

7. ELEMENTS OF RASTER SCAN GRAPHICS

7.1 LINE DRAWING ALGORITHMS

Since a cathode ray tube (CRT) raster display can be considered a matrix of discrete cells (pixels) each of which can be made bright, it is not possible to directly draw a straight line from one point to another. The process of determining which pixels will provide the best approximation to the desired line is properly known as **rasterization**. Combined with the process of rendering the picture in scan line order it is known as scan conversion. For horizontal, vertical, and 45° lines the choice of raster elements is obvious. For any other orientation the choice is more difficult. This is shown in Fig. 7.1.

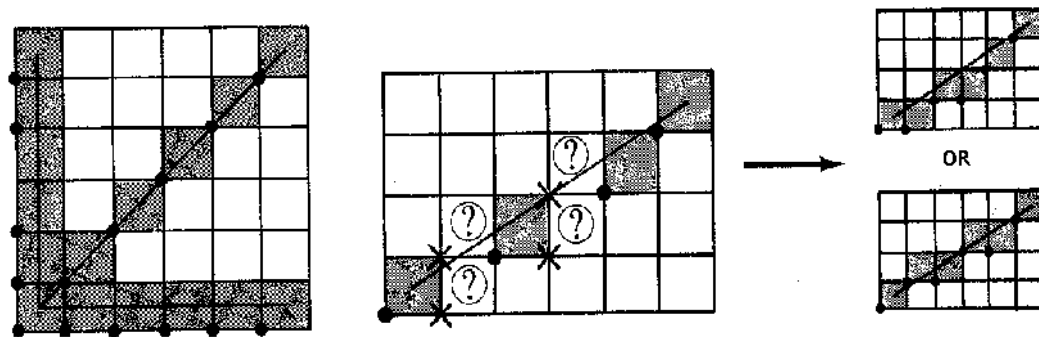


Figure 7.1

Before discussing specific line drawing algorithms it is useful to consider the general requirements for such algorithms, i.e., what are the desirable characteristics for these lines. Certainly, straight lines should appear as straight lines and they should start and end accurately. Further, displayed lines should have constant brightness along their length independent of the line length and orientation. Finally the lines should be drawn rapidly. As with most design criteria not all can be completely satisfied. The very nature of a raster scan display precludes the generation of a completely straight line except for special cases. It is also not possible for a line to precisely begins and ends at specified locations. However, with reasonable display resolution, acceptable approximations are possible.

In any case, the line drawing algorithm should light up only those pixels that are close to the theoretical line. On the other hand, the requirements for constant brightness, independent of the line slope could not be achieved on a single black and white screen. A simple frame buffer will produce only horizontal, vertical, and 45° lines with the brightness constant along the length. For all other orientations the rasterization will yield uneven brightness. This is shown in Fig. 7.1. Even for the special cases, the brightness is orientation dependent; e.g., note that the effective spacing between pixels for the 45° line is greater

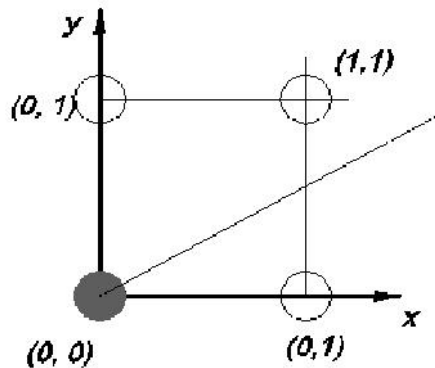
than for the vertical and horizontal line. This will make the vertical and horizontal lines appear brighter than the 45° line. Providing equal brightness along lines of varying length and orientation requires a frame buffer with additional bit-planes and calculation of a square root. Such a situation is discussed briefly in section 7.1.3. Additional calculations however slow down the line generation speed.

There are number of line drawing algorithms developed. Most of them use incremental methods to simplify the calculations.

7.1.1 BRESENHAM'S ALGORITHM

Although originally developed for digital plotters, Bresenham's algorithm is equally suited for use with CRT raster devices. The algorithm seeks to select the optimum raster locations to represent a straight line. To accomplish this, the algorithm always increments in by one unit either x or y depending on the slope of the line. The increment in the other variable, either zero or one, is determined by examining the distance between the actual line location and the nearest grid locations. This distance is called the **error**.

The algorithm is cleverly constructed so that only the sign of this error term need to be examined. This is illustrated in Fig. 7.2 for a line in the first octant, i.e., for a line with a slope between zero and one. From Fig. 7.2 note that, if the slope of the required line through $(0, 0)$ is greater than $1/2$, then its intercept with the line $x = 1$ will be closer to the line $y = 1$ than to the line $y = 0$. Hence, the raster point at $(1,1)$ better represents the path of the line than that at $(1, 0)$. If the slope is less than $1/2$, then the opposite is true. For a slope of precisely $1/2$ there is no clear choice. Here the algorithm chooses $(1,1)$.



$$\frac{1}{2} \leq \frac{\Delta y}{\Delta x} \leq 1 \text{ (error} \geq 0) \Rightarrow \text{Plot}(1,1)$$

$$0 \leq \frac{\Delta y}{\Delta x} < \frac{1}{2} \text{ (error} < 0) \Rightarrow \text{Plot}(1,0)$$

Figure. 7.2

Not all lines pass precisely through a raster point. This is illustrated in Fig. 7.3 where a line of slope $3/8$ initially passes through the raster point at $(0, 0)$ and subsequently crosses three pixels. Also illustrated is the calculation of the error in representing the line by discrete pixels. Since it is desirable to check only the sign of the error term, it is initialized to $-1/2$. Thus, if the slope is greater or equal to $1/2$, its value at the next raster point one unit away

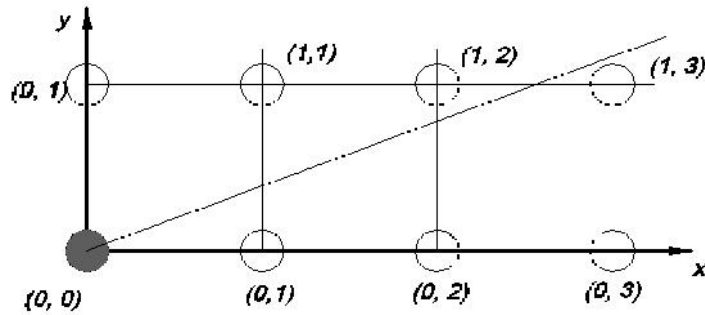


Figure 7.3

$(1,0)$ can be determined by adding the slope of the line to the error term: $e = e + m$ where m is the slope. In the case with e initialized to $-1/2$:

$$e = -1/2 + 3/8 = -1/8$$

Since e is negative, the line will pass below the middle of the pixel. Hence the pixel at the same horizontal level better approximates the location of the line so y is not incremented. Again incrementing the error term by the slope yields

$$e = -1/8 + 3/8 = 1/4$$

at the next raster point $(2,0)$. Here e is positive which shows that the line passes above the midpoint. The raster element at the next higher vertical level $(2,1)$ better approximates the position of the line. Hence y is incremented by one unit. Before considering the next pixel, it is necessary to reinitialize the error term. This is accomplished by subtracting one from it.

$$e = 1/4 - 1 = -3/4$$

Notice that the intercept of the vertical line at $x = 2$ and the desired line is $-1/4$ with respect to the line $y = 1$. Reinitializing to $-1/2$ relative to zero for the error term yields as above $-3/4$. Continuing to the next raster unit yields:

$$e = -3/4 + 3/8 = -3/8$$

Since the value of e is negative the y value is not incremented. This discussion illustrates that the error term is a measure of the y intercept of the desired line at each raster element referenced to $-1/2$.

7.1.2 INTEGER BRESENHAM'S ALGORITHM

Bresenham's algorithm as presented above requires the use of floating point arithmetic and division to calculate the slope of the line and to evaluate the error term. The speed of the algorithm can be increased by using integer arithmetic and eliminating the division. Since only the sign of the error term is important, the simple transformation

$$\bar{e} = 2e\Delta x$$

of the error term in the previous algorithm yields an integer algorithm. This allows the algorithm to be efficiently implemented in hardware or firmware.

In this case the value of the modified error term at the first point will be $(-1/2)$ is initial value for the error term):

$$\bar{e} = 2 \cdot \left(-\frac{1}{2} + \frac{\Delta y}{\Delta x}\right) \cdot \Delta x = 2\Delta y - \Delta x. \text{ For the next point:}$$

$$\bar{e} = 2 \cdot \left(e + \frac{\Delta y}{\Delta x}\right) \cdot \Delta x = 2e\Delta x + 2\Delta y = \bar{e} + 2\Delta y$$

If the modified error term $\bar{e}$ appears to be greater than 1, $\bar{e} > 1$, the reinitialization by subtracting one from the previous value will be:

$$\bar{e} = 2 \cdot (e - 1) \Delta x = 2e\Delta x - 2\Delta x = \bar{e} - 2\Delta x$$

$$\begin{aligned} \bar{e} < 1 &\Rightarrow \bar{e} = \bar{e} + 2\Delta y \\ \text{Finally: if } \bar{e} \geq 1 &\Rightarrow \bar{e} = \bar{e} - 2\Delta x \end{aligned}$$

Let us consider the same line with the slope of $3/8$, passing through the point $(0,0)$. This slope will yield $\Delta y = 3$, $\Delta x = 8$.

line(1,0) $\Rightarrow \bar{e} = 2\Delta y - \Delta x = 2 \cdot 3 - 8 = -2 < 0 \Rightarrow$ point (1,0) is set bright

line(2,0) $\Rightarrow \bar{e} = \bar{e} + 2\Delta y = -2 + 2 \cdot 3 = 4 > 0 \Rightarrow$ point (2,1) is set bright

Reinitialization is made by setting $\bar{e} = \bar{e} - 2\Delta x = 4 - 2 \cdot 8 = -12$

line(3,0) $\Rightarrow \bar{e} = \bar{e} + 2\Delta y = -12 + 2 \cdot 3 = -6 < 0 \Rightarrow$ point (3,1) is set bright

line(4,0) $\Rightarrow \bar{e} = \bar{e} + 2\Delta y = -6 + 2 \cdot 3 = 0 \Rightarrow$ point (4,2) is set bright

Reinitialization is made by setting $\bar{e} = \bar{e} - 2\Delta x = 0 - 2 \cdot 8 = -16$

A full implementation of Bresenham's algorithm requires modification for lines lying in the other octants. These can be easily developed by considering the quadrant in which the line lies and its slope. When the absolute value of the slope of the line is greater than 1, y is incremented by one and the error criteria is used to determine whether or not to increment x .

Example 3 program and line_b program listings (Listing 3.) given in Appendix A represent a MATLAB program code to draw a line using generalized Bresenham's algorithm. The result of such generation is shown on Fig.7.4.

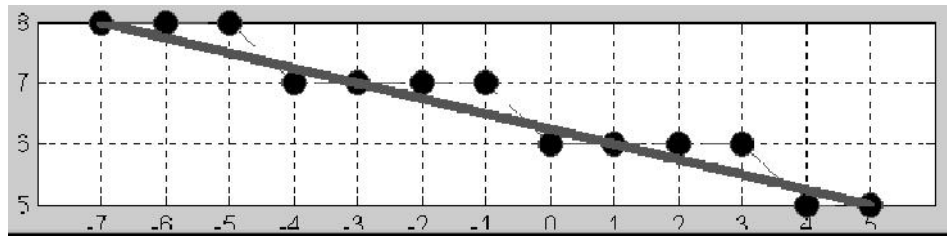


Figure 7.4

7.1.3 ANTIALIASING OF LINES

The finite size of the screen pixels produces lines with a jagged or stepped appearance. The jagged effect can of course be reduced by increasing the screen resolution. Alternatively, the jagged appearance or *aliasing* of lines can be reduced by incorporating **antialiasing** into the line-drawing algorithm. Although a thorough understanding of antialiasing requires the use of filtering theory, it can be readily seen that aliasing can be reduced by using a range of pixel intensities to shade the pixels in proportion to their distance from the exact line vector. That is, a pixel will be at full intensity only if it is sufficiently close to the theoretical line. Pixels will fall off in intensity as they increase in distance from the line.

Antialiasing can considerably improve video images because video screens are of relatively low resolution, and the jagged edges of animated lines can be very noticeable as pixels are turned on and off while a line moves. Also, antialiasing techniques are incorporated into some new graphics systems for CAD, even though in the past they have not been used much because of the cost and loss of speed (antialiased lines typically take two or three times as long to draw as an ordinary line).

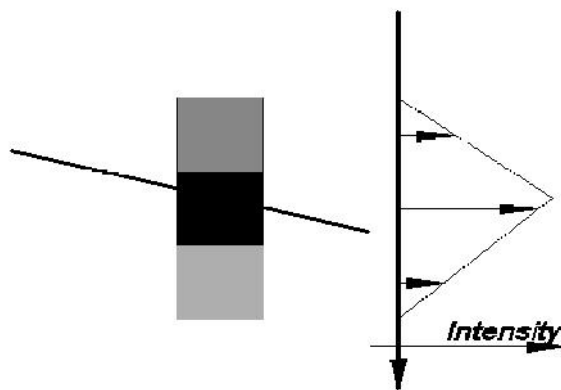


Figure 7.5

the figure determines the pixel intensities. Adjacent pixels have intensities that are separated by one-third the total vertical distance covered by the distribution function; that is, by two-thirds the distance from the maximum intensity to zero intensity.

Figure 7.5 illustrates how intensities may be assigned to pixels. In this example the ordinary Bresenham's algorithm would fill only the middle pixel (with full intensity, or black in this figure). With antialiasing, the middle pixel will be at full intensity only if the pixel is sufficiently close to the exact line position. The triangular distribution function on the right of

Defining a parameter $\mathcal{E} = e/ax$, where e is the error term in the integer Bresenham's algorithm, the pixel intensities can be assigned by the equations given below. In these relations i can be considered to be an intensity index number, with i_{min} and i_{max} representing the index numbers for minimum and maximum intensities. The range of intensities available depends on the computer graphics system.

$$i_{1,2} = i_{min} + \frac{2}{3}(i_{max} - i_{min}) \left(1 \pm \frac{\mathcal{E}}{2} \right)$$

$$i_3 = i_{min} \pm \frac{2}{3}(i_{max} - i_{min}) \left(\frac{\mathcal{E}}{2} \right)$$

For $\mathcal{E} < 0$, the top equation gives intensities for the middle and bottom pixels (y increases downward in the figure), with the minus used for the middle pixel and the plus used for the bottom pixel. The bottom equation with the minus gives the intensity index number of the top pixel. For $\mathcal{E} \approx 0$, the top equation with the plus is used for the middle pixel, the bottom equation with the plus for the bottom pixel, and the top equation with the minus for the top pixel. The above equations apply for $-1 \leq \mathcal{E} \leq +1$. For $\mathcal{E} < -1$ or $\mathcal{E} > +1$ the intensity of the middle pixel should be set to the maximum, with the intensities

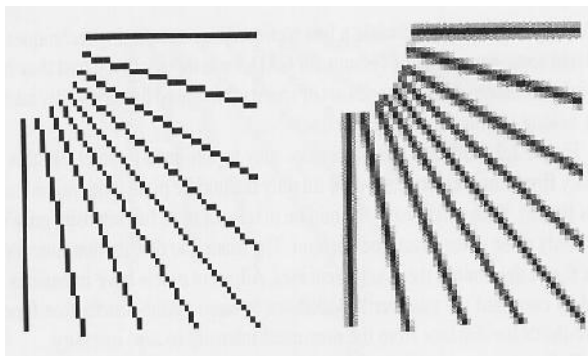


Figure 7.6

of the other two pixels computed from the above formulas. In applying the above formulas you should compute $\mathcal{E}$ just before the start of the `while` loop.

