Advanced Computer Graphics

3 - 2D Affine Transformations, Graphics Hardware

Yoonsang Lee

Fall 2018

Today's Topics

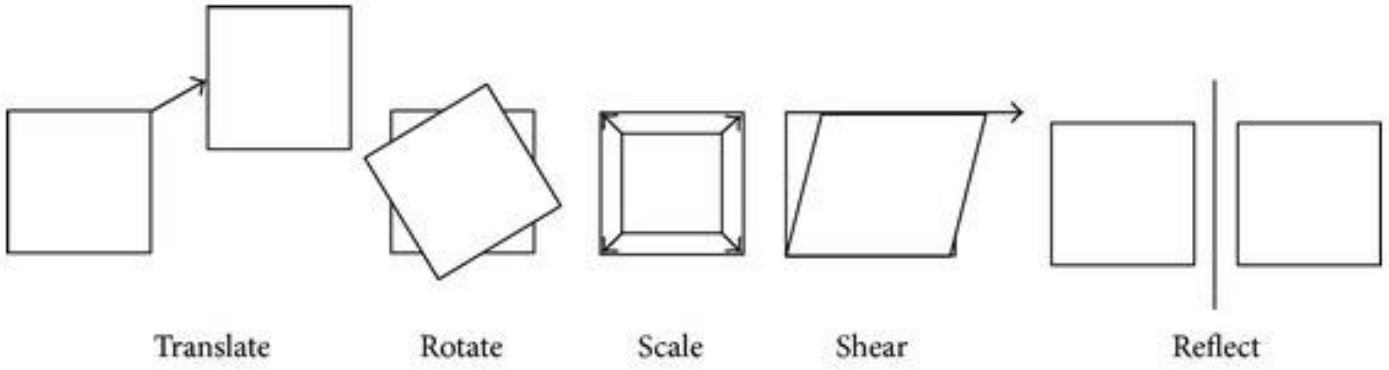
- 2D Affine Transformation
  - What is Transformation?
  - Linear Transformation, Translation
  - Affine Transformation

- Graphics Hardware
  - Vector & Raster Display
  - Frame Buffer
  - Double Buffering
  - VSync

2D Affine Transformations

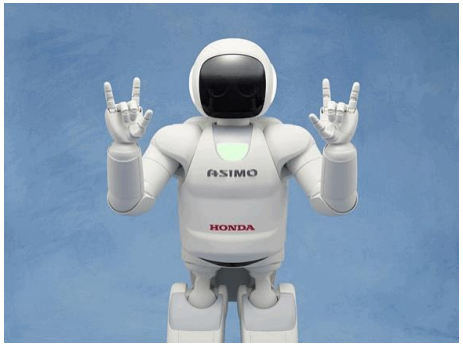
What is Transformation?

- (Geometric) **Transformation** - (기하) 변환
  - One-to-one mapping (function) of a set having some geometric structure to itself or another such set.
  - More easily, *"moving a set of points"*
- Examples:

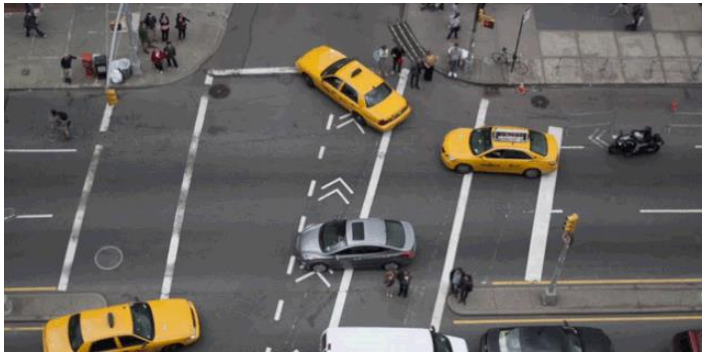


Where are Transformations used?

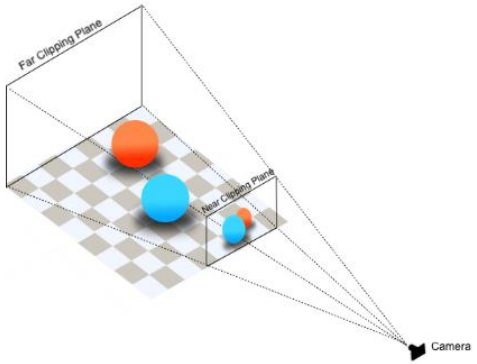
- Movement



- Image/object manipulation



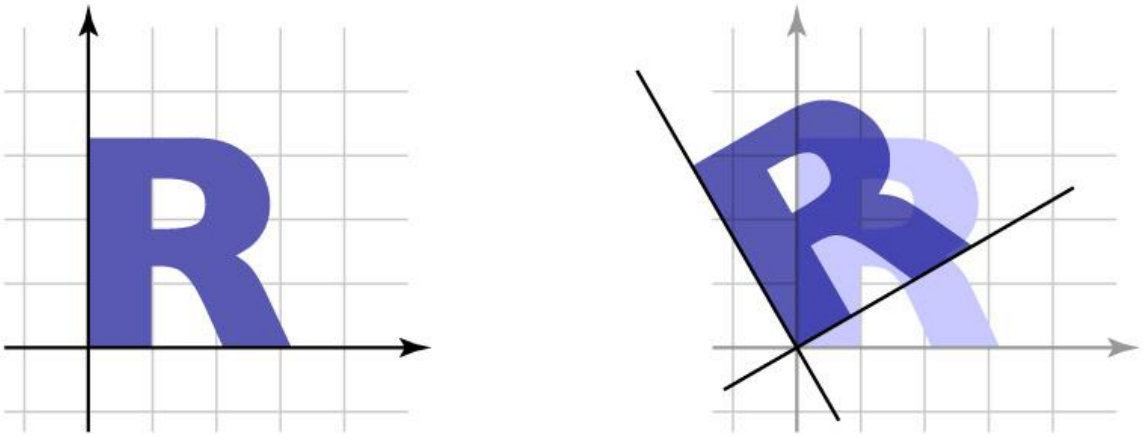
- Viewing transform



Transformation

- Moving a subset of the space using a mapping from the space to itself

$$S \to \{T(\mathbf{v}) \mid \mathbf{v} \in S\}$$



Linear Transformation

- One way to define a transformation is by matrix multiplication:

$$T(\mathbf{v}) = M\mathbf{v}$$

- This is called linear transformation because matrix multiplication represents linear mapping.

