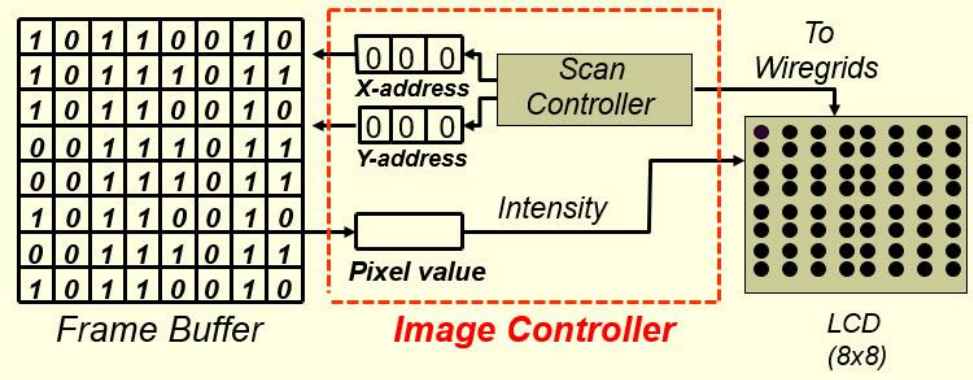
# CS 535 Computer Graphics

## Homework Assignment 1 Solution Set (40 points)

## Due: 9/5/2024

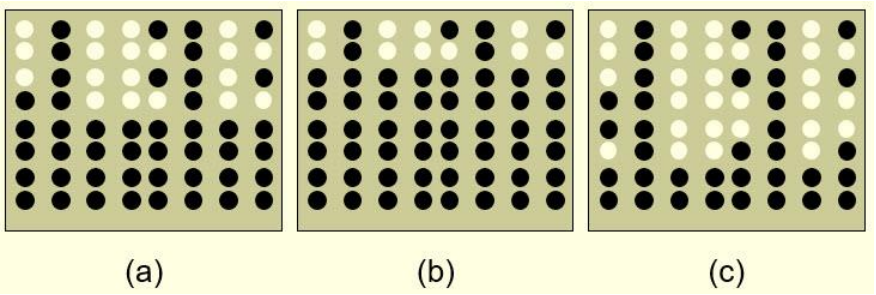
1. Given the following simple two-color raster scan system (a frame buffer with 8 x 8 entries, an image controller (also called a display processing unit) and a LCD display with a resolution of 8 x 8 pixels)




if the refresh rate is 60 Hz (meaning the screen will be refreshed 60 times per second), then

(a) after the first 3/240 seconds, how would the screen look like, the one shown in (a), (b), (c), or, none of these? (4 points)

(b) after the first 9/240 seconds, how would the screen look like, the one shown in (a), (b), (c), or, none of these? (4 points)




**Sol**:

(a) Each refresh cycle of this raster scan system is 1/60 seconds. 3/240 seconds is 3/4 of 1/60 seconds. So after the first 3/240 seconds, only 3/4 of the frame buffer have been processed, so only 3/4 of the image stored in the frame buffer would have been rendered and shown on screen, the remaining 1/4 of the screen is still dark. Hence the answer

is (c).

(b) 9/240 seconds = 2*(1/60) + 1/240, and 1/240 seconds is 1/4 of 1/60 seconds.

This means, after 9/240 seconds, the raster scan system has finished 2 refresh cycles and ¼ of the third refresh cycle and is about to finish the next 75% of the 3rd next refresh cycle. So, theoretically we would think the answer is (a). But remember that, after the first refresh cycle, the entire image is shown on screen, the following refresh cycles are simply to prevent the image from fading away. So after the first 9/240 seconds, you would see the entire image shown on screen, not just 1/4 of it. So the correct answer in this case is “none of these”.

