CMPT 370 Fall 2025 Assignment #2
Instructor: Dana Cobzas

Raycasting - 80 points (8%, individual)

Due: Wednesday October 1st , midnight

Goal

In this assignment you will practice the ray casting methods that we discuss in class.

What to submit

- code containing also geometry files that you tested with
- document with the figures and explanations as detailed below

General instructions

You will render ellipsoids in this assignment, which are described in an input json file. We will test your program using several different input files, so it would be wise to test your program with several such files. We provided a geometry files as examples. Each ellipsoid is described using it's center `(x,y,z)`, it's radii `(a,b,c)` and the material properties needed for the shading part.

We provided a shell (`index.html` and `drawstuff.js` file) that you can use to build your code. You are only allowed to change the `drawstuff.js` file. The html should stay the same. The shell shows how to read the json file and a call for the main drawing function `raymarchInputEllipses`. You will have to fill the `raymarch(ro,rd)` sub-function that calculated the color of the pixel `rd=P` given the eye location at `ro=E`. The pixel location `rd` was already transformed for you in world coordinates.

We also provided `gl-matrix.js` a good library for matrix and vector operations that contains all operations you need for this assignment.

Here is the **view configuration** in `ellipsoids.json`
All vertex locations should be described in world coordinates, meaning they do not require any transformation. Locate the eye at (0.5, 0.5, -0.5), with a view up vector [0,-1,0]
and a look at vector of [0 0 1]. Locate the window a distance of 0.5 from the eye, and make it a 1x1 square normal to the look at vector and centered at (0.5,0.5,0), and parallel to the view up vector. With this scheme, you can assume that everything in the world is in view if it is located in a 1x1x1 box with one corner at the origin, and another at (1,1,1). Put a white (1,1,1) (for ambient, diffuse and specular) light at location (0.5, 0.5, -2).

Advice: be careful to implement the algorithm we described in class, which loops first over pixels, then over the objects. Most WebGL code implements rasterization, which loops first over objects (triangles), then pixels. We provided some information on raytracing for ellipsoids at the end of this document.

You should code the core of this assignment yourself. You may not use others' code to determine the location of pixels in the world, to do ray-triangle intersection, or to color a pixel. You may use other math libraries you find, but you must credit them in comments. You may recommend libraries to one another, speak freely with one another about your code or theirs, but you may never directly provide any code to another student.

Part 1 [50p] : Using ray casting, render unlit, colored ellipsoids
Use ray casting to render unlit ellipsoids, with every pixel inside each ellipsoid is having the unmodified diffuse color of that ellipsoid (e.g, if the diffuse color is (1,0,0), every pixel on the ellipsoid should be red). You will have to test for depth, to ensure that each ellipsoid is correctly colored. You should see flat ellipsoids, no shading.

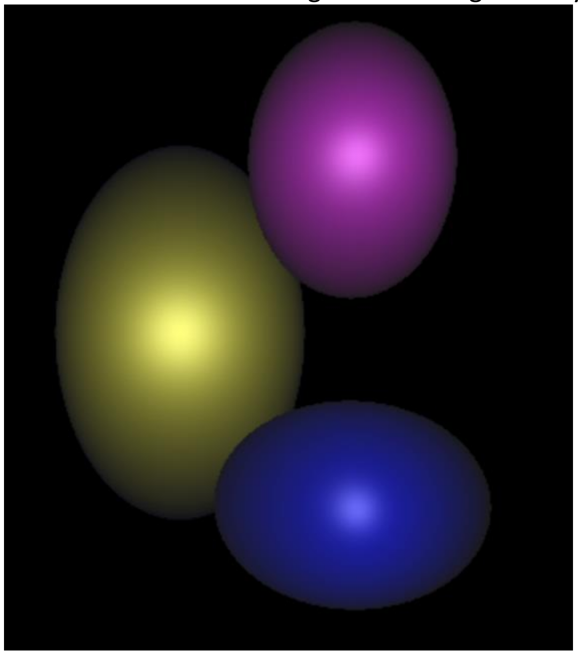
The `ellipsoids.json` provides an example with 3 objects one blue, one yellow, one magenta.

Part 2 [20]: Using ray casting, render lit ellipsoids
Now you will have to perform a local Blinn-Phong lighting calculation at each intersection. As you perform that lighting calculation, don't forget to normalize your vectors. You should now see ellipsoids with depth revealed by illumination, in the same locations and with the same silhouettes as in part 1. The shading is given by a white (1,1,1) (for ambient, diffuse and specular) light at location (0.5, 0.5, -2).

In your document [10p]

- *Sketch the view configuration (as described above including location of light) and the 3 ellipsoids from our example file.*
- *Explain how the screen* `(x,y)` *to world coordinate* `rd` (`vec3` object) *conversion is done given the simplified view geometry. This code is found in the function* `raymarchInputEllipses`
- *Show fragment code for all the following steps in the algorithm involving ray calculation, its intersection(s) with ellipsoids, calculation of the closest one .*
- *Show the fragment code that does shading calculation, explaining each term of Blinn-Phong model (ambient, diffuse, specular)*

Here is a correct rendering for the two geometry files:

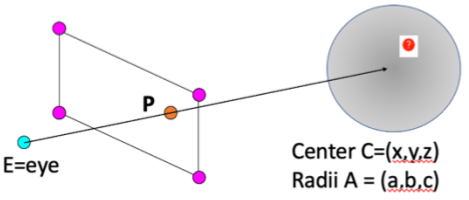


Ray casting algorithm [same as in the class slides]

For **each screen pixel**

Convert from pixel(u,v) to world point P(x,y,z)

- Bilinear interpolation using the 4 screen corners (UL UR LL LR)
- OR direct conversion for a simpler configuration



Find the **ray** from the eye through the pixel

$R(t) = E + t(P-E) = E + tD$

For **each object** (ellipsoid A B C**)** in the scene

If the ray **intersect**s the object,

- ellipse, S on surface: $((S-C)/A)\bullet((S-C)/A) = 1$
- S = ray: $D/A\bullet D/A\ t^2 + 2\ D/A\bullet(E-C)/A\ t + (E-C)/A\bullet(E-C)/A - 1 = 0$

  quadratic equation
  - $at^2 + bt + c = 0$
  - with: $a = D/A\bullet D/A$, $b = 2\ D/A\bullet(E-C)/A$, $c = (E-C)/A\bullet(E-C)/A - 1$
- calculate solution

  quadratic formula: $t = (1/2a)\ (-b \pm (b^2 - 4ac)^{½})$

  or: $t = (1/2a)\ (-b \pm \Delta^{½})$
- If $\Delta<0$ no solution; if $\Delta=0$ two identical; if $\Delta>0$ two solutions keep closest

and is **closest yet**

Closest intersection = smallest t that is visible

Record intersection and object

Find **color** for **closest intersection**

Color = color of triangle point + shading

NOTE:

- *Pixel coordinates are already converted to 3D for you*
- ***Skip the test t<1** from the algorithm for ray intersection with triangles. All our triangles are in front of the screen and the screen is at 0.5 distance from origin so the test should be "if t<0.5 then behind the screen".*