Figure 7.6 presents antialiased lines on the right and ordinary Bresenham lines on the left. The use of more than one pixel per position broadens the line somewhat but reduces the jagged appearance.

Also, it is easy to see that a moving line would undergo smoother changes in the antialiasing because a driven pixel would grow in intensity and then decay as a line passes over it.

7.2 CIRCLE GENERATION - BRESENHAM'S ALGORITHM

One of the most efficient and easiest to derive of the circle algorithms is due to Bresenham. To begin, note that only one octant of the circle needs be generated. The other parts can be obtained by successive reflections. If the first octant (0 to 45° counterclockwise) is generated, the second octant can be obtained by reflection through the line $y = x$ to yield the first quadrant. The results in the first quadrant are reflected through the line $x = 0$ to obtain those in the

second quadrant. The combined results in the upper semicircle are reflected through the line $y = 0$ to complete the circle.

To derive Bresenham's circle generation algorithm consider the first quadrant of an origin-centered circle. Notice that, if the algorithm begins at $x = 0, y = R$, then for clockwise generation of the circle y is a monotonically decreasing function of x in the first quadrant. Similarly, if the algorithm begins at $y = 0, x = R$, then for counterclockwise generation of the circle a monotonically decreasing function of y . Here, clockwise generation starting at $x = 0, y = R$ is chosen. The center of the circle and the starting point are both assumed to be located precisely at pixel elements.

For any given point on the circle, (x_i, y_i) then, for clockwise generation of the circle, there are only three possible selections for the next pixel which best represents the actual curve: horizontally to the right, diagonally downward to the right, and vertically downward. These are probable movements (and respective pixels) are labeled m_H, m_D, m_V , respectively, on Fig. 7.7.

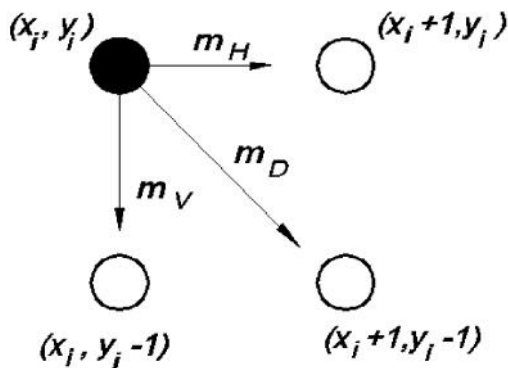


Figure 7.7

The algorithm chooses the pixel which minimizes the square of the distance between one of these pixels and the true circle, i.e. the minimum of:

$$m_H = |(x_i + 1)^2 + (y_i)^2 - R^2|$$

$$m_D = |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

$$m_V = |(x_i)^2 + (y_i - 1)^2 - R^2|$$

The difference between the square of the distance from the center of the circle to the diagonal pixel at (x_i+1, y_i-1) and the distance to a point of the circle R^2 is

$$\Delta_i = (x_i + 1)^2 + (y_i - 1)^2 - R^2$$

As with the Bresenham's line rasterizing algorithm, it is desirable to use the sign of an error term, rather than the magnitude, to select the appropriate pixel which best represents the actual circle.

If $\Delta_i < 0$, then the diagonal point $(x_i + 1, y_i - 1)$ is inside the actual circle. It is clear that either the pixel at $(x_i + 1, y_i)$, or that at $(x_i + 1, y_i - 1)$ must be chosen. To decide which, we will first examine the difference between the squares of the distance from the actual circle to the pixel at m_H and the distance from the actual circle to the pixel at m_D , (Fig. 7.8).

$$\delta = \delta_1 - \delta_2$$

$$\delta = |(x_i + 1)^2 + (y_i)^2 - R^2| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|$$

If $\Phi < 0$, then the distance from the actual circle to the diagonal pixel is greater than that to the horizontal pixel. Conversely, if $\Phi > 0$, then the distance to the horizontal pixel is greater. Thus, if:

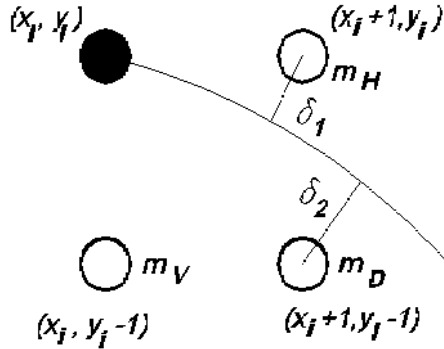


Figure 7.8

$\delta \leq 0$ choose m_H at $(x_i + 1, y_i)$

$\delta > 0$ choose m_D at $(x_i + 1, y_i - 1)$

The horizontal move has been selected when $\delta = 0$, i.e., when the distances are equal. The work involved in evaluating δ can be reduced by noting that if $\Delta_i < 0$ the pixel at $(x_i + 1, y_i)$ is always outside the actual circle, while the pixel at $(x_i + 1, y_i - 1)$ lies inside it, or:

$$(x_i + 1)^2 + (y_i - 1)^2 - R^2 < 0$$

$$(x_i + 1)^2 + (y_i)^2 - R^2 \geq 0$$

In this case we can skip the module and to form:

$$\delta = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + (x_i + 1)^2 + (y_i)^2 - R^2$$

$$\delta = 2.[(x_i + 1)^2 - R^2] + (y_i - 1)^2 + (y_i)^2$$

Completing the square term by adding and subtracting $-2y_i + 1$ yields:

$$\delta = 2.[(x_i + 1)^2 - (y_i - 1)^2 - R^2] + 2.y_i - 1$$

Using the definition for Δ_i gives : $\Phi = 2.(a_i + y_i) - 1$ which is considerably simpler.

If $(x_i + 1)^2 + (y_i)^2 - R^2 < 0$ - the pixel horizontally to the right is situated inside the actual circle as well. Since y is a monotonically decreasing function, the pixel at $(x_i + 1, y_i)$ must be chosen. Hence $\Phi < 0$ and the correct pixel is selected using the same criteria as above.

If $a_i > 0$, then the diagonal point $(x_i + 1, y_i - 1)$ is outside the actual circle. It is clear that either the pixel at $(x_i, y_i - 1)$, or that at $(x_i + 1, y_i - 1)$ must be chosen. Again the decision criteria can be obtained by examine the difference between the squares of the distance from the actual circle to the diagonal pixel at m_D and the distance from the actual circle to the pixel at m_V (Fig.7.9).

$$\delta' = \left| (x_i + 1)^2 + (y_i - 1)^2 - R^2 \right| - \left| (x_i)^2 + (y_i - 1)^2 - R^2 \right|$$

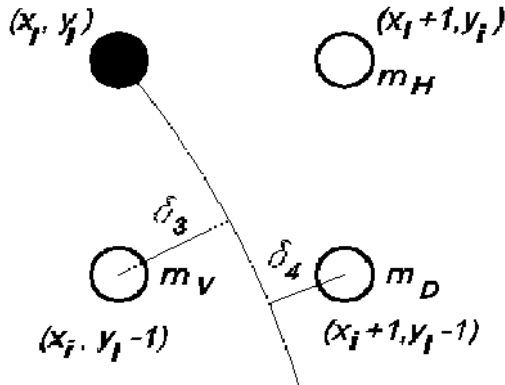


Figure 7.9

If $\Phi' < 0$, then the distance from the actual circle to the vertical pixel at $(x_i, y_i - 1)$ is greater and the diagonal move to the pixel m_D should be chosen and visa versa.

Similarly we can form the criteria in the form:

$$\delta' = (x_i + 1)^2 + (y_i - 1)^2 - R^2 + (x_i)^2 + (y_i - 1)^2 - R^2$$

since the diagonal pixel is outside the actual circle, while the vertical pixel is inside it. Thus, if:

$\delta' \leq 0$ choose m_D at $(x_i + 1, y_i - 1)$

$\delta' > 0$ choose m_v at $(x_i, y_i - 1)$

Completing the square for $(x_i)^2$ term by adding and subtracting $2x_i + 1$ yields:

$$\delta' = 2.[(x_i + 1)^2 - (y_i - 1)^2 - R^2] - 2x_i - 1$$

Using the definition for Δ_i gives: $\Phi' = 2.(a_i + x_i) - 1$ which again is considerably simpler.

Summarizing the results from the above yields:

$\Delta_i < 0$

$\delta \leq 0$ choose the pixel at $(x_i + 1, y_i)$ m_H

$\delta > 0$ choose the pixel at $(x_i + 1, y_i - 1)$ m_D

$\Delta_i > 0$

$\delta \leq 0$ choose the pixel at $(x_i + 1, y_i - 1)$ m_D

$\delta > 0$ choose the pixel at $(x_i, y_i - 1)$ m_v

$\Delta_i = 0$

choose the pixel at $(x_i + 1, y_i - 1)$ m_D

Simple recursion relationships which yield an incremental implementation of the algorithm are easily developed. First consider the horizontal movement m_H to the pixel at $(x_i + 1, y_i)$. Call it the next location $(i+1)$. The coordinates of the new pixel and the value of a_i are then:

$$x_{i+1} = x_i + 1$$

$$y_{i+1} = y_i$$

$$\Delta_{i+1} = (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2$$

$$\begin{aligned}\Delta_{i+1} &= (x_{i+1})^2 + 2 \cdot x_{i+1} + 1 + (y_i - 1)^2 - R^2 \\ \Delta_{i+1} &= (x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2 \cdot x_{i+1} + 1 \\ \Delta_{i+1} &= \Delta_i + 2 \cdot x_{i+1} + 1\end{aligned}$$

Similarly, the coordinates of the new pixel and the value of a_i for the move m_D to the pixel $(x_i + 1, y_i - 1)$ are:

$$\begin{aligned}x_{i+1} &= x_i + 1 \\ y_{i+1} &= y_i - 1 \\ \Delta_{i+1} &= (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2 \\ \Delta_{i+1} &= (x_{i+1})^2 + 2 \cdot x_{i+1} + 1 + (y_{i+1})^2 - 2 y_{i+1} + 1 - R^2 \\ \Delta_{i+1} &= (x_i + 1)^2 + (y_i - 1)^2 - R^2 + 2 \cdot x_{i+1} - 2 y_{i+1} + 1 \\ \Delta_{i+1} &= \Delta_i + 2 \cdot x_{i+1} - 2 y_{i+1} + 1\end{aligned}$$

Those for the move to $(x_i, y_i - 1)$ are:

$$\begin{aligned}x_{i+1} &= x_i \\ y_{i+1} &= y_i - 1 \\ \Delta_{i+1} &= (x_{i+1} + 1)^2 + (y_{i+1} - 1)^2 - R^2 \\ \Delta_{i+1} &= (x_i + 1)^2 + (y_{i+1})^2 - 2 y_{i+1} + 1 - R^2 \\ \Delta_{i+1} &= (x_i + 1)^2 + (y_i - 1)^2 - R^2 - 2 y_{i+1} + 1 \\ \Delta_{i+1} &= \Delta_i - 2 y_{i+1} + 1\end{aligned}$$

The implementation of Bresenham's incremental circle algorithm for the first quadrant is given as MATLAB programming code in Example 4 (Listing 4) of Appendix A. The result of circle generation is shown in Fig. 7.10.

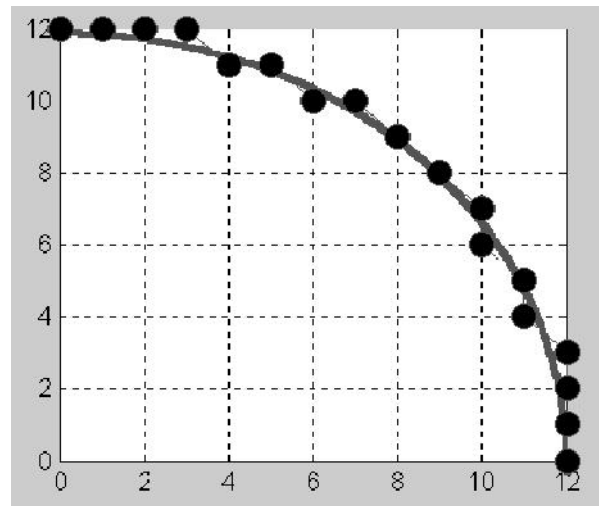


Figure 7.10

7.3 MIDPOINT LINE-SCAN ELLIPSE GENERATING

This approach can be used for generating arbitrary oriented conic sections such as ellipses, parabolas, hyperbolas and other curves. In this section the midpoint line-scan algorithm will be utilized in order to generate an ellipse, located at the origin of the coordinate system with axes $2a$ - the major axis and $2b$ - the minor axis respectively.

In order to simplify the considerations, the ellipse will be drawn in first quadrant. All the other quadrants can be obtained via mirroring the result through the coordinate axes.

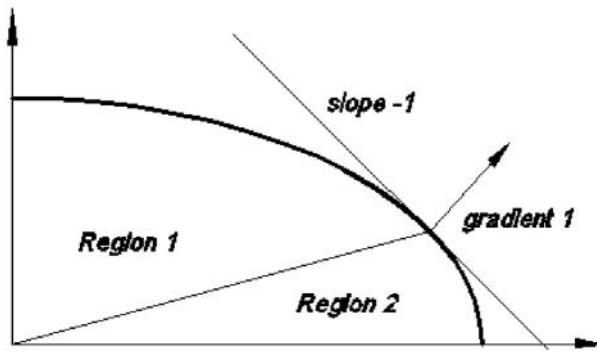


Figure 7.11

The midpoint line-scan algorithm differs from the Bresenham's algorithm in a way that the ellipse will be divided in two regions; the border between the two is the point where the gradient vector has value of 1. As the gradient vector is perpendicular to the slope of the curve at this point, the slope will have the value of -1 . Value of 1 also

means that the both projections of the gradient vector on the coordinate axes will have the same length. In the first region the projection of the gradient vector upon the y axis will be greater than the projection upon the x axis (the slope or the gradient vector will be greater than 45°). In the second region – the projection on the x axes will be greater. (Fig. 7.11).

