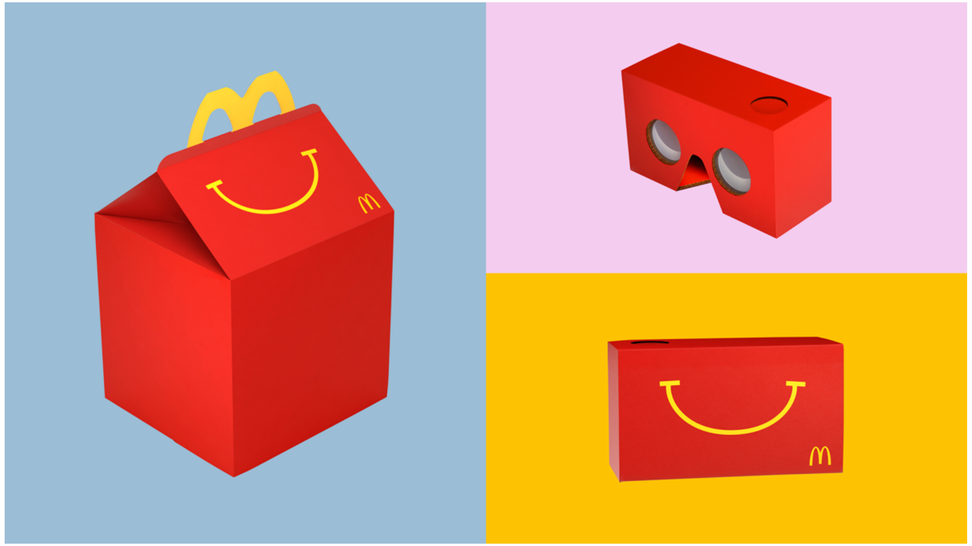
# Omni-directional Stereo Renderings and More

