Rensselaer Polytechnic Institute
ECSE-4750 Computer Graphics
Class 4

Transformations

Design Considerations:

- We were discussing options of how to organize our graphics objects to best fit our design criteria:
  Compact
  Since we know graphics data tends to be very large, we need to design our structures to minimize computer memory requirements.
  Efficient
  Scene data needs to be accessed many times and we would like to retrieve and store data in our structures in constant time.
  Mappable
  Data representations need to efficiently map into graphics primitives. We must be able to easily convert external data into internal
  graphics data structures.
  Minimal Coverage
  We must create a data representation that minimizes the number of data types while covering a large portion of external data types.
  We wish to balance efficiency with the number of data types.
  Simple
  Simple algorithms and data sets are easier to implement and also easier to extend.
- We will be using a collection based system in which we will use the STL container classes for vectors.
- The CGWindow will collect Actors, Actors will collect shapes, and shapes will hold pointers to geometries.
- This all works if we are looking at a static scene, but of course nothing is static. We need geometry transformations to animate our scene.

Two-Dimensional Geometric Transformations

Basic Transformations:

- We will first look at the Cartesian coordinate transformations and then apply matrix formulas which can be expressed in a more convenient form that allows efficient combination of object transformations.

- **Translation**:
  - Translation is applied to an object by repositioning it along a straight line path from one coordinate to another.

    We translate a two-dimensional point by adding translation distances tx and ty to the original coordinate position (x,y) to move the point to a new position (x',y')

    $$(1) \qquad x' = x + tx \qquad\qquad y' = y + ty$$

  - The translation distance pair (tx,ty) is called a **translation vector** or **shift vector**.
  - We can express the translation equations 1 as a single matrix equation by using column vectors to represent coordinate positions and the translation vector:

    $$(2) \qquad P = \left| \begin{matrix} x \\ y \end{matrix} \right| \qquad P' = \left| \begin{matrix} x' \\ y' \end{matrix} \right| \qquad T = \left| \begin{matrix} tx \\ ty \end{matrix} \right|$$

  - This allows us to write the two-dimensional translation as:

    $$(3) \qquad P' = P + T$$

  - Translation is a *rigid-body transformation* that moves an object without deformation.
  - Every point on the object is translated by the same amount.
  - A Line segment is translated by applying the translation vector to each of the endpoints and redrawing the object at these new points.
  - A Polygon is translated by applying the translation vector to the coordinate of each vertex and then regenerating the polygon at these new points.
  - Similar methods can be used to translate circular objects. The control points for these objects are translated and then each object is redrawn using these new control points.

- **Rotation**:
  - Rotations are applied to an object by repositioning it along a circular path in the xy plane.
  - To generate a rotation we specify a **rotation angle *0*** and the position (xr,yr) of the **rotation point** (or **pivot point)** about which the object is to be rotated.
  - Positive values define counter clockwise rotations and negative values specify clockwise rotations.
  - We will first define rotation formula for rotations about the origin

- In this formula ***r*** is the distance from the origin to the point, **@** is the original rotation angle of the vector to the original point, and ***0*** is the desired rotational angle.

$$(4) \qquad \begin{aligned} x' &= r\cos(@+o) = r\cos(@)\cos(o) - r\sin(@)\sin(o) \\ y' &= r\sin(@+o) = r\cos(@)\sin(o) - r\sin(@)\cos(o) \end{aligned}$$

- The original coordinates of the points in polar coordinate are:

$$(5) \qquad x = r\cos(@) \qquad y = r\sin(@)$$

- Substituting expressions 5 into 4 we get an equation for rotating a point through an angle ***0*** about the origin:

$$(6) \qquad \begin{aligned} x' &= x\cos(o) - y\sin(o) \\ y' &= x\sin(o) + y\cos(o) \end{aligned}$$

- We can write these equations in a matrix form:

$$(7) \qquad P' = R \cdot P$$

- Where the rotation matrix is

$$(8) \qquad R = \begin{vmatrix} \cos(o) & -\sin(o) \\ \sin(o) & \cos(o) \end{vmatrix}$$

- Rotation about an arbitrary pivot position is

$$(9) \qquad \begin{aligned} x' &= xr + (x - xr)\cos(o) - (y - yr)\sin(o) \\ y' &= yr + (x - xr)\sin(o) + (y - yr)\cos(o) \end{aligned}$$

- This equation introduces the pivot point as a additive terms to the original points.
- We could add these terms to the transformation matrix by including the pivot point transformation. We will see in a moment a better way to handle this translation.
- As with Translations, Rotations are rigid-body transformations that move objects without deformation.