The originally positioned ellipse is given by the equation in the form:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1; \quad F(x, y) = b^2 \cdot x^2 + a^2 \cdot y^2 - a^2 \cdot b^2 = 0$$

where $2a$ and $2b$ are the lengths of the major and minor axis respectively.

Thus the gradient vector is computed as:

$$\text{grad}F(x, y) = \frac{\partial F}{\partial x} \vec{i} + \frac{\partial F}{\partial y} \vec{j} = 2b^2 \cdot x \cdot \vec{i} - 2a^2 \cdot y \cdot \vec{j}$$

If we have a particular point of the ellipse, and following the strategy of the Bresenham's algorithm, we must decide which of the neighboring points – horizontal one, diagonal one or vertical one will best represent the actual curve. Given a point in the first region however we have an obvious choice of two points – one to the right and one at the diagonal position (down and to the

right). The third point (moving vertically) will not be relevant at all because of the slope of the gradient vector at that region.

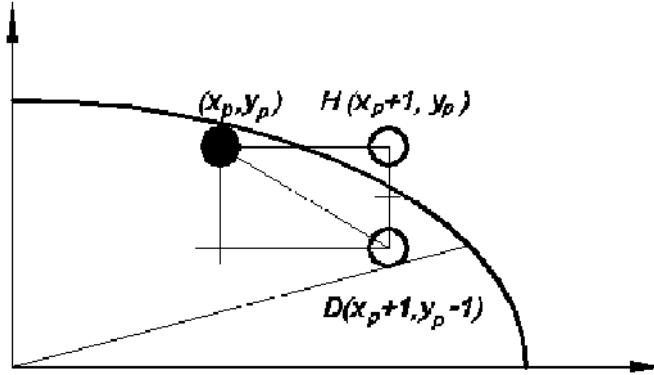


Figure 7.12

At this stage we can calculate the position of the midpoint between the horizontal (H) and diagonal (D) points we must choose between. It could be easily proved, that if the midpoint lays inside the actual ellipse, the point H will be closer; if the midpoint lays outside – the point D will represent the curve best (Fig. 7.12).

Generally the algorithm works the following way:

A. For the first region:

Considering a particular point (x_p, y_p) on the ellipse, a decision variable is calculated for the point $(x_p + 1, y_p - \frac{1}{2})$ in the form:

$$d_1 = F(x_p + 1, y_p - \frac{1}{2}) = b^2 \cdot (x_p + 1)^2 + a^2 \cdot (y_p - \frac{1}{2})^2 - a^2 \cdot b^2$$

If $d_1 < 0$ (the midpoint lies inside the ellipse) point H is chosen and x expands by 1).

If $d_1 \geq 0$ (the midpoint is outside or on the ellipse) point D is chosen and x increases by 1 while y decreases by 1.

Moving horizontally we calculate the next decision variable at the point $(x_p + 2, y_p - \frac{1}{2})$:

$$d_{1H} = F(x_p + 2, y_p - \frac{1}{2}) = b^2 \cdot (x_p + 2)^2 + a^2 \cdot (y_p - \frac{1}{2})^2 - a^2 \cdot b^2$$

$$d_{1H} = b^2 \cdot [(x_p + 1) + 1]^2 + a^2 \cdot (y_p - \frac{1}{2})^2 - a^2 \cdot b^2 = d_1 + b^2 \cdot (2x_p + 3)$$

For the move to D , the next midpoint is one increment for x and one decrement for y – point $(x_p + 2, y_p - \frac{3}{2})$. We calculate decision variable:

$$d_{1D} = F(x_p + 2, y_p - \frac{3}{2}) = b^2 \cdot (x_p + 2)^2 + a^2 \cdot (y_p - \frac{3}{2})^2 - a^2 \cdot b^2$$

$$d_{1D} = b^2 \cdot [(x_p + 1) + 1]^2 + a^2 \cdot [(y_p - \frac{1}{2}) - 1]^2 - a^2 \cdot b^2$$

$$d_{1D} = d_1 + b^2 \cdot (2x_p + 3) + a^2 \cdot (-2y_p + 2)$$

The algorithm usually starts at the point on the vertical (y) axis and continues in clockwise direction. In this case, for the starting point $(0, b)$ the decision has to be made between the points $(x+1, b)$ and $(x+1, b-1)$. The decision variable at the point $(x+1, b-1/2)$ is calculated as:

$$d_1 = b^2 \cdot 1 + a^2 \cdot (b - \frac{1}{2})^2 - a^2 \cdot b^2$$

$$d_1 = b^2 - a^2 \cdot (b - \frac{1}{4})$$

B. For the second region:

The choice will be between the pixels D and V , because of the slope of the gradient vector (which is less than 45° (Fig. 7.13). In this case, given a

particular point on the ellipse (x_p, y_p) , computations are made for the midpoint between the two possible next points:

$(x_p + 1/2, y_p - 1)$. The decision variable will be

$$d_2 = b^2 \cdot (x_p + \frac{1}{2})^2 + a^2 \cdot (y_p - 1)^2 - a^2 \cdot b^2$$

If d_2 is less than zero, point D is selected, otherwise point V will represent the curve best.

Proceeding to point D , next decision variable has to be computed with increment of $3/2$ in x direction and decrement of 2 in y direction:

$$d_{2D} = b^2 \cdot (x_p + \frac{3}{2})^2 + a^2 \cdot (y_p - 2)^2 - a^2 \cdot b^2$$

$$d_{2D} = b^2 \cdot [(x_p + \frac{1}{2}) + 1]^2 + a^2 \cdot [(y_p - 1) - 1]^2 - a^2 \cdot b^2$$

$$d_{2D} = d_2 + b^2 \cdot (2x_p + 2) + a^2 \cdot (-2y_p + 3)$$

When point V is chosen, the next midpoint will be obtained by no increment in x direction and decrement of 1 in y direction:

$$d_{2V} = b^2 \cdot (x_p + \frac{1}{2})^2 + a^2 \cdot (y_p - 2)^2 - a^2 \cdot b^2$$

$$d_{2V} = b^2 \cdot (x_p + \frac{1}{2})^2 + a^2 \cdot [(y_p - 1) - 1]^2 - a^2 \cdot b^2$$

$$d_{2V} = d_2 + a^2 \cdot (-2y_p + 3)$$

Figure 7.14 shows the result of the midpoint line-scan ellipse generating algorithm. This diagram is obtained using the MATLAB code presented in Example 5 (Listing 5) of Appendix A.

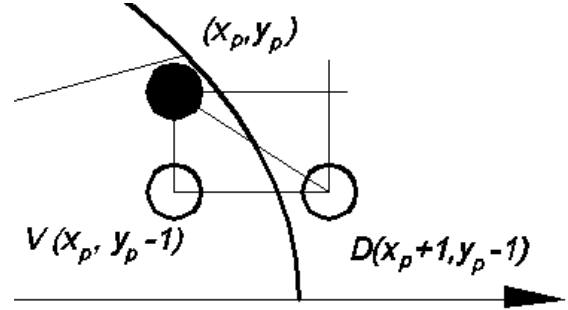


Figure 7.13

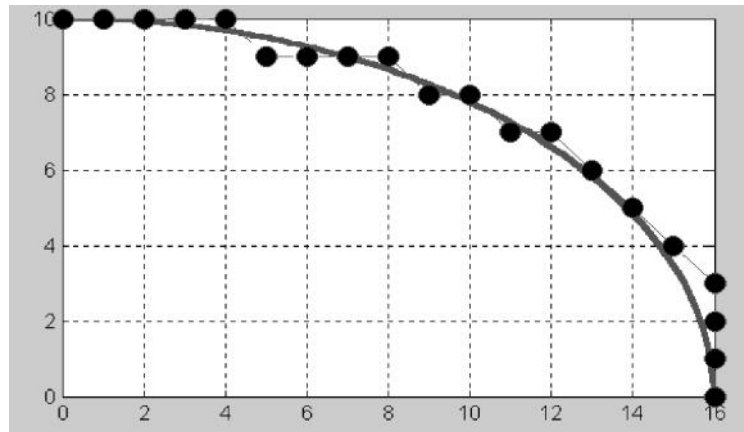


Figure 7.14

7.4 POLYGON FILLING

So far the discussion has been concerned with the presentation of lines on a raster scan device. However, one of the unique characteristics of a raster scan device is the ability to present solid areas. The generation of solid areas from simple edge or vertex descriptions is called solid area scan conversion, polygon filling, or **contour filling**. Several techniques can be used to fill a contour. They generally divide into two broad categories: **scan conversion** and **seed fill** (flood-fill).

Scan conversion techniques attempt to determine, in scan line order, whether or not a point is inside a polygon or contour. These algorithms generally proceed from the 'top' of the polygon or contour to the 'bottom'. The scan conversion techniques are equally applicable to line drawing displays. With line drawing displays they are used for cross-hatching or shading of contours.

Seed fill or flood-fill techniques assume that some point inside the closed contour is known. The algorithms then proceed to search for points adjacent to the seed point that is inside the contour. If the adjacent point is not inside the contour, then a boundary of the contour has been found. If the adjacent point is inside the contour, then it becomes a new seed point and the search continues recursively. Seed fill algorithms are only applicable to raster devices.

7.4.1 SCAN CONVERT POLYGONS

Many closed contours are simple polygons. If the contour is composed of curved lines, it can be approximated by a suitable polygon or polygons. The simplest method of filling a polygon is to examine every pixel in the raster to see if it is inside the polygon. Since most pixels will not be inside the polygon, this technique is wasteful. The amount of work can be reduced by computing the bounding box for the polygon. This is shown on Fig. 7.16. The bounding

box is the smallest rectangle that contains the polygon. Only those points inside the bounding box are to be examined. Using a bounding box for the polygon can significantly reduce the number of pixels examined. However, some of the polygons could be considerably smaller than the bounding box which again makes such approach unefficient.

A more efficient technique than the inside test can be developed by

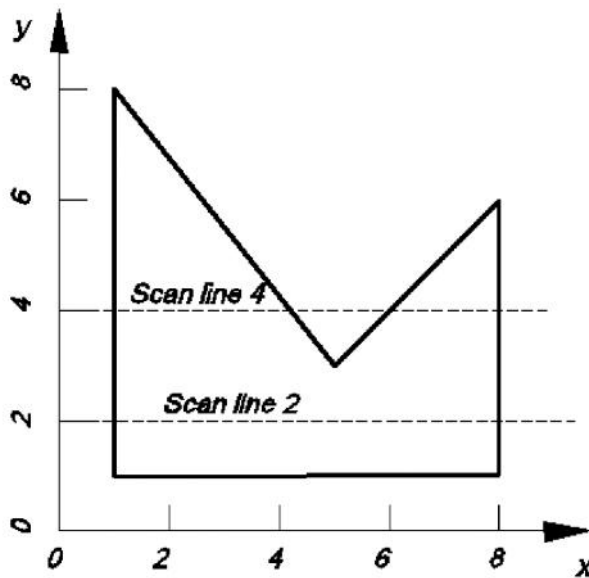


Figure 7.15

taking advantage of the fact that, except at boundary edges, adjacent pixels are likely to have the same characteristics. This property is referred to as **spatial coherence**. For a raster scan graphics device adjacent pixels on a scan line also can have the same characteristics. This is scan line coherence.

The characteristics of pixels on a given scan line change only where a polygon edge intersects the scan line. These intersections divide the scan line into regions.

For the simple polygon shown in Fig. 7.15 the scan line labeled 2 intersects the polygon at $x = 1$ and $x = 8$. These intersections divide the scan line into three regions:

$x < 1$	outside the polygon
$1 \leq x \leq 8$	inside the polygon
$x > 8$	outside the polygon

Similarly the scan line labeled 4 is divided into five regions:

$x < 1$	outside the polygon
$1 \leq x \leq 4$	inside the polygon
$4 < x < 6$	outside the polygon
$6 \leq x \leq 8$	inside the polygon
$x > 8$	outside the polygon

Thus the intersections of the edges with scan line 4 will be determined as 8, 6, 4, 1. They must then be sorted into ascending order in x , i.e. 1, 4, 6, 8.

In determining the intensity, color, or shade of the pixels on a scan line the sorted intersections are considered in pairs. For each interval formed by a pair of intersections, the intensity or color is that of the polygon. For intervals between pairs of intersections, the intensity or color is that of the background. Of course, from the beginning of the scan line until the first intersection and

from the last intersection to the end of the scan line the intensity or color is that specified for the background. For the polygon in Fig. 7.15 the pixels from 0 to 1, 4 to 6, and 8 to 10 on scan line 4 are set at the background color, while those from 1 to 4 and 6 to 8 are set at the polygon intensity or color.

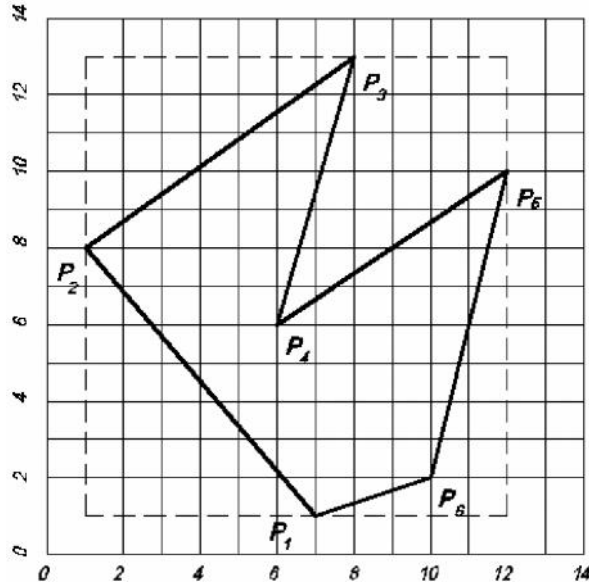


Figure 7.16

Horizontal edges cannot intersect a scan line and are thus ignored. This does not mean that horizontal edges are not formed. They are formed by the bottom and top edges of the rows of pixels.

An additional difficulty occurs when a scan line intersects the polygon precisely at a vertex. Figure 7.16 illustrates this problem. Using the scan line convention, the scan line at $y = 6$ intersects the polygon at 2.5, 6 and 11. This is an odd number of intersections. Hence, extracting the pixels in pairs will yield an incorrect result because an

exact number of pairs could not be obtained.

