Vector Graphics

Multimedia Techniques & Applications

Yu-Ting Wu

Outline

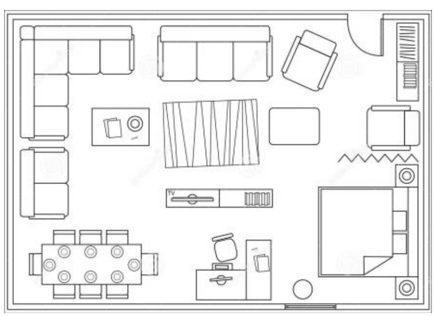
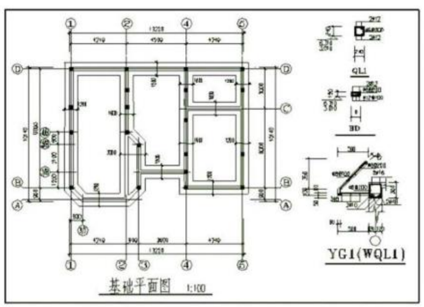
- Overview
- Fundamentals
- Shapes
- Stroke and fill
- Transformation

Overview

- Images of vector graphics are built up using shapes that can easily be described **mathematically**
- Vector graphics provide an elegant way of constructing digital images whose representation is
  - Compact
  - Scaleable
  - Resolution-independent
  - Easy to edit

Uses of Vector Graphics

- Graphics that will be scaled (or resized)
  - Architectural drawings or CAD programs
  - Flowcharts
  - Logos
- Cartoons and clipart
- Graphics on websites
- Fonts and specialized text effects

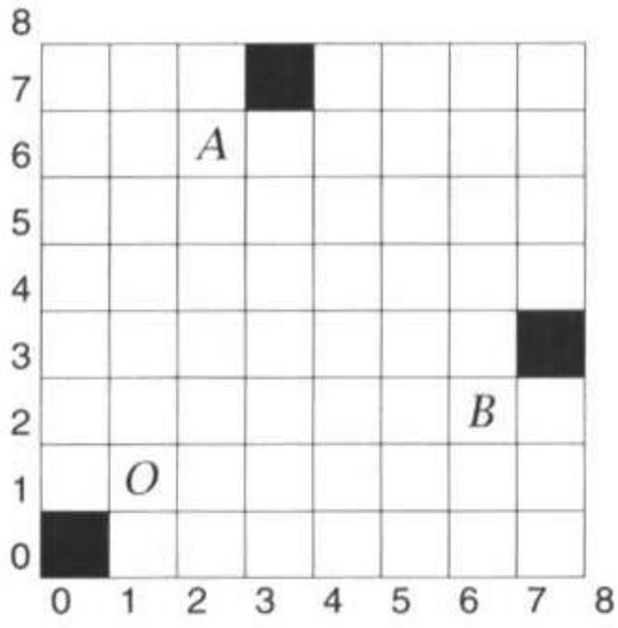
Uses of Vector Graphics (cont.)

- 3D computer graphics can also be considered as one type of vector graphics
  - Use math to describe shapes, materials, and light-surface interaction
  - Generate an image captured by a virtual camera

Fundamentals

Coordinates

- An image is stored as a rectangular array of pixels, so a natural way of identifying a single pixel is by giving its **column** and **row** number in the rectangular array
- The pair of column and row number is called **coordinate**



coordinate

A (3, 7)

column row

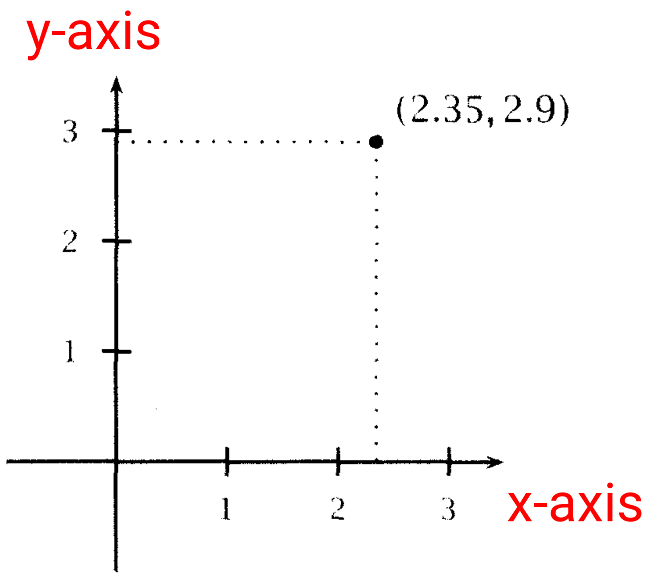
B (7, 3)

C (0, 0)

origin

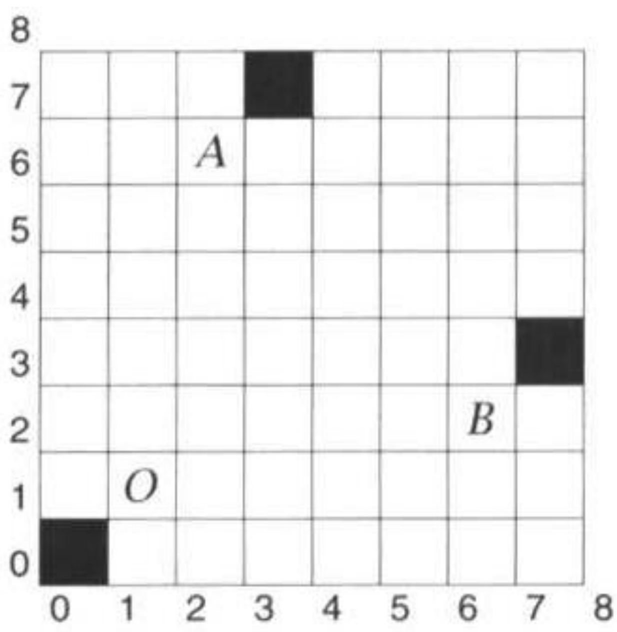
Coordinates (cont.)

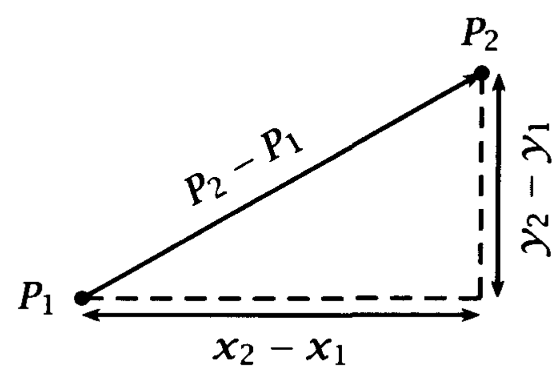
- The coordinates of pixels in an image must be integer values between **zero** and the **horizontal** (for x coordinates) or **vertical** (for y coordinates)
- But we can generalize to a coordinate system that has any real value (including negative ones)



