Stanford CS248A: Computer Graphics
Written Assignment 1

Chaining Transforms

Problem 1: (Graded on Effort Only - 20 pts)

A. Consider a normalized coordinate system (post-projection) where the bottom-left of the screen corresponds to (x,y) = (-1,-1) and the top-right of the screen corresponds to (x,y) = (1,1). Give a 4×4 matrix **S** that describes a transform that takes points in this normalized space to screen space where the bottom-left corner of the screen is (0,0) and the top-right is (1000, 500). Note we're asking for a 4×4 matrix, so assume that the transform leaves the "z" component and "w" components of a 3D homogeneous input point unchanged.

$$\mathbf{S} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

B. Consider a point **p** on an object ($\mathbf{p} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$), which is represented in homogeneous form in an "object-space" coordinate system where the object is centered at the origin.

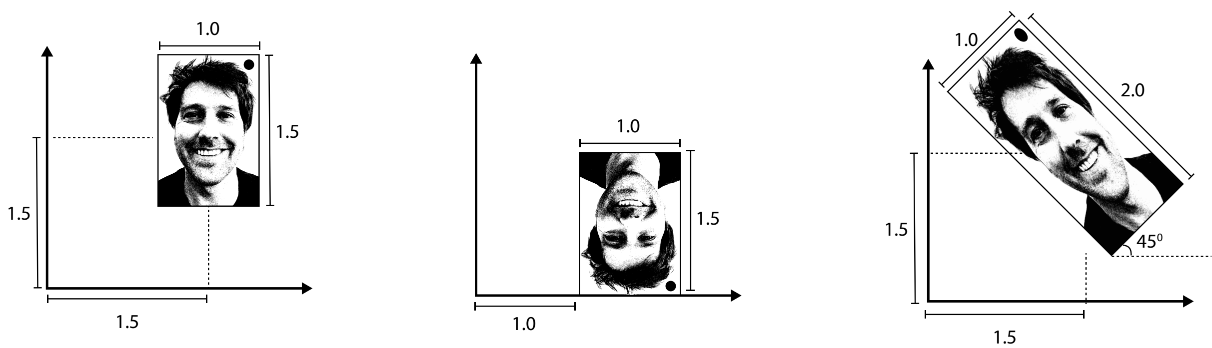
Imagine that you are also given the following transforms, all represented as 4×4 matrices.

- **O** - which transforms points in the object-local coordinate system to world space.
- **C** - which transforms points in a world-space coordinate system to a "camera space" coordinate system where the camera is at the origin and looking down -Z.
- **P** - which performs perspective projection on points in the camera space, yielding points in the "normalized space" defined in part A.

Given the transforms above, along with your transform **S** from part A, please give an expression for computing (x,y), the screen space point of projection corresponding to **p**. Recall that the screen is the same setup as in part A, where (0,0) is the bottom-left of the screen, and (1000, 500) is the top-right. **Hint: don't forget, we are asking for a 2D screen-space point, not a point represented in homogeneous space.** (This answer does not depend on a correct answer to A. Just assume you have a correct solution to A, and the matrix is called **S**.)

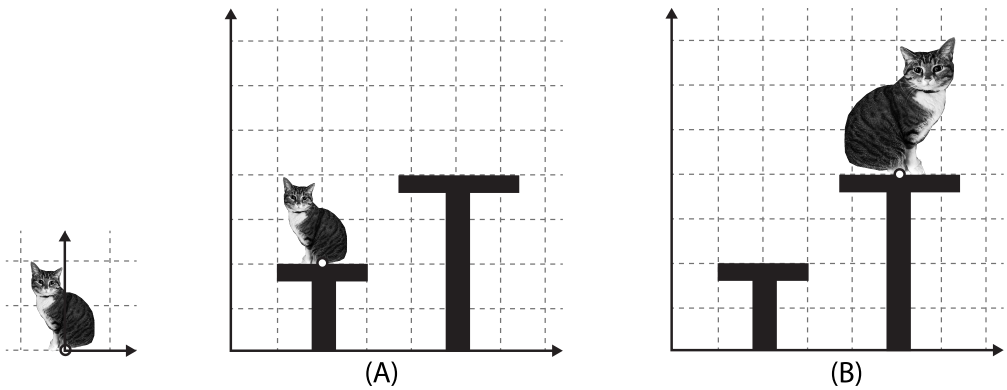
Problem 2: (Graded on Effort Only - 20 pts)

A. Please describe a sequence of basic transforms needed to transform the rectangular object on the left (a rectangle of width 1.0 and height 1.5, and centered at the point (1.5, 1.5)) into its position in (A). Also describe the transformations needed to take its position on the left into its position in (B). (These are separate questions). You may assume that basic transforms available to you include: rotate(X), translate(X,Y), scale(X,Y), shear(X,Y). **Be careful, rotations and reflections can sometimes look similar, so we placed a dot on the object to help to disambiguate!**