Observation suggests that at point (6, 6) two intersections should be counted. This is true, because the scan line intersects two edges (P_3, P_4 and P_4, P_5) at that point. Following that notation, the scan line at $y = 8$ intersects 5 edges (P_1, P_2 and P_2, P_3 at point 1, 8 as well as P_3, P_4 and P_4, P_5) which again gives us an odd number of intersections.

The correct result is obtained by counting two intersections when the scan line-polygon vertex intersections occur at local maxima or minima of the polygon and only one if not.

Whether the vertex under consideration is a local polygon maximum or minimum can be determined by examining the end points of the two edges meeting at the vertex. If the y values of these edges are both greater than the vertex being considered, then the vertex is a local minimum. If both are less than the vertex being considered, then the vertex is a local maximum. If one is greater and one less, then the vertex is neither a local minimum nor a local maximum. In Fig. 7.15, P_1 , P_3 and P_5 are local maxima, while P_4 is a local minimum, P_2 and P_6 are neither local maxima nor minima. Hence, two scan line intersections are counted at P_1 , P_3 , P_5 and P_4 , and only one at P_2 and P_6 .

Using the techniques discussed above, efficient algorithms for scan-converting solid area polygons can be developed. These are called **ordered edge list** algorithms. They depend upon sorting the polygon edge-scan line intersections into scan line order. The efficiency of the algorithms depend on the efficiency of the sorting. A particularly simple algorithm is:

To prepare data:

Determine for each polygon edge the intersections with the scan lines. A Bresenham or DDA algorithm can be used for this. Horizontal edges are ignored. Store each intersection (x, y) in a list;

Sort the list by scan line and increasing x on the scan line; i.e.

To scan-convert the data:

Extract pairs of elements from the sorted list (x_1, y_1) and (x_2, y_2) . The structure of the list ensures that $y_1 = y_2$ and $x_1 \leq x_2$. Activate pixels on the scan line y for integer values of x such that $x_1 \leq x \leq x_2$.

A practical example for implementation of the above algorithm using MATLAB programming is given by Example 5 (program listing 6) in Appendix A. The procedure uses the polygon given in Fig. 7.16 as input data. The result of the polygon filling is presented in Fig. 7.17.

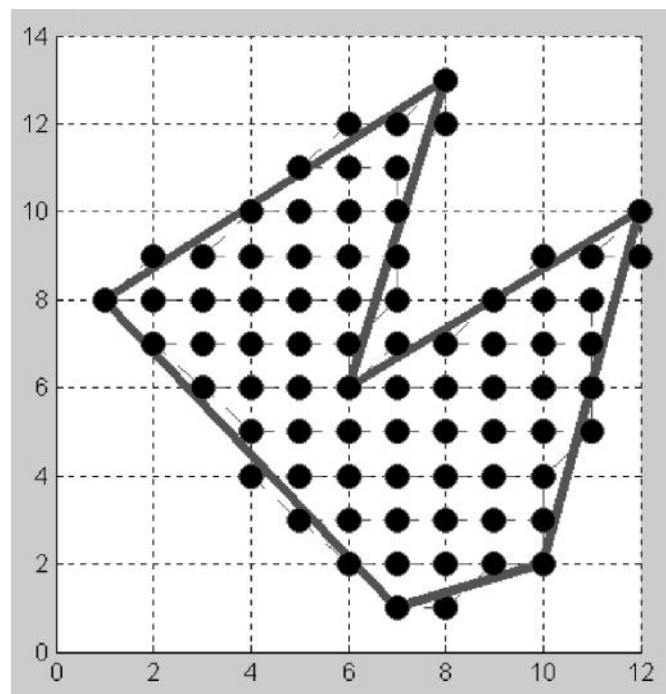


Figure 7.17

8. TRANSFORMATIONS AND PROJECTIONS

8.1 TWO DIMENSIONAL TRANSFORMATIONS

We will continue our study of the fundamentals of the mathematics underlying computer graphics by considering the representation and transformation of points and lines. Points and the lines, which join them, along with an appropriate drawing algorithm, are used to represent objects or to display information graphically.

Transformations can be applied in several situations. First they can be used for viewing the drawing as when one pans or zooms to translate or rescale the drawing on computer screen. Second, transformations can be applied to drawing entities (or group of entities) to change their positions and orientations relative to other entities in the drawing. Third, three dimensional transformations can be used to view 3D scenes from different points of view and thus to have clear vision on all surfaces of three-dimensional solid objects. It could also be desirable develop a perspective view of the object.

The ability to transform points and lines and curves is basic to computer graphics. All of these transformations can be accomplished using the mathematical techniques discussed in this chapter.

8.1.1 REPRESENTATION OF POINTS

A point is represented in two dimensions by its coordinates. These two values can be specified as the elements of a 1-row, 2-column matrix or 2 rows, 1 column matrix:

$$(x \ y), \begin{pmatrix} x \\ y \end{pmatrix} = (x \ y)^T \quad - \text{ in 2D space}$$

In three dimensions a 1 x 3 or 3 x 1 matrix is used:

$$(x \ y \ z), \begin{pmatrix} x \\ y \\ z \end{pmatrix} = (x \ y \ z)^T$$

Row matrixes or column matrixes as shown above are called position vectors. In this case they represent the projections of a vector connecting the origin of the coordinate system with the specified point on the coordinate axes. In this course a column matrix formulation will be used.

A series of points, each of which is a position vector relative to some coordinate system, is stored in a computer as a matrix or array of numbers. Ma-

nipulating the matrix, which defines the points, controls their position. Lines are drawn between points to generate lines, curves and pictures.

8.1.2 TRANSFORMATIONS AND MATRIXES

Consider the result of multiplication of matrix $(x\ y)^T$ containing the coordinates of a point P and general 2×2 transformation matrix:

$$(T).(X) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a.x+b.y \\ c.x+d.y \end{pmatrix} = \begin{pmatrix} x^* \\ y^* \end{pmatrix}$$

This means, that the original coordinates of the point P have been transformed to the x^* and y^* . In this case the coordinates $(x^* \ y^*)^T$ represent another point, i.e. we performed linear transformation.

Let's consider some special cases:

1. Let $a = a$; $b = 0$; $c = 0$; $d = 1$ then:

$\begin{pmatrix} a & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a.x \\ y \end{pmatrix} = \begin{pmatrix} x^* \\ y^* \end{pmatrix}$ produces a scale change in the x component of the position vector;

2. Let $a = a$; $b = 0$; $c = 0$; $d = d$ then:

$\begin{pmatrix} a & 0 \\ 0 & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a.x \\ y.d \end{pmatrix} = \begin{pmatrix} x^* \\ y^* \end{pmatrix}$ yields a scaling of both x and y coordinates of the original position vector. If $a \neq d$ then the scale changes are not equal. The scale factors a and d could be $\neq 0$. In these cases initial coordinates could be enlarged, compressed or left intact (if $a = d = 1$);

3. Let $a = -1$ or $a = 1$ $b = 0$; $c = 0$; then for example: $d = 1$; $d = -1$

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} -x \\ y \end{pmatrix} = \begin{pmatrix} x^* \\ y^* \end{pmatrix}$$

which yields a reflection about the y axes. If $b = 0$ and $c = 0$, $a = 1$ and $d = -1$ then reflection through x axes occurs.

4. Now consider the effect of off-diagonal terms. First consider $a = d = 1$ and $b = 0$.

$$\begin{pmatrix} 1 & 0 \\ c & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ x.c+y \end{pmatrix} = \begin{pmatrix} x^* \\ y^* \end{pmatrix}.$$

Note that x coordinate of the point P is not changed, while y^* depend linearly on original coordinates. This effect is called **shear**. Thus off diagonal terms produce a shearing effect on the coordinates of the position for P .

8.1.3 TRANSFORMATION OF STRAIGHT LINES

A straight line can be defined by two position vectors which specify the coordinates of its end points. The position and orientation of the line joining these two points can be changed by operating on these two position vectors. The actual operation of drawing a line between two points depends on the display device used. Here, we consider only the mathematical operations on the position vectors of the end points.

More compactly the line AB may be represented by 2×2 matrix where $A = (x_A \ y_A)^T$, $B = (x_B \ y_B)^T$, are the end points of the line. Combining the coordinates we get: $AB = \begin{pmatrix} x_A & x_B \\ y_A & y_B \end{pmatrix}$. Then given a 2×2 transformation matrix T we can transform original line AB into new line $A'B'$ using the matrix multiplication:

$$A'B' = T.AB = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_A & x_B \\ y_A & y_B \end{pmatrix} = \begin{pmatrix} a.x_A + b.y_A & a.x_B + b.y_B \\ c.x_A + d.y_A & c.x_B + d.y_B \end{pmatrix} = \begin{pmatrix} x'_A & x'_B \\ y'_A & y'_B \end{pmatrix}$$

In fact a 2×2 matrix transforms any straight into a second straight line. Points of the second line have one to one correspondence with the points of the first line. Similarly a midpoint of a line transforms in a midpoint of the transformed line. If a 2×2 matrix is used to transform two parallel lines, the result is a second pair of parallel lines.

Let consider a line with endpoint $A (x_1 \ y_1)^T$ and $B (x_2 \ y_2)^T$. Thus the slope of the line will be:

$$m = \frac{y_2 - y_1}{x_2 - x_1} \text{ while the slope of the transformed line : } m' = \frac{y'_2 - y'_1}{x'_2 - x'_1}.$$

Let $T = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, then the slope of the transformed line could be obtained:

$$m' = \frac{(cx_2 + dy_2) - (cx_1 + dy_1)}{(ax_2 + by_2) - (ax_1 + by_1)} = \frac{c.(x_2 - x_1) + d.(y_2 - y_1)}{a.(x_2 - x_1) + b.(y_2 - y_1)}$$

Dividing by $(x_2 - x_1)$ we obtain: $m' = \frac{c + d.m}{a + b.m}$. The slope of the transformed line does not depend on the endpoints of the original and transformed lines but only on the slope of the original line and the elements of the transformation ma-

trix which are the same for all the parallel lines. Thus parallel lines remain parallel after transformation.

It could be proved, that the intersection point of two lines transforms again in an intersection point.

8.1.4 ROTATION

What about rotation on an arbitrary angle about the origin of the coordinate system? To answer this question let consider the position vector from the origin to point P . The length of the vector is r and the angle is φ . The position vector is rotated about the origin by the angle θ to point P^* . Fig. 8.1 represents graphically this situation. We can write for the initial and final coordinates of the transformed vector the following equations:

$$x = r \cdot \cos \varphi$$

$$y = r \cdot \sin \varphi$$

$$x' = r \cdot \cos(\varphi + \theta) = r \cdot \cos \varphi \cdot \cos \theta - r \cdot \sin \varphi \cdot \sin \theta$$

$$y' = r \cdot \sin(\varphi + \theta) = r \cdot \cos \varphi \cdot \sin \theta + r \cdot \sin \varphi \cdot \cos \theta$$

$$x' = x \cdot \cos \theta - y \cdot \sin \theta$$

$$y' = x \cdot \sin \theta + y \cdot \cos \theta \quad \text{or}$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Thus the transformation for a general rotation about the origin by an arbitrary angle θ is given by the above matrix.

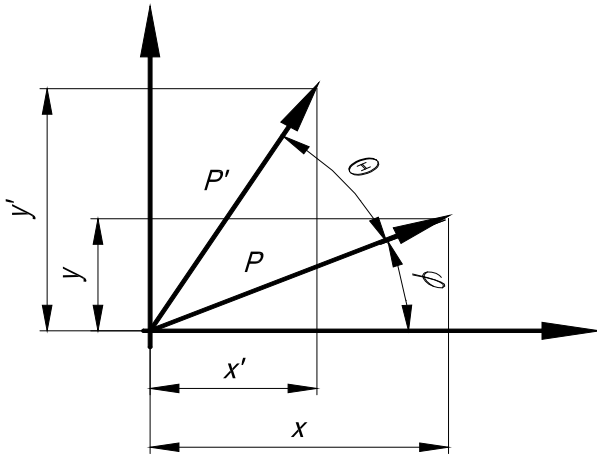


Figure 8.1

The inverse of the general rotation matrix T is its transpose. Since formally determining the inverse of a matrix is more computationally expensive than determining its transpose, this is an important and useful result. In general, the inverse of any pure rotation matrix, i.e., one with a determinant identically equal to $+1$, is its transpose. Such matrices are said to be orthogonal.

8.1.5 SCALING

Recalling the previous discussion we notice that the scaling of a position vector is controlled by the terms of the primary diagonal of the transformation matrix.

Let consider the following example: We have a triangle ABC given by the coordinates of the vertexes:

$$ABC = \begin{pmatrix} 4 & 4 & 2 \\ 2 & 4 & 4 \end{pmatrix}. \text{ Let also have a transformation matrix } T = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

The transformation of the given triangle produces a scaling of the vertexes

$$A'B'C' = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 4 & 4 & 2 \\ 2 & 4 & 4 \end{pmatrix} = \begin{pmatrix} 8 & 8 & 4 \\ 4 & 8 & 8 \end{pmatrix}$$

The scaling with the above transformation is graphically presented on

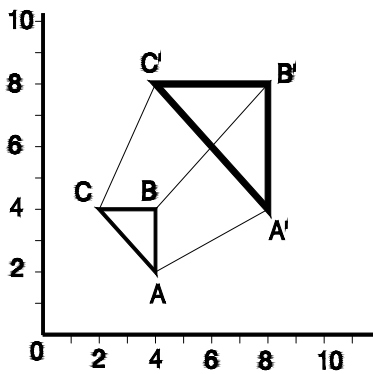


Figure 8.2

Figure 8.2. If $a = d$, $b = c = 0$, a uniform scaling occurs; and if $a \neq d$, $b = c = 0$, non-uniform scaling occurs. For a uniform scaling, if $a = d > 1$, a uniform expansion occurs; i.e., the figure gets larger. If $a = d < 1$, then a uniform compression occurs; i.e., the figure gets smaller. Non-uniform expansions and compression occur, depending on whether a and d are individually > 1 or < 1 .

The figure also reveals what at first glance is an apparent translation of the transformed triangles. This apparent translation is easily understood if we recall that the position vectors, not the points are scaled with respect to the origin. To obtain the pure scaling without apparent translation, the centroid of the figure must be at the origin. In particular CAD systems (AutoCAD for example), additional base-point or a centre of the scaling must be specified, and so placing the centre of the scene at the origin of a temporary coordinate system.

