MODUL 5

KOMPUTER GRAFIK 2D
TRANSFORMASI 2D

D3/D4 TEKNIK INFORMATIKA
JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
POLITEKNIK NEGERI BANDUNG

MENTARI 047 | KOMPUTER GRAFIK | FEBRUARI, 22 2023

CONTENTS

KONSEP TRANSFORMASI

With the procedures for displaying output primitives and their attributes, we can create a variety of pictures and graphs. In many applications, there is also a need for altering or manipulating displays. Design applications and facility layouts are created by arranging the orientations and sizes of the component parts of the scene. And animations are produced by moving the "camera" or the objects in a scene along animation paths. Changes in orientation, size, and shape are accomplished with **geometric transformations** that alter the coordinate descriptions of objects. The basic geometric transformations are translation, rotation, and scaling. Other transformations that are often applied to objects include reflection and shear. We first discuss methods for performing geometric transformations and then consider how transformation functions can be incorporated into graphics packages.

TRANSLASI

Translation

A **translation** is applied to an object by repositioning it along a straight-line path from one coordinate location to another. We translate a two-dimensional point by adding **translation distances**, t_x and t_y, to the original coordinate position (x, y) to move the point to a new position (x', y') (Fig. 5-1).

$$x' = x + t_x, \qquad y' = y + t_y \tag{5-1}$$

The translation distance pair (t_x, t_y) is called a **translation vector** or **shift vector**.

We can express the translation equations 5-1 as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

$$\mathbf{P} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \mathbf{P}' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5-2}$$

This allows us to write the two-dimensional translation equations in the matrix form:

$$\mathbf{P}' = \mathbf{P} + \mathbf{T} \tag{5-3}$$

Sometimes matrix-transformation equations are expressed in terms of coordinate row vectors instead of column vectors. In this case, we would write the matrix representations as $\mathbf{P} = [x \; y]$ and $\mathbf{T} = [t_x \; t_y]$. Since the column-vector representation for a point is standard mathematical notation, and since many graphics packages, for example, GKS and PHIGS, also use the column-vector representation, we will follow this convention.

Translation is a *rigid-body transformation* that moves objects without defor-

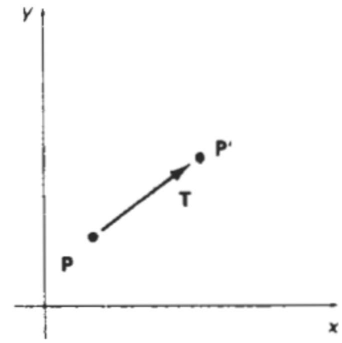

Figure 5-1
Translating a point from position **P** to position **P'** with translation vector **T**.

Similar methods are used to translate curved objects. To change the position of a circle or ellipse, we translate the center coordinates and redraw the figure in the new location. We translate other curves (for example, splines) by displacing the coordinate positions defining the objects, then we reconstruct the curve paths using the translated coordinate points.

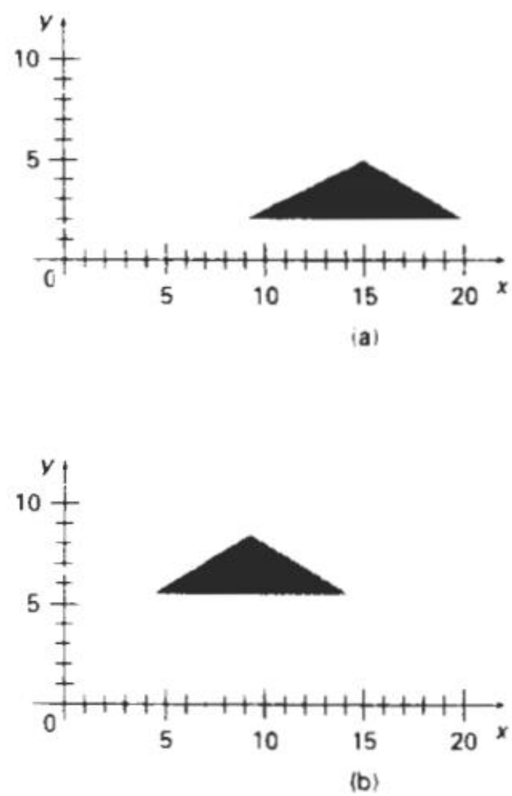

Figure 5-2
Moving a polygon from position (a) to position (b) with the translation vector (-5.50, 3.75).

