CS 184/284A

Spring 2019: Midterm 1

March 19th, 2019

Time Limit: 110 Minutes

Name: Ran Liao

SID number: 3034504227

Room: Dwinelle 145

- This exam contains 18 pages (including this cover page) and 5 problems. Check for missing pages.
- Put your initials on the top of every page, in case the pages become separated.
- This exam is closed book, except for one 8.5 × 11 page of notes (double sided), printed or handwritten.
- This exam is 110 minutes long, and has a total of 110 points.
- Problem difficulty varies throughout the exam, so don't get stuck on a time-consuming problem until you have read through the entire exam. Each problem's point value is roughly correlated with its expected difficulty.
- Answer each question in the space provided. Partial credit may be given on certain problems.
- To minimize distractions, do your best to avoid questions to staff. If you need to make assumptions to answer a question, write these assumptions into your answer.
- **For multiple choice questions, please fill the little bubble completely next to your answer. Do not just tick or circle.**

Problem	Points	Score
1	22	
2	24	
3	20	
4	22	
5	22	
Total:	110	

1. (Total : 22 points) True / False

Mark each statement true or false – indicate clearly. (1 point each)

(a) (1 point) T A raster display is one that traces out lines and curves, like an oscilloscope.

(b) (1 point) T Any polygon can be represented by a collection of triangles.

(c) (1 point) F When doing supersampled rasterization, the samples covered by each triangle are "averaged down" *before* they are stored into the frame buffer.

(d) (1 point) F Filtering (blurring) a function after sampling it is an effective way to remove aliasing.

(e) (1 point) T Any continuous signal can be represented as a combination of sines and cosines.

(f) (1 point) F Mipmapping is a technique for reducing aliasing due to texture *magnification.* minfica

(g) (1 point) F Translation is an example of a linear transformation.

(h) (1 point) T Z-buffering, which has $O(n)$ complexity for n triangles, is more common in graphics pipelines today than the painters algorithm, which has $O(n \log n)$ complexity.

(i) (1 point) F With Bézier splines, we commonly have to use higher-degree polynomials to represent more complex curves.

(j) (1 point) F Changing a mesh's topology necessarily changes the shape of its surface.

(k) (1 point) F Catmull-Clark and Loop subdivision surfaces provide similar smoothness guarantees to cubic splines (away from extraordinary vertices).

(l) (1 point) T The radiometric power incident on a surface point is called the irradiance.

(m) (1 point) T The luminance due to infrared light (longer in wavelength than the visible spectrum) is always zero.

(n) (1 point) F An isotropic light that puts out 800 lumens has a radiant intensity of 800 / 4π candelas.

$E(f(x)) = E$

(o) (1 point) T The expected value of evaluating $f(x)$ at a random point x drawn from a uniform random variable between a and b is equal to the area under curve of $f(x)$ between a and b, divided by $(b - a)$.

(p) (1 point) F When using importance sampling in Monte Carlo integration, we want to draw from a probability density function that is as dissimilar as possible from the function we are trying to integrate.

(q) (1 point) F In a KD-Tree, every triangle must appear in exactly one leaf node.

(r) (1 point) T In a BVH, every triangle must appear in exactly one leaf node.

(s) (1 point) F We used Russian Roulette in Monte Carlo estimates of direct lighting.

(t) (1 point) T The BRDF of a perfectly diffuse material is a constant function.

(u) (1 point) F Light sources cannot reflect light in path tracing.

(v) (1 point) T Point light sources create sharp shadows.

RL

Blank page – this is scratch space for you.

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 2 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 2 \\ \not{0} & \not{0} & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & -1 & 2 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

2. (Total : 24 points) Graphics Pipeline

(a) (2 points) Consider a ceiling fan with 6 identical blades. If we record with a video camera at 120 frames per second (assuming no motion blur), up to what rate can it spin before we will begin to see aliasing?

(iv)

(i) 60 rotations per second.

(ii) 40 rotations per second.

(iii) 20 rotations per second.

