Q.1 a. Computer graphics has enhanced the quality of work in many areas. Support this statement through a brief discussion on areas of application of computer graphics. Specify at least one specific application.

Answer:

- Displayof Information
- Geographicinformation system (GIS)
- ComputerizedTomography (CT)
- Magneticresonance imaging (MRI)
- Ultrasound
- Design
- Architecture
- Designof Mechanical part
- VLSI
- Simulation
- Graphical flight simulator
- reduce training process
- Robotic simulation
- TV, Movie, advertising industries
- generate photo realistic images
- Virtual Reality (VR)
- reduce risk of training
- surgery
- astronaut
- UserInterface
- Window system, Window2003
- Xwindow,MACOS
- Graphical Network browsers
- Mathematicalmodeling
- Entertainment etc.

b. While drawing a circle, co-ordinates of only one eighth of the total pixels lying on circumference of a circle are computed. Why?

Answer:

Circle is symmetric about its axis. So, if point (x,y) is on the circle, then we can trivially compute (45^0segment) seven other points (y,x),(y,-x),(x,-y),(-x,-y),(-y,-x),(-y,x),(-x, y) using the symmetry.

c. As a part of graphics design application, you are supposed to eliminate the hidden surfaces in your design. You have an option of choosing either the Z-Buffer technique or the Painters algorithm. Which algorithm will you choose for depth calculation at each pixel on a scan line can be done incrementally if the plane equation for each polygon is available?

Answer:

WKT the polygon is planar. We can simplify the calculation of z values for each point on a scan line by using Depth Coherence concept.

At (x, y) compute z by $Z=(-D-Ax-By)/C$.

At (x+dx, y) value of z= z_1-{A/C}(dx)

Only one subtraction is need to compute (z(x+1), y) given z(x, y) as dx=1.

d. In depth sorting method, indicate the tests that are to be carried out to determine if two surfaces R and S need not be ordered.

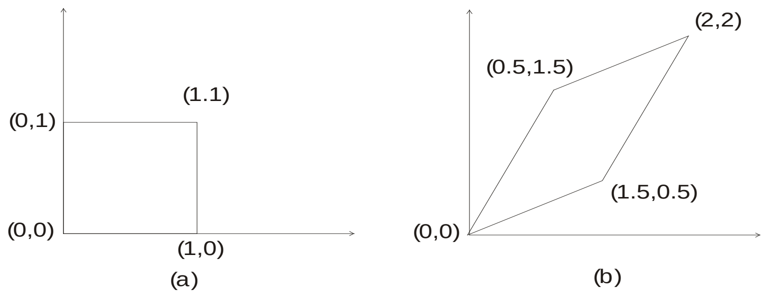
Answer:

All 5 tests =4 marks

Tests that are to be carried out are

- Do the polygon's x extent overlap
- Do the polygon's y extent overlap
- Is R entirely on the opposite side of S's plane from the viewpoint
- Is S entirely on the opposite side of R's plane from the viewpoint
- Do the projections of the polygons onto the (x,y) plane not overlap

e. A square shown in (a) is converted to a parallelogram shown in (b) using composite transformation matrix M. Determine M.



-

Answer:

Based on the data in the graph above:

$$M\begin{bmatrix}0\\1\\1\end{bmatrix}=\begin{bmatrix}0.5\\1.5\\1\end{bmatrix}\ M\begin{bmatrix}1\\0\\1\end{bmatrix}=\begin{bmatrix}1.5\\0.5\\1\end{bmatrix}\ M\begin{bmatrix}1\\1\\1\end{bmatrix}=\begin{bmatrix}2\\2\\1\end{bmatrix}$$

Then,

$$M\begin{bmatrix}0&1&1\\1&0&1\\1&1&1\end{bmatrix}=\begin{bmatrix}0.5&1.5&2\\1.5&0.5&2\\1&1&1\end{bmatrix}$$

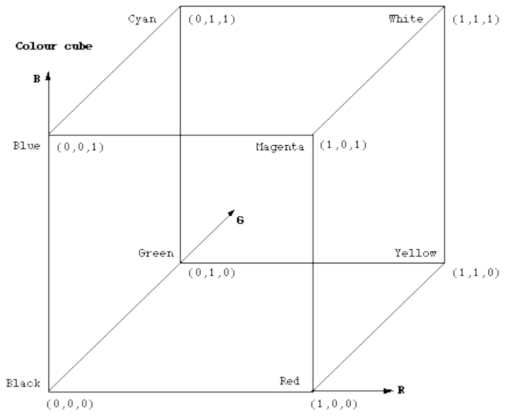
Therefore,

$$M = \begin{bmatrix} 1.5 & 0.5 & 0 \\ 0.5 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

f. How will you generate RGB Colour Cube

Answer:

The colour space for computer based applications is often visualized by a unit cube. Each colour (red, green, blue) is assigned to one of the three orthogonal coordinate axes in 3D space. An example of such a cube is shown below along with some key colors and their coordinates.



- Along each axis of the colour cube the colours range from no contribution of that component to a fully saturated colour.
- The colour cube is solid, any point (colour) within the cube is specified by three numbers, namely, an r,g,b triple.
- The diagonal line of the cube from black (0,0,0) to white (1,1,1) represents all the greys, that is, all the red, green, and blue components are the same.
- In practice different computer hardware/software combinations will use different ranges for the colours, common ones are 0-256 and 0-65536 for each component. This is simply a linear scaling of the unit colour cube described here.
- This RGB colour space lies within our perceptual space, that is, the RGB cube is smaller and represents fewer colours than we can see.

g. Briefly describe following OpenGL functions and their application:
(i) glutInit() (ii) GL_LINE_STRIP()
(iii) gluInitDisplayMode() (iv) glOrtho() (7×4)

Answer:

(i) glutInit():
glutInit will initialize the GLUT library and negotiate a session with the window system.

(ii) GL_LINE_STRIP()

Draws a connected group of line segments from the first vertex to the last.

(iii)gluInitDisplayMode()
sets the initial display mode.