**Andy McNeil**

# Omnidirectional Stereo (ODS) is an image/video format for VR.



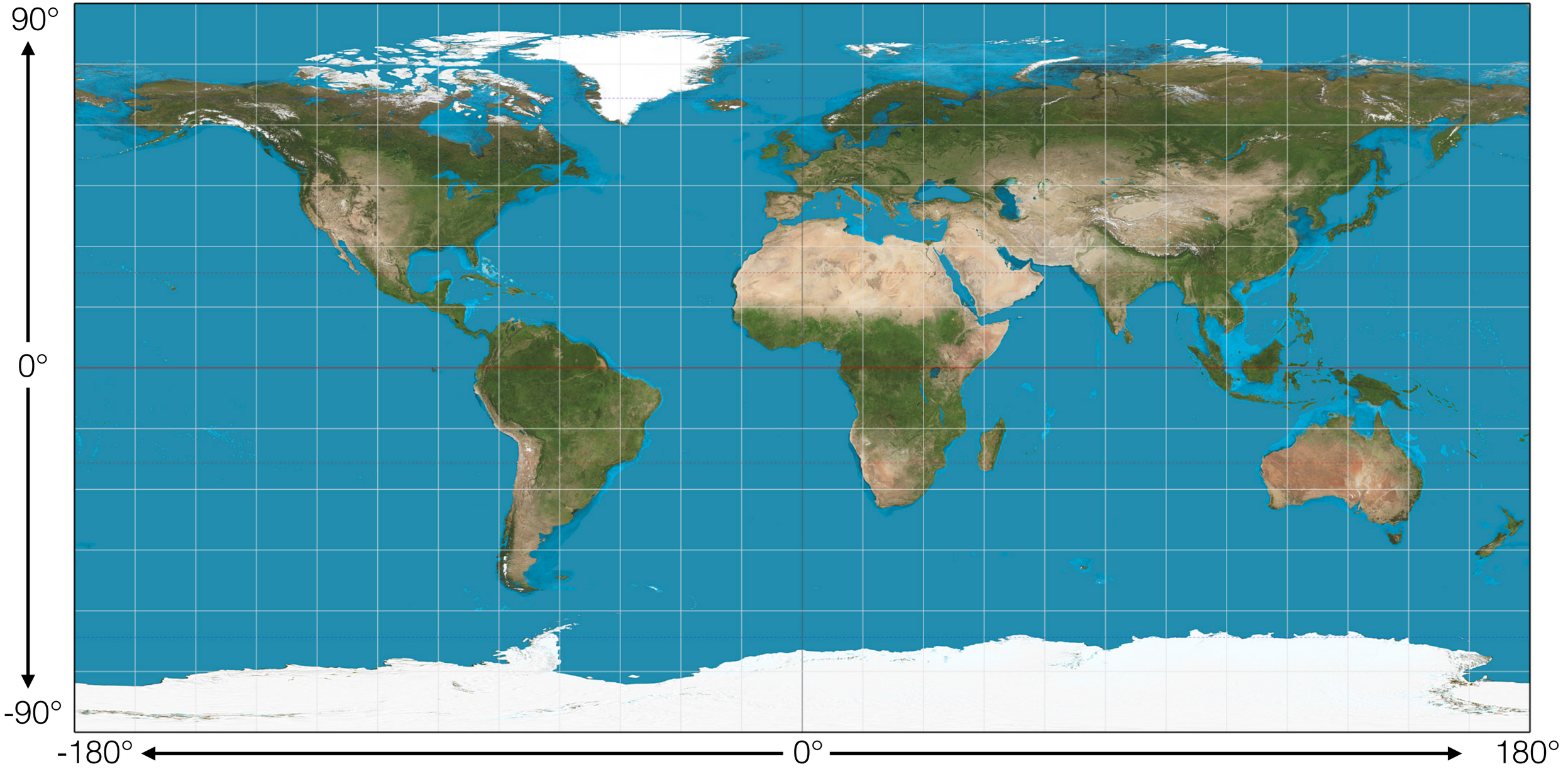


# Everybody loves VR…

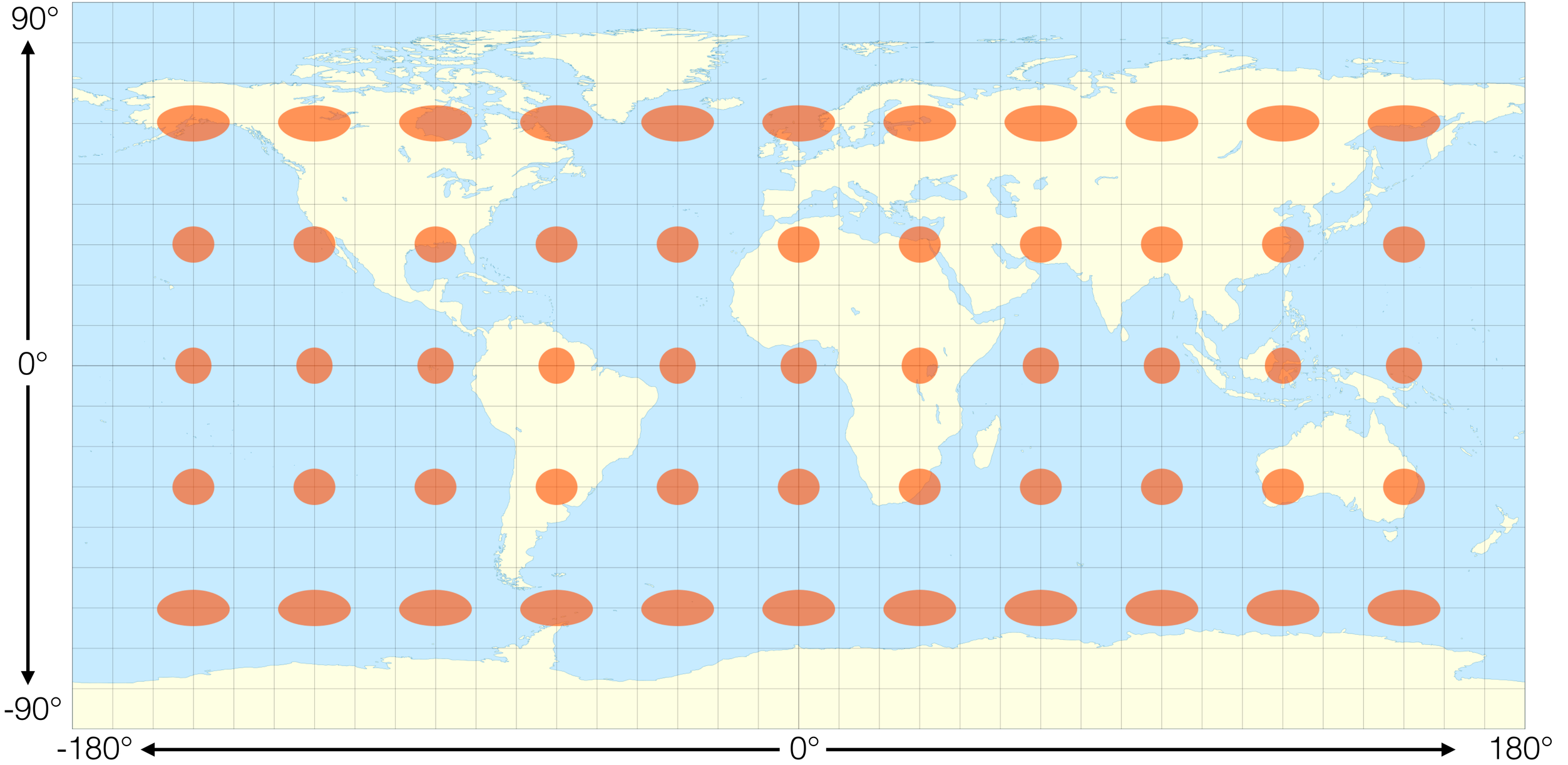

# … except world leaders.

# Omni-Directional (Mono) Image - Equiangular Projection

# Omni-Directional (Mono) Image - Equiangular Projection



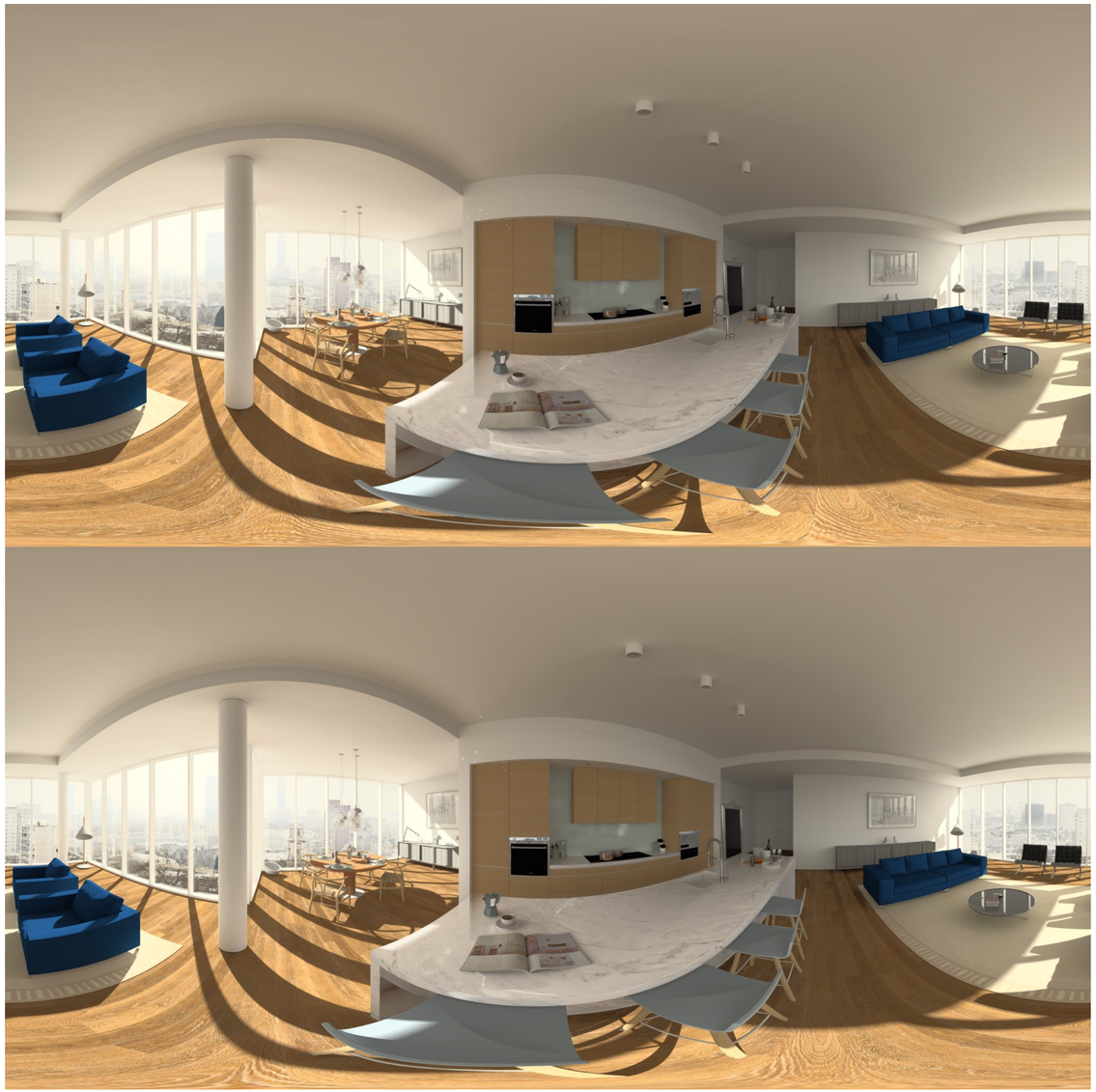
# Omni-Directional (Mono) Image - Equiangular Projection



# Omni-Directional Stereo Image

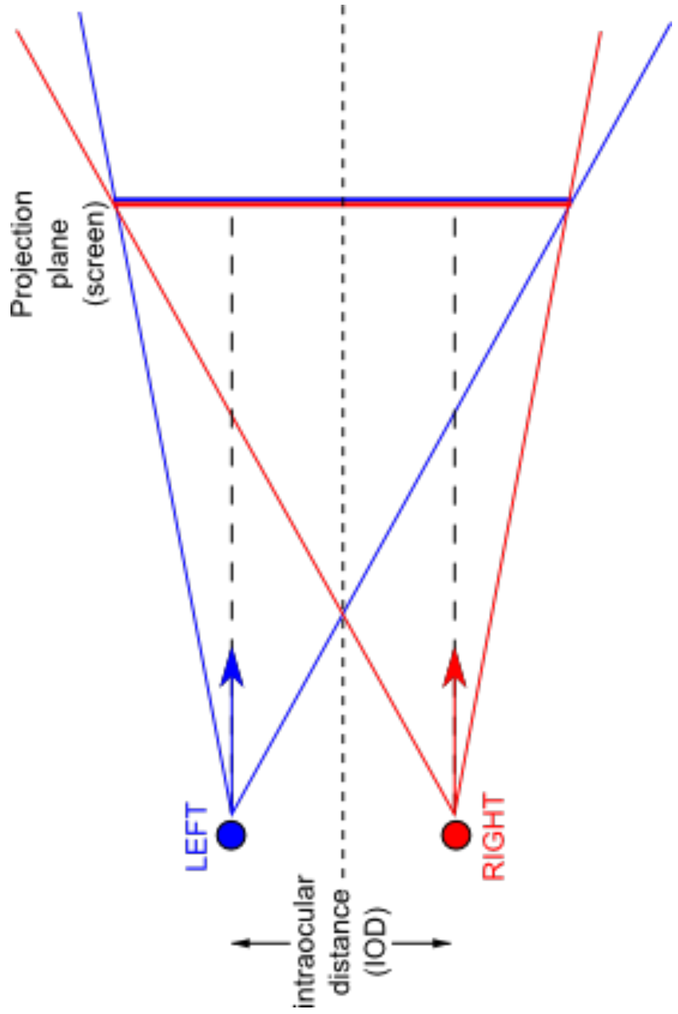
Left eye

Right eye

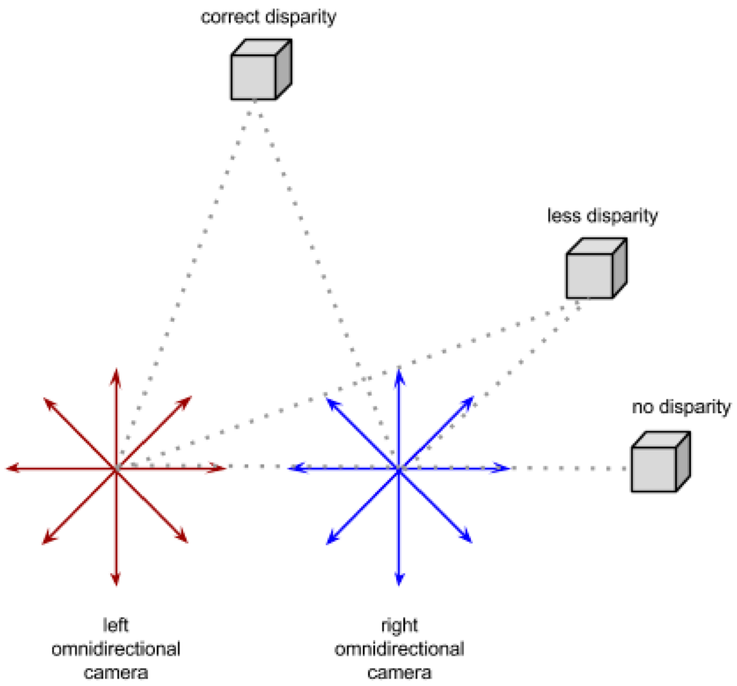


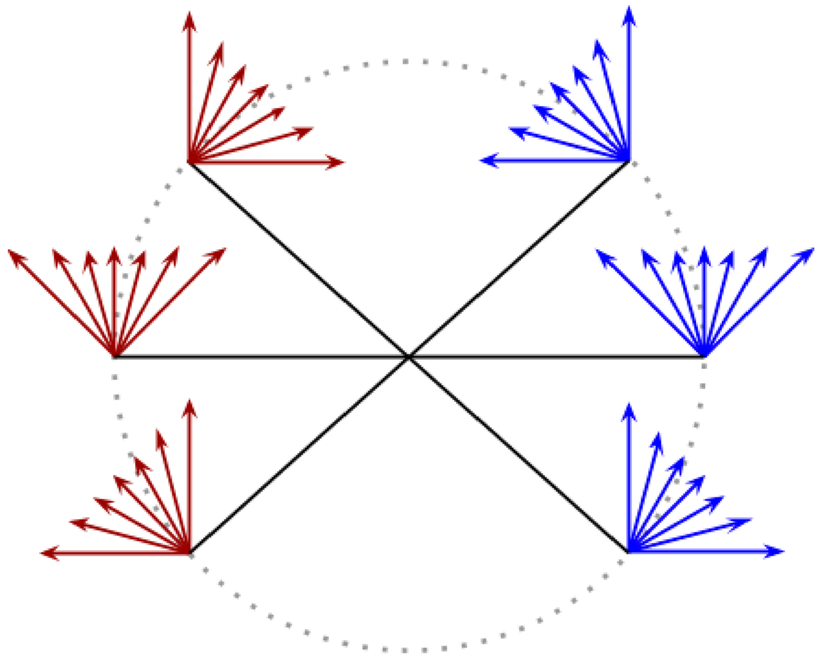
-180° ← 0° → 180°

# Stereo Perspective Rendering



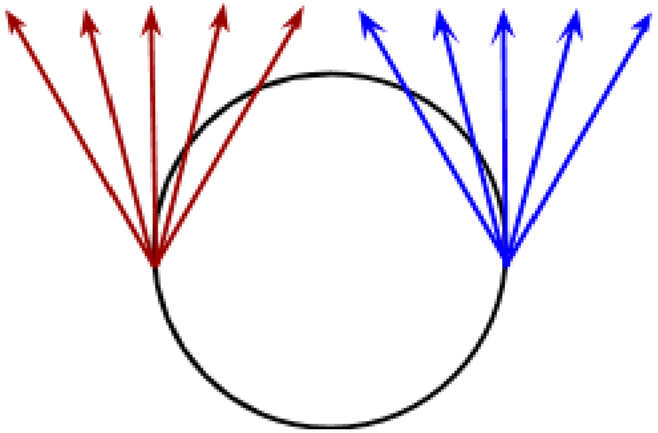
# Stereo for omnidirectional rendering



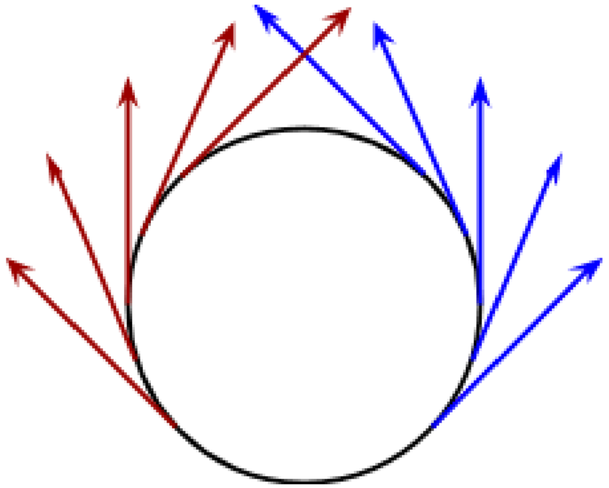



Google Inc., Rendering Omni-directional Stereo Content
https://developers.google.com/vr/jump/rendering-ods-content.pdf

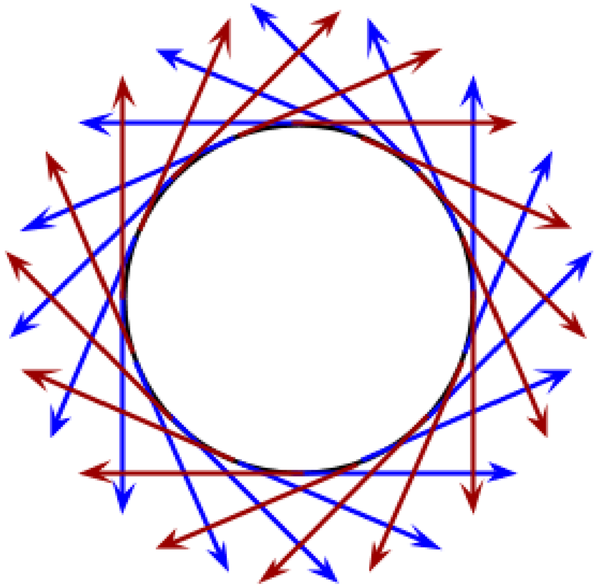
# Stereo for omnidirectional rendering



full-frame **left** and **right** eyes



ODS-approximated **left** and **right** eyes



Google Inc., Rendering Omni-directional Stereo Content
https://developers.google.com/vr/jump/rendering-ods-content.pdf

