Lecture 03: Geometric Transformations (Chapter 7.9)

I. Operations applied to the geometric description of an object to change its size, position, or orientation are called geometric transformations.
   A. Goal: Learn geometric transformations as they are the backbone to computer model coordinates to world coordinates in the rendering pipeline.

II. 2D Transformations (Chapter 7)
   A. Basic Geometric Transformations
      1. Translation - add offsets to the coordinates of a point to generate a new point.
         \[(x', y') = (x + t_x, y + t_y)\]
         \[(t_x, t_y)\] is a translation vector.
         a. as matrices:
         \[P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} t_x \\ t_y \end{bmatrix}\]
         we will use \(P, P'\) throughout lecture.
         \[P' = P + \mathbf{T}\]
      2. To translate a polygon - apply translation to each vertex and then regenerate the polygon. (same for circle, ellipse, curves)
         \[\text{Ask how for these}\]

   C. A rigid-body transformation makes an object without deformation.
      1. Rotation - specified by rotation angle and rotation axis.
         "rotate point by angle about axis."
         a. in 2D the axis is perpendicular to the plane. Thus the rotation axis is considered a rotation or pivot point.
         b. Angle \(\theta\) is applied in a counterclockwise direction.
         c. Assuming pivot at origin, point is distance from origin at an angle \(\theta\) from the x-axis:
         \[x' = r \cos(\theta + \phi) = r \cos \phi \cos \theta - r \sin \phi \sin \theta \quad x = r \cos \phi \]
         \[y' = r \sin(\theta + \phi) = r \cos \phi \sin \theta + r \sin \phi \cos \theta \quad y = r \sin \phi \]
         \[x' = x \cos \phi - y \sin \phi \]
         \[y' = x \sin \phi + y \cos \phi\]
         or as a matrix
         \[P' = RP\]
         where \(R = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}\)
         about a general pivot \((x_n, y_n)\): \[x' = x_n + (x - x_n) \cos \theta - (y - y_n) \sin \theta \]
         \[y' = y_n + (x - x_n) \sin \theta + (y - y_n) \cos \theta\]
         Ask how to apply to polygons, circle, ellipse, curve.
III. Scaling by scale factors $(s_x, s_y)$

a. $(x', y') = (s_x x, s_y y)$ or as matrices

\[ P' = S P \quad \text{where} \quad S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \]

b. If $s_x = s_y$ it is considered uniform otherwise differential.

c. Negative values cause reflection about axes.

1. To scale about a point $(x_f, y_f)$

\[ x' = x_f + (x - x_f) s_x \]
\[ y' = y_f + (y - y_f) s_y \]

*Ask how for polygons, circle, etc.

B. Matrix representations and homogeneous coordinates

1. Considering sequences of transformations, we want to compute these efficiently.

We could compute intermediate points after each transform, but is very inefficient.

Instead combine transforms into a single transformation.

2. Homogeneous coordinates - motivated by converting translation to a matrix multiplication. They have a 0 1 coordinate representation where the last parameter is constant.

\[ (x, y) \to (x_h, y_h, h) = (lx, y, h) \quad h \text{ must be non-zero} \quad \text{for now we will always use } h = 1. \]

\[ P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \]

a. Translation:

\[ T(x_t, y_t) = \begin{bmatrix} 1 & tx \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad P' = T(x_t, y_t) P \]

b. Rotation about origin:

\[ R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad P' = R(\theta) P \]

c. Scaling about origin:

\[ S(s_x, s_y) = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \quad P' = S(s_x, s_y) P \]

D. Inverting transforms

a. Translation:

\[ T_{-x_t, -y_t} = T(-x_t, -y_t) \]

b. Rotation:

\[ R(\theta)^{-1} = R(-\theta) = R(\theta)^T \quad \text{why? R matrix is orthonormal?} \]

c. Scaling:

\[ S(s_x^{-1}, s_y^{-1}) = S(1/s_x, 1/s_y) \]
Composite transformations - Apply series of transforms as 1 matrix. (order affects results)

$P' \equiv (A_2A_1P)$ is less efficient than $P' \equiv (A_2P)A_1P$ because array order.

The same matrix is applied to an entire geometric object.

i. Two translations: $T(t_{x1}, t_{y1}) \cdot T(t_{x2}, t_{y2}) = T(t_{x1} + t_{x2}, t_{y1} + t_{y2})$

ii. Two Rotations: $R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$

iii. Two scales: $S(s_{x1}, s_{y1}) \cdot S(s_{x2}, s_{y2}) = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2})$

iv. General rotation about pivot: (pivoted about origin)

a. Algorithm: (1) Translate to origin, (2) Rotate about origin, (3) Translate back

b. Matrices: $R(x_0, y_0, \theta) = T(x_0, y_0) \cdot R(\theta) \cdot T(-x_0, -y_0)$

c. Ex.

v. General scaling about point

a. Algorithm: (1) Translate to origin, (2) Scale, (3) Translate back

b. Matrices: $S(x_0, y_0, s_x, s_y) \cdot T(-x_0, -y_0) = T(x_0, y_0) \cdot S(x, y, s_x, s_y) \cdot T(-x_0, -y_0)$

c. Ex.

vi. General scaling about point

a. Algorithm: (1) Scale, (2) Translate to origin, (3) Translate back

b. Matrices: $R_1(\theta)S(s_x, s_y)R(\theta)$

c. Ex.

vii. Composite transforms are associative but not always commutative.

viii. No rigid transform is rotation + translation.

