CS 174 Week 3

Aidan Jan

April 27, 2023

Polygons

A polygon is an ordered sequence of vertices.

- Simple vs. Complex
  - Issue with Complex: hard to deal with self-intersecting polygons
- Convex vs. Concave
  - Issue with Concave: Point-inside-polygon test will fail in some cases
    - **Point inside polygon test:** all the edges making up the polygon are directed. If the given point is to the **left** of all the directed edges, then it is inside. Otherwise, it is outside.
  - Issue with Concave: Normal will point in wrong direction at concave vertices
- Planar vs. Non-planar
  - Issue with Non-planar: Normal are different at different at different locations on the non-planar face
- Tessellation (any poly) vs. Triangulation (only triangles)

Why Triangles?

- Ordered list of 3 non-colinear vertices
- They are simple
- They are planar
- They are convex
- Modern GPUs: 100M triangles/sec
- Shading, lighting, texture mapping all in GPU
- WebGL uses triangles only for rendering

What do we do with our point lists?

- **Immediate mode rendering:** Send them one at a time to GPU to draw every point of every triangle. **Bad.**
- **Retained mode:** Store them all in a data structure
  - Now we can store one **matrix** to represent the whole shape's movement/positioning, instead of repeating ourselves on the point-by-point level
  - Can draw the same shape in many places this way
- **Better yet:** Load the data onto the GPU once and re-use it there
  - Fewer slow GPU communications

Matrices found in Graphics

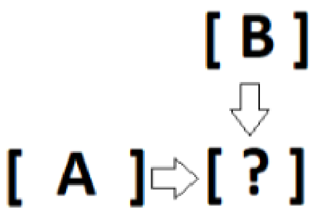
What does a matrix mean?

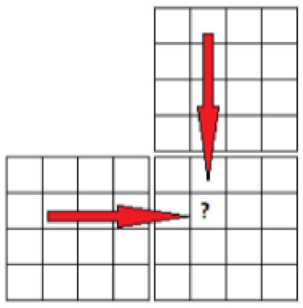
- A system of (first order) equations
- Our mathematical tool to tweak 3D functions or point clouds with
- Remember that a matrix is shorthand:

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \cdot \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} = \begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} \Rightarrow \begin{matrix} x_{new} = 2x_{old} + 3y_{old} \\ y_{new} = 4x_{old} + 5y_{old} \end{matrix}$$

Reminder: Matrix Multiplication

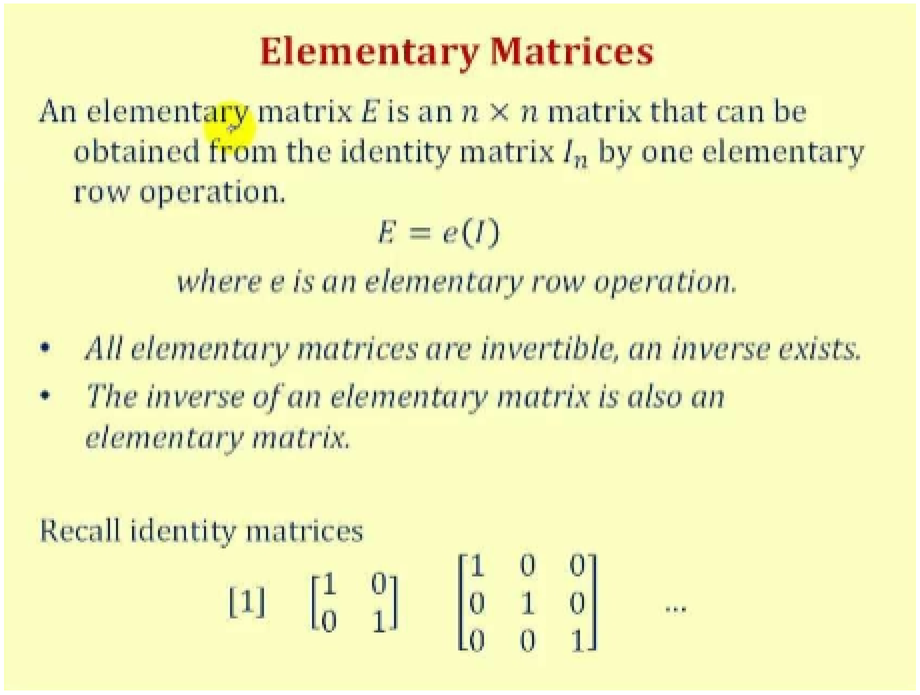
- $A \cdot B = M$
- Each cell of M is going to be a dot product of one row of A and one column of B
- Remember the rule:
  - Row size comes from A, column size comes from B
- If B is just a vector (1 column), that's fine, M will be too.
- It's helpful to line up A and B offset when you write them down