Vector

- Pairs of coordinates can be used not only to define points, but also to define displacements
- Example: to get from A (3, 7) to B (7, 3), we need to move 4 units to the right, and 4 units down (-4 units up)





displacement from P1 to P2:
(x2 − x1, y2 − y1)

two-dimensional vector

Coordinates and Vector

- The generalization of coordinate system lets us identify points in space
- Using **letters** to represent unknown values
- Using **equations** to specify relationships between coordinates
- Example:

$$x = y$$

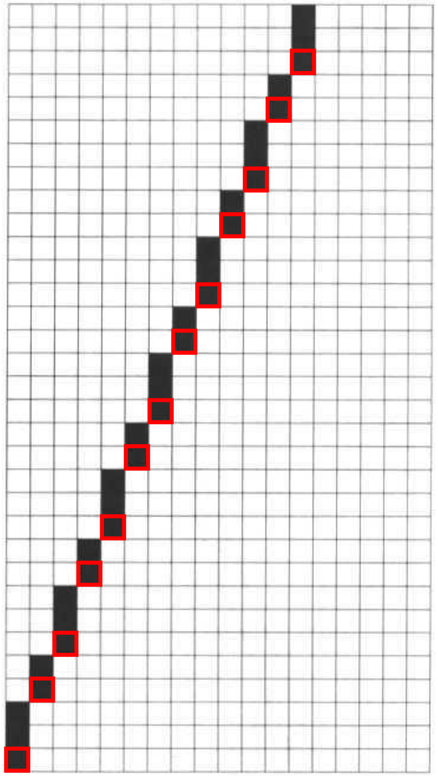
means a straight line passing through the origin at an angle 45 degree from south-west to north-east

or all points located on the line

Rendering of Math

- When it becomes necessary to render a vector drawing, the **stored values** (e.g., end points of a line) are used in conjunction with the **general form** of the description of each class of object
  - Can be considered as **sampling**
- Example: y = 5x/2 + 1

  pass through (0, 1), (1, 4), (2, 6), (3, 9) …



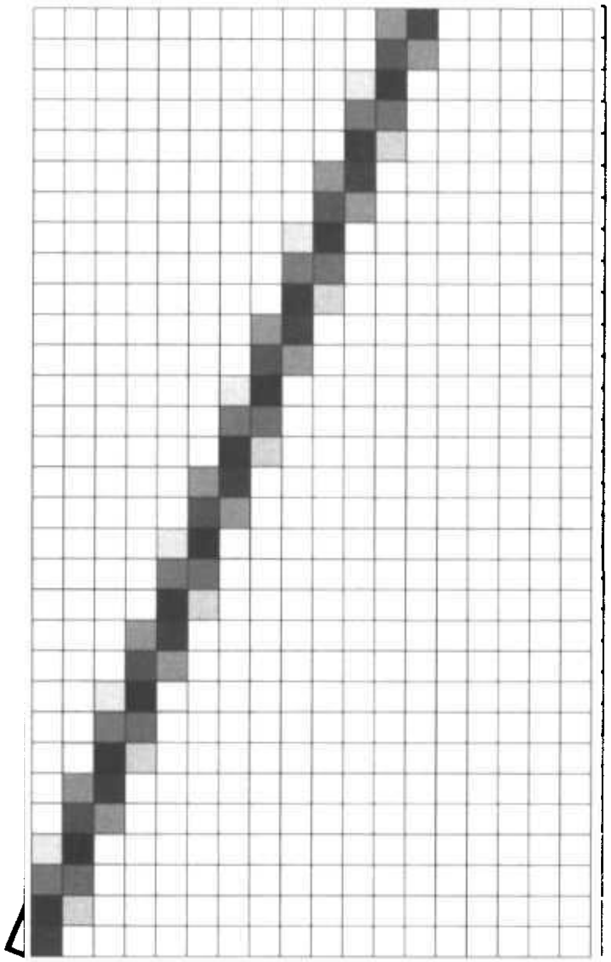
- Jaggedness are inevitable!
  - Due to the use of a grid of discrete pixels

Anti-aliasing

- The process of rendering a vector object to produce an image made up of pixels can usefully be considered as a form of **sampling** and **reconstruction**
  - The x and y coordinates can very infinitesimally
  - We approximate them by a sequence of pixel values at **fixed** finite intervals
  - Jaggies are a form of aliasing caused by undersampling
  - At an edge whose brightness change directly from one value to another without any intermediate gradation, its frequency domain will include **infinitely** high frequencies
  - As a result, no sampling rate will be adequate to ensure perfect reconstruction

Anti-aliasing (cont.)

- Anti-aliasing is a **practical** technique to reduce the jaggies
- Use intermediate grey values
  - In frequency domain, it relates to reduce the frequency of the signal
- Coloring each pixel in a shade of grey whose **brightness is proportional to the area** of the intersection between the pixels and a “**one-pixel-wide**” line



Shapes

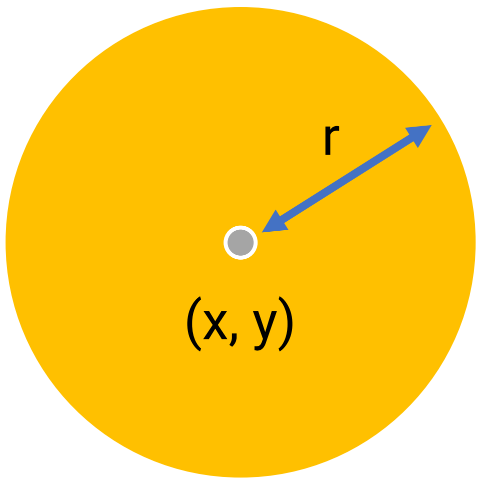
Shapes in Vector Graphics

- The shapes in a vector graphics editor are usually restricted to those with simple mathematical representation, such as
  - Rectangles (and squares)
  - Ellipses (and circles)
  - Straight lines
  - Polygons
  - **Smooth curves**

- Shapes built up out of these elements can be filled with **color**, **patterns**, or **gradients**
- We can also easily move, rotate, or scale these shapes

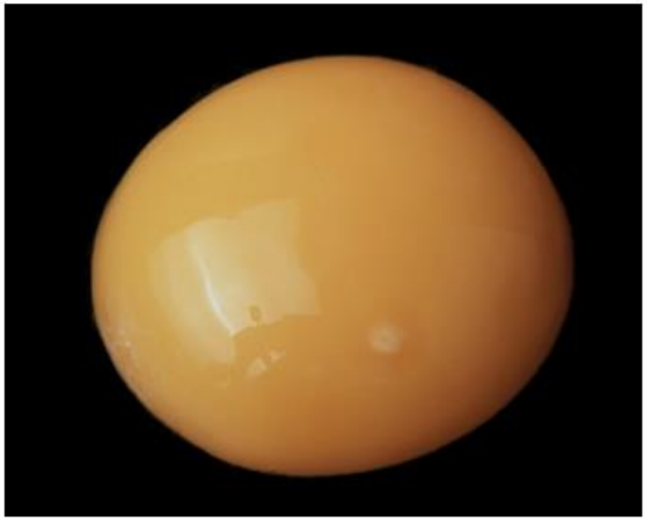
Shapes in Vector Graphics (cont.)

