LBNL virtual TracePro goniophotometer v 1.0

Jacob C. Jonsson and Christian Kohler

Environmental Energy Technology Department
Ernest Orlando Lawrence Berkeley National Laboratory, USA

jcjonsson@lbl.gov

November 5, 2009

Abstract

The LBNL Virtual Goniophotometer is a set of TracePro scripts designed to automate raytracing of complex fenestration systems. The data output from the scripts is tailored for use in Window 6 and with the Radiance preprocessor `mkillum`. This document describes the scripts, and demonstrate the usage of them for a system of Venetian blinds.

Contents

1 Introduction

The goal of the LBNL virtual goniophotometer is to obtain simulated bidirectional scattering distribution functions (BSDFs) for complex materials. In this document the acronym BSDF is used to encompass both reflectance BRDF and transmittance BTDF data rather than to use the more complicated construction of BR(T)DF.

The scripts automates raytracing for 145 different incident angles. Data collection for all outgoing transmittance and reflectance directions is automated. Raytracing from the back side is easily achieved for samples where the front and back scatters light differently, e.g. Venetian blind systems.

The basis system chosen for the data is the Klems' system[1]. It has 145 directions in each hemisphere. It differs from the Tregenza system in that there are fewer patches towards the really high theta-angles. This results in that the data points in the Klems coordinate have an even contribution to the total direct-hemispherical value.

The results from the simulations are formated for use in Window 6 and the Radiance postprocessor mkillum, but it is BSDF data and can therefore be used for whatever purpose needed. Window 6 has the capability to create a glazing system from several layers of glazings and shades in any imaginable configuration, so ideally your system is something that is a single layer and can be combined with other producs later to save you from raytracing every combination of glazing and shading that you need. However, there is no restriction in the virtual goniophotometer on this, it is possible to simulate multi-layer systems just as well.

1.1 Requirements and Usage

The LBNL virtual TracePro goniophotometer requires a copy of TracePro by Lambda Research Corporation to run. The version 1 set of scripts are written for TracePro 3.1 and need minor changes might be required for later versions of TracePro.

 If the virtual goniophotometer is used for published results it is desired that this document, and its LBNL document number is referenced. If you like the tools and modify them we are happy to include contributions in future versions.

2 Core scripts

There are four core scripts in the package, source code is found at the end of this paper in appendices. Note that these does not include any sequencing of your tasks or creation of your system.

2.1 `createDetectorSphere` and `renameDetectorPatches`

These two scripts generate the detector patches and names them for future use in `traceAndDetect`. A sphere radius argument is needed when calling `createDetectorSphere`, this is given in the TracePro standard length unit and should be much larger than your illuminated spot on your sample.

The script works by creating a spherical shell which is subdivided into the theta- and phi-bands according to the hard coded coordinates specified by the Klems' coordinate system.

2.2 `traceAndDetect`

This script loops over all incident dircetions and for each raytraces and records the detected transmittance and reflectance. The number of rays used for each angle of incidence has to be given as an argument. The filenames of the results, one for BRDF and one for BTDF, are also required arguments.

The geometry of the light spot is hard coded in the file and it is quite probable that you might need to rewrite that part for each new geometry that you want to study. This is examplified in section 3.3.

Data is written in ASCII to a text file after each angle of incidence, but there is currently no way to continue a partial run if one is aborted in mid run.

Raytracing is a random process and the accuracy of the result is dependent on the numbers of rays traced. The impact of chosing a higher number of rays can be demonstrated by raytracing a Lambertian surface, i.e. a surface with constant BSDF regardless of scattering angle. The difference between three different simulations are shown in table 1. Here we can see that the standard deviation of the values decrese with the square root of the number of beams, i.e. increasing the number of rays by 100 times reduces the standard deviation with a factor of 10. This table is specific for a Lambertian surface where approximately 1/145 of all the rays goes to each patch. For a specular sample almost all rays will go to a single or a few patches and therefore you would get better statistics, hence better accuracy, for those patches, but maybe less accuracy if there is a weak diffuse component. The accuracy if a patch is not in the total number of rays, but rather how many rays that hits that patch.

Number of rays	Mean value	Minimum	Maximum	Standard deviation
100000	0.31829	0.28276	0.34755	0.012617
1000000	0.31826	0.30663	0.32662	0.0036054
10000000	0.31828	0.31454	0.32075	0.0012575

Table 1: The impact from number of rays on the accuracy of a Lambertian surface.

2.3 `flipFrontBack`

The `flipFrontBack` script is included for complex systems that have different properties depending on which side is illuminated. The script simply rotates the sample around up-axis, calling `traceAndDetect` will then give the BSDF values for the backside of the sample.

3 Venetian blind example

This section demonstrates the application of the scripts and how to sequence them for repeated simulations. Usually a main script or "run file" is generated, and that is the script that is called from the macro menu in TracePro.

3.1 Core

Loading and calling the core and creating a detector sphere is achieved with the following code

Listing 1: Head of the main script

```
(load "createDetectorSphere.SCM")
(load "renameDetectorPatches.SCM")
(load "traceAndDetect.SCM")
(load "flipFrontBack.SCM")

(file:new) ; Creates a new project window in TracePro
(createDetectorSphere 20000)    ; 20m radius
(renameDetectorPatches)
```

3.2 Creating a set of Venetian blind slats

The next step is to generate your sample, this can be achieved by manually running TracePro, pasting a model from another window, or, probably easiest, write code that generate your structure. In appendix E is a function that generate blinds. We can now extend our listing with the following lines:

Listing 2: Creating blinds in the main script

```
(load "drawBlinds.SCM")
(drawBlinds 45 19 290 0.1 16 12 0 (list "solref070" "Jaj") (list→
    "solref070" "Jaj"))
```

This will draw 19 slats at 45 degree tilt angle and other geometrical parameters as specified by the arguments. The last two arguments are the surface properties of top and bottom of the slat.