(A) (B)

B. When he brought a new cat home from the animal shelter, Prof. Kayvon took a photo of the cat sitting on a cat tree. As you can see in Figure A below, the cat is 2 units tall, and its base is at coordinate (2,2). A year later, Kayvon's cat has grown by 50%, and he takes a second photo (photo B) of his cat sitting on a taller cat tree. Also notice that in Figure B the cat is mirrored in its position. **Please provide a sequence of transformations that moves the cat from its position in image (A) to its position in image (B). (You only need to consider the cat, not the cat trees.)** The catch is that you can ONLY USE two transforms: `translate(x,y)` and `scale(x,y)`. Please keep in mind that the arguments to these functions can be negative. For convenience, the left-most part of the figure illustrates the cat image from Figure A with its base at the origin.



Mapping Pixel Centers to Screen Coordinates

Problem 3: (Graded on Effort Only - 15 pts)

A. Assume you are rendering an image to a screen that is 5000 pixels wide and 2000 pixels tall. In lecture 2 we talked about 2D screen coordinates where (0,0) was the top left of the image, and (5000,2000) was the bottom-right corner of the image. In this coordinate system, what are the coordinates for the center of the top-left pixel of the screen?

B. Now image the screen is the same size (in pixels) as in part A, but now we are going to define the top-left of the image to be at the coordinate (1000,1000) and the bottom-right of the image to be at (2000, 3000). In this new coordinate system what are the coordinates for the center of the top-left pixel of the screen?

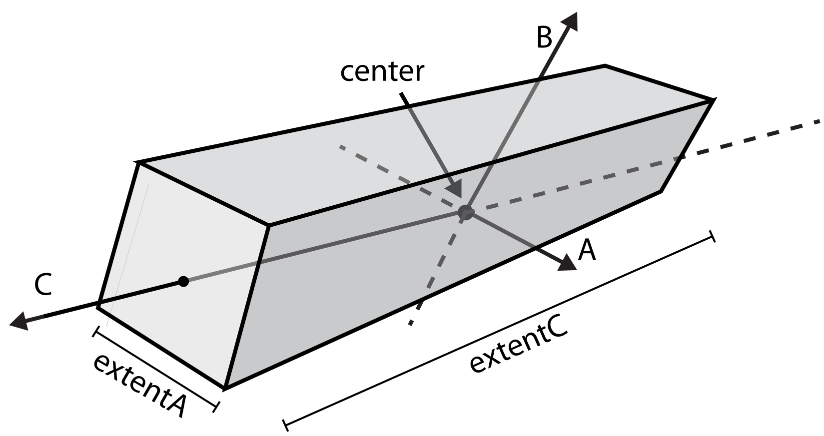
C. (This one is a little trickier.) Just like in the previous problem, assume you are rendering an image to a screen that is 5000 pixels wide and 2000 pixels tall. We change the coordinate system so that the top-left corner of the image is at coordinate (0,0) and the bottom-right corner of the image is at coordinate (1000,1000). Assume that the 5000×2000 image is displayed on a monitor that has pixels that are actually 1 mm×1 mm in size in the real world. (Each pixel is a square in the real-world.) Consider a quadrilaterial that is a square ($X \times X$) in image space. What does this square look like when displayed on this monitor? Is it square? Is it a rectangle? Does it taller than it is wide? Why?

Trace Everything!

Problem 4: (Graded on Effort Only - 20 pts)

A. Throughout the quarter we've talked about intersection of a ray with an axis-aligned bounding box (AABB). For example, it was a key primitive needed to implement ray-BVH traversal, since each sub-tree of a BVH was represented by an AABB. But there's nothing preventing a BVH from representing its nodes using *oriented bounding boxes*. Below is a definition of an oriented bounding box. The box is centered at the point `center` and oriented with a principle axis along the vector given by `axisC` as shown below. `axisA` and `axisB` are orthogonal axes of the OBB.

```
struct OBB {
  Vec3  center;  // position of center point
  float extentA, extentB, extentC; // length of bbox along axes
  Vec3  axisA;
  Vec3  axisB;
  Vec3  axisC;
};
```



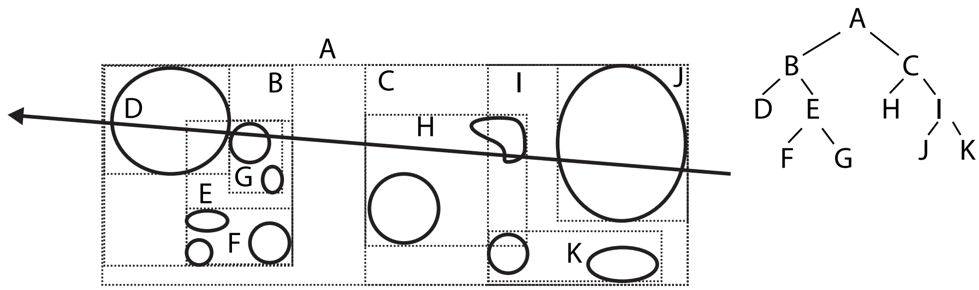
Question is on the next page...