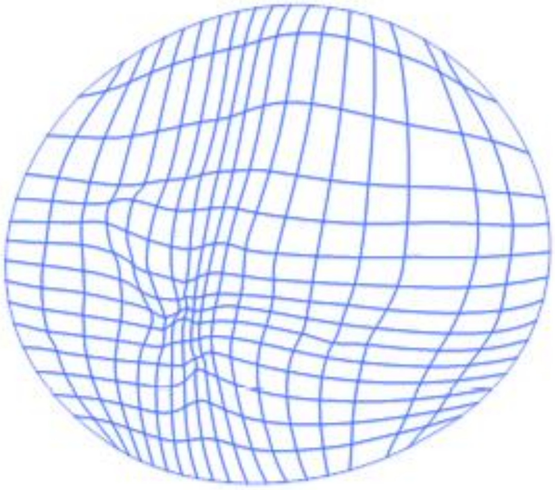
- Example: circle
  - Center point (x, y)
  - Radius (r)



Curves

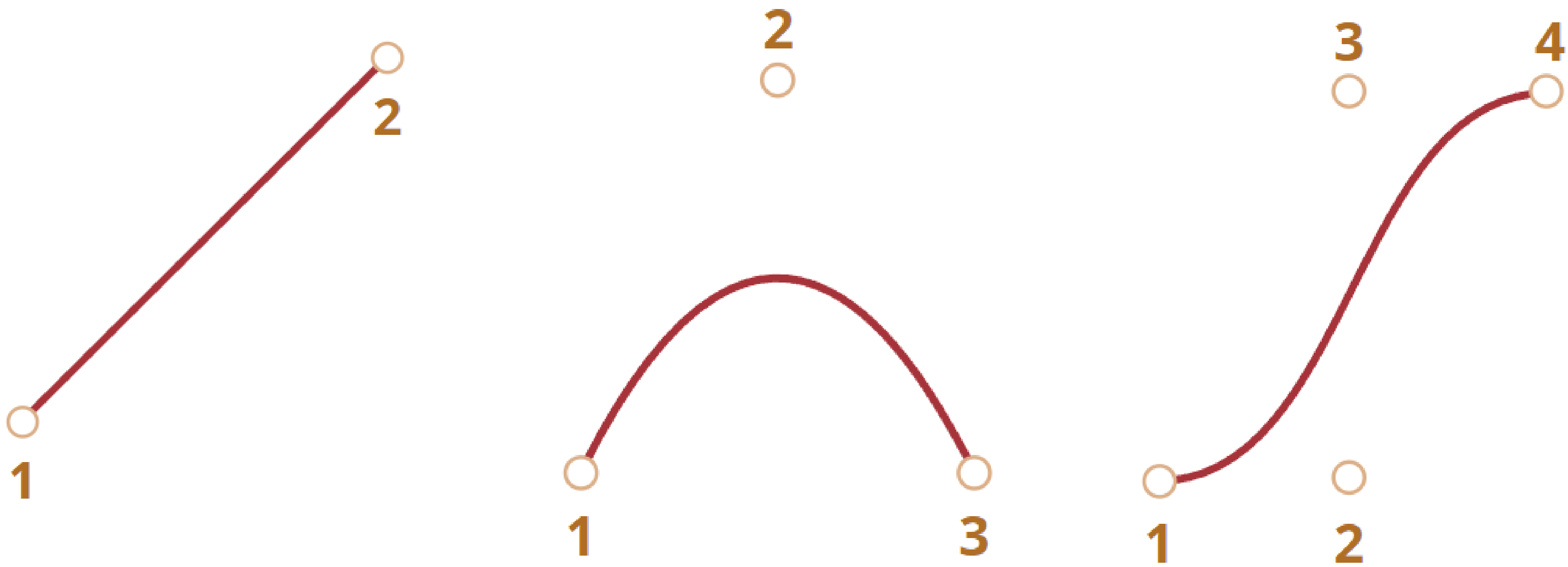
- Lines, rectangles, and ellipses are suitable for drawing technical diagrams
- But less constrained drawing and illustration requires more versatile shapes: **(Bezier) curves**





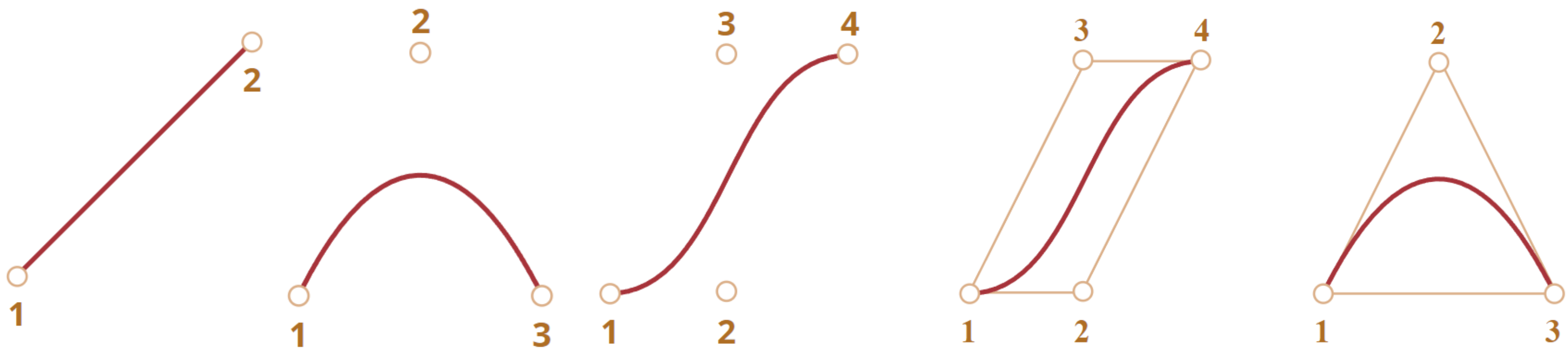
Bezier Curves

- Specified by **control points**
  - A set of points that influence the curve's shape
  - May be 2, 3, 4 or more



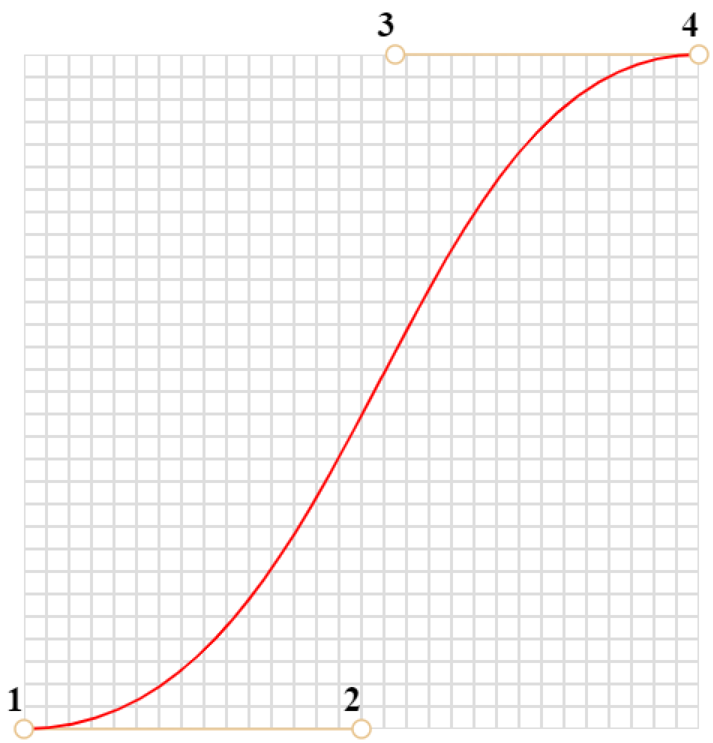
Bezier Curves (cont.)

