Interactive is the New Black

David Smith Design IALD, IES, EIT
Lighting Designer, Buro Happold

Buro Happold

First: A crash course in scripting for Radiance.

Choose a programming language that works on your system. *

It doesn’t matter which one, as long as it supports variables, loops, and calling external commands.

Most modern programming languages have these capabilities.

First: A crash course in scripting for Radiance.

Choose a programming language that works on your system. *

It doesn’t matter which one, as long as it supports variables, loops, and calling external commands.

Most modern programming languages have these capabilities.

* Or just choose Python. If you have to use Ecotect, choose Lua.
You don't have to use Ecotect.

Write down an assembly line process

```
gensky 4 1 12:00 -c -m 75 -o 73.96 -a 40.79 > sky.rad

oconv sky.rad model.rad > model.oct

rpict -vf view1.vf @rpict_options.txt model.oct > temp.hdr

ra_tiff temp.hdr view1.tiff
```

Figure out which parts you would ever consider changing.

```
gensky 4 1 12:00 -c -m 75 -o 73.96 -a 40.79 > sky.rad
oconv sky.rad model.rad > model.oct
rpict -vf view1.vf @rpict_options.txt model.oct > temp.hdr
ra_tiff temp.hdr view1.tif
```

Replace them with the variable names we will be using.

```
gensky M D HH:MM  -c -m 75 -o 73.96 -a 40.79 > sky.rad
oconv sky.rad model.rad > model.oct
rpict -vf VIEWFILE @rpict_options.txt model.oct > temp.hdr
ra_tiff temp.hdr IMAGE
```

Add a way to call this block of code, making it a *function*.

```
GenerateImage(M,D,HH,MM,VIEWFILE,IMAGE)
    gensky M D HH:MM -c -m 75 -o 73.96 -a 40.79 > sky.rad
    oconv sky.rad model.rad > model.oct
    rpict -vf VIEWFILE @rpict_options.txt model.oct > temp.hdr
    ra_tiff temp.hdr IMAGE
```

Formatting convention: Indented lines of code “belong” to the first non-indented line above it.

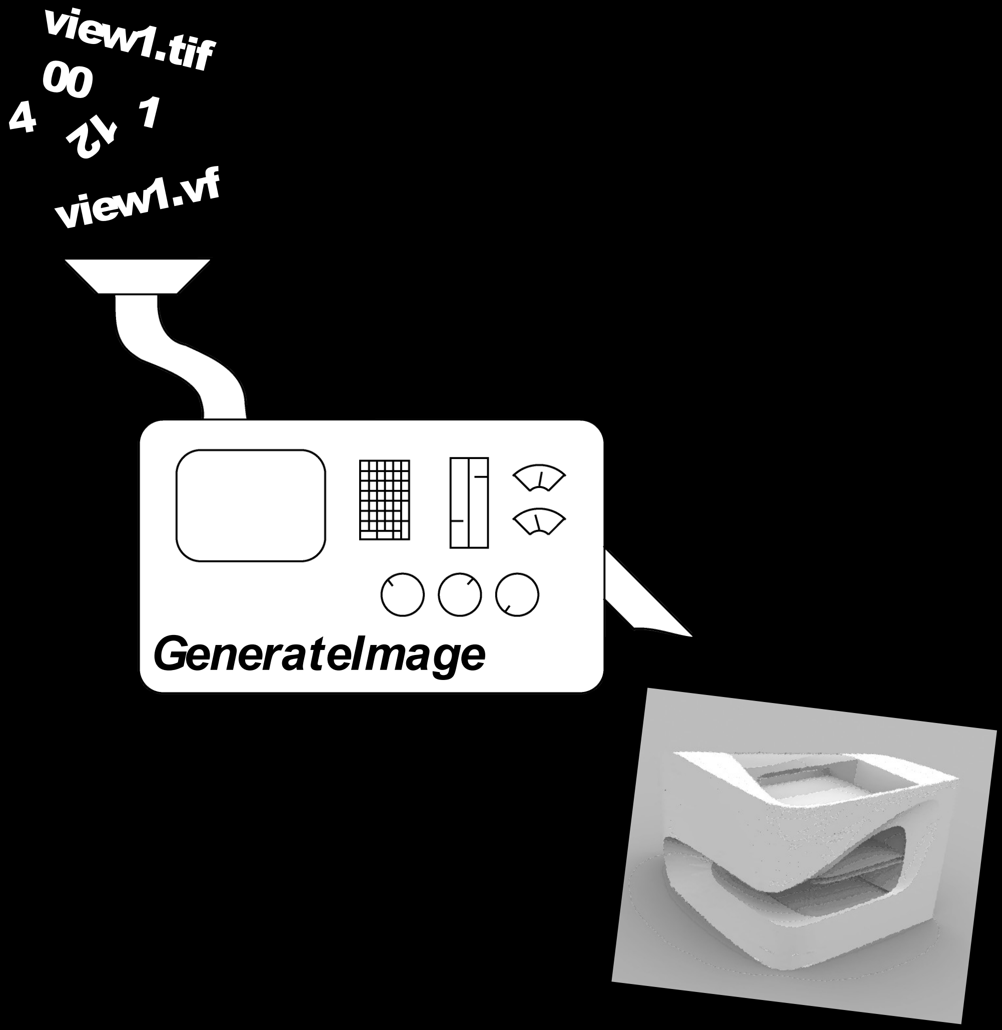
Replacing the variables with values when we call the function…

GenerateImage(‘4’,’1’,’12’,’00’,’view1.vf’,’view1.tif’)

…will replace the corresponding variables in the function. This set of variables happens to have the same result as executing the original bit of code.

```
GenerateImage('4','1','12','00','view1.vf','view1.tif')

    gensky 4 1 12:00 -c -m 75 -o 73.96 -a 40.79 > sky.rad

    oconv sky.rad model.rad > model.oct

    rpict -vf view1.vf @rpict_options.txt model.oct > temp.hdr

    ra_tiff temp.hdr view1.tif
```

It's easy to think of the function as a machine. Put the right things into the machine and the picture you want comes out the other end.



Loops are easy ways to iterate through a range of values. This simple loop prints numbers from 00 to 99.

```
for X=(0 to 9)
  for Y=(0 to 9)
    print "XY"
```

Explanation: This pseudocode starts by looping through the Xs but executing the code below it. The first bit is looping through the Ys, so the Ys get looped through for each X. For each time there's a new Y, it prints "XY": 00, 01, 02, 03, … , 10, 11, 12 … etc. If the last line was print "YX", you'd get 10, 20, 30, … ,11, 21, 31 … etc.

Loops are easy ways to iterate through a range of values. This simple loop prints numbers from 00 to 99.

```
for X=(0 to 9)
  for Y=(0 to 9)
    print "XY"
```

X: 0

Explanation: This pseudocode starts by looping through the Xs but executing the code below it. The first bit is looping through the Ys, so the Ys get looped through for each X. For each time there's a new Y, it prints "XY": 00, 01, 02, 03, … , 10, 11, 12 … etc. If the last line was print "YX", you'd get 10, 20, 30, … ,11, 21, 31 … etc.

Loops are easy ways to iterate through a range of values. This simple loop prints numbers from 00 to 99.

```
for X=(0 to 9)
  for Y=(0 to 9)
    print "XY"
```

X: 0

Y: 0

Explanation: This pseudocode starts by looping through the Xs but executing the code below it. The first bit is looping through the Ys, so the Ys get looped through for each X. For each time there's a new Y, it prints "XY": 00, 01, 02, 03, … , 10, 11, 12 … etc. If the last line was print "YX", you'd get 10, 20, 30, … ,11, 21, 31 … etc.

Loops are easy ways to iterate through a range of values. This simple loop prints numbers from 00 to 99.

```
for X=(0 to 9)
  for Y=(0 to 9)
    print "XY"
```

X: 0
Y: 0
Output: 00

Explanation: This pseudocode starts by looping through the Xs but executing the code below it. The first bit is looping through the Ys, so the Ys get looped through for each X. For each time there's a new Y, it prints "XY": 00, 01, 02, 03, … , 10, 11, 12 … etc. If the last line was print "YX", you'd get 10, 20, 30, … ,11, 21, 31 … etc.

Loops are easy ways to iterate through a range of values. This simple loop prints numbers from 00 to 99.

```
for X=(0 to 9)
  for Y=(0 to 9)
    print "XY"
```

X: 0
Y: 1
Output: 01

Explanation: This pseudocode starts by looping through the Xs but executing the code below it. The first bit is looping through the Ys, so the Ys get looped through for each X. For each time there's a new Y, it prints "XY": 00, 01, 02, 03, … , 10, 11, 12 … etc. If the last line was print "YX", you'd get 10, 20, 30, … ,11, 21, 31 … etc.

Loops are easy ways to iterate through a range of values. This simple loop prints numbers from 00 to 99.

```
for X=(0 to 9)
  for Y=(0 to 9)
    print "XY"
```

X: 0
Y: 2
Output: 02

Explanation: This pseudocode starts by looping through the Xs but executing the code below it. The first bit is looping through the Ys, so the Ys get looped through for each X. For each time there's a new Y, it prints "XY": 00, 01, 02, 03, … , 10, 11, 12 … etc. If the last line was print "YX", you'd get 10, 20, 30, … ,11, 21, 31 … etc.

Loops are easy ways to iterate through a range of values. This simple loop prints numbers from 00 to 99.

```
for X=(0 to 9)
  for Y=(0 to 9)
    print "XY"
```

X: 0
Y: 3
Output: 03

Explanation: This pseudocode starts by looping through the Xs but executing the code below it. The first bit is looping through the Ys, so the Ys get looped through for each X. For each time there's a new Y, it prints "XY": 00, 01, 02, 03, … , 10, 11, 12 … etc. If the last line was print "YX", you'd get 10, 20, 30, … ,11, 21, 31 … etc.

Loops are easy ways to iterate through a range of values. This will make an image every 5 minutes from 0500 to 2200 on 21 June, 21 September, and 21 December. The output filename is the time that the image represents.

```
for M=(6 to 12, multiples of 3)
  for D=(21)
    for HH=(5 to 22)
      for MM=(0 to 55, multiples of 5)
        GenerateImage(M,D,HH,MM,”view1.vf”,”M-D-HH-MM.tif”)
```

Important notes: This doesn’t check to see if the sun is above the horizon. Also, this can be very time consuming.

A few small changes and the image names can become a sequence. This can be important to other functions or programs.

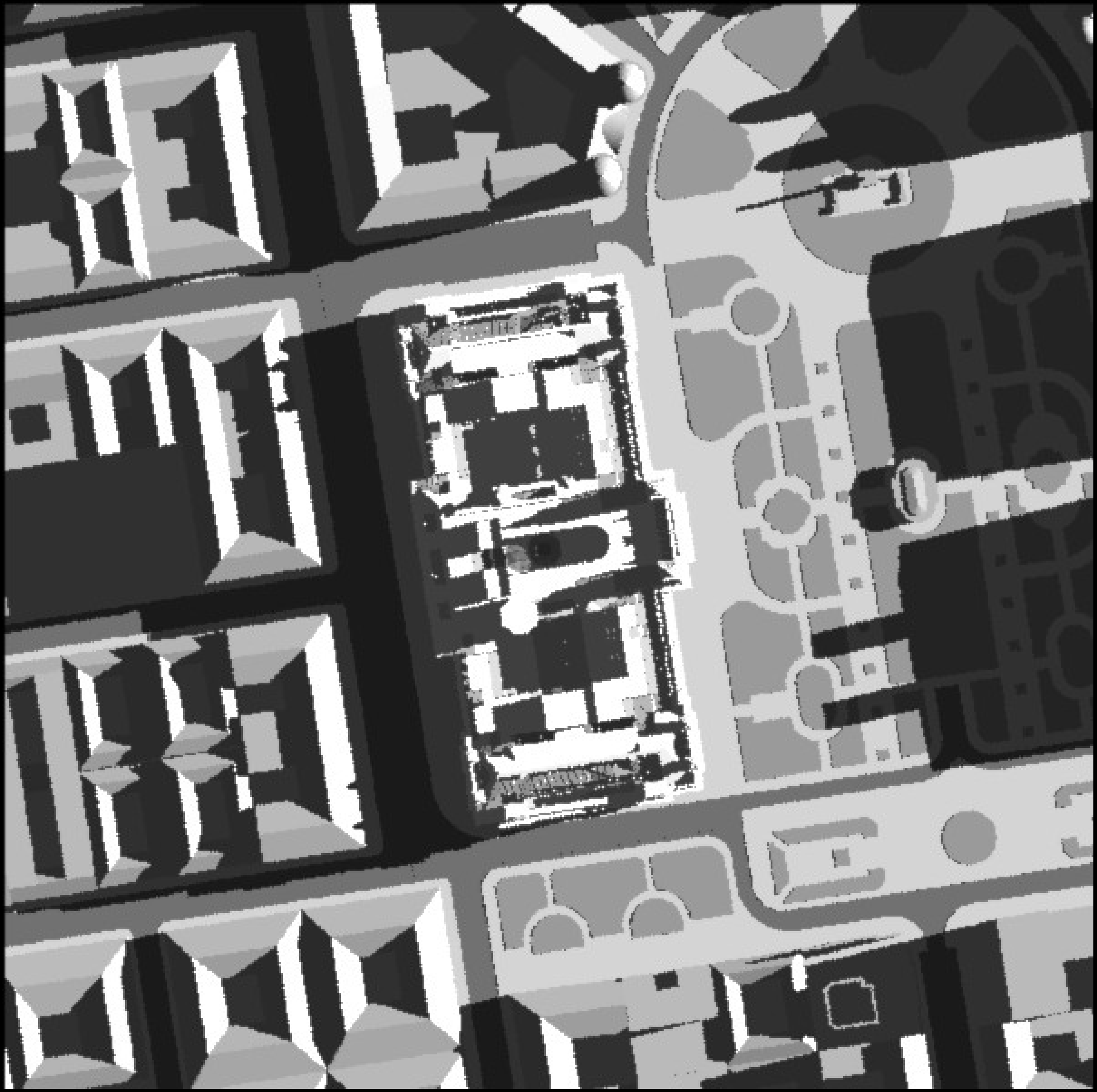
```
X=0
for M=(6 to 12, multiples of 3)
  for D=(21)
    for HH=(5 to 22)
      for MM=(0 to 55, multiples of 5)
        GenerateImage(M,D,HH,MM,"view1.vf","X.tif")
        X=X+1
```

