Modeling Complex Fenestration with rtcontrib

Greg Ward
Anyhere Software
What We’ll Do

- Create a simple one-window daylit space
- MGF model of curved, specular blinds
- Compute transmission using \texttt{genBSDF}
- Create window distribution using \texttt{mkillum}
- Render with and without blinds geometry
- Perform annual simulation via 3-phase method
Our Simple Model
Blinds Model
# Example of demi-specular blinds
# Upper surface is concave and specular
# Lower surface is diffuse
m _blind_specular =
  c
  rs .9 0
  rd .05
  sides 1
m _blind_diffuse =
  c
  rd .7
  sides 1
o demi-spec_blinds
xf -t 0 -.2 0 -rx -25 -a 22 -rx 2.273 -i 1 -t 0 .2 0 -a 30 -t 0 .1 0 -i 1 -t 0 0 -.1
v v1 =
  p -2 1e-4 .008
  n 0 .2 -.008
v v2 =
  p 2 1e-4 .008
  n 0 .2 -.008
v v3 =
  p 2 1e-4 -.008
  n 0 .2 .008
v v4 =
  p -2 1e-4 -.008
  n 0 .2 .008
m _blind_specular
f v1 v2 v3 v4
v v5 =
  p -2 -1e-4 -.008
v v6 =
  p 2 -1e-4 -.008
v v7 =
  p 2 -1e-4 .008
v v8 =
  p -2 -1e-4 .008
m _blind_diffuse
f v5 v6 v7 v8
xf
o

See http://radsite.lbl.gov/mgf
Compute BSDF

```
  genBSDF +mgf spec_blinds.mgf \n  > spec_blinds.xml
```

I added `-c 4000`,
which took 5 hours on my laptop(!)
Gosh, wouldn’t that be nice?
Running mkillum

Create illum with BSDF geometry

```sh
#@mkillum d=./spec_blinds.xml l= t=0 s=100
void polygon window
0 0
0 0 12
0 2 1
0 6 1
0 6 2.8
0 2 2.8
```

```sh
oconv sky.rad room-l.rad > base.oct
mkillum base.oct < room-l.rad \
| oconv sky.rad - > room-lmki.oct
```
rpict -ps 1 -vf room.vf -dj .99 -ab 2 -x 2048 -y 2048 room-1mki.oct | pfilt -1 -x /2 -y /2 -r .6 > room-1bmki.hdr

Rendered
Now, without Geometry

```plaintext
#@mkillum d=./spec_blinds.xml l+ s=100
void polygon window
0
0
0
12
  0 2 1
  0 6 1
  0 6 2.8
  0 2 2.8
mkillum base.oct < room+l.rad \n  | oconv -> room+lmki.oct
```

Notice sky.rad is gone
Comparison
Averages Similar
Q: Why Not Use mkillum by Itself?
Check out Ceiling
Diffuse Blind Case
Annual Simulation

Phase I:
Use `rtcontrib` to get daylight coefficients relating sky patches to incident directions

Phase II:
Use `rtcontrib` to relate exiting portal directions to desired measurement locations (e.g., image)

Phase III (time-step calculation):
sky * incident * BTDF * exiting
Phase I: Compute $D$

Klems full BSDF sampling basis
Phase I (cont’d)

```plaintext
oconv dummysky.rad room.rad > phase1.oct

# dummysky.rad

void glow skyglow
0
0
0
4 1 1 1 0

skyglow source sky
0
0
0
4 0 0 1 360
```
Phase I (cont’d)

Sample count

```plaintext
genklemsamp -ff -c 1000 -vd -1 0 0 \nwindow.rad | rtcontrib -c 1000 -ff \n-f tregenza.cal -b tbin -bn Ntbins \n-m skyglow -w phase1.oct > exterior.dmx
```

Window orientation

Subject material after bin options

Let’s look inside...
Input is ray direction from rtcontrib

```
alt = Asin(Dz)/DEGREE;
azi = Atan2(Dx,Dy)/DEGREE;

tazi(inc) = if(359.9999-.5*inc - azi, floor((azi+.5*inc)/inc), 0);

tbin = if(-alt, 0,
         select(floor(alt/12) + 1,
                1 + tazi(12),
                31 + tazi(12),
                61 + tazi(15),
                85 + tazi(15),
                109 + tazi(20),
                127 + tazi(30),
                139 + tazi(60),
                145
         ) ) ;

Ntbins : 146; { total number of bins }
```

Output is tbin given to rtcontrib -b option

```
tregenza.cal
```

Total number of bins for -bn option
Phase II: Compute $V$

```
oconv winlight.rad room.rad > phase2.oct
```

```
void light winlight
0
0
0
3 1 1 1

winlight polygon window
0
0
0
12
0 2 1
0 6 1
0 6 2.8
0 2 2.8
```
Phase II (cont’d)

Generates view rays for image

\[ \text{vwrays} \ -ff \ -x \ 1000 \ -y \ 1000 \ -vf \ \text{room.vf} \ \ |
\text{rtcontrib} \ `\text{vwrays} \ -x \ 1000 \ -y \ 1000 \ -vf \ \text{room.vf} \ -d` \ |
-ffc \ -o \ \text{rend/part%03d.hdr} \ -f \ \text{klems_int.cal} \ |
-b \ \text{kbinW} \ -bn \ \text{Nkbins} \ -m \ \text{winlight} \ -V- \ |
-ab \ 2 \ -ds \ .1 \ -dj \ .9 \ -ad \ 1500 \ -lw \ 4e-4 \ |
-n \ 3 \ \text{phase2.oct} \]

Images by bin number

Monte Carlo rendering parameters

2.5 hours later on my laptop...
View Components V
Phase III: Time Step

- Use `genskyvec` to create sky patch vector $s$
- Use `dctimestep` to multiply it all together

$$i = \text{VTD} \cdot s$$
Phase III (cont’d)

Gets Tregenza sky

gensky 7 12 17 | genskyvec -m 1 \
| dctimestep rend/part%03d.hdr spec_blinds.xml exterior.dmx \
> rend0712_17.hdr

Can use diffuse blinds, too

Takes 3.5 seconds on my laptop
Final Result
Compared to mkillum
Animation of Day

set i=1
foreach hr ( {6,7,8,9,10,11,12,13,14,15,16,17,18}:{00,15,30,45} )
gensky 7 12 $hr | genskyvec -m 1 \n| dctimestep rend/part%03d.hdr spec_blinds.xml exterior.dmx \n> anim1/frame$i.hdr
@ i++
end

set i=1
foreach hr ( {6,7,8,9,10,11,12,13,14,15,16,17,18}:{00,15,30,45} )
gensky 7 12 $hr | genskyvec -m 1 \n| dctimestep rend/part%03d.hdr diff_blinds.xml exterior.dmx \n> anim2/frame$i.hdr
@ i++
end
Specular Blinds
Diffuse Blinds
Resources

Axel Jacobs’ tutorial:
“Understanding rtcontrib”
http://luminance.londonmet.ac.uk/
➔ LEARNIX ➔ DOCUMENTATION