

Q.2 a. Differentiate between raster and random scan displays. Also differentiate between horizontal and vertical retracing with examples. (6)

Answer:	Raster Scan	Random Scan
Resolution	It has poor or less Resolution because picture definition is stored as a intensity value.	It has High Resolution because it stores picture definition as a set of line commands.
Electron-Beam	Electron Beam is directed from top to bottom and one row at a time on screen, but electron beam is directed to whole screen.	Electron Beam is directed to only that part of screen where picture is required to be drawn, one line at a time so also called Vector Display .
Cost	It is less expensive than Random Scan System.	It is Costlier than Raster Scan System.
Refresh Rate	Refresh rate is 60 to 80 frame per second .	Refresh Rate depends on the number of lines to be displayed i.e 30 to 60 times per second .
Picture Definition	It Stores picture definition in Refresh Buffer also called Frame Buffer .	It Stores picture definition as a set of line commands called Refresh Display File .
Line Drawing	Zig – Zag line is produced because plotted value are discrete.	Smooth line is produced because directly the line path is followed by electron beam .
Realism in display	It contains shadow, advance shading and hidden surface technique so gives therealistic display of scenes.	It does not contain shadow and hidden surface technique so it can not give realistic display of scenes.
Image Drawing	It uses Pixels along scan lines for drawing an image.	It is designed for line drawing applications and uses various mathematical function to draw.

b. Explain the concept of virtual reality with examples. (4)

Answer:

The term VIRTUAL REALITY, or VR, has recently come into widespread use; however, what this term actually means can vary widely with who is using it. Most often it is used to mean a computer-generated "world" or non-real background setting, usually immersive to the senses, and often with interactive features. Frequently, the term "cyberspace" is used interchangeably with virtual reality. Cyberspace, as noted in the text, has been defined as "The place where you are when you're on the phone." (John Perry Barlow, Technology reporter and lyricist for the Grateful Dead.)

We have all seen or heard of examples of virtual reality systems. Usually, we see a person wearing a type of helmet covering his eyes. This helmet is actually a viewing device which displays a three dimensional picture of some computer-generated scene. The user may be wearing a type of glove connected so that when he moves his hand a computer generated hand will move in the picture he is viewing. He reaches forward into nothing and closes his hand about thin air. In the view in the

helmet the hand grasps a box on a table and moves it to another location. These systems exist now and are used in a wide variety of applications.

In Japan, when you wish to select a floor plan in order to have a house constructed, instead of looking at floor plans you can view a computer-generated picture of the house as seen by someone walking through it. Using a joystick you can turn and look around you, move from room to room, open drawers or turn on the water in the sink. If you find, for example, the drawer in which you keep your pot holders can't be opened at the same time as the oven, you can request a change in design before the house is constructed. Such conditions may be very hard for an untrained lepers to discover from a set of blueprints but can be easily found in the computer "walk through."

In another system a player may sit in a small booth which includes visual and audio displays and "fight" a tank battle with another player in a similar booth.

A pilot learns to fly a new type of aircraft using a flight simulator. Advanced simulators include not only visual displays showing realistic views, but also audio and motion devices to provide a realistic "flight" experience. In one such simulator - the Air Force's Super Cockpit - an experienced pilot made an error causing the simulator to "crash." The effect was so realistic that he passed out.

These systems exist. The ones being developed go far beyond this and, for example, allow an operator sitting in a room in Texas to operate a mining vehicle on the moon. Or a doctor in Boston can observe and direct a delicate operation taking place in Kansas. All of these systems involve extensive computer use and high tech display and control devices. They are what we usually think of when we speak of "virtual reality."

- c. **Consider two raster systems with the resolutions of 640 x 480 and 1280 x 1024. How many pixels could be accessed per second in each of these systems by a display controller that refreshes the screen at a rate of 60 frames per second?**

(6)

Answer:

Since 60 frames are refreshed per second and each frame consists of 640 x 480 pixels, the access rate of such a system is:

$$(640 \times 480) \times 60 = 1.8432 \times 10^7 \text{ pixels/second.}$$

Likewise, for the 1280 x 1024 system, the access rate is:

$$(1280 \times 1024) \times 60 = 7.86432 \times 10^7 \text{ pixels/second.}$$

According to the definition of access rate, we know that the access time per pixel should be $1/(\text{access rate})$. Therefore, the access time is around 54 nanoseconds/pixel for the 640 x 480 system, and the access time is around 12.7 nanoseconds/pixel for the 1280 x 1024 system.