SCALING

A **scaling** transformation alters the size of an object. This operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by **scaling factors** s_x and s_y to produce the transformed coordinates (x', y'):

$$x' = x \cdot s_x, \qquad y' = y \cdot s_y \tag{5-10}$$

Scaling factor s_x scales objects in the x direction, while s_y scales in the y direction. The transformation equations 5-10 can also be written in the matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{5-11}$$

or

$$\mathbf{P'} = \mathbf{S} \cdot \mathbf{P} \tag{5-12}$$

where **S** is the 2 by 2 scaling matrix in Eq. 5-11.

Any positive numeric values can be assigned to the scaling factors s_x and s_y. Values less than 1 reduce the size of objects; values greater than 1 produce an enlargement. Specifying a value of 1 for both s_x and s_y leaves the size of objects unchanged. When s_x and s_y are assigned the same value, a **uniform scaling** is produced that maintains relative object proportions. Unequal values for s_x and s_y result in a **differential scaling** that is often used in design applications, where pictures are constructed from a few basic shapes that can be adjusted by scaling and positioning transformations (Fig. 5-6).

Objects transformed with Eq. 5-11 are both scaled and repositioned. Scaling factors with values less than 1 move objects closer to the coordinate origin, while values greater than 1 move coordinate positions farther from the origin. Figure 5-7 illustrates scaling a line by assigning the value 0.5 to both s_x and s_y in Eq. 5-11. Both the line length and the distance from the origin are reduced by a factor of 1/2.

We can rewrite these scaling transformations to separate the multiplicative and additive terms:

$$x' = x \cdot s_x + x_f(1 - s_x)$$
$$y' = y \cdot s_y + y_f(1 - s_y) \qquad (5\text{-}14)$$

where the additive terms $x_f(1 - s_x)$ and $y_f(1 - s_y)$ are constant for all points in the object.

ROTASI

Rotation

A two-dimensional **rotation** is applied to an object by repositioning it along a circular path in the xy plane. To generate a rotation, we specify a **rotation angle** θ and the position (x_r, y_r) of the **rotation point** (or **pivot point**) about which the object is to be rotated (Fig. 5-3). Positive values for the rotation angle define counterclockwise rotations about the pivot point, as in Fig. 5-3, and negative values rotate objects in the clockwise direction. This transformation can also be described as a rotation about a **rotation axis** that is perpendicular to the xy plane and passes through the pivot point.

We first determine the transformation equations for rotation of a point position **P** when the pivot point is at the coordinate origin. The angular and coordinate relationships of the original and transformed point positions are shown in Fig. 5-4. In this figure, r is the constant distance of the point from the origin, angle ϕ is the original angular position of the point from the horizontal, and θ is the rotation angle. Using standard trigonometric identities, we can express the transformed coordinates in terms of angles θ and ϕ as

$$x' = r\cos(\phi + \theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi + \theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \qquad (5\text{-}4)$$

The original coordinates of the point in polar coordinates are

$$x = r\cos\phi, \qquad y = r\sin\phi \tag{5-5}$$

Substituting expressions 5-5 into 5-4, we obtain the transformation equations for rotating a point at position (x, y) through an angle θ about the origin:

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned} \tag{5-6}$$

With the column-vector representations 5-2 for coordinate positions, we can write the rotation equations in the matrix form:

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P} \tag{5-7}$$

where the rotation matrix is

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5-8}$$

When coordinate positions are represented as row vectors instead of column vectors, the matrix product in rotation equation 5-7 is transposed so that the transformed row coordinate vector $[x' \; y']$ is calculated as

$$\begin{aligned} \mathbf{P}'^T &= (\mathbf{R} \cdot \mathbf{P})^T \\ &= \mathbf{P}^T \cdot \mathbf{R}^T \end{aligned}$$

where $\mathbf{P}^T = [x \; y]$, and the transpose $\mathbf{R}^T$ of matrix $\mathbf{R}$ is obtained by interchanging rows and columns. For a rotation matrix, the transpose is obtained by simply changing the sign of the sine terms.