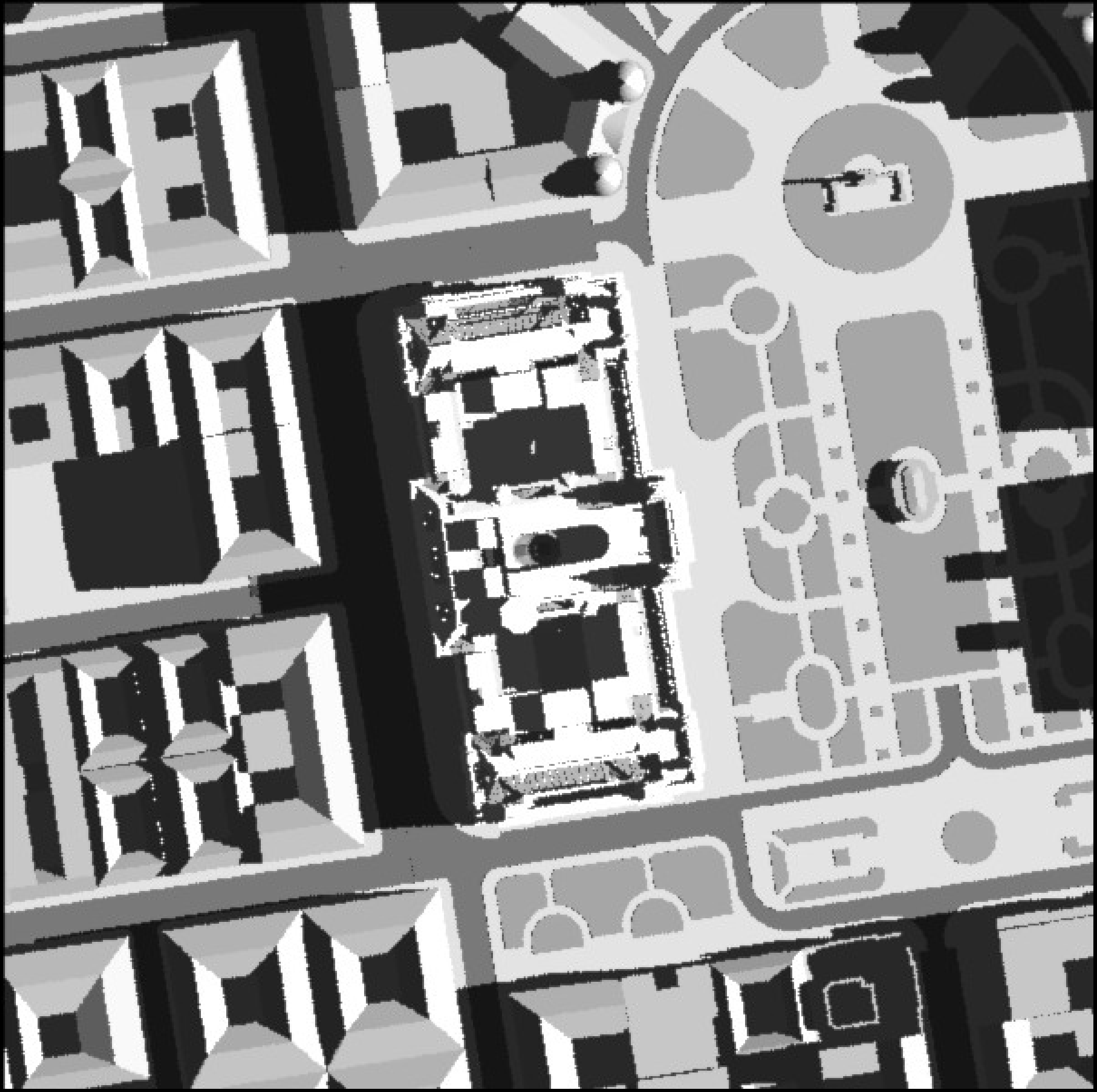
Note: A sequence is easily machine readable because it is a series of consecutive numbers. The time sequence is also machine readable, but it may need to be “told” what each number represents. This also doesn’t check to see if the sun is above the horizon.

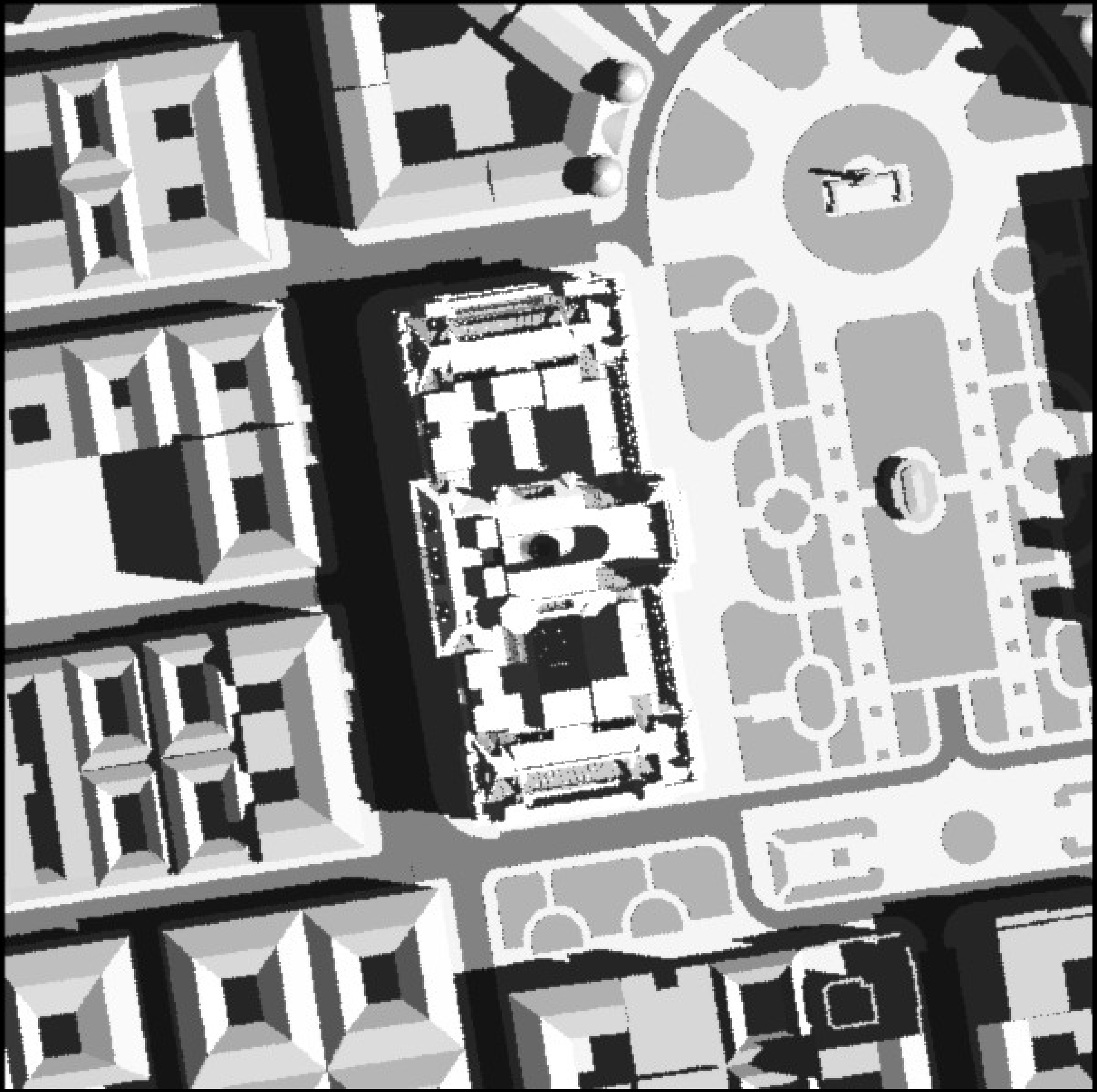
Some Examples

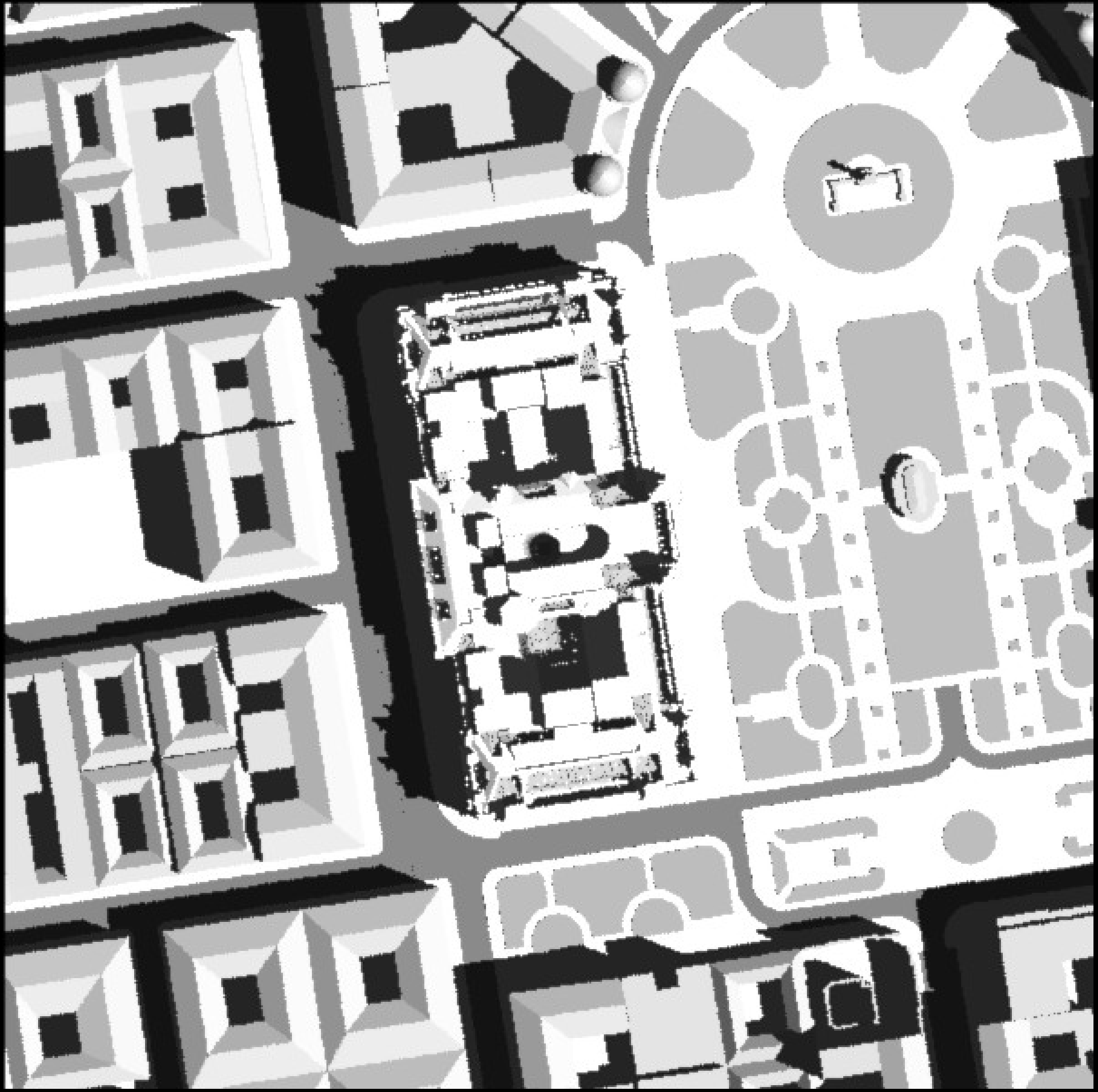
What box?