After the blinds are created we can raytrace the system with the line

Listing 3: Creating blinds in the main script

```
(traceAndDetect 100000 "BlindBTDF.txt" "BlindBRDF.txt")
```

And to obtain the interior properties, i.e. the properties from the back we add

Listing 4: Flipping blinds front to back

```
(flipFrontBack)
(traceAndDetect 100000 "BlindBTDFback.txt" "BlindBRDFback.txt")
```

3.3 Illumination pitfalls

Illumination of your system is sometimes critical, especially for structures with periodic features such as our venetian blinds. This is examplified by three different illumination conditions as shown in figure 1. Beam size nr 2 illuminates exactly one period of the slats and will give a correct value for how a regular system behaves. Beam size 1 doubles

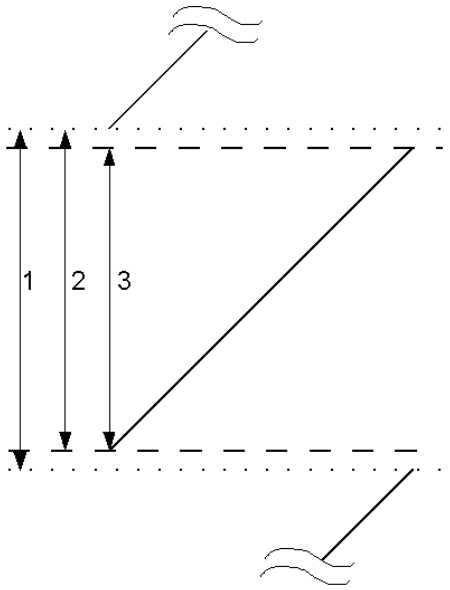

Figure 1: Three possible beam sizes illuminating the periodic system of Venetian blinds in different ways.

the amount of open space hence drastically increasing the transmittance and reducing the reflectance compared to size 1. Size number 3 misses both open areas and therefore gets a higher reflectance and a lower transmittance. Numbers for a white slat with a reflectance of .7 are given in table 2.

Propery	Beam 1 - large	Beam 2 - correct	Beam 3 - small
R_{dh}	.372	.397	.418
T_{dh}	.241	.191	.149

Table 2: Direct-hemispherical values for normal angle of incidence illuminations shown in figure 1.

The virual goniometer always illuminates the same area regardless of angle of incidence so this does not pose a problem at oblique angles of incidence.

3.4 Running multiple sample configurations

The strength of a script-based approach is to automate multiple similar runs. For example a script running three different materials would in its whole look like:

Listing 5: Running multiple blind configurations

```
(load "createDetectorSphere.SCM")
(load "renameDetectorPatches.SCM")
(load "traceAndDetect.SCM")
(load "flipFrontBack.SCM")
```

```
(load "drawBlinds.SCM")

;First configuration
(file:new)
(createDetectorSphere 20000)
(renameDetectorPatches)
(drawBlinds 45 19 290 0.1 16 12 0 (list "solref070" "Jaj") (list "solref070" "Jaj"))
(traceAndDetect 100000 "BlindR70BTDF.txt" "BlindR70BRDF.txt")
(flipFrontBack)
(traceAndDetect 100000 "BlindR70BTDFback.txt" "BlindR70BRDFback.txt")

;Second configuration
(file:new)
(createDetectorSphere 20000)
(renameDetectorPatches)
(drawBlinds 45 19 290 0.1 16 12 0 (list "solref090" "Jaj") (list "solref090" "Jaj"))
(traceAndDetect 100000 "BlindR90BTDF.txt" "BlindR90BRDF.txt")
(flipFrontBack)
(traceAndDetect 100000 "BlindR90BTDFback.txt" "BlindR90BRDFback.txt")

;Third configuration
(file:new)
(createDetectorSphere 20000)
(renameDetectorPatches)
(drawBlinds 45 19 290 0.1 16 12 0 (list "solref040" "Jaj") (list "solref040" "Jaj"))
(traceAndDetect 100000 "BlindR40BTDF.txt" "BlindR40BRDF.txt")
(flipFrontBack)
(traceAndDetect 100000 "BlindR40BTDFback.txt" "BlindR40BRDFback.txt")
```

4 The Future of the LBNL TracePro virtual goniophotometer

The virtual goniophotometer is scheduled for rewriting during 2010 to incorporate TracePro version 5 enhancements.

References

[1] J. H. Klems. A new method for predicting the solar heat gain of complex fenestration systems – i. overview and derivation of the matrix layer calculation. In *ASHRAE*

Transactions, volume 100(pt.1), pages 1065–1072, 1994. LBL-34715.

Appendices

Below are listings of the current code used in the LBNL virtual goniometer. All code is available in its latest version from the corresponding author at email `jcjonsson@lbl.gov`.