Rotation of a point about an arbitrary pivot position is illustrated in Fig. 5-5. Using the trigonometric relationships in this figure, we can generalize Eqs. 5-6 to obtain the transformation equations for rotation of a point about any specified rotation position (x_r, y_r):

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$
$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta \qquad (5\text{-}9)$$

These general rotation equations differ from Eqs. 5-6 by the inclusion of additive terms, as well as the multiplicative factors on the coordinate values. Thus, the matrix expression 5-7 could be modified to include pivot coordinates by matrix addition of a column vector whose elements contain the additive (translational) terms in Eqs. 5-9. There are better ways, however, to formulate such matrix equations, and we discuss in Section 5-2 a more consistent scheme for representing the transformation equations.

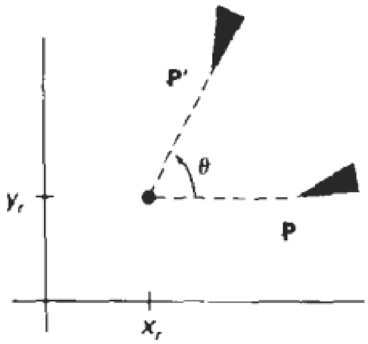

Figure 5-3
Rotation of an object through angle θ about the pivot point (x_r, y_r).

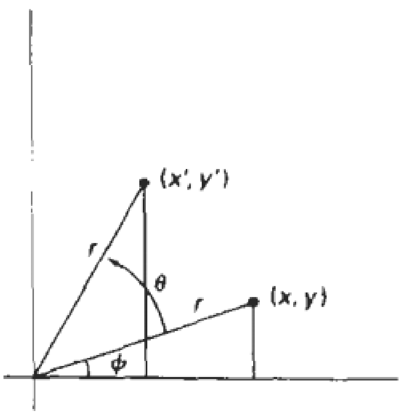

Figure 5-4
Rotation of a point from position (x, y) to position (x', y') through an angle θ relative to the coordinate origin. The original angular displacement of the point from the x axis is ϕ.

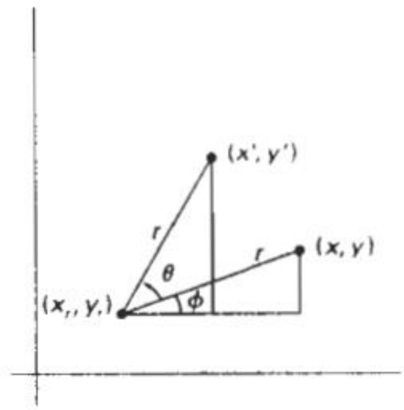

Figure 5-5
Rotating a point from position (x, y) to position (x', y') through an angle θ about rotation point (x_r, y_r).

HOMOGENEOUS MATRIKS

Many graphics applications involve sequences of geometric transformations. An animation, for example, might require an object to be translated and rotated at each increment of the motion. In design and picture construction applications, we perform translations, rotations, and scalings to fit the picture components into their proper positions. Here we consider how the matrix representations discussed in the previous sections can be reformulated so that such transformation sequences can be efficiently processed.

We have seen in Section 5-1 that each of the basic transformations can be expressed in the general matrix form

$$\mathbf{P}' = \mathbf{M}_1 \cdot \mathbf{P} + \mathbf{M}_2 \tag{5-15}$$

with coordinate positions $\mathbf{P}$ and $\mathbf{P}'$ represented as column vectors. Matrix $\mathbf{M}_1$ is a 2 by 2 array containing multiplicative factors, and $\mathbf{M}_2$ is a two-element column matrix containing translational terms. For translation, $\mathbf{M}_1$ is the identity matrix. For rotation or scaling, $\mathbf{M}_2$ contains the translational terms associated with the pivot point or scaling fixed point. To produce a sequence of transformations with these equations, such as scaling followed by rotation then translation, we must calculate the transformed coordinates one step at a time. First, coordinate positions are scaled, then these scaled coordinates are rotated, and finally the rotated coordinates are translated. A more efficient approach would be to combine the transformations so that the final coordinate positions are obtained directly from the initial coordinates, thereby eliminating the calculation of intermediate coordinate values. To be able to do this, we need to reformulate Eq. 5-15 to eliminate the matrix addition associated with the translation terms in $\mathbf{M}_2$.

