CS395/495: IBMR Take-Home Final Exam | SOLUTIONS

Assigned: 3:30pm,Thurs June 5, 2003
Due: **no later than** 3:30pm,Thurs June 12, 2003

Turn in your work either on paper brought to my office, or by e-mail to the both TA and the instructor (abhinav @cs.northwestern.edu, jet@cs.northwestern.edu). Be sure to number the pages and put your name on each page. Please do your own work—do not discuss the problems or your solutions with other students until after the due date.

1) **P³ Planes:**
   (a) Find the 4-vector describes the plane in P³ that passes through these three Cartesian R³ points; the x axis at (xp,0,0), the y axis at (0, yp,0) and the z axis at (0,0,zp).

   In P³ space, point p =[x,y,z,w]^T is on plane π = [a b c d]^T if and only if ax+by+cz+dw=0. We could find the answer by cross-products, but there is an easier way:
   --Scaling π by a constant has no effect; thus scale by (1/d), or equivalently, let d=1.
   --Similarly, scaling p by a constant has no effect; thus let w=1. Then πp=0 for each of the 3 points becomes a*xp+1=0, b*yp+1=0, c*zp+1=0. By inspection,
   | π = [-1/xp, -1/yp, -1/zp, 1], | or a scaled version (scale by –1 is popular).

   (b) Write an H matrix that will transform this plane to the x-y plane.

   --Find the 4-vector x-y plane: [0,0,1,0].
   --Then devise an H matrix that will convert to it from the plane π. There are many, many correct answers, including translation by [1/xp,1/yp,1/zp] followed by rotations that put the us in the x-y plane, but I think the simplest one simply zeroes out all x1,x2 and x4 terms and scales the x3 term to
   1:   H= [0   0   0   0]
   **   [0   0   0   0]**
   **   [0   0 -zp   0]**
   **   [0   0   0   0]**

   How did I find [0,0,1,0]? In general, how can I find any desired plane?
   Recall that for plane [a,b,c,d]^T the plane's normal direction is given by [a,b,c,0]. Like any P3 point, we can scale by any desired constant without affecting the position in P3; if we define len = sqrt(a²+b²+c²), then the direction can also be written D = [a/len, b/len,c/len,0]. If we move a point from the origin outwards in this direction by a distance of 1.0, the point is [a/len,b/len,c/len,1]. If we had instead moved the point outwards by distance S, we would arrive at point [Sa/len,Sb/len,Sc/len,1] or equivalently [a/len,b/len,c/len, 1/S]. Now—what distance S$_0$ would take the point out to the plane [a,b,c,d]? Solve for it: [a,b,c,d]^T[a/len,b/len,c/len, 1/S$_0$]=0, rewrite as (a² + b² + c²)/len + d/S$_0$ =0, divide both sides by len to get (a² + b² + c²)/len² + d/(len*S$_0$) = 0, or S$_0$ = -d / len. The x-y plane has a unit normal vector of [0,0,1] = [a/len,b/len,c/len]; len=1, so [a,b,c]=[0,0,1]. The plane passes through the origin, so S$_0$=0, thus d=0.

   (c) Find the direction vector (a 4-vector) D that is perpendicular to the plane given in (a).

   (See the last paragraph of previous answer for explanation). The direction vector perpendicular to the plane is D = [a,b,c,0]^T or any scaled version of this.

   (d) Construct a plane π that includes the Cartesian R3 point [a,b,c] and the line described by the

plane-span matrix `W* = [d e f g]`
`              [h j m n]`

THE HARD WAY: USE SPANS

Recall that P^3 plane π contains point p iff $\pi^T p=0$, and that the line given by plane-span matrix W* intersects point p iff W* p=0. But spans don't give us a convenient way to find an unknown plane π. (if I gave you W instead of W* the problem is trivial--just stack W and p^T to form matrix M; solve $M\pi=0$ by null space methods).