Given the definition above, assume that you have a ray struct `Ray` and the following method, which intersects a ray with a standard AABB.

```
struct Ray {
  Vec3 o;  // origin
  Vec3 d;  // direction
};

// returns distance along ray to first intersection if there is a hit,
// returns INFINITY if no intersection
// bmin and bmax store the min and max values of the AABB on each axis.
float rayBoxIsect(Ray r, Vec3 bmin, Vec3 bmax);
```

Given the structures above, please write high-level sketch of an algorithm for intersecting a ray `r` with an OBB `obb`. You can assume you have a regular math library where you can construct 3×3 matrices representing transforms and perform matrix-vector math. In your algorithm please define any matrices you use by giving their coefficients.

B. Consider the ray below tracing through the given BVH.



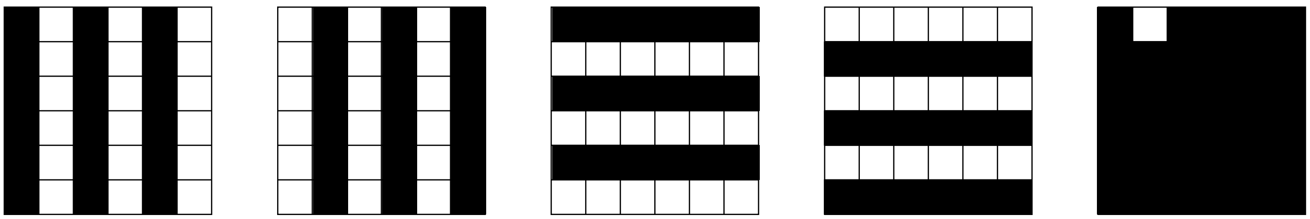
Imagine a ray traversal algorithm that **ALWAYS traverses to the "left child" first (if it hits the left child) irregardless of the direction of the ray.** (e.g, the "left child" is the child listed to the left in the tree illustration of the BVH... so D is the left child of B and F is the left child of E.)

Please give the ordered sequence of BVH nodes that ray must visit when determining the closest intersection along the ray. **(We say a ray "visits" a node during traversal if it is determined that the node might contain the primitives resulting in the closest hit, and thus we must recurse into the node.)** Assume that if a ray has already found an intersection that is closer than one of the child bounding boxes of the current node, it will skip visiting that node.

C. Now assume the same setup as the prior problem, but assume that the BVH first visits the "closer" of the two child nodes (instead of the left child); "closer" means closer to the ray's origin. Please give the ordered sequence of BVH nodes that ray must visit when determining the closest intersection along the ray, and also **describe why this strategy is more efficient than the first "left child first" strategy.**

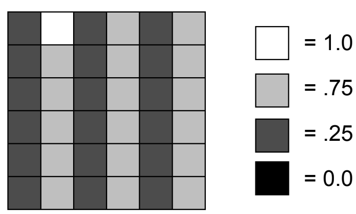
Problem 5: (Graded on Effort Only - 10 pts)

In class we talked about how all images can be represented in the 2D *cosine basis*. Now consider a very different image representation scheme that represents 6×6 pixel patches in terms of a linear combination of the five $6{\times}6$ basis images given below:



B_0 B_1 B_2 B_3 B_4

A. Consider representing the following 6×6 image in terms of the base patches above. What are the coefficients of the image under this representation? Explain why, **for the specific case of this image**, we have devised a very efficient image compression scheme. (Hint: what is the size of the 6×6 image in the pixel basis? What about the size of the representation in terms of the image patches above?)

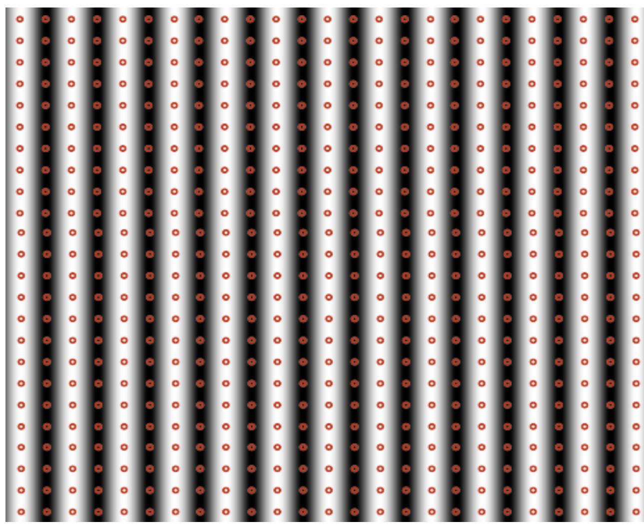


B. Although the scene in part A can be a very efficient encoding for some 6×6 images (only 5 numbers to represent an image!), the problem with the scheme is that is cannot accurately represent all 6×6 images. Draw one example of an image that cannot be represented as a combination of the provided "basis" images.