- Q.3 a. Digitize a line from (10, 12) to (15, 15) on a raster screen using Bresenham's straight line algorithm.**

(8)

Answer:

To illustrate the algorithm, we digitize the line with endpoints (20, 10) and (30, 18)

This line has a **slope** of **0.8**, with

$$\mathbf{dx = 10, \quad dy = 8}$$

The **initial decision parameter** has the value:

$$p = 20dy - dx = 6$$

and the **increments** for calculating successive decision parameters are

$$20dy = 16, \quad 2dy - 2dx = -4$$

- We plot the **initial point** = **(20, 10)**, and determine successive pixel positions along the line path from the decision parameter as:

k

0	6	(21, 11)
1	2	(22, 12)
2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

- b. Explain a two-dimensional world-coordinate reference frame in OpenGL. How will you set the window color using OpenGL? (8)**

Answer:

OPENGL SYNTAX AND PROGRAM STRUCTURE

BASIC OPENGL FUNCTIONS, CONSTANTS, AND DATA TYPES

Function names in the basic OpenGL library begin with the letters gl, and each component word within a name has its first letter capitalized. The following examples illustrate this naming convention: glClear (), glCopyPixels () and glPolygonMode ().

Certain functions require arguments that are constant values specifying a parameter name, a value for a parameter, a particular mode, etc. All such constants begin with the letters GL, component words in the constant are written in capital letters, and the underscore (_) is used as a separator between the words. A few examples of the several hundred constants available in OpenGL are GL_CCW, GL_RGB, and GL_AMBIENT_AND_DIFFUSE.

A variety of data types for ANSI C implementations can be used in OpenGL functions. Some examples are GLfloat, GLubyte, GLbitfield, and GLvoid.

Some arguments of OpenGL functions can be assigned values using a vector that lists a set of data values. This is an option for specifying a list of values as a pointer to an array, rather than specifying each element of the list explicitly as a parameter argument. A typical example of the use of this option is in specifying xyz coordinate values.

RELATED LIBRARIES AND ROUTINES

In addition to the basic, or core, library of functions, a set of "macro" routines that use core functions are available in the OpenGL Utility Library (GLU). These routines provide methods for setting up viewing projection matrices, describing complex objects with line and polygon approximations, surface rendering, and other complex tasks. In particular, GLU provides methods for displaying quadrics using linear-equation approximations.

There are a number of window-system libraries that support OpenGL functions. The OpenGL Extension to the X Window System (GLX) provides a set of routines that are prefixed with the letters glX. The WGL routines provide a Microsoft Windows-to-OpenGL interface. These routines are prefixed with the letters wgl. The PGL routines, which are prefixed with pgl, provide a Presentation Manager to OpenGL interface for the IBM OS/2. And the GLUT (OpenGL Utility Toolkit) routines, prefixed with glut, provide a toolkit that is independent of any window system. This set of routines also contains methods for describing and rendering quadric curves and surfaces. An object-oriented toolkit based on OpenGL, called Open Inventor, provides routines and predefined object shapes for interactive three-dimensional applications. This toolkit is written in C++.

HEADER FILES

#include <GL/gl.h>	Includes the OpenGL core header file. This file is required by all OpenGL applications.
#include <GL/glu.h>	Includes the OpenGL Utility Library header file. This file is needed by most OpenGL applications.

And we would also need a window interface library. Alternatively, we can just include the GLUT header file:

#include <GL/glut.h>	Includes the OpenGL Utility Toolkit header file. This statement automatically includes gl.h, glu.h, and glx.h. And with Microsoft Windows, it includes the appropriate header file to access WGL.
----------------------	---

SETTING UP DISPLAY WINDOWS USING GLUT

glutInit (options);	Initializes GLUT and specifies command-line options for the window system in use. This function should be called before any other GLUT routine.
glutInitWindowPosition (x, y);	Specifies the screen position for one corner of the window.
glutInitWindowSize (w, h);	Specifies the width (w) and height (h) for the window in pixels (integer values).
glutCreateWindow (string);	Opens a window with the previously specified size, position, and other properties. The specified text string may be displayed in the window title bar, depending on the options available in the window system.