Instead, you need to find two points on the W* line, and these use these 2 points with p to find plane π. Any two different points on the line will do, so let's find the points where the line hits the planes given by x=0,y=0,or z=0. In P3 coords, these planes are πx = [1,0,0,0], πy = [0,1,0,0] and πz= [0,0,1,0]. We can find the intersection of the line W* with each plane using the 'join' operation of Lecture 8, and we call the points we find px,py,pz respectively. (at least 2 of these points are guaranteed to exist when the line is parallel to the x,y, or z axis). To find px,py,pz, augment the W* matrix by adding a 3rd row whose elements are given by πx, πy or πz respectively, forming the 3x4 matrices Mx*, My*, and Mz*. To find points px,py,pz use SVD to solve Mx* px = 0, My* py = 0 and Mz* pz=0. Then use two of these points and the given point p to find the unknown plane π. How? Convert to 3D Cartesian coordinates (e.g. scale the P3 points so that x3=1) and then use cross products: [(px-p) $\times$ (py-p) | -p·(px $\times$ py)].

THE EASY WAY: USE PLUCKER MATRICES

(see pg. 52 in book). We know that the rows of a plane-span matrix W* are planes that intersect at the desired line in P3. But we can also use these planes to make a (plane) Plucker matrix: name the planes P = [d e f g]^T, Q = [h j m n]^T, and then L* = PQ^T-QP^T. Then the plane π that includes both the line L* and the point p is given by:

$\boxed{\pi = L^* \ p}$

```
L*=[h][d e f g]-[d][h j m n]=[hd-dh he-dj hf-dm hg-dn]=[  0    he-dj hf-dm hg-dn]
   [j]          [e]          [jd-eh je-ej jf-em jg-en] [dj-he   0    jf-em jg-en]
   [m]          [f]          [md-fh me-fj mf-fm mg-fn] [dm-hf em-jf   0    mg-fn]
   [n]          [g]          [nd-gh ne-gj nf-gm ng-gn] [dn-hg en-jg fn-mg   0  ]
L*p=π =
[  0   he-dj hf-dm hg-dn][a] = [b(he-dj)+c(hf-dm)+ (hg-dn)]
[dj-he   0   jf-em jg-en][b]   [a(dj-he)+c(jf-em)+ (jg-en)]
[dm-hf em-jf  0    mg-fn][c]   [a{dm-hf}+b(em-jf)+ (mg-fn)]
[dn-hg en-jg fn-mg   0  ][1]   [a(dn-hg)+b(en-jg)+c(fn-mg)]
```

2) **P^3 Lines:**

Consider a line through two points $[a,b,c]^T$ and $[d,e,f]^T$ in Cartesian R^3 space:

(a) Write a parametric equation for a line L(t) in P^3 space that passes through the two R^3 points, where t measures distance along the line, where $L(0) = [a,b,c,1]^T$, and t is >0 at the 'd,e,f' point.

L(t) = ([a b c 1] + [d-a e-b f-c 0]·t)^T

(b) Write this P^3 line as a point-span matrix W:

W = [a b c 1]
**   [d e f 1]**

(c) Write this P^3 line as a plane-span matrix W*:

Find the 4-vectors for two different P^3 planes $\pi1$ and $\pi2$ that intersect along the line; then
$$W^* = \begin{bmatrix} \pi1^T \\ \pi2^T \end{bmatrix}$$

Both planes include the given points p1=[a,b,c]^T and p2=[d,e,f]^T, but to determine the two planes we need 3rd point for each of them that I will call p3,p4. To ensure (p1,p2,p3) and (p1,p2,p4) define different planes, p3 must not be collinear with p1,p2, and p4 must not be coplanar with p1,p2,p3.

There are many ways to choose p3 and p4. My favorite: swap the largest and smallest coordinates of the vector (p2-p1) to form vector v, but if the magnitude of these two swapped coordinate values are nearly the same, change the sign of one of them. Then define p3 as p1+v. To find the p4, use the cross product to find a vector perpendicular to the plane defined by p1,p2,p3: find v1 = (p2-p1)x(p3-p1), then define p4 = p1+v1.

Next, find the 4-vectors $\pi1^T$ and $\pi2^T$ using cross-products: (see page 47 in book)

$\pi1^T$ = [(p1-p3) x (p2-p3) | -p3^T·(p1 x p2)]
$\pi2^T$ = [(p1-p4) x (p2-p4) | -p4^T·(p1 x p2)]

(d) Write the line as a Plucker matrix P:

Recall that the Plucker matrix from two points A,B can be written as L= AB^T-BA^T:

```
[a][d e f 1]-[d][a b c 1]=[ad-da  ae-db  af-dc a-d]=[  0     ae-db  af-dc a-d]
[b]          [e]          [bd-ea  be-eb  bf-ec b-e] [db-ae    0     bf-ec b-e]
[c]          [f]          [cd-fa  ce-fb  cf-fc c-f] [dc-af  ec-bf     0   c-f]
[1]          [1]          [ d- a   e- b   f- c 1-1] [ d- a   e- b    f- c  0 ]
```

