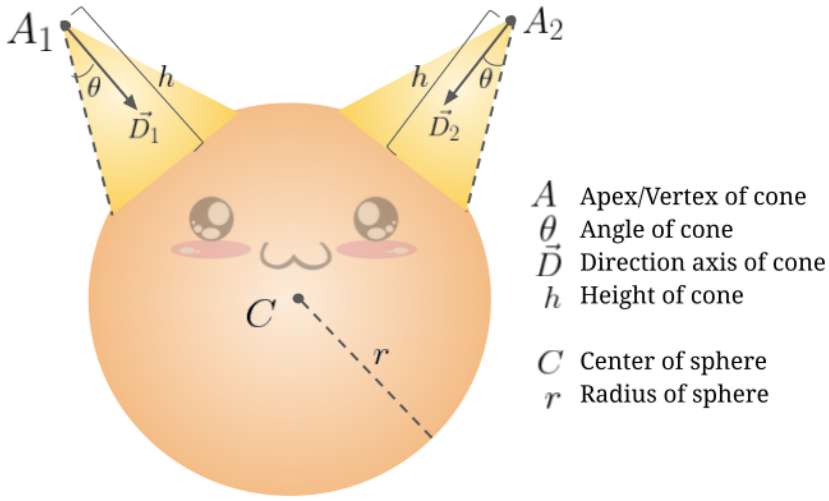
Stanford CS248: Interactive Computer Graphics
Participation Exercise 4

Problem 1: Ray-Cat Intersection

One way of modeling a cat head is with a union of a sphere centered at C with radius r and two cones of height h).



Assume you have access to the following:

(1) an `InfiniteCone` struct (a cone with a height of infinity) that stores A, D and θ
(2) a `Sphere` struct that stores sphere center point C and radius r
(3) a `rayInfiniteConeIsect()` function
(4) and a `raySphereIsect()` function

Questions are on the next page...

A. Give an algorithm for finding the closest intersection of a ray with the cat above. Your solution can be described in words, but make sure it's clear what your algorithm is. Be precise how you use `rayInfiniteConeIsect()` to determine ray-cone intersection with a cone of height h.

```
struct InfiniteCone {
  Vec3D A;        // apex position
  Vec3D D;        // direction of axis
  float theta;    // cone angle
};
struct Sphere {
  vec3D C;        // center of sphere
  float r;        // radius
};

// in the functions below t1 is the ''closer hit'': t1 <= t2
void rayInfiniteConeIsect(InfiniteCone b, Ray r, float* t1, float* t2);
void raySphereIsect(Sphere c, Ray r, float* t1, float* t2);
```

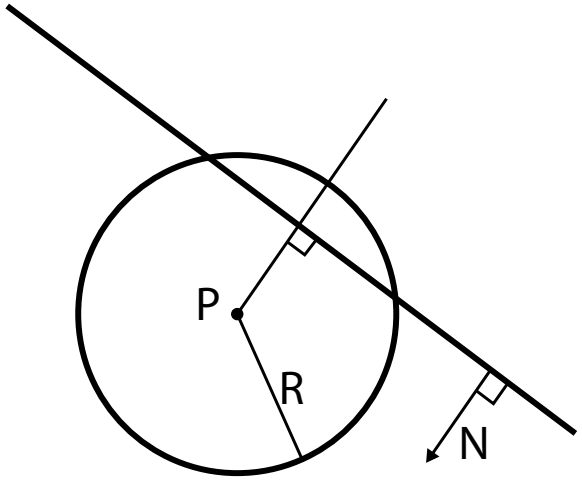
B. Ah, you thought you got off easy without having to solve a ray-primitive intersection problem! Consider a simpler problem where you need to intersect a ray with a cone that has an apex at the origin, is oriented along the z axis (D=(0,0,1)), and has half angle θ. An implicit form of this infinite cone is $x^2 + y^2 = (z \tan \theta)^2$. (Convince yourself this is true!)

Please give an equation for the t value of the intersection point between the infinite cone and a ray with origin $\mathbf{o}$ and ray direction $\mathbf{d}$. You do not need to apply the quadratic formula to solve for t (just give us an equality involving t and components of $\mathbf{o}$ and $\mathbf{d}$).

C. Now imagine you want to compute the intersection of a ray with an infinite cone has apex at point A and is oriented in the direction D (just like in part A). How would you modify the ray's origin and direction to reduce this problem to intersection with the cone from the previous problem that has apex at the origin and is oriented along $(0, 0, 1)$? A description (in words) that involves rotations and translations is fine. But in what direction do you translate, and how do you define the rotation?

Problem 2: More Geometry Intersection

An interesting fact is that testing whether a plane given by $\mathbf{N}^T\mathbf{x} = C$ and a sphere with center point $P = (P_x, P_y, P_z)$ and radius r intersect can be boiled down to closest point on plane. Consider a ray moving from P toward the plane (in the direction -**N**. (The intersection of this ray with the plane gives the closest point on the plane to the center of the sphere.)



A. Give an algorithm for determining if the sphere and plane intersect. As part of this algorithm, give an expression for the closest point on the plane to P. Recall that the equation for points on a sphere of radius r centered at P is:

$$(x - P_x)^2 + (y - P_y)^2 + (z - P_z)^2 = r^2$$

B. Use your algorithm from part A as a subroutine in an algorithm for computing **whether a triangle with vertices T_0, T_1, T_2 intersects a sphere with center point P and radius r.** In addition to results from part A, your solution may also use ray-sphere intersection and point-in-triangle tests as subroutines (see functions below). However, you should clearly state how you use the results of these subroutines. (e.g., how to you use the T-value returned on a hit.)

Your solution can assume that the triangle is not entirely inside the sphere, but should make no other assumptions.

```
// returns true if intersection exists, sets r.min_t to the closest of the
// (potentially two) intersection points (including the case where the ray
// starting within the sphere).
bool ray_sphere_isect(Ray r, vec3 sphere_center, float sphere_radius);

// returns true if the point, assumed to be in the plane of the triangle,
// is within the triangle, false otherwise.
bool in_triangle(vec3 pt, vec3 t0, vec3 t1, vect3 t2);
```

Problem 3: Intersecting Triangles (THIS PROBLEM IS OPTIONAL PRACTICE AND NOT GRADED)

Assume that you are given two triangles, A and B, defined by vertices $(\mathbf{p_{a0}}, \mathbf{p_{a1}}, \mathbf{p_{a2}})$ and $(\mathbf{p_{b0}}, \mathbf{p_{b1}}, \mathbf{p_{b2}})$ and function `(bool, float) ray_tri_isect(Ray r, Triangle tri)` which returns true if an intersection exists and, if so, the value of the ray's t at which a ray intersects the triangle. Given these methods, describe an algorithm that computes whether triangle A intersects with triangle B. (Rough pseudocode is fine, it need not be compilable, but be precise about how you will use the results of `ray_tri_isect()`.) To keep things simple, you do not need to worry about the case where triangles are co-planar and self-contained.