Various options for the window are set with the `glutInitDisplayMode` function. These options include the color mode and single or double buffering. The default is RGB color mode with single buffering.

SETTING THE WINDOW COLOR

Background color for a window is set with the functions `glClearColor` and `glClear`. For example, the following two statements set the background color to black (the default), then this color is applied to the color buffer which displays the window with the specified color.

```
glClearColor (0.0, 0.0, 0.0, 0.0);  
glClear (GL_COLOR_BUFFER_BIT);
```

In the `glClearColor` function, the first three parameters set the RGB color components in the range from 0 to 1. The fourth parameter is used to set an "alpha value", as discussed in the section Attributes of Output Primitives, Color.

WINDOW COORDINATES AND SCREEN DISPLAY AREA

There are a number of options for setting viewing parameters and generating a screen display of a scene. The various options for generating views of two-dimensional and three-dimensional scenes are discussed in the section on Viewing Transformations.

For a two-dimensional application, we can obtain a screen display of a defined scene with the following code segment which sets up a coordinate "viewing" range and specifies an area of the screen for displaying the scene.

```
glViewport (xvmin, yvmin, vpWidth, vpHeight);  
glMatrixMode (GL_PROJECTION);  
glLoadIdentity ();  
gluOrtho2D (xwmin, xwmax, ywmin, ywmax);
```

The first command above defines the rectangular screen area (called the "viewport") that is to be used for the display of the scene. All parameters in the `glViewport` function are assigned integer screen coordinate values, and the lower left corner of the viewport is displayed at screen position(`xvmin`, `yvmin`) in the display "window" set up by the window manager. The second command sets the mode for matrix operations. In this case, we want to project a scene description to the screen. The third command initializes the transformation matrix for these operations to the identity matrix. And the fourth and last command defines the range of coordinates that will map to the viewport, with coordinate values expressed as floating-point numbers. This fourth function defines a "clipping window", so that objects inside this rectangle are displayed within the viewport. Objects outside this rectangle are clipped; i.e., not displayed. As an alternative, the function `gluOrtho2D` could be replaced with `glOrtho`, as discussed in the section on Viewing Transformations.

DISPLAYING A SCENE USING GLUT

After initializing GLUT, setting the window parameters (position, size, and background color), and specifying the coordinate system, we can display a scene with the following functions

```
glutDisplayFunc(sceneDescrip);  
glutMainLoop ();
```

The parameter sceneDescrip in the first function is a routine that describes the primitives and attribute settings for the scene. Rendering and display of window contents are effective with the final GLUT function: glutMainLoop.

Q.4 a. Explain Cyrus-Beck clipping algorithm for a convex polygon with an example. (8)

Answer:

The **Cyrus-Beck algorithm** is a generalized **line clipping** algorithm. It was designed to be more efficient than the **Sutherland-Cohen algorithm** which uses repetitive clipping.^[1] Cyrus-Beck is a general algorithm and can be used with a convex polygon clipping window unlike Sutherland-Cohen that can be used only on a rectangular clipping area.

Here the parametric equation of a line in the view plane is:

$$\begin{aligned} p(t) &= tp_1 + (1-t)p_0 \\ &= p_0 + t(p_1 - p_0) \end{aligned}$$

where $0 \leq t \leq 1$.

Now to find intersection point with the clipping window we calculate value of dot product. Let p_E be a point on the clipping plane E .

Calculate $n \cdot (p(t) - p_E)$.

if > 0 vector pointed towards interior

if $= 0$ vector pointed parallel to plane containing p

if < 0 vector pointed away from interior

Here n stands for normal of the current clipping plane (pointed away from interior).

By this we select the point of intersection of line and clipping window where (dot product = 0) and hence clip the line.