(e) Find the expression for the point at the intersection of this line with the P^3 plane given by $[g,h,j,k]^T$

Plane/Line intersection point X is easy to find with Plucker coordinates (page 52)

```
    X = L π =
[  0     ae-db  af-dc a-d][g] = [h(ae-db)+j(af-dc)+k(a-d)]
[db-ae    0     bf-ec b-e][h]   [g(db-ae)+j(bf-ec)+k(b-e)]
[dc-af  ec-bf     0   c-f][j]   [g(dc-af)+h(ec-bf)+k(c-f)]
[ d- a   e- b    f- c  0 ][k]   [g(d -a )+h(e -b )+j(f-c)]
```

3) **P3 Lines:** Why are Plucker matrices useful? Give an example where a Plucker Matrix formulation of lines is a better choice than a simple span W or W*.

Several reasons: Plucker matrices are quick to compute from planes or points (especially in the Plucker coordinates form) and allow us to find geometric properties more easily than spans. For example, given a point p and a line W (point span), to find the plane π that includes both requires us to find the null space of the matrix M made by stacking W on top of p^T (see pg. 50, or 'hard way' answer to problem 1D). This usually means running the SVD routine that can be expensive and sometimes inaccurate. But if we make the dual Plucker matrix L* instead, we find π by a simple matrix multiply: π = L* p. Similarly, Plucker matrix L multiplied by plane π gives the line/plane intersection point. Plucker matrices can also quickly test for coplanar lines (page 53).

4) **Constraints**: In P^3 space, what are the number of degrees of freedom (e.g. what is the rank) of:
   a) a point?
3-DOF; even though it is a 4-vector, projective coordinates are invariant to scaling, thus removing one degree of freedom.

   b) an ideal point?
2-DOF: all ideal points are on the π_∞ plane which constrains us to 2 dimensions

c) a line?

4-DOF: as shown in the book and in lecture notes, all lines can be uniquely defined by 2 points located within two non-parallel planes.

d) a plane?

3-DOF, because they are the duals of points, and thus are also scale-invariant.

e) the plane at infinity?

0-DOF. The π_∞ plane is [0,0,0,1], and no other plane is located there; remember, this vector scaled by a constant still refers to the same plane—scale invariance applies to planes as well as points.

f) a direction?

2-DOF; even though direction D=[x,y,z,0] is a different vector for every Cartesian point x,y,z, the scale invariance of P3 restricts removes the distinction between points that are collinear with the origin—thus direction D1=[0,0,1,0] and D2=[0,0,2,0] are the same direction.

g) a camera?

11-DOF (for planar perspective camera)12-element matrix less 1 DOF for scale invariance. 11DOF= 6 extrinsic + 5 intrinsic. Extrinsic: 3DOF for camera center position (translation t_x,t_y,t_z) + 3DOF for camera rotation/orientation (roll,pitch,yaw). Intrinsic: 2DOF for location of origin within the image plane (p_x,p_y) + 2DOF pixel size and aspect ratio (s, alpha) + 1DOF for focal length.

h) a pair of cameras bolted together on the same frame?
17-DOF: If we assume both cameras have identical intrinsic parameters, both cameras share the same 5DOF for them. One of the cameras has 6DOF for its extrinsic parameters (camera center position, camera rotation/orientation), and the other camera has a fixed position offset (3DOF) and a fixed orientation offset(3DOF) measured from the first.
   If the cameras use different intrinsic parameters, then add 5DOF more→22DOF.

Be sure to justify each answer. Single-number answers will receive only half credit; for full credit you must explain why you chose that number.

5) **Infinities:** In P^3 space, show that all lines include an ideal point, and prove that all lines intersect the plane at infinity.
All points on any line in P^3 can be written as a linear extrapolation between two points p1 = [a b c 1]^T and p2 = [d e f 1]^T. For every value of t there is a corresponding P^3 point on the line given by line(t) = p1 + (p2-p1)t.
   But P^3 is scale invariant; for any nonzero constant A, the P3 point given by A*line(t) is the same as line(t). Now suppose we let A =1/t; then A*line(t) = p/t + (p2-p1) = [a b c 1]^T /t + [d-a e-b f-c 0]^T. In the limit as t→∞, line(t) = [d-a e-b f-c 0], an ideal point.
   Like any other plane, the plane at infinity π_∞ = [0,0,0,1]^T includes a point p iff π_∞^T p = 0. We know every line includes an ideal point that can be written p_i=[a b c 0]^T, and every ideal point is included in the plane at infinity becauseπ_∞^T p_i = [0,0,0,1]^T[a b c 0] = 0. Thus every line contains at least one point p_i that is also part of the plane at infinity.