Sampling

Problem 6: (Graded on Effort Only - 10 pts)

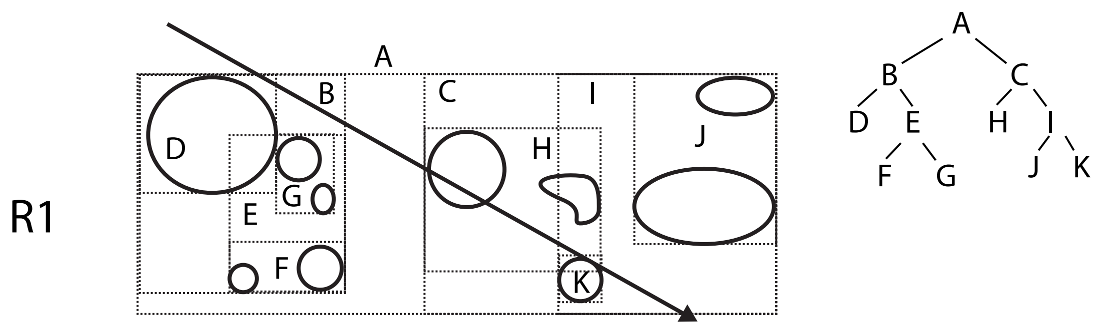
A. Say you wanted to sample the intensity of a 2D signal $f(x, y)$ that is given by a sign wave with a period 64 units in x, $f(x, y) = \sin(2\pi(x/64))$. Assume that we sample the signal at evenly spaced intervals of 32 units in x and y, and then attempt to reconstruct the signal with **piecewise constant interpolation** (also called nearest-neighbor interpolation). Please describe what your reconstruction looks like? Now consider the case were the signal is sampled every 64 pixels! (not shown in the picture). What will the result reconstructed image look like in this case?



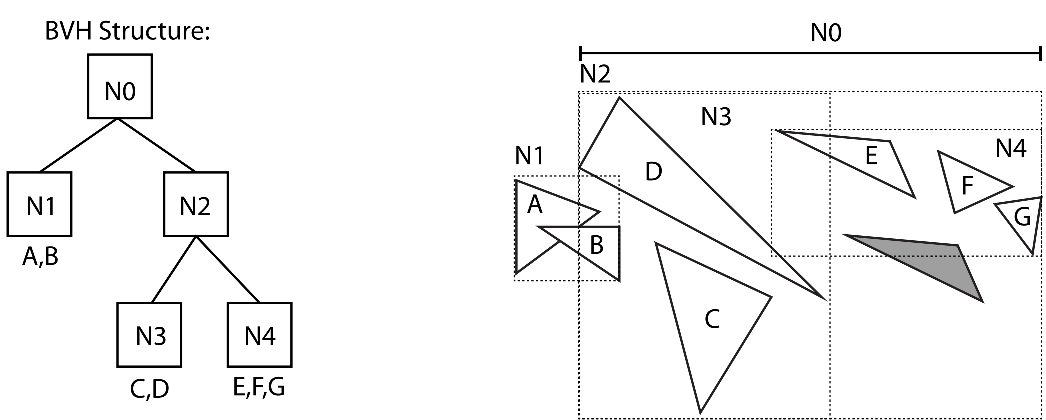
Ray Tracing With BVH

PRACTICE PROBLEM 1:

A. Consider the scene below organized into a BVH. The structure of the BVH is shown in the top-right. Please give the ordered sequence of BVH nodes that ray R1 must visit when determining the **closest** intersection along the ray. **(We say a ray "visits" a node if during traversal it is determined that the node might contain the primitives resulting in the closest hit, and thus we must recurse into the node.)** You should assume that: (1) when a ray hits both child bounding boxes of the current node, the ray will visit the node with the closest bounding box first, (2) If a ray has already found an intersection that is **closer than** one of the child bounding boxes of the current node, it will skip visiting that node.



B. Consider the scene geometry (8 white triangles) and corresponding BVH shown below. The dotted boxes on the right indicate the bounding boxes of interior nodes N0-N4. Now imagine that triangle E moves so that it is now located at the position indicated by the shaded triangle. Assume that the BVH structure is not changed (the topology stays the same and so do the boundign boxes of interior nodes.) Precisely describe a ray-scene intersection test error that could occur as a result. In your answer describe or draw a ray that would trigger the error and describe what tracing this ray could not yield a correct hit result.