2] Cyrus-Beck is a general algorithm and can be used with a convex polygon clipping window.

```
p(t) = p0 + t(p1-p0)          /* it's parametric function */
```

3] if > 0 ; vector says $p(t)$ is OUTSIDE && $A < 90$ degree.

```
if < 0 ; vector says p(t) is INSIDE && a > 90 degree.
if = 0 ; vector says p(t) is on edge E .. here outer normal edge is
perpendicular to the E and p(t)-B
```

```

.. we will writing here a function code for it as given below :
/*

if( DtProd (N,P(t)-B) > 0)
    {
        p(t) OUTER & A < 90 degree ;    /* P(t) is OUTSIDE ..

    }
else if( DtProd (N,P(t)-B) < 0)
    {
        p(t) INNER & A > 90 degree ;    /* P(t) is INSIDE ..

    }
else( DtProd (N,P(t)-B) = 0)
    {
        p(t) lies on to the edge E ;    /* where outer normal edge N
would be perpendicular to both E and p(t)-B..

    }

*/

```

b. Explain various OpenGL point-attribute and OpenGL line attribute functions with examples. (8)

Answer:

Function names in the basic OpenGL library begin with the letters gl, and each component word within a name has its first letter capitalized. The following examples illustrate this naming convention: glClear (), glCopyPixels () and glPolygonMode ().

Certain functions require arguments that are constant values specifying a parameter name, a value for a parameter, a particular mode, etc. All such constants begin with the letters GL, component words in the constant are written in capital letters, and the underscore (_) is used as a separator between the words. A few examples of the several hundred constants available in OpenGL are GL_CCW, GL_RGB, and GL_AMBIENT_AND_DIFFUSE.

A variety of data types for ANSI C implementations can be used in OpenGL functions. Some examples are GLfloat, GLubyte, GLbitfield, and GLvoid.

Some arguments of OpenGL functions can be assigned values using a vector that lists a set of data values. This is an option for specifying a list of values as a pointer to an array, rather than

specifying each element of the list explicitly as a parameter argument. A typical example of the use of this option is in specifying xyz coordinate values.

- Q.5 a. Calculate the new coordinates of a block rotated about x axis by an angle of = 30 degrees. The original coordinates of the block are given relative to the global xyz axis system (8)**
 A(1, 1, 2), B(2, 1, 2), C(2, 2, 2), D(1, 2, 2), E(1, 1, 1), F(2, 1, 1), G(2, 2, 1), H(1, 2, 1).

Answer:

- b. Derive the transformation matrix for translation and scaling. (8)**

Answer:

A transformation matrix is simply a short-hand for applying a function to the x and y values of a point, independently. In the case of translation, $x' = 1*x + 0*y + dx*1$ and $y' = 0*x + 1*y + dy * 1$. The matrix representation of these two equations is as follows:

$$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

The other matrices can be similarly derived--simply determine what x' and y' should be, in terms of x, y and 1.

- Q.6 a. Compare the Cohen-Sutherland and Liang-Barsky line clipping algorithms on the basis of their merits and demerits. Explain. (8)**

Answer:

In computer graphics, the Liang–Barsky algorithm (named after You-Dong Liang and Brian A. Barsky) is a line clipping algorithm. The Liang–Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clipping window. With these intersections it knows which portion of the line should be drawn. This algorithm is significantly more efficient than Cohen–Sutherland.

The idea of the Liang-Barsky clipping algorithm is to do as much testing as possible before computing line intersections. Consider first the usual parametric form of a straight line:

$$\begin{aligned} x &= x_0 + u(x_1 - x_0) = x_0 + u \Delta x \\ y &= y_0 + u(y_1 - y_0) = y_0 + u \Delta y \end{aligned}$$

The Liang-Barsky algorithm is more efficient than the Cohen-Sutherland algorithm, since intersection calculations are reduced. Each update of parameters u1 and u2 requires only one division and window intersections of the line are computed only once, when the final values of u1 and u2 have been computed.