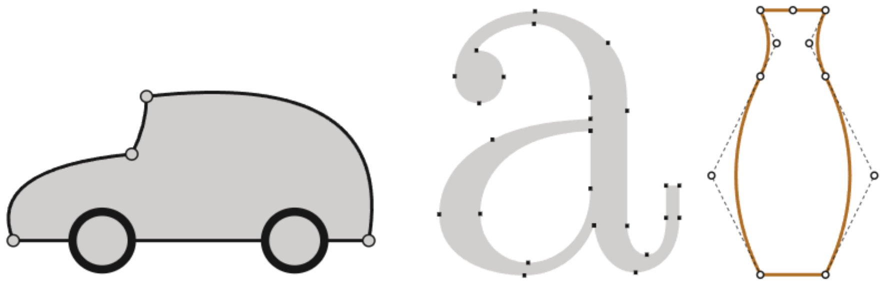
- Properties of **control points**
  - Control points are not always on curve
  - The order of curve equals the number of points minus one
    - Two points: linear curve (straight line)
    - Three points: quadratic curve (parabolic)
    - Four points: cubic curve
  - A curve is always inside the **convex hull** of control points



Bezier Curves (cont.)

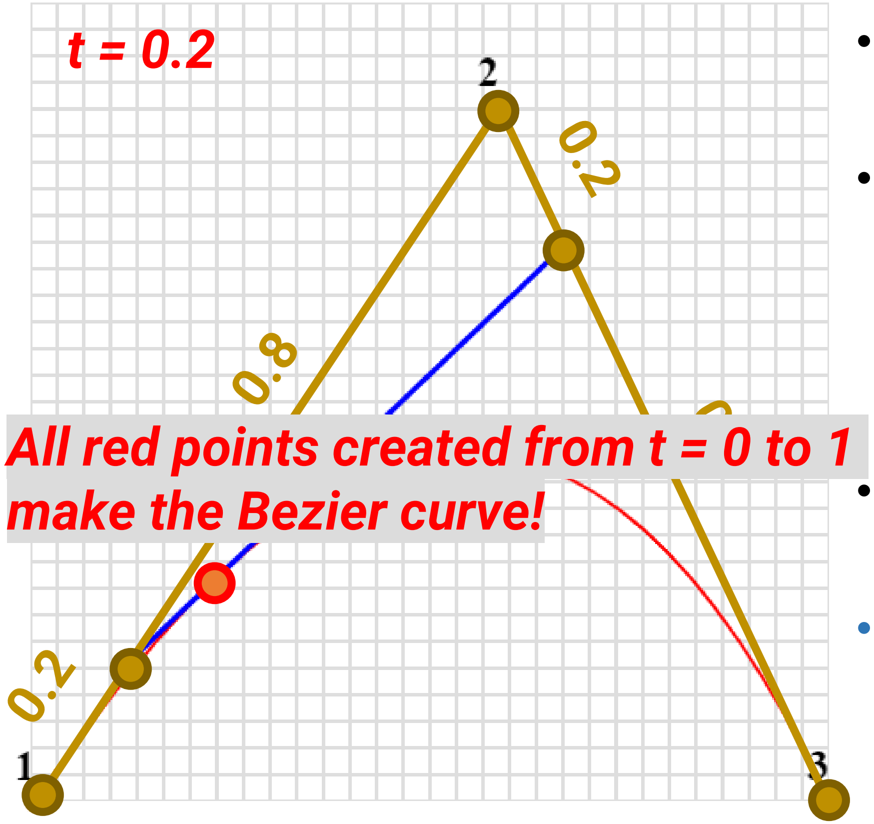
- Main value of Bezier curves
  - By moving the points, the curve is changing in an intuitive way
  - Demo: https://javascript.info/bezier-curve





Bezier Curves (cont.)

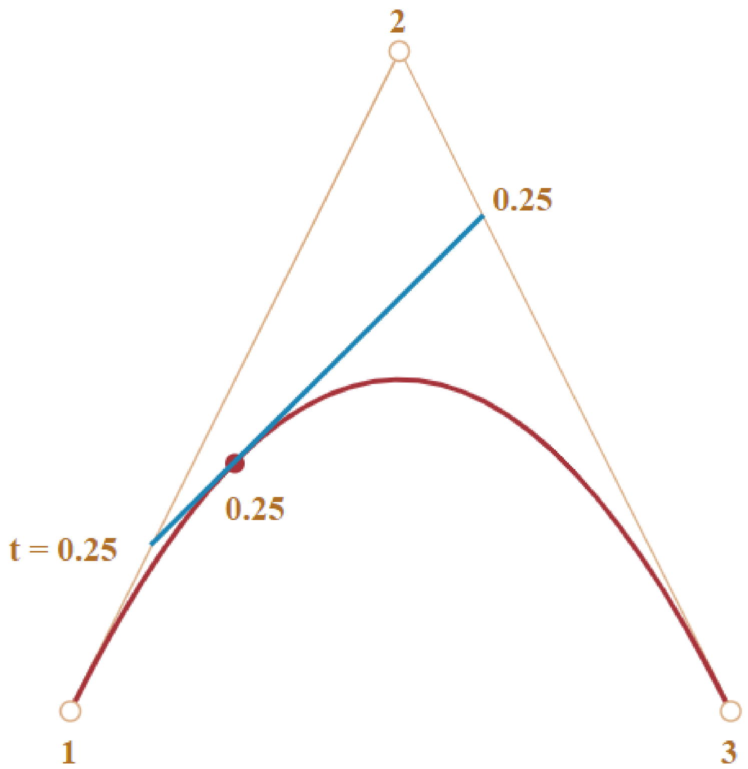
- Construct a Bezier curve using **De Casteljau's algorithm**
- Example: three-points Bezier curve

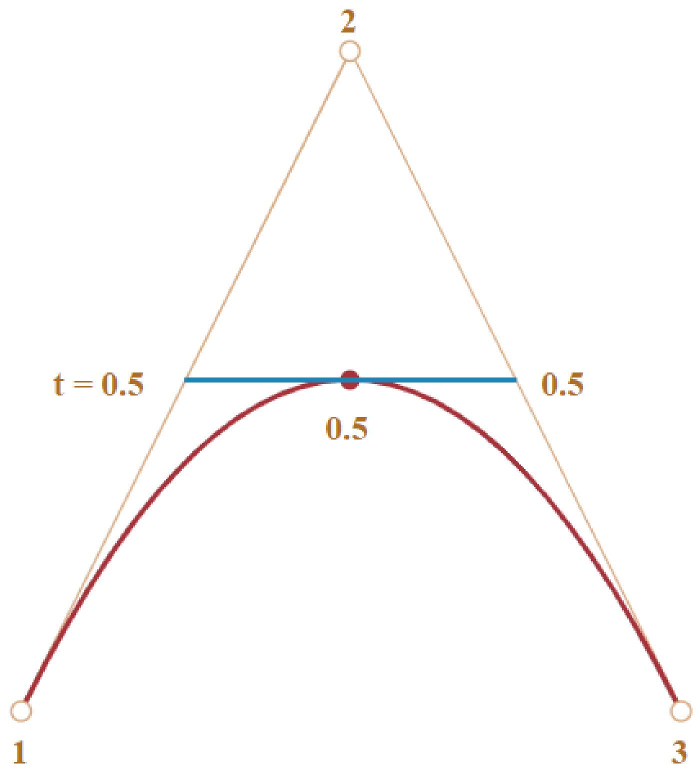


- Build line segments using P1, P2, and P3 (**two brown segments**)
- For a value ***t*** moving from 0 to 1, **on each brown segment**, take a point located on the distance proportional to ***t*** from its beginning (**two brown points**)
- Connect the **two brown points**, forming a **blue segment**
- **On the blue segment**, take a point located on the distance proportional to t from its beginning (**red point**)

Bezier Curves (cont.)