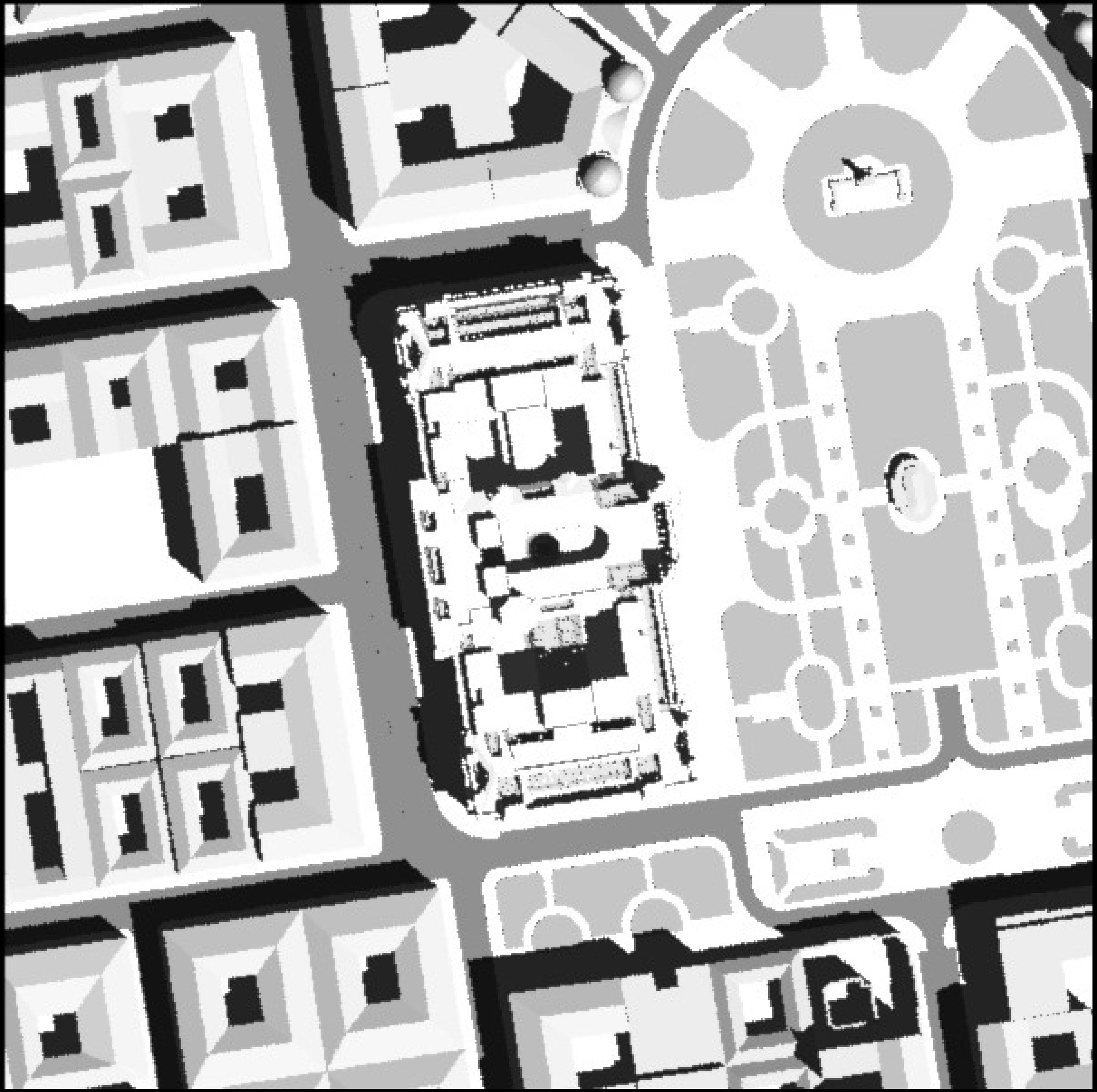
1: A series of images to manually step through.