- b. Explain: (8)**
(i) Perspective projection of a point

- (ii) Perspective projection of a line**
Answer: **(i) Refer page 361 of Text Book-I**
(ii) Refer page 365 of Text Book-I

Q.7 a. How do you create shading and draw shadows? Explain. (8)

Answer:

Light and shadows visually define objects. Before you can draw the light and shadows you see, you need to train your eyes to see like an artist.

Values are the different shades of gray between white and black. Artists use values to translate the light and shadows they see into *shading*, thus creating the illusion of a third dimension.

Hatching and *crosshatching* are simple and fun techniques for drawing shading.

A full range of values is the basic ingredient for shading. When you can draw lots of different values, you can begin to add shading, and therefore depth, to your drawings.

With shading, the magical illusion of three-dimensional reality appears on your drawing paper. Figure 1 demonstrates how to take a simple line drawing of a circle and add shading to transform it into the planet Earth.

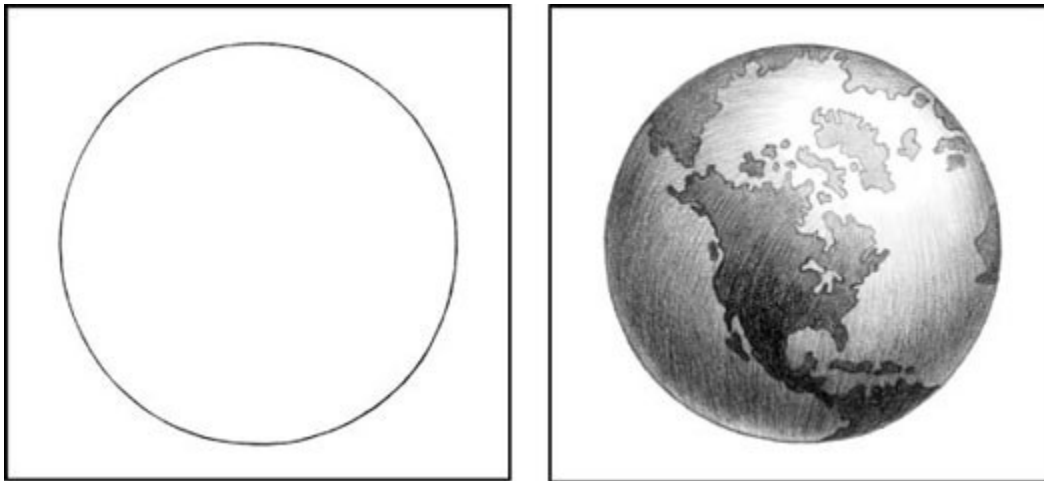


Figure 1: Turning a simple line drawing into planet Earth.

You know the objects around you are three-dimensional because you can walk up to them, see them from all sides, and touch them. Take a moment to look around you at familiar objects. Try to discover why you see their actual three-dimensional forms. Look for the different values created by the light and shadows.

Taking a closer look at light and shadow

Before you can draw the appropriate values that illustrate light and shadows correctly, you need to be able to visually identify the following:

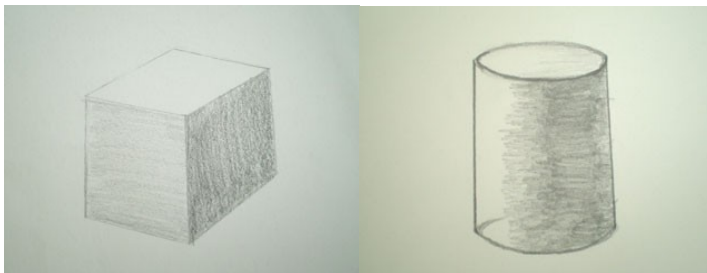
- **Light source:** The direction from which a dominant light originates. The placement of this light source affects every aspect of a drawing.
- **Shadows:** The areas on an object that receive little or no light.
- **Cast shadow:** The dark area on an adjacent surface where the light is blocked by the solid object.

The light source tells you where to draw all the light values and shadows.

