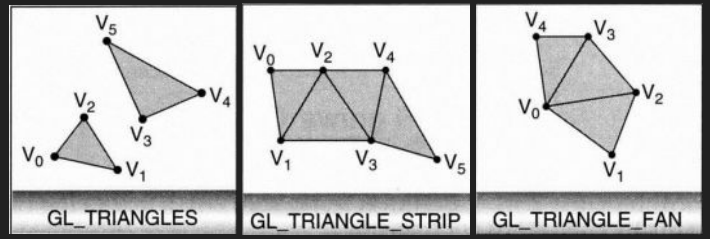
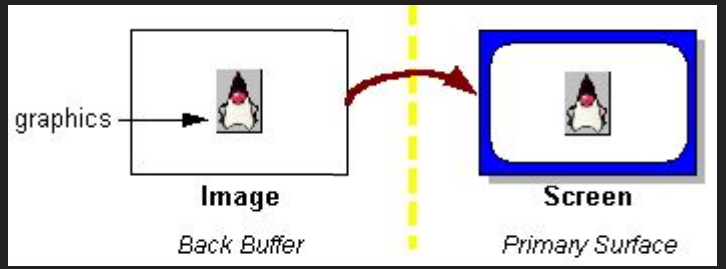
Question 1

- OpenGL supports the **`GL_TRIANGLES`** primitive type. Why do you think that OpenGL also supports **`GL_TRIANGLE_FAN`** and **`GL_TRIANGLE_STRIP`**?
  - *Usually*, a surface can be defined using many **connected triangles** (a triangle mesh), and these triangles **share vertices** with neighbouring triangles.
  - `GL_TRIANGLE_FAN` and `GL_TRIANGLE_STRIP` allows us to **send fewer vertices to the rendering pipeline** to draw the surface. For example, to draw a disc made of 20 triangles, `GL_TRIANGLES` would require 60 vertices but `GL_TRIANGLE_FAN` only needs 22 vertices. This saves system bus bandwidth and reduces the amount of vertex processing on the graphics hardware.



Question 2

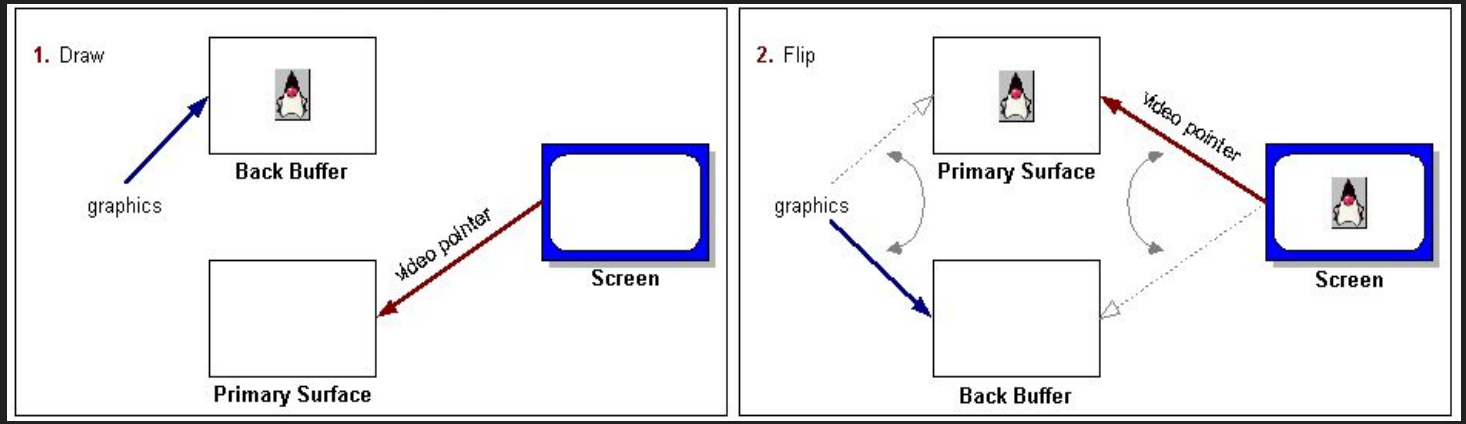
- How does Double Buffering work? Why do we use it?
  - Double Buffering utilises two color buffers:
    - **Front Buffer** - The one that is being displayed on the display device.
    - **Back Buffer** - Where all the rendering operations are applied to.
  - Double-buffering **prevents a partially rendered frame from being displayed** on the display device, i.e. only when the Back Buffer is done with rendering our graphics do we copy it over to the Front Buffer. This reduces the *perceived* **flickering** effect during **animation**.