$$T(a\mathbf{u} + \mathbf{v}) = aT(\mathbf{u}) + T(\mathbf{v})$$

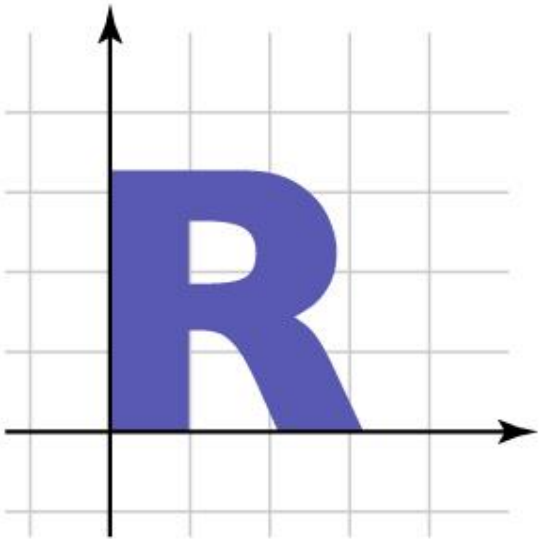
$$\mathbf{M} \cdot (a\mathbf{u} + \mathbf{v}) = a\mathbf{M}\mathbf{u} + \mathbf{M}\mathbf{v}$$

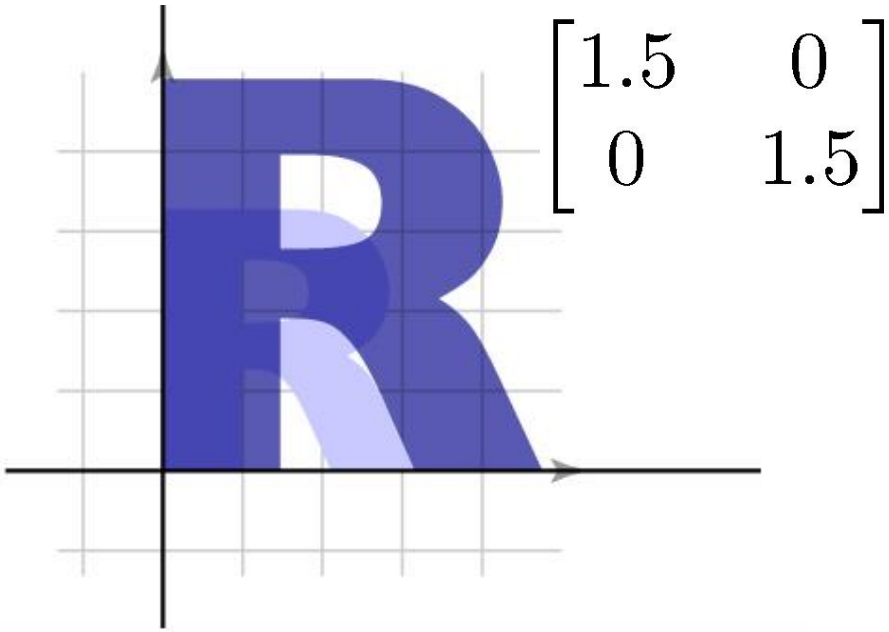
2D Linear Transformation

- 2x2 matrices have simple geometric interpretations
  - uniform scale
  - non-uniform scale
  - rotation
  - shear
  - reflection

2D Linear Trans. – Uniform Scale

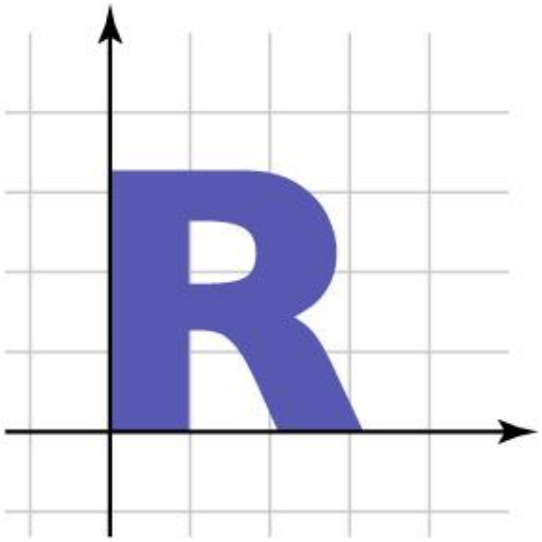
$$\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ sy \end{bmatrix}$$