2. What is the main difference between the *event mode* and the other two input modes (*request mode* and *sample mode*)? Why? (4 points)

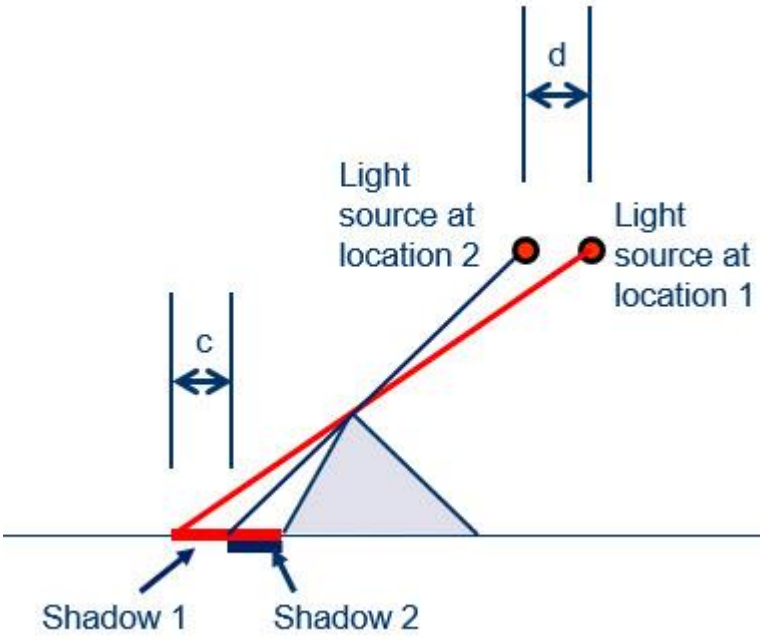
**Sol:** In event mode the 'waiting time looping' problem would not occur because in this mode OS does not interrupt the application program when receiving I/O events. The events are queued in the “event queue” and monitored and processed by OpenGL. OpenGL will retrieve events from event queue and process them, not the application program itself.

3. A modern *optical mouse* does not have to be operated on a special pad, it can be operated on any surface. Given a piece of black glass and a piece of transparent class, equally smooth, a modern optical mouse would do better on which glass and why? (4 points)

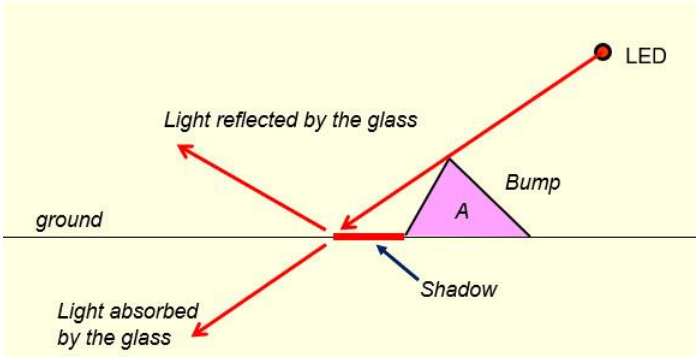
**Sol:**

It would do better on the transparent glass.

When a surface is lit at a grazing angle by the LED of a mouse, the texture of the surface casts shadows on itself and the difference between the shadows (‘c’ in the following figure) is used by the Displacement Processor of the mouse to compute the displacement vector (‘d’ in the following figure, distance the mouse has moved between location 1 and location 2) and send it to the computer to update the location of the cursor on screen.