8.1.6 TRANSLATIONS AND HOMOGENEOUS COORDINATES. COMBINED TRANSFORMATIONS

A number of transformations governed by the general 2×2 transformation matrix, e.g., rotation, reflection, scaling, shearing etc., were discussed in the previous sections. As noted previously, the origin of the coordinate system is invariant with respect to all of these transformations. However, it is necessary to be able to modify the position of the origin, i.e., to transform every point in

the two dimensional plane. This can be accomplished by translating the origin or any other point in the two-dimensional plane, i.e.

$$\begin{aligned}x' &= a.x + c.y + m \\ y' &= b.x + d.y + n\end{aligned}$$

Unfortunately, it is not possible to introduce the constants of translation m, n into the general 2×2 transformation matrix; there is no room!

This difficulty can be overcome by introducing homogeneous coordinates. The homogeneous coordinates of a nonhomogeneous position vector $(x \ y)^T$ are $(x' \ y' \ h')^T$ where $x = x'/h$ and $y = y'/h$ and h is any real number. Note that $h = 0$ has special meaning representing points in the infinity.

There are not unique presentation of a position vector in homogenous coordinates as h can take any value. One set of homogeneous coordinates is always of the form $(x \ y \ 1)^T$. We choose this form to represent the position vector $(x \ y)^T$ in the physical xy plane. All other homogeneous coordinates are of the form $(x.h \ y.h \ h)^T$. There is no unique homogeneous coordinate representation, e.g., $(6 \ 4 \ 2)^T, (12 \ 8 \ 4)^T, (3 \ 2 \ 1)^T$ all represent the physical point $(3 \ 2)$. If we use homogeneous representation, the general transformation matrix is now 3×3 . Specifically:

$$T = \begin{pmatrix} a & b & m \\ c & d & n \\ 0 & 0 & 1 \end{pmatrix} \text{ where the elements } a, b, c, d, \text{ of the upper left submatrix}$$

are exactly the same as discussed above, m and n are translation factors in x and y directions respectively. When the pure two dimensional translation occurs, it could be written in the form:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & m \\ 0 & 1 & n \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + m \\ y + n \\ 1 \end{pmatrix} \text{ which is what we want to be.}$$

If we have a position vector P we can perform a linear transformation using a transformation matrix T_1 . The result vector will be: $P' = T_1 \cdot P$. Now we can do a second transformation with the new vector using new transformation matrix T_2 . Thus it yields: $P'' = T_2 \cdot P' = T_2 \cdot T_1 \cdot P$. In this case we have a new transformation matrix $T = T_1 \cdot T_2$. In general case we can form a transformation matrix obtained as matrix multiplication of several matrixes each of them representing some basic transformation. $T = T_n \cdot T_{n-1} \cdot \dots \cdot T_2 \cdot T_1$. Since the matrix multiplication is noncommutative, the order of application of the transformations is important.

8.1.7 ROTATION ABOUT AN ARBITRARY POINT

Previously we have considered rotation as occurs about the origin. Homogeneous coordinates provide a mechanism for accomplishing rotations about points other than the origin. The general idea under the transformations is as follows:

- ◆ make the point of rotation to be coincident with the origin of the current coordinate system;
- ◆ perform rotation about the origin (rotate a positional vector about the origin) using the rotational matrix as shown in section 8.1.4;
- ◆ restore the initial situation by back translation to the original position.

We will denote the coordinates of the rotational point as $(m \ n)^T$ (Figure 8.3). In general, a rotation about an arbitrary point in the plane could be accomplished in the following steps:

1. First translate the point of rotation, while it becomes coincident with the origin of the coordinate system. The transformation matrix will be:

$$T_1 = \begin{pmatrix} 1 & 0 & -m \\ 0 & 1 & -n \\ 0 & 0 & 1 \end{pmatrix}$$

2. Second, perform the desired rotation as rotation about the origin. The transformation matrix for pure rotation is:

$$T_2 = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Finally translate the result back to the original centre of rotation A. The third transformation matrix:

$$T_3 = \begin{pmatrix} 1 & 0 & m \\ 0 & 1 & n \\ 0 & 0 & 1 \end{pmatrix}$$

The all three transformations can be written together, bearing in mind the order of the transformations. We will obtain the transformed vector: $P' = T_3.T_2.T_1.P$ or:

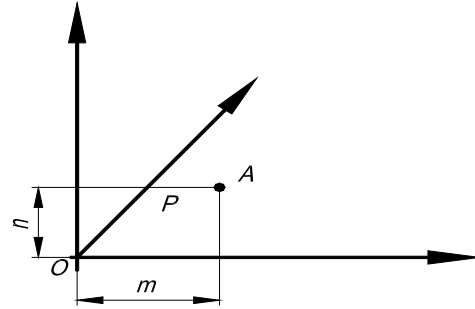


Figure 8.3

$$P' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & m \\ 0 & 1 & n \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -m \\ 0 & 1 & -n \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Example: Let's perform a rotation of a straight line defined by the end-points (in homogeneous coordinates) $P_1 = (4 \ 3 \ 1)^T$ and $P_2 = (5 \ 2 \ 1)^T$ about the point $A = (2 \ 1 \ 1)^T$. The rotational angle will be 45° .

The MATLAB code to perform all the necessary calculations is given in the Table 8.1.

Table 8.1

```
% Transformation 1
%
% Rotation of a line about an point in a plane
% First enter the angle of rotation
% and the endpoints of the line as 3 X 2 matrix

fi=45;
fi=fi*pi/180; % Convert to radians
P1P2=[4 5; 3 2; 1 1];
%
% Form the transformation matrixes
%
T1=[1 0 -2; 0 1 -1; 0 0 1];
T2=[cos(fi) -sin(fi) 0; sin(fi) cos(fi) 0; 0 0 1];
T3=[1 0 2; 0 1 1; 0 0 1];
T=T3*T2*T1;
%
% Perform the final multiplication and obtain the result
%
P1P2_prim=T*P1P2
% Results

P1P2_prim =

    2.0000    3.4142
    3.8284    3.8284
    1.0000    1.0000
```

One can use a particular available CAD system to perform the operation rotation and to verify the results.

8.1.8 REFLECTION THROUGH AN ARBITRARY AXIS

While the pure two dimensional rotation is regarded as rotation in the xy plane around an axes perpendicular to that plane, reflection could be regarded as rotation on angle 180° in three dimensional space around a line lying in the xy plane.

Previously (see Sec. 8.1.2) reflection through coordinate axes (passing through the origin) was discussed. Occasionally reflection of an object through a line that does not pass through the origin (arbitrary line in the plane) is required (such reflection is represented graphically in Fig. 8.4). This can be accomplished utilizing the idea, that we can always make the arbitrary mirror axis coincident with one of the coordinate system axes. We can use some simple transformations to do that. Next we can perform the reflection and then restore the initial situation utilizing inverse transformations. The necessary transformations involved in this case are similar to these for rotation about an arbitrary point. Specifically:

1. Translate the line and the object so that the line passes through the origin. You can use any point of the line to perform the translation. If the line is given by two endpoints, choose one of them. The first transformation matrix postulates pure translation - T_1 ;

2. Rotate the line and the object about the origin until the line is coincident with one of the coordinate axes. The angle of rotation depends on the slope of the mirror line and whether you decide to rotate towards x or y coordinate axes. Use the simple rotational matrix as discussed in section 8.1.4 - T_2 ;

3. Reflect through that coordinate axis - T_3 ;

4. Apply the inverse rotation about the origin - T_4 ;

5. Translate back to the original location - T_5 ;

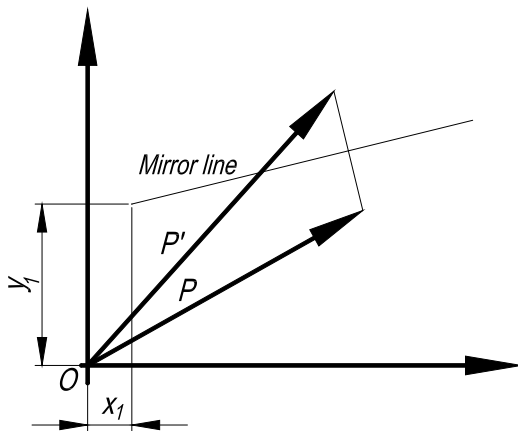


Figure 8.4

In matrix notation the resulting concatenated matrix is:

$$P' = T_5.T_4.T_3.T_2.T_1$$

If the mirror line is represented by two points $P_1 [x_1 \ y_1 \ 1]^T$ and $P_2 [x_2 \ y_2 \ 1]^T$, the rotational angle will be determined by the slope of the line.

$$k = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}; \quad \varphi = \text{atan2}(\Delta y, \Delta x)$$

Here the atan2 function uses two arguments with their signs to determine the angle in $0 \leq \theta < 2\pi$ interval.

$$P' = \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(-\phi) & -\sin(-\phi) & 0 \\ \sin(-\phi) & \cos(-\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The use of the above transformations presumes making the mirror line coincident with the x axis - the rotation is in clockwise direction, hence the negative angle. In this case the mirror transformation is performed through the x axis.

Example: Let perform a reflection of a straight line defined by the endpoints $A_1 = [4 \ 3 \ 1]^T$ and $A_2 = [5 \ 2 \ 1]^T$ through the mirror line defined by endpoints $C_1 [1 \ 2 \ 1]^T$ and $C_2 [4 \ 6 \ 1]^T$.

The MATLAB code to perform all the necessary calculations is given in the Table 8.2.

Table 8.2

```
% Transformation 2
%
% Reflection of a line through an
% arbitrary line in plane
%
% Enter the endpoints of the mirror line as 3 X 2 matrix
% Enter the endpoints of the transformed line as 3 X 2 matrix
C=[1 4; 2 6; 1 1];
A=[4 5; 3 2; 1 1];

% Define the slope of the line (angle of rotation
k1=C(2,2)-C(2,1);
k2=C(1,2)-C(1,1);
fi=atan2(k1,k2);

% Form the transformation matrixes
T1=[1 0 -C(1,1); 0 1 -C(2,1); 0 0 1];
T2=[cos(-fi) -sin(-fi) 0; sin(-fi) cos(-fi) 0; 0 0 1];
T3=[1 0 0; 0 -1 0; 0 0 1];
T4=[cos(fi) -sin(fi) 0; sin(fi) cos(fi) 0; 0 0 1];
T5=[1 0 C(1,1); 0 1 C(2,1); 0 0 1];
T=T5*T4*T3*T2*T1;
```

```

Aprim=T*A;
% The transformed line

% Results
Aprim =

1.1200   -0.1200
5.1600    5.8400
1.0000    1.0000

```

Use an available CAD system to perform the operation mirroring and to verify the results.

8.2 THREE DIMENSIONAL TRANSFORMATIONS

The ability to represent or display a three-dimensional object is fundamental to the understanding of the shape of that object. Furthermore, the ability to rotate, translate, and project views of that object is also, in many cases, fundamental to the understanding of its shape. This is easily demonstrated by picking up a relatively complex unfamiliar object. Immediately it is rotated held at arm's length, moved up and down, back and forth, etc., in order to view it properly. To do this with a computer we must extend our previous two-dimensional analysis to three dimensions. Based on our previous experience, we immediately introduce homogeneous coordinates. Hence, a transformation in three-dimensional space $(x \ y \ z)^T$ is represented by a four-dimensional position vector and 4×4 transformation matrix:

$$(x' \ y' \ z' \ h')^T = T \cdot (x \ y \ z \ h)^T$$

where T is some transformation matrix. Again, the transformation from homogeneous coordinates to ordinary coordinates is given by:

$$(x^* \ y^* \ z^*)^T = (x'/h' \ y'/h' \ z'/h')^T$$

The generalized 4×4 transformation matrix for three-dimensional homogeneous coordinates is:

$$T = \begin{pmatrix} a & b & c & l \\ d & e & f & m \\ g & i & j & n \\ p & q & r & s \end{pmatrix}$$

The above matrix could be partitioned into four separate sections:

$$\left(\begin{array}{c|c} & 3 \\ \hline 3 \times 1 & \times \\ & 1 \\ \hline 1 \times 3 & 1 \times 1 \end{array} \right)$$

The upper left 3×3 submatrix produces a linear transformation in the form of scaling, shearing, reflection and rotation. The 1×3 lower left submatrix produces a perspective transformation, and the upper right 3×1 submatrix produces translation. The final lower right-hand 1×1 submatrix produces overall scaling. The total transformation obtained after operating on a homogeneous position vector with this 4×4 matrix and obtaining the ordinary coordinate is called bilinear transformation. In general, this transformation yields combination of shearing, local scaling, rotation, reflection, translation, perspective and overall scaling.

8.2.1 THREE DIMENSIONAL SCALING AND SHEARING

The diagonal terms of the general 4×4 transformation produce local and overall scaling. To illustrate this, consider the transformation:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = (a.x \ e.y \ j.z \ 1)^T \text{ which shows the local scaling effect.}$$

Overall scaling is obtained by using the forth diagonal element.

$$\begin{pmatrix} x' \\ y' \\ z' \\ s \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

The ordinary or physical coordinates are $(x'/s \ y'/s \ z'/s \ 1)$ which is

equivalent to the transformation:

$$\begin{pmatrix} 1/s & 0 & 0 & 0 \\ 0 & 1/s & 0 & 0 \\ 0 & 0 & 1/s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The off-diagonal terms in the upper left 3x3 submatrix of the generalized 4 x 4 transformation matrix produce shear in three dimensions.

$$\begin{pmatrix} x' \\ y' \\ z' \\ s \end{pmatrix} = \begin{pmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ g & i & 1 & 0 \\ 0 & 0 & 0 & s \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + b.y + c.z \\ d.x + y + f.z \\ g.x + i.y + z \\ 1 \end{pmatrix}$$

8.2.2 THREE DIMENSIONAL ROTATION

Before considering three-dimensional rotation about an arbitrary axis, we examine rotation about each of the coordinate axes. For rotation about the x axis, the x coordinates of the position vectors do not change. In effect, the rotation occurs in planes perpendicular to the x axis. Similarly, rotation about the y and z axes occurs in planes perpendicular to the y and z axes, respectively. The transformation of the position vectors in each of these planes is governed by the general two-dimensional rotation matrix. Recalling that matrix, and again noting that for rotation about the x-axis the x coordinate of the transformed position vectors does not change, allows writing down the 4 x 4 homogeneous coordinate transformation by the angle α . Similarly we will have for the rotation about y and z axes by the angles β and γ respectively:

$$\text{Rot}(\vec{x}, \alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad \text{Rot}(\vec{y}, \beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{Rot}(\vec{z}, \gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation is assumed positive in a right-hand sense, i.e., clockwise as one looks outward from the origin in the positive direction along the rotation axis.

