Lecture 8 | Transformation

CS230
Game
Implementation
Techniques

1. Transformation

1.1. Introduction

- Even if 2 or more sprites share the same image, texture or shape, they might:
  - Be located at different locations in the game.
  - Have different orientations.
  - Have different sizes.
- Each sprite must save its own position, usually in 2 variables: X & Y.
- Each sprite must also save its orientation value, usually a variable indicating the current angle of rotation: α
- Finally, it should have two scaling values: scaleX and scaleY.
- Scale, rotation and translation are capable of representing most transformations.
- Upon updating a sprite, we must calculate its transformation matrix.
- The transformation matrix transforms the sprites from its local coordinates system to the world coordinates system.
  - This transformation matrix should be a combination of the sprite scale, rotation and translation.
- When transforming a 2D image, the transformation matrix isn't applied to each point inside the image, but only to the 4 vertices (points) which represent the image's corners.

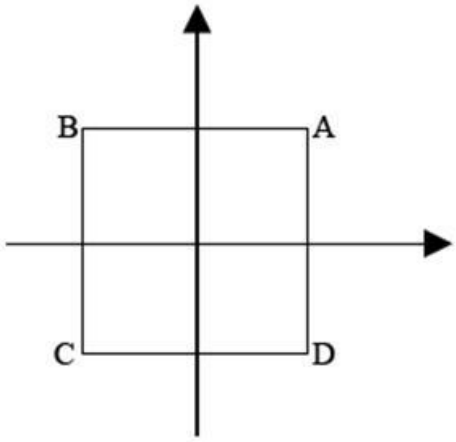
1.2. Scaling

- In games, you might need to make some sprites bigger or smaller in size.
- This is done by integrating a scaling factor into the sprite's transformation matrix.
- The scale on the X-axis can be different than the one of the Y-axis, but this will distort the image's original ratio.
  - That might be done on purpose
- The scale factors of a sprite are saved in 2 values scaleX and scaleY.
- scaleX and scaleY should be equal to 1 in case we do not want to scale the image, or in other words we want to display the image using its original size.
- To double the size of a sprite, scaleX and scaleY should be equal to 2.
- To reduce the size of a sprite by half, scaleX and scaleY should be equal to $^1/_2$
- In general, if you want to increase the sprite's size by a certain factor, set scaleX & scaleY to that particular number.
- On the other hand, if you want to decrease a sprite's size by a certain factor, set scaleX and scaleY to one over that particular number.

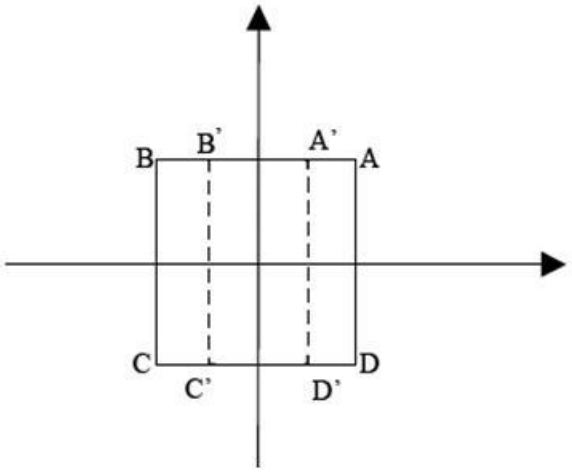
$$\begin{bmatrix} scaleX & 0 & 0 \\ 0 & scaleY & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

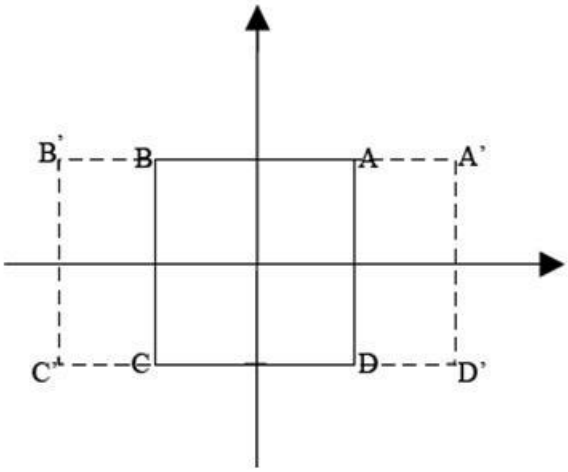
- scaleX and scaleY should contain the current sprite's scaling factors, and not the incremental ones.
- You should scale a sprite before rotating and translating it.

- Example:



- After Scaling the above rectangle by (½, 1) and (2, 1):





1.3. Rotation

- When a sprite rotates in the game, its angle of rotation α is changed.
- In 2D, all rotations are done around the depth axis.
- A rotation matrix is needed to rotate the sprite before rendering in it.

$$\begin{bmatrix} cos\,\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- This matrix rotates a point around the Z-axis by an angle α with the point (0; 0) as the center of rotation.
- Usually when we want to rotate a point A around a center C, we translate A so that C coincides with the center of the coordinates system (0; 0), apply the rotation, and then translate back to the original position. In order to avoid these 2 extra translations, we just need to rotate the sprite in its local coordinates system before applying the translation matrix.
- There are 2 ways to rotate an object frame after frame:
  - Rotate the original object by the absolute rotation value
  - Rotate the object from the previous' frame orientation using an incremental value
- Remember that float numbers aren't saved accurately on computers.
  - Therefore each rotation matrix will rotate the object with a very small fractional error margin.
  - These errors are insignificant if you rotate an object just few times, but since in most games, many objects are rotated each frame, which will accumulate all these fractional errors.
  - This will eventually lead to a clear and disturbing visual problem.
- Therefore, it's better the make the angle α hold the current absolute angle of rotation of the sprite and not an incremental value, therefore it should always be the current rotation angle of the sprite, and not the difference between the previous and the current one.
- As mentioned previously, transformations are applied to the corner vertices and not each point of a certain image. Translating each pixel of an image might result in a translated image, but applying a rotation matrix to each pixel will surely lead to an incorrect image with gaps in it.
- The main reason for that problem is that pixel's locations are integers and not floating points.

1.4. Translation

- When a sprite moves in the game, its position (X & Y) variable is changed.
- Generally, a translation matrix moves a point from the position it holds to a new one. It's the same as vector addition.
- In games, a translation matrix is needed to transform the sprite from its local coordinates system to the world coordinates system.
- The translation matrix is built by taking an identity matrix and filling the position (X & Y) as the 1st two components of the 3rd column.

$$\begin{bmatrix} 1 & 0 & X \\ 0 & 1 & Y \\ 0 & 0 & 1 \end{bmatrix}$$

- This matrix translates any point by X on the X-axis and by Y on the Y-axis.
- X & Y are the real world position and not incremental values. This means that they translate a 2D point from (0;0) to (X;Y). However a sprite moves, its position should always hold its current absolute world position, and not the difference between the previous position and the current one.