A createDetectorSphere

Listing 6: createDetectorSphere.scm

```
(define createDetectorSphere
  (lambda (iradius)
    ;iradius is the inner radius, of the final detector
    ;oradius is the outer radius of the sphere, to give it some wall thickness
    ;(define iradius 2000)
    ; make outer shell 5% larger
    (define oradius (* iradius 1.05))

    ;make a sphere that is larger then the target sphere, to create some wall thickness
    (define ck-detector_ID_1
      (insert:sphere oradius ))

    ;draw the inner sphere, the size we really want
    (define ck-detector_ID_2
      (insert:sphere iradius ))

    ;subtract one from the other, to create a shell
    (bool:subtract
     (entity 1)
     (entity 2)
     )

    ;draw a block that will be used to subtract from the sphere, to get a hemisphere
    ;make it 1 unit larger to ensure a clean cut

    (define ck-detector_ID_3
      (insert:block  (+ (* oradius 2) 1) (+ (* oradius 2) 1) (+ oradius 2) ) )

    (entity:move ck-detector_ID_3 0.000000 0.000000 (- (+ (/ oradius 2) 1)) )
```

```
    (edit:clear-selection)
    (edit:add-selection (entity 1))
    (edit:add-selection (entity 3))
33  (bool:subtract
     (entity 1)
     (entity 3)
     )

38  (define thetaList (list    ;; this is a list of theta angles
                       75
                       65
                       55
                       45
43                     35
                       25
                       15
                       5
                       ))
48
    (define PI 3.14159265358979323846)
    (define deg-rad
      (lambda (deg)
        (* deg (/ PI 180))
53      )
      )

    ;loop through all the theta's in the list, creating a sheet →
       for each of them
    (for-each
58   (lambda (theta1)

       ;define the theta cutting sheets
       ;make each sheet 10 unit larger then the outer radius of →
          the spheres

63     ; Construct a list of the edges
       ;
       (define edge-list
         (list
          (edge:linear (position (+ oradius 10) (+ oradius 10) →
             (* iradius (sin (deg-rad (- 90 theta1)))) )  (→
             position (+ oradius 10) (-(+ oradius 10)) (* →
             iradius (sin (deg-rad (- 90 theta1)))) ) )
68        (edge:linear (position (+ oradius 10) (-(+ oradius 10)→
             ) (* iradius (sin (deg-rad (- 90 theta1)))) )  (→
             position (-(+ oradius 10)) (-(+ oradius 10)) (* →
             iradius (sin (deg-rad (- 90 theta1)))) ) )
```

```
          (edge:linear (position (-(+ oradius 10)) (-(+ oradius →
             10)) (* iradius (sin (deg-rad (- 90 theta1)))) ) (→
             position (-(+ oradius 10)) (+ oradius 10) (* →
             iradius (sin (deg-rad (- 90 theta1)))) ) )
          (edge:linear (position (-(+ oradius 10)) (+ oradius →
             10) (* iradius (sin (deg-rad (- 90 theta1)))) ) (→
             position (+ oradius 10) (+ oradius 10) (* iradius (→
             sin (deg-rad (- 90 theta1)))) ))
          ) ; end list
        )       ; end define
;
; The boundary is enclosed so we can create the wire body
;
(define the-wire-body (wire-body edge-list))
;
; Create a sheet body from the wire-body.
;
(define the-sheet-body (sheet:cover-wires the-wire-body))
;
; Change the sheet body into a double-sided sheet body
;
(sheet:2d the-sheet-body)

) ;end lambda
thetaList) ;end for-each

;cut the phi patches
;i had to enter the theta=80 point, because the width of the→
    theta rings changes. this way there is a ring from 65 to→
    75 and one from 75-90
(define theta (list 97.5 82.5 80 70 60 50 40 30 20 10 0 -10)→
    )

(define phi0 (list 0 30 60 90 120 150 180 210 240 270 300 →
    330 360))
(define phi0a (list 0 30 60 90 120 150 180 210 240 270 300 →
    330 360))
(define phi1 (list 0 22.5 45 67.5 90 112.5 135 157.5 180 →
    202.5 225 247.5 270 292.5 315 337.5 360))
(define phi2 (list 0 15 30 45 60 75 90 105 120 135 150 165 →
    180 195 210 225 240 255 270 285 300 315 330 345 360))
(define phi3 (list 0 15 30 45 60 75 90 105 120 135 150 165 →
    180 195 210 225 240 255 270 285 300 315 330 345 360))
(define phi4 (list 0 15 30 45 60 75 90 105 120 135 150 165 →
    180 195 210 225 240 255 270 285 300 315 330 345 360))
(define phi5 (list 0 18 36 54 72 90 108 126 144 162 180 198 →
    216 234 252 270 288 306 324 342 360))
```

```
    (define phi6 (list 0 22.5 45 67.5 90 112.5 135 157.5 180 →
       202.5 225 247.5 270 292.5 315 337.5 360))
    (define phi7 (list 0 45 90 135 180 225 270 315 360))
    (define phi8 (list 0 ))
103 (define phi (list phi0 phi0a phi1 phi2 phi3 phi4 phi5 phi6 →
       phi7 phi8))
    (define t 0) ;theta counter
    (define p 0) ;phi counter
    (define CTheta 0)
    (define DTheta 0)
108 (define CPhi 0)
    (define DPhi 0)
    (define rOffset 0)
    ;make the cutting planes half the thickness of the distance →
       between the 2 spheres
    (set! rOffset ( / ( - iradius oradius) 2))
113

    ;;; Calulate the X position for a given radius and angle
    (define xpoint
      (lambda (r theta phi)
118     (* r (sin  (deg-rad theta)) (cos (deg-rad phi)))
        )
      )

    ;;; Calulate the Y position for a given radius and angle
123 (define ypoint
      (lambda (r theta phi)
        (* r (sin (deg-rad theta)) (sin (deg-rad phi)))
        )
      )
128
    ;;; Calulate the Z position for a given radius and angle
    (define zpoint
      (lambda (r theta phi)
        (* r (cos  (deg-rad theta)))
133     )
      )
    ;define a plane based on four coordinates
    ;zOffset defines how far the plane should go into the →
       z-direction on either side
    (define GenPlane
138   (lambda (CTheta CPhi DTheta DPhi rOffset)
        ;(print (string-append (number->string CTheta) "  " (→
           number->string CPhi) "  " (number->string DTheta) "  →
           "(number->string DPhi)))
        ; Construct a list of the edges
```

```
        ;
        (solid:sheet (list
                      (position (+ (xpoint iradius CTheta CPhi) →
                         (* rOffset (cos (deg-rad CPhi)))) (+ (→
                         ypoint iradius CTheta CPhi) (* rOffset →
                         (sin (deg-rad CPhi)))) (zpoint iradius→
                          CTheta CPhi))
                      (position (- (xpoint iradius CTheta CPhi) →
                         (* rOffset (cos (deg-rad CPhi)))) (- (→
                         ypoint iradius CTheta CPhi) (* rOffset →
                         (sin (deg-rad CPhi)))) (zpoint iradius→
                          CTheta CPhi))