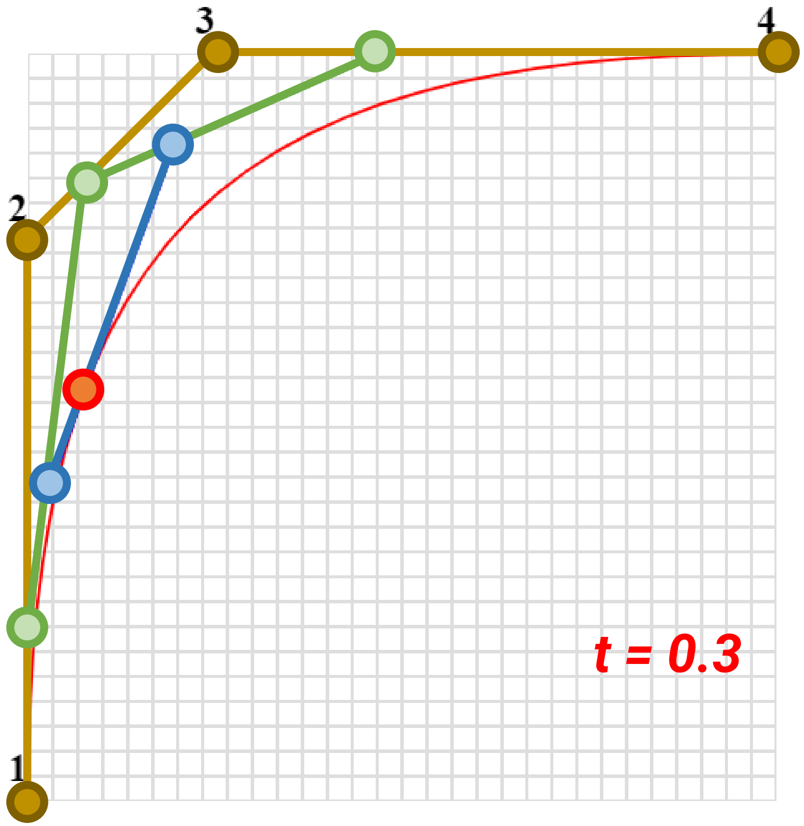
- Construct a Bezier curve using De Casteljau's algorithm
- Example: three-points Bezier curve

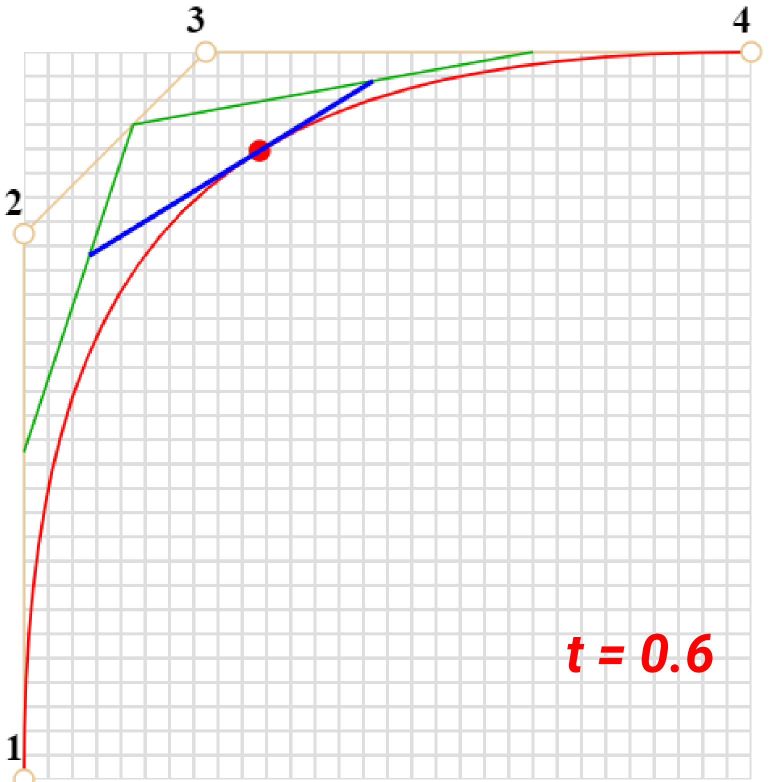




Bezier Curves (cont.)

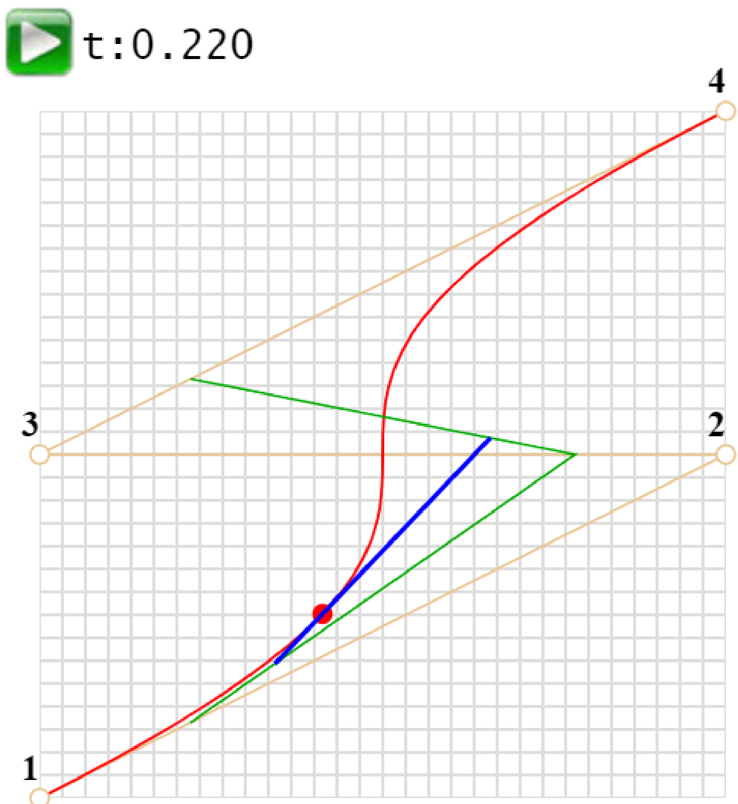
- Construct a Bezier curve using **De Casteljau's algorithm**
- Example: four-points Bezier curve