# My Script for ODS rendering

```
#! /bin/bash

## Viewpoint Coordinates
vpx=4200
vpy=-1520
vpz=163

## Image Resolution
res=16384

## Octree file
oct=octs/octree.oct
amb=octs/ambient.amb

## Image filename
out=image.hdr

## distance between pupils
##Inches
#  ipd=2.48031
##Feet
# ipd=0.206693
##cm
ipd=6.30
##Meter
# ipd=0.063

halfres=$((res/2))

cnt $res $res | \
        rcalc -of -e 'px=$2+rand(($2+1)*($1+3)); py=$1+rand(($2+1)*($1+3)*($1+2))' \
                        -e 'theta=px/'"${res}"'*2*PI-PI/2' \
                        -e 'phi=if(py-'"${halfres}"',PI/2-(py-'"${halfres}"')/'"${halfres}"'*PI,PI/2-py/'"${halfres}"'*PI)' \
                        -e 'IPD='"${ipd}"'' \
                        -e 'scale=if($1-'"${halfres}"'+1,IPD/2,-IPD/2)' \
                        -e '$1=scale*sin(theta)+'"${vpx}"';$2=scale*cos(theta)+'"${vpy}"';$3='"${vpz}"';
$4=sin(theta-PI/2)*cos(phi);$5=cos(theta-PI/2)*cos(phi);$6=sin(phi)' | \
        rtrace -ffc -n 40 -ab 5 -ad 8000 -as 2000 -lw 5e-4 -af $amb -dj 1 -st 0 -ss 50  \
                -x $res -y $res -ld- $oct | \
        ra_rgbe -r - > ${out}
```

# My Script for ODS rendering

```
cnt $res $res | \
    rcalc -of -e 'px=$2+rand(($2+1)*($1+3)); py=$1+rand(($2+1)*($1+3)*($1+2))' \
            -e 'theta=px/'"${res}"'*2*PI-PI/2' \
            -e 'phi=if(py-'"${halfres}"',PI/2-(py-'"${halfres}"')/'"${halfres}"'*PI,PI/2-py/'"${halfres}"'*PI)' \
            -e 'IPD='"${ipd}"'' \
            -e 'scale=if($1-'"${halfres}"'+1,IPD/2,-IPD/2)' \
            -e '$1=scale*sin(theta)+'"${vpx}"';$2=scale*cos(theta)+'"${vpy}"';$3='"${vpz}"'; $4=sin(theta-PI/2)*cos(phi);$5=cos(theta-PI/2)*cos(phi);$6=sin(phi)' | \
    rtrace -ffc -n 40 -ab 5 -ad 8000 -as 2000 -lw 5e-4 -af $amb -dj 1 -st 0 -ss 50  -x $res -y $res -ld- $oct | \
    ra_rgbe -r - > ${out}
```

**Pixel Jittering**

**Left eye / right eye**

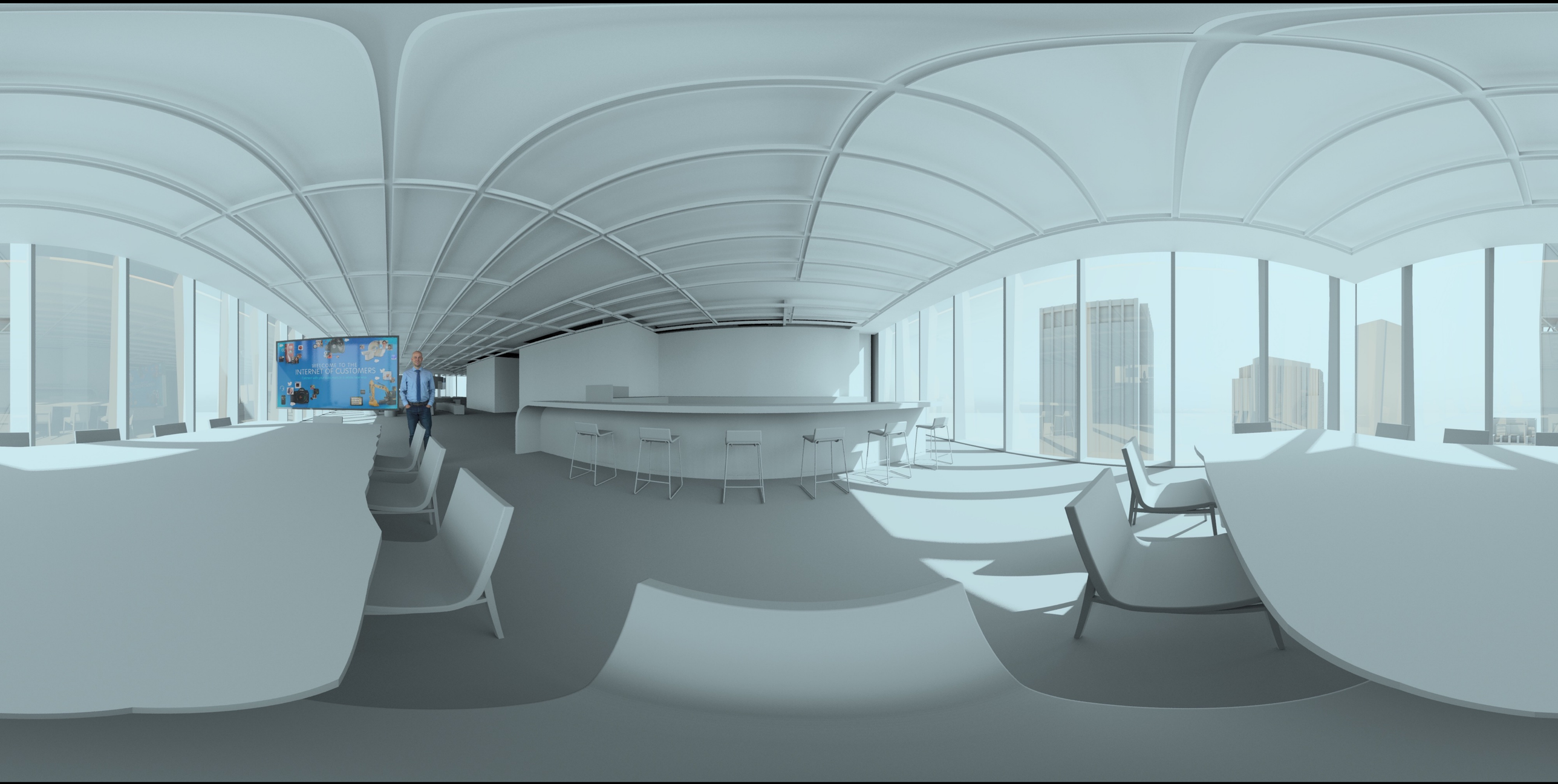
# Easier - Mark Stock's 360.cal

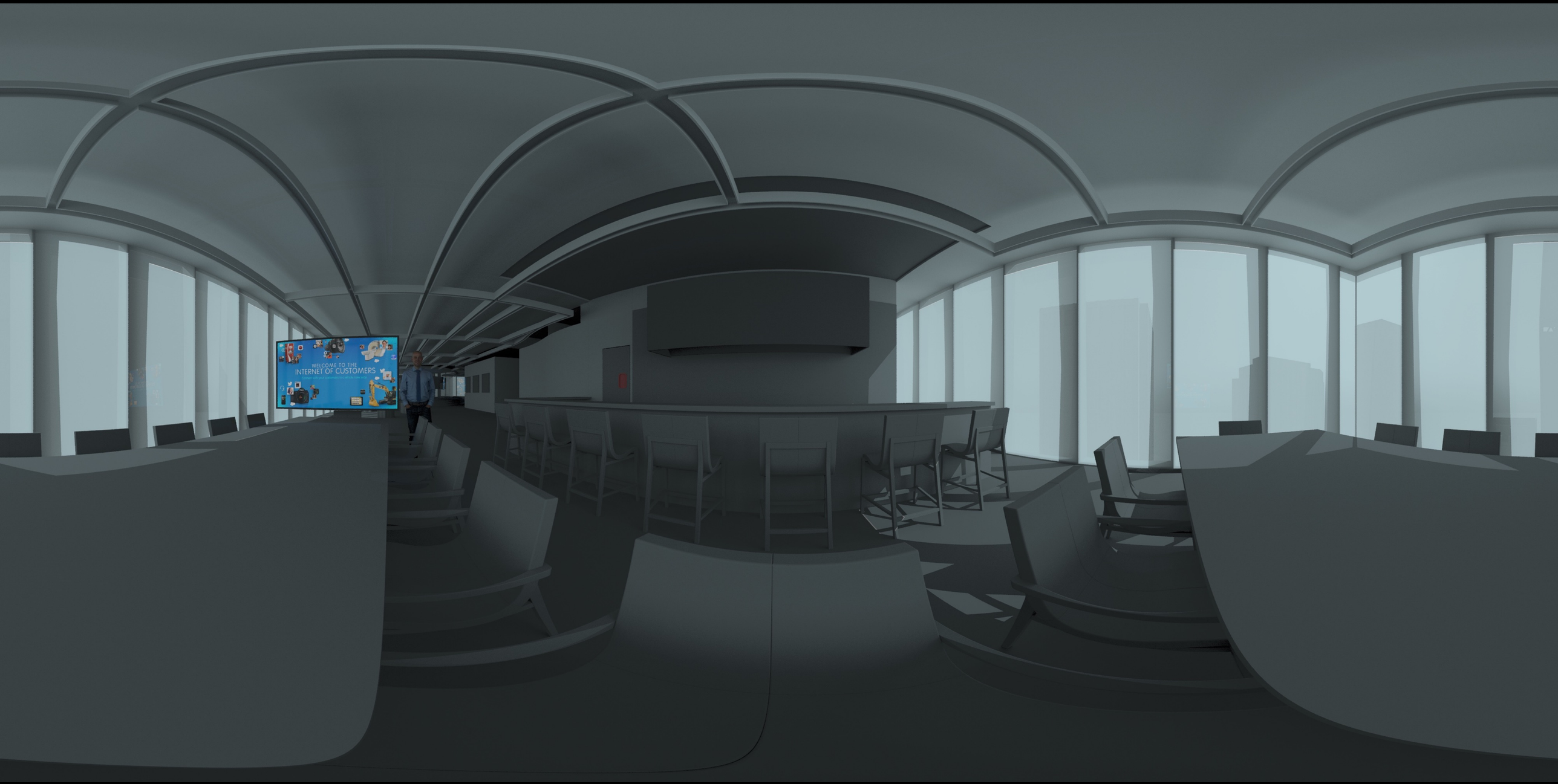
```
{
  3d360.cal
  Definitions for full 360 over-under stereo equirectangular projection
  (c)2014 Mark J. Stock

  Use it like this:
  X=2048; Y=2048; cnt $Y $X | rcalc -f 3d360.cal -e "XD=$X;YD=
$Y;X=0;Y=0;Z=-0.1;IPD=0.06;EX=0;EZ=0" | rtrace [rpict options] -x $X -y $Y -
fac scene.oct > out.hdr

  Parameters defined externally:
  X : neck rotation origin x
  Y : neck rotation origin y
  Z : neck rotation origin z
  XD : horizontal picture dimension ( pixels )
  YD : vertical picture dimension ( pixels )
  IPD : inter-pupillary distance
       this is between 0.055m and 0.07m on most humans
  These don't seem to work all that well:
  EX : forward distance between neck rotation center and bridge of nose
(between eyes)
       this is between 0.05m and 0.07m on most humans
  EZ : vertical distance between neck rotation center and eye elevation when
altitude is 0 degrees
       this is around 0.1m on most humans
}

{ Direction of the current pixel (both angles in radians) }
px = $2;
py = YD - $1;
frac(x) : x - floor(x);
altitude = (frac((py-0.5)/(YD/2)) - 0.5) * PI;
{ to do over-under stereo, azimuth is easy }
azimut = px * 2 * PI / XD;

{ Transformation into a direction vector }
xdir = cos(azimut) * cos(altitude);
ydir = sin(azimut) * cos(altitude);
zdir = sin(altitude);

{ Transform the viewpoint to account for the eye position }
dx = EX;
dy = if($1 - YD/2, 0.5*IPD, -0.5*IPD);
dz = EZ;
xpos = X + xdir*dx - sin(azimut)*dy + cos(azimut)*zdir*dz;
ypos = Y + ydir*dx + cos(azimut)*dy + sin(azimut)*zdir*dz;
zpos = Z - zdir*dx +          0*dy + cos(altitude)   *dz;

{ Output line to rtrace; each ray needs: xorg yorg zorg xdir ydir zdir }
$1 = xpos; $2 = ypos; $3 = zpos;
$4 = xdir; $5 = ydir; $6 = zdir;

{ EOF }
```

# Easiest - Radiance view type?

- -vto[ms]
- direction vector length = inter pupil distance?