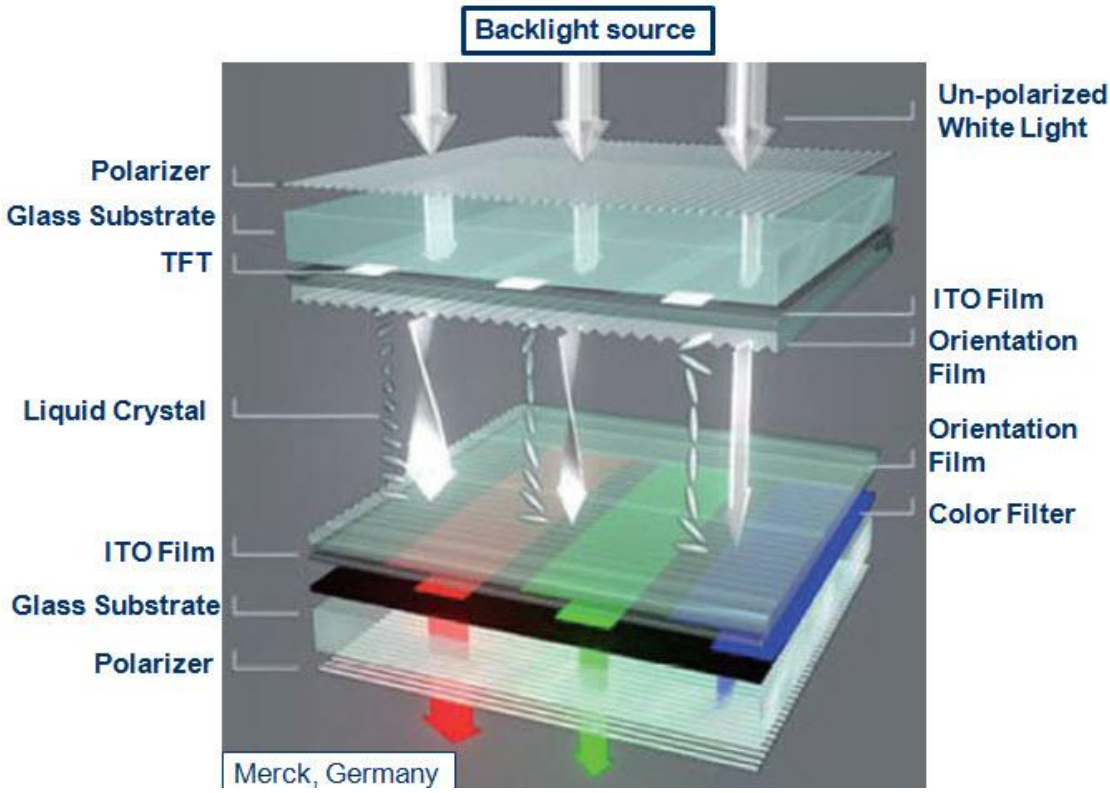



If the mouse is operated on a piece of glass, there are two problems. First, a significant amount of the light emitted by the LED will be absorbed by the glass, only a small portion will be reflected by the glass (see the following figure). Therefore, the images taken by the sensor of the mouse sometime will not be bright enough for the computation process. Second, a more serious problem is that, since the surface of a glass is very smooth, shadows cast by the texture of the glass surface on itself are either too small or there are no shadows at all. Combined, it would be difficult for the Displacement Processor to identify the locations of the LED at different moments based on difference of the shadows. So it would be difficult for the Displacement Processor to compute the distance the mouse has moved either.



For a piece of black glass, the first problem is even more serious because most of the light emitted by the LED will be absorbed by the glass, almost nothing will be reflected. The contrast between the shadow and the neighboring region will be so small that one cannot even tell where the boundary of the shadow is. Therefore, computing the value of c in the first figure is not possible for most of the points of the black glass surface.

4. In the following figure, what are the functions of the ITO film, the Orientation film and the Color filter ? You have to do a little research here. (10 points)




**Sol:**

**ITO** (indium tin oxide) **film** – ITO is transparent and conductive, therefore, can be used to build transparent electrode patterns (playing the role of vertical and horizontal grids) on the glass substrates in an LCD display.

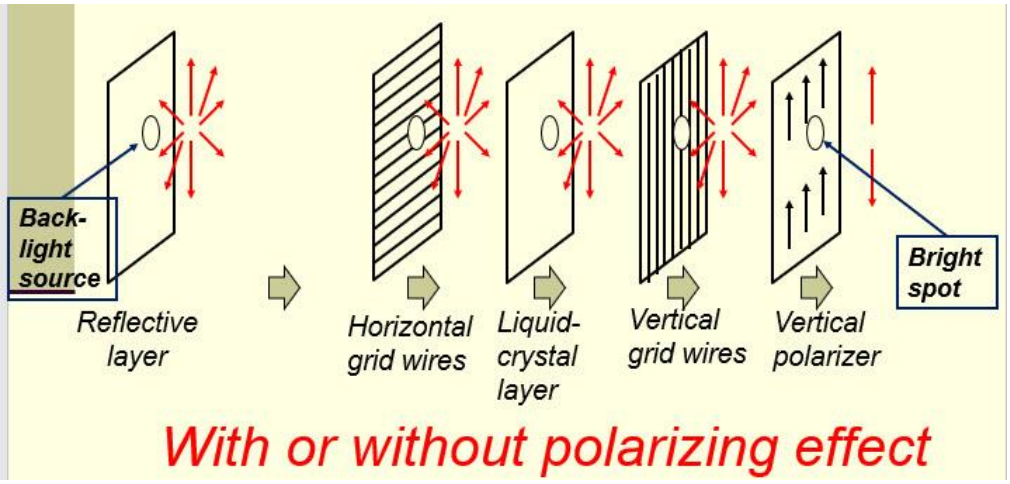
**Orientation film** – this film is used to control the initial arrangement of the liquid crystal molecules (controlling the molecular arrangement of the liquid crystal affects the stability of the image displayed in the LCD).