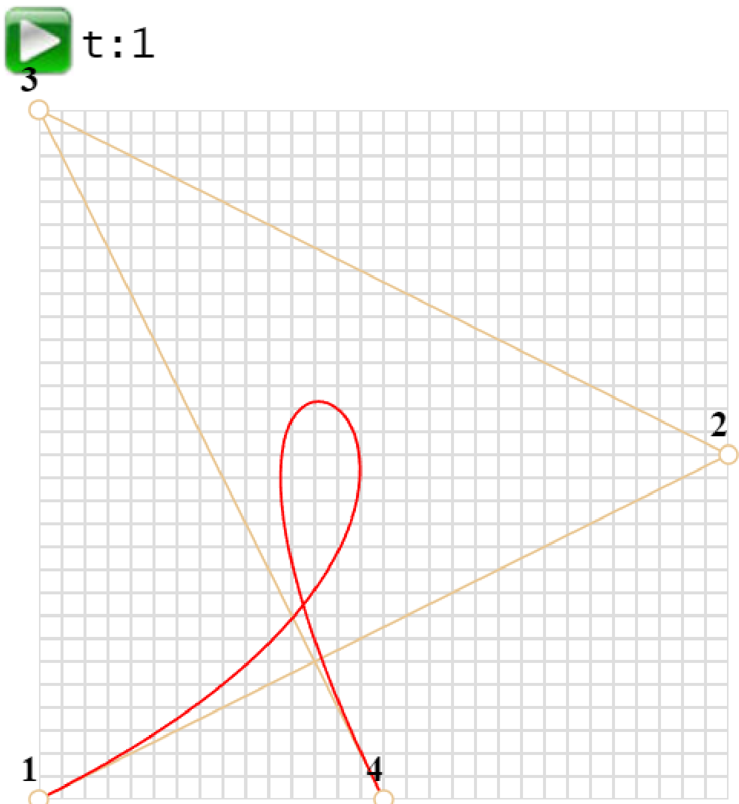


Bezier Curves (cont.)

- Other possible shapes of Bezier curves



zig-zag

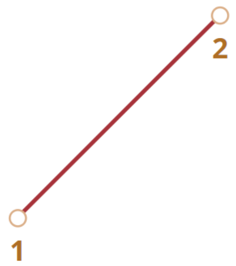


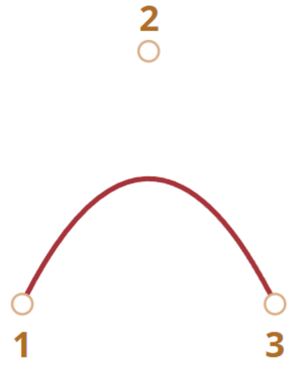
loop

Bezier Curves (cont.)

- Construct a Bezier curve using mathematical formula
- Two-points curve

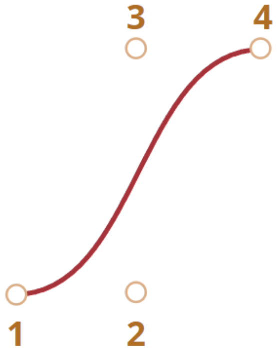
$$P = (1-t)P_1 + tP_2$$





- Three points curve

$$P = (1-t)^2 P_1 + 2(1-t)tP_2 + t^2 P_3$$

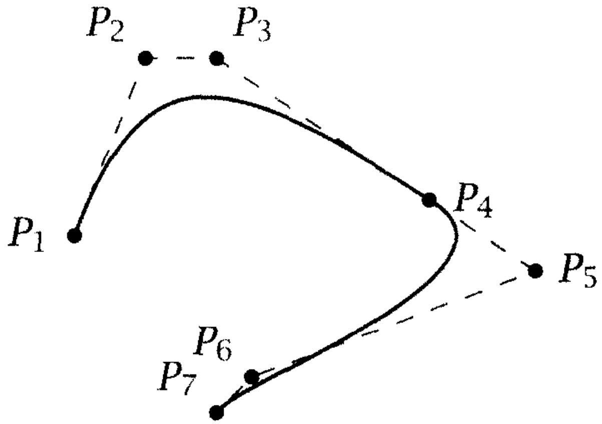


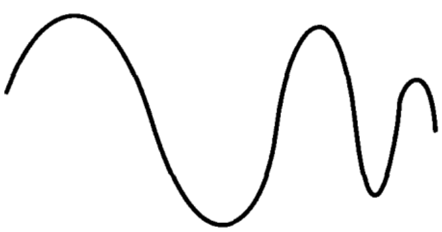
- Four points curve

$$P = (1-t)^3 P_1 + 3(1-t)^2 tP_2 + 3(1-t)t^2 P_3 + t^3 P_4$$

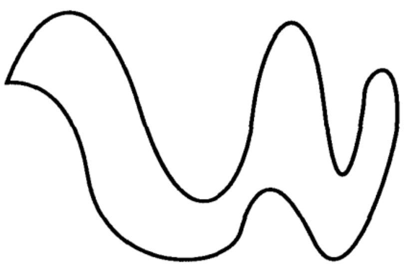
Path

- A single Bezier curve on its own is rarely something we want in a drawing
- What makes Bezier curve useful is the ease with which they can be combined to make more **elaborate curves** and **irregular shapes**
- A collection of lines and curves is called a **path**





an open path

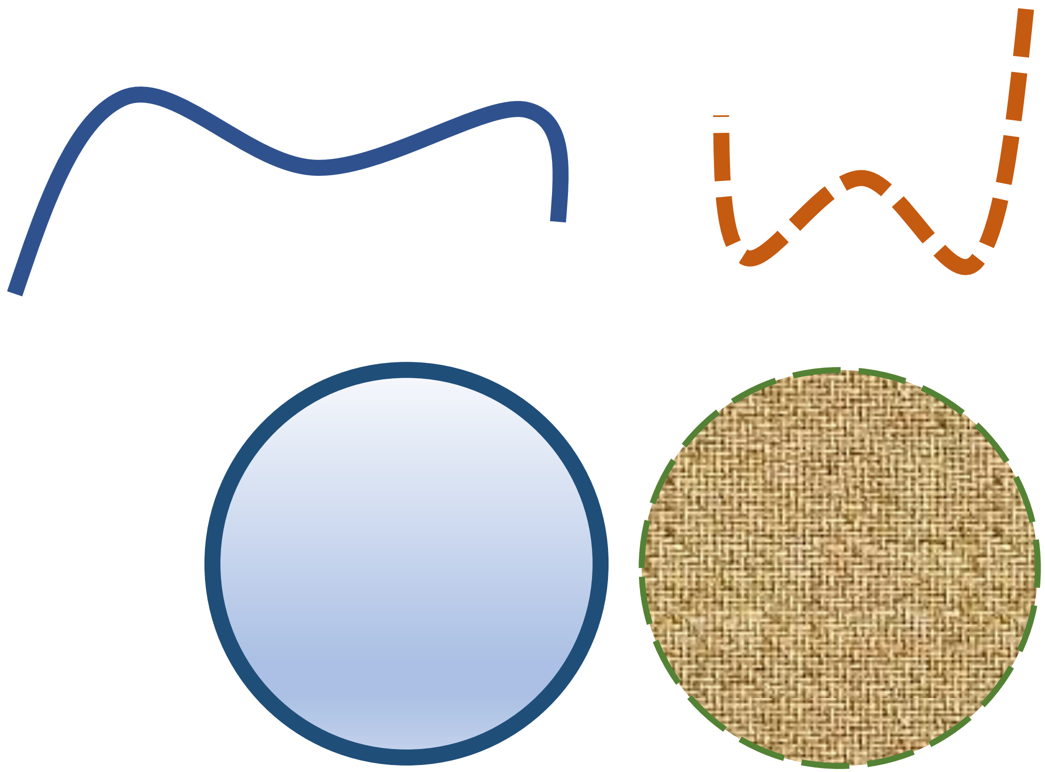


a closed path

Stroke and Fill

Stroke and Fill

- Mathematically a path is infinitesimally thin because points are infinitesimally small
- Two ways to make a path visible
  - **Stroke**
    - Weight (width)
    - Color
    - Dashed
  - **Fill**
    - Single color
    - Gradient
    - Patterns



