Stanford CS248: Interactive Computer Graphics
Exercise 4

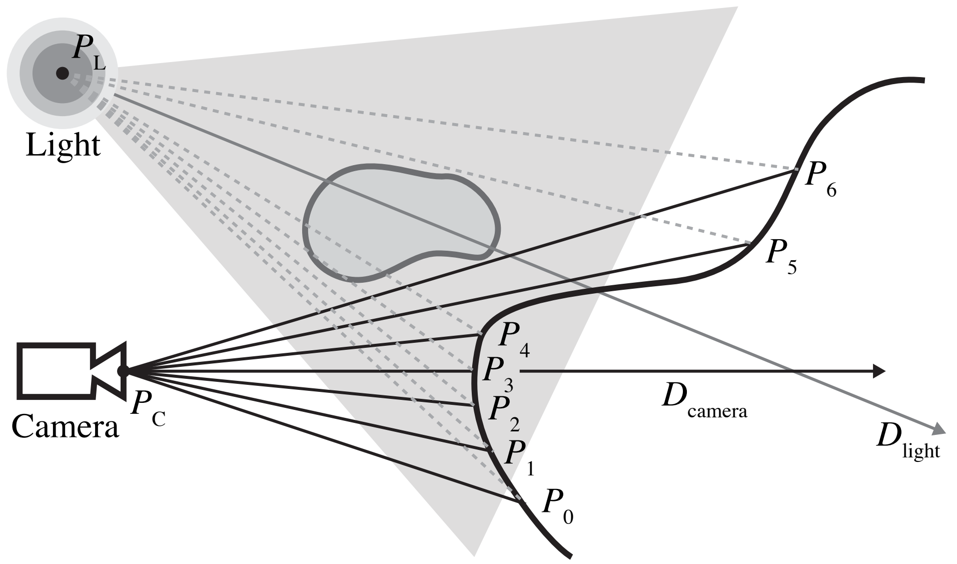
Problem 1: A Highly Irregular Rasterizer

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.)

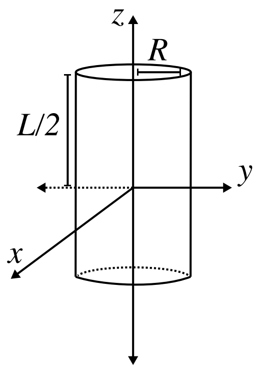
   Hint: Be careful, `fancyRasterize` wants points in 2D (represented in a space defined by the $[-1, 1]^2$ "image plane") so your solution will need to describe how it converts points in world space to a list of 2D sample points in this plane. **This involves transformation, perspective projection via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

Problem 2: Intersecting Cylinders

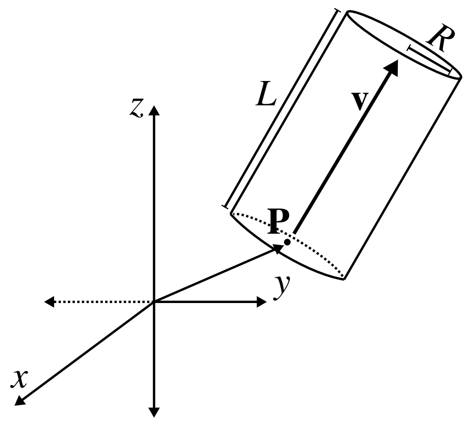
Assume that you have a correct implementation of ray intersection with a cylinder of length L and radius R that is centered at the origin and extends along the Z axis as shown below.



```
bool rayCylinderIsect(Ray r, float length, float radius, float* t)
```

This method returns TRUE/FALSE based on the results of the intersection test, and updates the value of t to corresponding to the point of closest intersection.

Using `rayCylinderIsect` as a subroutine, give a sketch of an algorithm for computing ray-cylinder intersection with a cylinder with radius R and length L, but whose base starts at the point P and is oriented along the arbitrary vector **v**. Hint 1: you can also utilize basic transforms (translate, rotate, etc.) as subroutines. We'd like you to write down a rotation matrix. Hint 2: how do rotation matrices transform basis vectors? (and how can you create an orthonormal coordinate system around **v** using cross-products? You can assume **v** is not [0,0,1].)



This page left blank for solutions...

Problem 3: Refitting a BVH

One of the nice properties of a bounding volume hierarchy (BVH) is that it can be efficiently updated ("refit") when a single primitive contained in the hierarchy is moved. Refitting modifies the BVH so that the BVH property is maintained: *the bounding box of a node is a bound containing the primitives in all child nodes*. Refitting does not modify the structure of the BVH – **it only updates bounding boxes for some of the tree's nodes based on the new bounds of leaf nodes.**

Below you're given a definition of a `BVHNode` and interfaces for getting the bounding box of a primitive. Please implement the function `refitBVH(node)` which should update bounding boxes of the *necessary nodes* in the BVH so that the BVH property holds. **You should assume that only primitives in the specified leaf node (`node`) have moved**. To keep things simple, all BVH nodes hold a pointer to their `parent` in the BVH. **NOTE THAT EVEN THOUGH WE GAVE YOU VALID C STRUCTS IN THE PROBLEM FOR CLARITY, YOU ONLY NEED TO SKETCH OUT AN APPROACH TO A SOLUTION. VALID C CODE IS NOT REQUIRED. A DESCRIPTION OF AN APPROACH IS FINE.**

```
struct BBox {
   void clear();               // resets bbox to empty
   void union(const BBox& b);  // enlarges bbox to include volume in b
};

struct Primitive {
  BBox getBBox() const;    // returns primitive's world-space axis-aligned bbox
};

struct BVHNode {
   BVHNode* left, *right, *parent  // parent is NULL if root node, left/right NULL if leaf
   BBox bbox;         // node's bbox, you need to update this!
   int numPrims;      // 0 if node is interior node, non-zero otherwise
   Primitive* prims;  // prims[i]->getBBox() returns bbox of i'th prim in node
};

// Refit the entire BVH assuming that the contents of the leaf node 'node' have changed.
void refitBVH(BVHNode* node) {








}
```