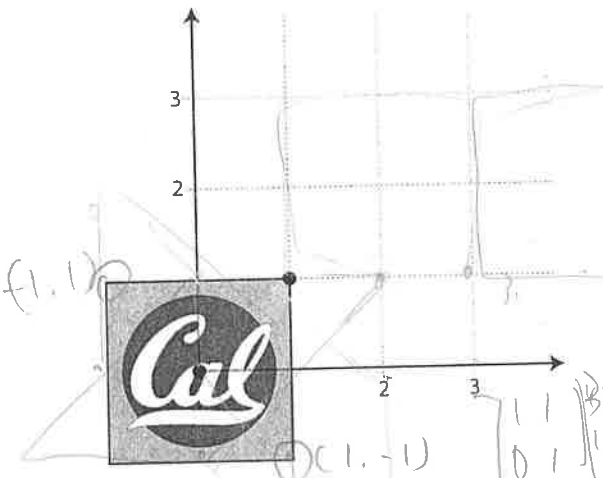
[✓ (iv)] 10 rotations per second.

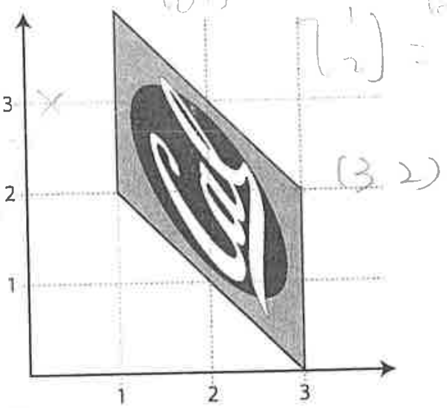
(v) 5 rotations per second.

$120 = 6 \times 2 \times X$

$10 = X$

(b) (5 points) Consider these two images:





Which sequences of elementary transformations, applied to the image on the left, would produce the image on the right? (Fill in all that apply.)

(Note: $\text{shear}_x(s) = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}$ and $\text{shear}_y(s) = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}$)

~~(i)~~ translate(2, 2) → $\text{shear}_x(1)$ → rotate(−270) X

[✓ (ii)] $\text{shear}_x(1)$ → rotate(90) → translate(2, 2)

~~(iii)~~ translate(2, 2) → $\text{shear}_y(1)$ → $\text{shear}_x(-1)$ X

(iv) $\text{shear}_y(1)$ → $\text{shear}_x(-1)$ → translate(2, 2)

(v) translate(4, −2) → rotate(−270) → $\text{shear}_y(-1)$ X

$\boxed{\begin{matrix}(ii)\\(iv)\end{matrix}}$

(1, −1)

(3, 1)

RL

(c) (5 points) Write down a 3 × 3 matrix representing the affine transformation from question 2(b) in homogeneous coordinates.

translate(2,2) shearx(-1) sheary(1)

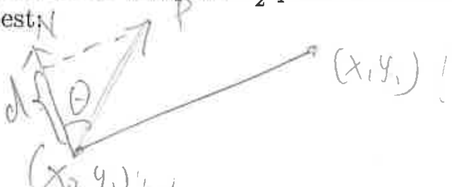
$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -1 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 2 \\ 1 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

(d) (6 points) Let's add line drawing to our rasterizer! One way we could do this is with a similar strategy to what we used for triangles:

```
// assume the line segment goes from (x0,y0) to (x1,y1)
for each pixel px, py {
  if (pixel_covered_by_line(px+0.5, py+0.5, x0, y0, x1, y1)) {
    fill_pixel(px, py);
  }
}
```

Assuming we want to draw lines as one pixel thick (they cover samples ±$\frac{1}{2}$ pixel from the center of the line), implement a point-line coverage test:



```
bool pixel_covered_by_line(float px, float py,
                           float x0, float y0,
                           float x1, float y1)
{
  Vector2D n = {-(y1-y0), x1-x0};
  Vector2D p = { px-x0, py-y0 };
  double cos_theta = dot(n, p)/n.norm()/p.norm();
  double d = p.norm() * cos_theta;
  return abs(d) <= 0.5
}
```

(Hint: consider the relationship between a line and the edge equations we used for rasterization.)