PRACTICE PROBLEM 2:

The implicit form of an ellipsoid centered at point (x_0, y_0, z_0) can be given by:

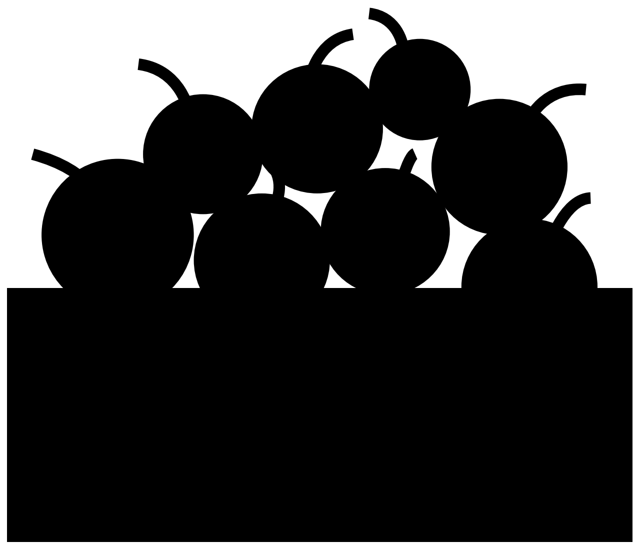
$$\frac{(x-x_0)^2}{A^2} + \frac{(y-y_0)^2}{B^2} + \frac{(z-z_0)^2}{C^2} = 1$$

Recall that the equation for all points on an array with origin $\mathbf{o}$ and direction $\mathbf{d}$ is given by $\mathbf{o} + t\mathbf{d}$. Please give an espression for the values of t that correspond to the ray-ellisoid intersection points (Hint: set up a quadratic equation and solve for t using the quadratic formula.) After giving an expression for t, please describe how many unique solutions for t there can be. Why is this the case?

Cherry Picking

PRACTICE PROBLEM 3:

After a raining winter week, you find yourself craving some fresh air and sunlight. Therefore, you've decided to go cherry picking with your CS248A friends. However, instead of the traditional method of picking cherries by hand, you'll be shooting light rays at a box of cherries.



Each cherry is represented by two spheres: a large sphere for the fruit, and a smaller internal sphere for the cherry pit. Assume the light rays can travel through the flesh of all cherries and continue to intersect with primitives behind them, but stops if it hits the box or any cherry pit. Each time the ray hits any part of a cherry, you get to keep it!

You are given two subroutines below to use. Both functions will return true when an intersection occurs (and false if not), and populate the ray's `t`-values with information for up to two points of impact. In cases where there's only one point of intersection, like when the ray just grazes the shape, you can consider both `t`-values to be identical. If there is no intersection, then it's assumed that the `t`-values are assigned a significantly large number.

```
// intersect ray with the box
bool rayBoxIsect(Box b, Ray r, float* t1, float* t2);

// intersect ray with a sphere centered at the origin
bool raySphereIsect(Ray r, float radius, float* t1, float* t2);
```

Question continues onto the next page...

You can assume that you have the following information for each cherry.

```
Cherry {
  vec3 cherry_center;
  float cherry_radius;
  CherryPit cherry_pit;
}

CherryPit {
  vec3 pit_center;
  float pit_radius;
}
```

Warning!: Notice that `raySphereIsect()` assumes that the sphere is centered at the origin.

Given that you get to keep all cherries that your ray hits, please give an algorithm for finding the number of cherries you get to take home given an input ray `r`, `Box b`, and list of `Cherry`s `cherry_list`. Remember that the ray stops if/when it hits the `Box` or a `CherryPit`. Psuedocode is fine!

```
// TODO: Implement the function below...

int getNumCherries(Ray r, Box b, std::vector<Cherry> cherry_list) {

}
```

Refitting a BVH

PRACTICE PROBLEM 4:

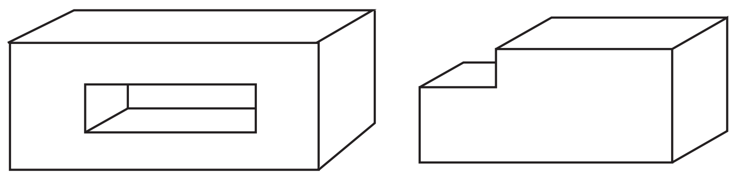
One of the nice properties of a bounding volume hierarchy (BVH) is that it can be efficiently updated ("refit") when a single primitive contained in the hierarchy is moved. Refitting modifies the BVH so that the BVH property is maintained: *the bounding box of a node is a bound containing the primitives 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) {











}
```

PRACTICE PROBLEM 5:

You wish to intersect a ray with a shape defined by two axis aligned boxes B_1 and B_2. The shape is defined by the volume given by $\mathrm{B}_1 - \mathrm{B}_2$. In other words, B_2 cuts volume out of B_1. Depending on the size and shapes of the two boxes, you can get some interesting shapes. Two examples are given below. B_2 **need not intersect with** B_1**, and when this is the case, the resulting shape is just** B_1**.** **(**$\mathrm{B}_1 - \mathrm{B}_2 = \mathrm{B}_1$**).**



Assume you are given code for computing the closest point of intersection with a box:

```
// Returns true if ray r hits box b, fills in t with distance to closest hit.
// IMPORTANT: If r originates inside the box, or if ray R originates on the
// surface of the box and points towards the inside of the box, then the closest
// hit will be the point where r leaves the box, not the origin of r.
bool rayBoxIsect(Ray r, Box b, float* t);
```

Please use `rayBoxIsect` as a subroutine in an algorithm for computing the closest intersection between a ray R **and the shape defined by** B_1 **and** B_2**.** To make things easier, you can assume that the ray R originates "outside" both boxes, and you can ignore the case where the ray grazes any box.

You can assume that `Ray.o` and `Ray.d` give the origin and direction of a ray. Pseudocode is fine (it need not compile), but be precise enough for the grader to understand how to perform key steps. **Hint: first give pseudocode for how to find the point where the ray r enters each box AND the point where the ray leaves each box. Then what do you with this information to solve the problem?**

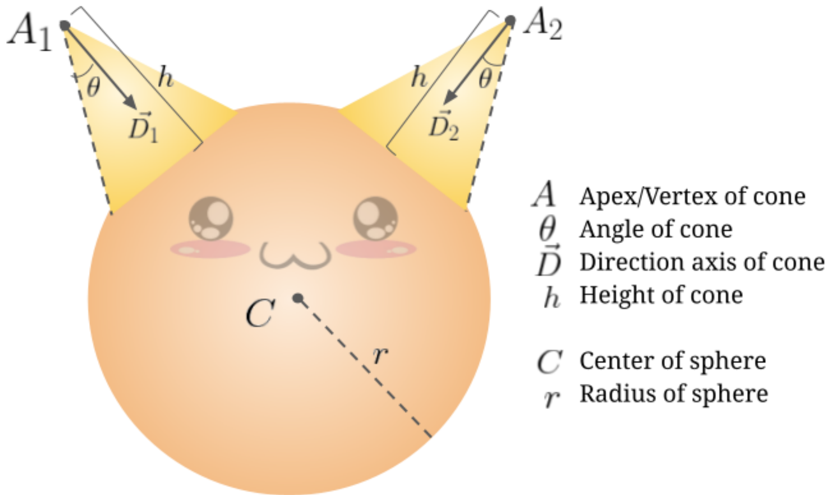
Please provide your answer on the next page.

```
// returns true if ray hits shape
// fills in t to give the position of the closest hit
bool hitsShape(Ray r, Box b1, Box b2, float* t) {

