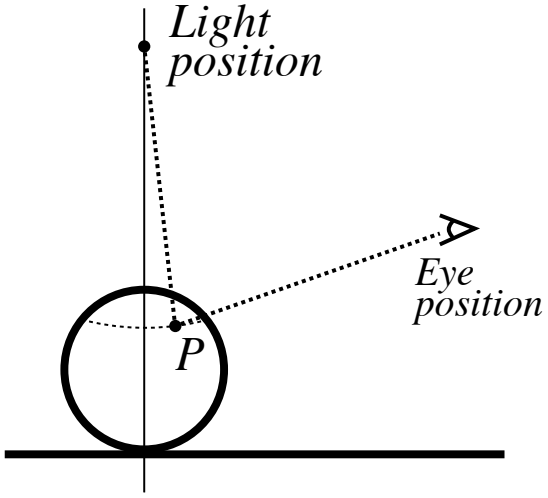
Stanford CS248: Interactive Computer Graphics
Participation Exercise 4

Problem 1

Give the full premultiplied alpha representation of 20% opaque blue and 50% opaque cyan. (Note without considering transparency, blue is [0,0,1] and cyan is [0,1,1]) Then provide the premultiplied alpha representation of the result of compositing 20% blue OVER 50 opaque cyan OVER fully opaque white.

Problem 2

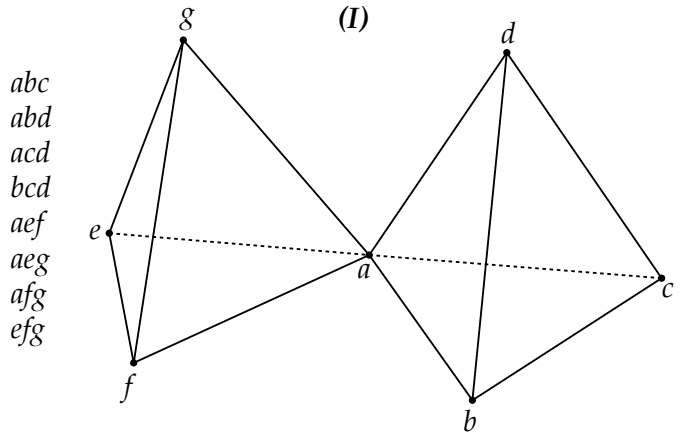
A. Consider a room with single *point light source* at the center of the ceiling. The light is shining on a sphere lying on the floor directly below it. **This sphere has a diffuse brdf, which means it scatters incoming light equally in all directions**. Now imagine a person staring at a *single point P* on the sphere as illustrated below. The person then bends their knees to lower their head, but continues to stare at the same point on the sphere. Does the appearance of the sphere change with the change in head position? Why or why not? (Note that moving one's head does not change the position of the sphere or the position of the light.)

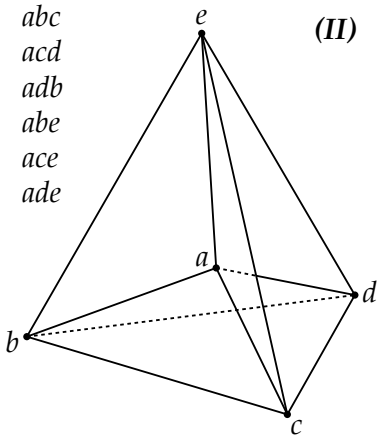


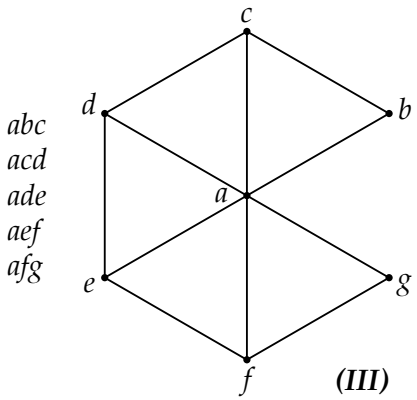
B. Now consider the same setup as in the previous problem, but now the surface of the sphere is a *perfect mirror reflector*. The person bends their knees to the point where the reflection of the light source in the sphere is visible to the eye. Now the person walks around the sphere in a circle of equal radius, keeping their head at exactly the same height, and continuing to look at the closest point on the sphere. (As the person walks, the closest point traces out a circle, as shown by the dotted line on the sphere in the figure from the previous problem.) Does the appearance of the sphere change as the person walks around? Why or why not?

Problem 3: Meshes and Manifolds

A triangle mesh has *boundary* if at least one of its edges is contained in only one triangle. A triangle mesh is a *manifold* if (i) every edge is contained in one or two triangles, and (ii) every vertex is contained in a single loop or fan of triangles. Using these definitions, indicate whether each of the examples below are manifold and/or have boundary. If a mesh is non-manifold, specify one of its non-manifold vertices or edges. If a mesh has boundary, specify one of its boundary edges.