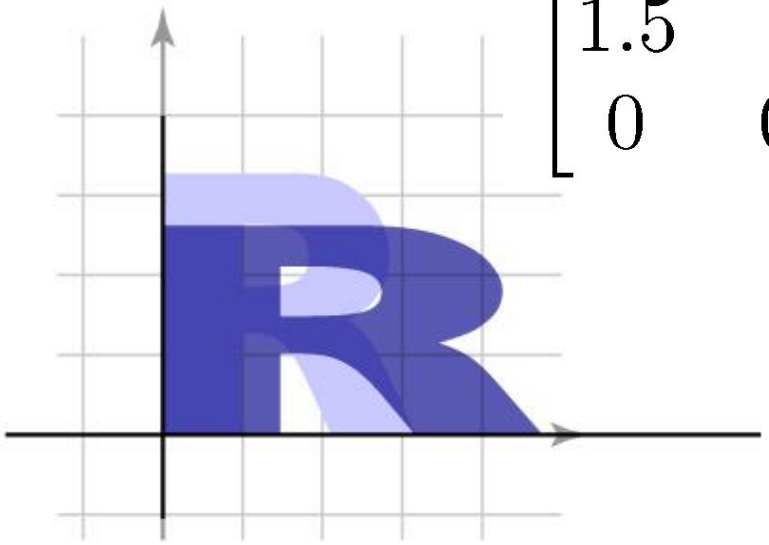


2D Linear Trans. – Nonuniform Scale

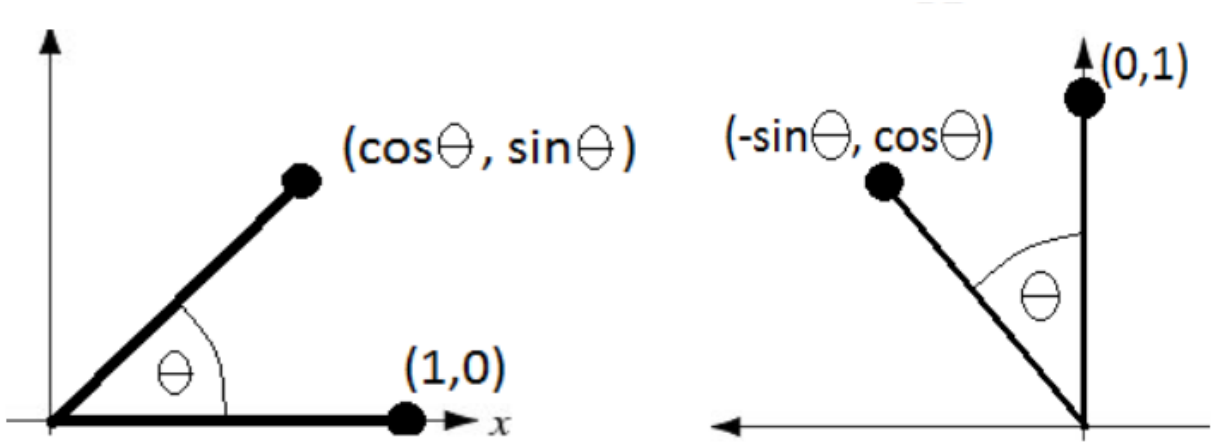
$$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$



