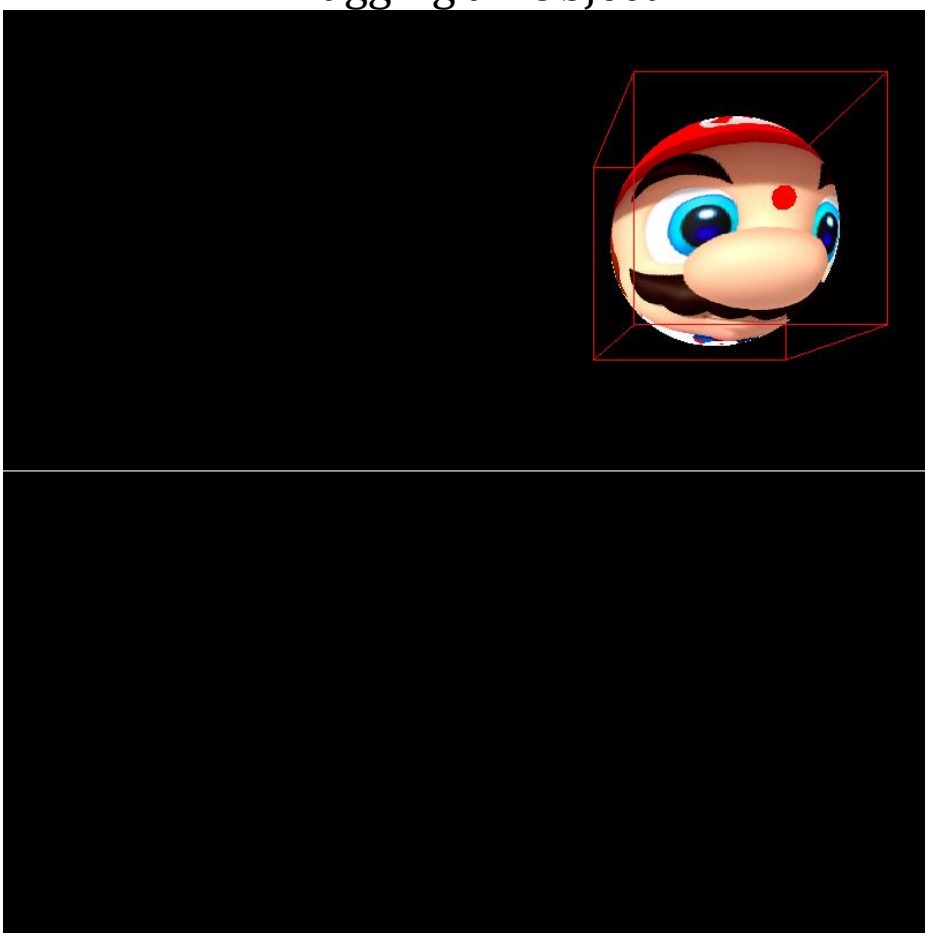
Dragging an Object



Description:

We have figured out how to intersect with an object using orthographic projection. Now we are going to extend that to using perspective transform, and then utilize that to do more advanced interaction. In particular, we're going to be using ray casting to implement: (1) 3D object selection and (2) dragging of a sphere.

Your Task:

- As a background, in this lab, the sphere is sitting at spherePosition, which is initialized to (0, 0, 0). Your camera is sitting at (0, 0, 3) and looking down the negative z axis (0, 0, -1). See updateCamera() and drawScene() for how the program calls Camera.h
    1. You will need your own Camera.cpp for this lab. Fingers crossed that it's working!!
- There are 3 different components to this lab:
    1. Extend Lab 6 to do sphere intersections using your Camera.cpp (that is, use perspective transform). Just like in Lab 6, a left mouse click triggers a ray. You should be able to reuse your sphere-intersection code from Lab 6, but your **generateRay**,

getEyePoint, **getIsectPointWorldCoord** functions will need to be updated.

2. Once you can intersect with a sphere using perspective transform, you will need to be able to "drag" the sphere using the right mouse button. A few notes on this:
    - You need to be careful about making sure that when you move the sphere, the sphere doesn't "snap" to where your mouse point is (that is, you cannot set the **spherePosition** to be where your mouse is pointing). Instead, be mindful that the same **isectPoint** is always under the mouse.
    - It's a little ambiguous as to what the "depth" of the new sphere should be. As a general rule of thumb, use the old "**t**" value to determine where along your ray you should place the sphere.
    - The function that you will be editing for this subtask is: **handle**. In particular, the **FL_DRAG** sub-routine (search for "TODO"). Note that you will likely need to read the sub-routine under **FL_PUSH** to have an understanding of how dragging works.
3. You might have noticed that once the sphere is no longer at the origin, you might need to update your sphere-**intersection** function to take into account the transform.

- While this seems like quite a bit of work for a lab, keep in mind that (1) and (3) are both part of your Assignment 4. So time spent on this lab to get those working will not be wasted!!

Going Further:

Did you enjoy this in class assignment?

- Add handles in the interface for rotation and scaling.
    1. Try doing a non-uniform scale
    2. How would you add interactions for object rotation?
    3. How do you manipulate the "depth" of where to place the object?