Figure 2 gives you some practice in locating the light source, shadows, and cast shadows around an object, which in this case is a sculpture. As you look at two drawings of the sculpture, ask yourself the following questions:

- **Where are the light values?** Look for the lightest areas on the object. The very brightest of the lightest values are called *highlights*.
- **Where are the dark values?** Dark values often reveal the sections of the object that are in shadow. By locating shadows, you can usually identify the light source.
- **Where is the cast shadow?** The section of the cast shadow closest to the object is usually the darkest value in a drawing. By locating an object's cast shadow, you can easily discover the direction from which the light source originates.

shadows are the darkest areas of the surface. They are the product of light striking certain areas of an object while not being able to reach others and shadows are cast as a result. Shadows can have really sharp edges between it and the mid tone or it can gradually blend into the mid tone. The way in which light, falling on an object, separates into distinct areas of light and dark depends upon the surface of the object. If the object has hard edges, then the point at which light becomes dark will be sharp. If the surface is curved, there will be a gradual change from light to dark.



Cast Shadows

In addition to the shadows on the object itself, there's also the cast shadow. This is the shadow created by the object itself upon the surface on which it stands. In bright light, the cast shadow is darker than the object's darkest side, becoming progressively lighter toward its perimeter. A low angle of light will cast a long shadow, whereas an overhead light will cast a short one. The edges of cast shadows can be sharp when it is an angular object or soft when the object is rounded. Including shadows in your drawing will not only add to the realism of the drawing but help to create an

illusion of weight since the object will appear to be resting firmly on a surface. Shadows also add to the atmosphere of a drawing.

Previously, we learned how to draw objects in three-dimensional form using predominantly linear methods. This section will address the role of light, shadow and shading techniques in rendering realistic, three dimensional form. Form is the illusion of a three-dimensional structure created using shading and/or perspective.

Light and shadow create depth and atmosphere in a drawing. Without them even a well constructed drawing with perspective and proportion is going to appear somewhat flat. If you look closely, you will notice that everything in the real world has a shadow, therefore no drawing is truly realistic without them. By incorporating light and shadow from this point forward, your drawings will really start to jump off the page.

Light Source

Generally speaking, at least one light source influences the look of any drawing. This could be sunlight or artificial light. At other times, both could be present. For the time being, we'll work with a single light source to avoid confusion. Identifying the direction from which the light is coming is key when it comes to shading. The location of the light source relative to the subject will impact where the light hits the subject and at what intensity. The direction of the light source must be factored into all shading so that tonal consistency is maintained throughout the drawing. .

Tone

Tone represents the lightness or darkness of an area in terms of black to white. Tone can be thought of as a range of values; highlights being the whitest whites, mid tones the medium values of grey and shadows encompassing the darkest tones.



Highlights are the areas of the subject closest to the light source. Consequently, they are the brightest and thus whitest areas of the subject, with progressively darker shading from this point outwards. Used in conjunction with shadows, highlights serve to enhance the illusion of three-dimensional form. The number of highlights on a subject is limited only by the number of reflective surfaces and the ability of the surfaces to reflect light. If the subject is smooth and shiny, highlights will be very bright and intense. If the subject has a rougher consistency, the highlight will be dulled and blend more smoothly into the mid tones.

Mid tones fall midway between highlight and shadow and generally make up the majority tone. They represent the true color and consistency of the subject whereas highlights are brighter than the "true" color and shadows are darker.

- b. Discuss the three parameters that OpenGL allows to be set to specify general rules for applying the lighting model. (8)**

Answer: Refer page 416 of Text Book-I

- Q.8 a. Explain the steps involved in scan-line polygon fill algorithm. Illustrate with an example. (8)**

Answer:

For each scan-line, start by selecting the next pixel to be filled by clicking on it with the mouse. If you select the incorrect pixel, a message will appear at the bottom of the window prompting you to select another.

Once all of the pixels for the scan-line have been filled in, increment the scan-line by clicking on the Scan-line button.

Next, click on the Remove Edges From AET button to check for and remove any edges from the active edge table for which the scan-line is equal to the maximum y value.

Select the Update X Values button to update the x values in the active edge table for the current scan-line.

If there are any edges left in the global edge table, select the Add Edges To AET button to check for any edges in the global edge table for which the minimum y value is equal to the current scan-line and add them to the active edge table.