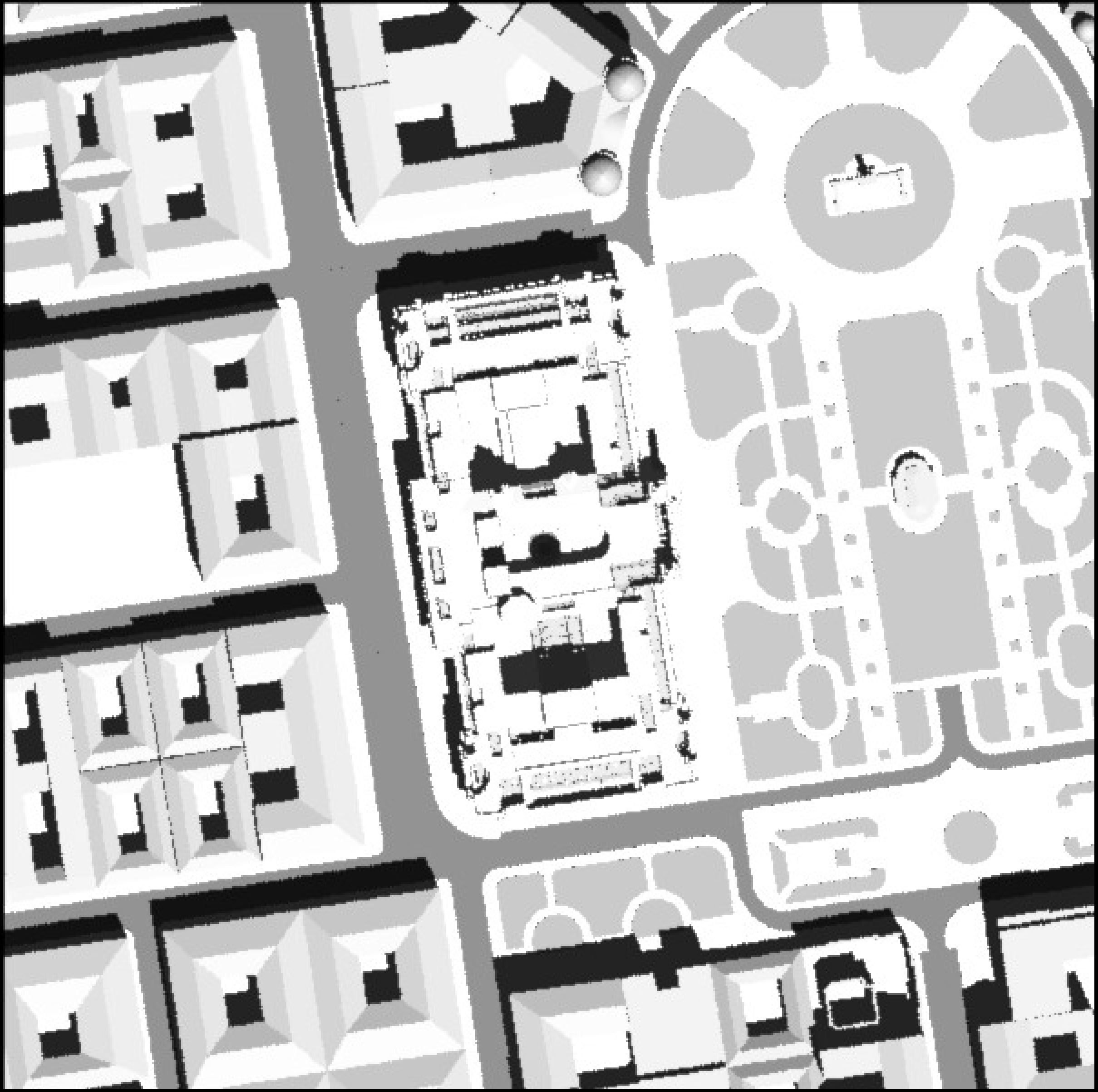


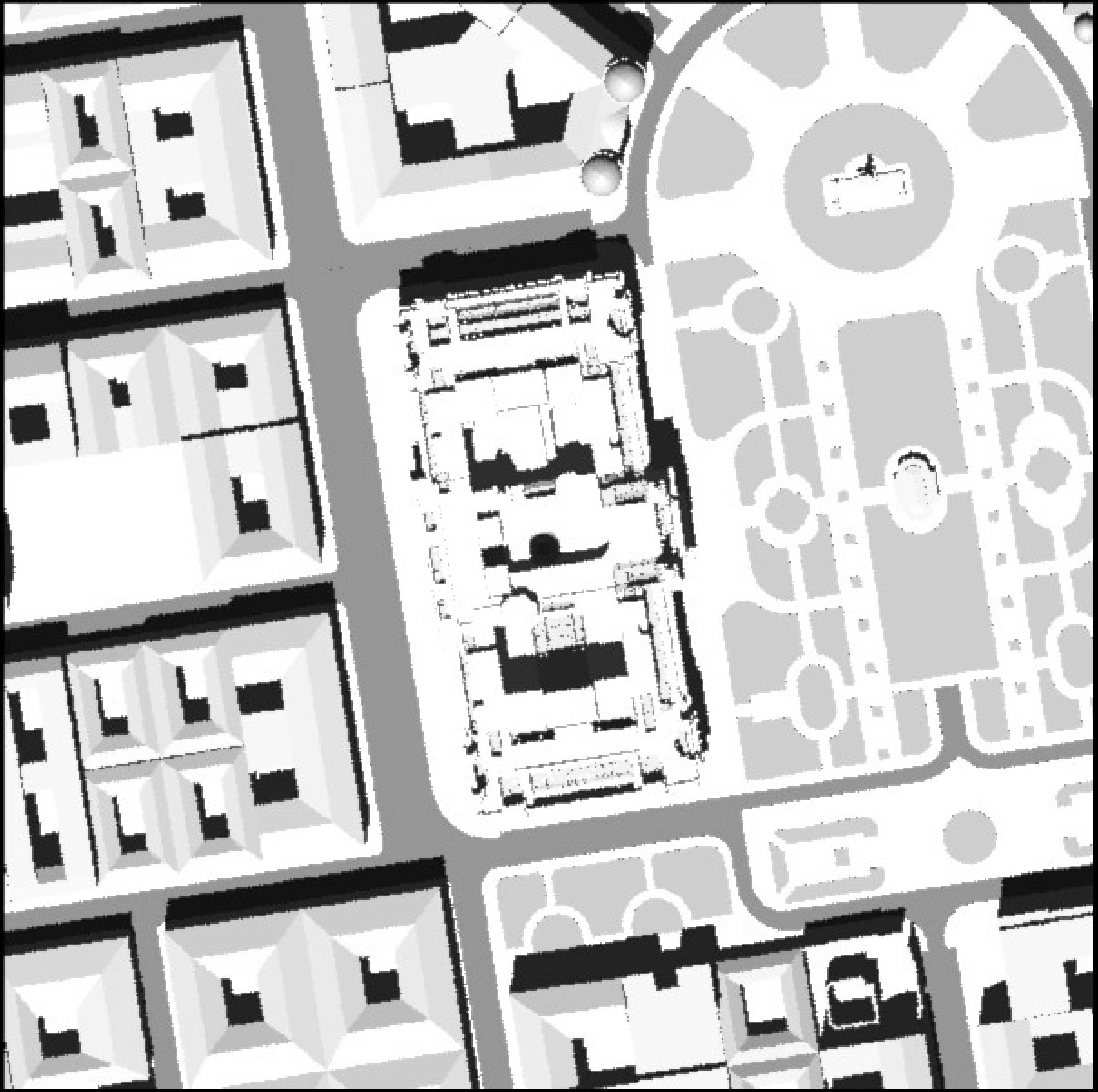


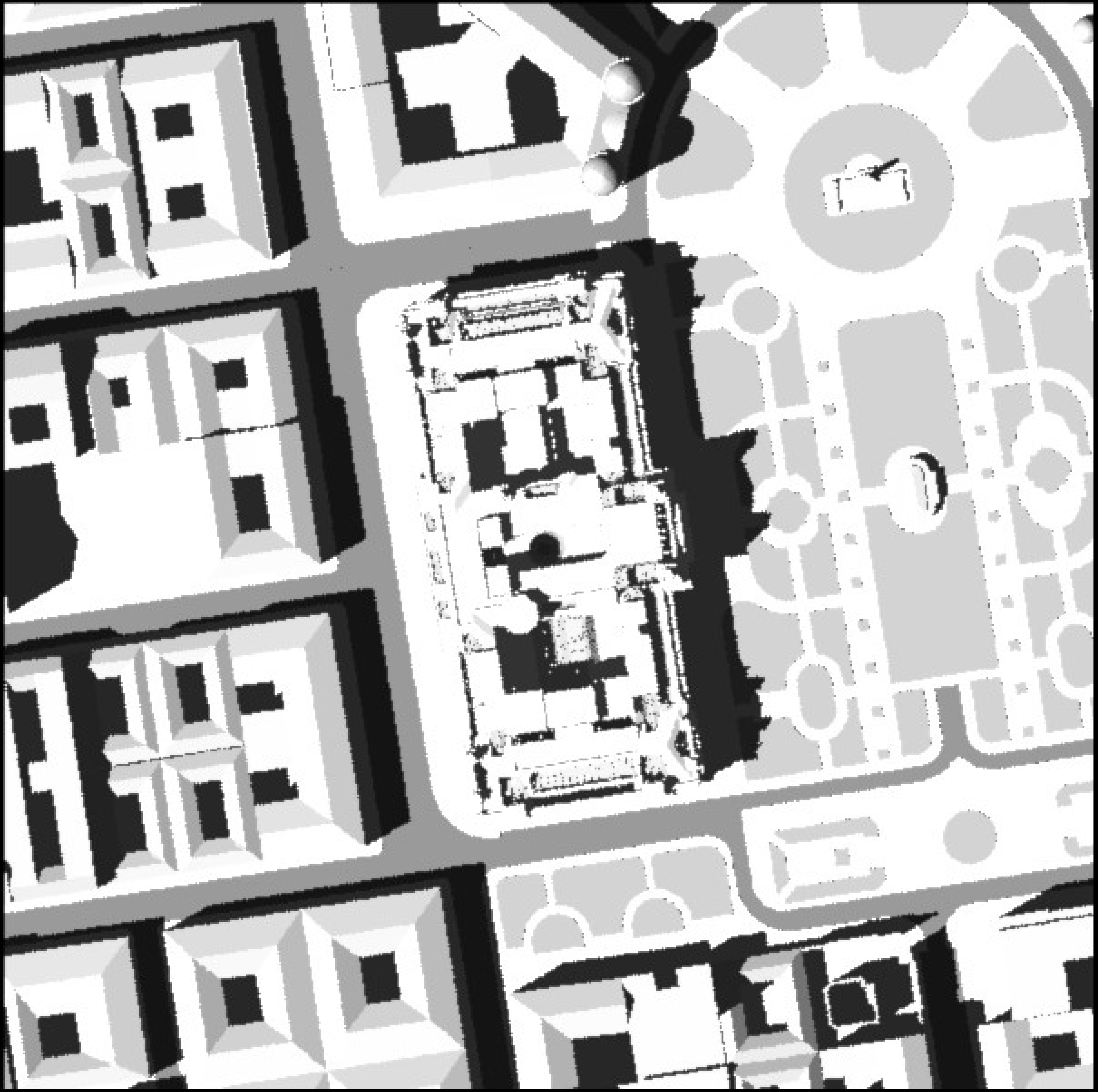


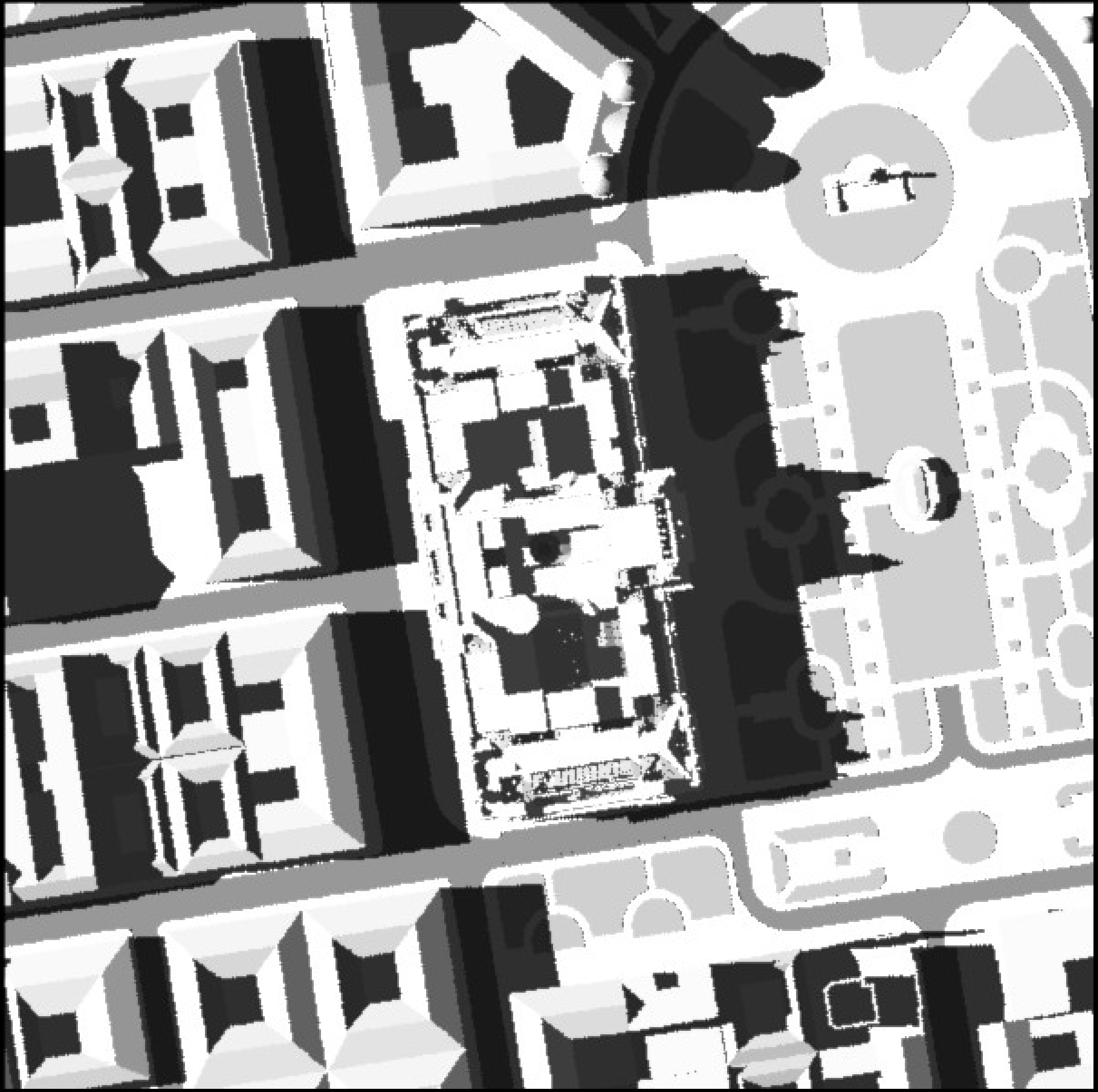


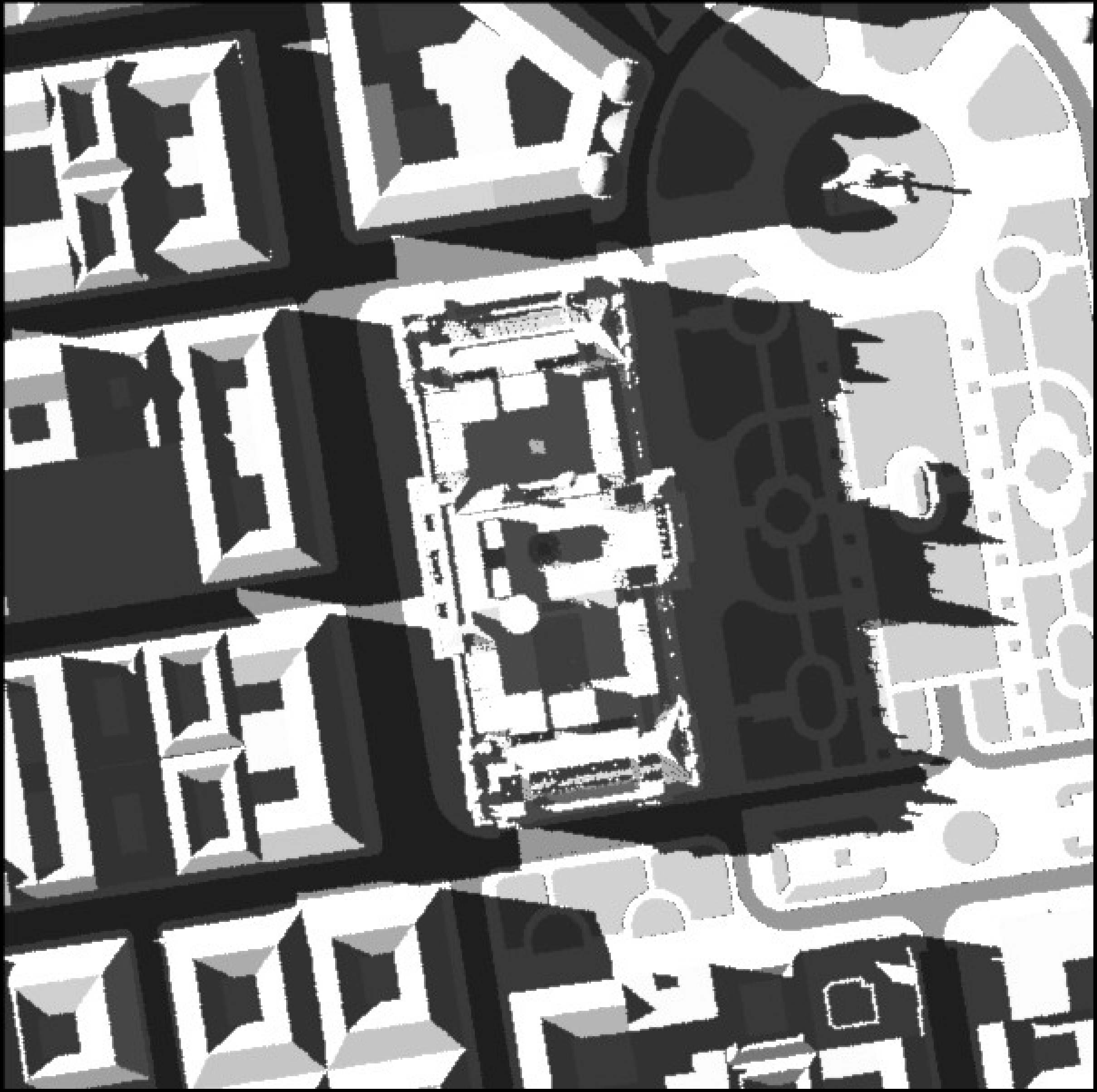


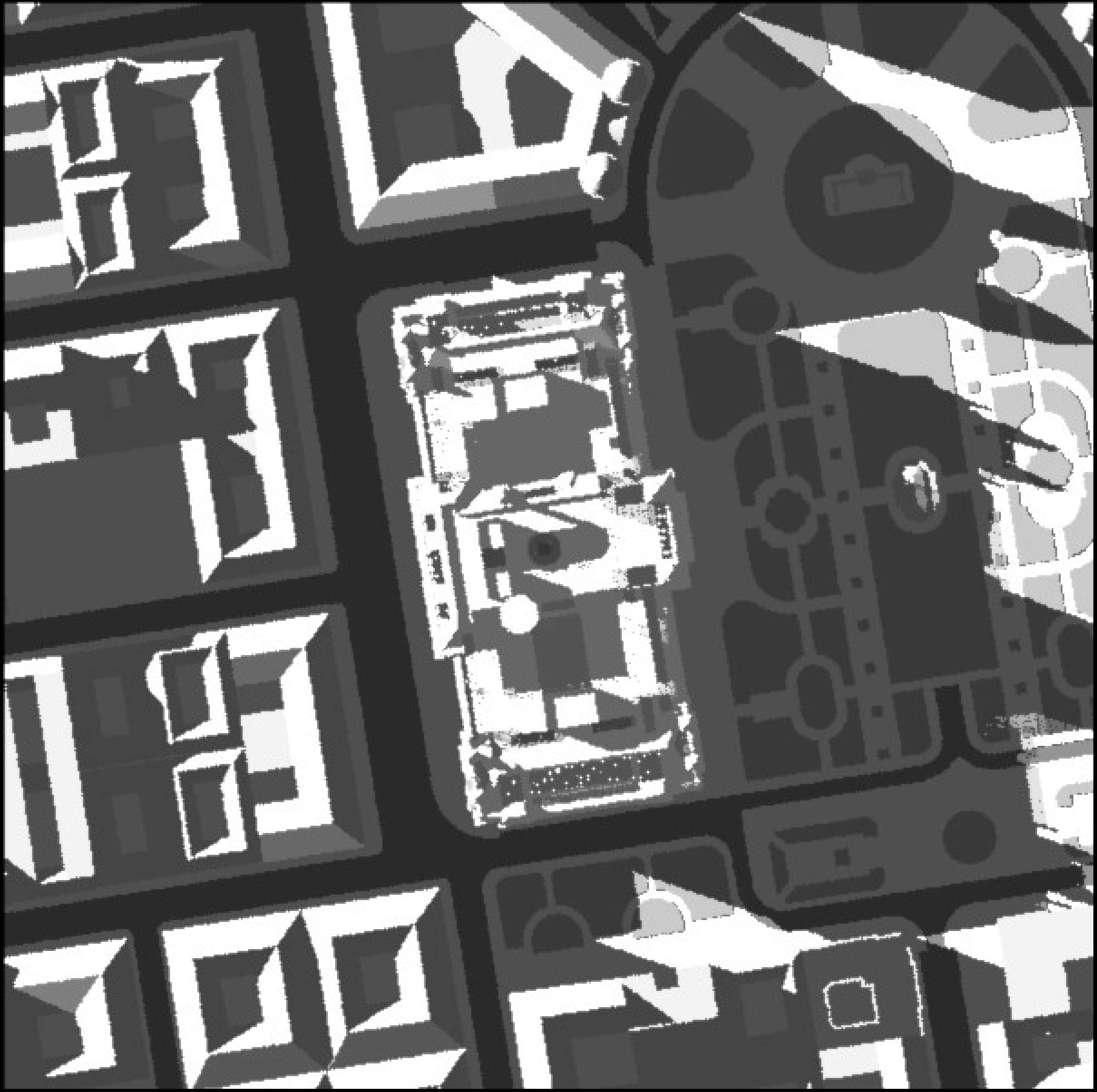












2: A set of images to be explored manually

This is especially useful for any set of images that isn't in a sequence – if it makes logical sense to be able to go from any single image to another.

Presentation methods include PDF, Impress, PowerPoint, Flash, and web pages, all you need is a hyperlink to jump between images.

3: A series of images to be played automatically (an animation).

First, we generate a sequence of images, which can be very time consuming.

```
X=0
for M=(6 to 12, multiples of 3)
  for D=(21)
    for HH=(5 to 22)
      for MM=(0 to 55, multiples of 5)
        GenerateImage(M,D,HH,MM,"view1.vf","X.tif")
        X=X+1
```

First, we generate a sequence of images, which can be very time consuming.

```
X=0
for M=(6 to 12, multiples of 3)
  for D=(21)
    for HH=(5 to 22)
      for MM=(0 to 55, multiples of 5)
        GenerateImage(M,D,HH,MM,"view1.vf","X.tif")
        X=X+1
```

Then we stitch them together in an animation.