# Examples


WELCOME TO THE
INTERNET OF CUSTOMERS


WELCOME TO THE
INTERNET OF CUSTOMERS

# Tone mapping is tricky

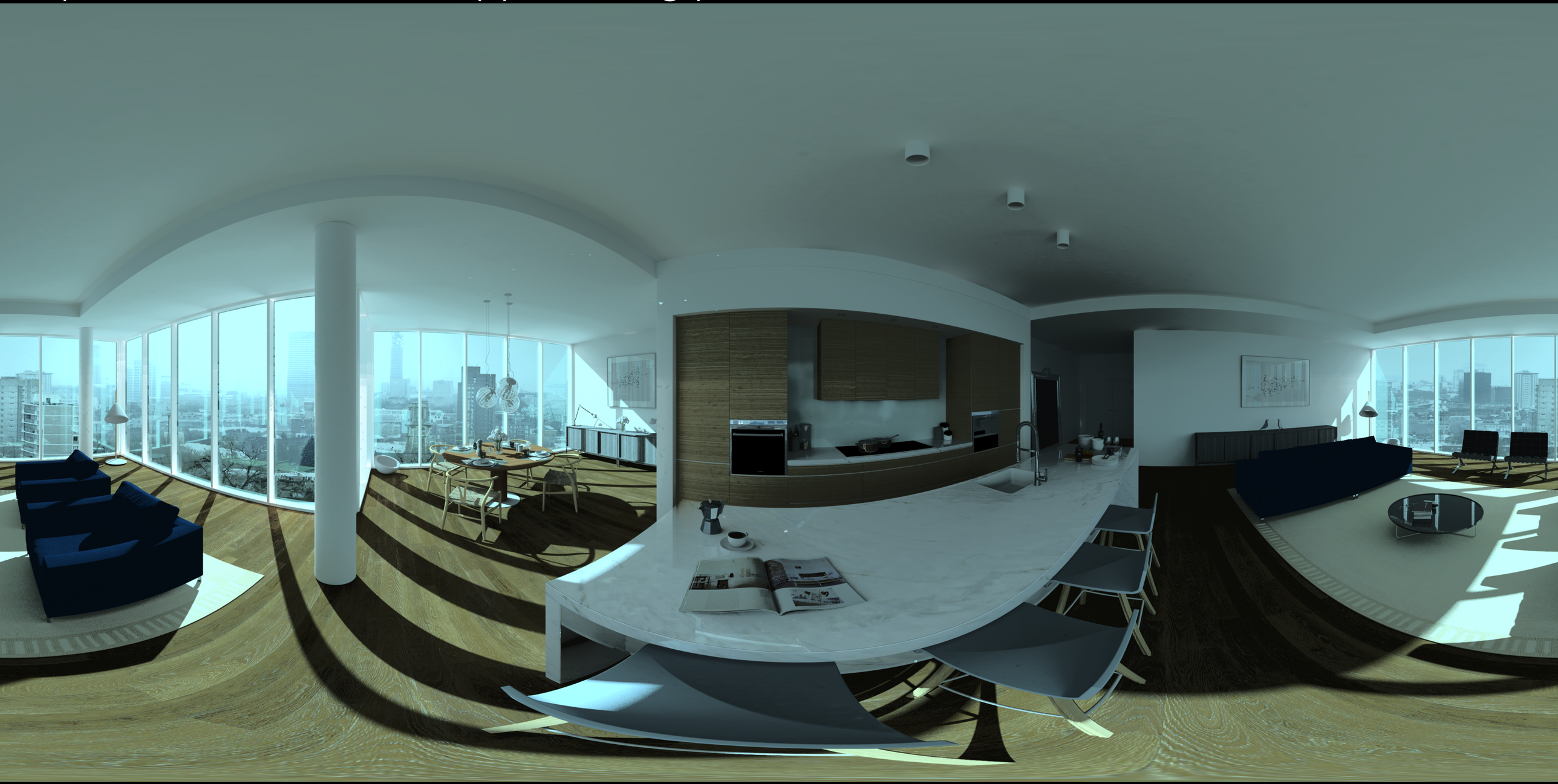
Tone mapped using pond

# One pane tinted - tone mapped using pond

Pretty good so far!

Back to clear - Tone mapped using pond

All panes tinted - Tone mapped using pcond



Other than color did anything change?

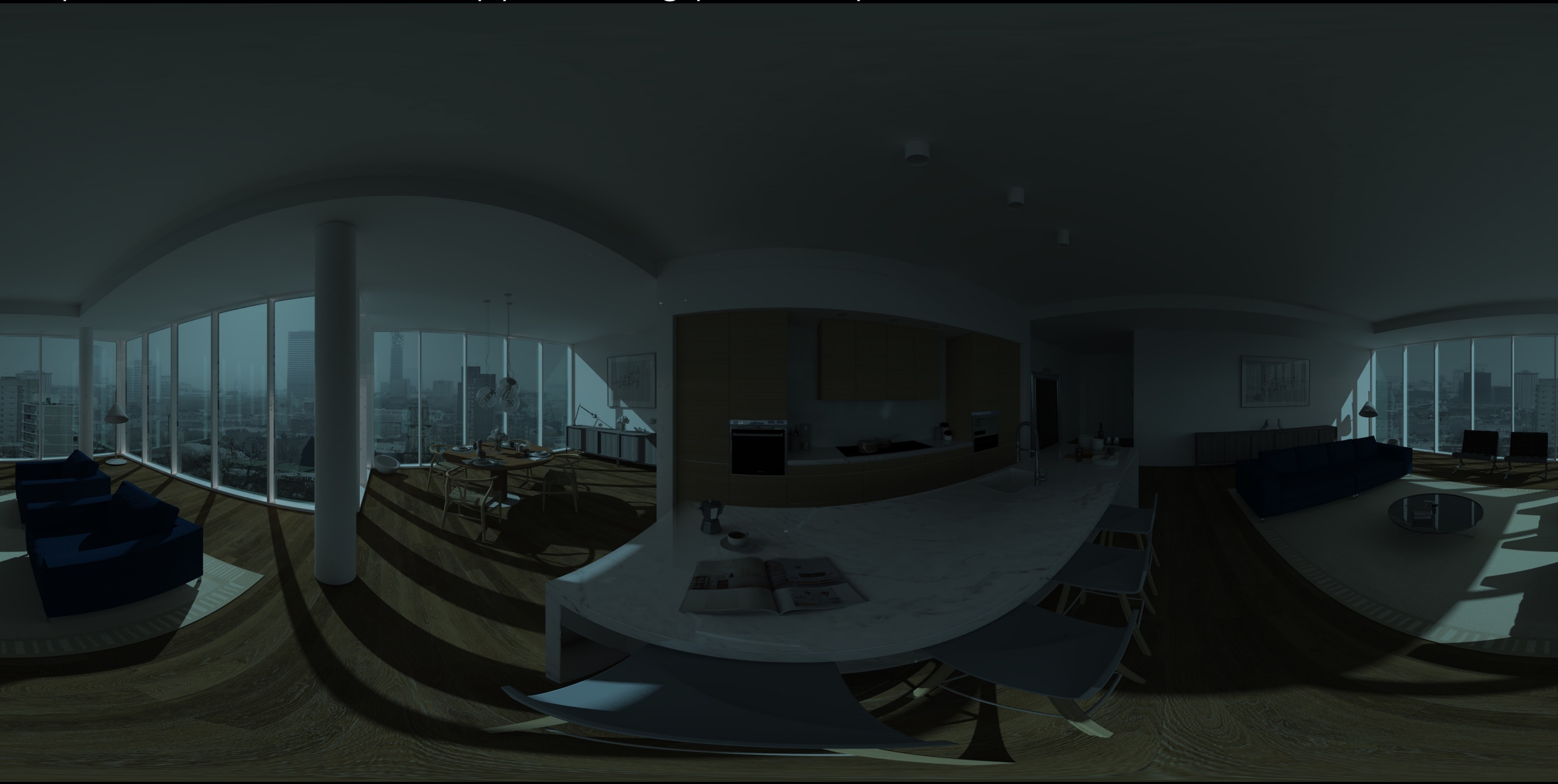
# I know, Phisto!

- phisto clear.hdr tinted.hdr > both.hist
- pcond -I <both.hist clear.hdr > clear.pic
- pcond -I <both.hist tinted.hdr > tinted.pic

    - That’ll work for sure!

Back to clear - Tone mapped using phisto & pcond

All panes tinted - Tone mapped using phisto & pond



Better, but ugh.