- **Scaling**:
  - This transformation alters the size of an object by multiplying the coordinate values of each vertex by **scaling factors** sx and sy.

$$(10) \qquad x' = x * sx \qquad y' = y * sy$$

- Scaling factor sx scales in the x direction while sy scales in the y direction.
- These equations can be written using matrix form:

$$(11)\qquad \begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} sx & 0 \\ 0 & sy \end{vmatrix} . \begin{vmatrix} x \\ y \end{vmatrix}$$

- or

$$(12)\qquad P' = S \cdot P$$

- Where S is the two by two scaling matrix in 11.
- Any positive scaling factors can be applied. Values less than one reduce the size of the object while values greater than one enlarge the object.
- Using a value of 1 leaves the object unchanged.
- When sx and sy are assigned the same value, a **uniform scaling** is produced keeping the objects proportions equal.
- Specifying different values for sx and sy results in **differential scaling**. This can be used in applications that specify a few basic building blocks that can be modified and transformed using scaling and transformations. This is an important factor in our design criteria concerning
- The scaling performed by equation 11 both scales and translates objects. Again because the scaling is performed about the coordinate origin.
- We can control where the scaling origin is by specifying a position, called the **fixed point,** that will remain unchanged after the scaling procedure.
- Similar to to the rotation equation, we will add the fixed point to equation 10:

$$(13)\qquad x' = xf + (x - xf)sx \qquad y' = yf + (y - yf)sx$$

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

$$(14)\qquad \begin{aligned} x' &= x \cdot sx + xf(1 - sx) \\ y' &= y \cdot sy + yf(1 - sy) \end{aligned}$$

- where the additive terms xf(1-sx) and yf(1-sy) are constant for all points in the object.
- Including coordinates for the fixed point int he scaling equation is similar o including coordinates for a pivot point in the rotations equations.
- We can setup a column vector whose elements are the constant terms in 14, then we add this column vector to the product **S·P** in 12.
- We of course would like to reduce this to matrix multiplication to reduce our work and combine all operations into matrix multiplication.
- Polygons and lines are scaled by applying transformations 14 to each vertex and then regenerating or drawing the objects.
- Other objects are scaled by applying the appropriate equations to the term defining the object. An ellipse is resized by scaling the semi major and semi minor axes and redrawing the object. Uniform scaling of a circle is done by adjusting the radius.

Matrix Representations and Homogeneous Coordinates:

- Animation and simulation systems require many changes to an objects appearance for every step of the animation.
- If there are many objects in your animation then you will have a great deal of work for each frame.
- First we look at the general matrix form of our transformations:

$$(15) \qquad P' = M1 \cdot P + M2$$

- where positions P and P' are represented as column vectors.
- Matrix M1 is a 2 by 2 array containing multiplicitive factors and M2 is a 2 element column matrix containing translational terms.
- For translation, Matrix M1 is the identity matrix.
- For rotation or scaling, M2 contains the translational terms associated with the pivot point or scaling fixed point.
- To reproduce a transformation, these equations have to be applied in a specific sequence to produce the desired effect.
- 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 scaled coordinates are rotated.
- A more efficient approach would be to combine transformations so that the final coordinate transformations are obtained directly from the initial coordinates.
- To do this we need to reformulate 15 to eliminate the matrix addition associated with the translational terms.
- 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 a 3 by 3 matrices.
- We can then express all transformations as matrix multiplication provided we expand the matrix representations for coordinate positions.
- The coordinate position (x,y) will be replaced with the homogeneous coordinate triple *(xh,yh,h)*, where:

$$(16) \qquad x = \frac{xh}{h} \qquad\qquad y = \frac{yh}{h}$$

- This representation can also be written as *(h.x,h.y,h)*.
- For two-dimensional geometric transformations we can choose any value for h thus giving an infinite number of representations for coordinate point *(x,y)*
- An obvious choice is to pick h=1 and our coordinates reduce to *(x,y,1)*.
- Using homogenous representations of coordinates allows us to represent equations containing x and y as homogeneous equations using x,y, and h.
- This simply 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.

- This frees us to reduce all transformations to matrix multiplication.
- Coordinates are represented as a three element column vector and transformation operations as 3 by 3 matrices.
- For translation we have:

```
(17)      | x' |   | 1  0  tx |   | x |
          | y' | = | 0  1  ty | . | y |
          | 1  |   | 0  0   1 |   | 1 |
```

- which can be expressed as:

```
(18)      P' = T(tx,ty)·P
```

- where T(tx,ty) is the 3 by 3 translation matrix in 17.The inverse is obtained by replacing tx and ty with -tx and -ty.
- The rotation transformation equations about the coordinate origin can be expressed

```
(19)      | x' |   | cos(@) -sin(@) 0 |   | x |
          | y' | = | sin(@)  cos(@) 0 | . | y |
          | 1  |   |  0         0   1 |   | 1 |
```

- or as

```
(20)      P' = R(@)·P
```

- where R(@) is the 3 by 3 translation matrix in 17. The inverse rotations is obtained by using -@
- Finally the scaling transformation relative to the coordinate origin is:

```
(21)      | x' |   | sx   0  0 |   | x |
          | y' | = | 0   sy  0 | . | y |
          | 1  |   | 0    0  1 |   | 1 |
```

- or

```
(22)      P' = S(sx,sy)·P
```

- where S(sx,sy) is the 3 by 3 matrix in 21. Replacind sx and sy with 1/sx and 1/sy produces the inverse scaling matrix.
- Matrix representations are standard methods for implementing transformaitons in graphics systems.
- Some APIs provide these functions which only produce transformations about the coordinate origin.