We can combine the multiplicative and translational terms for two-dimensional geometric transformations into a single matrix representation by expanding the 2 by 2 matrix representations to 3 by 3 matrices. This allows us to express all transformation equations as matrix multiplications, providing that we also expand the matrix representations for coordinate positions. To express any two-dimensional transformation as a matrix multiplication, we represent each Cartesian coordinate position (x, y) with the **homogeneous coordinate** triple (x_h, y_h, h), where

$$x = \frac{x_h}{h}, \qquad y = \frac{y_h}{h} \tag{5-16}$$

Thus, a general homogeneous coordinate representation can also be written as $(h \cdot x, h \cdot y, h)$. For two-dimensional geometric transformations, we can choose the homogeneous parameter h to be any nonzero value. Thus, there is an infinite number of equivalent homogeneous representations for each coordinate point (x, y). A convenient choice is simply to set $h = 1$. Each two-dimensional position is then represented with homogeneous coordinates $(x, y, 1)$. Other values for parameter h are needed, for example, in matrix formulations of three-dimensional viewing transformations.

Thus, a general homogeneous coordinate representation can also be written as $(h \cdot x, h \cdot y, h)$. For two-dimensional geometric transformations, we can choose the homogeneous parameter h to be any nonzero value. Thus, there is an infinite number of equivalent homogeneous representations for each coordinate point (x, y). A convenient choice is simply to set $h = 1$. Each two-dimensional position is then represented with homogeneous coordinates $(x, y, 1)$. Other values for parameter h are needed, for example, in matrix formulations of three-dimensional viewing transformations.

The term *homogeneous coordinates* is used in mathematics to refer to the effect of this representation on Cartesian equations. When a Cartesian point (x, y) is converted to a homogeneous representation (x_h, y_h, h), equations containing x and y, such as $f(x, y) = 0$, become homogeneous equations in the three parameters x_h, y_h, and h. This just means that if each of the three parameters is replaced by any value v times that parameter, the value v can be factored out of the equations.

represented with three-element column vectors, and transformation operations are written as 3 by 3 matrices. For translation, we have

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-17}$$

which we can write in the abbreviated form

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P} \tag{5-18}$$

Similarly, rotation transformation equations about the coordinate origin are now written as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-19}$$

or as

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P} \tag{5-20}$$

Finally, a scaling transformation relative to the coordinate origin is now expressed as the matrix multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{5-21}$$

or

$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P} \tag{5-22}$$

```
#include <math.h>
#include "graphics.h"

typedef float Matrix3x3[3][3];
Matrix3x3 theMatrix;

void matrix3x3SetIdentity (Matrix3x3 m)
{
  int i,j;

  for (i=0; i<3; i++) for (j=0; j<3; j++) m[i][j] = (i == j);
}

/* Multiplies matrix a times b, putting result in b */
void matrix3x3PreMultiply (Matrix3x3 a, Matrix3x3 b)
{
  int r,c;
  Matrix3x3 tmp;

  for (r = 0; r < 3; r++)
    for (c = 0; c < 3; c++)
      tmp[r][c] =
        a[r][0]*b[0][c] + a[r][1]*b[1][c] + a[r][2]*b[2][c];

  for (r = 0; r < 3; r++)
    for (c = 0; c < 3; c++)
      b[r][c] = tmp[r][c];
}

void translate2 (int tx, int ty)
{
  Matrix3x3 m;

  matrix3x3SetIdentity (m);
  m[0][2] = tx;
  m[1][2] = ty;
  matrix3x3PreMultiply (m, theMatrix);
```

```
void scale2 (float sx, float sy, wcPt2 refpt)
{
  Matrix3x3 m:

  matrix3x3SetIdentity (m);
  m[0][0] = sx;
  m[0][2] = (1 - sx) * refpt.x;
  m[1][1] = sy;
  m[1][2] = (1 - sy) * refpt.y;
  matrix3x3PreMultiply (m, theMatrix);
}

void rotate2 (float a, wcPt2 refPt)
{
  Matrix3x3 m;

  matrix3x3SetIdentity (m);
  a = pToRadians (a);
  m[0][0] = cosf (a);
  m[0][1] = -sinf (a);
  m[0][2] = refPt.x * (1 - cosf (a)) + refPt.y * sinf (a);
  m[1][0] = sinf (a);
  m[1][1] = cosf (a);
  m[1][2] = refPt.y * (1 - cosf (a)) - refPt.x * sinf (a);
  matrix3x3PreMultiply (m, theMatrix);
}

void transformPoints2 (int npts, wcPt2 *pts)
{
  int k;
  float tmp;

  for (k = 0; k < npts; k++) {
    tmp = theMatrix[0][0] * pts[k].x + theMatrix[0][1] *
      pts[k].y + theMatrix[0][2];
    pts[k].y = theMatrix[1][0] * pts[k].x + theMatrix[1][1] *
      pts[k].y + theMatrix[1][2];
    pts[k].x = tmp;
  }
}
```

```
void main (int argc, char ** argv)
{
  wcPt2 pts[3] = { 50.0, 50.0, 150.0, 50.0, 100.0, 150.0};
  wcPt2 refPt = {100.0, 100.0};
  long windowID = openGraphics (*argv, 200, 350);

  setBackground (WHITE);
  setColor (BLUE);
  pFillArea (3, pts);
  matrix3x3SetIdentity (theMatrix);
  scale2 (0.5, 0.5, refPt);
  rotate2 (90.0, refPt);
  translate2 (0, 150);
  transformPoints2 (3, pts),
  pFillArea (3,pts);
  sleep (10);
  closeGraphics (windowID);
}
```