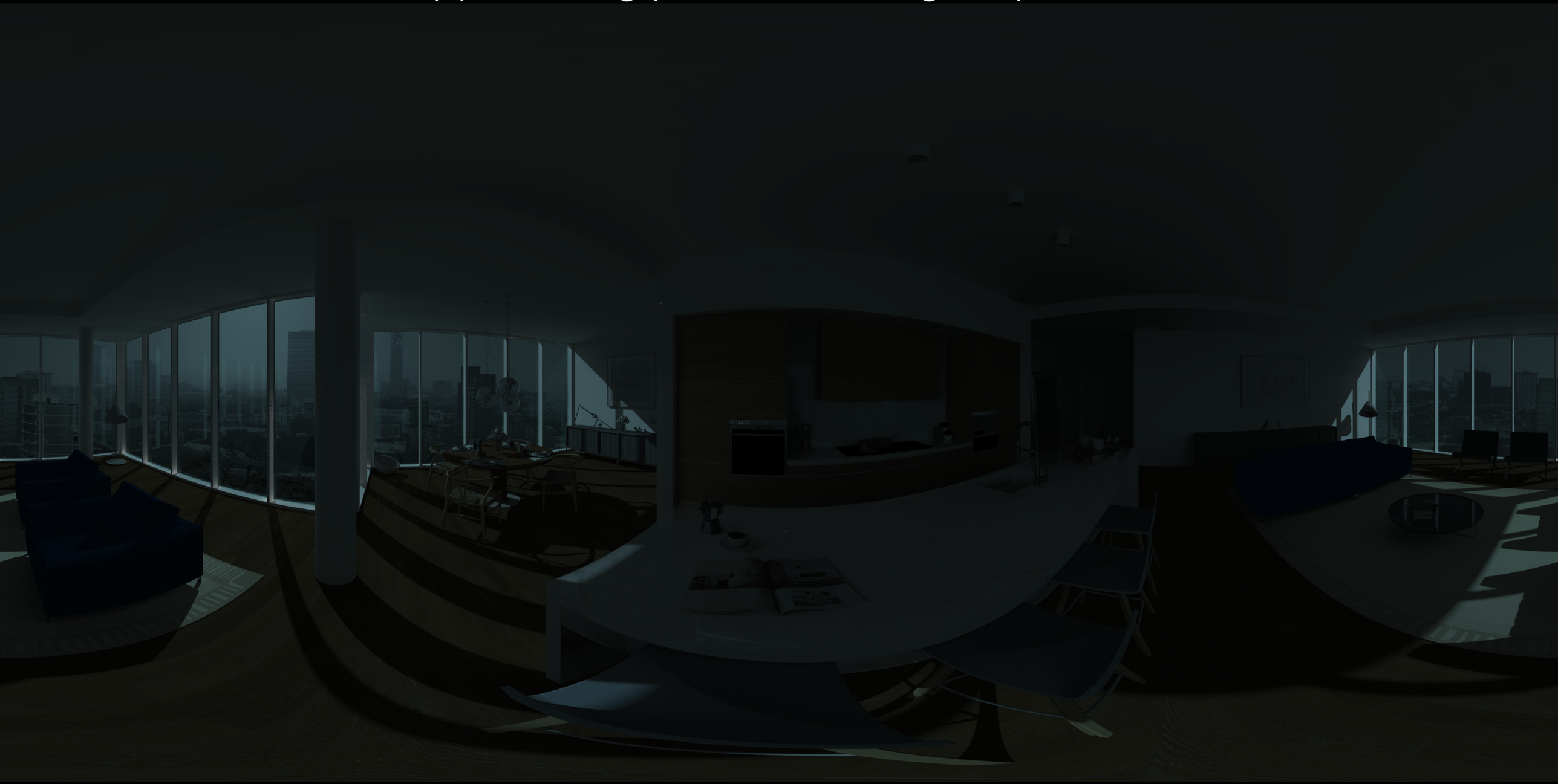
# Hmm… maybe still Phisto!

- phisto clear.hdr > clear.hist
- pcond -I <clear.hist clear.hdr > clear.pic
- pcond -I <clear.hist tinted.hdr > tinted.pic

    - I bet that’ll be better!

Back to clear - Tone mapped using phisto one image & pcond

Back to clear - Tone mapped using phisto one image & pcond



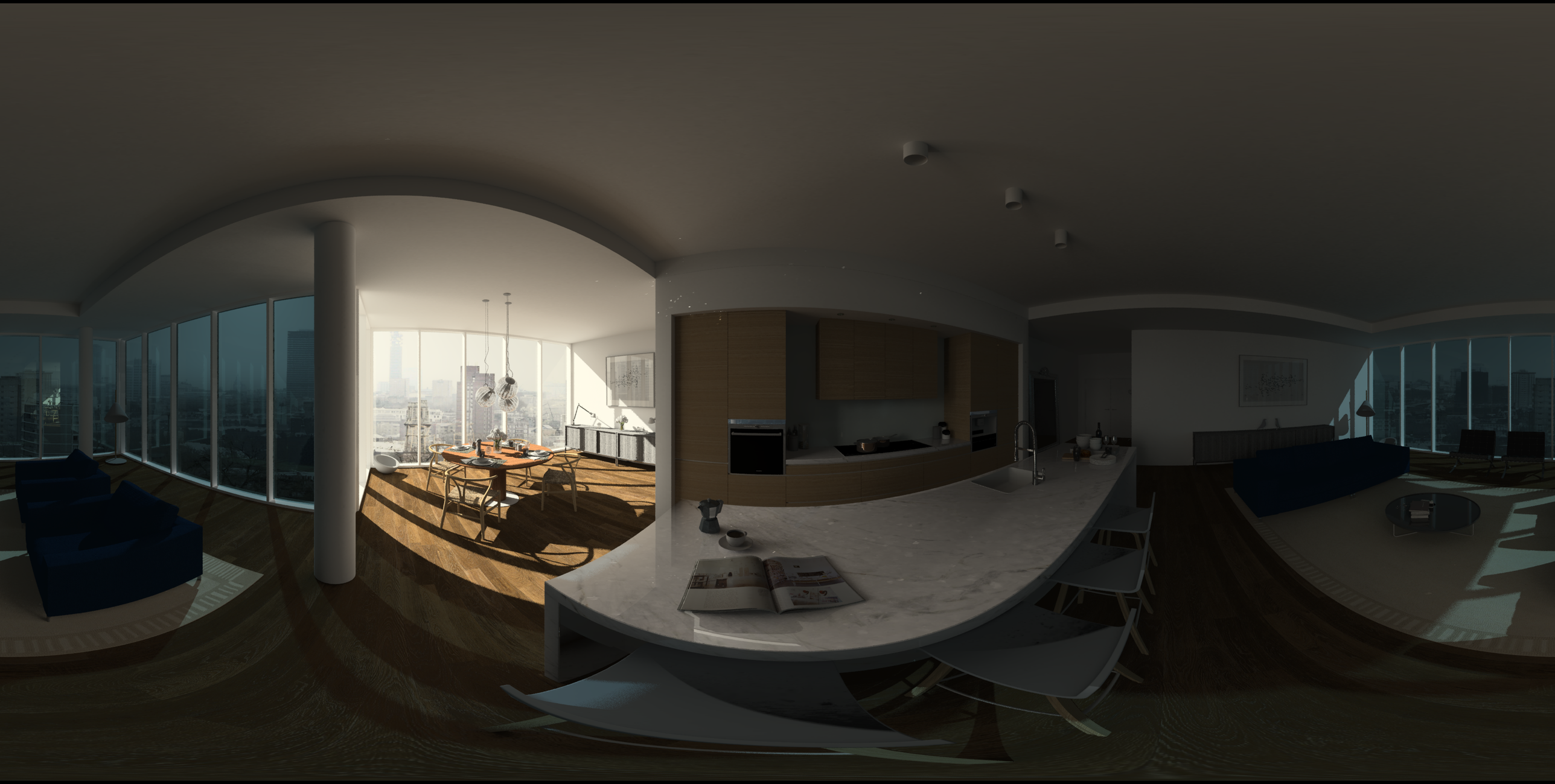
Better yet, but still ugh.

# We need some ambient lighting.

- Electric lighting and/or leave dining room clear.

- Then pcond using phisto from clear image
- phisto clear.hdr > clear.hist
- pcond -I <clear.hist clear.hdr > clear.pic
- pcond -I <clear.hist tinted.hdr > tinted.pic

    - fingers crossed.