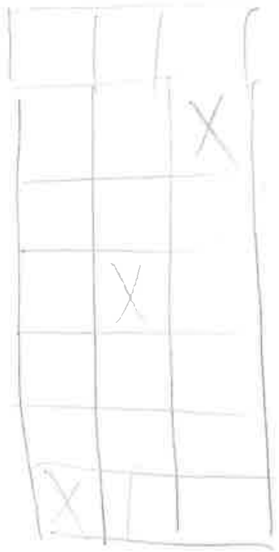
(e) (4 points) Reflecting on our solution to 2(d), we realize that it seems inefficient: even if we restrict our rasterization loop to only test pixels inside the bounding box of the line, we're still testing a whole area just to fill a thin line through it. (We're doing up to $O(n^2)$ tests to fill $O(n)$ pixels.) Thinking about this, we note that it should be possible to walk incrementally along the line and directly fill covered pixels in linear time.

Starting from the following sketch, fill in the missing details for a basic incremental line rasterizer:

```
// Assume float x0, y0, x1, y1 are in scope as arguments, and x1 > x0.

int start_x = x0

int end_x = x1

float y = y0

for (int px = start_x; px <= end_x; px++) {

  int py = y

  fill_pixel(px, py);

  y += (y1 - y0) / (x1 - x0);

}
```

$$\frac{y_1 - y_0}{(x_1 - x_0)}$$

(f) (2 points) Thinking further about 2(e), we realize its behavior is somewhat sensitive to the slope of the line we are drawing. This algorithm works well for lines with slopes between ±45°, but explain in one brief sentence: what artifacts would we observe if we tried to draw a line with a steeper slope?