$$\begin{bmatrix} 1.5 & 0 \\ 0 & 0.8 \end{bmatrix}$$



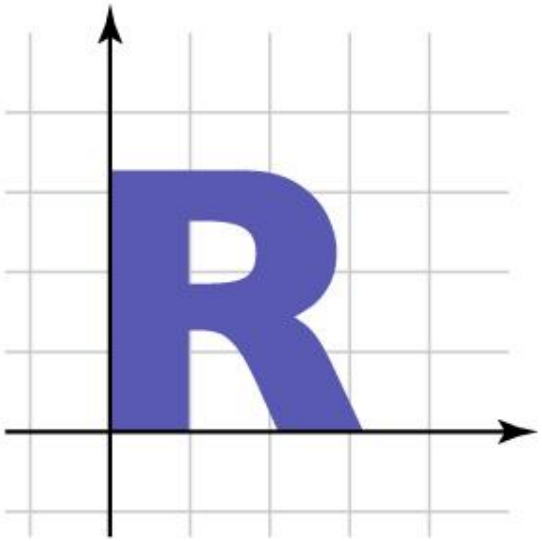
Rotation

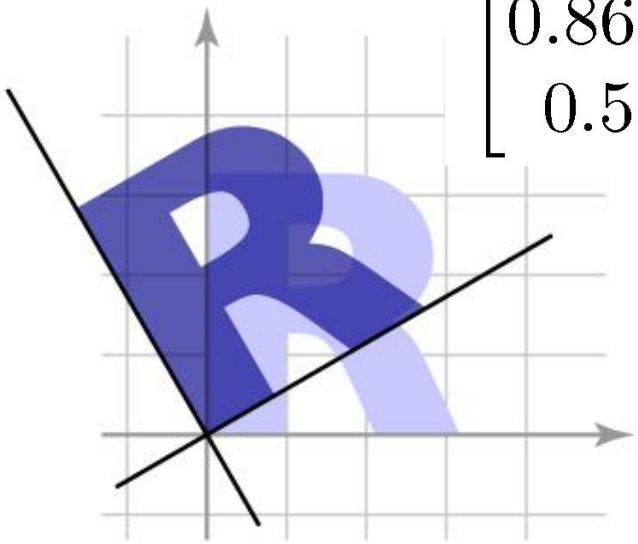


$$\Rightarrow R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

2D Linear Trans. – Rotation

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$



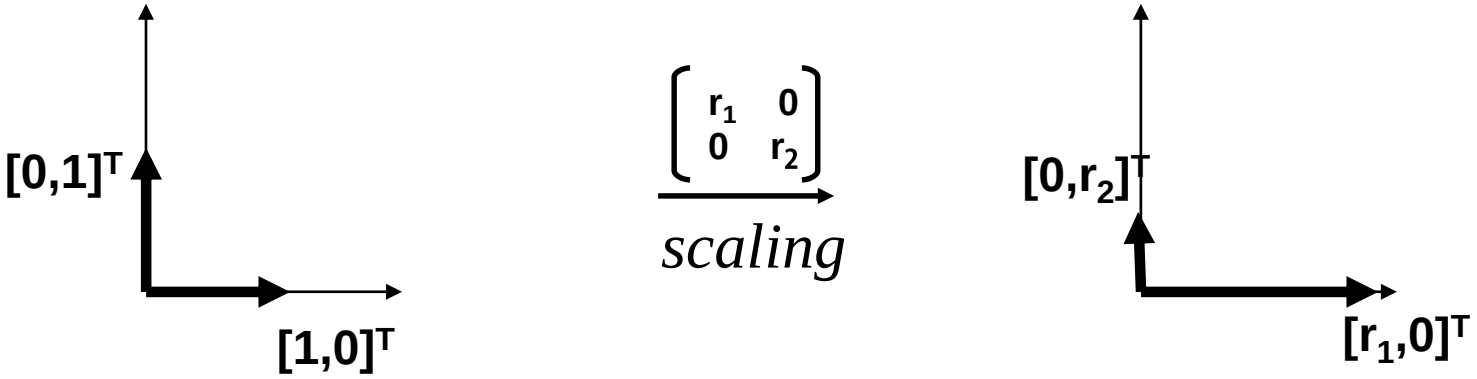


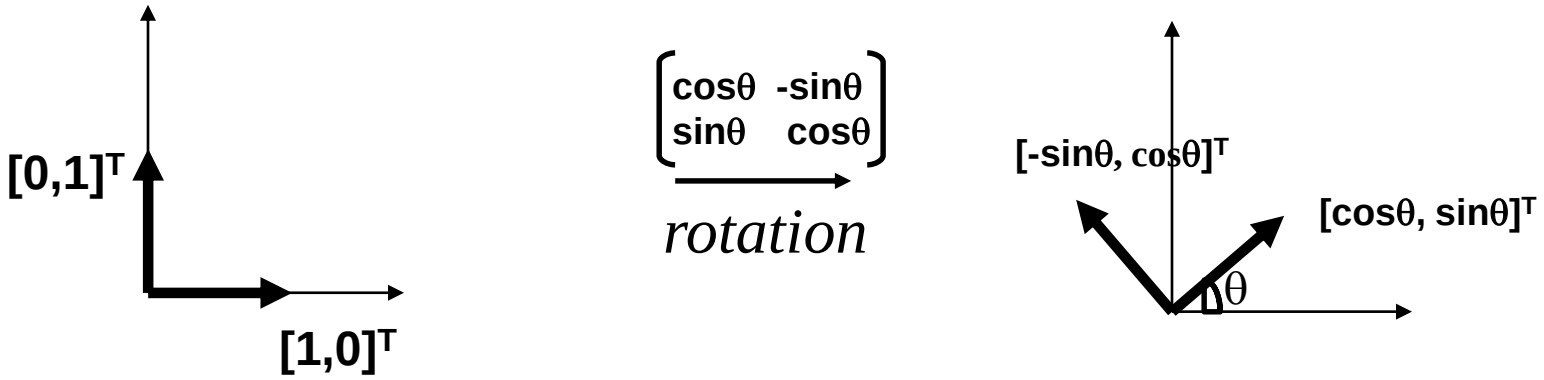
$$\begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix}$$

(Rotate 30 deg ccw)

Matrices: Scaling, Rotation, Identity

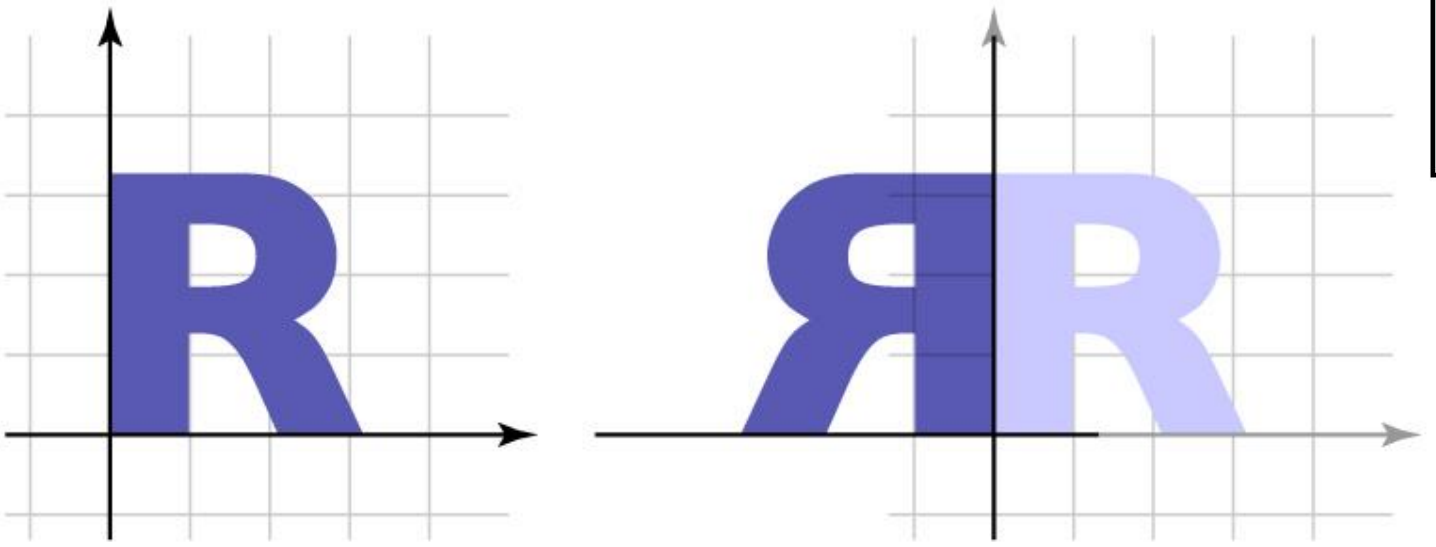
- Scaling without rotation => "diagonal matrix"
- Rotation without stretching => "orthonormal matrix"
- Identity ("do nothing") matrix = unit scaling, no rotation





2D Linear Trans. – Reflection

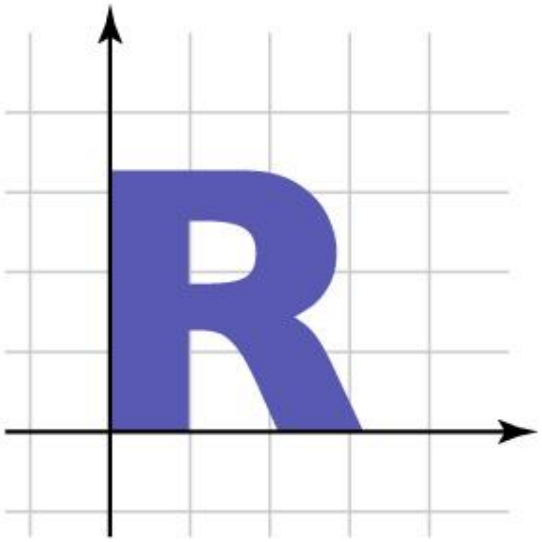
- You can consider it a special case of nonuniform scale