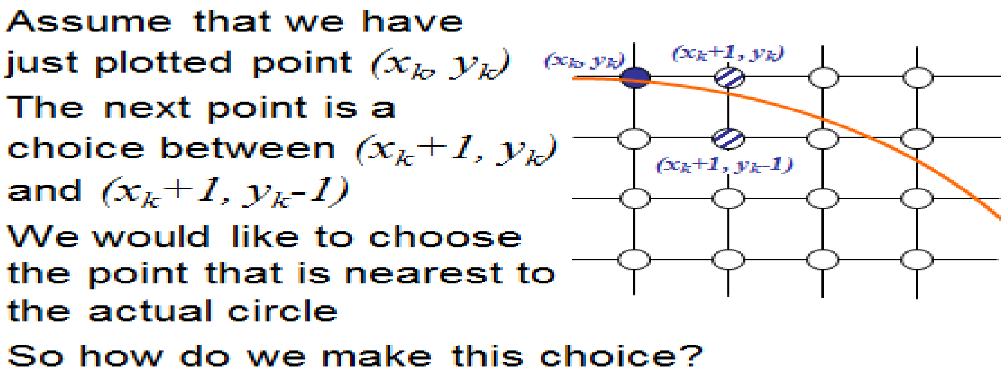
(iv)glMatrixMode()
specifies which matrix is the current matrix

Q.2 a. Explain the midpoint circle drawing algorithm. Illustrate the algorithm, assuming radius equal to 10 cm and the Centre of the circle at origin. (10)

Answer:

Midpoint circle Algorithm

1. Input radius r and circle center (xc,yc) and obtain the first point on the circumference of the circle centered on the origin as (x0,y0) = (0,r)
2. Calculate the initial value of the decision parameter as P0=(5/4)-r
3. At each xk position, starting at k=0, perform the following test. If Pk<0 the next point along the circle centered on (0,0) is (xk+1,yk) and Pk+1=Pk+2xk+1+1 Otherwise the next point along the circle is (xk+1,yk-1) and Pk+1=Pk+2xk+1+1-2 yk+1 Where 2xk+1=2xk+2 and 2yk+1=2yk-2
4. Determine symmetry points in the other seven octants.
5. Move each calculated pixel position (x,y) onto the circular path centered at (xc,yc) and plot the coordinate values. x=x+xc y=y+yc
6. Repeat step 3 through 5 until x>=y


Assume that we have just plotted point (x_k, y_k)
The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1)
We would like to choose the point that is nearest to the actual circle
So how do we make this choice?

Example:

Midpoint Circle Drawing

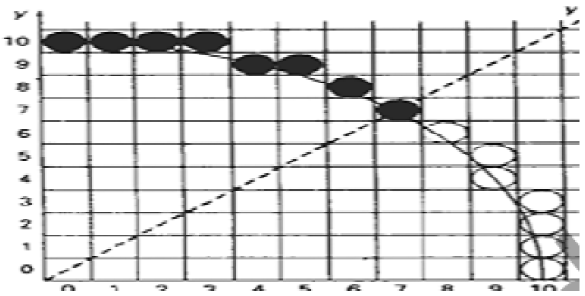
Given a circle radius r=10

The circle octant in the first quadrant from x=0 to x=y.

The initial value of the decision parameter is

P0=1-r = -9 or the circle centered on the coordinate origin, the initial point is (x0,y0)=(0,10) and initial increment terms for calculating the decision parameters are 2x0=0 , 2y0=20

Successive midpoint decision parameter values and the corresponding coordinate positions along the circle path are listed in the following table.



k	pk	(xk+1, yk-1)	2xk+1	2yk+1
0	-9	(1,10)	2	20
1	-6	(2,10)	4	20
2	-1	(3,10)	6	20
3	6	(4,9)	8	18
4	-3	(5,9)	10	18
5	8	(6,8)	12	16
6	5	(7,7)	14	14

b. Consider a line from (0, 0) to (4, 6). Rasterize this line using simple DDA algorithm. (8)

Answer:

Evaluating steps 1 to 5 in the DDA algorithm we have

$X_1 = 0$ $Y_1 = 0$

$X_2 = 4$ $Y_2 = 6$

Length = $|Y_2-Y_1| = 6$

$\Delta X = |X_2- X_1| / \text{Length}$

$= \frac{4}{6}$

$\Delta Y = |Y_2- Y_1| / \text{Length}$

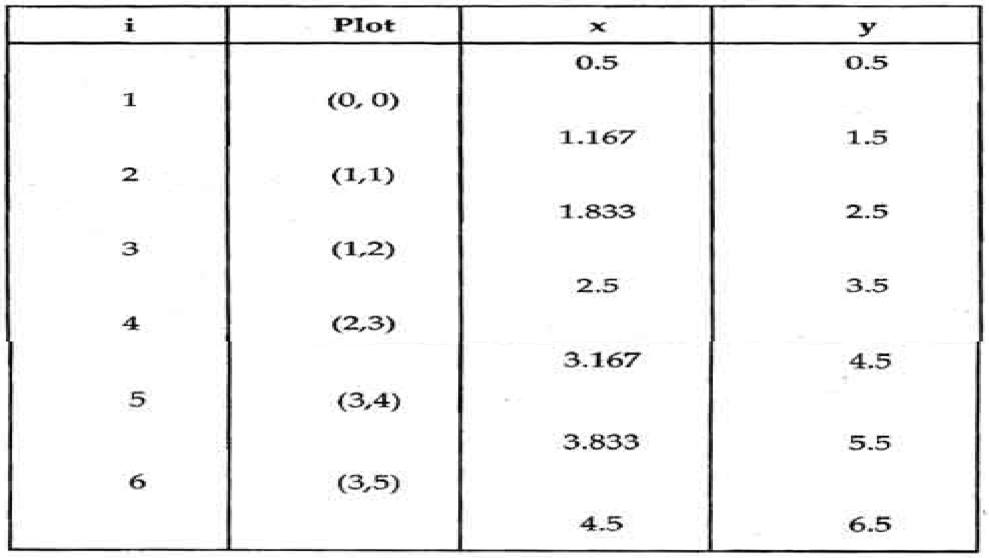
$= 6/6=1$

Initial value for

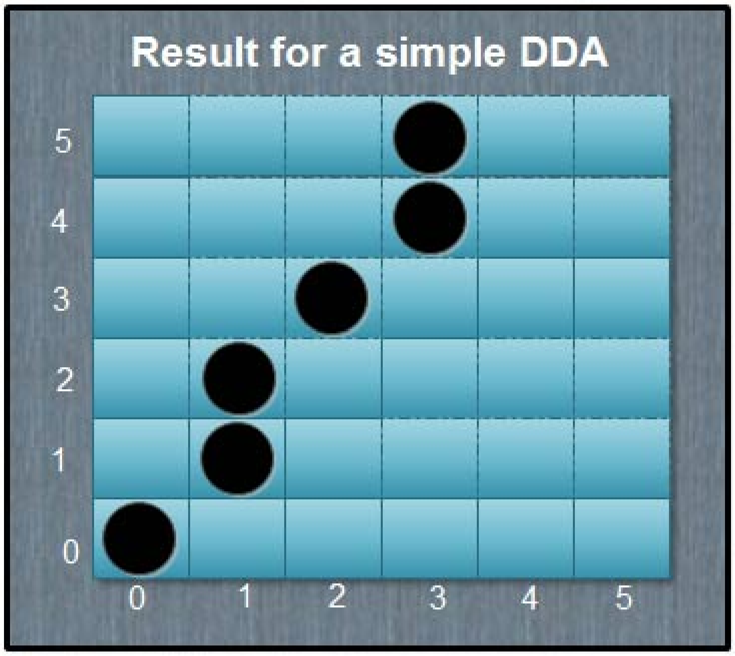
$$X = 0 + 0.5 * \text{Sign}\left(\frac{4}{6}\right) = 0.5$$