Read More: https://docs.oracle.com/javase/tutorial/extra/fullscreen/doublebuf.html

Question 2

- How does Double Buffering work? Why do we use it?
  - Page-Flipping can also be employed. A **video pointer** is used to switch between the two buffers.
  - Page-Flipping is used to eliminate *perceived* **tearing**: a splitting effect that occurs when drawing to the screen happens faster than the screen's refresh rate.

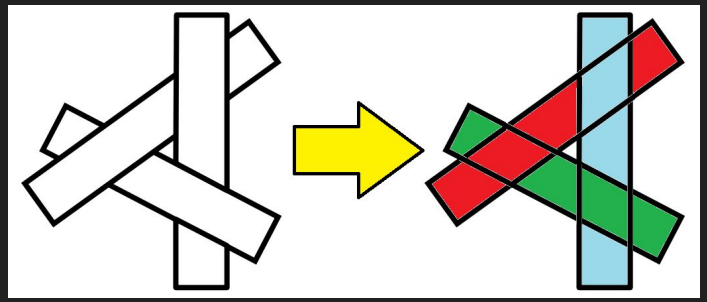


Question 3

- What is the use of the GLUT function **`glutPostRedisplay()`**?
  - The execution of the `glutPostRedisplay()` function tells GLUT to call the display callback function **at the end of the current event loop**.
  - This is done by setting a **flag**, which GLUT will check at the end of the event loop (and execute the function since the flag has been set).
  - Why don’t we just call the display callback function directly to update the image?

Question **4**

- Hidden Surface Removal (HSR) is not necessary if we can **sort the polygons** in a **back-to-front order** and render these polygons in that order. Is it always possible that any set of polygons can be sorted in a back-to-front order? Show examples.
    - **It is not always possible**. The following diagram shows an example in which the polygons **occlude one another cyclically**; there is no sorting! One way to fix this problem is to “**split**” the **original polygons**.



Question 5

A. What is an OpenGL viewport?

B. How do you specify one?

C. Can we have multiple viewports in a window?

D. Can a viewport be larger than the window?

E. If yes, what will happen?

F. When you use `glClear(GL_COLOR_BUFFER_BIT)`, are you clearing the entire window or just the viewport?

Question 5

A. What is an OpenGL viewport?

- An OpenGL viewport defines a rectangular region of the window in which OpenGL can draw.

B. How do you specify one?

- We specify one by using the function `glViewport(x, y, w, h)`, where `x` and `y` specify the window location (in pixels) of the lower left corner of the viewport, and `w` and `h` specify the width and height of the viewport in pixels.

C. Can we have multiple viewports in a window?

- Yes. They can even overlap each other.

Question 5

D. Can a viewport be larger than the window?
   - Yes.

E. If yes, what will happen?
   - In this case, what is supposed to be in the viewport but is outside the window region will not appear.

F. When you use `glClear(GL_COLOR_BUFFER_BIT)`, are you clearing the entire window or just the viewport?
   - It clears the entire window.