Matrix Concepts

- In graphics: Guiding a shape into place in a scene
- A few common graphics matrices perform movement, or ”warp” space
- What kind? Recall elementary row operations



Elementary Matrices

An elementary matrix E is an $n \times n$ matrix that can be obtained from the identity matrix I_n by one elementary row operation.

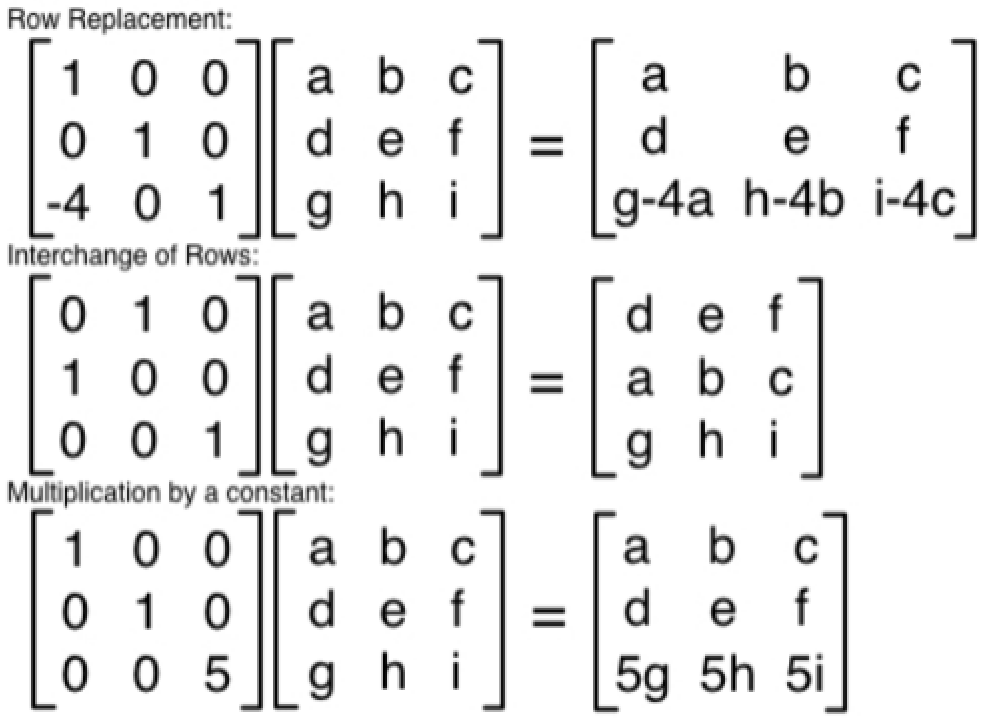
$$E = e(I)$$

where e is an elementary row operation.

