

Tutorial 2

CS3241 Computer Graphics (AY22/23)

August 30, 2022

Wong Pei Xian



eo389023@u.nus.edu

Attendance taking

Question 1

What is a GLUT display callback function? Give example events for which the display callback function should be called.

GLUT function

GLUT: OpenGL Utility Toolkit (Lecture 2 slide 10),

a **library** that provides I/O functionality common to [all window systems](#).

```
// Register the callback functions.  
glutDisplayFunc( MyDisplay );  
glutReshapeFunc( MyReshape );  
glutMouseFunc( MyMouse );  
glutKeyboardFunc( MyKeyboard );  
  
glutIdleFunc( UpdateAllDiscPos ); //*** MODIFY THIS ***
```

GLUT display callback

- `glutDisplayFunc()`
- User-defined callback to register
- Executed on each window refresh.

Question 2

What is the use of the GLUT function `glutPostRedisplay()`?

glutPostRedisplay

The execution of the `glutPostRedisplay()` function tells GLUT to call the display callback function at the end of the current event loop.

Question 2

When do we want to call the `glutPostRedisplay()` function?

When we explicitly want the rendered image to be updated.

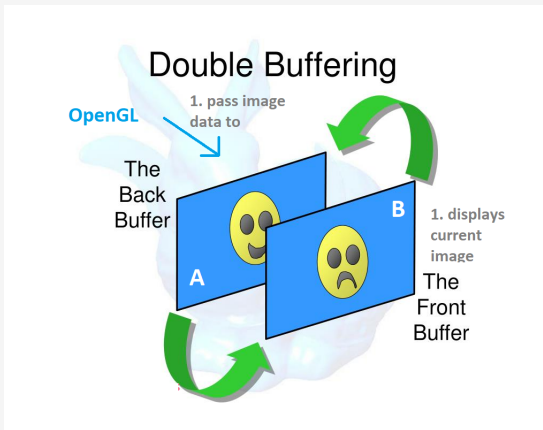
Q: Why don't we call the display callback function directly to update the image?

A: **Multiple calls** to `glutPostRedisplay` may be made in a single iteration of main loop, we don't want to redisplay (reapply graphics to buffer) everytime. [\[Reference\]](#)

Question 3

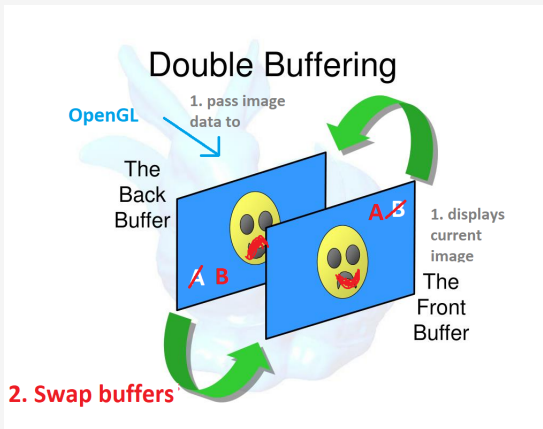
How does double buffering work?

Double buffering



- Back buffer: apply graphics **WHILE**
- Front buffer: display graphics

Double buffering

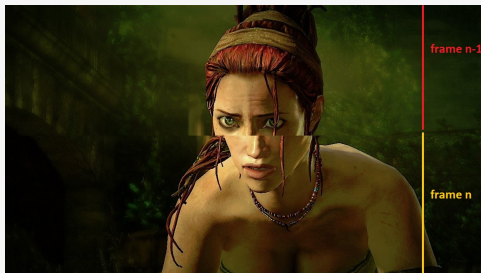


Swapping is fast and seamless.

Question 3

Why do we use double buffering?

Prevents screen tearing



Screen tearing: when the rate of graphics feed application \neq window refresh rate.

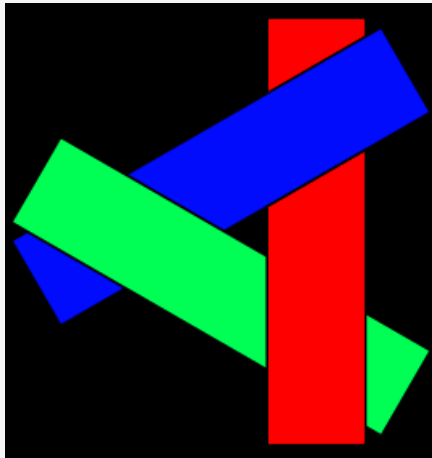
Notice how double buffering solves this by making sure graphics are not applied to the currently displayed frame, only swapping the frames when the application is complete.