   float b1_t1 = INF;  // distance to first hit with B1 (if applicable)
   float b1_t2 = INF;  // distance to second hit with B1 (if applicable)
   float b2_t1 = INF;  // distance to first hit with B2 (if applicable)
   float b2_t2 = INF;  // distance to second hit with B2 (if applicable)
```

Ray-Cat Intersection

PRACTICE PROBLEM 6:

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!)

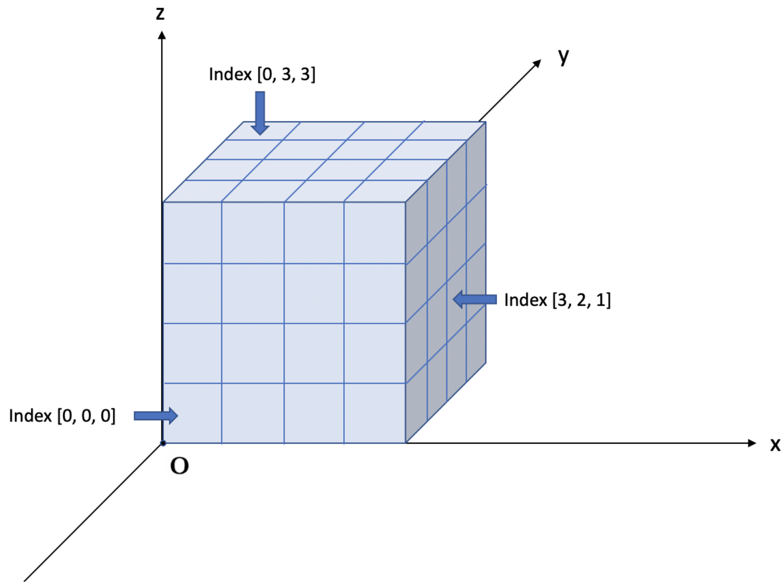
   Please give an equation for the t value of the intersection point between the infinite cone and a ray with origin $\mathbf{o}$ and ray direction $\mathbf{d}$. You do not need to apply the quadratic formula to solve for t (just give us an equality involving t and components of $\mathbf{o}$ and $\mathbf{d}$).

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?

Ray-Grid Intersection

PRACTICE PROBLEM 7:

We are interested in casting rays onto a uniform 3D grid shown below. The grid is $4\times4\times4$ and each grid cell is a cube with side length 1. The origin of the grid ($\mathbf{O}$) is located at the origin of the coordinate system: $\mathbf{O} = (0, 0, 0)$.



Recall that a ray can be represented as $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$. Assume that the origin of the ray is ALWAYS outside of the grid and recall that a plane can be represented as $\mathbf{N}^T\mathbf{x} = c$.

Hint: A plane parallel to the XY plane has $\mathbf{N} = (0, 0, 1)$*, a plane parallel to the XZ plane has* $\mathbf{N} = (0, 1, 0)$ *and a plane parallel to the YZ plane has* $\mathbf{N} = (1, 0, 0)$.

Assume that you have access to the following:
(1) Struct `Plane` that stores normal `N` and offset `c`
(2) Struct `Ray` that stores origin `o` and direction `d`
(3) Function `bool rayPlaneIsect(Plane plane, Ray r, float* t)` that takes as input a plane and a ray and set `t` to be the intersection. A boolean is returned to indicate if an intersection is found.

THE QUESTIONS BEGIN ON THE NEXT PAGE

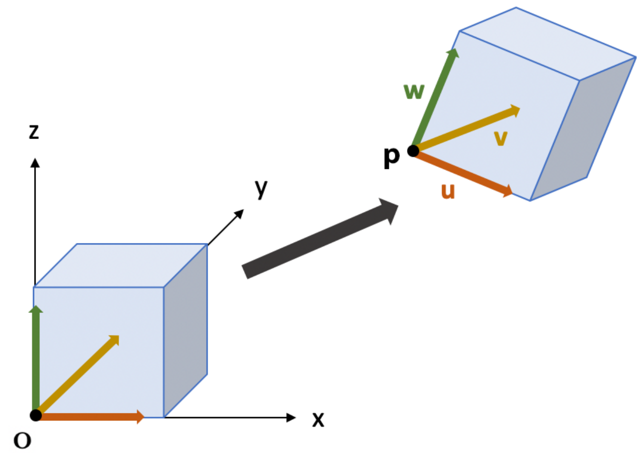
A. Give an algorithm for finding the index of the first cell getting hit by the ray. The index of a cell should be represented as a 3D vector (a, b, c) corresponding to the indices in x, y, and z axis, as shown in the figure above. **The complexity of your algorithm SHOULD NOT scale as the number of cells in the grid increases.** Your solution can be described in words or rough pseudocode, but make sure it's clear what your algorithm is. Be precise about how you initialize the given structs and how you use `rayPlaneIsect()` to determine the ray-grid intersection. Hint: figure out where the ray first hits the volume of the grid, then turn that into an index.

```
struct Plane {
  vec3 N;       // unit normal
  float c;      // offset
};
struct Ray {
  vec3 o;       // ray origin
  vec3 d;       // ray direction
};

bool rayPlaneIsect(Plane plane, Ray r, float* t);
```

B. Now we want to remove some cells from the grid. Given a collection of cell indices that are removed `removedCellIndices`, give an algorithm for finding the index of the first cell hit by the ray. **Now, the complexity of your algorithm CAN scale linearly with the number of grid cells.** Your solution can be described in words.

```
vec3[M] removedCellIndices = [vec3(0,1,0), vec3(1,2,1), ...]
```

C. Now consider the case where the grid is translated and rotated such that the grid origin **O** is located at position **p** and the three sides next to **O** now have unit directional vector **u**, **v** and **w**. An illustration is provided below:



We want to find the index of the first grid cell hit by a ray. Assume that you can access your solution to part B via function `getCellIndex(Ray r)`. Please implement the function:
`getCellIndex(Ray r, vec3 p, vec3 u, vec3 v, vec3 w)`.
Be precise about how to construct arguments to `getCellIndex()`. **Specifically, please be precise about how you construct and use transformation matrices.**