6) **Conics in P^3 to measure angles**: Consider two P3 planes named p1 and p2. After projective transformation by the 4x4 matrix H, these planes become p1' and p2' respectively. We don't know p1, p2, or the H matrix, but we *do* know the transformed absolute dual quadric Q*∞' and the transformed plane p1'. We also know that the plane p1 is perpendicular to p2. Show how to find p2' using equations and the information we have.

Recall that the absolute dual quadric detects perpendicular planes: p1 Q*∞ p2=0 if p1 ⊥ p2. Better yet, this property is invariant under transformation; p1' Q*∞' p2'=0 if p1 ⊥ p2. Because we are given p1' and Q*∞' we can just multiply them together and solve for p2' using the SVD: (p1' Q*∞') p2'=0

7) **P³ Camera**: Let integer values of (x,y) in the image plane of a camera define pixel, and construct a camera matrix P that meets these requirements:

--the field-of-view is 48 degrees horizontally, 36 degrees vertically;

--the image has 1280 pixels horizontally and 1024 scanlines vertically, with pixel (0,0) at the lower-left corner of the image, and the principal point at (640,512).

--the image has no affine distortion,

--the camera is located at (a,b,c) in R3 Cartesian coordinates. The principal point is an image of the point at (d,e,f) in Cartesian coordinates, and the P3 direction $[0,1,0,0]^T$ (e.g. the world-space y-axis direction) is always transformed to an image line parallel to the image's y axis.

Refer to slide 25 of lecture 13: the basic camera matrix P_0 can be written:

```
[αₓ f   s    pₓ   0]
[ 0   α_y f  p_y  0]
[ 0    0    1    0]
```

First, find the size of a pixel, given by α_x, α_y. Note that both are multipled by focal length f, so we can choose any f we want if we adjust α_x, α_y accordingly. I chose to use f=1. An image plane rectangle parallel to the x-y plane, centered on the z axis at z=1 that has a viewing size of 48 x 36 degrees will have a half-width w and half-height h given by:

w/1 = sin(48/2)/cos(48/2) = tan(24 degrees) = 0.4452

h/1 = sin(36/2)/cos(36/2) = tan(18 degrees) = 0.3249

This 2h x 2w-sized rectangle is divided into a grid of 1280x1024 pixels; a world-space point at (w,h,1) is (640,512) pixel-spacings away from the center of the rectangle: w α_x = 640, h α_y = 512, thus

f = 1.0, α_x = 1436.4 α_y = 1574.3

Next, we know world-space point at (0,0,1) corresponds to the principal point, and it has image coordinates 640,512:

p_x = 640, p_y = 512

We now have the 'basic' camera P_0 (its first 3x3 elements are the intrinsic param matrix K). The matrix expression P = K[R|-RC] positions the camera at world-space point C and then rotates the camera about its center by matrix R. Simply set C = $[a\ b\ c]^T$ and then find the R matrix that will aim the z axis in the direction given by [d-a e-b f-c]. Rotate first around the y axis by theta, then about the rotated x axis by phi:

theta = arctan2((f-c)/(d-a)) –90degrees (-90 degrees to aim the z axis instead of the x axis)

phi = arctan2((e-b)/sqrt((d-a)²+(c-f)²))

R = R_s(theta)R_y(phi), C = [a b c], and thus P = K[R | -RC].