TASK 1: EKSPLORASI TRANSFORMASI

Tugas Task 1

1. Lanjutkan hasil praktikum modul 4
2. Persentasikan Bentuk Bentuk Dasar dengan berbagai Transformasi
3. Ubahlah code kelopak bunga menggunakan Transformasi
4. Ubahlah code pada quiz, komponen jarum jam menggunakan Rotasi

Lesson Learnt
Hal baru yang saya pelajari dalam materi serta tugas kali ini ialah tentang bagaimana kita melakukan perubahan suatu objek terhadap sumbu x dan y dengan menggunakan cara transformasi. Sebemnya yang baru saya tau hanyalah transformasi rotasi saja. Namun sekarang ini saya mendapat pengetahuan baru tentang cara dan ilmu transformasi yang lain seperti scalling dan juga transform. Awalnya saya cukup kesulitan ketika awal-awal mencoba, namun Alhamdulillah setelah beberapa kali dicoba, saya merasa sedikit lebih mudah, meskipun masih ada yang salah. Permisi pak, untuk quiz kemarin apabila diperbolehkan, saya membuat jam-jam yang baru untuk perbaikan dikarenakan pada quis kemarin saya rasa kode saya masih belum sempurna. Mohon maaf karena saya sering terlambat dalam mengumpulkan tugas, pak

TASK 2: KARYA 2D BEBAS (MEMANFAATKAN TRANSFORMASI)

Tugas Task 1

1. Buatlah Kelompok random yang terdiri dari 4 orang
2. Buatlah Karya 2D yang boleh sama dari pertemuan sebelumnya yang terdiri dari bentuk dasar yang bertransformasi
3. Gunakan line style
4. Buatlah Scene untuk mempersentasikan hasil manipulasi bentuk dasar dengan tema bebas, minimal ada 4 objek (kontribusi dari 4 orang masing-masing 1 objek)
5. Setiap objek dibuat menjadi fungsi parametrik, untuk memudahkan penduplikasian.

Lesson Learnt
Pada tugas objek kali ini, karena saya menggunakan objek yang telah saya buat pada pertemuan yang lalu, maka saya meresa cukup lancer dalam mengerjakannya. Apalagi kali ini menggunakan fungsi transformasi sehingga beberapa objek dapat dibuat dengan perulangan yang cukup singkat. Transformasi ini juga memudahkan dalam membuat objek yang memiliki dasar yang saya namun dengan ukuran atau tataletak yang berbeda. Namun, dalam ada perbedaan dalam fungsi transformasi yang saya dan teman kelompok saya gunakan. Selain itu, laptop saya juga cukup keberatan ketika menjalankan scene tugas kelompok ini, maka dari itu, beberapa objek teman saya tidak saya tampilkan. Dan untuk tampilan tugas kelompok ini saya buat dengan menampilkan objek saya (gamebox) dan objek Aini (kelinci) pada scene pertamanya. Hal tersebut karena awalnya banyak error yang muncul serta laptop saya menajdi not responding. Mohon maaf atas kekurangan saya dalam mengerjakan tugas ini, serta ketidak-tepatan waktu saya dalam mengumpulkannya. Terima kasih, pak

PENGUMPULAN

Ikuti Format yang diberikan di Google Classroom.