Easier done than said thanks to ffmpeg, a cross-platform video encoder.

Note: See http://ffmpeg.org

Easier done than said thanks to ffmpeg, a cross-platform video encoder.

```
ffmpeg -sameq -i INPUT OUTPUT
```

Note: -sameq tries to match output quality based on input quality.

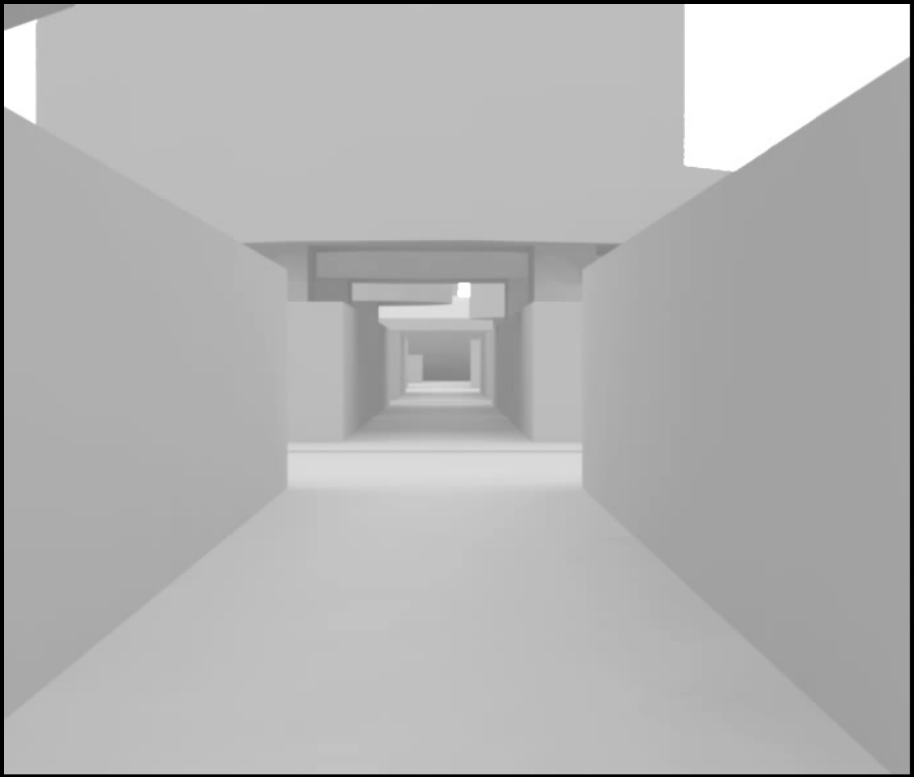
Our images are named 1.tif, 2.tif, 3.tif, etc., which is a machine readable sequence of consecutive numbers.

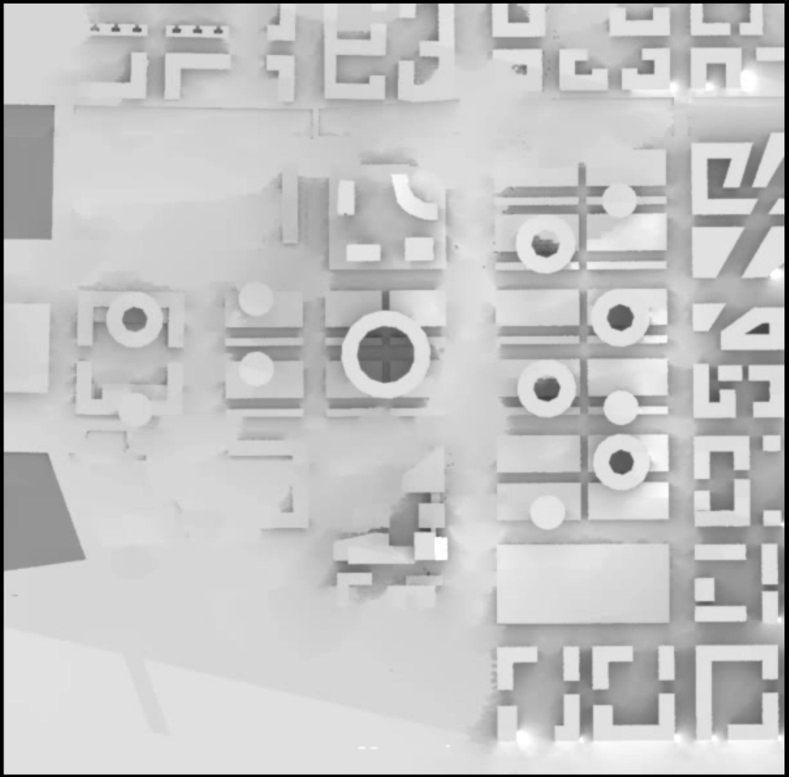
```
ffmpeg -sameq -i %d.tif OUTPUT
```

Note: ffmpeg follows the vprintf scheme for the d format string. For example, if you had images called 00001.tif, 00002.tif, 00003.tif, etc., the format string would be %05d.tif. For further information, see http://linux.die.net/man/3/vfprintf

Finally, tell ffmpeg what video file type (container) to use.

```
ffmpeg -sameq -i %d.tif movie.mov
                              .mpg
                              .avi
                              .wmv
                              .mkv
                              .divx
                              .flv
                              .swf
                              .asf
                              etc.
```


View in Pedestrian Corridor


Plan View

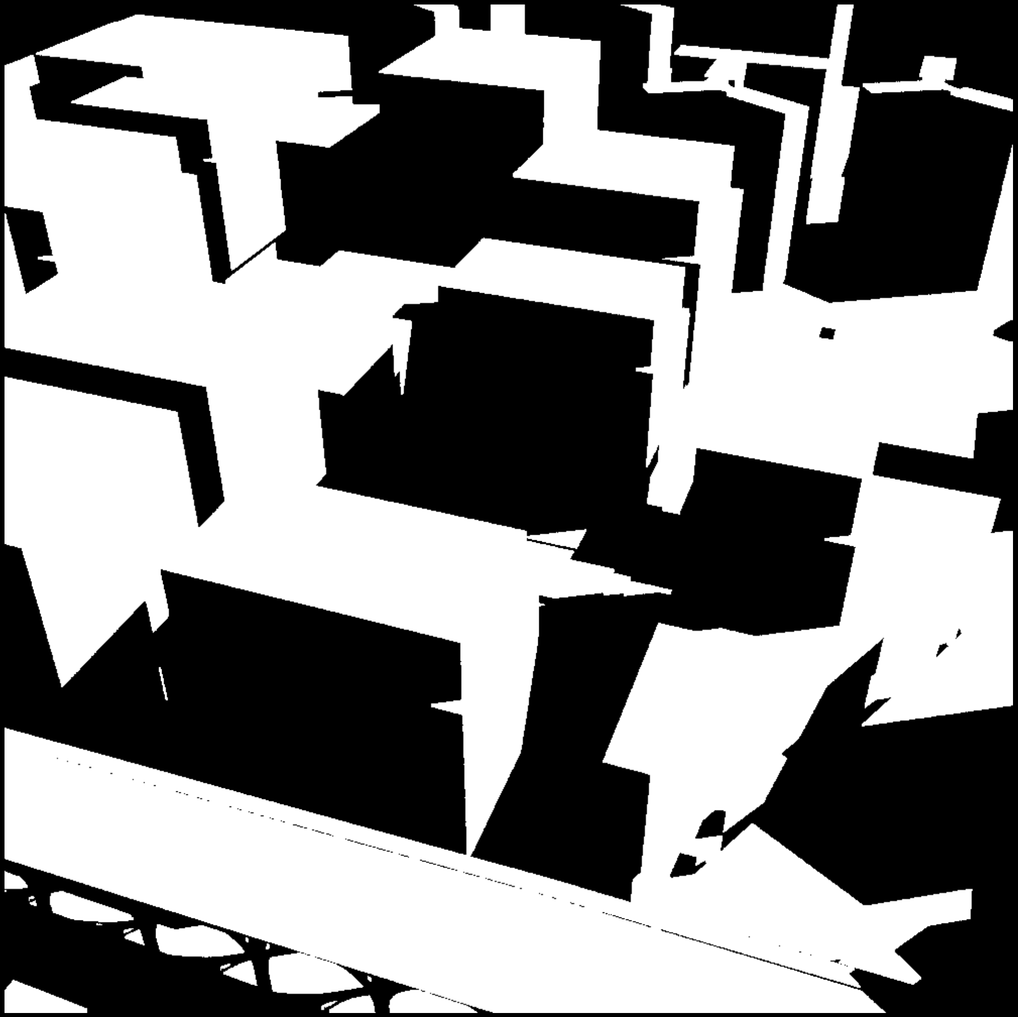
"Galaxy" Option - twofour54° - 6/21 @ 05:10

Buro Happold Lighting

4: A simple shadow study animation.

Since in shadow studies we don't mind that we're not getting proper illuminances everywhere, why don't we just model the shadows?

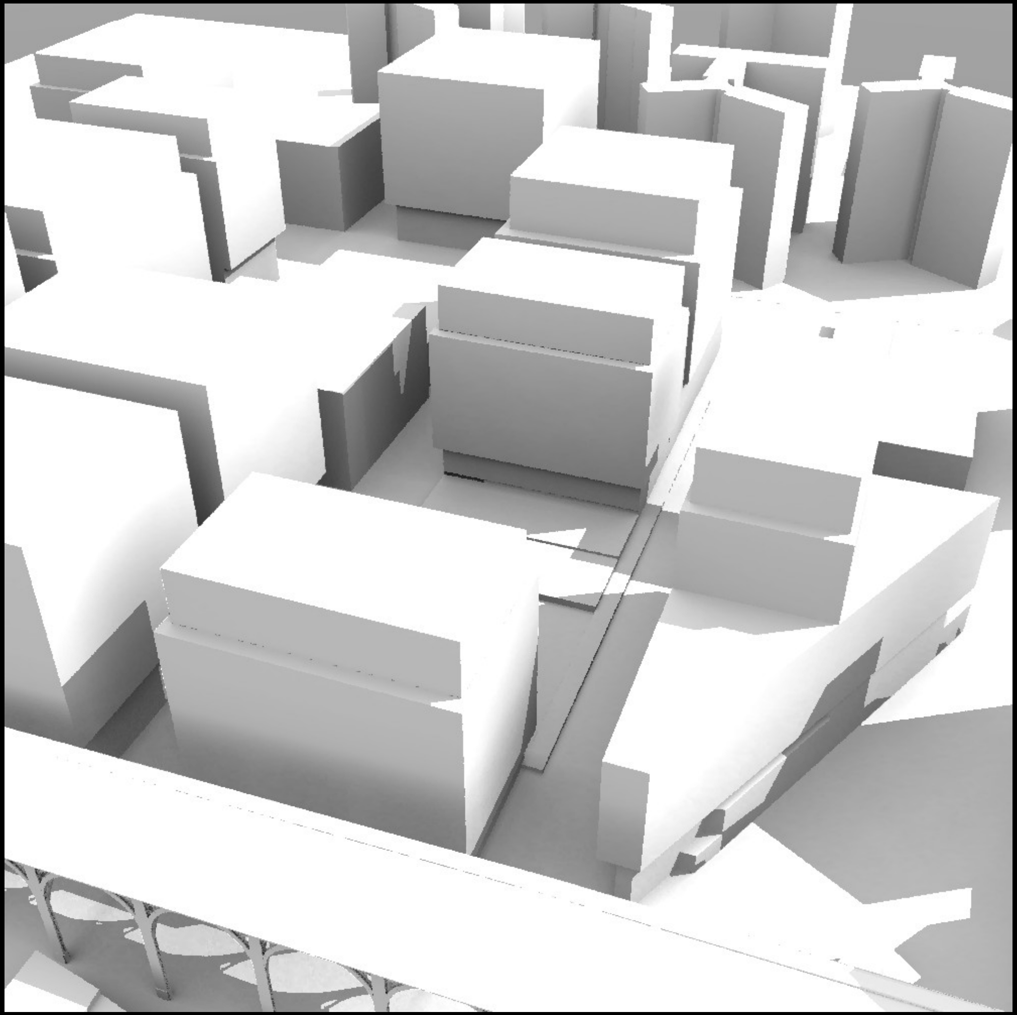
On its own, it's an abstract image without any context. But it renders very quickly.