$$Y = 0 + 0.5 * \text{Sign}(1) = 0.5$$

Tabulating the results of each iteration in the step 6 we get,



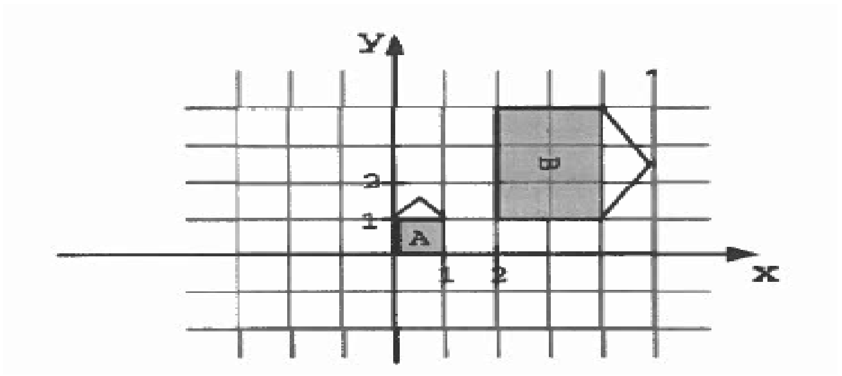
i	Plot	x	y
		0.5	0.5
1	(0, 0)		
		1.167	1.5
2	(1,1)		
		1.833	2.5
3	(1,2)		
		2.5	3.5
4	(2,3)		
		3.167	4.5
5	(3,4)		
		3.833	5.5
6	(3,5)		
		4.5	6.5



The results are plotted as shown in the Fig. It shows that the rasterized line lies to both sides of the actual line, i.e. the algorithm is orientation dependent.

Q.3. a. Find the homogeneous matrix that transforms the 2D vertices of object A to the corresponding vertices of object B. Express the matrix as a composition of elementary transformations namely translation, scaling and rotation. (6)

Answer:



Matrix is $S_{3,2} * R_{90} * T_{2,4}$

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -3 & 0 \\ 2 & 0 & 0 \\ 2 & 4 & 1 \end{bmatrix}$$

b. Translate the rectangle (2,2), (2,8), (10,8), (10,2) 2 units along x-axis and 3 units along y-axis. (6)

Answer:

Using the matrix equation for translation, we have

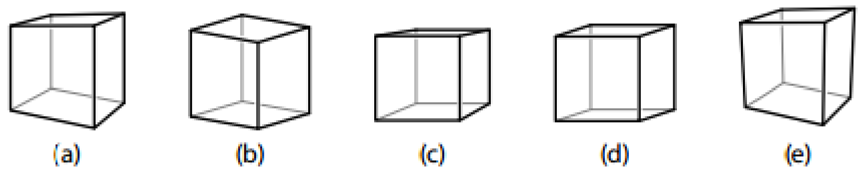
$[P^*] = [P]\,[T_t]$, substituting the numbers, we get

$$[P^*] = \begin{pmatrix} 2 & 2 & 0 & 1 \\ 2 & 8 & 0 & 1 \\ 10 & 8 & 0 & 1 \\ 10 & 2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 3 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 4 & 5 & 0 & 1 \\ 4 & 11 & 0 & 1 \\ 12 & 11 & 0 & 1 \\ 12 & 5 & 0 & 1 \end{pmatrix}$$

Note that the resultant coordinates are equal to the original x and y values plus the 2 and 3 units added to these values, respectively.

c. Consider following line drawings of different views of a cube.



Classify the views based on projections: parallel projection, one-point perspective, two-point perspective, or three-point perspective. (6)

Answer:

a. a2-point
b. parallel
c. one-point
d. parallel
e. three-point

Q.4 a. What is the Parametric Sweeping? Give the derivation of solving any point on the surface using Bezier curve. (12)

Answer:

Creating a Surface by Parametric Sweeping

In the examples given above, sweeping a curve parametrically generated the surfaces. In parametric sweeping procedure, a surface is generated through the movement of a line or a curve along or around a defined path. The curve is sweeped as the sweep parameter is varied from the values of 0 to 1, creating several instances of the curve along the sweep path. In general, the equation of the surface can be given as,Q (t, s) = P (t) T (s) Where, P (t) is the parametric equation of a curve and T(s) is the sweep transformation based on the shape of the path. The sweep transformation can consist of translation, scaling, rotation or a combined transformation. If the path isa straight line, the points along the path on the line can be represented by,

x(s) = as
y(s) = bs
z(s) = cs
and
T (s) is given as,

1 0 0 0
0 1 0 0
T(s) = 0 0 1 0
as bscs 1

Where,

a, b, c are coordinate values, and $0 \leq s \leq 1$

This is equivalent to a three-dimensional translation of a curve with several traces generated along the path, controlled by how the parameter s is varied.

Example

Consider the Bezier curve defined by the control points P1= (0,5,0), P2 = (3,4,0), P3 = (2,0,0), and P4 = (5,0,0). Translate the curve five units along the z-axis to generate a swept surface.

<u>Solution</u>

Q (t,s) = [P(t)] [Tt],substituting the numbers, we get,

-1 3 -3 1 0 5 0 1 1 0 0 0
3 -6 3 0 3 4 0 1 0 1 0 0
Q (t, s) = [t3 t2 t 1] -3 3 0 0 2 0 0 1 0 0 1 0 1 0 0 0 5 0 0 1 0 0 5s 1

Substituting the value of s and solving the matrices can calculate any point on the surface.

b. Distinguish between Bezier Surface and B-Spline Surface. (4)

Answer:

Bezier Surface

This is a synthetic surface similar to the Bezier curve and is obtained by transformation of a Bezier curve. It permits twists and kinks in the surface. The surface does not pass through all the data points.

B-Spline Surface

This is a synthetic surface and does not pass through all data points. The surface is capable of giving very smooth contours, and can be reshaped with local controls. Mathematical derivation of the B-spline surface is beyond the scope of this course. Only limited mathematical consideration will be given here.