8) **Epipolar Geometry :**

a) Construct a pair of camera matrices P0, P1. Matrix P0 is the basic camera P_0 from class lecture notes after it is translated to [0,0,-1] in Cartesian R3 space. Matrix P1 is a companion camera, made by moving an exact copy of the P0 camera to place its camera center at (+2, +1, 0) in R^3. The P1 camera was then rotated, first about its y axis so that the principal axis intersects the world-space y axis, and then about the camera's x axis so that the principle axis intersects the origin.

```
Basic K= I= [1 0 0];  if moved to position 'Cam' and rotated by R, then the
            [0 1 0]      camera matrix becomes P = K[R|-RCam]
            [0 0 1]
```

```
P0=[1 0 0 0]   Put the P1 camera center at (2,1,0). Next, twist it around Y axis
   [0 1 0 0]   to aim it's principal axis (z axis) at the world-space Y axis;
   [0 0 1 1]   this is a -90 degree rotation: Ry(-90). Next, rotate around
```

```
                   the camera's x axis to aim at the world-space origin. The
                   rotation amount is:
                   arctan2(1/2) = θ =  26.565 degrees: let S = sin θ = 0.4472
                                                           C = cos θ = 0.8944
   R = Rx(26.5) Ry(-90) =   [1  0  0][ 0  0  1]  =  [ 0  0  1]
                            [0  C -S][ 0  1  0]     [ S  C  0]
                            [0  S  C][-1  0  0]     [-C  S  0]
```

Next, find −RCam, as Cam=$[2,1,0]^T$:
```
                       -RCam = [0, -(2S+C), -(S-2C)] = [0 -1.789 1.342]
```
Finally, assemble P1 = K[R|−RCam];   we know K=I, so P1 = [R|−RCam]

```
P1=[ 0    0    1     0    ] = [ 0.0      0.0      1   0.0  ]
   [ S    C    0  -(2S+C)]    [ 0.4472 0.8944   0  -1.789]
   [-C    S    0   2C-S  ]    [-0.8944 0.4472   0   1.342]
```

b) Find the fundamental matrix F for this pair of cameras.

Refer to lecture 14, slide 18.
If we define P0 as P' and P1 as P, we can write $F = [e_0]_x P_0 P_1^+$, where e_0 is the is the camera center C for camera P1 as seen in the image plane of camera P0. By inspection, $e_0 = [2,1,1]$ (remember, e_0 is in P2 coords, not R3!). We already know camera P0, but the pseudo-inverse of P1 is messy, and may require you to do a little programming in C or MATLAB: $P_1^+ = P_1^T (P_1 P_1^T)^{-1}$. My code yields:
```
P₁⁺  =      [0 -0.149 -0.447] and P₀P₁⁺ =    [0 -0.149 -0.447]
            [0  0.596  0.671]               [0  0.596  0.671]
            [1  0.0    0.0  ]               [1 -0.298  0.224]
            [0 -0.298  0.224]
```
Then make a skew-symmetric matrix $[e_0]_x$ out of e_0, do matrix multiply, and get fundamental matrix:

$F = [e_0]_x P_0 P_1^+$ =
```
        [ 1 -0.894 -0.447]
        [-2  0.447 -0.894]
        [ 0  1.342  1.789]
```

c) Both cameras form an image of an ordinary sharpened pencil. In the 2D image from camera P0, we found the tip of the pencil is at point x0 = [0.5, 0.5]. Find the corresponding epipolar line in the image from the P1 camera; (write it as a P2 line).

We know that $x'^T F x = 0$, and $F^T x' = L$. Using terms of this problem, $F^T x0 = L1$, where x0 is the P^2 point $[0.5\ 0.5\ 1]^T$ and L1 is the epipolar P^2 line in the camera 1 image. (See lecture 14 slide 21, or book, pg. 226). Do matrix multiply and get:
$L1 = [-0.5\ 1.118\ 1.118]^T$

d) Along the epipolar line in the image from camera P1, we searched and found the tip of the pencil again at point x1; by chance, it was the point closest to the principal point of the image. Compute x1.

The principal point is the origin in P^2 image space for both cameras. Thus the point x1 is the point where the epipolar line L1 is closest to the origin. We can find this point by considering the vector from the origin to the nearest point on the line L1; it will be a unit-length normal vector perpendicular to the line, scaled by the line's distance from the origin. Given the P2 line, the normal vector and the distance-to-origin can be found almost by inspection:
**   Remember that P^2 lines are scale-invariant; we can scale L1 by any nonzero constant and it still represents the same line. Suppose we scale L1=$[a\ b\ c]^T$ so that the 2-vector [a,b] has unit length—in other words, divide every element in L1 by sqrt(a^2+b^2), and call the result L1'.**
** $L1' = [a'\ b'\ c']^T = [a\ b\ c]^T/sqrt(a^2+b^2)$.**
Now suppose we have a point x1 = $[d\ e\ 1]^T$ that is on line L1': then $L1'^T x1 = 0$, or (a'd +b'e) +c' = 0. Note that the part in parentheses is an R^2-space dot-product between the point x1 and the unit-length vector [a' b'] from the origin towards the line. If the point x1 is on the line L1', then the dot product gives us the min. distance from the line to the origin. To keep the equation true, (-c') must be that