73%

30%

13%

6%

2%

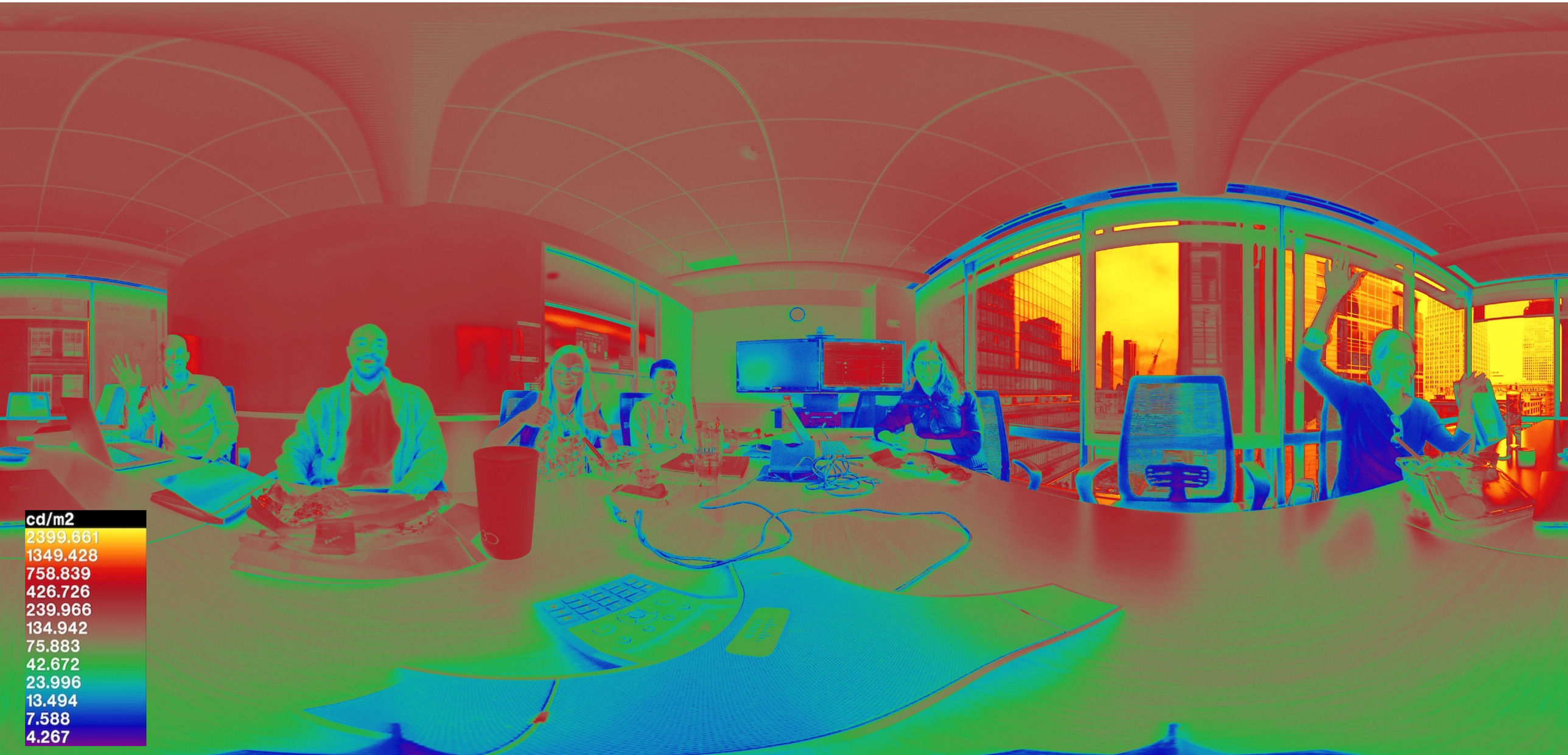
# Next steps

- Luminance Calibration
- Luminance Scaling

This is a “workshop” so…

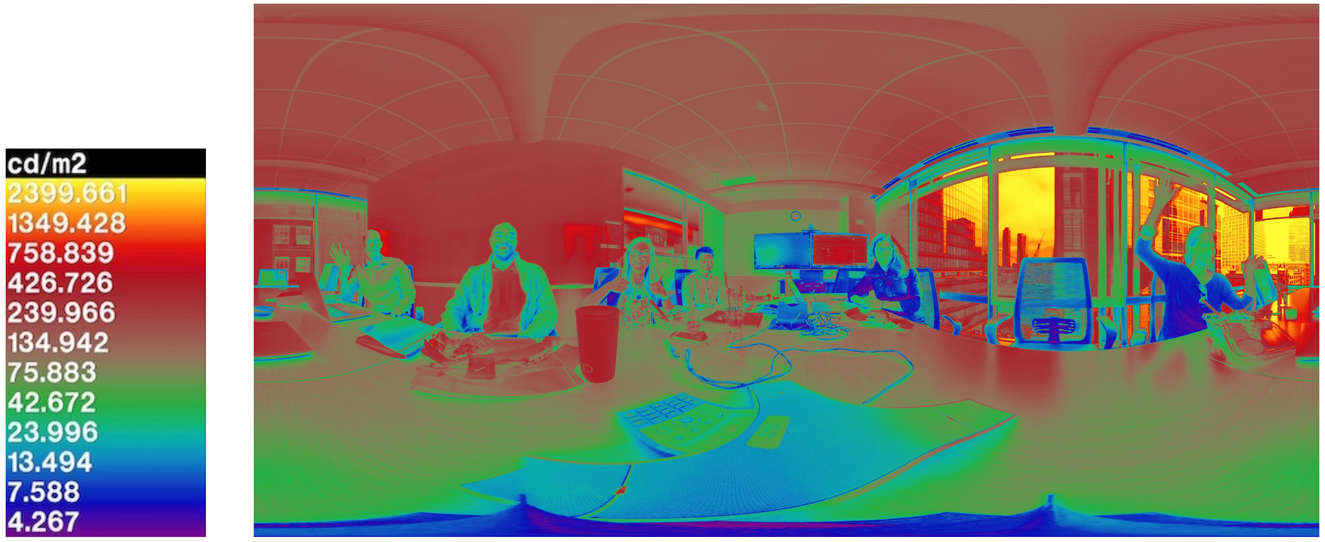
# Omni-directional HDR photographs

# Omni-directional HDR photographs



# Omni-directional HDR photographs

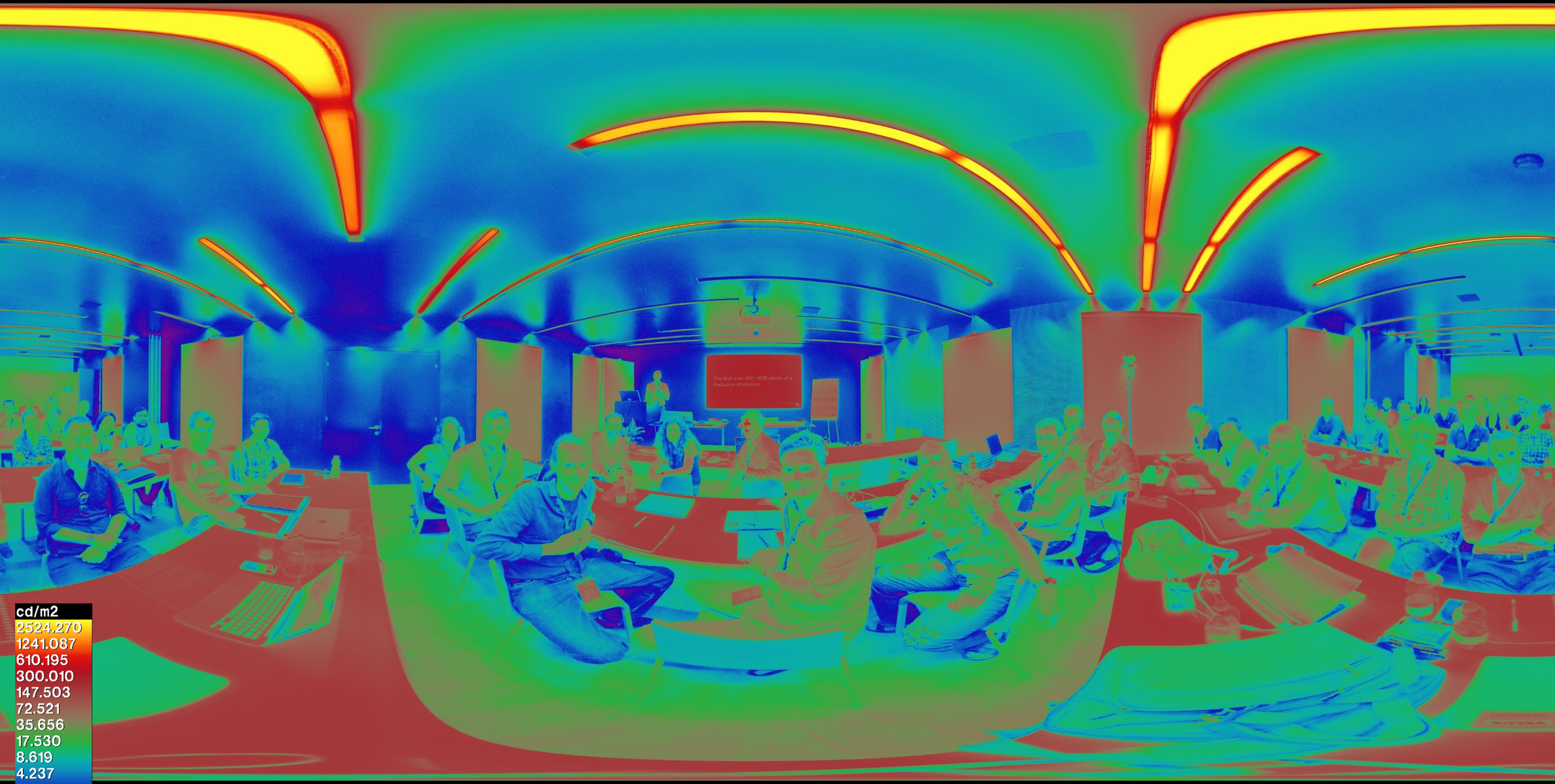

- Real world ‘Adaptive zone’ glare analysis (Jakubiec & Reinhart 2011)
- Image based rendering



# The first ever 360° HDR photo of a Radiance Workshop.

30 August 2016


The first ever 360° HDR photo of a
Radiance Workshop.


The first ever 360° HDR photo of a Radiance Workshop.
cd/m2
2524.270
1241.087
610.195
300.010
147.503
72.521
35.656
17.530
8.619
4.237
2.083
1.024

# Economical HDR image logging

## $2,540 (not including photometer for calibration)

### Licor Photometer ($550)

The photometer measures vertical illuminance just above the lens. HDR images are calibrated against the vertical illuminance measured while photos are taken.

### Canon EOS 60D Camera ($899)

The Canon 60D is a relatively inexpensive digital SLR camera with a cropped frame sensor.

### Sigma 4.5mm Fisheye Lens ($899)

The Sigma 4.5mm F2.8 EX DC HSM circular fisheye ($899) lens is designed for DSLR cameras with a cropped sensor.

### Apple Mac mini Computer ($599)

The mac mini controls the image capture and performs image processing and glare analysis. We no longer use mac specific software, so this could be a linux computer instead.

### LabJack U3-HV DAQ ($114)

The LabJack U3-HV reads the signal from the licor photosensor (and also a temperature sensor we've installed for fun). The labjack connects to the mac mini via a USB cable, communication can be programmed using a python library.

### EME Systems UTA Amp ($150)

The UTA converts the micro-amp output of the licor photosensor to volts.

### Canon ACK-E6 Adapter ($138)

The ACK-E6 power adapter plugs into an AC power outlet to power the Canon camera.

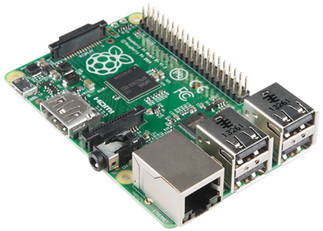
## $135 (not including photometer for calibration)

Raspberry Pi Camera Module

$25

wide angle lens

$25



Raspberry Pi

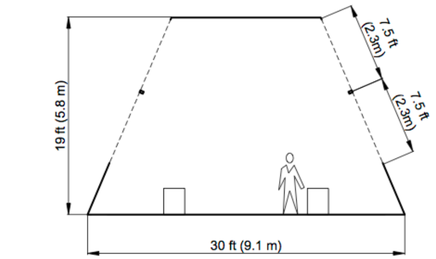
$35


64 GB micro SD

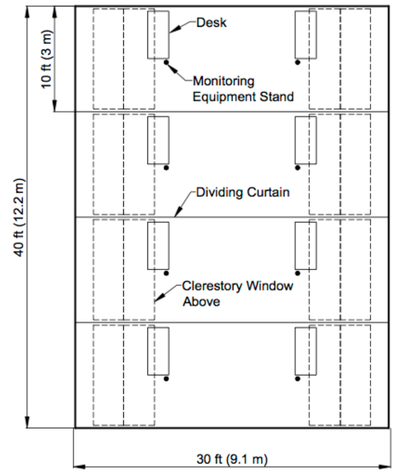
$40

# Testing Glare Metrics in a Brightly Daylit Mockup

McNeil, A., Burrell, G. 2016 Applicability of DGP and DGI for Evaluating Glare in a Brightly Dalit Space. SimBuild, Salt Lake City




MOCKUP SECTION




MOCKUP PLAN

Bay 4:
Diffuse
10% VLT

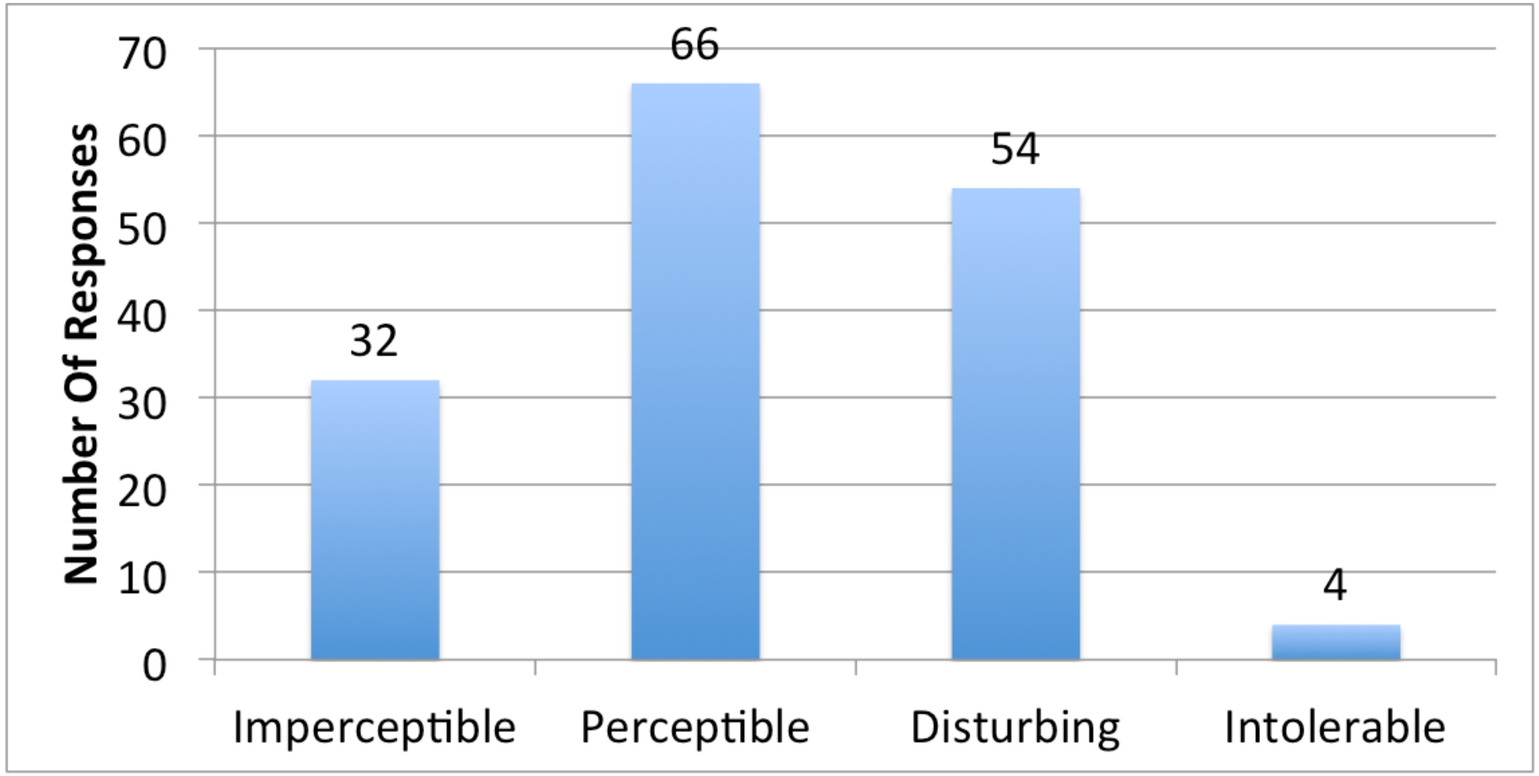
Bay 3:
Clear with 50% white frit
70% VLT