                      (position (- (xpoint iradius DTheta DPhi) →
                         (* rOffset (cos (deg-rad DPhi)))) (- (→
                         ypoint iradius DTheta DPhi) (* rOffset →
                         (sin (deg-rad DPhi)))) (zpoint iradius→
                          DTheta DPhi))
                      (position (+ (xpoint iradius DTheta DPhi) →
                         (* rOffset (cos (deg-rad DPhi)))) (+ (→
                         ypoint iradius DTheta DPhi) (* rOffset →
                         (sin (deg-rad DPhi)))) (zpoint iradius→
                          DTheta DPhi))
                      ) ;end list
                     ) ;end solid


        )
      )

    ; loop through all the values in the theta array
    (do ( (t 1 (+ t 1)))
      ( (= t (- (length theta) 1)) t)
      ;loop through all the values in the phi array associated →
          with that theta
      (do ( ( p 0 (+ p 1)))
        ( (= p (- (length (list-ref phi (- t 1) ) ) 1 )) p)
        ;(newline)
        ;(display (list-ref theta t))
        ;(display #\space)
        ;(display (list-ref (list-ref phi (- t 1)  ) p ) )
        ;(newline)

        (set! CTheta (/  ( + (list-ref theta (- t 1)) (list-ref →
           theta t)) 2) )
        (set! CPhi    (/  ( + (list-ref (list-ref phi (- t 1)) →
           (+ p 1)) (list-ref (list-ref phi (- t 1)) p )) 2))
```

```
        (set! DTheta (/  ( + (list-ref theta t) (list-ref theta →
           (+ t 1) )) 2) )
        (set! DPhi    (/  ( + (list-ref (list-ref phi (- t 1)) →
           (+ p 1)) (list-ref (list-ref phi (- t 1)) p )) 2))
        ;(print (string-append (number->string CTheta) "  " (→
           number->string CPhi) "  " (number->string DTheta) "  →
           "(number->string DPhi)))

        (GenPlane CTheta CPhi DTheta DPhi rOffset)
        )
      ;(newline)
      )

    ;Clean up the geometry by subtracting all the new surfaces →
       from the sphere
    ;cut the theta planes

    (bool:subtract
     (entity 1)
     (entity 8)
     (entity 13)
     (entity 18)
     (entity 23)
     (entity 28)
     (entity 33)
     (entity 38)
     (entity 43)
     (entity 44)
     (entity 45)
     (entity 46)
     (entity 47)
     (entity 48)
     (entity 49)
     (entity 50)
     (entity 51)
     (entity 52)
     (entity 53)
     (entity 54)
     (entity 55)
     (entity 56)
     (entity 57)
     (entity 58)
     (entity 59)
     (entity 60)
     (entity 61)
     (entity 62)
     (entity 63)
```

```
        (entity 64)
        (entity 65)
        (entity 66)
213     (entity 67)
        (entity 68)
        (entity 69)
        (entity 70)
        (entity 71)
218     (entity 72)
        (entity 73)
        (entity 74)
        (entity 75)
        (entity 76)
223     (entity 77)
        (entity 78)
        (entity 79)
        (entity 80)
        (entity 81)
228     (entity 82)
        (entity 83)
        (entity 84)
        (entity 85)
        (entity 86)
233     (entity 87)
        (entity 88)
        (entity 89)
        (entity 90)
        (entity 91)
238     (entity 92)
        (entity 93)
        (entity 94)
        (entity 95)
        (entity 96)
243     (entity 97)
        (entity 98)
        (entity 99)
        (entity 100)
        (entity 101)
248     (entity 102)
        (entity 103)
        (entity 104)
        (entity 105)
        (entity 106)
253     (entity 107)
        (entity 108)
        (entity 109)
        (entity 110)
```

```
      (entity 111)
258   (entity 112)
      (entity 113)
      (entity 114)
      (entity 115)
      (entity 116)
263   (entity 117)
      (entity 118)
      (entity 119)
      (entity 120)
      (entity 121)
268   (entity 122)
      (entity 123)
      (entity 124)
      (entity 125)
      (entity 126)
273   (entity 127)
      (entity 128)
      (entity 129)
      (entity 130)
      (entity 131)
278   (entity 132)
      (entity 133)
      (entity 134)
      (entity 135)
      (entity 136)
283   (entity 137)
      (entity 138)
      (entity 139)
      (entity 140)
      (entity 141)
288   (entity 142)
      (entity 143)
      (entity 144)
      (entity 145)
      (entity 146)
293   (entity 147)
      (entity 148)
      (entity 149)
      (entity 150)
      (entity 151)
298   (entity 152)
      (entity 153)
      (entity 154)
      (entity 155)
      (entity 156)
303   (entity 157)
```

```
        (entity 158)
        (entity 159)
        (entity 160)
        (entity 161)
308     (entity 162)
        (entity 163)
        (entity 164)
        (entity 165)
        (entity 166)
313     (entity 167)
        (entity 168)
        (entity 169)
        (entity 170)
        (entity 171)
318     (entity 172)
        (entity 173)
        (entity 174)
        (entity 175)
        (entity 176)
323     (entity 177)
        (entity 178)
        (entity 179)
        (entity 180)
        (entity 181)
328     (entity 182)
        (entity 183)
        (entity 184)
        (entity 185)
        (entity 186)
333     (entity 187)
        (entity 188)
        (entity 189)
        (entity 190)
        (entity 191)
338     (entity 192)
        (entity 193)
        (entity 194)
        (entity 195)
        (entity 196)
343     (entity 197)
        (entity 198)
        (entity 199)
        )
      ;rotate the hemi-sphere 180 around the x axis, to make the →
         axis correct for a transmission detector
348   ;(entity:copy (entity 1))
      ;(entity:rotate (entity 1) 0 0 0 1 0 0 180)
```

```
    ;(property:apply-name (entity 2) "Reflectance_Detector")
    ;(print "doing add-selection..")
    (print "createDetectorSphere starting")
    (edit:add-selection (entity 1))
    (view:zoom-all)
    ;(print "doing copy")
    (edit:copy)
    (view:zoom-all)
    ;(print "doing paste")
    (edit:paste)
    (view:zoom-all)
    ;(print "doing rotate")
    (entity:rotate (entity 1) 0 0 0 1 0 0 180)
    ;(view:zoom-all)
    ;(print "doing apply name-transmission")
    (property:apply-name (entity 1) "Transmission_Detector")
    ;(view:zoom-all)
    ;(print "doing apply name-reflection")
    (property:apply-name (entity 200) "Reflectance_Detector")
    ;There shall be not other object but the detectors
    ;There are if you call the lambda from a script!
    (edit:select-all-objects)
    (edit:remove-selection (entity:get-by-name
                            "Transmission_Detector" ))
    (edit:remove-selection (entity:get-by-name
                            "Reflectance_Detector"))
    (edit:cut)

    (view:zoom-all)
    (print "createDetectorSphere completed.")
    )
  )
```