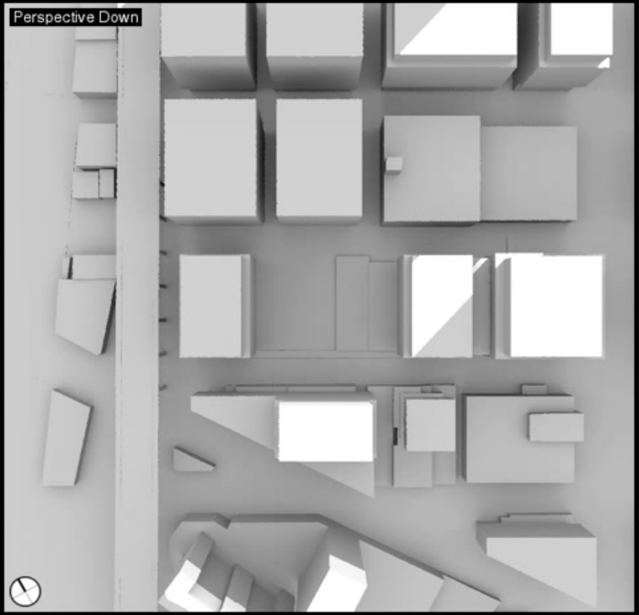
Relevant *rpict* options: -i -ab 0 -av 0 0 0 -ps 1

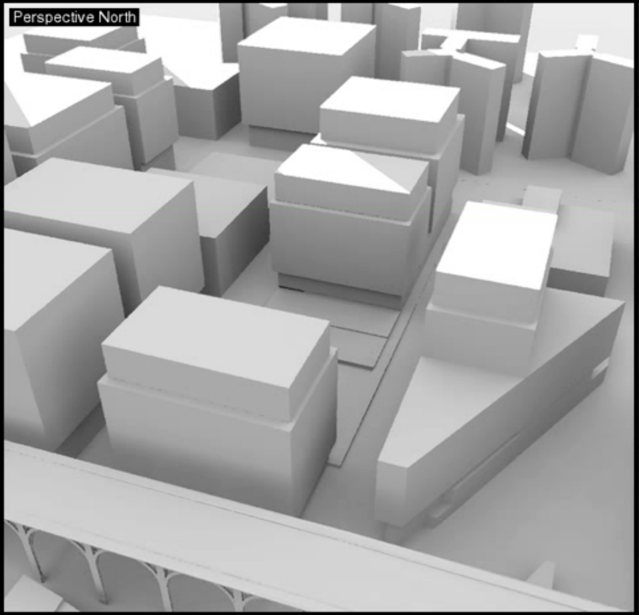
So we'll overlay it on a background image, like this one of the model under a uniform sky.

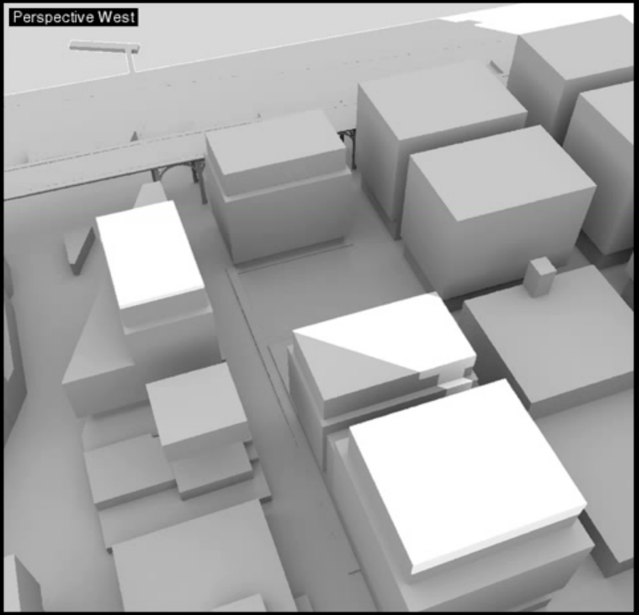
You can do a linear or non-linear combination with *pcomb*, or use photo manipulation toolkits like PIL or ImageMagick.



Note: See the *pcomb* -e option for non-linear combinations.







Columbia Business School - 21 June, 04:50

Buro Happold Lighting

5: An animation from the sun's perspective.

The vector to the sun is part of the output from *gensky*.

```
#gensky 4 1 12
#Local solar time: 11.80
#Solar altitude and azimuth: 56.1 -5.4
#Ground ambient level: 18.8

void light solar
0
0
3 6.99e+006 6.99e+006 6.99e+006

solar source sun
0
0
4 0.052926 -0.555878 0.829577 0.5

void brightfunc skyfunc
2 skybr skybright.cal
0
7 1 1.63e+001 2.59e+001 7.97e-001 0.052926 -0.555878 0.829577
```

First, make sure you're using the parallel projection view type.

From *gensky*: 0.052926 -0.555878 0.829577

View parameters:
-vtl

Next, multiply each component by a large number, say 10000, and add the x, y, and z of the camera target (say, 50, 150, 10) to get the view point.

From *gensky*: 0.052926 -0.555878 0.829577

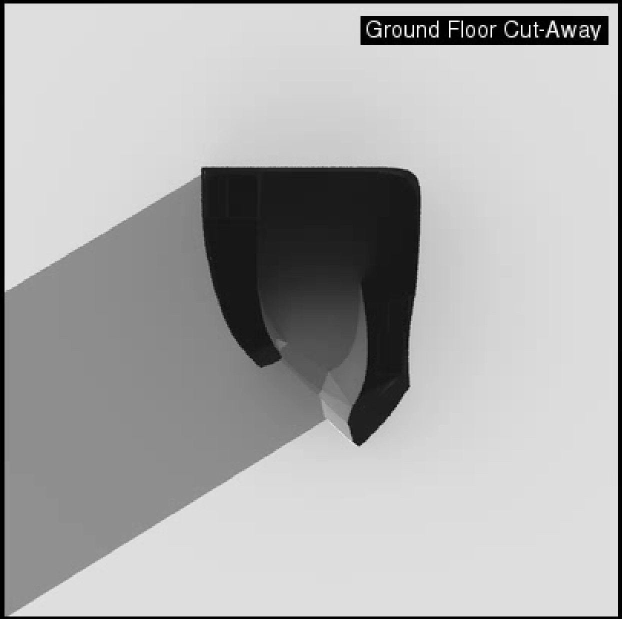
View parameters:
-vtl -vp 579.3 5708.8 8305.8

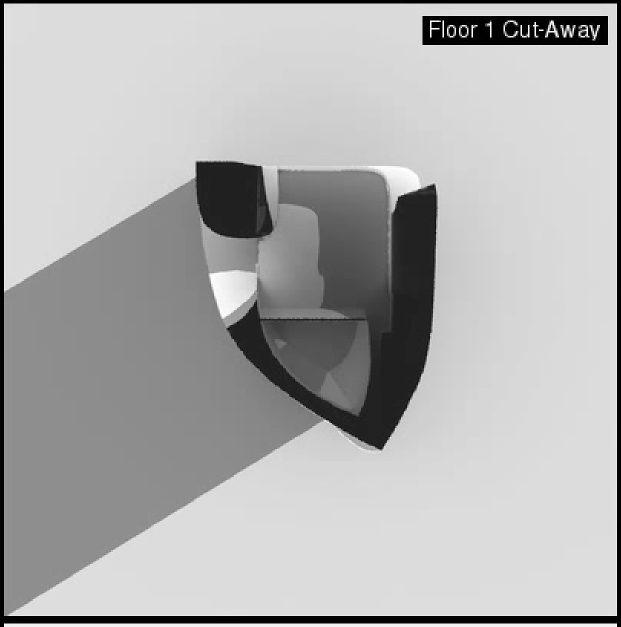
Then multiply each component by -1 to get the view vector from the sun.

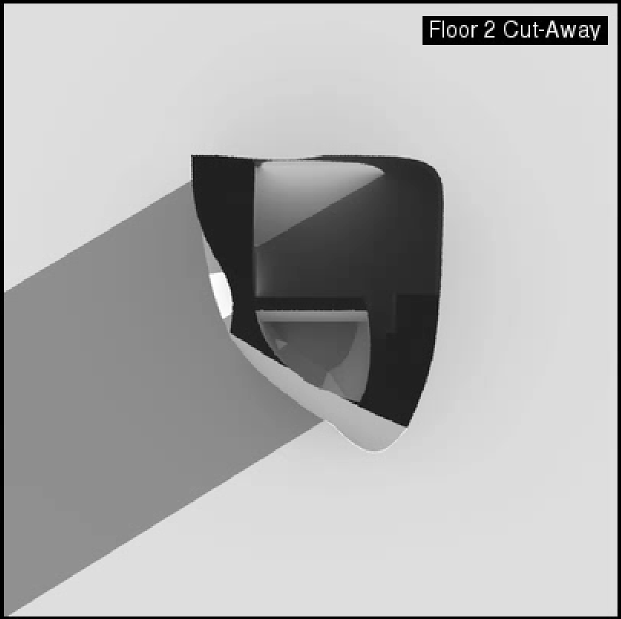
From *gensky*: 0.052926 -0.555878 0.829577

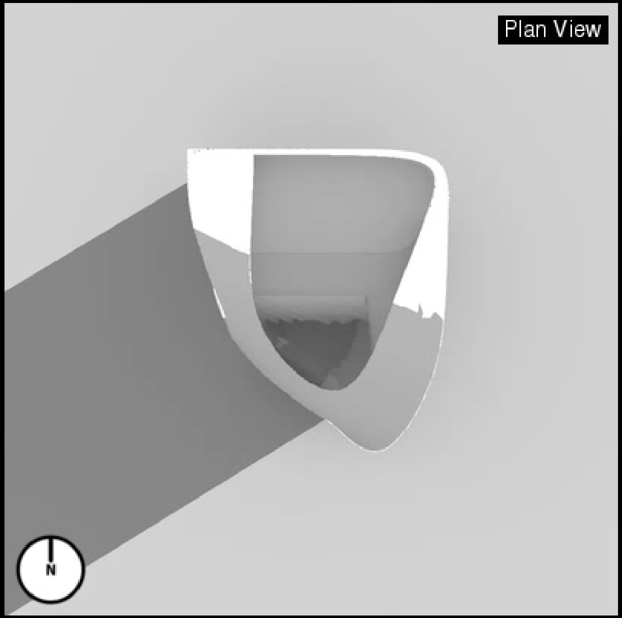
View parameters:

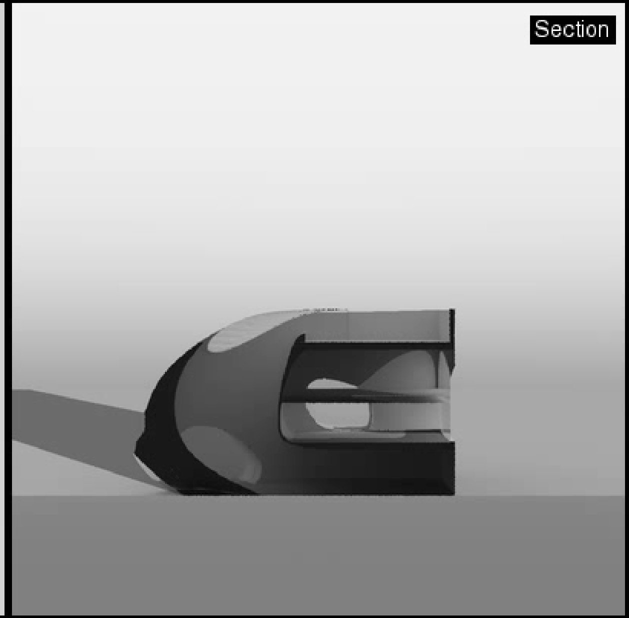
```
-vtl -vp 579.3 5708.8 8305.8 -vd -0.052926 0.555878 -0.829577
```