Bay 2:
Clear with 25% silver frit
58% VLT

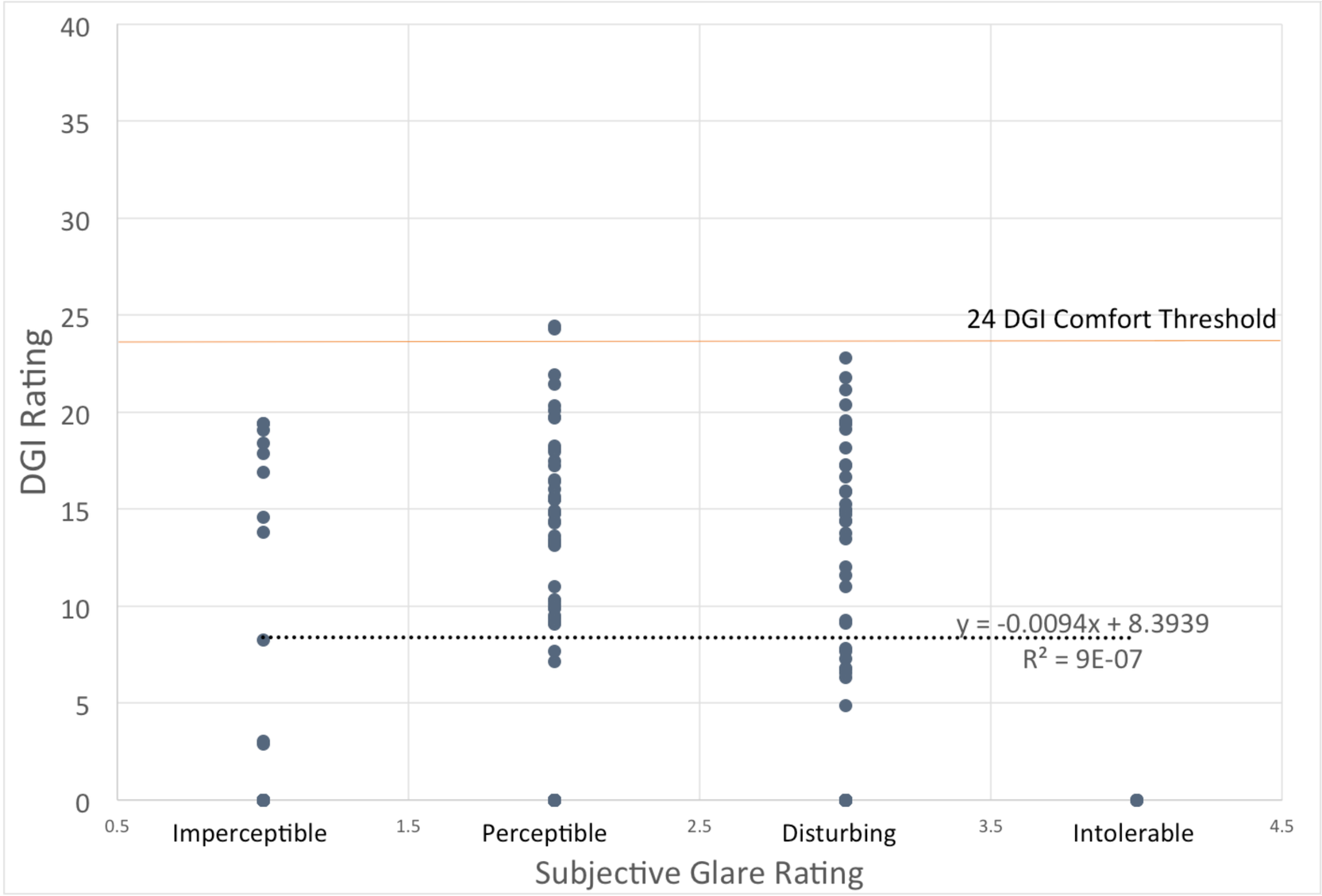
Bay 1:
Clear with 60% silver frit
41% VLT

# Subjective rating collected with accompanying HDR images.

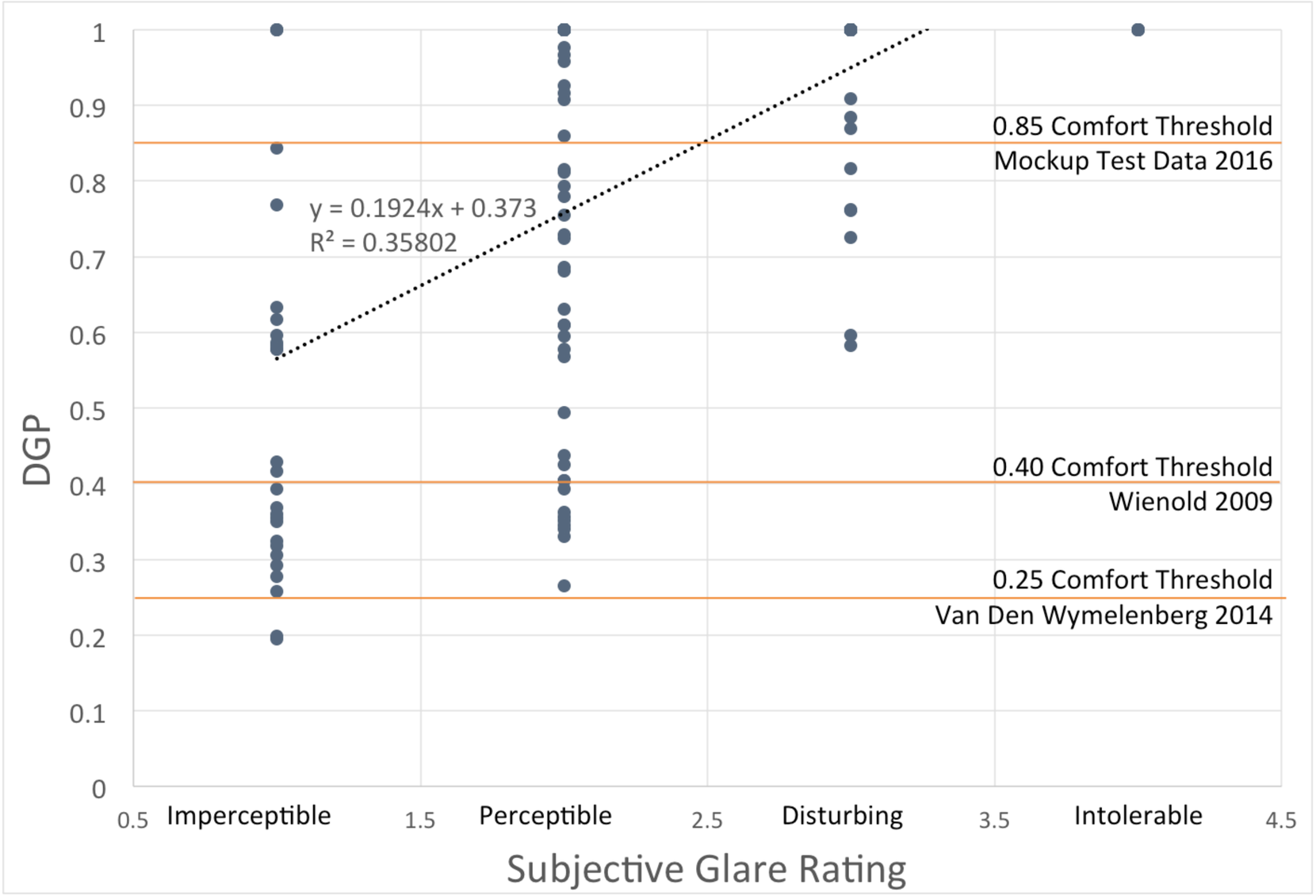
- 16 subjects
- 156 responses



# DGI - No Correlation with Subjective Rating



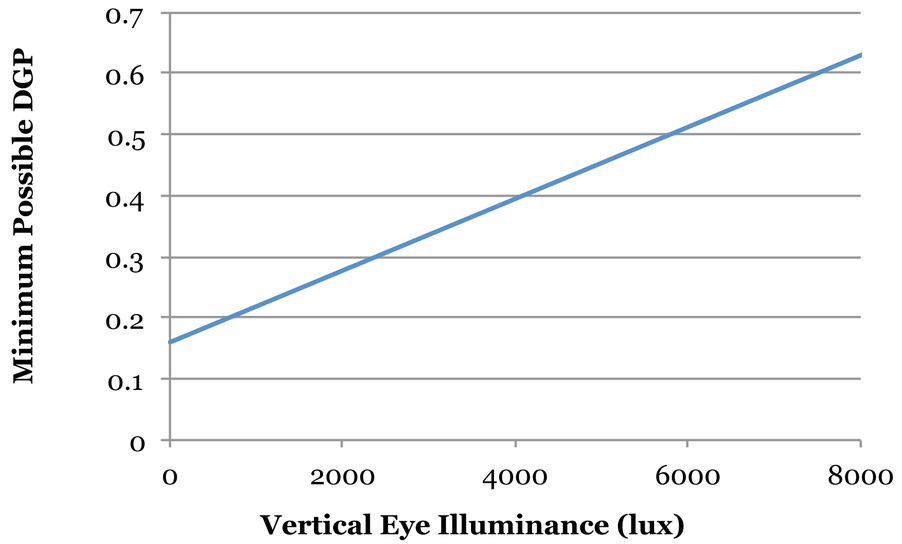
# DGP - Strong Correlation with Subjective Rating, but comfort threshold at 0.85



# DGP's vertical illuminance term

$$DGP = 5.87 \cdot 10^{-5} \cdot E_v + 9.18 \cdot 10^{-2} \cdot \log\left(1 + \sum_{i=1}^{n} \frac{L_{s,i}^2 \, \omega_{s,i}}{E_v^{1.87} + P_{s,i}^2}\right) + 0.16$$