B renameDetectorPatches

Listing 7: renameDetectorPatches.scm

```
(define renameDetectorPatches
  (lambda ()
    (print "renameDetectorPatches starting")
    ;this script selects first the Transmission and then the →
       Reflectance Detector in the current file
    ;it then goes through all the patches, finding all the →
       non-planar ones, and figures out the coordinate of its →
       vertices
```

```
;it then finds the corner vertices, and calculates Theta and Phi spherical coordinates for it
;it then names all the patches based on the Theta and Phi values
;Written by Christian Kohler, May/June 2005

(define v 0)
(define f 0)
(define x1 0)
(define y1 0)
(define z1 0)
(define r1 0)
(define Theta1 0)
(define Phi1 0)

(define center-theta 0) ; theta for the center (mid) point of a patch
(define center-phi 0) ; phi for the center (mid) point of a patch
(define SurfaceCount 0)      ;the number of surfaces that are not planar and have 4 vertices
(define v_list (list 0)) ;
(define faces_list (list 0))
(define PatchName "")

;(define myOutputPort (open-output-file "C:/OutputFile.txt"))

(define PI 3.14159265358979323846)
(define deg-rad
  (lambda (deg)
    (* deg (/ PI 180))
    )
  )

(define rad-deg
  (lambda (rad)
    (* rad (/ 180 PI))
    )
  )

;custom atan function to make sure that the phi angles don't become negative. it makes sure atan always returns something between 0 and 2pi
(define atan2
  (lambda(y x)
```

```
        (if (>= y 0)
            (atan y x)
            (+ (* 2 PI) (atan y x))
            )
49      ))

    ; find the minimum value in a list
    (define vector_min
      (lambda(input_vector)
54      (define minim (vector-ref input_vector 0))
        (do ((x 0 (+ x 1)))
          ((= x  (vector-length input_vector)) x)
          (if ( < (vector-ref input_vector x) minim)
              (set! minim (vector-ref input_vector x))
59            ) ;end -if
          ) ;end -do
        ; just a bogus multiplication by 1, so that the function→
            returns this value. there must be a cleaner way
        (* minim 1)
        ))
64
    ; find the maximum value in a list
    (define vector_max
      (lambda(input_vector)
        (define maxim (vector-ref input_vector 0))
69      (do ((x 0 (+ x 1)))
          ((= x  (vector-length input_vector)) x)
          (if ( > (vector-ref input_vector x) maxim)
              (set! maxim (vector-ref input_vector x))
              ) ;end -if
74        ) ;end -do
        ; just a bogus multiplication by 1, so that the function→
            returns this value. there must be a cleaner way
        (* maxim 1)
        ))

79
    (define Detector_list (list "Transmission_Detector" "→
        Reflectance_Detector"))

    ;execute the same code twice, once for a Transmission →
        detector and once for a Reflectance detector
    (for-each
84   (lambda (Detector)
       (define my_detector (entity:get-by-name Detector))
       (define faces_list (entity:faces my_detector))
```

```
        ;loop through the faces in the entity my_detector
89      (do ((f 0 (+ f 1)))
          ((= f (length faces_list)) f )

          ;create a list of vertices associated with entry f in →
             the faces_list
          (define v_list (entity:vertices (list-ref faces_list f)→
             ))
94
          ;(print (string-append "number of vertices " (number->→
             string (length v_list))) )
          (if (not(face:planar? (list-ref faces_list f)))

              (if (< (length v_list ) 8)
99                ; this do loop with x as a variable is just a →
                     dummy so that i can execute a bunch of code →
                     following an 'if' statement.
                  ; i can't figure out how to execute multiple →
                     statements following an 'if' without the do →
                     loop
                  (do ((x 0 ( + x 1) ) )
                    ((= x 1) x)
                   ; (print "****** got one" )
104                 (set! SurfaceCount (+ SurfaceCount 1))

                    (define theta_vertex_vector (make-vector (→
                       length v_list)))
                    (define phi_vertex_vector (make-vector (→
                       length v_list)))

109                 ; loop through the vertices, calculating →
                       theta and phi for each and storing it in a→
                        list
                    (do ((v 0 ( + v 1) ) )
                      ((= v (length v_list)) v)

                      (set! x1 (position:x (vertex:position (→
                         list-ref v_list v))))
114                   (set! y1 (position:y (vertex:position (→
                         list-ref v_list v))))
                      (set! z1 (position:z (vertex:position (→
                         list-ref v_list v))))

                      (set! r1 (sqrt (+ (math:pow x1 2) (math:pow→
                          y1 2) (math:pow z1 2) )))
                      (set! Theta1 (rad-deg (acos (/ z1 r1))))
119                   (set! Phi1 (rad-deg (atan2 y1 x1 )))
```

```
    (vector-set! theta_vertex_vector v Theta1)
    (vector-set! phi_vertex_vector v Phi1)
    ) ;end do

  (set! center-theta ( / ( + (vector_min →
     theta_vertex_vector) (vector_max →
     theta_vertex_vector)) 2))
  ;do a special case when center-phi=0, this →
     happens when maxPhi-minPhi>180
  ;the two phi's are centered around phi=0 but →
     average to 180 Phi1=345 Phi2=15, in this →
     case PhiCenter should be 0
  (if (>( - (vector_max phi_vertex_vector) (→
     vector_min phi_vertex_vector)) 180)
      (do ((x 0 ( + x 1) ) )
        ((= x 1) x)
       ; (print "special case")
        (set! center-phi 0.0)
        ;(display center-phi)
        ;(newline)
        )
      ;else just do the regular averaging
      (set! center-phi ( / ( + (vector_min →
         phi_vertex_vector) (vector_max →
         phi_vertex_vector)) 2))
      ) ;end -if
  ;(display center-phi)
  ;(newline)

  (if (< (abs ( - (round center-theta) →
     center-theta)) 0.01)
      (set! center-theta (round center-theta)))

  (if (< (abs ( - (round center-phi) center-phi→
     )) 0.01)
      (set! center-phi (round center-phi)))

  (set! PatchName (string-append "t" (number->→
     string center-theta) "p" (number->string →
     center-phi)))
  ;(print PatchName)
  (property:apply-name (list-ref faces_list f) →
     PatchName)
```

```
                    )
                ;this is a special case for theta=0 or 180 and →
                    phi=0, the surface will have as many →
                    vertices as there are segments in the ring
                ;next to it. in case of the Klems basis this is→
                     8
                (if (= (length v_list) 8)
                    (if (string=? Detector "→
                        Transmission_Detector")
                        (property:apply-name (list-ref →
                            faces_list f) "t180.0p0.0")
                        ;else it is a reflectance detector and →
                            the theta=0
                        (property:apply-name (list-ref →
                            faces_list f) "t0.0p0.0")
                        )
                    )

                )
            ) ; end if not planar


        ) ; end do faces
      ) ; end for-each lambda
    Detector_list)
    (print "renameDetectorPatches finished")
    )
  )
```