- We are then forced to translate our objects to the coordinate origin , sclale, rotate, and then transform back to the final position.
- Alternately we would like to specify our fixed and pivot point coordinates to our transforms to minimize operations.

Composite Transformations:

- Composite transformations allow us to combine succesive transforms by calculating the matrix product of the individual transformations.
- forming products of transformation matrices is often refered to as a **concatenation**, or **composition**, of matrices.
- For column matrix representaions of coordinate positions we form these composite transformations by multiplying matrices in order from right to left.
- Each successive transformation matrix premultiplies the product of the preceding transformaiton matrices.

- **Translations**:
    - If two successive translation vectors (tx1,ty1) and (tx2,ty2) are applied to position P, P' is calculated as:

$$(23)\qquad \begin{aligned} P' &= T(tx2,ty2).\{T(tx1,ty1).P\} \\ &= \{T(tx2,ty2).T(tx1,ty1)\}.P \end{aligned}$$

    - where P and P' homogenous coordinate column vectors.

- **Rotations**:
    - Two succesive rotations applied to point P produce the transformed position:

$$(26)\qquad \begin{aligned} P' &= R(@2)\cdot\{R(@1)\cdot P\} \\ &= \{R(@2)\cdot R(@1)\}\cdot P \end{aligned}$$

    - By multiplying the the two rotation matrices, we can verify that two succesive rotations are additive:

$$(27)\qquad R(@2)\cdot R(@1) = R(@1 + @2)$$

    - so the final rotated coordinates can can be calculated with the composite rotation matrix as

$$(28) \qquad P' = R(@1 + @2)\cdot P$$

- **Scalings:**
  - Concatenating transformation matrices for two succesive scaling operations produces the following:

$$(30) \qquad S(sx2,sy2)\cdot S(sx1,sy1) = S(sx1\cdot sx2,sy1\cdot sy2)$$

  - The resulting matrix indicates that succesive scaling operations are multiplicative.
  - That is, if we were to triple the size of an object twice in succesion, the final object would be nine times the original.

- **General Pivot-Point Rotation:**
  - Again some graphics packages only supply rotations about the coordinate origin.
  - We can generate rotations about pivot points by using the following sequence of steps:
    1. Translate the object so the pivot point is at the origin.
    2. Rotate the object about the origin
    3. Translate the object from the origin to the original pivot point.
  - This can be expressed as:

$$(32) \qquad T(xr,tr)\cdot R(@)\cdot T(-xr,-yr) = R(xr,yr,@)$$

- **General Fixed-Point Scaling:**
  - Similar to fixed point rotation, we can perform scaling about a fixed point by following these steps:
    1. Translate the object so the pivot point is at the origin.
    2. Scale the object about the origin
    3. Translate the object from the origin to the original pivot point.
  - Which produces the following sequence:

$$(34) \qquad T(xf,yf)\cdot S(sx,sy)\cdot T(-xf,-yf) = S(xf,yf,sx,sy)$$

- **Concatenation Properites:**
  - Matrix multiplication is associative.

$$(36) \qquad A\cdot B\cdot C = (A\cdot B)\cdot C = A\cdot (B\cdot C)$$

  - Conversely matrix products are not commutative. A.B is not equal to B.A, in general.
  - This means we need to be careful of the order with which the composite matrix is evaluated. Some transformations are commutative, for instance two successive rotations can be performed in either order. The same is true for scaling.

- Concatenating the matrices also reduces the total number of operations needed to transform positions.
- Standard matrix dot products requires nine multiplications and six additions. Our matrix representaion with homogenous coordinates reduces the transform equations to:

```
(39)    x' = x·rsxx + y·rsxy + trsx
        y' = x·rsyx + y·rsyy + trsy
```

- This requires four multiplications and four additions, greatly reducing our processing requirements.
- This is also the maximum number of computations required for any transformation sequence, once the individual matrices have been concatenated and the elements of the compoosite matrix are evaluated. (Kinda cool).
- An implementation would be to formulate transformational matrices, concatentate any transformation sequence, and calculate transformed coordinates using 39.