Jaggy like this

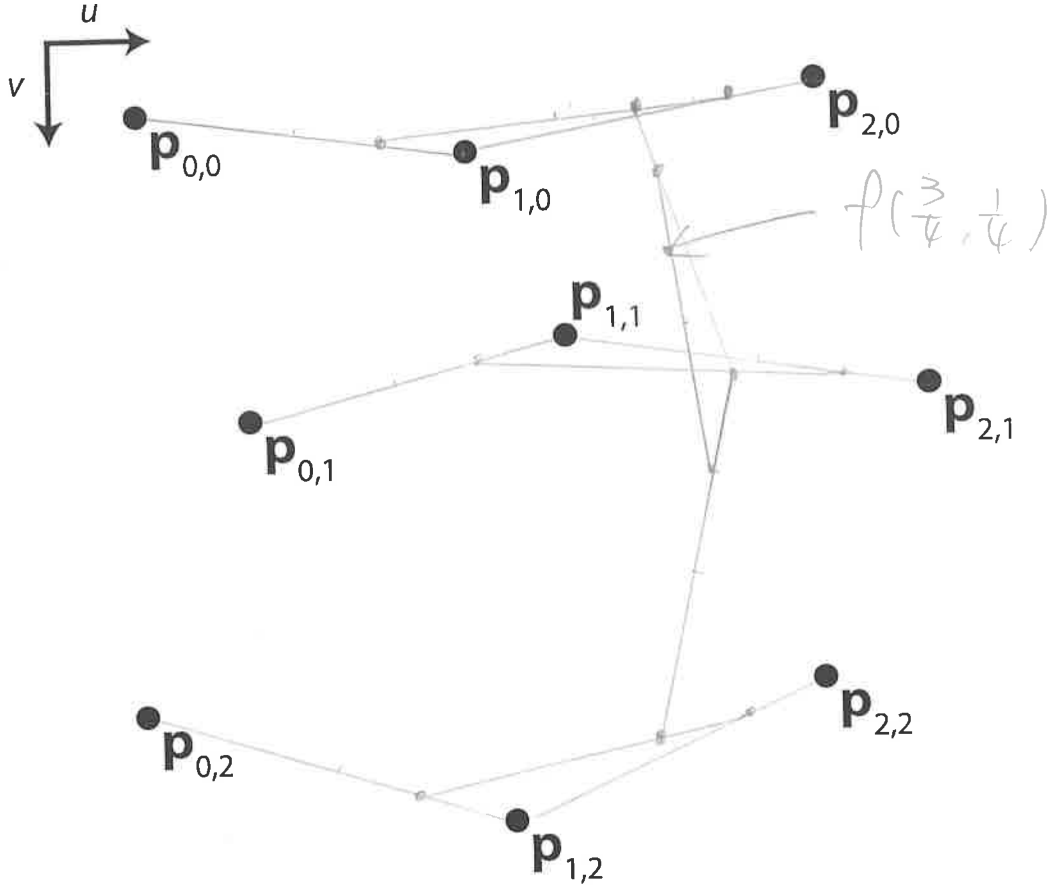
RL

Blank page -- this is scratch space for you.

$$\frac{3}{4}\left(\frac{3}{4}P00 + \frac{1}{4}P10\right) + \frac{1}{4}\left(\frac{3}{4}P10 + \frac{1}{4}P20\right)$$

3. (Total : 20 points) Geometry

(a) (6 points) In the figure below, $\mathbf{p}_{0,0}$ through $\mathbf{p}_{2,2}$ are the control points of a quadratic Bézier patch $f(u, v)$ $(0 \leq u, v \leq 1)$. Draw the de Casteljau algorithm derivation of the point $f(\frac{3}{4}, \frac{1}{4})$.



(b) (6 points) Compute the coefficient for the total contribution of $\mathbf{p}_{1,1}$ to $f(\frac{3}{4}, \frac{1}{4})$. (Note: $\frac{3}{4} = 1 - \frac{1}{4}$ and vice versa, so there are some symmetries and repeated terms you might exploit to simplify your computation.)

$\frac{6}{16} + \frac{3}{16}$

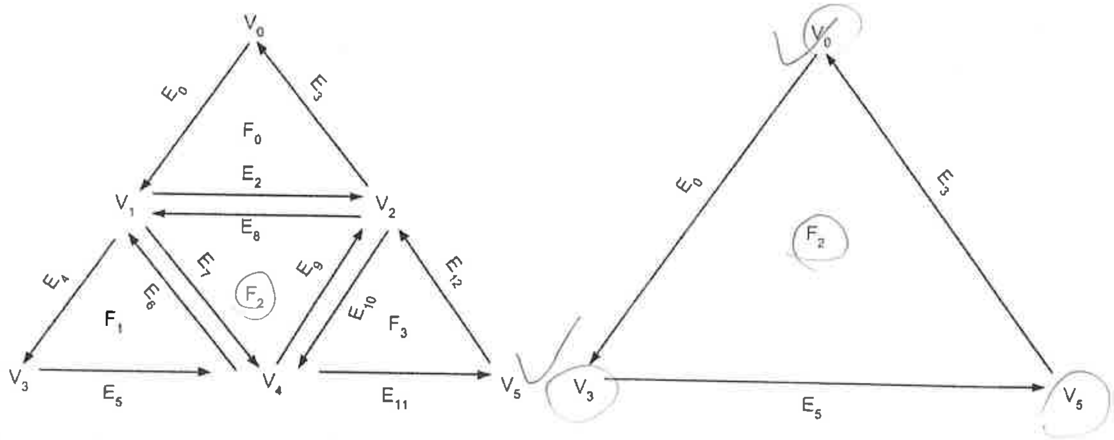
$\frac{6}{16}$ $\frac{3}{8}$ $\overline{64}$

$$\left[\frac{3}{4} \cdot \frac{1}{4} + \frac{1}{4} \cdot \frac{3}{4}\right]^2$$

$$= \frac{9}{64}$$

RL

(c) (4 points) Consider the following initial (left) and final (right) half-edge meshes.