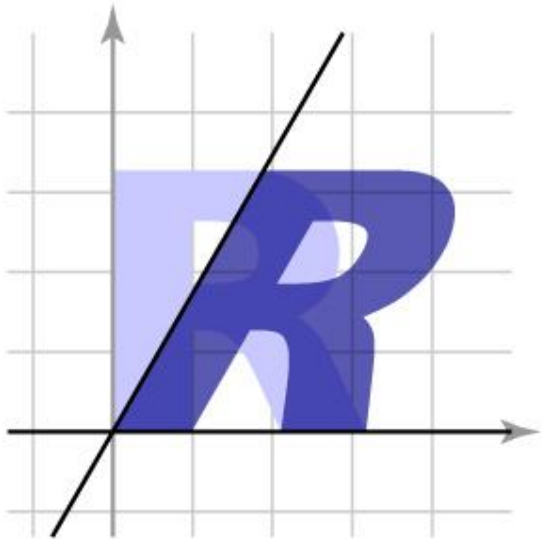


$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

2D Linear Trans. – Shear

$$\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + ay \\ y \end{bmatrix}$$



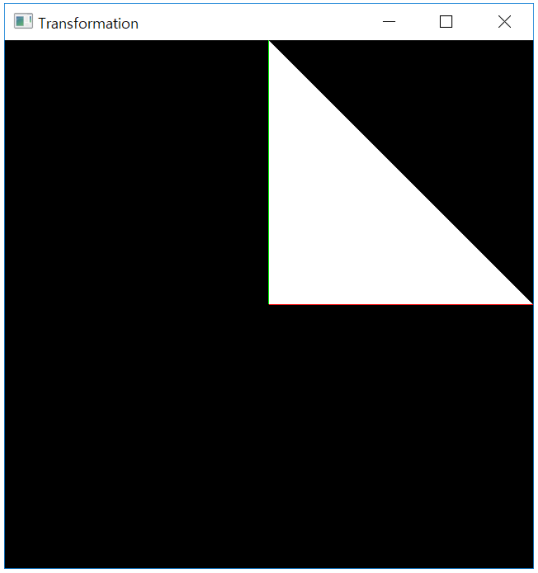


$$\begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}$$

[Practice] Uniform Scale

```
import glfw
from OpenGL.GL import *
import numpy as np

def render(T):
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()

    # draw cooridnate
    glBegin(GL_LINES)
    glColor3ub(255, 0, 0)
    glVertex2fv(np.array([0.,0.]))
    glVertex2fv(np.array([1.,0.]))
    glColor3ub(0, 255, 0)
    glVertex2fv(np.array([0.,0.]))
    glVertex2fv(np.array([0.,1.]))
    glEnd()

    # draw triangle
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 255, 255)
    glVertex2fv(T @ np.array([0.0,0.5]))
    glVertex2fv(T @ np.array([0.0,0.0]))
    glVertex2fv(T @ np.array([0.5,0.0]))
    glEnd()
```

[Practice] Uniform Scale




```
def main():
    if not glfw.init():
        return
    window = glfw.create_window(640,640,"2D
Trans", None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)

    while not glfw.window_should_close(window):
        glfw.poll_events()

        T = np.array([[2.,0.],
                      [0.,2.]])
        render(T)

        glfw.swap_buffers(window)

    glfw.terminate()

if __name__ == "__main__":
    main()
```

[Practice] Animate It!

```
def main():
    # ...

    # let's just skip - we'll see this later
    glfw.swap_interval(1)

    count = 0
    while not glfw.window_should_close(window):
        glfw.poll_events()

        s = (count % 10) * .1
        T = np.array([[s,0.],
                      [0.,s]])
        render(T)
        count += 1
        # ...
```

[Practice] Nonuniform Scale

```
def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        sx = (count % 10) * .1
        sy = sx * 2
        T = np.array([[sx,0.],
                      [0.,sy]])
        render(T)
        # ...
```

[Practice] Rotation

```
def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        th = np.radians(count % 360)
        T = np.array([[np.cos(th), -np.sin(th)],
                      [np.sin(th), np.cos(th)]])
        render(T)
        # ...
```

[Practice] Reflection

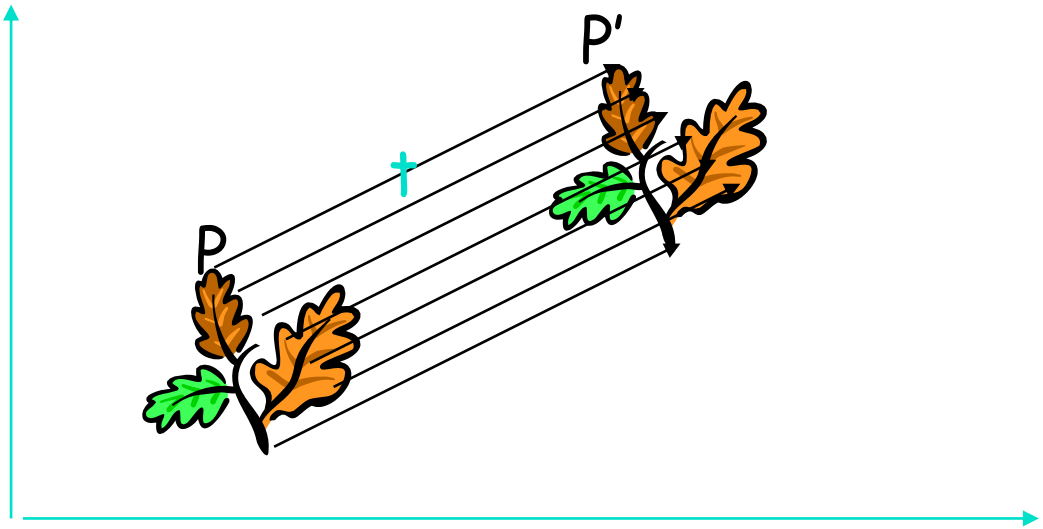
```
def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        T = np.array([[-1.,0.],
                      [0.,1.]])
        render(T)
        # ...
```

[Practice] Shear

```
def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        a = (count % 10) * .2
        T = np.array([[1.,a],
                      [0.,1.]])
        render(T)
        # ...
```

2D Translation

- Translation is the simplest transformation:
  $T(\mathbf{v}) = \mathbf{v} + \mathbf{u}$
- Inverse:
  $T^{-1}(\mathbf{v}) = \mathbf{v} - \mathbf{u}$



[Practice] Translation

```
def render(u):
    # ...
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 255, 255)
    glVertex2fv(np.array([0.0,0.5]) + u)
    glVertex2fv(np.array([0.0,0.0]) + u)
    glVertex2fv(np.array([0.5,0.0]) + u)
    glEnd()

def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        s = (count % 10) * .1
        u = np.array([s, 0.])
        render(u)
        # ...
```

Is translation linear transformation?