**Color Filter** – a color filter is made of colored glass which lets the wavelengths of its own color pass and block any other. For instance, a red color filter absorbs all lights in the spectrum except the red component.

5. In a transmissive Liquid-crystal Display (LCD), if we remove the horizontal polarizer layer from the LCD, what would happen and why? (6 points)

**Sol:**

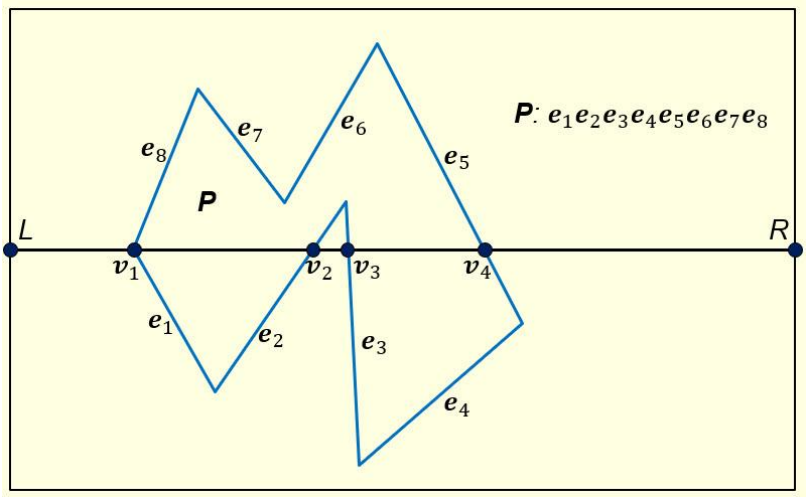
Without the horizontal polarizer layer, for a black and white display, the entire screen would turn white because no matter if the liquid crystal layer has polarizing effect or not (the figure below), light will always go through the vertical polarizer layer, so each pixel of the screen will turn white. For a color display, all dark pixels of an image will turn white. So, in either case, you would not be able to generate a correct image on screen.




6. In the notes "2D Raster Algorithms", an algorithm to efficiently scan convert a 2D polygon is introduced. For each active edge (an edge that intersects the current scanline), the algorithm gets the intersection point of the edge with the current scanline with only one floating-point addition, and there is no need for the algorithm to sort the intersection points. However, when building the bucket-sorted edge table (ET) for the given polygon (such as a triangle), certain edges should be shortened by one unit in y-direction and certain edges should be removed from the edge table. Why? (8 points)

**Sol**:

To ensure the number of intersection points of a scan line with the bounding edges of a polygon to be a (finite) even number all the time. This property is needed in the scan conversion process of a polygon.




Consider the case shown in the above figure. The scan line and polygon edge intersection process is done in the clockwise or counterclockwise order, depending on the representation of the polygon. In this case, it is done in counterclockwise order.