distance. Thus to find the point on the line L1' that is nearest to the origin, just multiply: [a',b']c'.
Apply this trick to the L1 from part c):
   L1'= [-0.5 1.118 1.118]^T / sqrt((-0.5)^2+(1.118)^2) = [a' b' c'] = [-0.408 0.913 0.913]
Nearest point x1 = [a' b']c' = [-0.373 0.833]

e) From the image-space points x0 and x1, compute the 3D position X of the pencil point.
The world space point X is at the intersection of two world-space rays:
   --one starts from camera P0 center at [0,0,-1], passes through image point x0 and out to X
   --another starts from camera P1 center at [2,1,0] and passes through x1 and out to X.
We can describe each ray parametrically, by defining a 'ray(t)' function for each camera that uses parameter 't' to select a point along a ray.   At t=0 the ray starts at the camera center point C: ray(0) = C. As 't' increases, the point ray(t) moves outwards through the image plane at point x0 or x1, continues out into the scene, passes through the world-space point X, and as t→∞ the ray hits the infinity plane. Finding world-space coordinates for image points x0 or x1 is a mess—instead, we use the direction towards the point at infinity. As shown in the book and in Slide 28 of Lecture 13, the world-space 'direction' D for any chosen image point x_d is found by: D = (KR)$^{-1}$ x_d, where K is the intrinsic camera matrix, and R is rotation matrix used to position the camera. You can make a ray function by treating D as a direction vector in Cartesian (R3) coords:
   For camera 0: $ray_0(t_0)$ = [0,0,-1] + D0·t_0 where D0 = $(K_0R_0)^{-1}$ x0,
   For camera 1: $ray_1(t_1)$ = [2,1, 0] + D1·t_1 where D1 = $(K_1R_1)^{-1}$ x1.
From solutions above, we know K_0R_0 is the identity matrix (basic camera, no rotations); thus D0=x0;

```
ray₀(t₀) = [0,0,-1] + [0.5,0.5,1]·t₀
```

For the other camera's ray, K_1R_1 = R_1 = Rx(26.5) Ry(-90) =   [0   0   1]
                                                                 [0.4472 0.8944 0]
                                                                 [-0.8944 0.4472 0]

```
R₁⁻¹ x1 = [0 0.447 -0.894][-0.373] = [-0.522] = D1
          [0 0.894  0.447][ 0.833]   [ 1.193]
          [1  0      0    ][ 1.0  ]   [-0.373]
```
Thus
```
ray₁(t₁) = [2,1, 0] + [-0.522, 1.193, -0.373]·t₁
```

```
and the point X is at the intersection of these 2 rays: find the t0,t1 pair that
satisfies ray₀(t₀) × ray₁(t₁) = 0. We have 3 equations and 2 unknowns (t0, t1);
rearrange as a null-space problem, stack, and use SVD to solve—it will give a
least-squares best-fit solution.
```

9) **Photometry:** Write the BRDF function of a purely diffuse gray material that re-radiates exactly half of all the incident light energy it receives, and radiates that light equally in all directions. Be sure to express your answer in the proper units (e.g. 1/steradians).
GREAT explanation of BRDF: www.cs.huji.ac.il/~danix/advanced/RenderingEq.pdf
Recall that:
   Flux = Φ = rate of energy transfer; photons/second, watts, joules/second, etc.
To measure all the light energy arriving at point x on a surface, use irradiance:
   Irradiance =E(x)= Flux/Area = Watts/m^2. We can't really measure irradiance at a point x—we have to measure flux in a small area around x, then divide by area. In the limit as the area around x shrinks to zero we have:
       E(x)= dΦ ← the infinitesimal flux that we measured
           dA ← within an infinitesimal area.
But photons do not all travel in the same direction. To describe light energy arriving at point x from a direction w, we define radiance L(x,w):
   Radiance = L(x,w)=Flux/(area·solid angle) = Watts/(m^2·sr). Once again, we can't really measure radiance at a point x—we have to measure flux in a small area around x, and measure only the light

within a small bit of solid angle around w, then divide by both the area and the solid angle used. In the limit as the area around x shrinks to zero AND the solid angle shrinks to zero we have:

$$L(x,w)= \frac{d^2\Phi}{dA_\perp \cdot dw}$$

$\leftarrow$ **the infinitesimal flux that we measured**

$\leftarrow$ **within an infinitesimal area*, from an infinitesimal solid angle.**

"$A_\perp$" means this area is measured perpendicular to angle w.