$E_v$ = Vertical illuminance at the eye
$L_b$ = Average background luminance in the field of view
$L_{s,i}$ = Luminance for glare source i
$P_i$ = Position index for glare source i
$\omega_{s,i}$ = Solid angle for glare source i




| Subjective rating | DGP range |
|---|---|
| Imperceptible Glare | < 0.35 |
| Perceptible Glare | 0.35 – 0.40 |
| Disturbing Glare | 0.40 – 0.45 |
| Intolerable Glare | > 0.45 |

# Perceived Monitor Contrast Ratio

$$\text{Monitor Contrast Ratio} = \frac{L_H + L_r}{L_L + L_r}$$

(Jakubiec et al. 2015)

$$\text{Perceived Monitor Contrast Ratio} = \frac{L_H + L_r + L_V}{L_L + L_r + L_V}$$

$L_H$ = Luminance of a white pixel

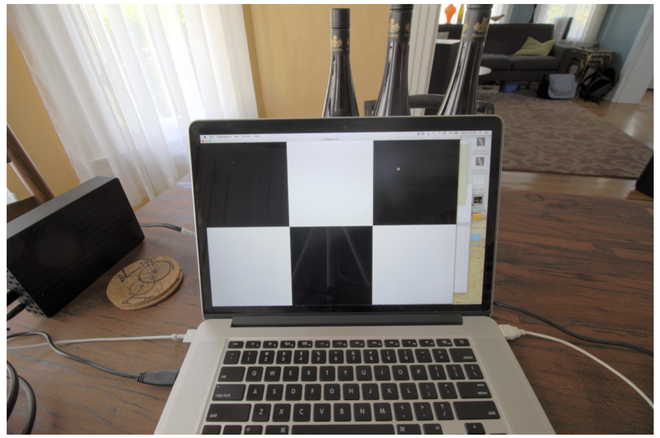
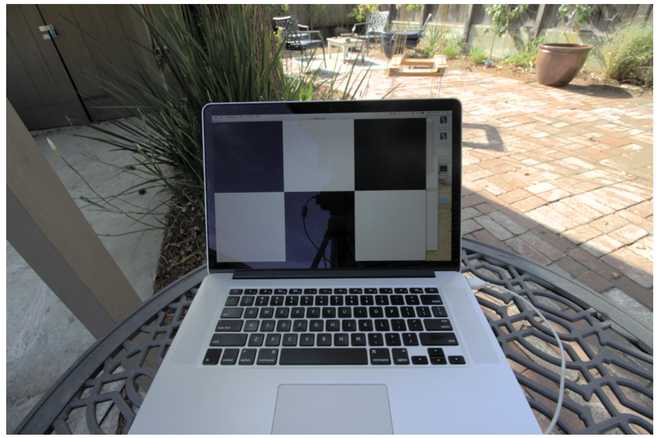
$L_L$ = Luminance of a black pixel

$L_r$ = Veiling Reflection (screen)

$L_v$ = Veiling Luminance (ocular)

# Perceived Contrast

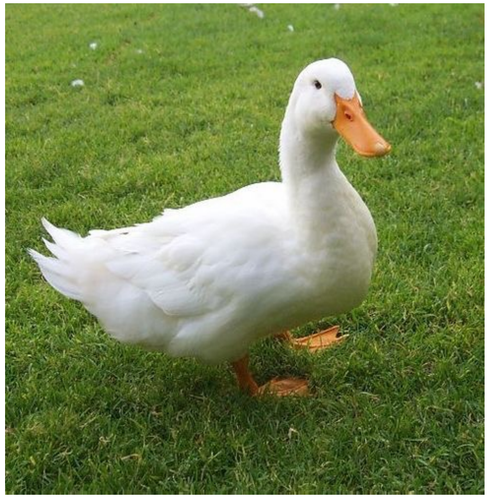
resulting from veiling luminance (eye) and veiling reflections (screen)

| | Inside | Outside Shade | Outside Sun |
|---|---|---|---|
| Monitor Contrast Ratio | 84.7 | 10.3 | 8.3 |
| Perceived Monitor Contrast Ratio | 42.2 | 5.6 | 1.9 |
| Image |  |  |  |

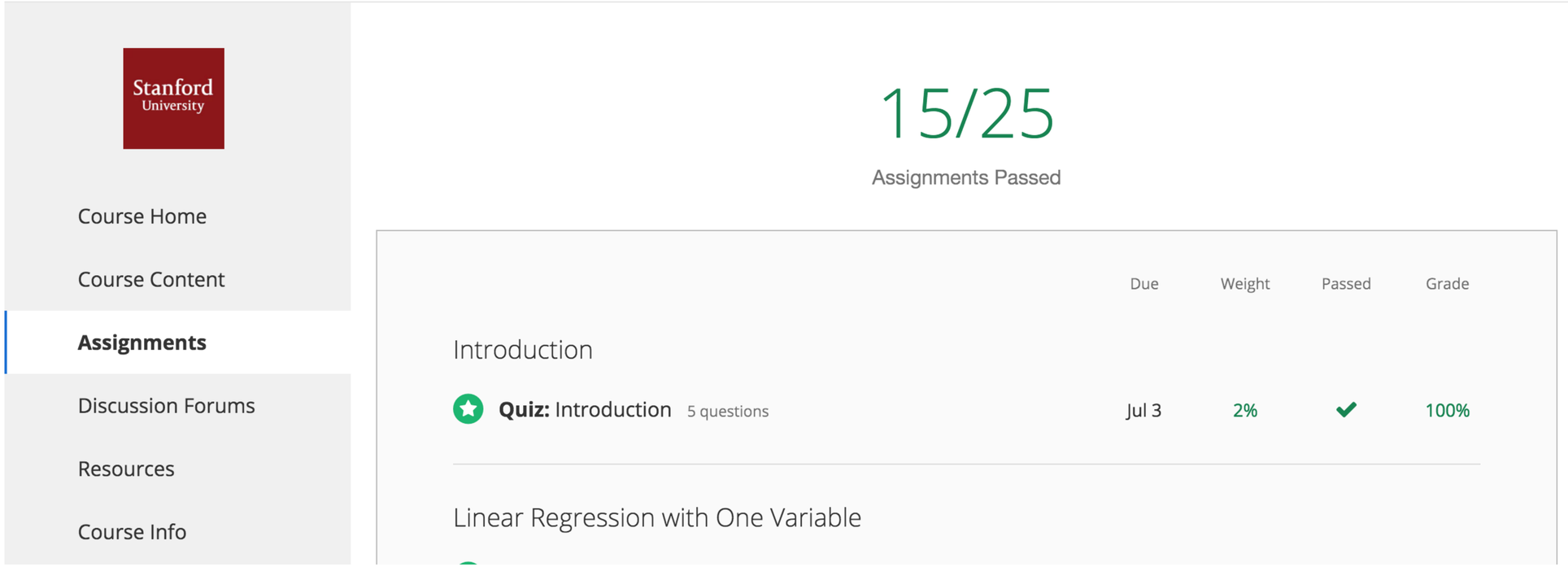
# Machine Learning & Glare Classification


We can train a neural network to classify images with cars.

Why not try identifying glare in HDR images?



# My credentials in machine learning - 60% of an online course

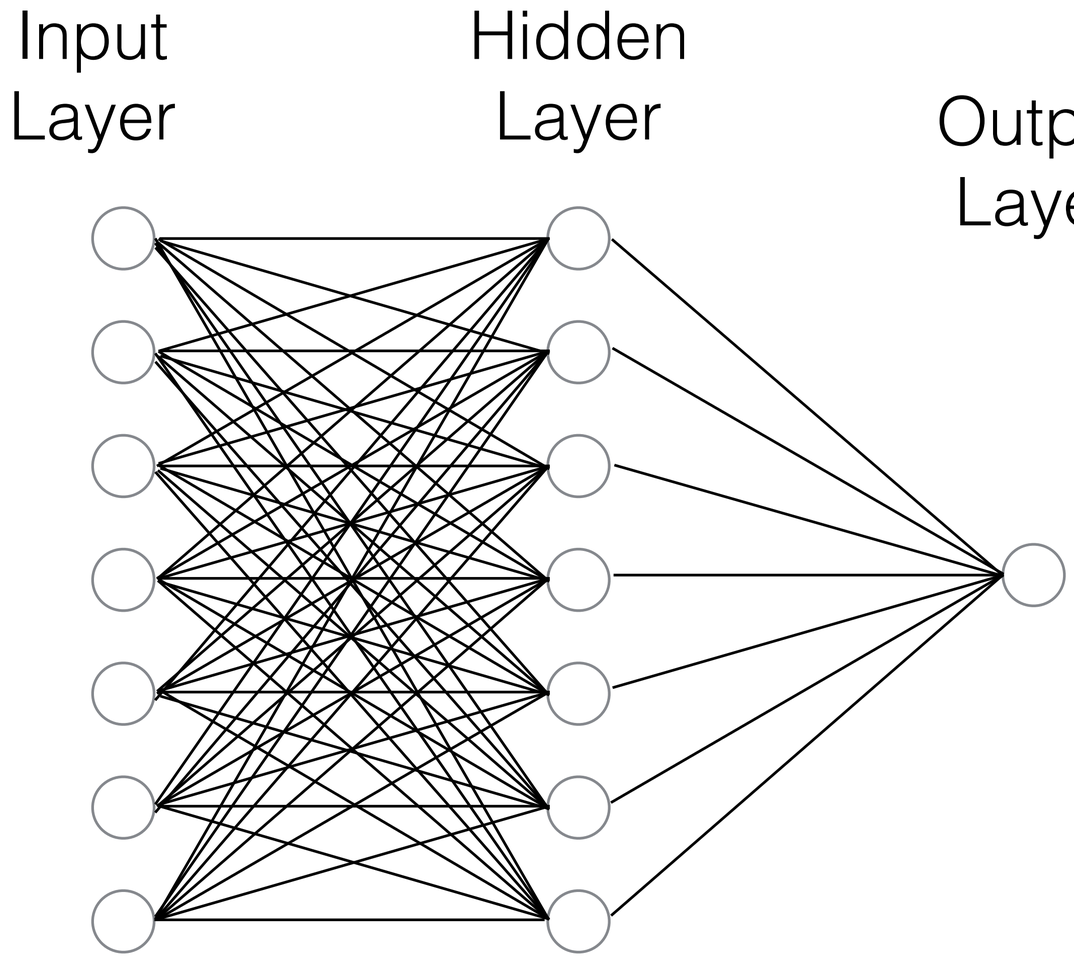
# Neural Network Architecture

Image resolution 100x75

Training set = 95 images
Test set = 29 images

Image sets were selected randomly

The analysis was run 4 times - each time with different randomly selected training and test sets.



# Results - test set

| | | Subjective Rating | |
|---|---|---|---|
| | | Glare | No Glare |
| Neural Network Classification | Glare | **22** | **11** |
| | No Glare | **18** | **65** |

Accuracy = 75%
Precision = 67%
Recall = 55%
F1 = 0.60

| | | Subjective Rating | |
|---|---|---|---|
| | | Glare | No Glare |
| DGP 0.4 threshold | Glare | **40** | **57** |
| | No Glare | **0** | **19** |

Accuracy = 51%
Precision = 41%
Recall = 100%
F1 = 0.58

| | | Subjective Rating | |
|---|---|---|---|
| | | Glare | No Glare |
| DGP 0.85 threshold | Glare | **38** | **34** |
| | No Glare | **2** | **25** |

Accuracy = 68%
Precision = 53%
Recall = 95%
F1 = 0.68