Computer generated surfaces play a very important part in manufacturing of engineering products. A surface generated by a CAD program provides a very accurate and smooth surface, which can be generated by NC machines without any room for misinterpretation. Therefore, in manufacturing, computer generated surfaces are preferred. Since surfaces are mathematical models, we can quickly find the centroid, surface area, etc. Another advantage of CAD surfaces is that they can be easily modified**(4)**

c. Define the applications of Bilinear surfaces. (2)

Answer:

Application of Bilinear Surfaces:

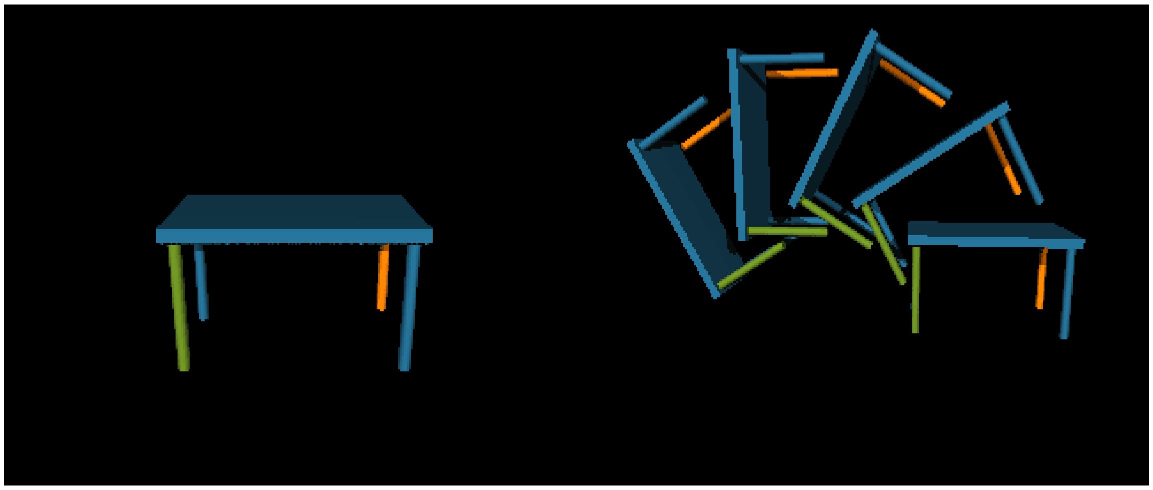
Bilinear patches are extensively used in 2-D finite element analysis (FEA). In FEA, anEngineering structure is defined by several bilinear surfaces (elements), which are created by joining points on the structure's geometry, called nodes. The nodes are connected to other nodesto create quadrilateral surfaces. Points not lying on the nodes are calculated by interpolation. Thus, the entire structure is completely defined by the nodes and the bilinear surfaces. **(2)**

Q.5 a. What are the various types of 3D transformations? Explain each briefly. (12)

Answer:

2D Transformations

Transformations are a fundamental part of computer graphics. Transformations are used to position objects, to shape objects, to change viewing positions, and even to change how something is viewed (e.g. the type of perspective that is used).



In 3D graphics, we must use 3D transformations. However, 3D transformations can be quite confusing so it helps to first start with 2D.

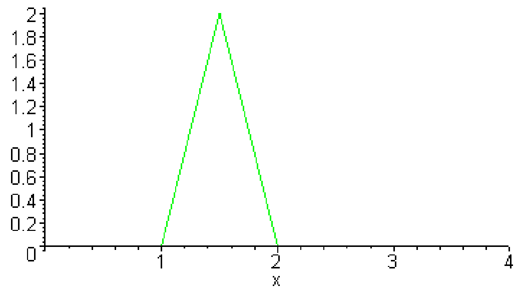
There are 4 main types of transformations that one can perform in 2 dimensions:

1. translations
2. scaling
3. rotation
4. shearing

These basic transformations can also be combined to obtain more complex transformations. In order to make the representation of these complex transformations easier to understand and more efficient, we introduce the idea of homogeneous coordinates.

Representation of Points/Objects

A point **p** in 2D is represented as a pair of numbers: **p**= (*x*, *y*) where *x* is the x-coordinate of the point **p** and *y* is the y-coordinate of **p** . 2D objects are often represented as a set of points (vertices), $\{p_1,p_2,...,p_n\}$, and an associated set of edges $\{e_1,e_2,...,e_m\}$. An edge is defined as a pair of points $e = \{p_i,p_j\}$. What are the points and edges of the triangle below?



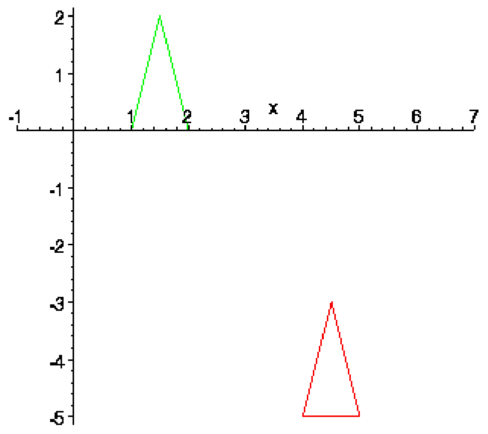
We can also write points in vector/matrix notation as

$$p = \begin{bmatrix} x \\ y \end{bmatrix}$$

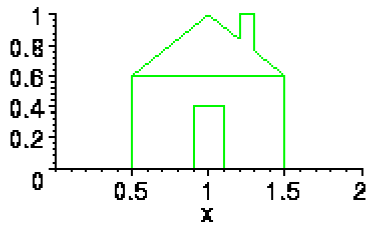
Translations

Assume you are given a point at (x,y)=(2,1). Where will the point be if you move it 3 units to the right and 1 unit up? Ans: (x',y') = (5,2). How was this obtained? - (x',y') = (x+3,y+1). That is, to move a point by some amount *dx* to the right and *dy up*, you must add *dx* to the x-coordinate and add *dy* to the y-coordinate.

What was the required transformation to move the green triangle to the red triangle? Here the green triangle is represented by 3 pointstriangle = { p1=(1,0), p2=(2,0), p3=(1.5,2) }



What are the points and edges in this picture of a house? What are the transformation is required to move this house so that the peak of the roof is at the origin? What is required to move the house as shown in animation?



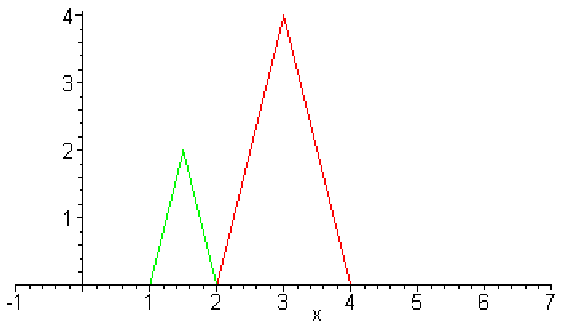
Matrix/Vector Representation of Translations

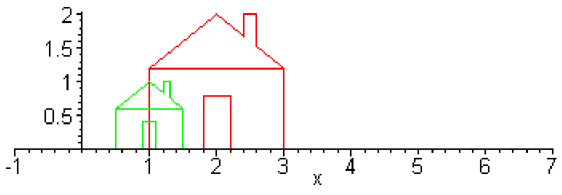
A translation can also be represented by a pair of numbers, t=(t_x,t_y) where t_x is the change in the x-coordinate and t_y is the change in y coordinate. To translate the point p by t, we simply add to obtain the new (translated) point q = p + t.

$$q = p + t = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

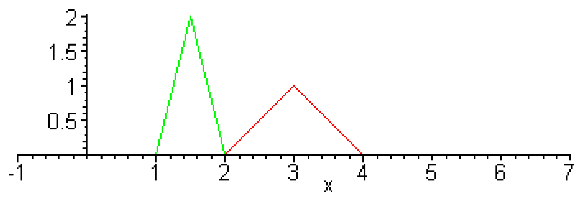
Scaling

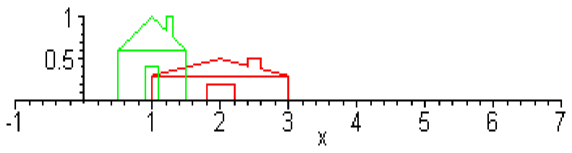
Suppose we want to double the size of a 2-D object. What do we mean by double? Double in size, width only, height only, along some line only? When we talk about scaling we usually mean some amount of scaling along each dimension. That is, we must specify how much to change the size along each dimension. Below we see a triangle and a house that have been doubled in *both* width and height (note, the area is more than doubled).