8.2.3 THREE DIMENSIONAL TRANSLATION

The three-dimensional translation matrix is written in the form bellow. The translated homogeneous coordinates are obtained in the form.

$$T = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & m \\ 0 & 0 & 1 & n \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad P' = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & m \\ 0 & 0 & 1 & n \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+1 \\ y+m \\ z+n \\ 1 \end{pmatrix}.$$

8.2.4 REVERSE TRANSFORMATIONS

Let consider one simple transformation – translation on distances respectively m and n . This translation will be described by a transformation matrix T_1 given in the previous section. If the reverse transformation – translation on $-m$ and $-n$ respectively is performed immediately afterwards, we do not expect any changes with the initial vector to occur.

Let denote the second (backwards) transformation by T_2 . In this case the combined transformation $T = T_2.T_1$ yields the identity matrix. Or T_2 is inverse matrix of T_1 , ($T_2 = T_1^{-1}$). These reflections could be extended to other simple transformations – rotation, mirroring etc. All this is useful in forming multiple transformations such as rotation about or mirroring through arbitrary points, lines or planes.

8.2.5 ROTATION ABOUT AN ARBITRARY AXIS IN SPACE

The general case of rotation about an arbitrary axis in space frequently occurs in geometric modeling, robotics, animation, and simulation. Following the previous discussion, rotation about an arbitrary axis in space is accomplished with a procedure using translations and simple rotations about the coordinate axes. Since the technique for rotation about a coordinate axis is known, the underlying procedural idea is to make the arbitrary rotation axis coincident with one of the coordinate axes.

Assume an arbitrary axis in space passing through the points $P_1 (x_1 \ y_1 \ z_1 \ 1)^T$ and $P_2 (x_2 \ y_2 \ z_2 \ 1)^T$. This line could be expressed by its parametric equation in the form:

$$\begin{aligned} x &= x_1 + (x_2 - x_1).t = x_1 + a.t \\ y &= y_1 + (y_2 - y_1).t = y_1 + b.t \\ z &= z_1 + (z_2 - z_1).t = z_1 + c.t \quad \text{where } t \text{ is a parameter.} \end{aligned}$$

Further we will denote: $a = x_2 - x_1$; $b = y_2 - y_1$; $c = z_2 - z_1$.

Rotation about this axis by some angle α is accomplished using the following procedure:

1. Make a line to pass through the origin of the coordinate system (Fig. 8.5 a,b). Translate so that the point $P_1 [x_1 \ y_1 \ z_1 \ 1]^T$ will lay at the origin of the coordinate system. The respective transformation matrix is:

$$T_1 = \begin{pmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2. Rotate about x axis while the rotational axis is placed in the xz plane (Fig. 8.5c). To determine the rotational angle we project the part of the line between the points P_1 and P_2 onto the yz plane. The coordinates of the endpoint P'_2 projected onto this plane are $(0 \ b \ c)$. Further we notice that the length of the projected part is $v = \sqrt{b^2 + c^2}$. Thus the desired angle will have its *sine* and *cosine* respectively:

$$\sin(\chi) = b/v \quad \text{and} \quad \cos(\chi) = c/v;$$

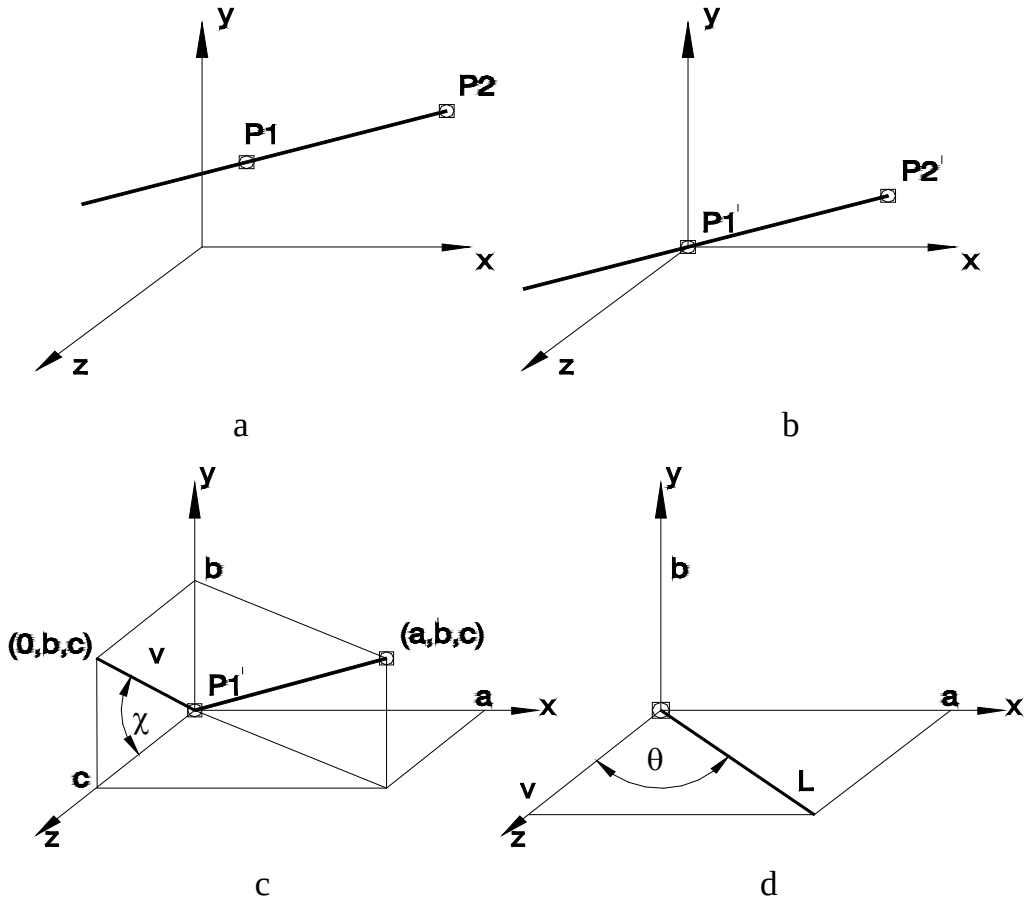


Figure 8.5

Note that the rotation is made in counterclockwise direction (looking from the positive direction of the x axis), so the rotational angle will be positive. The transformation matrix will be:

$$T_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c/v & -b/v & 0 \\ 0 & b/v & c/v & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Rotation about y axes while the rotational axis coincides with the z -axis. The rotational angle is θ as shown in the Fig. 8.5d. To determine this angle we first notice, that length of the line segment in xz plane after the previous rotation is equal to the length of the primary line segment between points P_1 and P_2 or we designate $L = \sqrt{a^2 + b^2 + c^2}$. Then the projections of the rotated segment onto x and z axes will be respectively equal to a and v . In this case we can find directly sine and cosine of the desired rotational angle as:

$$\sin(\theta) = a/L \quad \text{and} \quad \cos(\theta) = v/L.$$

Note that the rotation is made in clockwise direction and the rotational angle is negative. The transformation matrix will be:

$$T_3 = \begin{pmatrix} v/L & 0 & -a/L & 0 \\ 0 & 1 & 0 & 0 \\ a/L & 0 & v/L & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Since the rotational axes is made coincident with the z -axes we can rotate about the z -axis by desired angle α . The transformation matrix is known (T_4):

$$T_4 = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

5. Perform the inverse of the combined rotation - Transformation matrixes T_5 and T_6

$$T_5 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c/v & b/v & 0 \\ 0 & -b/v & c/v & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad T_6 = \begin{pmatrix} v/L & 0 & a/L & 0 \\ 0 & 1 & 0 & 0 \\ -a/L & 0 & v/L & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6. Perform the inversion of the translation $T_7 = \begin{pmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Finally the global transformation matrix for a rotation about an arbitrary axis in space, defined by two points will be obtained by a matrix multiplication of the defined above elementary transformation matrixes. We must pay the attention to the order of the multiplication.

$$T = T_7.T_6.T_5.T_4.T_3.T_2.T_1$$

Example: Let perform a rotation of a straight line given by the endpoints $A_1 = (0 \ 2 \ 3 \ 1)^T$ and $A_2 = (4 \ 1 \ 2 \ 1)^T$ around the axis defined by the endpoints $C_1 = (-1 \ -1 \ -1 \ 1)^T$ and $C_2 = (2 \ 3 \ 4 \ 1)^T$ by angle of 50° .

MATLAB code to perform the above example is shown in Table 8.3.

Table 8.3

```
% Transformation 3
%
% Rotation of a 3D line about an
% arbitrary line in space
%
% Enter the endpoints of the rotational line as 4 X 2 matrix,
% endpoints of the transformed line as 4 X 2 matrix
% Each column represents a point. Enter the angle of rotation.

C=[-1 2; -1 3; -1 4; 1 1];
A=[0 4; 2 1; 3 2; 1 1];
fi=50*pi/180;

% Compute the line of rotation elements
a=C(1,2)-C(1,1);
b=C(2,2)-C(2,1);
c=C(3,2)-C(3,1);

v=sqrt(b*b+c*c);
l=sqrt(a*a+b*b+c*c);

% Set the transformation matrixes
T1=[1 0 0 -C(1,1);0 1 0 -C(2,1); 0 0 1 -C(3,1); 0 0 0 1];
T2=[1 0 0 0; 0 c/v -b/v 0; 0 b/v c/v 0; 0 0 0 1];
T3=[v/l 0 -a/l 0; 0 1 0 0; a/l 0 v/l 0; 0 0 0 1];
T4=[cos(fi) -sin(fi) 0 0; sin(fi) cos(fi) 0 0; 0 0 1 0;...
    0 0 0 1];
```

```
T=inv(T1)*inv(T2)*inv(T3)*T4*T3*T2*T1;
Aprim=T*A;
```

```
Aprim =
```

```
0.5013    3.2451
1.1702    3.1049
3.3631    0.7691
1.0000    1.0000
```

8.2.6 REFLECTION THROUGH AN ARBITRARY PLANE

Some orientation of a three-dimensional objects could not be obtained through the pure rotation, they require a reflection. Unlikely the two-dimensional case, reflections in space are performed through a particular plane instead through a particular axes. Similarly to the two-dimensional transformation, the reflection through xy plane i.e. $z = 0$ plane changes only the sign of the z coordinate. Thus the transformation matrixes for reflection through $z = 0$, $x = 0$ and $y = 0$ planes will be respectively:

$$T_Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; T_X = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; T_Y = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

It is often necessary to reflect an object through a plane other than these. Again this can be accomplished using a procedure incorporating the previously defined simple transformations. One possible procedure is to make the normal vector of the mirror plane coincident with some of the coordinate axes. Then we can reflect through the coordinate plane, perpendicular to that axis. Perform the following steps:

1. Translate a known point P of the mirror plane to the origin of the coordinate system;
2. Rotate the normal vector to the reflection plane at the origin until it is coincident with the $+z$ axis. This makes the reflection plane the $z = 0$ coordinate plane (this procedure is identical with the problem for the rotation about an arbitrary axis); This step is identical to the case of rotation about the arbitrary axis in space as discussed in section 8.2.5.
3. After also applying the above transformations to the object, apply reflection through the $z = 0$ coordinate plane;

4. Perform the inverse transformations to those given above to achieve the desired result;

The vector normal to the plane could be determined by different ways according to the plane representation. If the plane is given in the form: $a.x + b.y + c.z + d = 0$, the normal vector direction angles will be obtained as:

$$\begin{bmatrix} \hat{n} \end{bmatrix} = [n_x \ n_y \ n_z]^T = \frac{[a \ b \ c]^T}{\sqrt{a^2 + b^2 + c^2}}$$

If the plane is defined by three points P_1 , P_2 and P_3 , the normal vector direction angles could be defined by forming vector cross product of vectors:

$$(P_2 - P_1) \times (P_3 - P_1) \quad \text{or} \quad \begin{bmatrix} \hat{n} \end{bmatrix} = \begin{pmatrix} P_{2x} - P_{1x} \\ P_{2y} - P_{1y} \\ P_{2z} - P_{1z} \end{pmatrix} \times \begin{pmatrix} P_{3x} - P_{1x} \\ P_{3y} - P_{1y} \\ P_{3z} - P_{1z} \end{pmatrix}$$

Example: Let perform reflection through a plane presented by three points $A_1 = (1 \ 1 \ 1 \ 1)^T$, $A_2 = (5 \ 2 \ 1 \ 1)^T$ and $A_3 = (4 \ 1 \ 2 \ 1)^T$ of a line given by two endpoints $C_1 = (2 \ 1 \ -1 \ 1)^T$ and $C_2 = (-3, 4 \ 2 \ 1)^T$.

For MATLAB code to perform the above example see Table 8.4

Table 8.4

```
% Transformation 4

% Performs a reflection through a plane defined by three points
% a1 = [ 1 1 1 1 ] , a2 = [5 2 1 1] and a3 = [ 4 1 2 1 ]
% Position vectors: C1 = [ 2 1 -1 1] and C2 = [-3, 4 2 1]
a1=[1 1 1 1]';
a2=[5 2 1 1]';
a3=[4 1 2 1]';

C=[2 -3; 1 4; -1 2; 1 1];

% Define the normal vector:
d1 = a2(1:3)-a1(1:3);
d2 = a3(1:3)-a1(1:3);
n=cross(d1,d2)

a=n(1);
b=n(2);
c=n(3);

v=sqrt(b*b+c*c);
L=sqrt(a*a+b*b+c*c);
```

```

T1=[1 0 0 -a1(1); 0 1 0 -a1(2); 0 0 1 -a1(3); 0 0 0 1];
T2=[1 0 0 0; 0 c/v -b/v 0; 0 b/v c/v 0; 0 0 0 1];
T3=[v/L 0 -a/L 0; 0 1 0 0; a/L 0 v/L 0; 0 0 0 1];
T4=[1 0 0 0; 0 1 0 0; 0 0 -1 0; 0 0 0 1];

```

```

T5=[v/L 0 a/L 0; 0 1 0 0; -a/L 0 v/L 0; 0 0 0 1];
T6=[1 0 0 0; 0 c/v b/v 0; 0 -b/v c/v 0; 0 0 0 1];
T7=[1 0 0 a1(1); 0 1 0 a1(2); 0 0 1 a1(3); 0 0 0 1];

```

```

T=T7*T6*T5*T4*T3*T2*T1;
Cprim = T*C

```

Cprim =

```

1.4615    -1.5385
3.1538    -1.8462
0.6154    -2.3846
1.0000     1.0000

```

8.3 PARALLEL PROJECTION

While we create three-dimensional on a computer we must have means to represent (to view) these objects on the flat - two-dimensional screen of the computer display. To accomplish this we project the three-dimensional bodies onto the plane of the computer display.