C traceAndDetect

Listing 8: traceAndDetect.scm

```
(define traceAndDetect
  (lambda (numberOfRays myTOutputFile myROutputFile)


    ;raytrace an existing file and then select the detector →
       surfaces and output the flux and btdf associated with →
       them
    ;written by Christian, June 2005

    (define thetaR (list 82.5 70.0 60.0 50.0 40.0 30.0 20.0 10.0→
        0.0))
```

```
    (define thetaT (list 97.5 110.0 120.0 130.0 140.0 150.0 →
       160.0 170.0 180.0))
    (define phi0 (list 0.0 30.0 60.0 90.0 120.0 150.0 180.0 →
       210.0 240.0 270.0 300.0 330.0 ))

    (define phi1 (list 0.0 22.5 45.0 67.5 90.0 112.5 135.0 157.5→
        180.0 202.5 225.0 247.5 270.0 292.5 315.0 337.5))
    (define phi2 (list 0.0 15.0 30.0 45.0 60.0 75.0 90.0 105.0 →
       120.0 135.0 150.0 165.0 180.0 195.0 210.0 225.0 240.0 →
       255.0 270.0 285.0 300.0 315.0 330.0 345.0 ))
    (define phi3 (list 0.0 15.0 30.0 45.0 60.0 75.0 90.0 105.0 →
       120.0 135.0 150.0 165.0 180.0 195.0 210.0 225.0 240.0 →
       255.0 270.0 285.0 300.0 315.0 330.0 345.0))
    (define phi4 (list 0.0 15.0 30.0 45.0 60.0 75.0 90.0 105.0 →
       120.0 135.0 150.0 165.0 180.0 195.0 210.0 225.0 240.0 →
       255.0 270.0 285.0 300.0 315.0 330.0 345.0))
    (define phi5 (list 0.0 18.0 36.0 54.0 72.0 90.0 108.0 126.0 →
       144.0 162.0 180.0 198.0 216.0 234.0 252.0 270.0 288.0 →
       306.0 324.0 342.0 ))
    (define phi6 (list 0.0 22.5 45.0 67.5 90.0 112.5 135.0 157.5→
        180.0 202.5 225.0 247.5 270.0 292.5 315.0 337.5 ))
    (define phi7 (list 0.0 45.0 90.0 135.0 180.0 225.0 270.0 →
       315.0))
    (define phi8 (list 0.0 ))
    (define phi (list phi0 phi1 phi2 phi3 phi4 phi5 phi6 phi7 →
       phi8))
    (define t 0) ;theta counter
    (define p 0) ;phi counter
    (define x1 0)
    (define y1 0)
    (define z1 0)
    (define r1 0)
    (define Xdirection 0)
    (define Ydirection 0)
    (define Zdirection 0)
    (define BRDF 0)
    (define BTDF 0)

    (define PatchName "")
    (define numrays_power 0)
    (define IncidentFlux 0) ; we will assume that all rays are →
       incident on the sample
    (define DetectorFlux 0) ; the flux that hits a specific →
       detector surface

    (define PI 3.14159265358979323846)
    (define deg-rad
```

```
      (lambda (deg)
        (* deg (/ PI 180))
        )
      )

    (define xpoint
      (lambda (r theta phi)
        (* r (sin  (deg-rad theta)) (cos (deg-rad phi)))
        )
      )

    ;;; Calulate the Y position for a given radius and angle
    (define ypoint
      (lambda (r theta phi)
        (* r (sin (deg-rad theta)) (sin (deg-rad phi)))
        )
      )

    ;;; Calulate the Z position for a given radius and angle
    (define zpoint
      (lambda (r theta phi)
        (* r (cos  (deg-rad theta)))
        )
      )

    ; open 2 files, one for Transmission data, and one for →
       Reflection data
    ; myToutputPort is for transmission data, myROutputPart is →
       for reflection data
    ; myTOutputfile is passed as a parameter when the script is →
       called
    (define myTOutputPort (open-output-file myTOutputFile))
    (define myROutputPort (open-output-file myROutputFile))


    (raytrace:set-simulation-mode-off)
   ; (raytrace:set-analysis-mode-on)
;set simulation mode off so I can see the rays
  ;  (raytrace:set-simulation-mode-on)
    (raytrace:threshold 0) ; Allow for low probability →
       scattering
    (raytrace:set-importance-sampling-off) ; Force hard ray →
       tracing.
    (raytrace:set-ray-splitting-off)
    (raytrace:clear-wavelengths) ; Only trace at 546 nm

 ; Force unpolarised light
```

```
(raytrace:set-polarization-on)
(raytrace:set-polarization-state-unpolarized)

(raytrace:set-simulation-prompt-off)

(raytrace:set-grid-origin (position 0 0 11)) ; Constant
(raytrace:set-grid-orientation-direction-vectors (gvector 0 0 -1) (gvector 0 1 0))

(raytrace:set-grid-boundary-annular 25 0)

;figure out the radius of the sphere. select patch t180.0p0.0 (could be any patch), get the x,y, z coordinates and calculate r

(define my_surf (entity:get-by-name "t180.0p0.0"))
(define v_list (entity:vertices my_surf))

(set! x1 (position:x (vertex:position (list-ref v_list 0))))
(set! y1 (position:y (vertex:position (list-ref v_list 0))))
(set! z1 (position:z (vertex:position (list-ref v_list 0))))
(set! r1 (sqrt (+ (math:pow x1 2) (math:pow y1 2) (math:pow z1 2) )))


 ;loop through all the patches and print the names to the file, as a header info line
   (display "Tranmission " myTOutputPort)

   (do ( (t2 0 (+ t2 1)))
     ( (= t2 (length thetaT) ) t2)
     ;loop through all the values in the phi array associated with that theta
     (do ( ( p2 0 (+ p2 1)))
       ( (= p2 (length (list-ref phi t2))) p2)

       (set! PatchName (string-append "t" (number->string (list-ref thetaT t2)) "p" (number->string (list-ref (list-ref phi t2) p2) )))
       (display PatchName myTOutputPort)
       (display " " myTOutputPort)
       )
     )


   (display "Reflection " myROutputPort)