Question 4

The use of any special hidden surface removal method is not necessary if we can sort the polygons in a back-to-front order and render these polygons in that order. (Tutorial 1 Q6)

Is it **always possible** that any set of polygons can be sorted in a back-to-front order?

Cyclic overlap



Question 5a

What is an OpenGL viewport?

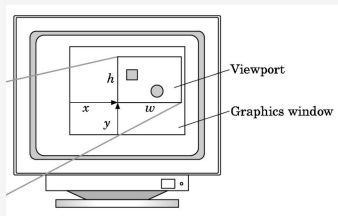
Viewport

OpenGL viewport: A rectangular region of the window in which OpenGL can draw.

Question 5b

How do you specify one?

glViewport



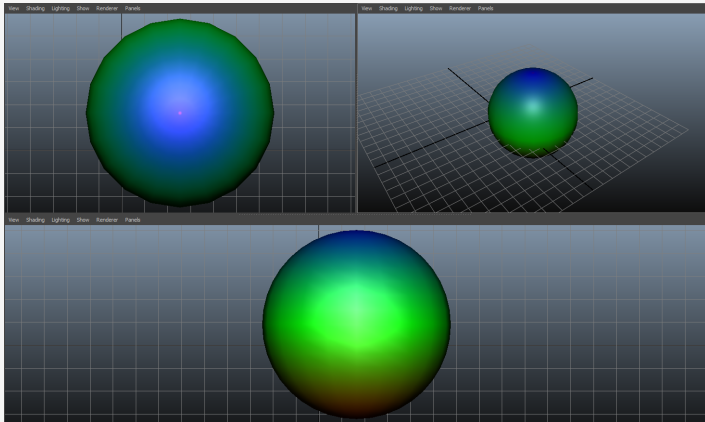
```
glViewport(GLint x, GLint y, GLsizei w, GLsizei  
           h)
```

Note: x, y, w, h are in window coordinates.

Question 5c

Can we have **multiple viewports** in one window?

Yes!



QUESTION 1
○○○

QUESTION 2
○○○○

QUESTION 3
○○○○○

QUESTION 4
○○

QUESTION 5
○○○○○○●○○○○○

QUESTION 6
○○

QUESTION 7
○○○○○○○○○○○○

Yes!



Question 5d, 5e

Can a viewport be larger than the window? If yes, what will happen?

Yes!

C++

 Copy

```
void WINAPI glViewport(  
    GLint x,  
    GLint y,  
    GLsizei width,  
    GLsizei height  
);
```

Parameter types are GLint for x and y coordinates, so they can be negative and go out of the screen.

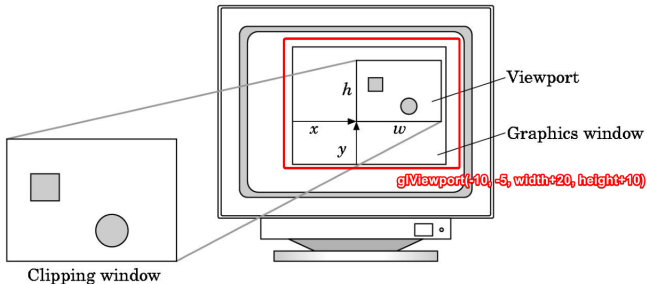
Or width or height could also exceed window size.

Viewport size is independent of window size.

Specification example

Viewports

- Do not have to use the entire window for the image:
`glViewport(x, y, w, h)`
 - Values in pixels (window coordinates)



Question 5f

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

Question 5f

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

Answer: the window.

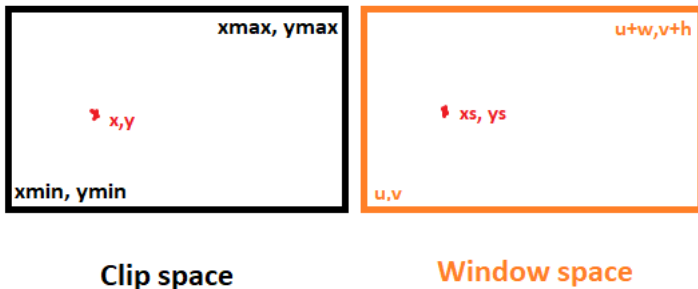
Question 6

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 (xs, ys) that lies within the viewport.

Clip space to window space



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

$$y_s = v + (y - y_{\min}) \left(\frac{h}{y_{\max} - y_{\min}} \right)$$

Question 7a

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, centered at $(100, 200, 0)$, and has a radius of 10.

You want to draw the entire disc as big as possible inside the window, and it should appear circular and not oval.

If the window size is ____, how would you set up the **viewport** and the **orthographic projection** using OpenGL?