```
vec3 getCellIndex(Ray r, vec3 p, vec3 u, vec3 v, vec3 w) {

// implement your solution here...
```

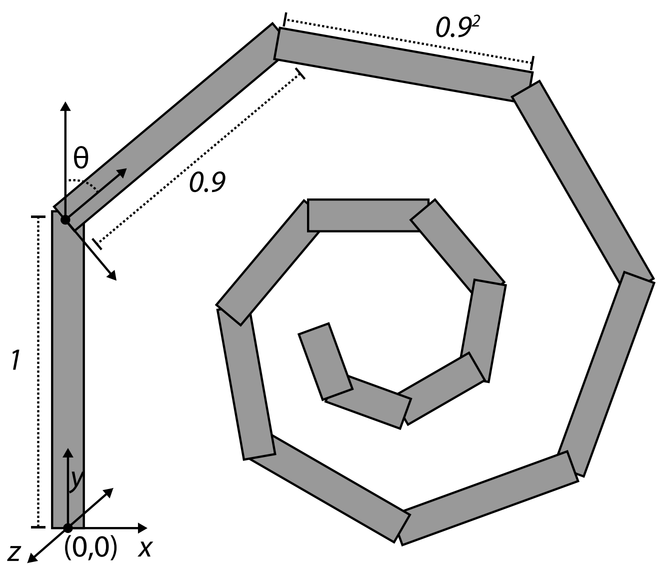
Solution:

Create a transform:

Ray Tracing A Funky Scene

PRACTICE PROBLEM 8:

Consider a scene contain 15 rectangular-boxes as shown below. The scene is formed by starting with a box of length=1.0, with it's center-bottom at the origin, and extending down the Y axis (the width and depth of the box do not matter in this problem). The next box begins at the end of the first box (at point (0,1,0)), but it has a length that is 0.9 times the first box, and rotated with an angle θ relative to the first box. Note that the rotation is about the Z axis, in the *clockwise direction* as viewed when looking down the -Z axis. This pattern of starting the next rectangle at the end of the previous, rotating by θ, and shrinking length by a factor of 0.9 repeats for a total of 15 boxes.



You are given a function `void ray_rect_isect(Ray r, float rect_length)` that computes the first hit of a ray r, with a box aligned with the Y axis of length `rect_length` (see code on next page) For example, the call `ray_rect_isect(r, 1.0)` would compute the intersection of a ray with the first box. Upon return `r.min_t` would reflect the distance to the closest hit. *If the original* `r.min_t` *of the ray passed into the function is smaller than the t computed from the intersection, then* `r.min_t` *is unchanged.*

You are also given a function `rotatez` that takes a 3-vector v and rotates it θ degrees about the Z axis in the **counter-clockwise direction.**. Using only these routines, on the next page, please implement the function `isect_funky_scene(Ray& r)`, which will fill in `r.min_t` to contain the closest point on the ray. You should make no assumptions about the origin or direction of the ray in 3D, except that it does not originate from inside any of the boxes.

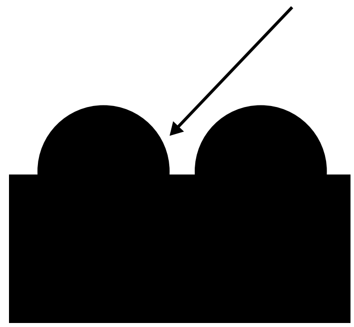
Hint: be careful about the direction of your rotations. The geometry rotations in the figure are clockwise, and the function `rotatez()` specifies a rotation in the counter-clockwise direction as was typical in class.

```
// students are free to assume that useful constructors, or common ops like
// addition, multiplication on vectors is available..
struct vec3 {
  float x, y, z;
};

struct Ray {
  vec3 o;
  vec3 d;
  float min_t;  // assume this is initialized to INFINITY
};

// rotate counter-clockwise about Z axis (counter clockwise defined when looking down -z)
vec3 rotatez(float theta, vec3 v);

// fills in r.min_t if intersection with rectangular box is closer than current r.min_t
void ray_rect_isect(Ray r, float rect_length);

// assume r.min_t is INFINITY at the start of the call
// result: fill in r.min_t as a result of intersecting r with the 15 segments
void isect_funky_scene(Ray& r) {

}
```

Intersecting Solids

PRACTICE PROBLEM 9:

Consider the 2D shape given below, which is made up of the intersection of volumes contained by two circles and a rectangle.

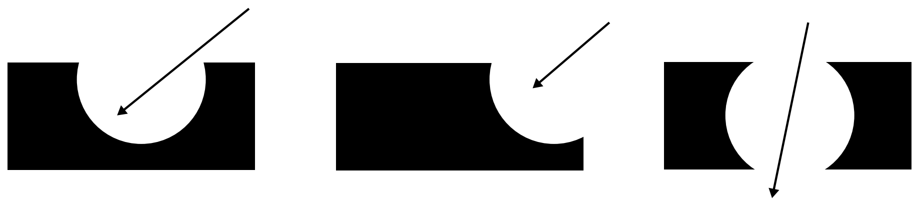


Assume you are given two functions for ray-shape intersection as given below. Both functions return true if there is an intersection (false otherwise), and fill in ray t values for up to two hits. If there is a single intersection point, such as when the ray grazes the shape, just assume that both t values are the same. If there is no intersection, assume the t's are set to a very large number.

```
bool rayBoxIsect(Box b, Ray r, float* t1, float* t2);
bool rayCircleIsect(Circle c, Ray r, float* t1, float* t2);
```

A. Assuming we call the circles c_1 and c_2, and the box b, give an algorithm for finding the closest intersection of a ray with the shape above.

B. Now consider a new kind of shape, which is formed by *subtracting the region inside one circle from the region inside one box. (a few cases are given to help your understanding.)*



Give an algorithm for finding the closest intersection of a ray with this new type of shape. *Note, please make sure you handle the case where the shape is actually two disjoint pieces, and the case where there is no intersection! However, to make things simpler, you can assume the ray origin is outside the area of the box or the circle.*