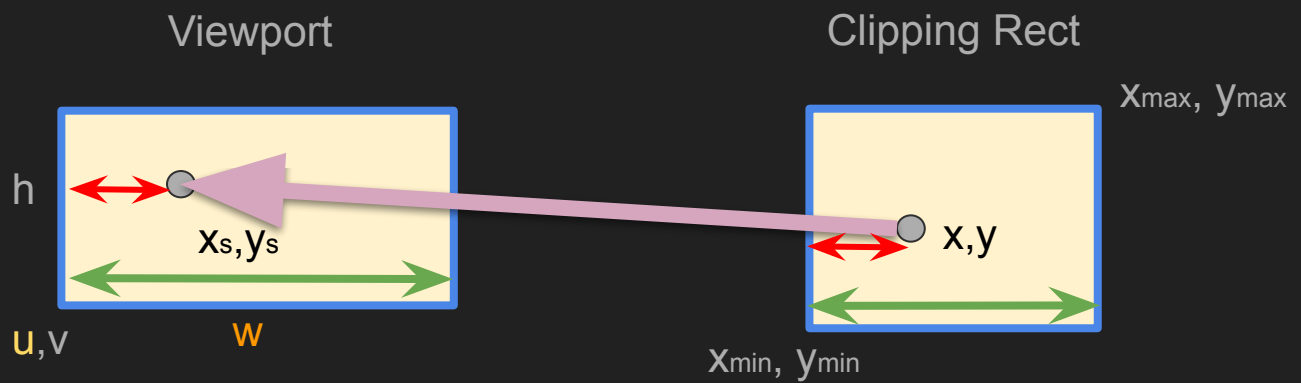
Question 6

- Assume we have the following OpenGL function calls:

```
glViewport(u, v, w, h);
 ...
gluOrtho2D(x_min, x_max, y_min, y_max);
```

Find the mathematical expressions that map a point (x, y) that lies within the **clipping rectangle**, to a point (x_s, y_s) that lies within the **viewport**.

Question 6



$$\frac{x - x_{min}}{x_{max} - x_{min}} = \frac{x_s - u}{w} \rightarrow x_s = u + \frac{w(x - x_{min})}{x_{max} - x_{min}}$$

Question 7

- In many old CRT monitors, the pixels are not square. Let’s assume the pixel width-to-height aspect ratio is 4:3. Suppose in the camera coordinate frame, there is a disc in the z = 0 plane, ced at (100, 200, 0), and has a r of 10. You want to draw the entire disc as big as possible inside the window, and it should appear circular and not oval.
    a. If the window size is 600x300 (Width x Height), how would you set up the viewport and the orthographic projection using OpenGL?
    b. What if the window size is now 300x600?
    c. What if the window size is now 300x320?

Question 7

- Recall **Lecture 3 Slide 34**, where `winWidth` and `winHeight` are **fixed**:

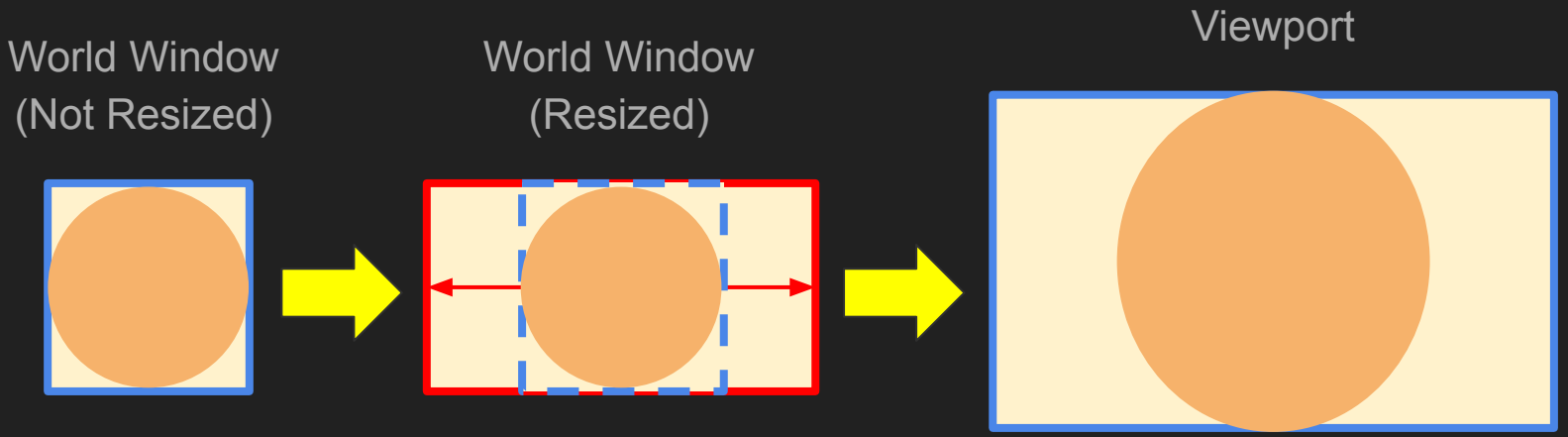
```
glViewport(0, 0, winWidth, winHeight);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

if (winWidth >= winHeight) { // Assume that we execute the if-branch.
    gluOrtho2D(-x * winWidth / winHeight, x * winWidth / winHeight, -y, y);
}
else {
    gluOrtho2D(-x, x, -y * winHeight / winWidth, y * winHeight / winWidth);
}
```

What is this doing to the viewport and world window?