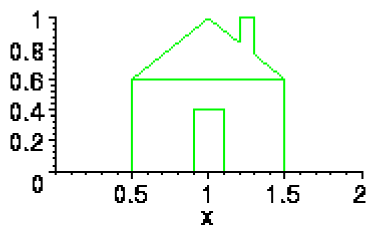


The scaling for the x dimension does not have to be the same as the y dimension. If these are different, then the object is distorted. What is the scaling in each dimension of the pictures below?





And if we double the size, where is the resulting object? In the pictures above, the scaled object is always shifted to the right. This is because it is scaled with *respect to the origin*. That is, the point at the origin is left fixed. Thus scaling by more than 1 moves the object away from the origin and scaling of less than 1 moves the object toward the origin. This can be seen in the animation below.



This is because of how basic scaling is done. The above objects have been scaled simply by multiplying each of its points by the appropriate scaling factor. For example, the point p=(1.5,2) has been scaled by 2 along x and .5 along y. Thus, the new point is
q = (2*1.5,.5*2) = (3,1).

Matrix/Vector Representation of Translations

Scaling transformations are represented by matrices. For example, the above scaling of 2 and .5 is represented as a matrix:

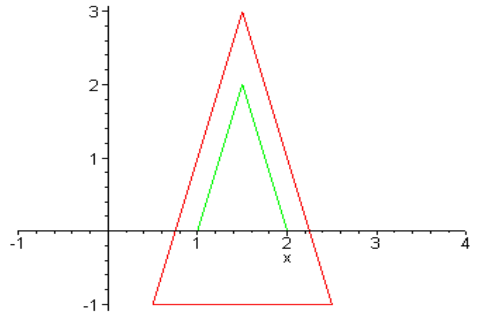
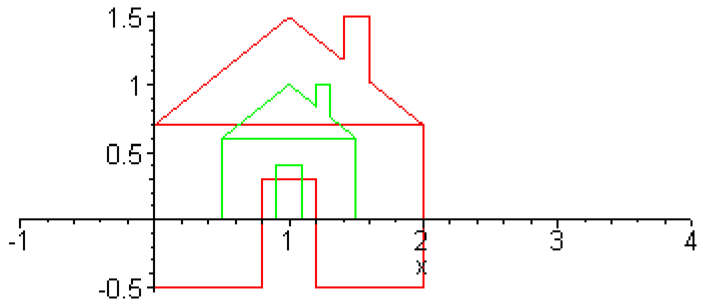
scale matrix:

$$s = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & .5 \end{bmatrix}$$

new point:

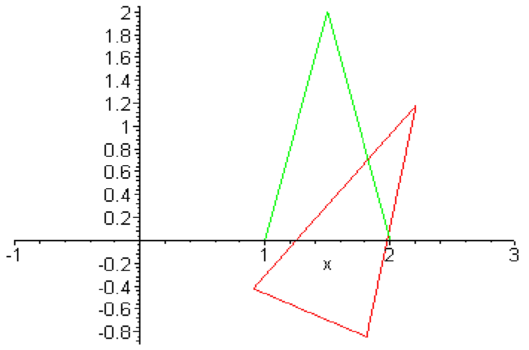
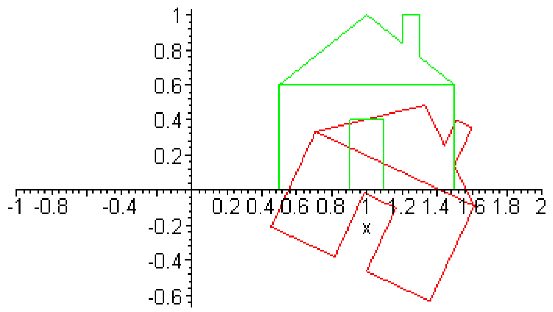
$$\mathbf{q = s*p} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx\,x \\ sy\,y \end{bmatrix}$$

Scaling about a Particular Point

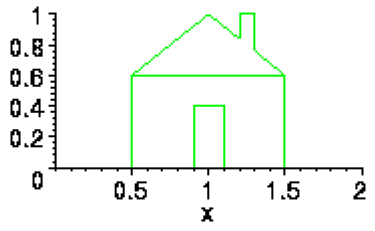



Rotation

Below, we see objects that have been rotate by 25 degrees.

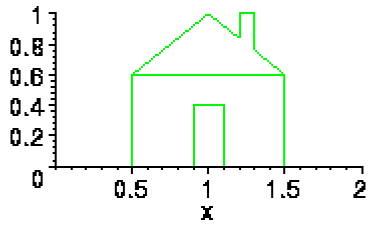
Again, we see that basic rotations are with respect to the origin:



Matrix/Vector Representation of Translations

counterclockwise rotation of ***a*** degrees = $\begin{bmatrix} \cos(a) & \sin(a) \\ -\sin(a) & \cos(a) \end{bmatrix}$

Shear



Matrix/Vector Representation of Translations

shear along x axis = $\begin{bmatrix} 1 & shearx \\ 0 & 1 \end{bmatrix}$

shear along y axis = $\begin{bmatrix} 1 & 0 \\ sheary & 1 \end{bmatrix}$

b. How do we perform the perspective projection from eye space into screen space? (6)

Answer:

A typical eye space w Eye
- Acts as the COP
- Placed at the origin
- Looks down the z-axis

w Screen
- Lies in the PP
- Perpendicular to z-axis
- At distance d from the eye
- Centered on z-axis, with radius s

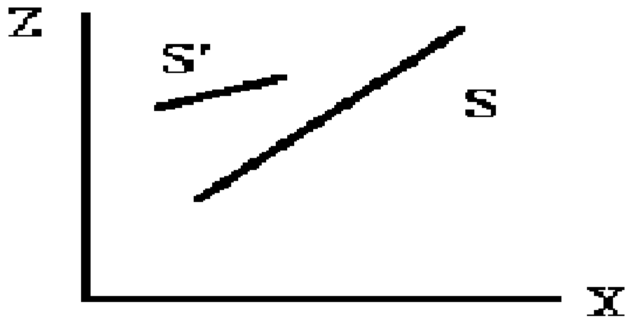
Q.6 a. How visible surface determination is done? Explain the painter's algorithm for detecting combined object and image space. (10)

Answer:

Visible Surface Determination: Painter's Algorithm

The painter's algorithm is based on depth sorting and is a combined object and image space algorithm. It is as follows:
Sort all polygons according to z value (object space); Simplest to use maximum z valueDraw polygons from back (maximum z) to front (minimum z). This can be used for wireframe drawings as well by the following:Draw solid polygons using Polyscan (in the background color) followed by Polyline (polygon color).Polyscan erases Polygons behind it then Polyline draws new Polygon.