- *All elementary matrices are invertible, an inverse exists.*
- *The inverse of an elementary matrix is also an elementary matrix.*

Recall identity matrices

$$[1] \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \dots$$



Row Replacement:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & 0 & 1 \end{bmatrix}\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g-4a & h-4b & i-4c \end{bmatrix}$$

Interchange of Rows:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} d & e & f \\ a & b & c \\ g & h & i \end{bmatrix}$$

Multiplication by a constant:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 5 \end{bmatrix}\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 5g & 5h & 5i \end{bmatrix}$$

Translation

- A translation is simply moving the point, polygon, or object on the grid.
- To do this, we add a position matrix.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$

- t_x is the amount to move in the x-direction and t_y is the amount to move in the y-direction.

Scaling

- Scaling refers to changing the size of a polygon, or the size of an object which may contain thousands of polygons
- To scale in the x-direction, you would multiply all the x-coordinates of the points by a constant S_x, and to scale in the y-direction, you would multiply all the y coordinates of the points by a constant S_y.
- Written in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

- A flip can also be done if the scaling factor is negative.
- If $S_x = S_y$, then the scaling in both directions are equal. This is known as **uniform scaling**.

Rotation

- Rotation is rotating a polygon or an object with many polygons around a given point in some direction.
- We use the variable θ to denote the direction and magnitude of rotation.
  - A positive θ would rotate in the counter-clockwise direction
  - A negative θ would rotate in the clockwise direction.

To rotate, a point, we can multiply it by the following matrix:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Shears

- **Shearing** is when you translate in a direction, but based on the coordinate of the point. Thus, the object would be deformed.
- A shear in the x-direction would not change the y-coordinate of any point. However, the higher the y-coordinate of the point being sheared, the more it moves in the x-direction. The opposite happens when shearing in the y-direction.
- A shear can be written as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & a \\ b & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

- A shear in the x-direction would have $b = 0$, and a shear in the y-direction would have $a = 0$.

Combining Operations

- Suppose we wanted to combine transformations, such a rotation followed by a shear, followed by a scaling. In this case, we can use all their matrices and multiply them, and only have to do one transformation in the end.
- However, what if we wanted to do a translation? Since translation is an addition of a matrix rather than a multiplication, this cannot be done.
- To do this, we would instead use a 3x3 matrix to represent the transformations. A translation of a matrix can be written as a product:

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

- For all the other matrices, same thing, except add a 1 at the bottom right of the 3x3 matrix and the original 2x2 matrix in the top left.
- Now, all the matrices can be multiplied.

Inverse Transformations

- Suppose you wanted to go back. What matrix operations would you do?
- To translation, simply move the object back by adding the inverse matrix.

$$T^{-1}(t_x, t_y) = T(-t_x, -t_y)$$

- To reverse a scaling, scale by the reciprocal scaling factor.
- To undo a rotation, rotate by $-\theta$.
- Finally, to undo a shear, shear in the opposite direction. ($-a$ or $-b$).

Affine Transformations in 3D

Translation

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shearing

$$\begin{bmatrix} 1 & 0 & SH_x & 0 \\ 0 & 1 & SH_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation Matrices in 3D

X Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Y Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Z Rotation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Affine Transformations

- Affine transformations preserve colinearity, planarity, and parallelism (parallel lines and planes stay parallel)
- Ratios of edge lengths are also kept constant.
- Transformations we have discussed, including translations, rotations, shears, and scalings, are all affine transformations.

Properties of Rigid Body Transformations

- Rigid body transformations are a **subset** of Affine transformations
- Additionally, Length, internal angles
- Translations and rotations are rigid body transformations
- Any combination of rigid body transformations is also a rigid body transformation.
  - If given a random matrix, if all the vectors are orthonormal, then the matrix can be composed of rigid body transformations, and is thus also a rigid body transformation.

Skipping Matrix Multiplication

- If you know you are doing one transformation after another (and not rendering in between), you can combine them using matrix multiplication. However, matrix multiplication is inefficient.
- In cases where you are doing many translations, one after another, you can simply add the translations in each direction. Adding matrices is much faster than multiplying matrices. This is possible because translations are commutative.
- In cases where you are doing many rotations over the same axis, one after another, you can simply add the rotation angles. Note that this does not work with rotations over different axis because then it would no longer be commutative.