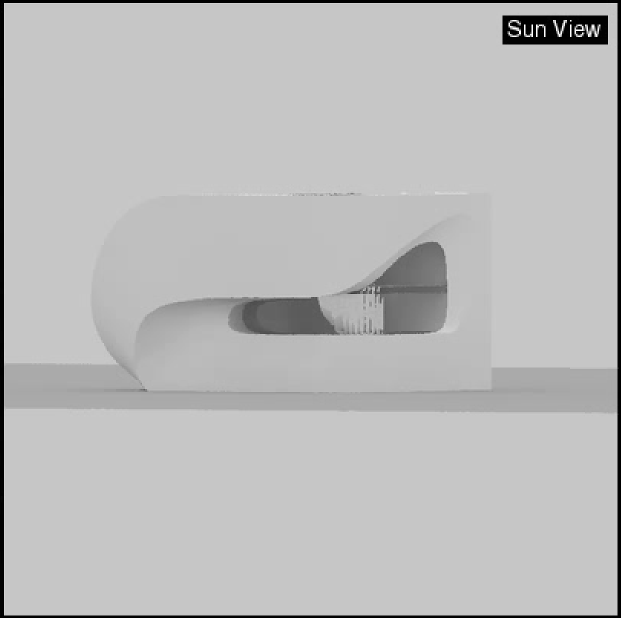












Sheldon Art Museum - 21 June, 05:05

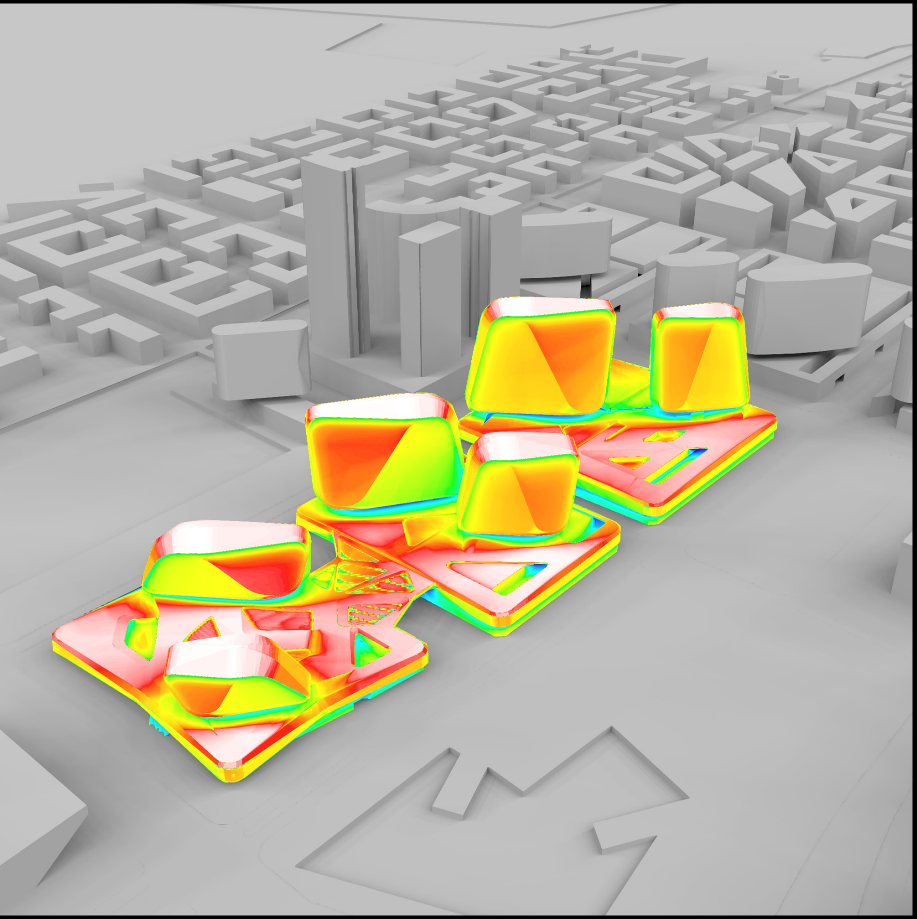
Buro Happold Lighting

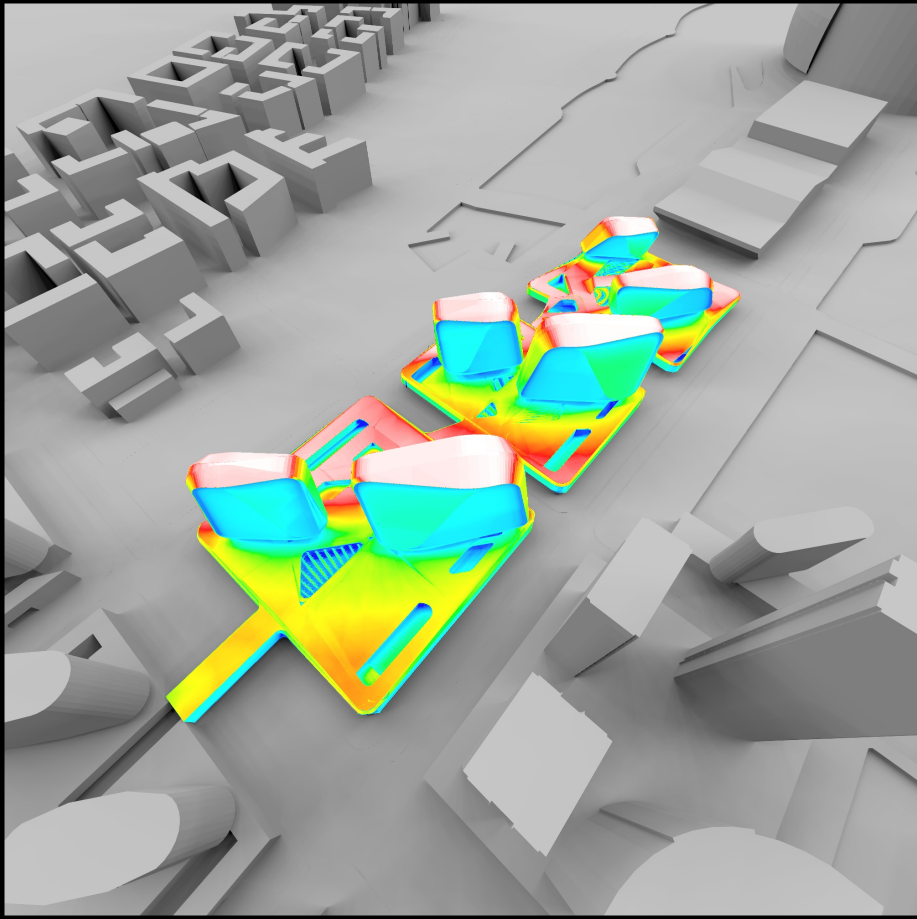
6: Quantifying solar exposure.

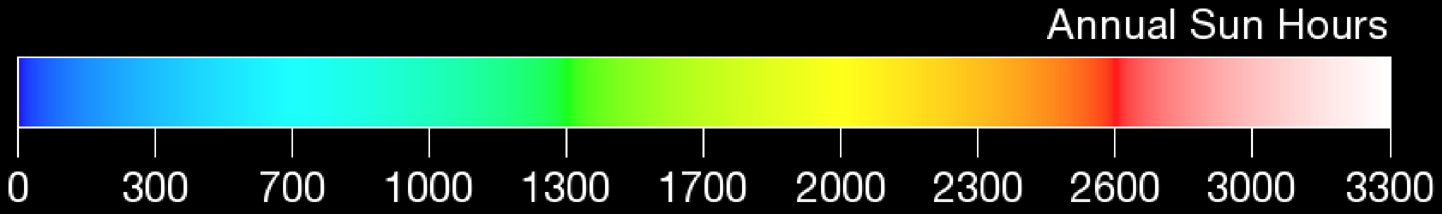
Since we can easily calculate the sun/shadow plot, why not calculate it for an entire year?

For every daylight hour, create a sun/shadow rendering, then add them together with *pcomb*.

This is useful for aligning solar shading efforts to the regions that would benefit the most.

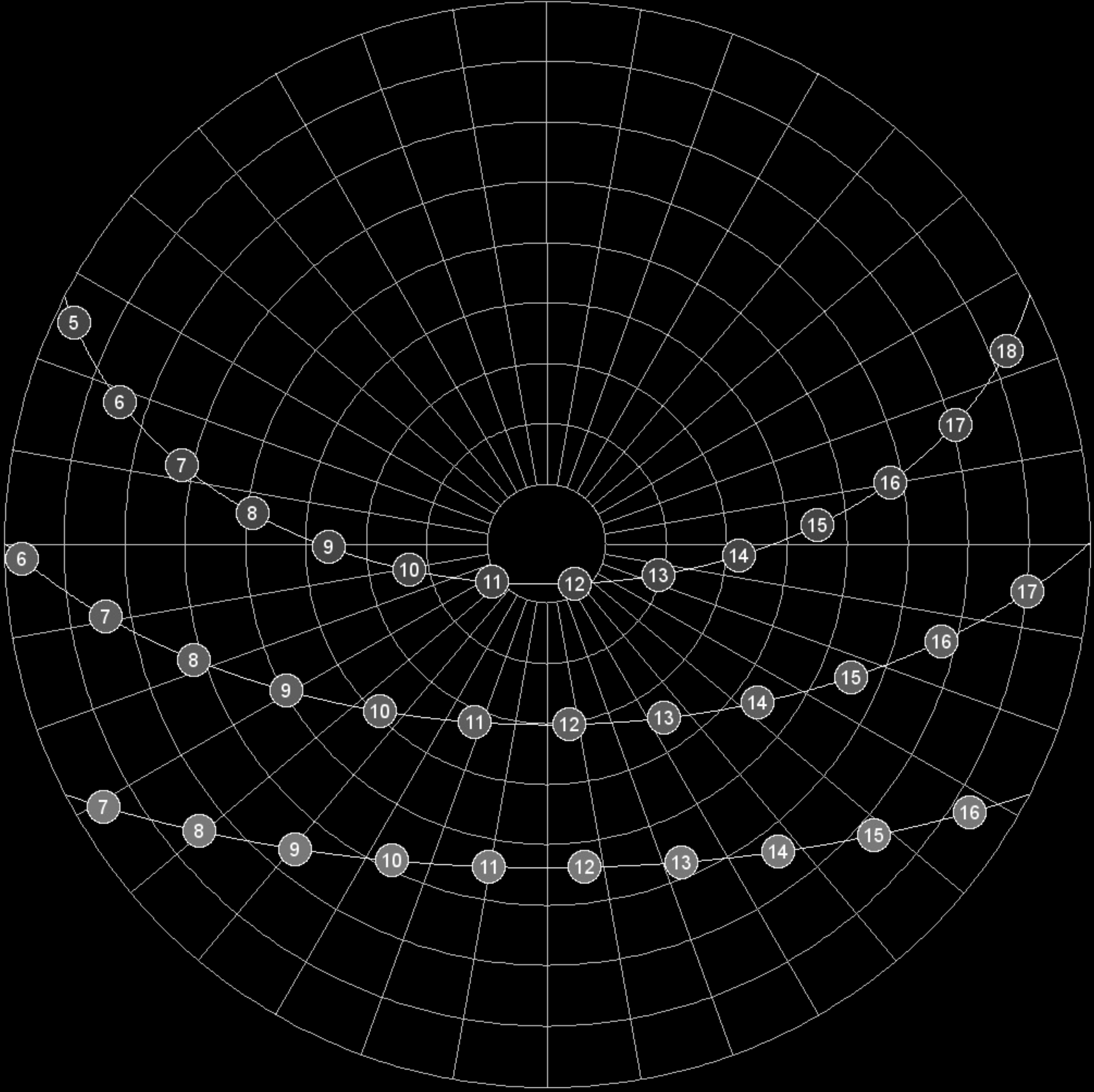






Note: This isn't the Radiance false color scale, just a Photoshop Gradient Map.

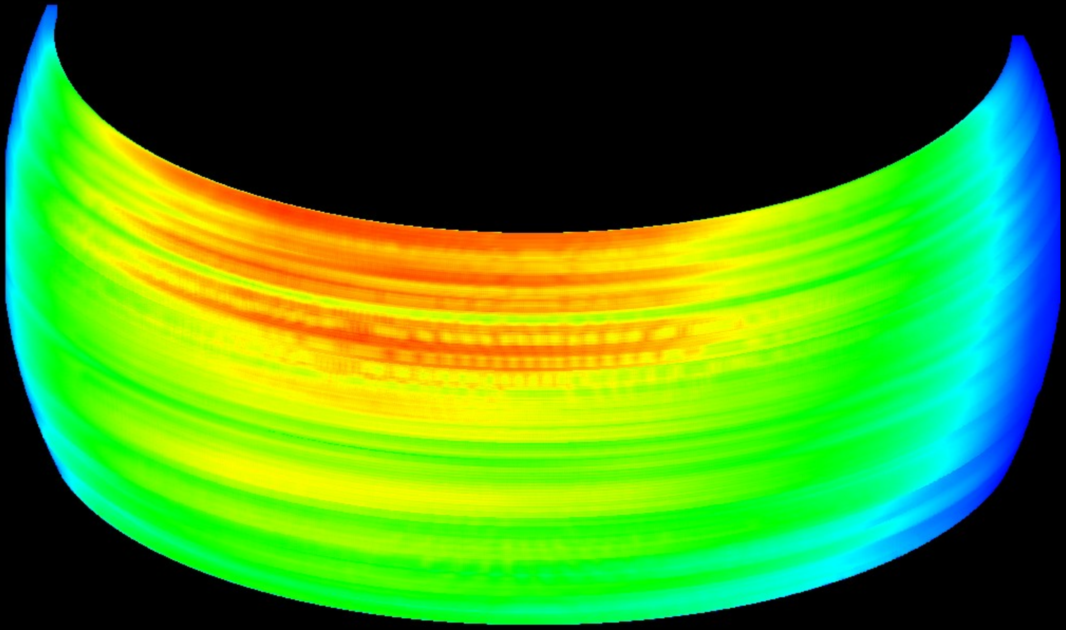
7: Solar analemma from *gensky* output.



Note: This is the view from laying on the ground and looking up at the sky, with your head to the North. See http://en.wikipedia.org/wiki/Spherical_coordinate_system#Coordinate_system_conversions for conversions.