- 600×300
- 300×600
- 300×320

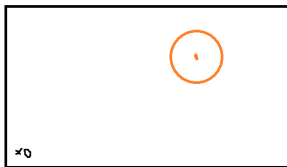
Template

```
glViewport(u, v, w, h);
glMatrixMode(          );
glLoadIdentity(); /// Reset matrix

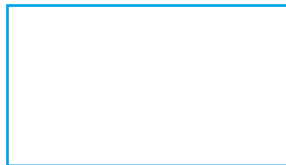
double apparentHeight =          ;

// Setup projection matrix
if (          ) {
    gluOrtho2D();
} else {
    gluOrtho2D();
}
```


Visualize

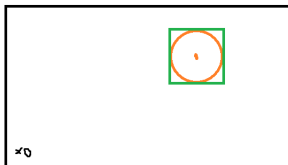


Camera Space

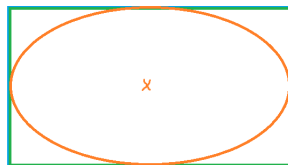


Window Space (Viewport)

Consider the case where the pixels are square first. Let w , h be the width and height of the viewport, c be the 2D coordinates of the center of the circle, and r be the radius.

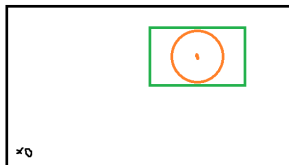


Camera Space

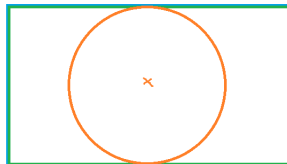


Window Space (Viewport)

```
glViewport(0, 0, w, h);  
glOrtho(c.x - r, c.x + r, c.y - r, c.y + r);
```



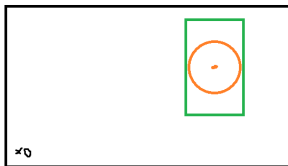
Camera Space



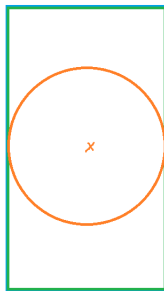
Window Space (Viewport)

Assuming the pixels are square, to get this we can:

```
glViewport(0, 0, w, h);  
glOrtho(c.x - r * w/h, c.x + r * w/h,  
c.y - r, c.y + r);
```



Camera Space



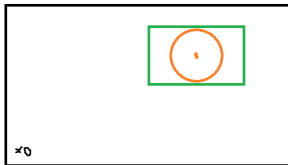
Window Space (Viewport)

Assuming the pixels are square, to get this we can:

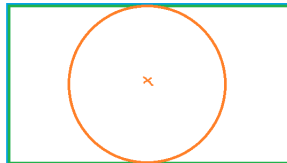
```
glViewport(0, 0, w, h);  
glOrtho(c.x - r, c.x + r,  
c.y - r * h/w, c.y + r * h/w);
```

What if we consider the 4:3 pixels?

Then we have to make sure the clipping space scales to the **apparent aspect ratio**, i.e. $\text{apparentWidth} = w \times \frac{4}{3}$.



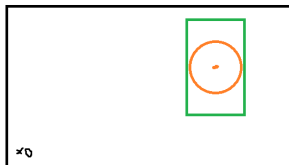
Camera Space



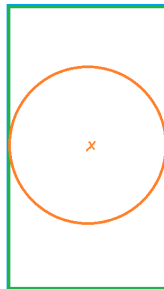
Window Space (Viewport)

Assuming the pixels are 4:3, to get this we can:

```
glViewport(0, 0, w, h);  
glOrtho(c.x - r * apparentWidth/h ,  
c.x + r * apparentWidth/h, c.y - r, c.y + r);
```



Camera Space

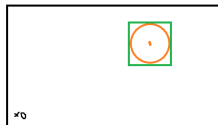


Window Space (Viewport)

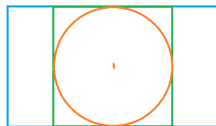
Assuming the pixels are 4:3, to get this we can:

```
glViewport(0, 0, w, h);  
glOrtho(c.x - r , c.x + r,  
c.y - r * h/apparentWidth,  
c.y + r * h/apparentWidth);
```

Alternative: scaled viewport



Camera Space



Window Space (Viewport)

The pixels are 4:3.

```
int squishedWidth = w * 3/4;  
glViewport(0, w / 2 - squishedWidth / 2,  
squishedWidth, h);  
glOrtho(c.x - r , c.x + r, c.y - r, c.y + r);
```

You can similarly account for the case where $h > w$.

Key takeaway

In 2D orthographic projection, **aspect ratios must match** between the clipping space and the window space (**assuming uniform pixels**) to not be distorted.

Thanks! Get the slides here after the tutorial.



<https://trxe.github.io/cs3241-notes>