Transforming Vectors

- How do we transform a vector?
- The reason is because vectors only have a direction and magnitude, not a location. In fact, if we try it...

$$\begin{bmatrix} 1 & 0 & 0 & t_X \\ 0 & 1 & 0 & t_Y \\ 0 & 0 & 1 & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_X \\ v_Y \\ v_Z \\ 0 \end{bmatrix} = \begin{bmatrix} v_X \\ v_Y \\ v_Z \\ 0 \end{bmatrix}$$

Point of Rotation

- When rotations are done, where do we determine the center of rotation? We can choose any point in space as the center of rotation, but it is often the center or corner of the object.
- To do a rotation around a point (x_p, y_p), we first move the entire shape so the point of rotation relative to the shape rests on the origin. Then, we do the rotation and translate it back. Thus, we would first do a translation $t(-x_p, -y_p)$, then the rotation, $R(\theta)$, then the translation to move the shape back $t(x_p, y_p)$.

Inverse of Rotations

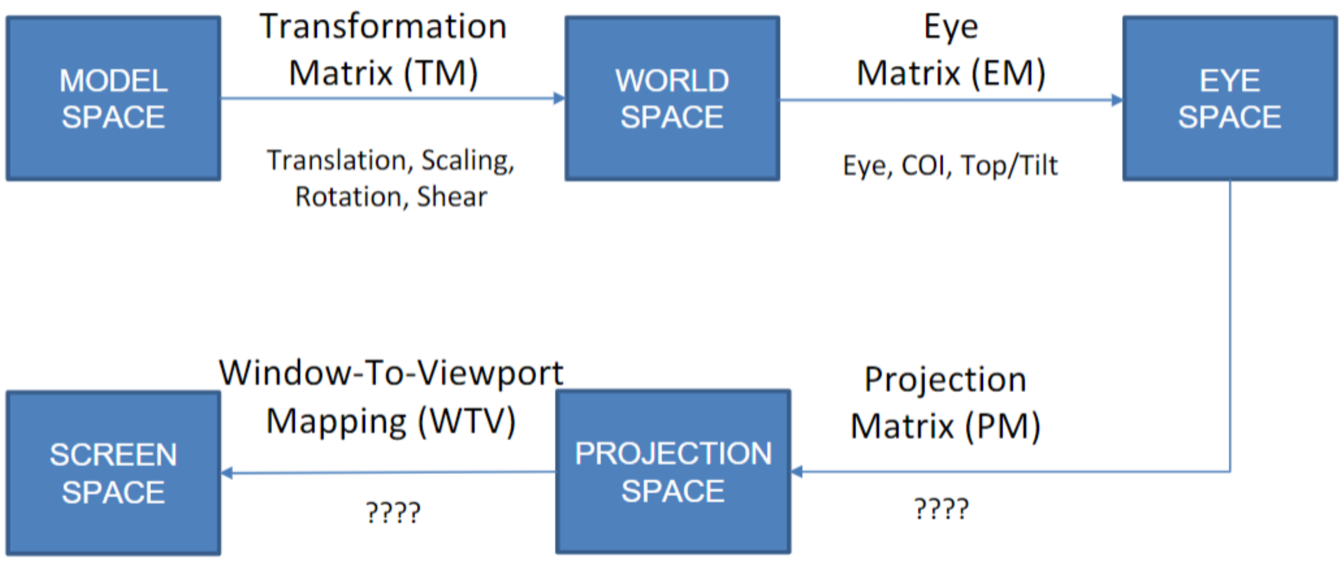
- **Note:** This only works for pure rotations, no scalings or shears.

$$M^{-1} = M^T$$

where M^T is the transpose of matrix M.

- This works because plugging $-\theta$ in for θ would just happen to produce the transpose. This is due to the property that sin and cos are even and odd functions, respectively.

Rendering Pipeline



- The Eye Space is also known as the Camera Space.