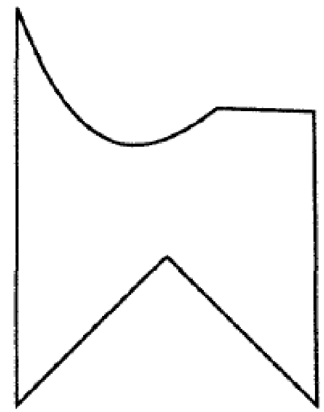
Transformation

Transformation of Vector Graphics

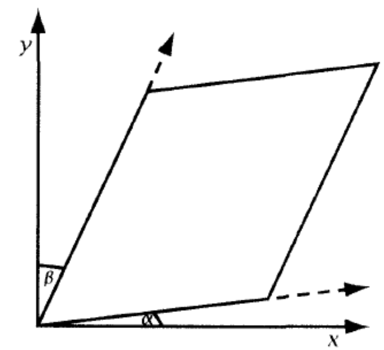
- The actual pixel values that makes up a vector image needs to be computed until it is displayed
- We can transform the image by **editing the model** of the shape stored in the computer
  - Transform the control points or parameters
- Example: move a line segment: (4, 2) ⬄ (10, 2) up by 5 units
  - Add 5 units to the y-coordinates
  - Produce a new line segment: (4, 7) ⬄ (10, 7)

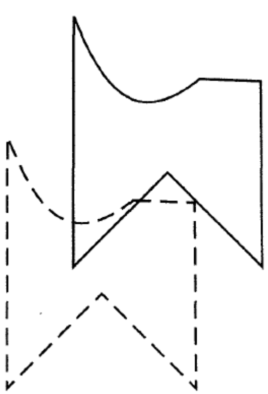
Transformation of Vector Graphics (cont.)

- Types of transformation
  - Translation
  - Scaling
  - Rotation (about a point)
  - Reflection (about a line)
  - Shearing

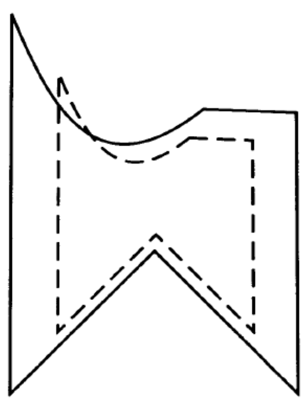


origin

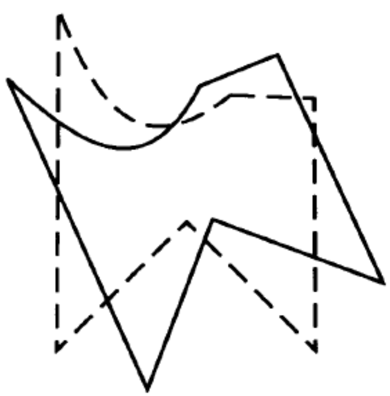




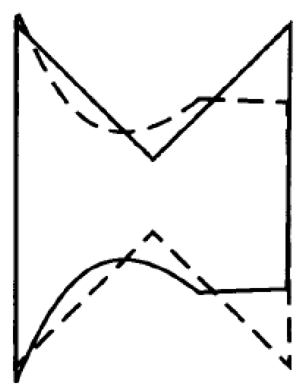
translation



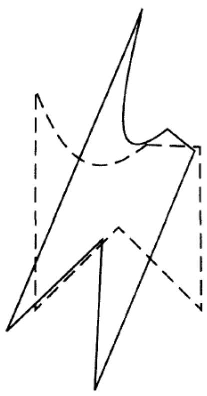
scaling



rotation



reflection



shearing

Transformation of a Point

- Transformation of a point can be represented by the multiplication of a **column vector (point, 3 x 1)** and a **transformation matrix (3 x 3)**

transform matrix

$$p' = Mp$$

new point — original point

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}_{p'} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}_{p}$$

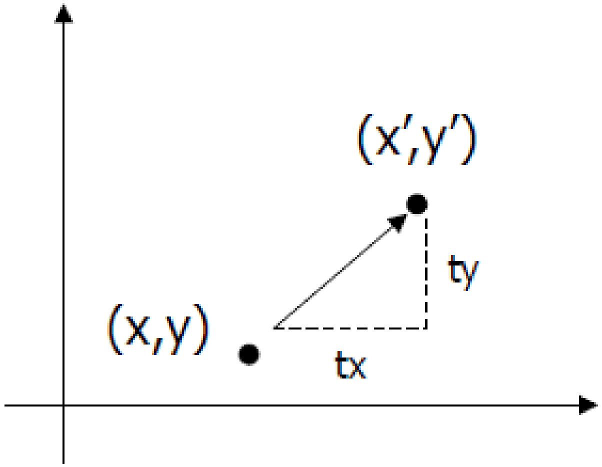
$$x' = ax + by + c$$

$$y' = dx + ey + f$$

Translation

- Given a point ***p(x, y)*** and a translation distance ***T(t_x, t_y)***, the new point p' after translation is ***p' = p + T***

$$x' = x + t_x$$
$$y' = y + t_y$$



- Matrix-vector multiplication

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

Scaling

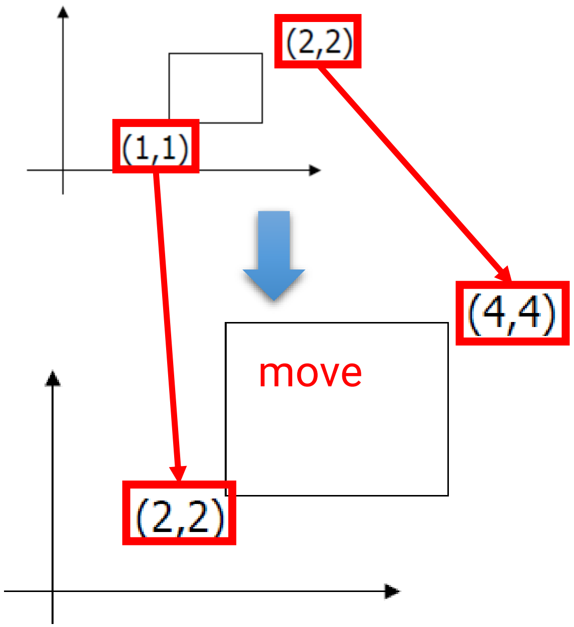
- Given a point ***p(x, y)*** and a scaling factor ***S(s_x, s_y)***, the new point p' after scaling is ***p' = S p***