Select the Reorder AET button to reorder the edges in the active edge table.

If the active edge table still contains edges, start again at step 1 for the current scan-line.

1. Initializing All of the Edges:

We want to determine the minimum y value, maximum y value, x value, and $1/m$ for each edge and keep them in the all_edges table. We determine these values for the first edge as follows:

Y-min:

Since the first edge consists of the first and second vertex in the array, we use the y values of those vertices to choose the lesser y value. In this case it is 10.

Y-max:

In the first edge, the greatest y value is 16.

X-val:

Since the x value associated with the vertex with the highest y value is 10, 10 is the x value for this edge.

1/m:

Using the given formula, we get $(10-10)/(16-10)$ for 1/m.

The edge value results are in the form of Y-min, Y-max, X-val, Slope for each edge array pointed to in the all_edges table. As a result of calculating all edge values, we get the following in the all_edges table.

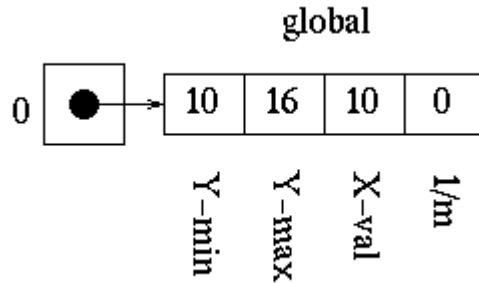
all_edges

0	●	→	10	16	10	0
1	●	→	16	20	10	1.5
2	●	→	10	20	28	-1.2
3	●	→	10	16	28	0
4	●	→	10	16	22	1
5	●	→	10	10	10	inf
			Y-min	Y-max	X-val	1/m

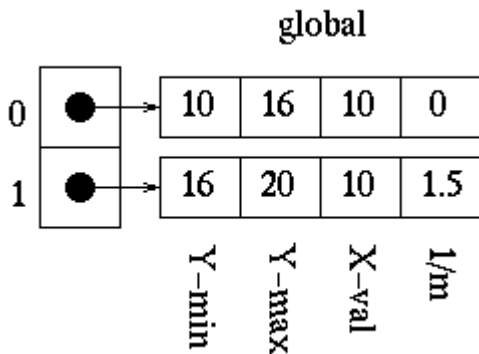
2. Initializing the Global Edge Table:

We want to place all the edges in the global edge table in increasing y and x values, as long as slope is not equal to zero.

For the first edge, the slope is not zero so it is placed in the global edge table at index=0.



For the second edge, the slope is not zero and the minimum y value is greater than that at zero, so it is placed in the global edge table at index=1.



For the third edge, the slope is not zero and the minimum y value is equal the edge's at index zero and the x value is greater than that at index 0, so the index is increased to 1. Since the third edge has a lesser minimum y value than the edge at index 2 of the global edge table, the index for the third edge is not increased again. The third edge is placed in the global edge table at index=1.

global

0	●	→	10	16	10	0
1	●	→	10	20	28	-1.2
2	●	→	16	20	10	1.5
			Y-min	Y-max	X-val	1/m

We continue this process until we have the following:

global

0	●	→	10	16	10	0
1	●	→	10	16	22	1
2	●	→	10	16	28	0
3	●	→	10	20	28	-1.2
4	●	→	16	20	10	1.5
			Y-min	Y-max	X-val	1/m

Notice that the global edge table has only five edges and the all_edges table has six. This is due to the fact that the last edge has a slope of zero and, therefore, is not placed in the global edge table.

