Computer Graphics
Lab: Picking

This lab explores picking in OpenGL. All files are available online.

1 Simple Manual Case

Copy the file named picklab1.c into your directory. It draws a red rectangle in the upper left corner of the viewport, and a blue rectangle in the lower right corner. Create a square viewport that is hard-coded to the size of the window (400 X 400), located at the origin. In the mouse function, add code that checks to see if the cursor falls within the bounds of either of the two rectangles. If it does, print out a message that states which rectangle has been picked. Execute the program without changing the window size to make sure that your code works. Now change the window as follows.

1. Stretch the window to the right by dragging the right side.
   **Q1:** Does this affect picking? Why or why not?

2. Restore the window to the orginal size and stretch the window to the left by dragging.
   **Q2:** Does this affect picking? Why or why not?

3. Restore and stretch the window toward the top by dragging the top edge.
   **Q3:** Does this affect picking? Why or why not?

4. Stretch the window toward the bottom.
   **Q4:** Does this affect picking? Why or why not?

2 Simple Manual Case with Variable Viewport

Modify the above program so that reshape function changes the size of the viewport when the window size changes. The viewport should remain a square at all times (see interaction lab). Everything else in the program should remain the same. Compile, and change the window as follows.

1. Stretch the window to the right.
   **Q5:** Does this affect picking? Why or why not?

2. Stretch the window to the left.
   **Q6:** Does this affect picking? Why or why not?

3. Stretch the window toward the top.
   **Q7:** Does this affect picking? Why or why not?

4. Stretch the window toward the bottom.
   **Q8:** Does this affect picking? Why or why not?

3 Manual Inverse Mapping with Variable Viewport

We can modify the above program so that we map the current cursor position into the projection plane. Once we have done this, we can use the same criteria for determining whether we have picked one of the rectangles, but instead of using the actual cursor coordinates, we will use the inverse-mapped coordinates. The mapping is based on the ratio of the current viewport dimensions to those of the projection plane (which is fixed in our example). This can be done by your supplying the appropriate computations for doing the inverse mapping (fairly easy in 2D), or letting OpenGL do the reverse mapping for you using `gluUnproject`. Modify the code accordingly, and test it. If your methodology is correct, you will get hits whenever you click in a rectangle regardless of the size of the window and the size of the viewport.

**Q9:** Submit your code for inverse mapping.

Note that in all of the above, we are dealing with a fairly simple situation. The view volume/projection plane is anchored at the origin, the view volume is fixed, and the viewport is a square. For a challenge, consider the situation in which the view volume is *not* located at the origin of the window, and one in which the width to height ratio of the projection lane is not one. And of course, the greatest simplification above is that we are working in 2D, and not 3D!
4 Picking with the Back Buffer

This part uses file *picklab2.c* The program simply prints out the color of the picked pixel. This is fine for demonstration purposes, but is not practical at all. Modify this program as follows.

Store the pertinent color data for each object in an array indexed by the name of the object. If you want to do it right, an array of structs would be more appropriate, where each struct stores object data that would include the color info, its name (so that the name would not necessarily have to coincide with the object’s position in the array), and its vertices. When the mouse button is pressed, your code should do a search of the object array and print out the name of the picked object.

Q10: Submit your code that does the object identification.

As you can see, searching an array against three values will be rather clumsy for large numbers of objects. A hash table might be more appropriate.