Plane geometric projections of objects are forming by intersection of lines called **projectors** with a plane called **projection plane**. Projectors are lines from an arbitrary point called **center of projection**, through each point of the object. If the centre of the projection is located at a finite point in space, the result is perspective projection. If the centre of the projection is located at infinity, all the projectors are parallel and the result is parallel projection.

The view of the three-dimensional object onto projecting plane depends on the view point or the position of the observer. So, before starting the projection activities we must define the viewing point or projection point.

In projecting a three dimensional object two alternate approaches could be used. The first assumes that the center of projection or eye point is fixed. The object is manipulated to obtain any required view. The second assumes that the object is fixed, that the center of projection is free to move anywhere in three space. Both approaches are mathematically equivalent.

By analogy the first approach is similar to the actions of a human observer when asked to describe a small object, e.g., a book. The object is picked up, rotated and translated in order to view all sides and aspects of the object. The center of projection is fixed and the object is manipulated. The second approach is similar to the actions of the human observer when asked to describe a large object, e.g., an automobile. The observer walks around the object to view

the various sides, climbs up on a ladder to view the top, and kneels down to look at the bottom. Here the object is fixed, and the center of projection and eye point are moved.

When designing or viewing an object on a computer graphics display the location of the eye is typically fixed and the plane of projection i.e., the face of the CRT, is typically perpendicular to the viewing direction. Hence, the first approach is generally more appropriate. However, if the graphics display is used to simulate the motion of a vehicle or of an observer moving through a computer generated model, then the second approach is more appropriate.

In this course the second approach will be used.

8.3.1 VIEW POINT AND VIEW TRANSFORMATION

If we want to view a three-dimensional object from different positions - from above, below, to obtain side view or other views we must first define the view point of the observer. We will use the camera analogy, i.e. the object is fixed, while a camera is placed at the viewing point. The two-dimensional representation of a object is the same as taken by this camera. In this case the camera film will represent the projection plane. We can move the camera to the every point, pointing the camera at the different points of the object as well as

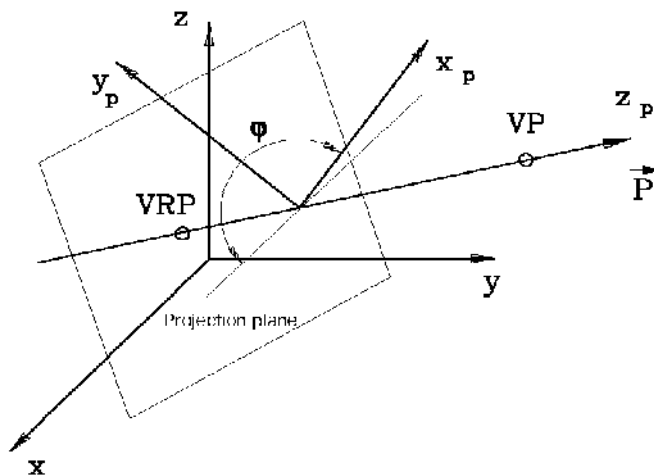


Figure 8.6

to the object. If we have a cord attached to the hook, the other end of this cord will be connected to the camera. We can always move the camera around the object always pointing towards the hook. We can rotate the camera around the cord. We can also change the view reference point.

The view vector is defined by these two points and is directed from the view reference point to the view point as shown on Fig. 8.6.

Direction of this hypothetical cord is given also by the normal vector of the view plane (the plane of the camera film or computer screen). There is another parameter under consideration - the rotational angle around the normal

rotate the camera about the optical axis.

We define the viewing direction by two points - the point where the camera is located called the **view point** and the point at which the camera is pointed called the **view reference point**. We can think of the view reference point as a point where a hook is attached

vector - " which gives the slope of the image - vertical, inclined, even upside-down.

The user creates a model using a World coordinate system. Three-dimensional visualization however means transformation of the coordinates of each point of the model into the coordinate system of the view plane. This is transformation of the coordinates from one coordinate system into another, a process which already has been solved. In this case the overall aim of the transformation will be to make the view vector coincident with the z axis of the world or model coordinate system which problem has been already solved in section 8.2.

Let designate the coordinates of the view reference point as:

$(x_{vrp} \ y_{vrp} \ z_{vrp} \ 1)^T$ and the coordinates of the view point as $(x_{vp} \ y_{vp} \ z_{vp} \ 1)^T$. In this case we can obtain the direction of view plane normal vector as:

$$dx_n = x_{vp} - x_{vrp} ; \quad dy_n = y_{vp} - y_{vrp} ; \quad dz_n = z_{vp} - z_{vrp} .$$

We will perform the transformation of the coordinate system using the following steps:

1. Translate so that the view reference point is at the origin of the coordinate system;

2. Rotate about the x axis while the normal vector is placed in the xz plane. The rotational angle is determined as: $\cos i = dz_n/v$; $\sin i = dy_n/v$;

$$v = \sqrt{dy_n^2 + dz_n^2} ;$$

3. Rotate in counterclockwise direction about y axis while the normal vector coincides with the z axis. The rotational angle is: $\sin^- = dx_n/L$; $\cos^- = v/L$;

$$L = \sqrt{dx_n^2 + dy_n^2 + dz_n^2} ;$$

4. Rotate about the z axis in order to obtain the correct view (vertical placement of the vertical lines.

Finally we obtain the transformation matrix as:

$$W' = \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot W \quad \text{where}$$

$$W = \begin{pmatrix} v/L & 0 & -dx_n/L & 0 \\ 0 & 1 & 0 & 0 \\ dx_n/L & 0 & v/L & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & dz_n/v & -dy_n/v & 0 \\ 0 & dy_n/v & dz_n/v & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -x_{vrp} \\ 0 & 1 & 0 & -y_{vrp} \\ 0 & 0 & 1 & -y_{vrp} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Performing parallel projection we first transform the coordinates of a point to the new coordinate system and then use only x and y coordinates, because the z coordinate is lost during the projection.

If we want to receive normally oriented images (vertical lines straight up) we must define the rotational angle around the view axes (normal vector). This could be achieved by recalling that points which lay on the same vertical line in the world coordinate system will have the same x coordinate in the transformed system. To illustrate this let consider two points from the world coordinate system laying on the same vertical line: $P_1 = (x \ y \ z_1 \ 1)^T$ and $P_2 = (x \ y \ z_2 \ 1)^T$.

The view transformation will produce the following notation:

$$P_{P1} = \begin{pmatrix} x_{P1} \\ y_{P1} \\ z_{P1} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot W \cdot \begin{pmatrix} x \\ y \\ z_1 \\ 1 \end{pmatrix}$$

$$P_{P2} = \begin{pmatrix} x_{P2} \\ y_{P2} \\ z_{P2} \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot W \cdot \begin{pmatrix} x \\ y \\ z_2 \\ 1 \end{pmatrix}$$

The requirement $x_{P1} = x_{P2}$ (after performing the matrix multiplication) yields:

$$w_{13} \cdot \cos\phi - w_{23} \cdot \sin\phi = 0; \quad \phi = \text{atan2}(w_{13}, w_{23})$$

where w_{13} and w_{23} are respective elements from the view transformation matrix W ,

8.4 PERSPECTIVE TRANSFORMATION

When any of the first three elements of the forth row of the general 4×4 homogeneous matrix is nonzero, a perspective transformation results. In contrast to the parallel transformations, in perspective transformations parallel lines converge, object size is reduced with increasing distance from the center of the projection (Fig. 8.7).

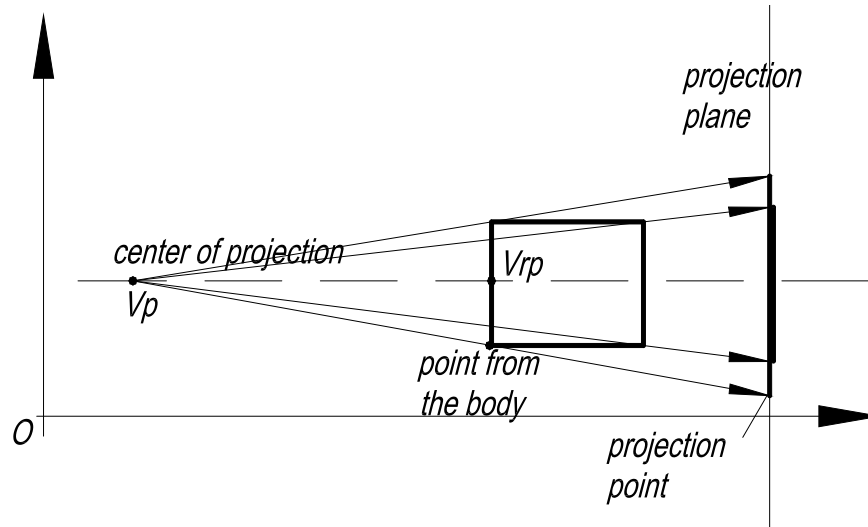


Figure 8.7

If the center of the projection has coordinates $[x_c \ y_c \ z_c \ 1]^T$, the projection line could be defined as a line passing through two points - the center of the projection and a point from the three-dimensional object: $[x_1 \ y_1 \ z_1 \ 1]^T$. The parametric equation of the projection line is:

$$x = x_1 + (x_c - x_1).t; \quad y = y_1 + (y_c - y_1).t; \quad z = z_1 + (z_c - z_1).t;$$

where t is a parameter $0 \leq t \leq 1$.

The projection point $[x_2 \ y_2 \ z_2 \ 1]^T$ is a point where the projection line crosses the projection plane i.e. $z = 0$. This could be obtained by substituting in the parametric equation:

$$t = -\frac{z_1}{z_c - z_1} \quad \text{and} \quad x_2 = x_1 - \frac{(x_c - x_1)}{(z_c - z_1)} \cdot z_1; \quad y_2 = y_1 - \frac{(y_c - y_1)}{(z_c - z_1)} \cdot z_1$$

$$x_2 = \frac{x_1 \cdot z_c - x_c \cdot z_1}{z_c - z_1}; \quad y_2 = \frac{y_1 \cdot z_c - y_c \cdot z_1}{z_c - z_1}$$

Which can be written in matrix form:

$$\begin{pmatrix} x_2' \\ y_2' \\ z_2' \\ w \end{pmatrix} = \begin{pmatrix} z_c & 0 & -x_c & 0 \\ 0 & z_c & -y_c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & z_c \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ w \end{pmatrix}, \text{ and}$$

the Euclid coordinates will be: $x_2 = x_2'/w$ and $y_2 = y_2'/w$.

The MATLAB code to perform the perspective transformation and view transformation of a cube with diagonal vertexes at $(2 \ 2 \ 2 \ 1)^T$ and $(4 \ 4 \ 4 \ 1)^T$ is given in Table 8.5.

Table 8.5

```

% Transformation 5

% View transformation and perspective transformation
% Perform the view and perspective transformations of a cube
% with view point at 7,8,9 and view reference point at
% 2, 2, 2
% The centre of projection is coincident with the view point
% Input data
    cube=[2 4 4 2 2 4 4 2;
          2 2 4 4 2 2 4 4;
          2 2 2 2 4 4 4 4;
          1 1 1 1 1 1 1 1];
    vrp=[2 2 2 1]';
    vp=[7 8 9 1]';

% Initial calculations
    a=vp(1)-vrp(1);
    b=vp(2)-vrp(2);
    c=vp(3)-vrp(3);
    v=sqrt(b*b+c*c);
    l=sqrt(a*a+b*b+c*c);

% Write the transformation matrixes
    T1=[1 0 0 -vrp(1); 0 1 0 -vrp(2); 0 0 1 -vrp(3); 0 0 0 1];
    T2=[1 0 0 0; 0 c/v -b/v 0; 0 b/v c/v 0; 0 0 0 1];
    T3=[v/l 0 -a/l 0; 0 1 0 0; a/l 0 v/l 0; 0 0 0 1];
    W=T3*T2*T1;

% Obtain the rotational angle
    fi=atan2(W(1,3),W(2,3));
    ROT=[cos(fi) -sin(fi) 0 0; sin(fi) cos(fi) 0 0; 0 0 1 0; 0 0 0 1];
    VT=ROT*W

% Form the perspective transformation matrix
    PT=[vp(3) 0 -vp(1) 0; 0 vp(3) -vp(2) 0; 0 0 0 0; 0 0 -1 vp(3)];

% First apply perspective transformation, then the view transformation
    cube_prim=VT*PT*cube

% Normalization
    for col=1:8
        cube_prim(1:4,col)=cube_prim(1:4,col)/cube_prim(4,col);
    end

```

cube_prim =

0.0000	-1.9754	-0.3292	1.6462	0.0000	-2.7656	-0.4609	2.3047
0.0000	-1.0987	-2.4172	-1.3184	2.6808	1.1427	-0.7032	0.8350
-2.9966	-1.7707	-0.2997	-1.5255	-5.9877	-4.2715	-2.2120	-3.9283
1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

8.5 COORDINATE SYSTEMS

In general, every kind of engineering activity takes information from the surrounding world and processes such information in order to produce new designs, technologies, materials and other applications. On the other hand, there is enormous amount of information – diverse and varying in extremely wide ranges – form micrometers of the tolerance analysis to the light years of the interstellar distances. Prior to any practical utilization such information must be measured and evaluated. Here comes the need of a particular, convenient measurement system in which particular data can be expressed. In order to compare different sets of data however, we also need a coordinate systems, according to which measures are taken.

In computer aided engineering applications it is paramount to have graphical output i.e. to express different kind of information (measured in different measurement systems) on computer display by means of computer graphics.

We can consider the information representing the ‘real world’ as associated to the **world coordinate system** which exists outside the graphical system. Such coordinate system can utilize different measurement units (millimeters, kilograms, volts etc.), and can be used in description of actual objects - design parts, installations or physical processes (to mention some). In order to express data graphically (even with conventional graphics used), the measurements taken in the world coordinate system and expressed in different units must be transformed in units suitable for graphical representation which are strictly restricted. For example, in conventional drawing diagrams on a sheet of paper, one can use the coordinate system with millimeters to represent quantities of different kind (consider changing of applied force measured in kilograms versus time measured in seconds).

It is obvious that any graphical representation is two dimensional case. In many cases however, such 2D viewing is used to show a variety of three dimensional object as well as functions of two independent variables. That leads to the necessity of the **projection coordinate system** as supplement to the world coordinate system (see section 8.3.1) The projection coordinate system is primarily used to draw 3D objects on a flat surface of paper or screen, but can also be used for graphical visualization of mathematical functions which visu-

alization can take form of complex surfaces. Different colors can also be used to add an additional, four dimension to the graphic.