3. Initializing Parity

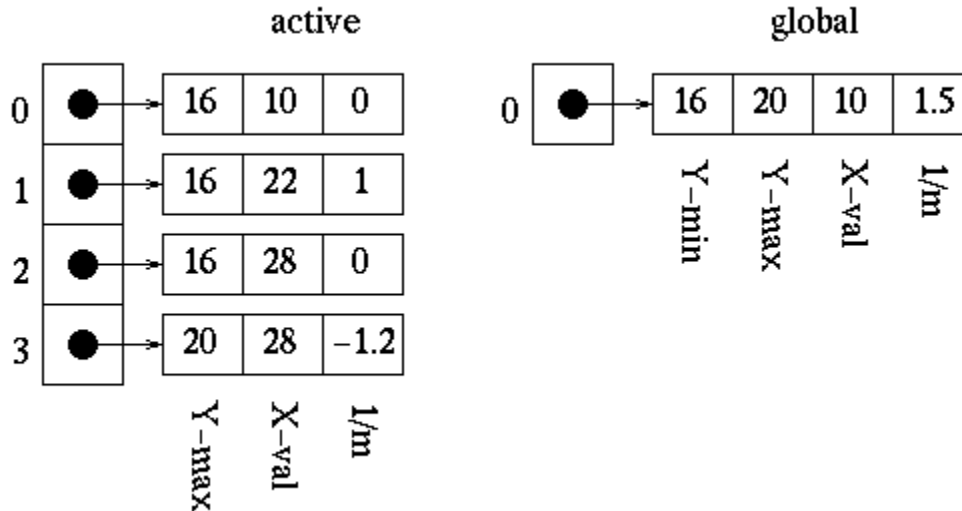
Parity is initially set to even.

4. Initializing the Scan-Line

Since the lowest y value in the global edge table is 10, we can safely choose 10 as our initial scan-line.

5. Initializing the Active Edge Table

Since our scan-line value is 10, we choose all edges which have a minimum y value of 10 to move to our active edge table. This results in the following.



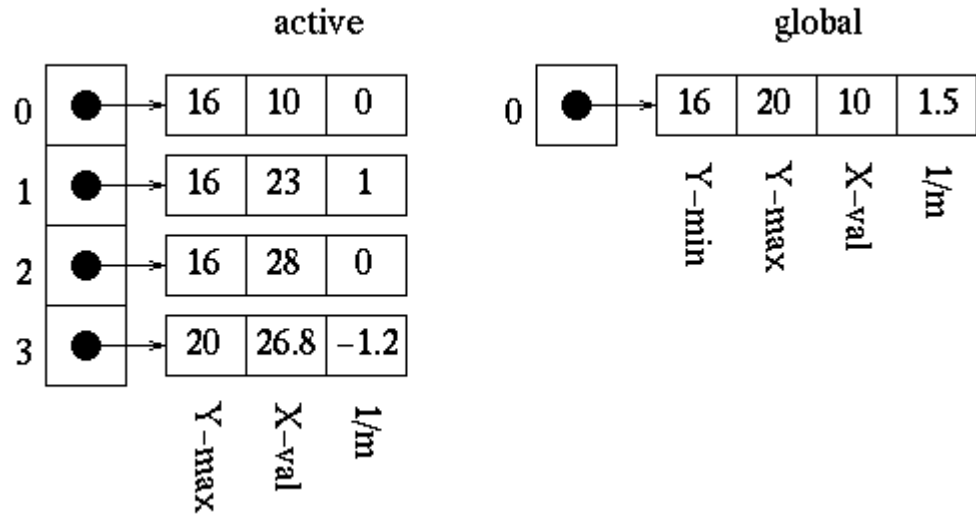
6. Filling the Polygon

Starting at the point (0,10), which is on our scan-line and outside of the polygon, we will want to decide which points to draw for each scan-line.

1. Scan-line = 10:

Once the first edge is encountered at $x=10$, parity = odd. All points are drawn from this point until the next edge is encountered at $x=22$. Parity is then changed to even. The next edge is reached at $x=28$, and the point is drawn once on this scan-line due to the special parity case. We are now done with this scan-line.

First, we update the x values in the active edge table using the formula $x_1 = x_0 + 1/m$ to get the following:



- b. Discuss the different approaches to antialiasing. (8)

Answer: Refer pages 509-511 of Text Book-I

- Q.9 a. Discuss the different ways of describing curves by means of Polynomials. (8)

Answer: Refer pages 528-530 of Text Book-I

- b. What are the properties of Bezier curves? Discuss them. (8)

Answer: Refer page 539 of Text Book-I

TEXT BOOK

Computer Graphics Using OpenGL, F.S. Hill, Jr., Second edition, PHI/Pearson Education, 2005