Problems with simple Painter's algorithm

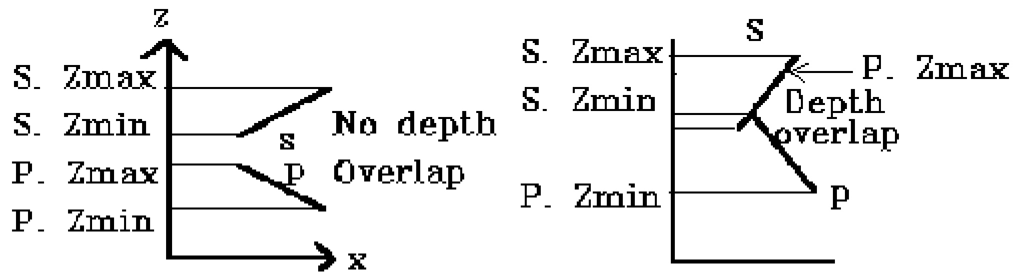


Look at cases where it doesn't work correctly. S has a greater depth than S' and so will be drawn first. But S' should be drawn first since it is obscured by S. We must somehow reorder S and S'.:

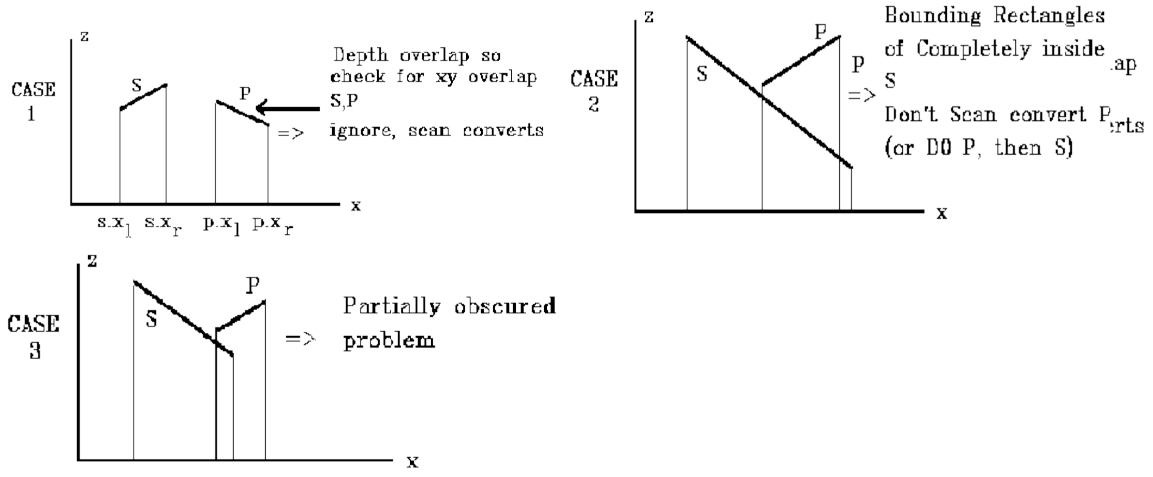
We will perform a series of tests to determine, if two polygons need to be reordered. If the polygons fail a test, then the next test must be performed. If the polygons fail all tests, then they are reordered. The initial tests are computationally cheap, but the later tests are more expensive.So look at revised algorithm to test for possible reordering
Could store Zmax, Zmin for each Polygon.

- sort on Zmax
- start with polygon with greatest depth (S)
- compare S with all other polygons (P) to see if there is any depth overlap(Test 0)

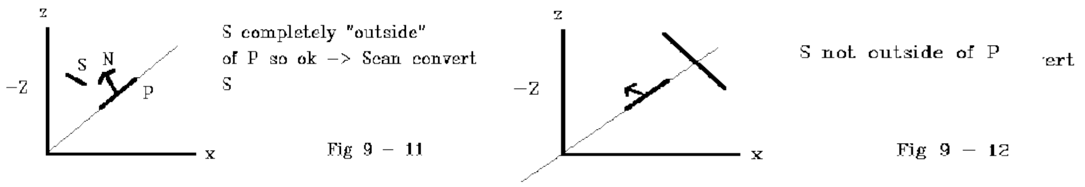
If S.Zmin<= P.Zmax then have depth overlap (as in above and below figures)



If have depth overlap (failed Test 0) we may need to reorder polygons.
Next (Test 1) check to see if polygons overlap in xy plane (use bounding rectangles)



Do above tests for x and y
If have case 1 or 2 then we are done (passed Test 1) but for case 3 we need further testing failed Test 1)
Next test (Test 2) to see if polygon S is "outside" of polygon P (relative to view plane)
Remember: a point (x, y, z) is "outside" of a plane if we put that point into the plane equation and get:
Ax + By + Cz + D > 0
So to test for S outside of P, put all vertices of S into the plane equation for P and check that all vertices give a result that is > 0.
i.e. Ax' + By' + Cz' + D > 0 x', y', z' are S vertices
A, B, C, D are from plane equation of P (choose normal away from view plane since define "outside" with respect to the view plane)



If the test of S "outside" of P fails, then test to see if P is "inside" of S (again with respect to the view plane)
(Test 3).

Compute plane equation of S and put in all vertices of P, if all vertices of P inside of S then P inside.
inside test: Ax' + By' + Cz' + D &lt0 where x', y', z' are coordinates of P vertices
so for above case:

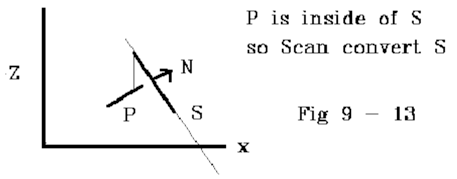


Fig 9 – 13

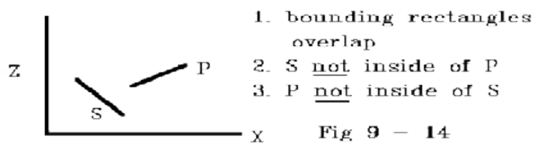
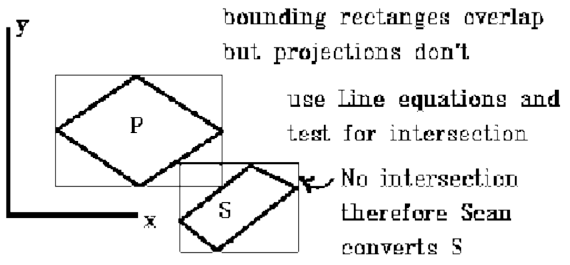


Fig 9 – 14

Then we do the 4th test and check for overlap for actual projections in xy plane since may have bounding rectangles overlap but not actual overlapFor example: Look at projection of two polygons in the xy planeThen have two possible cases.