Obtaining the coordinates in the projection coordinate system means to apply the view transformation (given a view vector) with subsequent parallel or perspective projection. Various projection coordinate systems can be utilized to produce various views of the same object or scene.

In computer-aided applications conventional sheet of paper is replaced by a computer screen, where positions are measured in pixels due to the raster nature of the display. The position of each pixel is determined by a pair of integer numbers. Thus we can introduce **screen coordinate system**. A typical screen coordinate system depends on the way pixels are addressed and usually has its origin at the upper left corner of the screen with the Y axis pointing downwards.

The screen coordinate system can be further divided, considering that one display can contain several images, formed in different areas of the screen and called **windows**. Every one of the windows can present different kind of information – text, pictures, graphics etc. In computer-aided design environment such windows are usually used to view 2D or 3D graphical objects and are referred as **viewports**. Different viewports are used to visualize various views of the same 3D geometrical object. In order to allow the program system to work efficiently with windows or viewports, a coordinate system is attached to each of them. This coordinate system is called **window (viewport) coordinate system**. Typically these windows are rectangular and their dimensions can be given by length and width in number of pixels. Windows are also positioned within the screen area by means of some base point or start pixel. In general, the origin of the window coordinate system is located at the lower left corner and the coordinates of the lower left and upper right pixels of the window determine its dimensions.

Usually the graphical systems provide the programmers with tools to process windows or such abilities are controlled by the operational system. Most of the high-level programming languages however have also means to form, place and scale views in viewports. The space defined by the viewport or screen coordinate systems is also referred as **image space**.

Different graphical systems (displays, raster printers or plotters) have different resolution i.e. different number of addressable points in horizontal or vertical directions. To make programs independent of the graphical output we must determine the coordinates of points in other units than pixels or dots. These units are usually called **normalized units** where every graphic output area – viewport or printer page has a height and width of 1. Thus the point (0,5 0,5) is located in the middle of the graphic area without regard to particular resolution.

An **area** can be defined as a rectangular region within the projection coordinate system and a viewport is a rectangular region of the normalized device coordinate space. Viewing transformation required to represent points on the

projection coordinate system to a normalized device is known as **area – to – viewport mapping**.

The process of viewing three dimensional models from the world coordinate system to viewport is presented as diagram on Figure 8.8.

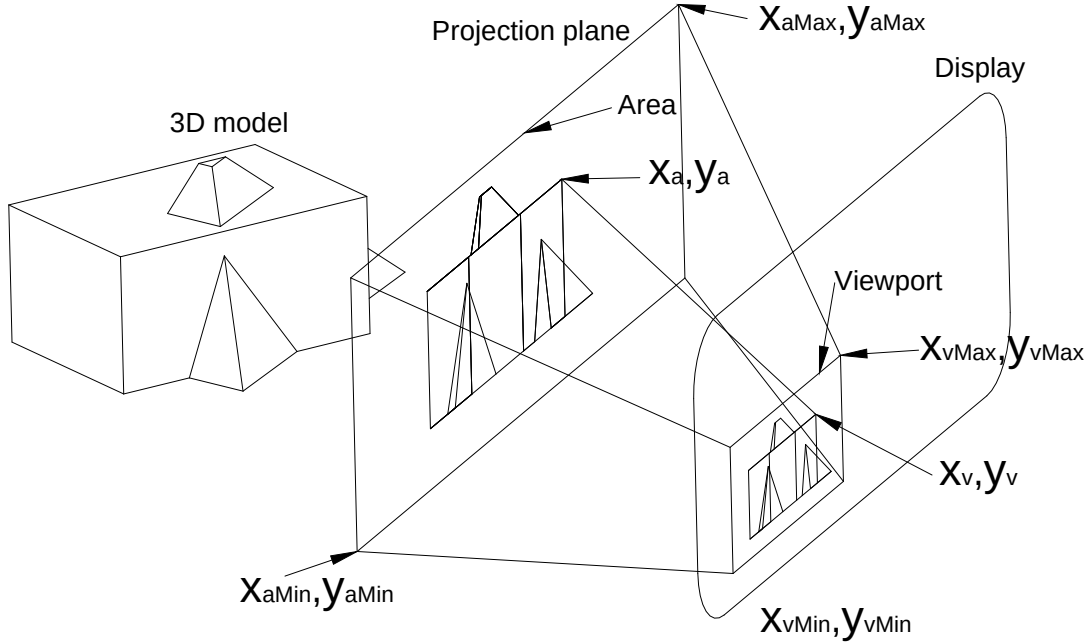


Figure 8.8

A point in the projection plane (x_a, y_a) will be transformed to a point in the viewport (x_v, y_v) using the following simple notation:

$$\frac{x_v - x_{vMin}}{x_{vMax} - x_{vMin}} = \frac{x_a - x_{aMin}}{x_{aMax} - x_{aMin}}$$

$$\frac{y_v - y_{vMin}}{y_{vMax} - y_{vMin}} = \frac{y_a - y_{aMin}}{y_{aMax} - y_{aMin}}$$

$$x_v = (x_a - x_{aMin}) \cdot \frac{x_{vMax} - x_{vMin}}{x_{aMax} - x_{aMin}} + x_{vMin}$$

$$y_v = (y_a - y_{aMin}) \cdot \frac{y_{vMax} - y_{vMin}}{y_{aMax} - y_{aMin}} + y_{vMin}$$

Expressions $S_x = \frac{x_{vMax} - x_{vMin}}{x_{aMax} - x_{aMin}}$ and $S_y = \frac{y_{vMax} - y_{vMin}}{y_{aMax} - y_{aMin}}$ are constants

and represent the scale factors in both directions. If $S_x \neq S_y$, distortion occurs in the picture. The concept of aspect ratio (AR) refers to this situation.

For a rectangular area or viewport the aspect ratio is given by the equation:

$$AR = \frac{x_{Max} - x_{Min}}{y_{Max} - y_{Min}}.$$

If AR for both area and viewport are the same, no distortion will occur in the picture. The above equations could be written in matrix form:

$$\begin{pmatrix} x_v \\ y_v \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & -x_{aMin}S_x + x_{vMin} \\ 0 & S_y & -y_{aMin}S_y + y_{vMin} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_a \\ y_a \\ 1 \end{pmatrix}$$

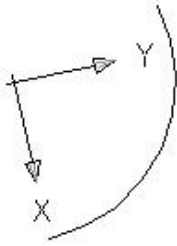


Figure 8.9

Object coordinate system. Every graphical object represented in the CAD database can be regarded as collection of points (vertexes), lines (edges), planes etc., combined in a way to form an integrated entity. Each object supports its own coordinate system called object coordinate system. When the object is moved or rotated it does no change in accordance to its own coordinate system while scaling and mirroring cause alteration. Figure 8.9 shows the object coordinate system for an arc. The origin is placed in the center of the arc, while X axis points towards the first point of the arc.

The object coordinate system is defined according to the world coordinate system by number of parameters (usually three positional and three rotational) to specify position and orientation. Each object then can be projected on view plane or represented on the screen or any other peripheral device through area-to-viewport mapping. Object coordinate systems of 3D solids are also utilized in assembly operations.

When a number of graphical objects are combined in order to make a 3D part (with solid modeling or parametric feature modeling) we can refer to the **part coordinate system**. Such coordinate system is attached to the solid by the system and its orientation usually depends on the way the particular part is built. The part coordinate systems are essential in the process of assembling parts (see section 6.4).

The following example demonstrates the principle of converting points in the World Coordinate System in order to be displayed on a raster screen with particular resolution. This conversion includes view transformation and area-to-viewport mapping.

Example:

Consider a scene consisting of a box and a pyramid with vertex coordinates in World coordinate system:

$$\text{box} = \begin{pmatrix} 0 & 45 & 45 & 0 & 0 & 45 & 45 & 0 \\ 0 & 0 & 30 & 30 & 0 & 0 & 30 & 30 \\ 0 & 0 & 0 & 0 & 22 & 22 & 22 & 22 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix};$$

$$\text{pyr} = \begin{pmatrix} 15 & 65 & 6 & 22,01 \\ -10 & 20 & 20 & 8,09 \\ 0 & 0 & 0 & 32,72 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

- A. Project the objects on a view plane defined by view point $vp = (2, -2, 1)^T$ and view reference point $vrp = (0, 0, 0)^T$;
- B. Perform the area to viewport transformation where the area is given by coordinates in the view plane as:

$$x_{aMin} = -10; x_{aMax} = 75; y_{aMin} = -12; y_{aMax} = 40.$$

The viewport has dimensions of 60 x 38 pixels with starting pixel at 0,0.

1. First compute the elements of the view transformation (see section 8.3.1):

$$\begin{aligned} dx_n &= x_{vp} - x_{vrp} = 2 - 0 = 2; \\ dy_n &= y_{vp} - y_{vrp} = -2 - 0 = -2; \\ dz_n &= z_{vp} - z_{vrp} = 1 - 0 = 1 \\ v &= \sqrt{dy_n^2 + dz_n^2} = \sqrt{-2^2 + 1^2} = 2,235; \\ L &= \sqrt{dx_n^2 + dy_n^2 + dz_n^2} = \sqrt{2^2 + (-2)^2 + 1^2} = 3 \end{aligned}$$

2. Form the view transformation matrix:

$$W = \begin{pmatrix} 2,235/3 & 0 & -2/3 & 0 \\ 0 & 1 & 0 & 0 \\ 2/3 & 0 & 2,235/3 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2,235 & 2/2,235 & 0 \\ 0 & -2/2,235 & 1/2,235 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} 0,745 & 0,596 & -0,298 & 0 \\ 0 & 0,447 & 0,894 & 0 \\ 0,667 & -0,667 & 0,333 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Define the angle of rotation and the view transformation:

$$\varphi = \text{atan2}(w_{13} \ w_{23}) = \text{atan2}(-0,298 \ 0,894) = -18,43^\circ.$$

View transformation matrix will take the form:

$$VT = \begin{pmatrix} \cos(-18.43) & -\sin(-18.43) & 0 & 0 \\ \sin(-18.43) & \cos(-18.43) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot W = \begin{pmatrix} 0,707 & 0,707 & 0 & 0 \\ -0,236 & 0,236 & 0,942 & 0 \\ 0,667 & -0,667 & 0,333 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4. Transforming the coordinates of the box and pyramid into the view coordinate system yields:

$$\text{box_v} = \text{box.VT} = \begin{pmatrix} 0 & 31.82 & 53.03 & 21.21 & 0 & 31.82 & 53.03 & 21.21 \\ 0 & -1061 & -3.54 & 7.07 & 20.74 & 10.13 & 17.21 & 27.81 \\ 0 & 30 & 10 & -20 & 7.33 & 37.33 & 17.33 & -12.66 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$\text{pyr_v} = \text{pyr.VT} = \begin{pmatrix} 3.54 & 60.10 & 18.38 & 21.28 \\ -5.89 & -10.61 & 3.30 & 27.57 \\ 16.66 & 30 & -9.33 & 20.19 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

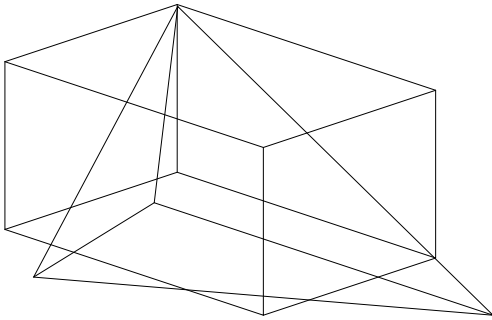


Figure 8.10

Figure 8.10 represents the two objects built by means of the vertex coordinates in the view plane.

The Z coordinate of the object vertexes in the view coordinate system could be used for other purposes – hidden lines removal, perspective projection etc., but they must be dropped out with the parallel projection in the present case.

5. Compute the aspect ratio and scaling factors:

$$AR_a = \frac{x_{aMax} - x_{aMin}}{y_{aMax} - y_{aMin}} = \frac{75 + 10}{40 + 12} = 1,6346; \quad AR_v = \frac{x_{vMax} - x_{vMin}}{y_{vMax} - y_{vMin}} = \frac{60 - 0}{37 - 0} = 1,621$$

$$S_x = \frac{x_{vMax} - x_{vMin}}{x_{aMax} - x_{aMin}} = \frac{60 - 0}{75 + 10} = 0,706; \quad S_y = \frac{y_{vMax} - y_{vMin}}{y_{aMax} - y_{aMin}} = \frac{37 - 0}{40 + 12} = 0,7115$$

No visible distortion is expected.

6. Apply the area to viewport transformation in matrix form:

$$\text{box_vp} = \begin{pmatrix} 0,706 & 0 & 10. & 0,706+0 \\ 0 & 0,7115 & 12. & 0,7115+0 \\ 0 & 0 & & 1 \end{pmatrix} \cdot \text{box_v}$$

$$\text{pyr_vp} = \begin{pmatrix} 0,706 & 0 & 10. & 0,706+0 \\ 0 & 0,7115 & 12. & 0,7115+0 \\ 0 & 0 & & 1 \end{pmatrix} \cdot \text{pyr_v}$$

7. Performing the matrix multiplication and rounding to the nearest integer finally yields:

$$\text{box_vp} = \begin{pmatrix} 7 & 30 & 44 & 22 & 7 & 30 & 44 & 22 \\ 9 & 1 & 6 & 14 & 23 & 16 & 21 & 28 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}; \quad \text{pyr_vp} = \begin{pmatrix} 10 & 49 & 20 & 22 \\ 4 & 1 & 11 & 28 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

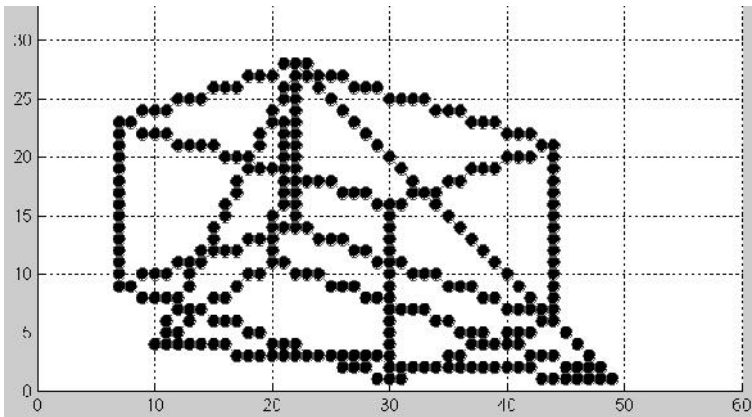


Figure 8.11 shows the representations of the box and the pyramid on the 60 x 37 pixel viewport. The corresponding MATLAB script is given in Example 7 listing in the Appendix A. A Bresenham's algorithm is used to draw the edges.

Figure 8.11