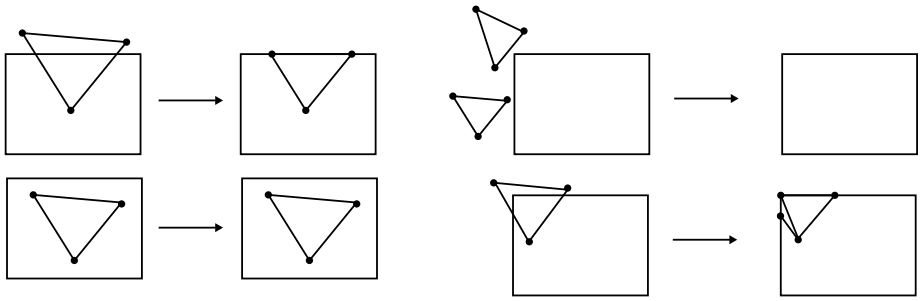






Problem 4: 2D Clipping

A common operation in computer graphics is to "clip" a triangle against a rectangle. Some examples of this operation are illustrated below. (Notice that clipping one triangle against a box may actually result in multiple output triangles!)



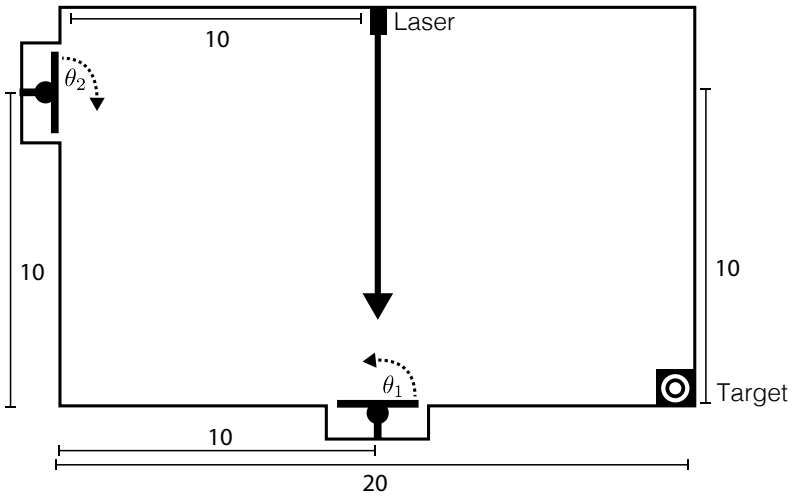
Assume you are given code for the intersection of two line segments in 2D, which returns TRUE if the segments defined by P0,P1 and P2,P3 intersect, as well as the point of intersection:
`bool seg_seg(Point p0, Point p1, Point p2, Point p3, Point* isect)`

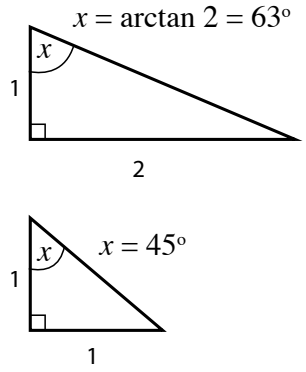
You are also given code for a 2D point-in-tri test for a triangle with vertices `v0,v1,v2` and point `p`:
`bool in_tri(Point v0, Point v1, Point v2, Point p)`

Please give an algorithm for clipping a triangle defined by `v0,v1,v2` to the unit box $[0,1]^2$. **Your solution NEED ONLY HANDLE CASES WHERE THE OUTPUT OF CLIPPING IS EXACTLY 0 or 1 triangles!** (But it's interesting to think of a case where clipping results in FOUR TRIANGLES!) Your solution need only describe the applicable cases and how to construct the output triangle from the subroutine results, it need not be compilable code.

Problem 5: Everyone Loves Lasers (THIS PROBLEM IS OPTIONAL AND NOT REQUIRED FOR PARTICIPATION CREDIT)

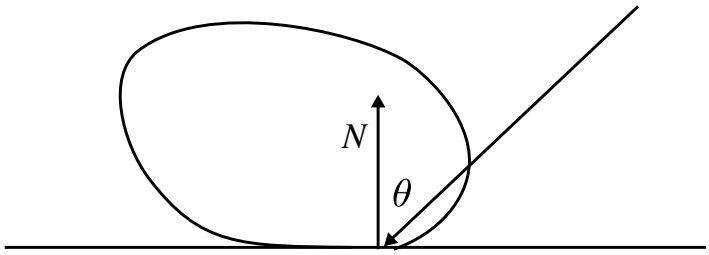
A mad scientist decides to design a fun physics experiment to amuse students in class. The goal of the experiment is to use two **perfectly reflective mirrors** to direct a laser beam, positioned downward from the top of the box, to hit a target in the bottom right corner of the box. The two mirrors can be rotated about their center point by the angles θ_1 and θ_2 as shown in the figure.





A. Please compute positive values of θ_1 and θ_2 to hit the box. (Some helpful triangles are given for you, which may or may not be useful.)

B. One challenge with perfect mirrors is that if you don't get them tilted just right, the laser will miss the target. One of the students, frustrated they couldn't hit the target, takes out a piece of sandpaper and scruffs up the two mirrors. The result is that the mirrors now have a BRDF that is almost fully diffuse, as given by the plot below. Note the surface reflects non-zero incoming light in all directions, but the fraction of light reflected in each of these directions is angle dependent. (More light is still reflected in the direction of perfect specular reflection.)