When edge e1 is considered, we get an intersection point v1. For e2, we get v2. For e3, we get v3. Nothing for e4. Then v4 for e5. Nothing for e6, e7 and then v1 again for e8. So, totally we will get five intersection points:

v1, v2, v3, v4, v1

After sorting, we get

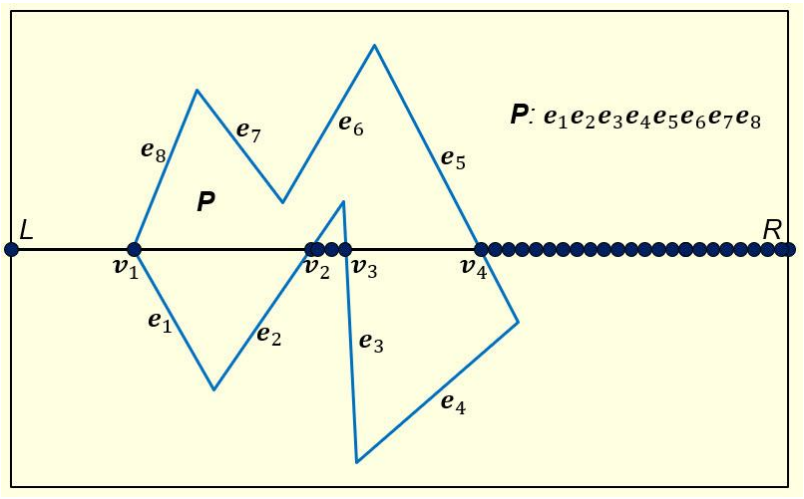
v1, v1, v2, v3, v4

When we group them into pairs, we get

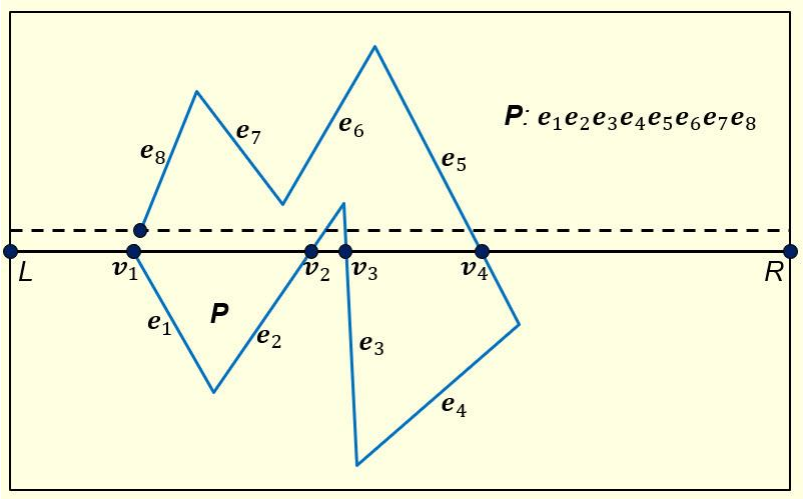
(v1, v1), (v2, v3), (v4, R)

where R is the right end of the scan line (the last intersection point will be paired with the right end of the scan line if that intersection point is an odd point). When we fill in the pixels between the left end point and the right end point of each pair, we get the following result for the current scan line.

The reason we get into this problem is because v1 has been reported as an intersection point twice, once by edge e1 and once by edge e8.




One way to avoid this problem is to shorten edge e8 by one unit (one pixel) in y-direction (see the following figure). This way, edge e8 will not report v1 as an intersection point when it is considered in the intersection computation process.




Therefore, we will only get four intersection points for the current scan line:

v1, v2, v3, v4

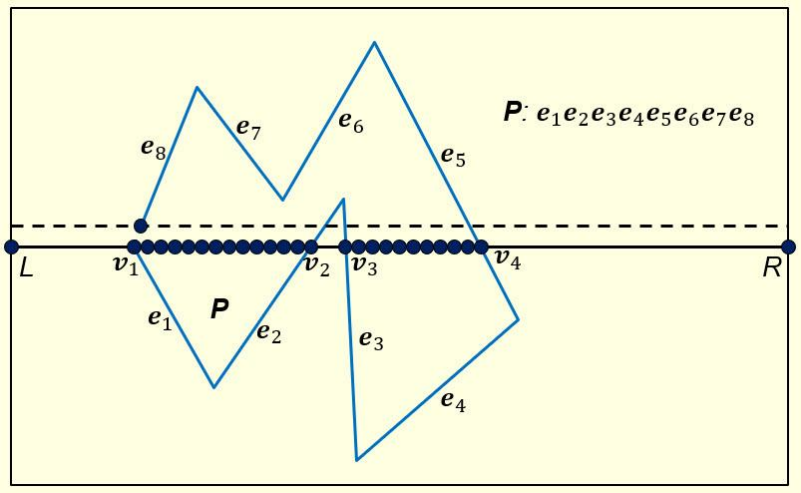
Note that we will not lose any information doing things this way because there is no scan line to consider between the current scan line and the scan line that passes through the new lower vertex of e8.