For simplicity, use halfedge(), vertex(), face(), twin(), next() as the calls to both get and set those items, given the appropriate pointer. For example, given a halfedge pointer E_0, you can set its twin to some other halfedge E_1's vertex by writing:
$E_0 \rightarrow$ twin() = $E_1 \rightarrow$ vertex()

You are initially only given a pointer to the halfedge E_0. How would you correctly assign E_0's new face, using the fewest number of calls? Assume you cannot access any elements that are not shown. Write this in one line.

$E_0 \rightarrow$ face() = $E_0 \rightarrow$ next() $\rightarrow$ twin() $\rightarrow$ face()

(d) (4 points) Again considering the above meshes, what is the fewest number of elements that must have their halfedge() reassigned in order to *guarantee* that we get the correct mesh on the right? Assume there are no changes outside of the parts of the mesh shown.

2, one for V_3 one for F_2

Blank page – this is scratch space for you.

$y = -t + 1$

$\int_0^x -t+1 \, dt$

$= \left(-\frac{1}{2}t^2 + t\right)\Big|_0^x$

$= -\frac{1}{2}x^2 + x$

$V = -x^2 + 2x$

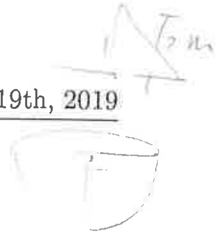
$= x(-x+2)$

$x^2 - 2x + V = 0$

$\frac{2 \pm \sqrt{4-4V}}{2}$

$\boxed{1 \pm \sqrt{1-V}}$

RL



4. (Total : 22 points) Monte Carlo Integration and Radiometry

(a) (4 points) Irradiance

Consider a point light source that outputs 100 watts of light uniformly into the hemisphere below it. Calculate the irradiance (including correct units) falling onto the floor, at a point 1 meter below and 1 meter to the side of the light source.

~~$\frac{100}{2\pi(\sqrt{2})^2}$~~

$$\frac{100\ W}{2\pi(\sqrt{2}\,m)^2} = \frac{100\ W}{4\pi\ m^2}$$

$$= \frac{25}{\pi}\ W/m^2.$$

(b) (2 points) Photometric Units

Which of these units is not the same as the others?

(iii)

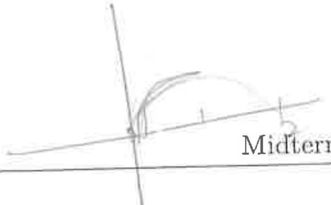
(i) Nit

(ii) Candela / meter^2

(iii) Lux * meter^2 / steradian

(iv) Lumen / steradian / meter^2

(v) None of the above



$x($

$ax^2+bx+C=0$

(c) (6 points) Random Sampling

$(0\ 0)$

$(1\ 1)$

$y=ax^2+bx$

$2ax+b=-1$

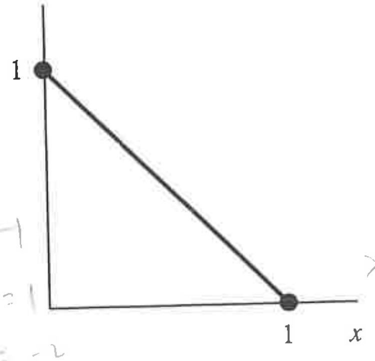
$2a+b=-1$

$ax^2+bx=0$

$a+b=1$

$\boxed{a+b=1}$

$a=-2$



$x^2-2x+2V=0$

$\frac{2\pm\sqrt{4-8}}{2}$

pdf $= -\cancel{t}+1$

cdf $= \int_0^x -t+1\,dt$

$x(-\frac{1}{2}x+1) = (-\frac{1}{2}t^2+t)\Big|_0^x$

$= -\frac{1}{2}x^2+x = V$

$x=?$

You are writing a program to do Monte Carlo integral estimation of a function $f(x)$ for x between 0 and 1, and you want to use importance sampling. You have a hunch that $f(x)$ decreases approximately linearly to 0, so you decide to do importance sampling with a probability density function proportional to the function graphed above. Given v, a uniform random value between 0 and 1, derive an expression for a random value x (as a function of v) drawn from the desired probability density function.