Assuming that (1) the mirrors are set so that $\theta_1 = \theta_2 = 0$ and that all the walls of the room reflect no light (they are perfectly black), does any laser light hit the target? If your answer is no, explain why not. If your answer is yes, please explain why, and also state whether target is brighter or darker compared to what it would look like in the case of well-aligned perfect mirrors from part A.

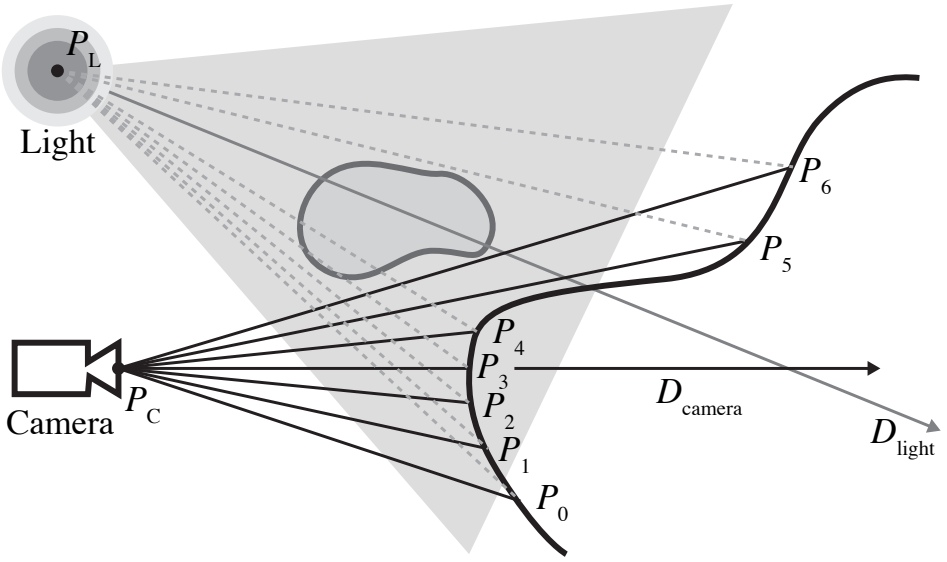
Problem 6: A Highly Irregular Rasterizer (THIS PROBLEM IS OPTIONAL AND NOT REQUIRED FOR PARTICIPATION CREDIT)

Imagine that you have a special kind of rasterizer which doesn't evaluate depth/coverage at uniformly spaced screen sample points, instead it evaluates depth/coverage **at a list of arbitrary 2D screen sample points provided by the application**. An example of using this rasterizer is given below. In this problem you should assume that depths returned by `fancyRasterize` are in WORLD SPACE UNITS.

```
vector<Point2D>  myPoints;  // list of 2D coverage sample points: in [-1,1]^2
vector<Triangle> geometry;  // list of scene triangles in WORLD SPACE
Transform worldToCam;       // 4x4 world space to camera space transform
Transform worldToLight;     // 4x4 world space to light space transform
Transform perspProj;        // 4x4 perspective projection transform


// this call returns the distance to the closest scene element from the camera
// for all points in myPoints (assume infinity if no coverage)
vector<float> depths = fancyRasterize(geometry, myPoints, worldToCam, perspProj);
```

You are now going to use `FancyRasterize` to render images with shadows. Consider the setup of a camera, scene objects, and a light source as illustrated below.



A. Assume you use a traditional rasterizer to compute the depth of the closest scene element at each screen sample point. In the figure, the closest point visible under each sample when the camera is placed at position P_C and looking in the direction D_{camera} is given by P_i. All points in the figure are given in **world space!**

Assume you are given a *world space to light space* transform `worldToLight`. (Light space is the coordinate space whose origin in world space is P_L and whose -Z axis is in the direction D_{light}.) Describe an algorithm that computes, for each point P_i, if the point is in shadow from a point light source located at P_L. Your algorithm accepts as input an array of world space points P_i, world space points P_C, P_l, and has access to all variables listed in the example code. The algorithm should call `fancyRasterize` only once. (No, you are not allowed to just implement a ray tracer from scratch.)

Hint: Be careful, `fancyRasterize` wants points in 2D (represented in a space defined by the $[-1, 1]^2$ "image plane") so your solution will need to describe how it converts points in world space to a list of 2D sample points in this plane. **This involves transformation, perspective projection via `perspProj`, then convert from a homogeneous 3D representation to 2D.**

B. Does the algorithm you gave above generate "hard" or "soft" shadows? Why? (You can answer this question even if you did not correctly answer part A—just assume a solution that does what was asked in part A exists.)

C. Prof. Kayvon quickly looks at the algorithm you devised above and waves his hand dismissively. He says, "remember I told you in class that shadow mapping is such a hack", it only yields an approximation to ray traced shadows. Haotian jumps in and says, "Wait a minute here, this algorithm seems to compute the same solution a ray tracer would to me!" Who is correct? Why?