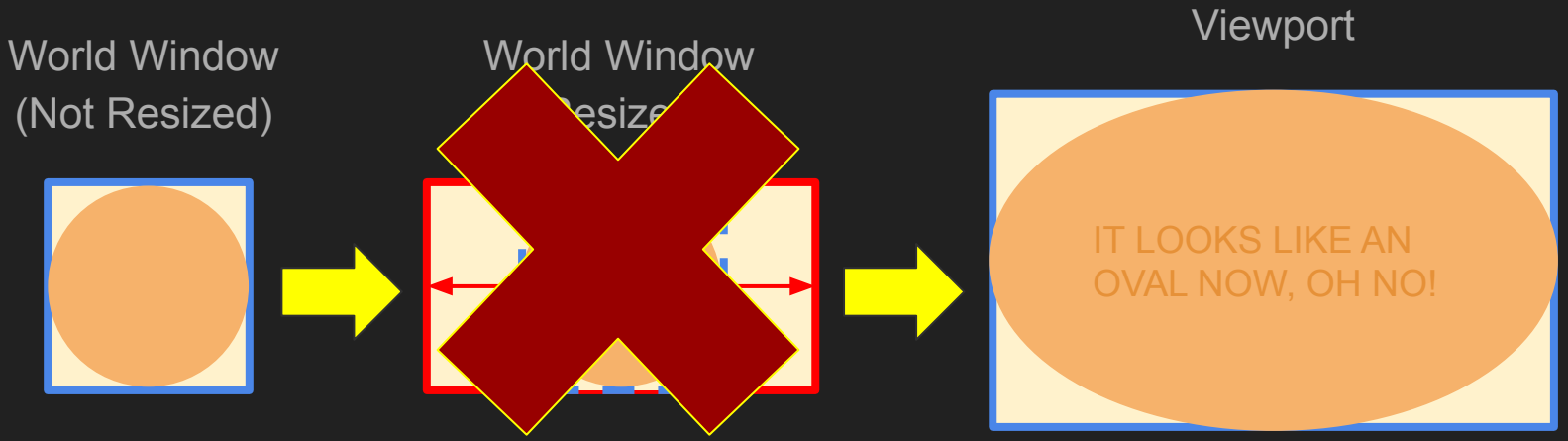
Question 7

- When **`winWidth >= winHeight`**, we have to **resize our world window** such that the viewport and world window have the **same aspect ratio**. This helps us to preserve the shapes.