$2x^2-x$

$x(2x-1)$

$-2x^2+3x$

pdf: $-t+1$

cdf: $-x^2+2x$

Let ~~x^2+~~ $x^2-2x+V=0$

$x = 1\pm\sqrt{1-V}$

Hence x is $1-\sqrt{1-V}$

(d) (2 points) Solid Angle

The magical planet of Solan has a radius of 1000 kilometers and is completely transparent. You take the elevator down to the very center of the planet and look up at the surface. I draw a disk of radius 1 kilometer on the surface. What is the solid angle, in steradians, of this disk from your perspective?

$$\frac{\pi(1000)^2}{4\pi(1000000)^2} = \frac{X}{4\pi}$$

$$X = \frac{\pi}{1000\,000}$$

RL

(e) (8 points) Monte Carlo Estimators

Which of the following is an unbiased Monte Carlo estimator for the given integral? Place a check mark in the appropriate column. The first two rows are completed as examples for you. In this question, H^2 denotes the hemisphere, and S^2 denotes the sphere.

	Definite Integral	**Estimator**	**PDF**	**Unbiased**	**Biased**
	$\int_a^b f(x)dx$	$f(X_i)$	X_i drawn uniformly at random from $[a,b]$		**X**
	$\int_a^b f(x)dx$	$\frac{b-a}{N}\sum_{i=1}^{N} f(X_i)$	X_i drawn uniformly at random from $[a,b]$	**X**	
(a)	$\int_{H^2} f(\omega)d\omega$	$\frac{2\pi}{N}\sum_{i=1}^{N} f(\omega_i)$	ω_i drawn uniformly at random from H^2	X	
(b)	$\int_{S^2} f(\omega)d\omega$	$4\pi f(\omega_i)$	ω_i drawn uniformly at random from S^2		X
(c)	$\int_{S^2} f(\omega)d\omega$	$\frac{4\pi}{N}\sum_{i=1}^{N} \frac{f(\omega_i)}{p(\omega_i)}$	ω_i drawn from pdf $p(\omega_i)$ over sphere S^2	X	
(d)	$\int_{-1}^{1}\int_{-1}^{1} f(x,y)dxdy$	$\frac{4}{N}\sum_{i=1}^{N} f(X_i,Y_i)$	X_i and Y_i drawn uniformly from $[-1,1]$	X	

Blank page – this is scratch space for you.

RL

5. (Total : 22 points) Rendering

(a) (8 points) Ray Tracing Acceleration Structures

Consider the following pseudocode from lecture for the recursive intersect function for the given ray against the given Bounding Volume Hierarchy (BVH). The function returns a value t for the closest intersection.

```
 // return a value t for the closest intersection, where
 // ray = origin + t * direction;
 //
Intersect (Ray ray, BVH node)
   if (node.bbox.intersect(ray) < 0) return; // bbox is bounding box
   if (node is a leaf node)
      test intersection with all objs;               null
      return closest intersection;
   //TODO: Change the code below:
   hit1 = Intersect (ray, node.child1);           ~~hit~~
   hit2 = Intersect (ray, node.child2);
   return closer of hit1, hit2;
```

Cross out the lines following the TODO comment of the pseudo-code above, and write on the next page an optimized psuedo-code replacement for those three lines that tests the BVH's child sub-trees in the optimal order and avoids testing the other sub-tree if not necessary. Careful! In considering whether it is necessary to test the other sub-tree, remember an important fact about the relative bounding boxes of a BVH node's child sub-trees.