When radiance L(x,w) is used to measure light at a surface, it is more convenient to measure area dA on the surface itself rather than measure area $dA_\perp$ on an imaginary plane perpendicular to the incoming light angle w. Measuring area dA on the surface yields $dA_\perp = (\cos\theta)dA$, where θ is zero in the surface normal direction and 90degrees when tangent to the surface. The $\cos\theta$ term is the foreshortening effect we discussed in class; the flux received by a surface is nearly zero for incoming lighting angles near 90 degrees. Re-written in these terms, radiance is:

$$L(x, \theta, \phi) = \frac{d^2\Phi}{dA \cdot dw \cdot \cos\theta}$$

$\leftarrow$ **the infinitesimal flux that we measured**

$\leftarrow$ **within an infinitesimal surface area,**

from an infinitesimal solid angle.

BRDF for a surface is a ratio; BRDF= (outgoing or 'emitted' radiance) / (incoming irradiance)

Note that incoming irradiance is entirely from one direction (θ_i, ϕ_i).

$$BRDF(\theta_{in}, \phi_{in}, \theta_{out}, \phi_{out}) = L_{out}(x, \theta_{out}, \phi_{out}) / E_{in}(x, \theta_{in}, \phi_{in})$$

Now, to construct our BRDF:

--The flux at a point on our surface is $\Phi_{in} = E_{in}\cdot dA$.

--Half this flux is absorbed by the material, and the other half is re-emitted: $\Phi_{out} = 0.5\cdot E_{in}\cdot dA$.

--The outgoing flux is spread out evenly over a hemisphere of solid angle. An entire sphere describes 4π steradians of solid angle, so a half-sphere is 2π steradians. Thus the outgoing flux in an infinitesimal bit of solid angle is:

$$\frac{d\Phi_{out}}{dw} = \frac{0.5\cdot E_{in}\cdot dA}{2\pi} = \frac{E_{in}\cdot dA}{4\pi}$$

--If we also divide through by the infinitesimal surface area dA we get outgoing radiance:

(Why don't we have a $\cos\theta$ term? because we defined the outgoing radiance as uniform in all directions.)

$$\frac{d\Phi_{out}}{dA\cdot dw} = \frac{E_{in}}{4\pi \text{ sr}} = L_{out}$$

$$\boxed{BRDF = L_{out}/E_{in} = 1/(4\pi \text{ sr})}$$ (sr==steradians)

10) **Discussion/Opinion Question:** In the EGRW2001 paper "Polyhedral Visual Hulls for Real-Time Rendering" (Week 9 reading assignment; handed out & on website) Matusik et al. show how to make each 3D polygon of a visual hull by 2D projection and 2D clipping. (They also describe a way to accelerate this process by 'edge bins', but ignore that for this question).

a) give a step-by-step description of this method, sufficient to implement it in Project D.

In their method, silhouettes are broken into line segments within each image. Each line segment defines a pair of lines that extend from the camera center through the line segment endpoints and outwards to infinity. These two world-space lines are then transformed into image space by the camera matrix for a different image in the image fan, forming a 2D wedge or triangle in that image space. This wedge is then clipped against all the silhouette edges in the image to form a 2D polygon. This 2D polygon is then projected back onto the plane of the original 3D line pair to form a portion of a closed polyhedral boundary.

b) Discuss the advantages and difficulties you might encounter if this method were used as a replacement for Project D.

This is far more efficient that ProjD's exhaustive point-by-point test because it only computes the boundaries of the polyhedra , and all tests are done in 2D instead of 3D. However, it is much more complicated to implement because it requires a geometric boundary representation (polygons of variable complexity) for each silhouette instead of a simple plane, it requires projection from 3D to 2D and then back to 3D to construct the polyhedra, and it requires robust, reliable clipping code for arbitrary 2D polyhedra—not an easy piece of code to write!