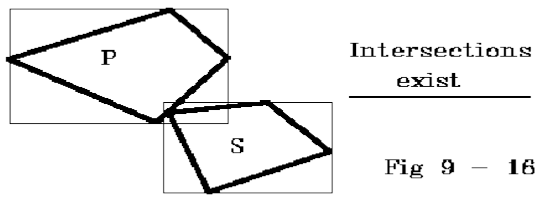
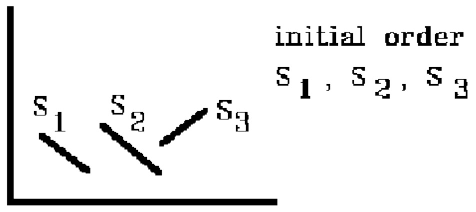


Fig 9 – 16

All 4 tests have failed therefore interchange P and S and scan convert P before S. But before we scan convert P we must test P against all other polygons. Look at an example of multiple interchanges



Test S1 against S2 and it fails all tests so reorder: S2, S1, S3

Test S2 against S3 and it fails all tests so reorder: S3, S2, S1

Possible Problem: Polygons that alternately obscure one another. These three polygons will continuously reorder.

One solution might be to flag a reordered polygon and subdivide the polygon into several smaller polygons.

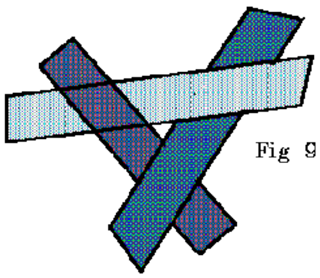
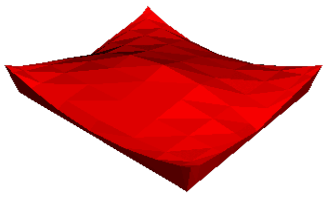


Fig 9 – 18

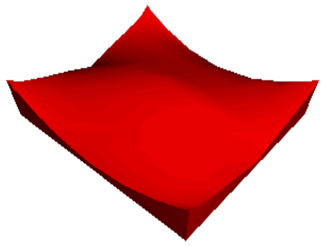
b. What is Gouraud shading? Explain it with an example. What are its drawbacks? (8)

Answer:

Gouraud shading is a method for linearly interpolating a colour or shade across a polygon. It was invented by Gouraud in 1971. It is a very simple and effective method of adding a curved feel to a polygon that would otherwise appear flat. It can also be used to depth que a scene, giving the appearance of objects in the distance becoming obscured by mist.



On the left is a curved surface rendered using flat shaded polygons, the other is Gouraud shaded. It is quite clearly smoother and more attractive and well worth spending the time to code.



Unlike a flat shaded polygon, you can specfy a different shade for each vertex of a polygon, the rendering engine then smoothly interpolates the shade across the surface. The technique is very similar to the standard scan converting, and can be handled very quickly with integer maths.

Some points to note though: It is best to restrict gouraud shading to three sided polygons. Sometimes polygons may not look quite as you expect when they have more than three.

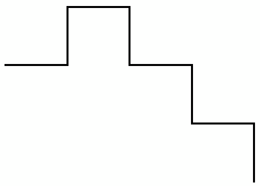
Gouraud shading is by no means perfect, but it can make a real difference over flat shaded polygons. Problem with Gouraud shading occur when you try to mix light sourcing calculations with big polygons.

Q.7 a. What is random midpoint displacement method? Explain with an example. (10)

Answer:

Random midpoint displacement method

Random midpoint displacement method introduced by Fouriner *et al.* which represents de facto standard in fractal terrains generation techniques. The principle is as follows.



An initial square is subdivided into four smaller squares. Let us have four

$$[x_0, y_0, f(x_0, y_0)],$$
$$[x_1, y_0, f(x_1, y_0)],$$
$$[x_0, y_1, f(x_0, y_1)],$$
$$[x_1, y_1, f(x_1, y_1)]$$

points .

In the first step we add one vertex into the middle. The vertex is denoted by $[x_{1/2}, y_{1/2}, f(x_{1/2}, y_{1/2})]$, where

$$\begin{aligned} x_{1/2} &= \frac{1}{2}(x_0 + x_1) y_{1/2} = \\ \frac{1}{2}(y_0 + y_1) f(x_{1/2}, y_{1/2}) &= \frac{1}{4}(f(x_0, y_0) + f(x_1, y_0) + f(x_0, y_1) + f(x_1, y_1)) \end{aligned}$$

The added vertex is shifted in *z*-coordinate direction by random value denoted by δ_1. This procedure is recursively repeated for each subsquare, then for every their descendants, and so on.

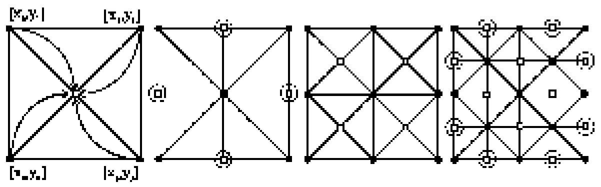


Figure 2: First four steps in random midpoint displacement method

In order to be resulting surface *fBm,* the random number δ_i must be generated with Gaussian distribution $[\mu = 0, \sigma = 1]$ and in the *i-th* iteration step the variation σ_i have to be modified according to

$$\sigma_i^2 = \frac{1}{2^{2H(i+1)}}\sigma^2, \qquad (1)$$

where *H* denotes Hurst exponent ($1 \leq H \leq 2$). From equation (1) we can see, that the first iteration has the biggest influence to the resulting shape of the surface and influence of the others decreases.

In the second step we calculate the points on the edges of initial square. We virtually rotate square by 45^o and calculate the values as in the previous step. The problem is in the cases when the new point has just three neighbors. In this case we calculate the average of three neighbors only. The error produced on the border could be neglected.

In the next step we virtually rotate the square back by 45^o and we recursively apply the first two steps on the four new squares as is mentioned above. This recursive process ends after given number of iteration.

Fractal dimension *D* of surface is obtained by

$$D = 3 - H.$$

The fractal dimension of this surface is *D*=2.5.

