<thead>
<tr>
<th>Geometry Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhang Depth</td>
<td>Distance from the façade to the</td>
</tr>
<tr>
<td>Overhang Height above Window</td>
<td>Distance from the top of the window to the bottom of the overhang</td>
</tr>
<tr>
<td>Window Height</td>
<td>Distance from the window sill to the window head</td>
</tr>
<tr>
<td>Window Width</td>
<td>Distance from the left window jamb to the right window jamb</td>
</tr>
<tr>
<td>Overhang Extension Left</td>
<td>Distance from the left edge of the window to the left edge of the overhang (looking from inside out)</td>
</tr>
<tr>
<td>Overhang Extension Right</td>
<td>Distance from the right edge of the window to the right edge of the overhang (looking from inside out)</td>
</tr>
<tr>
<td>Jamb Thickness</td>
<td>Distance from the inside edge of the window jamb to the outside edge of the window jamb</td>
</tr>
</tbody>
</table>
# Geometric Method Example - Overhang

<table>
<thead>
<tr>
<th>Geometry Parameters</th>
<th>Obstruction Map Image</th>
<th>Geometry Parameters</th>
<th>Obstruction Map Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>OD = 1.2 m</td>
<td><img src="image1.png" alt="Image" /></td>
<td>OD = 1.2 m</td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
<tr>
<td>OH = 0.1 m</td>
<td><img src="image3.png" alt="Image" /></td>
<td>OH = 0.1 m</td>
<td><img src="image4.png" alt="Image" /></td>
</tr>
<tr>
<td>WH = 2.5 m</td>
<td><img src="image5.png" alt="Image" /></td>
<td>WH = 1.5 m</td>
<td><img src="image6.png" alt="Image" /></td>
</tr>
<tr>
<td>WW = 1.0 m</td>
<td><img src="image7.png" alt="Image" /></td>
<td>WW = 1.0 m</td>
<td><img src="image8.png" alt="Image" /></td>
</tr>
<tr>
<td>OEL = 1.5 m</td>
<td><img src="image9.png" alt="Image" /></td>
<td>OEL = 1.5 m</td>
<td><img src="image10.png" alt="Image" /></td>
</tr>
<tr>
<td>OER = 1.5 m</td>
<td><img src="image11.png" alt="Image" /></td>
<td>OER = 1.5 m</td>
<td><img src="image12.png" alt="Image" /></td>
</tr>
<tr>
<td>JD = 0.1 m</td>
<td><img src="image13.png" alt="Image" /></td>
<td>JD = 0.1 m</td>
<td><img src="image14.png" alt="Image" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Geometry Parameters</th>
<th>Obstruction Map Image</th>
<th>Geometry Parameters</th>
<th>Obstruction Map Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>OD = 2.0 m</td>
<td><img src="image15.png" alt="Image" /></td>
<td>OD = 1.2 m</td>
<td><img src="image16.png" alt="Image" /></td>
</tr>
<tr>
<td>OH = 0.1 m</td>
<td><img src="image17.png" alt="Image" /></td>
<td>OH = 0.1 m</td>
<td><img src="image18.png" alt="Image" /></td>
</tr>
<tr>
<td>WH = 2.5 m</td>
<td><img src="image19.png" alt="Image" /></td>
<td>WH = 2.5 m</td>
<td><img src="image20.png" alt="Image" /></td>
</tr>
<tr>
<td>WW = 1.0 m</td>
<td><img src="image21.png" alt="Image" /></td>
<td>WW = 1.0 m</td>
<td><img src="image22.png" alt="Image" /></td>
</tr>
<tr>
<td>OEL = 1.5 m</td>
<td><img src="image23.png" alt="Image" /></td>
<td>OEL = 0.5 m</td>
<td><img src="image24.png" alt="Image" /></td>
</tr>
<tr>
<td>OER = 1.5 m</td>
<td><img src="image25.png" alt="Image" /></td>
<td>OER = 12.0 m</td>
<td><img src="image26.png" alt="Image" /></td>
</tr>
<tr>
<td>JD = 0.1 m</td>
<td><img src="image27.png" alt="Image" /></td>
<td>JD = 0.1 m</td>
<td><img src="image28.png" alt="Image" /></td>
</tr>
</tbody>
</table>
Geometric Method Example - Overhang