- No. because it cannot be represented using a simple matrix multiplication.

- We can express using vector addition:
  $T(\mathbf{v}) = \mathbf{v} + \mathbf{u}$

- Combining with linear transformation:
  $T(\mathbf{v}) = M\mathbf{v} + \mathbf{u}$

  ➡ **Affine transformation**

Let’s check again

- Linear transformation
  - Scale, rotation, reflection, shear
  - Represented as matrix multiplication

$$T(\mathbf{v}) = M\mathbf{v}$$

- Translation
  - Not a linear transformation
  - Can be expressed using vector addition

$$T(\mathbf{v}) = \mathbf{v} + \mathbf{u}$$

Affine Transformation

- Linear transformation + Translation

$$T(\mathbf{v}) = M\mathbf{v} + \mathbf{u}$$

- Preserves lines
- Preserves parallel lines
- Preserves ratios of distance along a line
- -> These properties are inherited from linear transformations.

Rigid Transformation

- Rotation + Translation

$$T(\mathbf{v}) = R\mathbf{v} + \mathbf{u}$$, where R is a rotation matrix.

- Preserves distances between all points
- Preserves cross product for all vectors (preventing reflection)

Summary of Transformations

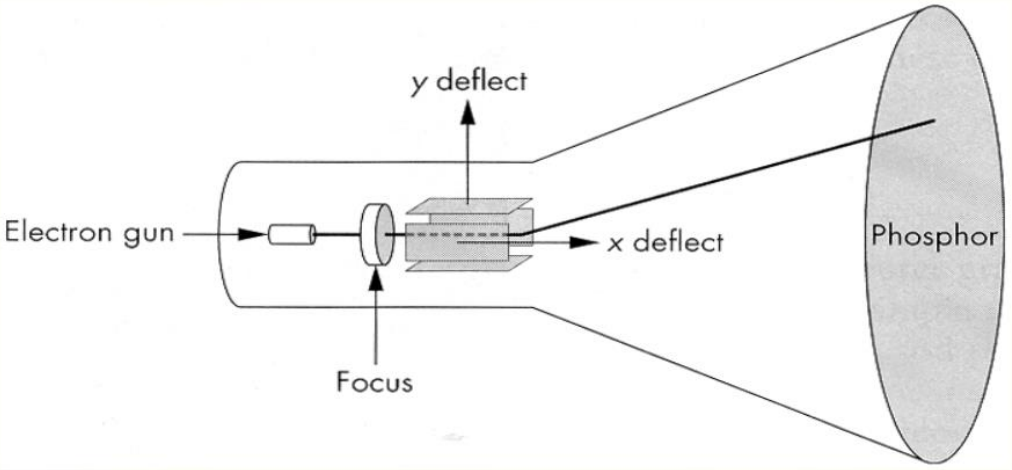
- Linear
  - Scale
  - Rotation
  - Reflection
  - Shear
  - ...
- Nonlinear
  - Translation
  - ...
- Affine
  - Linear transformation + Translation
- Rigid
  - Rotation + Translation

[Practice] Affine Transformation

```
def render(M, u):
    # ...
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 255, 255)
    glVertex2fv(M @ np.array([0.0,0.5]) + u)
    glVertex2fv(M @ np.array([0.0,0.0]) + u)
    glVertex2fv(M @ np.array([0.5,0.0]) + u)
    glEnd()

def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        th = np.radians(count % 360)
        T = np.array([[np.cos(th), -np.sin(th)],
                      [np.sin(th), np.cos(th)]])

        s = (count % 10) * .1
        u = np.array([s, 0.])
        render(T, u)
        # ...
```

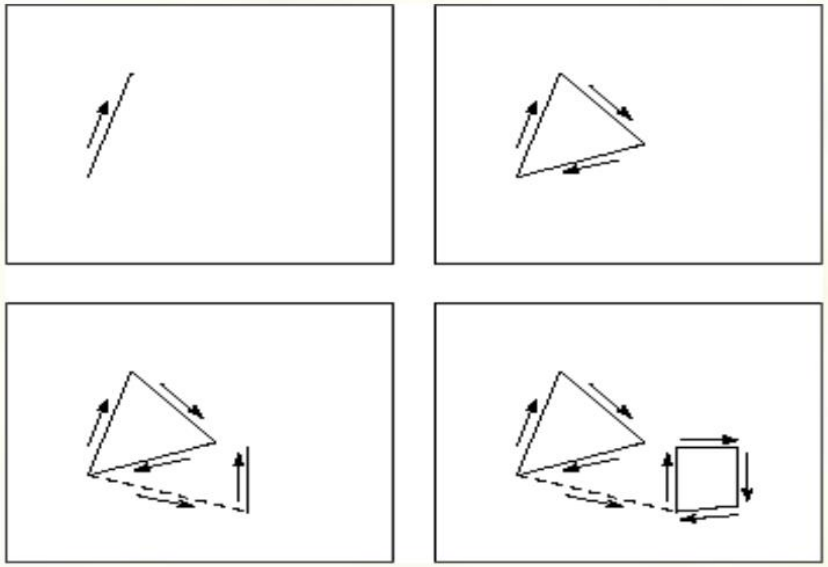
Graphics Hardware

Cathode ray tubes (CRTs)