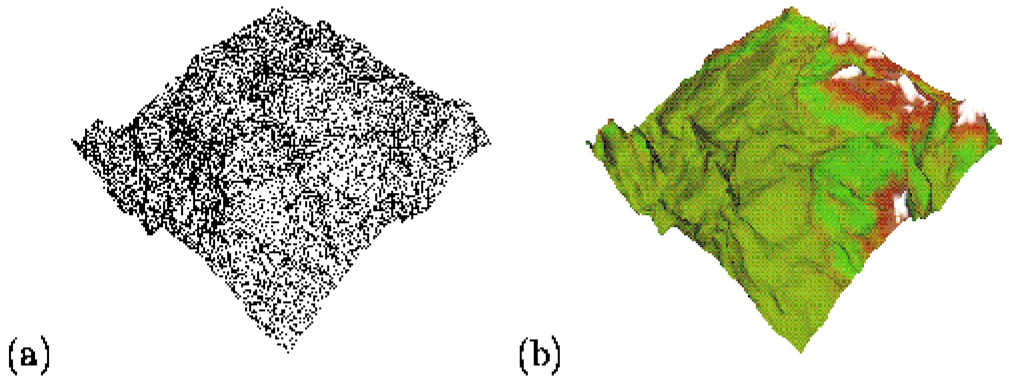


Figure: Example of fractal terrain with fractal dimension *D*=2.5. (a) wire frame model (b) the same model textured

b. Explain the concept of simulating accelerations in animation. (4)

Answer:

Simulating Accelerations: - An animation can be specified by giving motion parameters (position, acceleration).This determines the rejections for the in between frames. To simulate accelerations the time spacings can be adjusted for the in between frames.

(a)Zero Acceleration: - For constant acceleration equal time interval spacing is used in the in between frames motion path can be generated by using sp line curves or any straight line motion path.

(b)Non Zero Acceleration: - They are used to display speed changes, particularly at the beginning and the end of the motion sequence. We can model the startup and slow down portions of an animation path with spline or trigonometric functions or through Parabolic Curves.

(i)Positive acceleration: - To model increasing speed the time spacing between the frames is increased so that greater changes in position occurs as the object moves faster.

(ii)Decreasing accelerations: - The in between spacing is decrease to make the movement of object slaves

c. Write a short note on Self Similar fractals. (4)

Answer:

Geometric Construction of Deterministic Self-Similar Fractals

To geometrically construct a deterministic (nonrandom) self-similar fractal, we start with a given geometric shape, called the *initiator*. Subparts of the initiator are then replaced with a pattern, called the *generator*.

As an example, if we use the initiator and generator shown in Fig. 8-70, we can construct the snowflake pattern, or Koch curve, shown in Fig. 8-71. Each straight-line segment in the initiator is replaced with the generator pattern, consisting of four equal-length line segments. Then the generator is scaled and applied to the line segments of the modified initiator, and this process is repeated for some number of steps. The scaling factor at each step is $\frac{1}{3}$, so the fractal dimension is $D = \ln 4/\ln 3 \approx 1.2619$. Also, the length of each line segment in the initiator increases by a factor of $\frac{4}{3}$ at each step, so that the length of the fractal curve tends to infinity as more detail is added to the curve (Fig. 8-72). Figure 8-73 illustrates additional generator patterns that could be used for self-similar fractal curve constructions. The generators in Fig. 8-73(b) and (c) contain more detail than the Koch curve generator, and they have higher fractal dimensions.

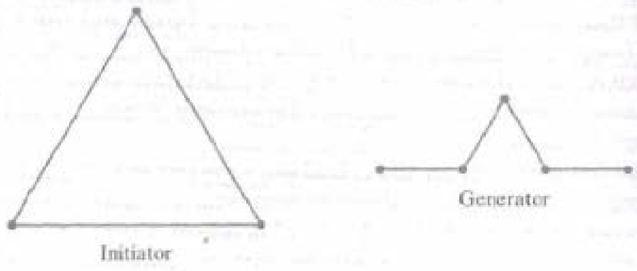


FIGURE 8-70 Initiator and generator for the Koch curve.

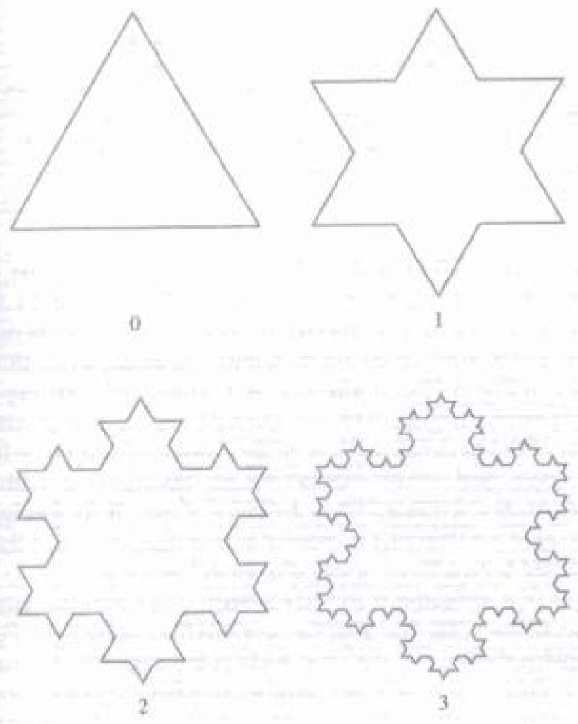


FIGURE 8-71 First three iterations in the generation of the Koch curve.

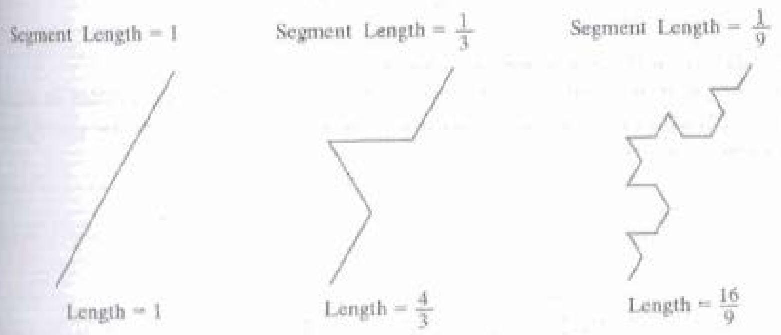


FIGURE 8-72 The length of each side of the Koch curve increases by a factor of $\frac{4}{3}$ at each step, while the line segment lengths are reduced by a factor of $\frac{1}{3}$.

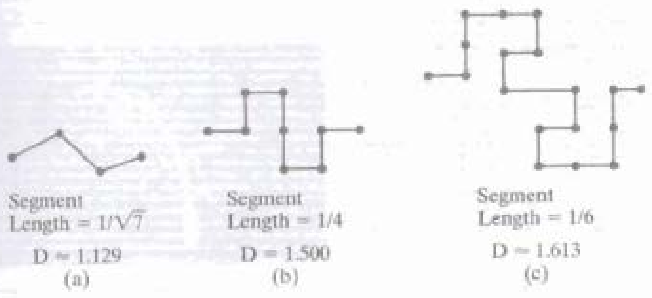


FIGURE 8-73 Generators for self-similar fractal curve constructions and their associated fractal dimensions.

TEXT-BOOK

I. **Computer graphics with OpenGL by Hearn and Baker, Third Edition, 2009 (Indian Edition) Pearson Education**