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhang Depth</td>
<td>1.19</td>
</tr>
<tr>
<td>Overhang Height Above Window</td>
<td>0.97</td>
</tr>
<tr>
<td>Window Height</td>
<td>0.80</td>
</tr>
<tr>
<td>Window Width</td>
<td>1.32</td>
</tr>
<tr>
<td>Overhang Extension Left</td>
<td>15.0</td>
</tr>
<tr>
<td>Overhang Extension Right</td>
<td>1.82</td>
</tr>
<tr>
<td>Jamb Depth</td>
<td>0.12</td>
</tr>
</tbody>
</table>
Raytracing Method - Example
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 = '{:.1f}'.format(origin[0]), '{:.1f}'.format(origin[1]), '{:.1f}'.format(origin[2]), '{:.1f}'.format(direction[0]), '{:.1f}'.format(direction[1]), '{:.1f}'.format(direction[2])
            print(ray)
There are slight differences between maps generated with different methods

<table>
<thead>
<tr>
<th>Photographic Map</th>
<th>Ray Traced Map</th>
<th>Difference</th>
</tr>
</thead>
</table>

The ray traced model didn’t include neighboring buildings.
There are slight differences between maps generated with different methods

<table>
<thead>
<tr>
<th>Photographic</th>
<th>Geometric</th>
<th>Difference</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>Ray Traced</th>
<th>Geometric</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Ray Traced Image" /></td>
<td><img src="image2" alt="Geometric Image" /></td>
<td><img src="image3" alt="Difference Image" /></td>
</tr>
</tbody>
</table>

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 windows
  - 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
Obstruction Map - Examples from NYC

- Hayward is great but…
Obstruction Map - Examples from NYC

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

New York is better.

Actually, Hayward isn’t great.
My Hotel Room
My Hotel Room
My Hotel Room
My Hotel Room
Arup’s Southwest Facade
Arup’s Southwest Facade
Arup’s Southwest Facade
Arup’s Southwest Facade
Arup’s Southeast Facade
Arup’s Southeast Facade
Arup’s Southeast Facade
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

Glass Building

Window

Water

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

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

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

<table>
<thead>
<tr>
<th>Reflectance</th>
<th>Direction of reflected ray</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000000e+00</td>
<td>6.221329e-01 -7.381519e-01 -2.609260e-01 0.000000e+00 0.000000e+00 0.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>6.189058e-01 -7.408598e-01 -2.609259e-01 0.000000e+00 0.000000e+00 0.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>6.156672e-01 -7.435533e-01 -2.609261e-01 0.000000e+00 0.000000e+00 0.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>6.124173e-01 -7.462323e-01 -2.609261e-01 0.000000e+00 0.000000e+00 0.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>6.091549e-01 -7.489897e-01 -2.609259e-01 0.000000e+00 0.000000e+00 0.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>6.058821e-01 -7.515481e-01 -2.609260e-01 0.000000e+00 0.000000e+00 0.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>6.025968e-01 -7.541848e-01 -2.609259e-01 0.000000e+00 0.000000e+00 0.000000e+00</td>
</tr>
<tr>
<td>2.615670e-01</td>
<td>5.993001e-01 -7.588071e-01 -2.609260e-01 1.000000e+00 1.000000e+00 1.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>5.993001e-01 -7.588071e-01 -2.609260e-01 2.615670e-01 2.615670e-01 2.615670e-01</td>
</tr>
<tr>
<td>2.615670e-01</td>
<td>5.959920e-01 -7.594150e-01 -2.609260e-01 1.000000e+00 1.000000e+00 1.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>5.959920e-01 -7.594150e-01 -2.609260e-01 2.615670e-01 2.615670e-01 2.615670e-01</td>
</tr>
<tr>
<td>2.615670e-01</td>
<td>5.926730e-01 -7.620081e-01 -2.609260e-01 1.000000e+00 1.000000e+00 1.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>5.926730e-01 -7.620081e-01 -2.609260e-01 2.615670e-01 2.615670e-01 2.615670e-01</td>
</tr>
<tr>
<td>2.615671e-01</td>
<td>5.893428e-01 -7.645867e-01 -2.609259e-01 1.000000e+00 1.000000e+00 1.000000e+00</td>
</tr>
<tr>
<td>1.000000e+00</td>
<td>5.893428e-01 -7.645867e-01 -2.609259e-01 2.615671e-01 2.615671e-01 2.615671e-01</td>
</tr>
</tbody>
</table>

**red** = sample rays  
**blue** = child rays
Reflection Map
Understanding the map
Verifying the map
Verifying the map
Verifying the map

Two Reflections!
Verifying the map