```

```
        (do ( (t2 0 (+ t2 1)))
          ( (= t2 (length thetaR) ) t2)
          ;loop through all the values in the phi array →
              associated with that theta
          (do ( ( p2 0 (+ p2 1)))
            ( (= p2 (length (list-ref phi t2))) p2)

            (set! PatchName (string-append "t" (number->string→
                (list-ref thetaR t2)) "p" (number->string (→
               list-ref (list-ref phi t2) p2) )))
            (display PatchName myROutputPort)
            (display "␣" myROutputPort)
            )
          )


        ;raytrace 2^numrays_power rays, and set the energy per→
            ray so that the total incident is 1000 watts
        ;set total intensity to 1,000
        ;(raytrace:set-grid-pattern-random (math:pow 2 →
           numrays_power) (/ 1000000 (math:pow 2 numrays_power→
           )))
      ; numberOfRays is passed to this script as a variable
      (raytrace:set-grid-pattern-random numberOfRays (/ 1000 →
         numberOfRays))


        (newline myTOutputPort)
        (display numberOfRays myTOutputPort)
        (display "␣rays" myTOutputPort)
        (newline myTOutputPort)

        (newline myROutputPort)
        (display numberOfRays myROutputPort)
        (display "␣rays" myROutputPort)
        (newline myROutputPort)

      ;this  is a dummy raytrace to get the timereported in the →
         header. below it is run 145 times
          (raytrace:grid)
        (display (reports:get-raytrace-time #t) myTOutputPort)
        (display (reports:get-raytrace-time #t) myROutputPort)
       (newline myTOutputPort)
      (newline myROutputPort)

        (display "␣␣")
        (display numberOfRays)
```

```
        (display "␣rays")
        (newline)


        ;(newline myOutputPort)
        ;(display "Incidentflux, detectorflux, area, r2, theta→
            , BTDF, Incidentflux, detectorflux, area, r2, theta→
            , BRDF" myOutputPort)
        ;(newline myOutputPort)

    ; loop through all the values in the theta array for the →
       incident angles
    (do ( (t1 0 (+ t1 1)))
      ( (= t1 (length thetaT) ) t1)
     ; (print (string-append "t1 " (number->string t1)))
      ;loop through all the values in the phi array associated →
         with that theta
      (do ( ( p1 0 (+ p1 1)))
        ( (= p1 (length (list-ref phi t1 ) ) ) p1)
            ;(print (string-append "p1 " (number->string p1)))

        (set! Xdirection (xpoint r1 (list-ref thetaT t1) (→
           list-ref (list-ref phi t1) p1)))
        (set! Ydirection (ypoint r1 (list-ref thetaT t1) (→
           list-ref (list-ref phi t1) p1)))
        (set! Zdirection (zpoint r1 (list-ref thetaT t1) (→
           list-ref (list-ref phi t1) p1)))

        ;(print (string-append "theta" (number->string (list-ref→
            thetaT t1)) "phi" (number->string (list-ref (→
           list-ref phi t1) p1)) " "))
        ;(display (string-append "theta" (number->string (→
           list-ref thetaT t1)) "phi" (number->string (list-ref →
           (list-ref phi t1) p1)) " ") myTOutputPort)

        ;(print (string-append "X " (number->string Xdirection) →
            "  Y " (number->string Ydirection) "  Z " (number->→
           string Zdirection)))
        (raytrace:set-beam-orientation-direction-vectors (→
           gvector Xdirection Ydirection Zdirection) (gvector 0 →
           1 0))
        (raytrace:set-grid-origin (position (* 0.9 (- Xdirection→
           )) (* 0.9 (- Ydirection)) (* 0.9 (- Zdirection))))
        (raytrace:grid)

        ;walk through each theta and phi combination and →
           construct a Patchname, then query the file for the →
```

```
        flux incident on that patch.
;still have to make this work cleanly, for reflectance and transmittance. right now I just duplicate the code



(do ( (t2 0 (+ t2 1)))
  ( (= t2 (length thetaT) ) t2)
  ;loop through all the values in the phi array associated with that theta
  (do ( ( p2 0 (+ p2 1)))
    ( (= p2 (length (list-ref phi t2))) p2)

    (set! PatchName (string-append "t" (number->string (list-ref thetaT t2)) "p" (number->string (list-ref (list-ref phi t2) p2) )))
    ;(display PatchName)
;     (print (string-append "Transmission Patchname " PatchName))
    (define my_surf (entity:get-by-name PatchName))
    (define v_list (entity:vertices my_surf))
    (set! x1 (position:x (vertex:position (list-ref v_list 0))))
    (set! y1 (position:y (vertex:position (list-ref v_list 0))))
    (set! z1 (position:z (vertex:position (list-ref v_list 0))))
    (set! r1 (sqrt (+ (math:pow x1 2) (math:pow y1 2) (math:pow z1 2) )))

    (set! IncidentFlux (* (raytrace:get-grid-peak-flux) (raytrace:get-grid-total-rays)))
    (set! DetectorFlux (raytrace:get-incident-flux my_surf))
;   (display DetectorFlux)
    (set! BTDF (* (/ DetectorFlux IncidentFlux) (/ (math:pow r1 2) (* (face:area my_surf) (abs( cos (deg-rad (list-ref thetaT t2))))))))

;   (display "  ")
;   (newline)
;   (print (string-append "BTDF " (number->string BTDF)))

;    (display IncidentFlux myOutputPort)
;    (display " " myOutputPort)
```

```
;    (display DetectorFlux myOutputPort)
;    (display " " myOutputPort)
;    (display (face:area my_surf) myOutputPort)
;    (display " " myOutputPort)
;    (display (math:pow r1 2) myOutputPort)
;    (display " " myOutputPort)
;    (display (list-ref thetaT t2) myOutputPort)
;    (display " " myOutputPort)
   (display BTDF myTOutputPort)
   (display "␣" myTOutputPort)
   ;        (print (string-append "  " (number->
      string (raytrace:get-incident-flux my_surf))))

   ;do all the same tricks for Reflection sphere

   (set! PatchName (string-append "t" (number->string->
       (list-ref thetaR t2)) "p" (number->string (->
      list-ref (list-ref phi t2) p2) )))
   ; (print (string-append "Reflection Patchname " ->
      PatchName))
  ; (display PatchName)

   (define my_surf (entity:get-by-name PatchName))
   (define v_list (entity:vertices my_surf))
   (set! x1 (position:x (vertex:position (list-ref ->
      v_list 0))))
   (set! y1 (position:y (vertex:position (list-ref ->
      v_list 0))))
   (set! z1 (position:z (vertex:position (list-ref ->
      v_list 0))))
   (set! r1 (sqrt (+ (math:pow x1 2) (math:pow y1 2) ->
      (math:pow z1 2) )))

   (set! IncidentFlux (* (raytrace:get-grid-peak-flux->
      ) (raytrace:get-grid-total-rays)))
   (set! DetectorFlux (raytrace:get-incident-flux ->
      my_surf))
  ;  (display "  ")
  ;  (display DetectorFlux)
   (set! BRDF (* (/ DetectorFlux IncidentFlux) (/ (->
      math:pow r1 2) (* (face:area my_surf) (abs( ->
      cos (deg-rad (list-ref thetaR t2)))))))))

  ; (display "  ")
  ; (display BRDF)
   ;(newline)
```

```
                ;(display IncidentFlux myOutputPort)
                ;(display " " myOutputPort)
                ;(display DetectorFlux myOutputPort)
                ;(display " " myOutputPort)
                ;(display (face:area my_surf) myOutputPort)
                ;(display " " myOutputPort)
                ;(display (math:pow r1 2) myOutputPort)
                ;(display " " myOutputPort)
                ;(display (list-ref thetaR t2) myOutputPort)
                ;(display " " myOutputPort)
                (display BRDF myROutputPort)
                (display "␣" myROutputPort)

                ;(newline myROutputPort)
                ;(print " looping phi")
                ) ;end do phi
              ;(print " looping theta")
            ) ; end do theta

        (newline myTOutputPort)
        (newline myROutputPort)
        )  ; end loop through phi array
        ) ; end loop through theta arra


    (close-output-port myTOutputPort)
    (close-output-port myROutputPort)
    )

  )
```