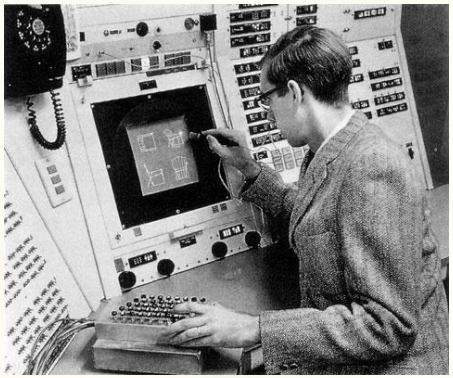


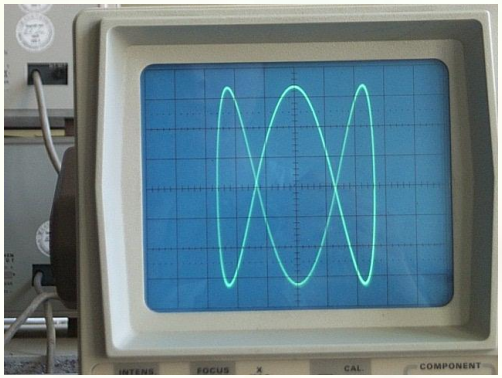
- Consists of:
  - electron gun
  - electron focusing lens
  - deflection plates/coils
  - electron beam
  - anode with phosphor coating

Calligraphic displays (Vector displays)



- Also called **vector displays**, **stroke displays**, or **random-scan displays**.
- Used by:
  - Sutherland's Sketchpad
  - Asteroids video game
  - Oscilloscopes

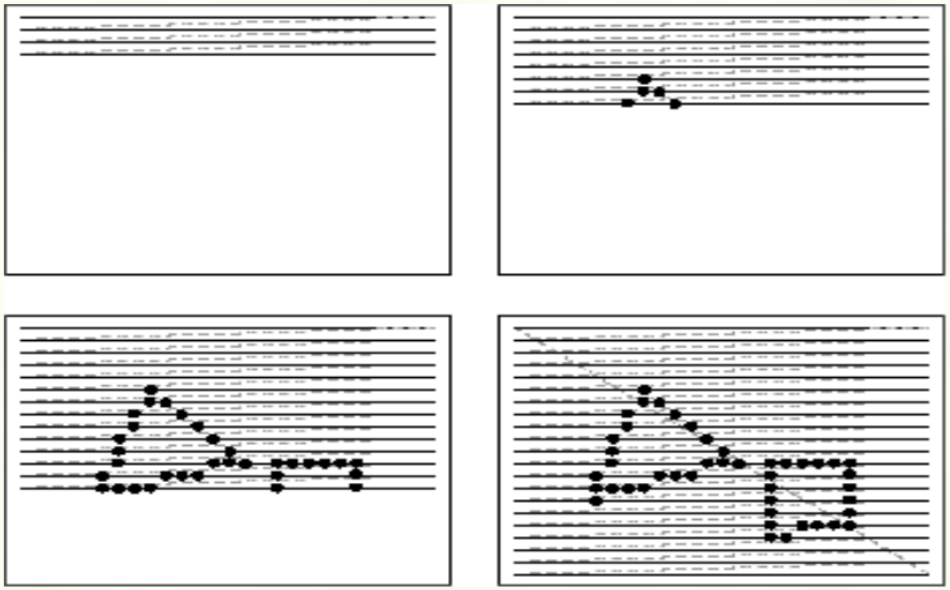




Modern use – Laser light shows

Raster displays

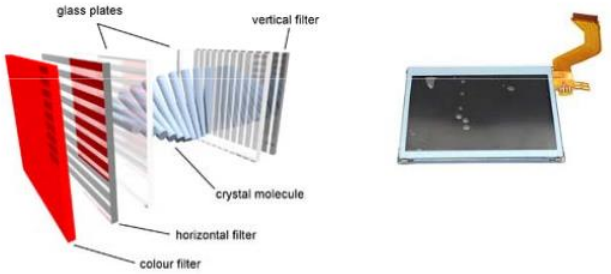
- **ras.ter**, from radere, "to scrape"



- Electron beam traces over screen in **raster scan order**.
  - Each left-to-right trace is called a **scan line**.
  - Each spot on the screen is a **pixel**.
  - When the beam is turned off to sweep back, that is a **retrace**, or a **blanking interval.**

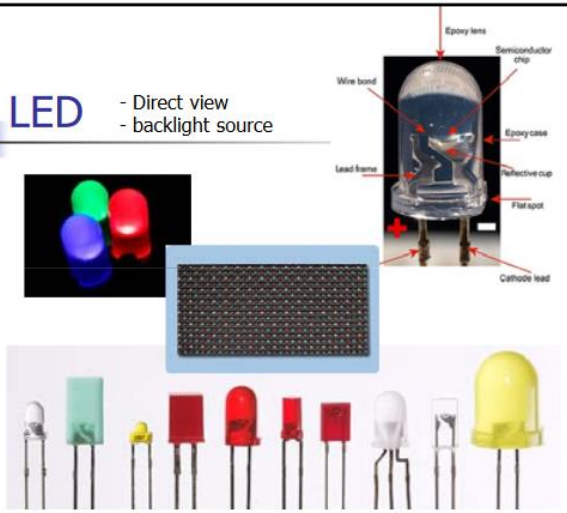
Raster Displays

LCD: backlit



LED

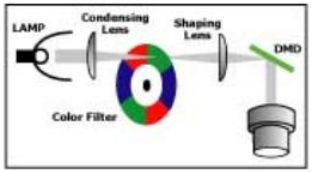
- Direct view
- backlight source



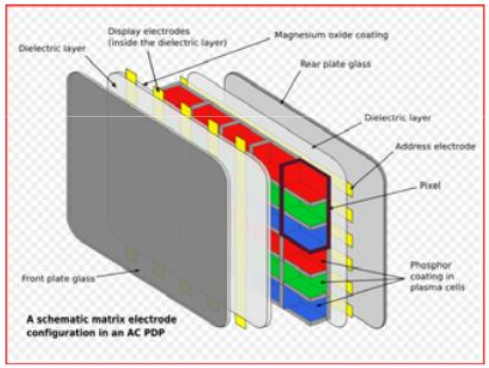
DLP: http://www.dlp.com/includes/video_demo.aspx