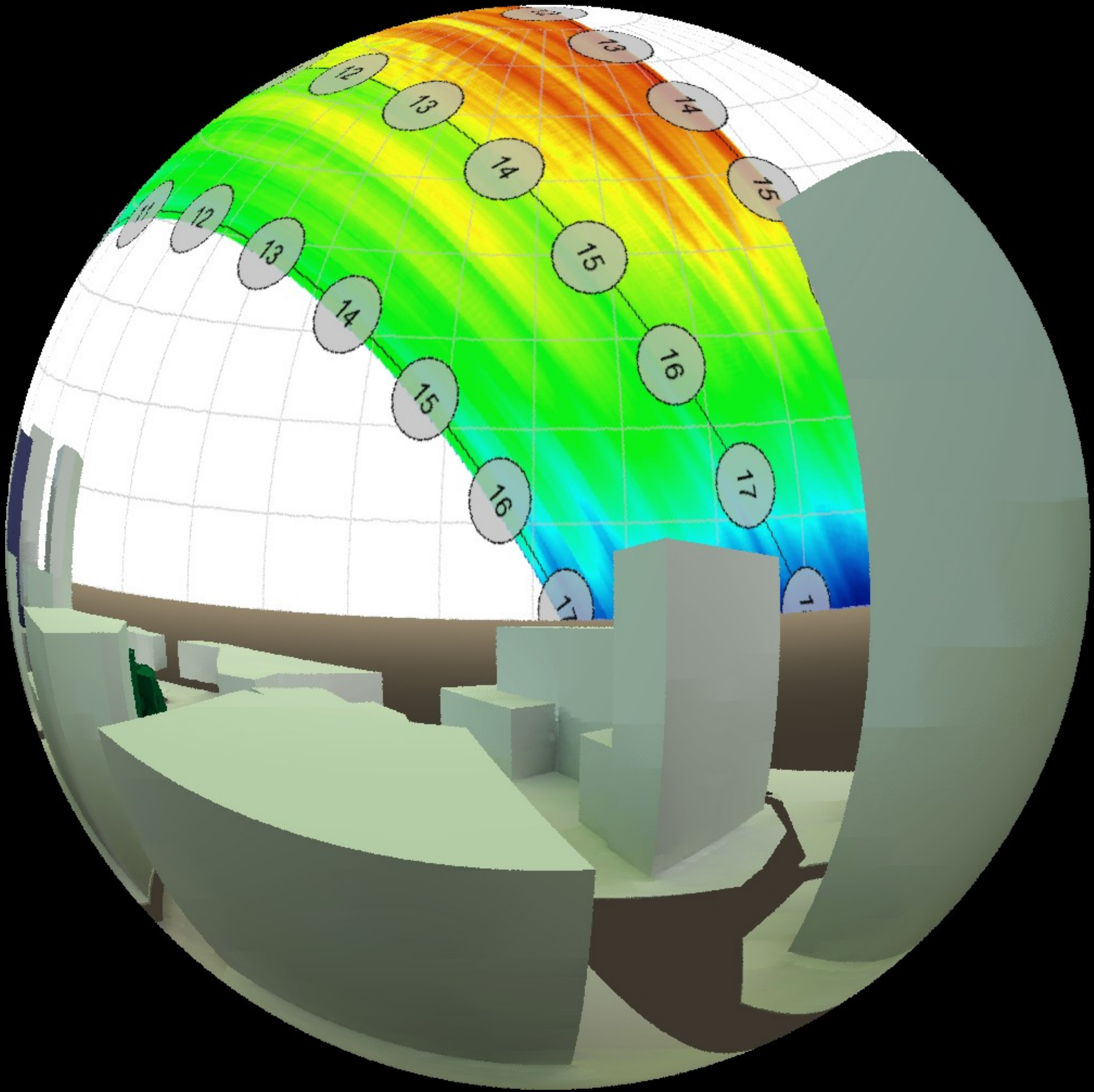
8: Solar analemma with weather information.

Instead of plotting a point at the sun's position, you can plot information from a weather file.



Note: See http://apps1.eere.energy.gov/buildings/energyplus/weatherdata_format_def.cfm for weather file format definitions.

9: Map the analemma to the sky dome.

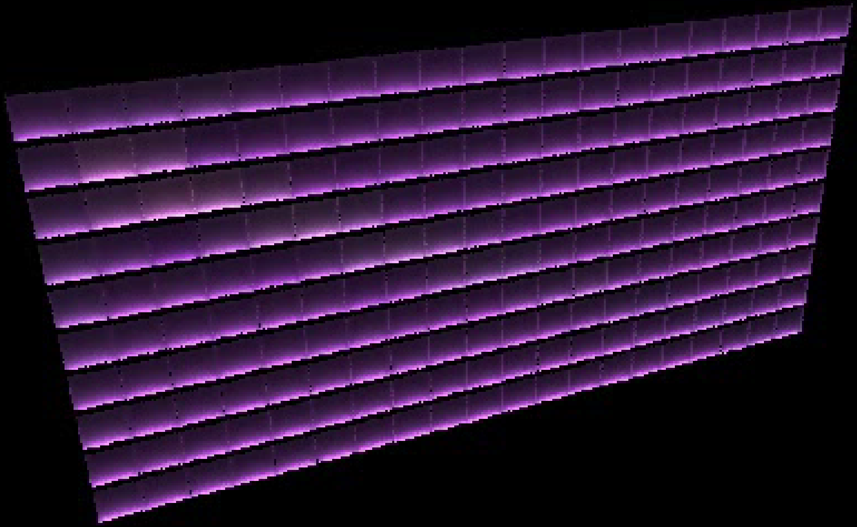


12
13
14
15
16
17
11
12
13
14
15
16
17
13
14
15
17

10: Adding in electric light.

Bonus: Just for fun...

Radiance Rendering



Original Animation

Media Facade Proof of Concept

Buro Happold Lighting

The Paperless Office

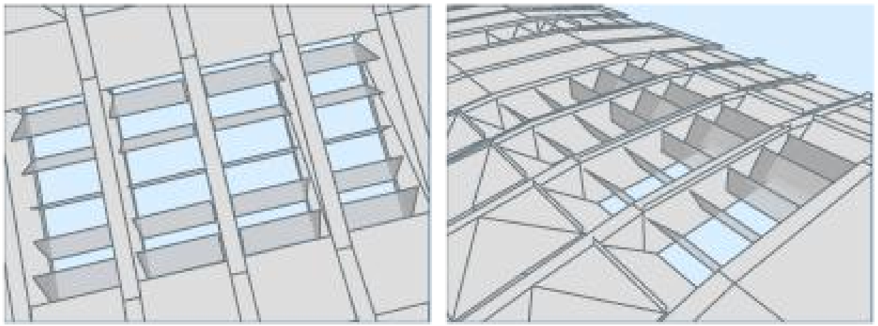
We’re not quite there yet.

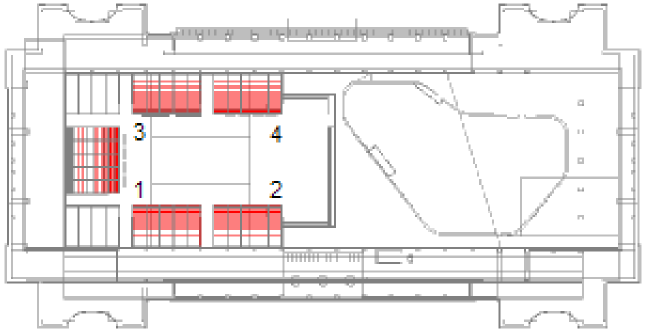
Putting it on Paper

It’s called “Paper Trail” for a reason. It's the contractual obligation to deliver static, hard-bound information.

Putting it on paper: Example

COSA SOLUTIONS





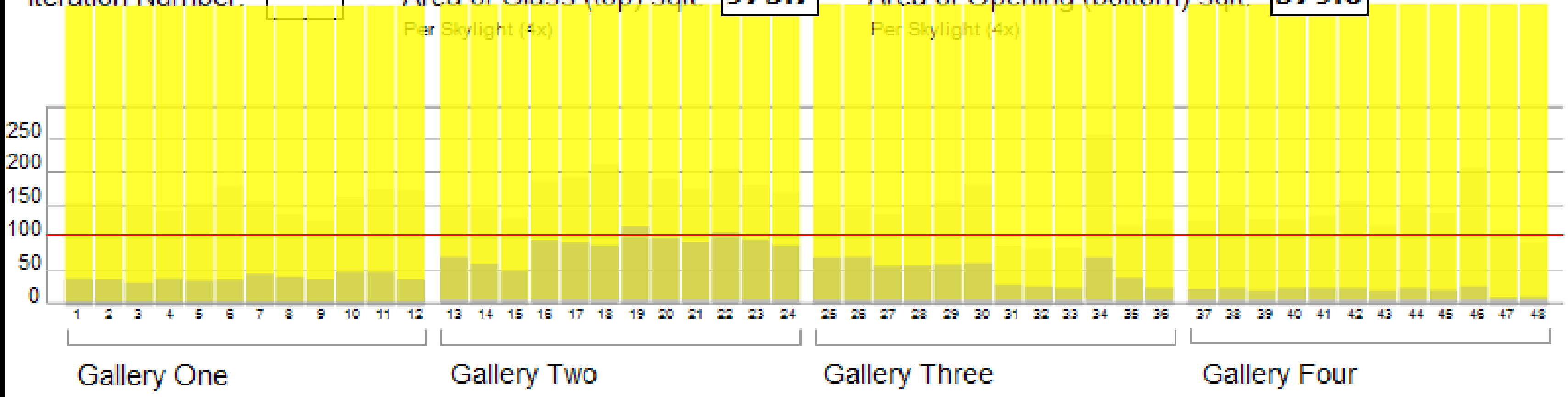
Iteration Number:

Area of Glass (top) sqft: **975.7**

Per Skylight (4x)

Area of Opening (bottom) sqft: **579.6**

Per Skylight (4x)



Putting it on paper: Narrative Summary

To refine the design for the QMA skylight, the team virtually manipulated its configuration and charted light levels at sensor points in the side galleries.

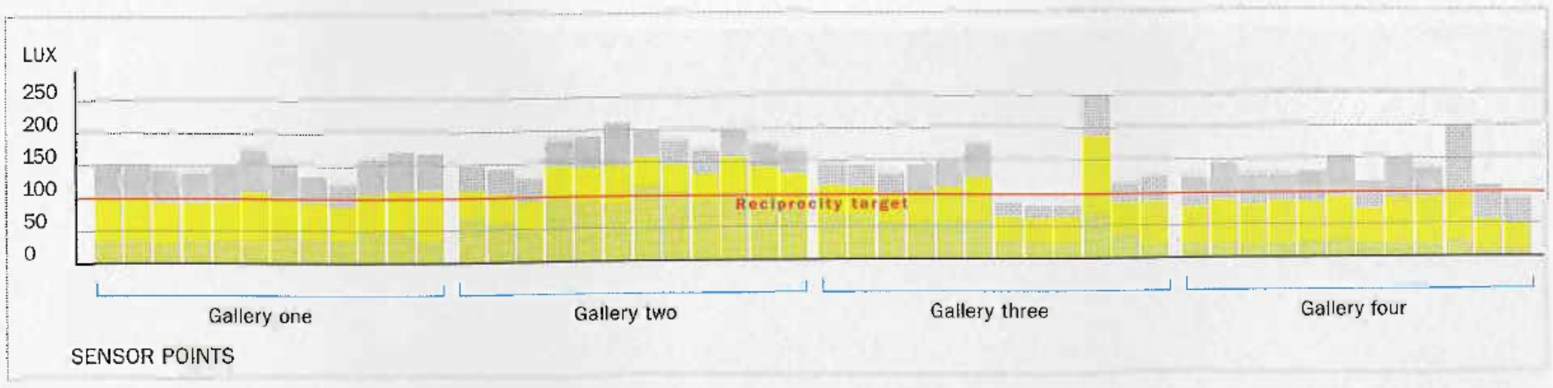
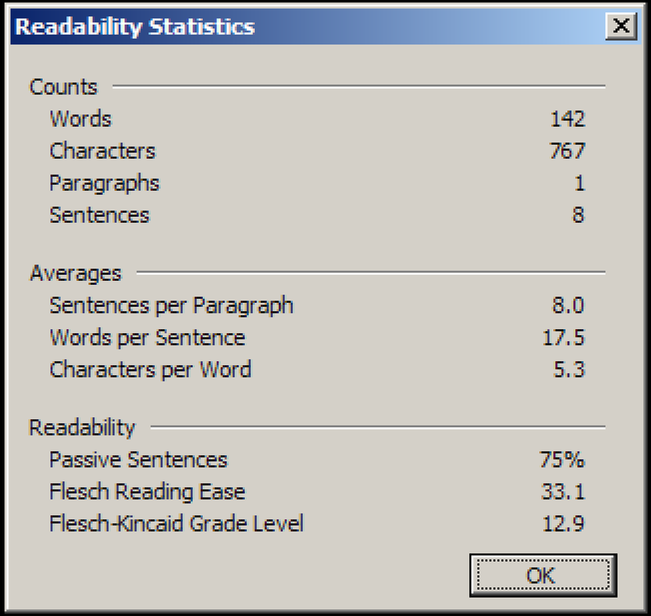


Image Credit: Architectural Record, 05.08 (Vol. 196, Iss. 5), Page 246

Putting it on paper: Narrative Summary

Test points were establish on various gallery walls to record light levels. A series of test days and times were used to measure daylight levels in galleries at the test points. *Radiance* models were run for each configuration of skylight. Results for each of the test points were documented and compared against a target value of 9.3 footcandles (100Lux). This target value was established as part of the reciprocity calculation method proposed to establish an annual daylight footcandle-hour budget figure. Refer to section 6.1 of this report for additional information on reciprocity calculations. Iteration number 8 was selected for its ability to manage daylight levels near the target with out additional diffusion and filtration of daylight by the winter garden enclosure. (Figure 9) The architects were provided the geometry of the successful iteration to integrate into the design (Figure 10 and 11).



Readability Statistics

Counts	
Words	142
Characters	767
Paragraphs	1
Sentences	8
Averages	
Sentences per Paragraph	8.0
Words per Sentence	17.5
Characters per Word	5.3
Readability	
Passive Sentences	75%
Flesch Reading Ease	33.1
Flesch-Kincaid Grade Level	12.9

Links:

(Note: This page did not appear in the version of the presentation given at the conference.)

Flash Kit
GotAPI Actionscript
O'Reilly Flash Publications
ActionScript.org

Python
Python's Beginners Guide
GotAPI Python
O'Reilly Python Publications
Python Imaging Library
Parallel Python

Perl
O'Reilly Perl Publications

Lua

Axel Jacobs' Learnix (and tutorials)
Ubuntu

FFmpeg
ImageMagick
RadSunpath and RadDisplay

Interactive is the New Black

David Smith Design IALD, IES, EIT
Lighting Designer, Buro Happold

Buro Happold