After sorting and grouping, we get the following two pairs:

(v1, v2), (v3, v4)

Filling in the pixels between the left end point and the right end point of each pair, we get the following result for the current scan line (see the following figure). This is exactly what should happen for the current scan line.

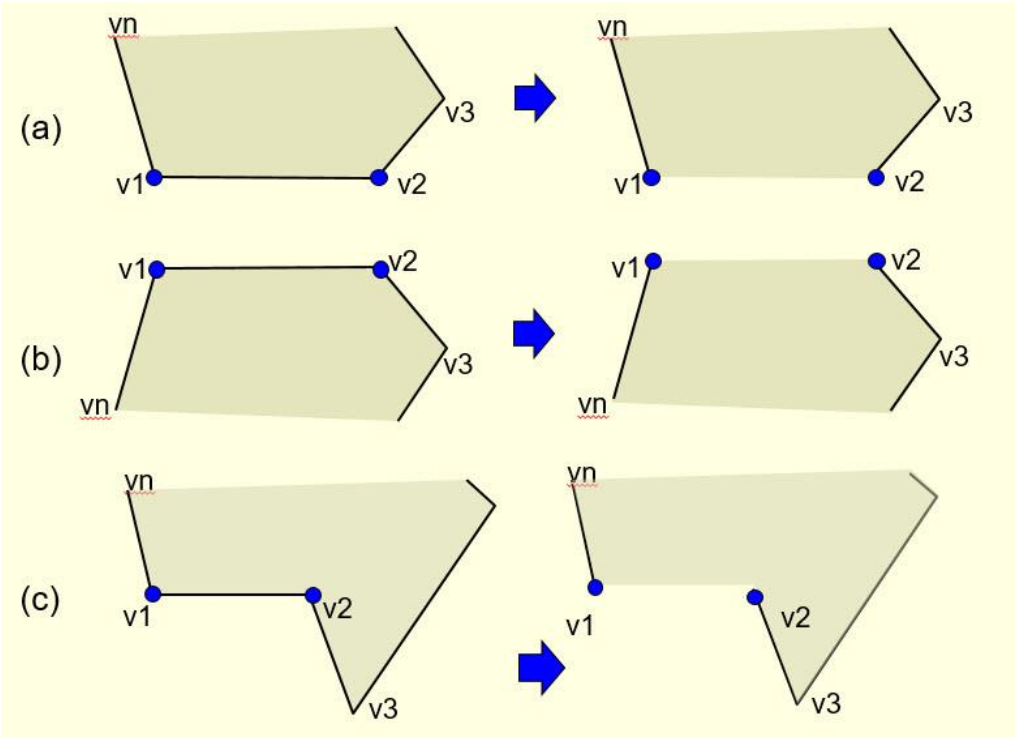
So, the remedy is: for each edge of the polygon whose lower vertex is not a local maximum nor a local minimum, shorten the edge by one unit (pixel) in y-direction on its lower vertex. This process will take O(n) time only where n is the number of vertices of the polygon.




We also need to identify and remove horizontal edges of the polygon to ensure the number of intersection points of the scan line with the edges of the polygon to be a finite number.

For a horizontal edge like the case shown in (a) or (b) of the following figure, we simply ignore that edge when we build the bucket-sorted edge table (ET). For a case like the one shown in (c) of the following figure, in addition to ignoring that horizontal edge, we also need to shorten one of its adjacent edges by one unit in y-direction. However, in this case, a better approach is not to shorten the edge with vertices v1 and vn, but the edge with vertices v2 and v3 to get a result like the one on the right side of case (c) of the

following figure. Do you see why?




Note that removing edges and shortening edges can be done at the same time. So the complexity for the complete process is O(n) where n is the number of edges of the polygon.