For digital projection

Digital Micromirror Device



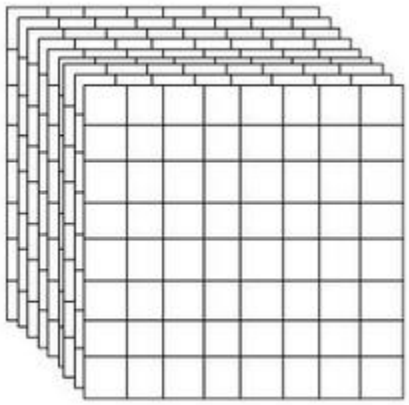
Plasma Panels: emit light; soon extinct?



Frame Buffer

- The portion of memory to hold the bitmapped image that is sent to the (raster) display device.

- A frame buffer is characterized by its width, height, and depth.
  - E.g. The frame buffer size for 4K UHD resolution with 32bit color depth = 3840 x 2160 x 32 bits



- Typically stored on the graphic card's memory.
  - But integrated graphics (e.g. Intel HD Graphics) use the main memory to store the frame buffer.

Single Buffering

- Using a single frame buffer both for displaying image data & “drawing” new image data

- You cannot update image data while sending image data to display
  - This causes lower frame rate (fps)
  - If you force update, you’ll see “screen tearing” artifact : showing information from multiple frames in a single screen draw

Double Buffering

- Using two buffers for displaying image data & drawing new image data

- Display image data in **front buffer**
- Draw new image data to **back buffer**

- When drawing image data for one frame is done, swap pointers of front & back buffer (or copy the back buffer to the front buffer)

- This swap or copy is what `glfw.swap_buffers()` does in our program

Double Buffering

- Pros
  - Higher frame rate
    - Because you can draw new image to back buffer while displaying image in front buffer
  - Much less screen tearing
    - Because it takes much less time to swap or copy than render all objects in a single frame buffer
- Cons
  - More memory (for 2nd buffer)
- Most applications are working with double buffering

VSync (Vertical Synchronization)

- Swap/copy back buffer to front buffer ONLY during vertical blanking time
  - During vertical blanking time, monitor prepares drawing next frame and draws nothing
  - It occurs at every monitor refresh cycle (e.g. 60 Hz)

- So VSync prevents screen tearing

- `glfw.swap_interval()` sets the number of VSync signals to wait before swapping the buffers

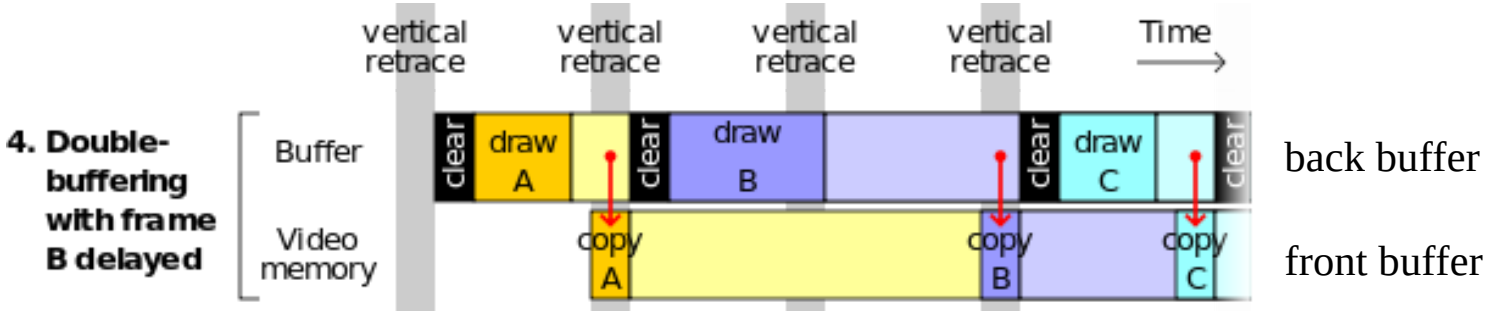
```
void glfwSwapInterval ( int  interval )
```

=the number of vsync signals to wait

This function sets the swap interval for the current OpenGL or OpenGL ES context, i.e. the number of screen updates to wait from the time glfwSwapBuffers was called before swapping the buffers and returning. This is sometimes called *vertical synchronization*, *vertical retrace synchronization* or just *vsync*.

VSync (Vertical Synchronization)

- FPS will be limited to monitor refresh rate (**only** when a frame image is generated **fast enough**)
  - E.g. Monitor at 60Hz, FPS will be 60Hz (only when a frame is rendered within 1/60 sec)
  - If it can't produce a frame fast enough, it'll have to wait for the screen's next refresh cycle before it can display
  - If this happens frequently, FPS will be 30 instead of 60



[Practice] Changing Swap Interval

- Change the code like: `glfw.swap_interval(10)`

- For some cases, it does not make any changes due to:

3.3 - Why doesn't glfwSwapInterval work?

Modern graphics drivers have settings that allow users to override an application's request for (among other things) swap interval. If such a setting is enabled, `glfwSwapInterval` will have no effect.

  - http://www.glfw.org/faq.html#why-doesnt-glfwswapinterval-work

Next Time

- Homogeneous Coordinates, 3D Affine Transformations

- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Taesoo Kwon, Hanyang Univ., http://calab.hanyang.ac.kr/cgi-bin/cg.cgi
  - Prof. Steve Marschner, Cornell Univ., http://www.cs.cornell.edu/courses/cs4620/2014fa/index.shtml
  - Prof. Don Fussell, Univ. of Texas at Austin, https://www.cs.utexas.edu/users/fussell/courses/cs384g-fall2011/
  - Prof. Rick Parent, Ohio State Univ., http://web.cse.ohio-state.edu/~parent.1/classes/581/