$T(x_{20}, y_{20}) \cdot R(x_{10}, y_{10}, \theta) = T(x_{20}, y_{20}) \cdot R(x_{10}, y_{10}) \cdot T(-x_{10}, -y_{10})$

ix. Other transforms

i. Reflection

- about $x$: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- about $y$: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

ii. Shear

- arbitrary point: $\begin{bmatrix} 1 & k & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

ix. Applies to general reference frames: translate origin to origin, rotate axes to axes.

$R(-\theta) \cdot T(x_{10}, y_{10})$
### III. 30 Geometric Transformations (Chapter 9)

A. Uses homogeneous coordinates w/ 4 dimensions. Most transforms are analogous to 2D version. Rotation will be different.

#### B. Basic Geometric Transformations

i. **Translation**

\[
T(\mathbf{e}_1, \mathbf{e}_2) = \begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix} \\
T^{-1}(\mathbf{e}_1, \mathbf{e}_2) = T(-t_x, -t_y, t_z)
\]

ii. **Rotations**

a. **About coordinate axes**

1. **Z-axis**

\[
R_z(\theta) = \begin{bmatrix}
\cos\theta & -\sin\theta & 0 & 0 \\
\sin\theta & \cos\theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \\
R_z(\theta)^T = R_z(\theta)^T
\]

2. **X-axis**

\[
R_x(\theta) = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos\theta & -\sin\theta & 0 \\
0 & \sin\theta & \cos\theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \\
R_x(\theta)^{-1} = R_x(\theta)^T = R_x(-\theta)
\]

3. **Y-axis**

\[
R_y(\theta) = \begin{bmatrix}
\cos\theta & 0 & \sin\theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin\theta & 0 & \cos\theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \\
R_y(\theta)^{-1} = R_y(\theta)^T = R_y(-\theta)
\]

*Different because of perspective.*

4. Generally can combine. This representation is an "Euler-Angle" or fixed angle rotation. Arbitrarily define order of rotation, e.g., z, y, x, z. 

\[
R(x, y, z) = R_z(\theta) R_y(\beta) R_x(\gamma)
\]

5. **To remember for general rotation about fixed point:**

1) Translate to origin
2) Rotate object
3) Translate back

\[p' = T^{-1} R T p\]

6. **About arbitrary axis:** (\(a_n, 0, 0\))

1) Translate axis to origin
2) Rotate axis coincide w/ axis
3) Rotate
4) Invert rotation
5) Invert translation

\[p' = T^{-1} R^{-1} R(\theta) R T p\]

Form new single rotation matrix easy to determine from rotation axis.
b. **Using Quaternions.**

1. **Issue w/ Rotation Matrices** is that 4x4 matrix multiplication uses lots of storage, requires many multiplications, and are difficult to interpret rotations.

2. **Quaternions** are a “30” complex number that solves these issues.
   
   \[ q = (s, \mathbf{v}) \]
   
   s - scalar component
   
   \[ \mathbf{v} \] - vector component

   \[ q = s + \mathbf{v} \hat{i} + \mathbf{v} \hat{j} + \mathbf{v} \hat{k} \]

   in full form

3. **Angle-axis rotations** are easy to express: \((\mathbf{a}, \theta)\)

   \[ q = (s, \mathbf{v}) = \left( \cos \frac{\theta}{2}, \mathbf{v} \sin \frac{\theta}{2} \right) \]

   Point to rotate as quaternion: \(P = (0, \mathbf{p})\)

   so rotations are as follows:

   \[
   p' = qPq^{-1} \quad \text{where} \quad q = (s, \mathbf{v}) \quad \text{and} \quad p' = (0, \mathbf{p})
   \]

   \[
   p' = s \mathbf{p} + \mathbf{v} (\mathbf{p} \cdot \mathbf{v}) + \mathbf{v} \times (\mathbf{v} \times \mathbf{p})
   \]

   *Based on \( q, \mathbf{v} = (s, \mathbf{v}) = (s, \mathbf{v}) = \left( \begin{array}{cc}
   s^2 - \mathbf{v} \cdot \mathbf{v} & s \mathbf{v} \times \mathbf{v} \\
   s \mathbf{v} \times \mathbf{v} & s^2 - \mathbf{v} \cdot \mathbf{v}
   \end{array} \right) \)

   with \( \mathbf{v} \times (\mathbf{a} \times \mathbf{b}) = (\mathbf{v} \times \mathbf{a}) \times \mathbf{b} \) - anti-commutative property

   and \( \mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c} \) - distributive property

4. **Easily convert back to a matrix.**

   **Scaling**

   a. About origin:

   \[
   S(2, 3, 5, 2) = \begin{bmatrix}
   2 & 0 & 0 & 0 \\
   0 & 3 & 0 & 0 \\
   0 & 0 & 5 & 0 \\
   0 & 0 & 0 & 2
   \end{bmatrix}
   \]

   \[
   S(2, 3, 5, 2) = S(\frac{1}{2}, \frac{1}{3}, \frac{1}{5}, \frac{1}{2})
   \]

   b. **About fixed point**

   (1) Translate to origin (2) Scale (3) Translate back

   \( \mathbf{p} = T^{-1} \mathbf{S} \mathbf{T} \mathbf{p} \)

   C. **Composite Transformations** - Multiply matrices first then apply to whole object.

   D. **Other Transforms**

   i. Rotation - negative values in scaling matrix

   \[
   \text{Shear about origin: } H = \begin{bmatrix}
   1 & \text{hxy} & \text{hxy} & 0 \\
   \text{hxy} & 1 & \text{hxy} & 0 \\
   \text{hxy} & \text{hxy} & 1 & 0 \\
   0 & 0 & 0 & 1
   \end{bmatrix}
   \]

   E. **Can transform between coordinate systems.** Translate, rotate to align axes.

   F. **All transforms discussed are affine; parallel lines stay parallel, points stay as points.**