You may use the three helper functions on the bounding box and ray that are shown by example in the following lines of code:

```
// returns the t-value corresponding to the closest intersection
// for the given ray, or a negative value if no intersection
float hit = bbox.intersect(ray);

// returns a boolean value for whether the given point p is
// inside the given bounding box
bool inside = bbox.contains(p);

// Gives a point p along the given ray at the given t-value.
// i.e. returns ray.o + t * ray.d;
Point p = ray.point(t);
```

[Please go to the next page to write your psuedo-code answer to this question]

Your answer (replacement for the lines of crossed-out pseudocode):

```
// return a value t for the closest intersection, where
// ray = origin + t * direction;
//
Intersect (Ray ray, BVH node)
  if (node.bbox.intersect(ray) < 0) return; // bbox is bounding box
  if (node is a leaf node)
     test intersection with all objs;
     return closest intersection;
  //TODO: Change the code below:
```

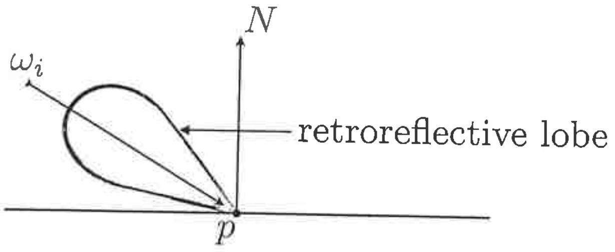
```
~~float hit1 = node.child1.intersect(ray);~~
~~if (hit1 >= 0) {~~
~~     point p = ray.point(t).~~
~~     if~~

if ( node.child1 bbox.contains( ray.o) )
     return Interset (ray, node.child1 )
else if (node.child2 .bbox. contains (ray.o) )
     return Intersed ( ray, nodechild2 ) ;
else
     hit1 = Intersect (ray, node. child1)
     hit2 = Intersel (ray. nodo.child2)
     return closer of hit1 and hit2
```

RL

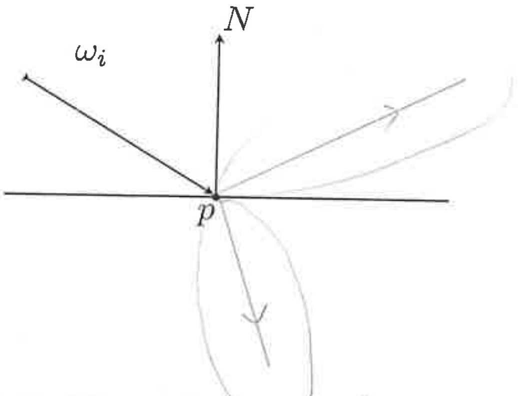
(b) (6 points) Material functions.

For surfaces with the BRDFs described below, consider a single incident ray of light as shown from direction ω_i to a surface point p with indicated surface normal vector N. Please sketch on these drawings the reflected distribution of light at point p. The first is done as an example for you.

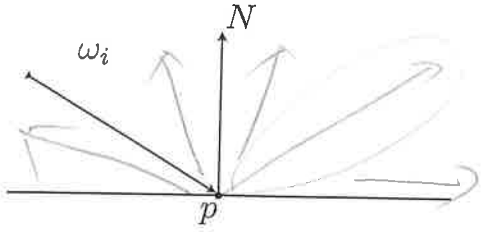
Example: BRDF: glossy retroflection.



i. BRDF: a material with both specular reflection and specular refractive transmission, such as glass.



ii. BRDF: a material with both diffuse and glossy specular components, such as sand-blasted aluminum.



(c) (8 points) Debugging A Path Tracing Implementation

Your friend is trying to debug his physically-based raytracing code, but something strange is happening. He is doing path tracing, except that rather than using Russian Roulette to terminate paths, he is cutting off the recursion after N bounces. Specifically, he calculates the total radiance by summing ZeroBounceRadiance + AtLeastOneBounceRadiance, and terminating the recursion at depth N. The strange thing is that as he increases N, the image keeps getting brighter, more quickly as N gets larger. Which of the following could be the problem (clearly indicate all that apply):

~~(i)~~ Missing $\cos\theta$ term for Lambert's law.

~~(ii)~~ Missing factor of $\frac{1}{4\pi}$ on point light sources from total power to radiant intensity.

[✓ (iii)] Missing factor of $\frac{1}{\pi}$ in computing diffuse surface BRDF value from given albedo (fraction of incident light that is reflected from diffuse surface) of 0.8.

[✓ (iv)] Missing normalization of importance sampling PDF to ensure the PDF integrates to one.

(iii)

(iv)