$$x' = x * s_x$$

$$y' = y * s_y$$

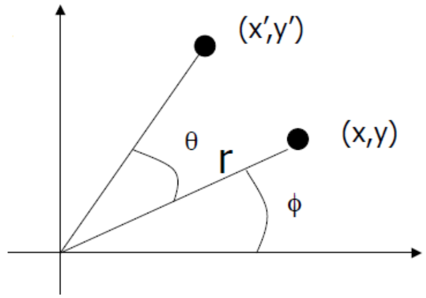
- Matrix-vector multiplication

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

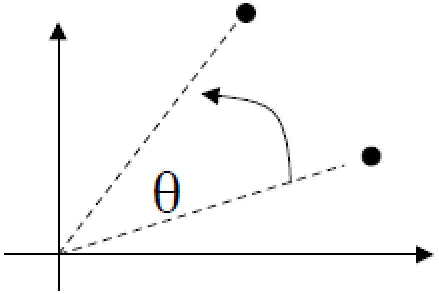


Rotation

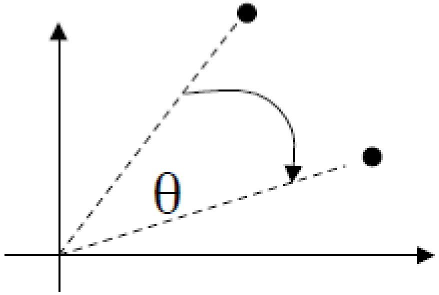
- Given a point ***p(x, y)***, rotate it with respect to the **origin** by ***θ*** and get the new point ***p'*** after rotation



- First define



θ > 0: rotate counterclockwise

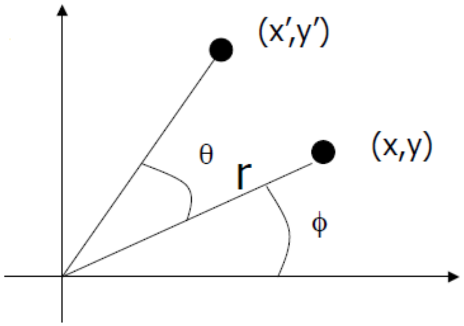


θ < 0: rotate clockwise

Rotation (cont.)

- Given a point ***p(x, y)***, rotate it with respect to the **origin** by ***θ*** and get the new point ***p'*** after rotation

$$x = r\cos(\phi) \qquad y = r\sin(\phi)$$
$$x' = r\cos(\phi + \theta) \qquad y' = r\sin(\phi + \theta)$$



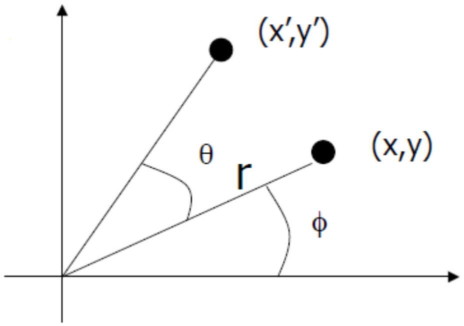
$$\begin{aligned} x' &= r\cos(\phi + \theta) \\ &= r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta) \\ &= x\cos(\theta) - y\sin(\theta) \end{aligned}$$

$$\begin{aligned} y' &= r\sin(\phi + \theta) \\ &= x\sin(\phi)\cos(\theta) + r\cos(\phi)\sin(\theta) \\ &= y\cos(\theta) + x\sin(\theta) \end{aligned}$$

Rotation (cont.)

- Given a point ***p(x, y)***, rotate it with respect to the **origin** by ***θ*** and get the new point ***p'*** after rotation

$$\begin{aligned} x' &= r\cos(\phi + \theta) \\ &= x\cos(\theta) - y\sin(\theta) \\ y' &= r\sin(\phi + \theta) \\ &= y\cos(\theta) + x\sin(\theta) \end{aligned}$$



- Matrix-vector multiplication

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

Put it All Together

- Translation

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

- Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

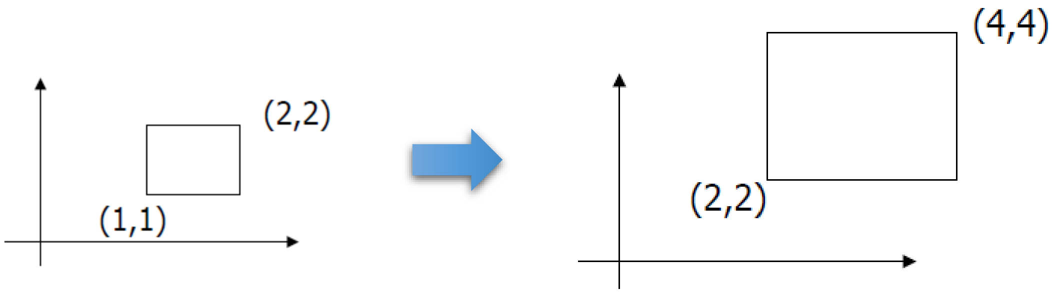
- Rotation

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

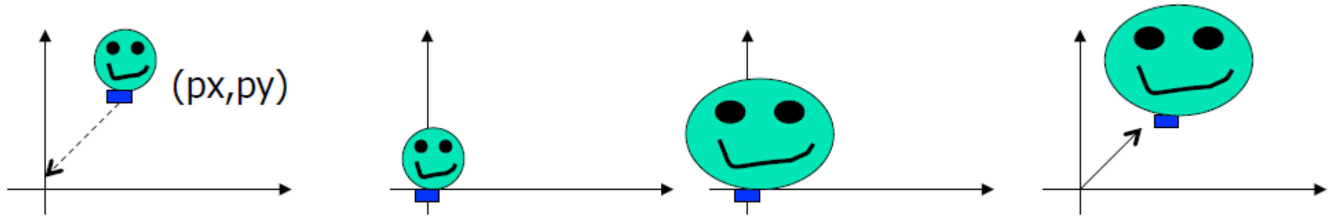
- Using 3x3 matrix allows us to perform all transformations using matrix/vector multiplications
  - We can also **pre-multiply** all the matrices together
- We call the (x, y, 1) representation for (x, y) **homogeneous coordinate**

Scaling Revisit

- The standard scaling matrix will only anchor at (0, 0)

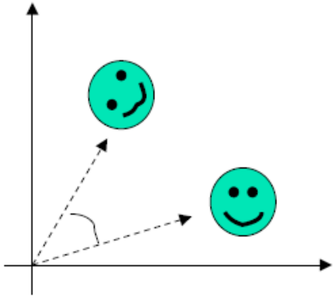


- Scaling about an arbitrary pivot point $\boldsymbol{Q(q_x, q_y)}$
  - Translate the objects so that Q will coincide with the origin: $T(-q_x, -q_y)$
  - Scale the object: $S(s_x, s_y)$
  - Translate the object back: $T(q_x, q_y)$



Rotation Revisit

- The standard rotation matrix is used to rotate about the origin (0, 0)



- Rotate about an arbitrary pivot point ***Q(q_x, q_y)*** by ***θ***
  - Translate the objects so that Q will coincide with the origin: ***T($-q_x$, $-q_y$)***
  - Rotate the object: ***R(θ)***
  - Translate the object back: ***T(q_x, q_y)***