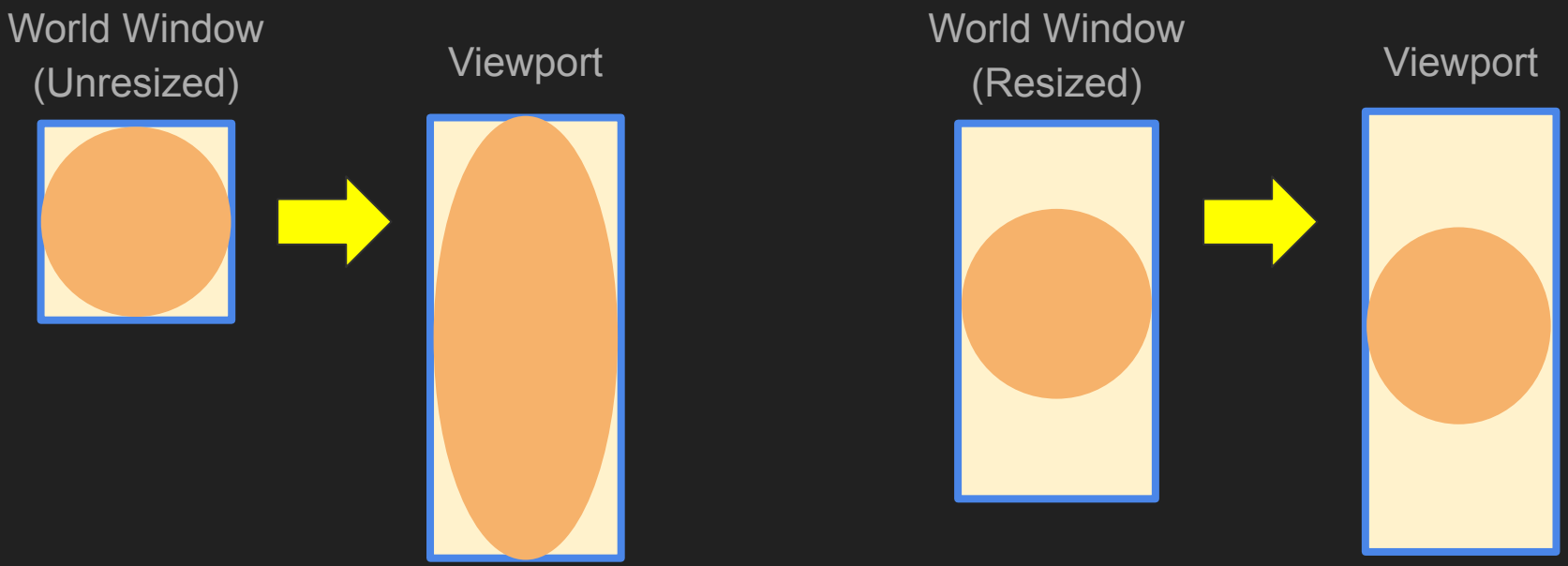
Question 7

- If we do not resize the World Window, we will not be able to preserve the actual shape in the Viewport.



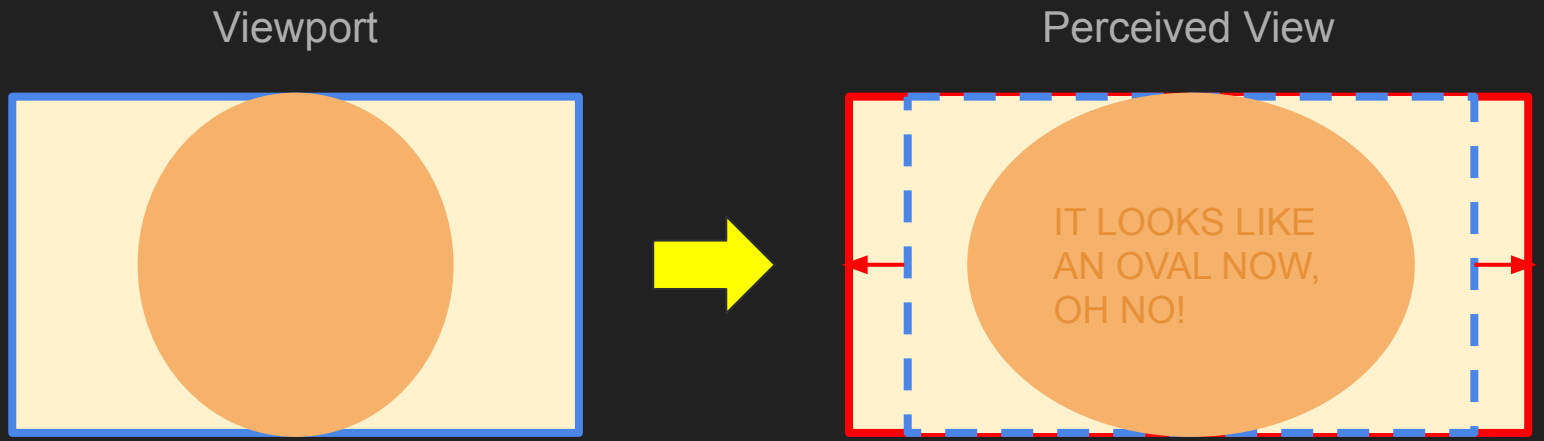
Question 7

- When `winWidth < winHeight`, resize accordingly (in the else-block):