D flipFrontBack

Listing 9: flipFrontBack.scm

```
(define flipFrontBack
  (lambda ()
    ;Select all objects
    (edit:select-all-objects)
    ;Deselect the spheres
    (edit:remove-selection (list (entity:get-by-name "→
        Transmission_Detector")
                                 (entity:get-by-name "→
                                     Reflectance_Detector")
                           )
    )
```

```
    ;Define function apply on each object
    (define flipY
        (lambda (flippedEntity)
          (entity:rotate flippedEntity  0 0 0 0 1 0 180)
        )
     )
    ;Apply flipY to all selected objects
    (for-each flipY (part:selection))
    (view:zoom-all)
  )
)
```

E drawBlinds

Listing 10: drawBlinds.scm

```
(define drawBlinds
  (lambda (blindAngle nrBlinds blindX blindY blindZ Yseparation →
     bendR topMaterial bottomMaterial)

;Define variables, you are responsible for defining a geometry →
   which does not
;lead to intersecting objects (one blind existing inside another→
   !)

;The slats are centered around (0,0,0) this means that for an →
   even number
;of slats you get an edge at 000 and for an odd number of slats →
   the center
;of a slat will be at 000.

;On tilt angles:
; Definition of Positive Z direction for this explanation  - →
   ----> +
; Positive tilt angle means slat edge on positive z side will be→
   lower
; than the slat edge on the negative side. Like a backslash:  \
; Negative tilt is the opposite:  /
; Upside down is possible by defining angles higher than 90 deg,→
   this
; affects both the bend and the facing of the bottomMaterial →
   since the
; blindAngle is a just like a physical rotation.


;;;Example of parameters:
```

```
;(define blindAngle                -65   )   ; Tilt angle (0 = fully→
    open 90 all closed)
;(define nrBlinds                   12   )   ; Number of blinds
;(define blindX (* scalingF       100 ))   ; dimentions for each →
   blind
;(define blindY (* scalingF         1 ))
;(define blindZ (* scalingF        25.4 ))
;(define Yseparation (* scalingF  25.4 ))   ; distance between →
   blinds
;(define bendR  (* scalingF       100 ))   ; Bend on the blinds, →
   cryptic parameters...


;Define materials. Major warning, these are properties for →
   surfaces.
;blindParametric assumes that the blind is opaque.
;Each material is a list of Catalog and Name which are used in →
   the
;surface property editor.
;Specular1 and Specular2 have been made by solving for →
   absorptance given
;the following input into the ABg scattering model used for →
   surfaces
;surface       Rs      A     B   g
;specular1     0.6   0.02  0.03  2
;specular2     0.2   0.06  0.01  1.2
;(define topMaterial     (list "Specular1"      "Jaj" ))
;(define bottomMaterial (list "Specular2"      "Jaj" ))

; Calculate starting value for current Y
;(define curY (- 0 (* Yseparation (/ nrBlinds 2))))
(define Y0  (- 0 (* Yseparation (/ (- nrBlinds 1) 2)
                    )
                )
)
(define Y '()) ;Define Y as an empty list to allow append op.
; Generate list of Y-values
(do
  (( i 0 (+ i 1))) ;loop from 0 in incs of 1
  ((= i nrBlinds)) ;terminate at nrBlinds
  (set! Y (append Y (list (+ Y0 (* Yseparation i))))
          ) ;assign to Y the list Y appended with new value
  )
)



; Function for drawing a blind at a given y value
```

```
(define drawSlab
  (lambda (y)
     (begin

       ;insert a solid slab
       (define curBlind (insert:block blindX blindY blindZ))

;Used to make it a solid, now apply only bottom mtrl to bottom, → rest is topMaterial
       (property:apply-surface curBlind topMaterial)

       (property:apply-color curBlind 84 84 84)

;define surface properties of bottom (#2) surface
       (define curBotSurface (caddr(entity:faces curBlind))
        )
       (property:apply-surface  curBotSurface    ;Entity is → current bottom
                                bottomMaterial   ;Defined → initially
       )

       ;Bend the entity
       (if (> bendR 0)
       (entity:bend curBlind (position 0 0 0) ;neutral rot
                             (gvector  1 0 0) ;bending axis
                             (gvector  0 1 0) ;bending dir
                             bendR            ; Two of three → params are used
                             0                ; to define bend. → 0 here works fine.
                             0                ; 0 = gotten from → two above
                             #t               ; True to bend all→ of the blind
       )
       ;else
       (entity:bend curBlind (position 0 0 0) ;neutral rot
                             (gvector  1 0 0) ;bending axis
                             (gvector  0 -1 0) ;bending dir
                             (- 0 bendR)      ; Two of three → params are used
                             0                ; to define bend. → 0 here works fine.
                             0                ; 0 = gotten from → two above
```

```
                                #t                ; True to bend all→
                                    of the blind
          )
        );endif
100     ;rotate the given angle
         (define curRotation (transform:rotation
                                (position 0 0 0)
                                (gvector  1 0 0)
                                blindAngle
105                            )
         )
         (entity:transform curBlind curRotation
        )

110       ;translate the blind
         (entity:transform curBlind (transform:translation
                                      (gvector 0 y 0)
                                     )
         )
115    ) ; end begin
  ) ; end lambda
) ; end drawSlab (y)

; The actual drawing of the slabs is done with this loop
120 (for-each drawSlab Y)

;zoom out to show all!
(view:zoom-all)
)
125 );ends lambda and define drawBlinds
```