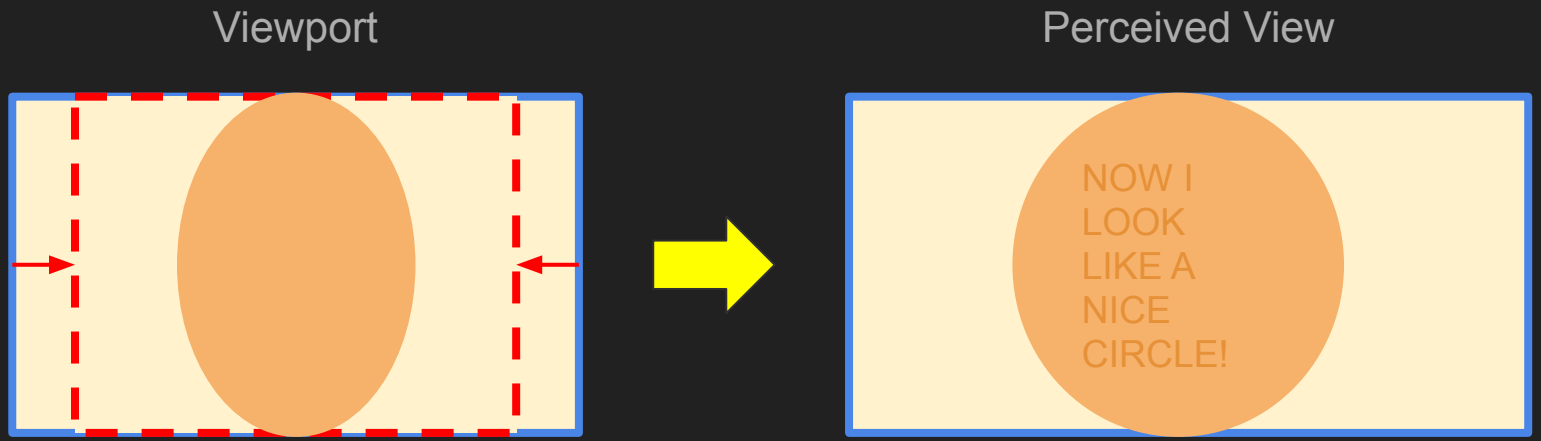
Question 7

- But what do we actually *perceive* if the pixel width-to-height aspect ratio is 4:3 instead of 1:1? Hint: It's no longer a circle...



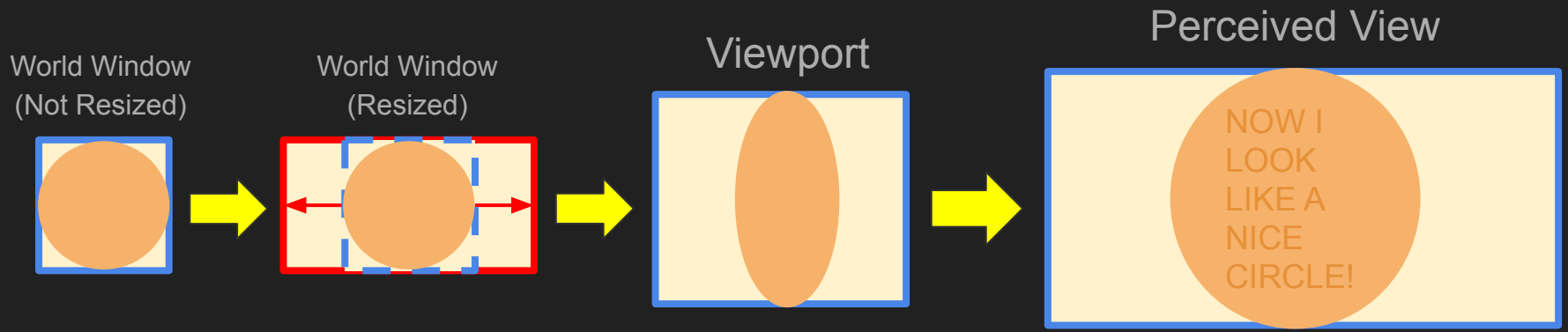
Question 7

- Now, we also need to account for the pixel width-to-height ratio! So we have to adjust our viewport somehow...



Question 7

- ...but wait, since Viewport window size is already fixed (as specified in the 3 questions), that means we can only adjust our World Window! Please note that these diagrams are not to scale…



Question 7

```
double pixelAspectRatio = 4.0/3.0;
int winWidth = 600; // Change the width and height for questions A, B and C.
int winHeight = 300;
double c[2] = { 100.0, 200.0 };
double r = 10.0;

glViewport(0, 0, winWidth, winHeight); // Viewport Set-up.
glMatrixMode(GL_PROJECTION);
glLoadIdentity();

double apparentWinHeight = winHeight/pixelAspectRatio;
if (winWidth >= apparentWinHeight)
     gluOrtho2D(c[0] - r * (winWidth/apparentWinHeight),
                c[0] + r * (winWidth/apparentWinHeight), c[1] - r, c[1] + r);
else
     gluOrtho2D(c[0] - r, c[0] + r, c[1] - r * (apparentWinHeight/winWidth),
                                    c[1] + r * (apparentWinHeight/winWidth));
```