Fundamentals

Hardware

- a computer screen is made up of a rectangular grid of *pixels*
- what is to be displayed on the screen is stored in the *frame buffer*
  - a block of memory containing a color value for each pixel
  - essentially a 2D array

direct color		**indexed color**	
1 bit	monochrome 2 different colors	4 bit palettized	16 different colors
16 bit (“high color”)	RGB components, 5 bits each (extra bit may be ignored, added to green, or used for alpha) 32,768 or 66,560 different colors	8 bit palettized	256 different colors
24 bit (“true color”)	RGB components, 8 bits each 16,777,216 different colors		

2D Graphics – Representing Images


Raster (PNG)

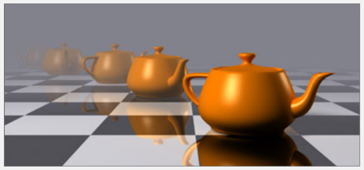

Vector (SVG)

- *raster image* – specify color for each pixel
  - formats: GIF, PNG, JPG, …
  - create/manipulate with painting programs
  - resolution dependent
- *vector graphics* – specify the geometric objects contained in the picture
  - formats: SVG, EPS, …
  - create/manipulate with drawing programs
  - can be a much more compact representation
  - scales well
  - not suitable for all kinds of images
  - requires *rasterization* step for display

https://commons.wikimedia.org/wiki/File:Orc_-_Raster_vs_Vector_comparison.png

3D Graphics

- 3D graphics is characterized by 3D geometry for the objects in the scene
  - a *scene* includes the geometry (position, orientation, size) and material properties (color, shininess, roughness, transparency, etc) of the objects in the scene
- the final display is inherently 2D
  - requires *rendering* step to compute a 2D image from 3D geometry for display



https://commons.wikimedia.org/wiki/File:CGfog.jpg

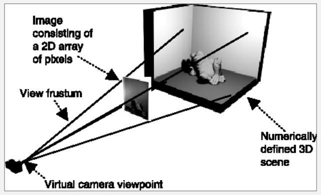
3D Graphics

Three main aspects –

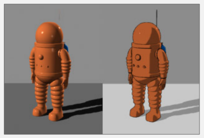
- modeling – defining the scene (geometry and materials)
- rendering – producing an image
- animation

3D Graphics – Rendering

Three steps –



- *viewing* – position a virtual camera in the world
- *projection* – map 3D coordinates into 2D
- assign colors to pixels
  - take into account lighting and materials
  - goal is often photorealism, but it doesn't have to be



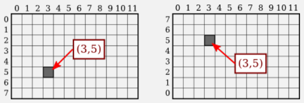
https://www.semanticscholar.org/paper/Computer-Graphics-Techniques-for-Capturing-and-the-Rushmeier/45ac9cd63624b4f7edc4a6897b7692c39749dbee https://commons.wikimedia.org/wiki/File:Toon_Shader.jpg

Coordinate Systems

- a *coordinate system* is a way of assigning numbers to points in space
  - can have 2D, 3D, etc – refers to how many numbers are needed to identify a point

Pixel Coordinates

- (row,col) identifies a location within a 2D array
  - integer values, >= 0

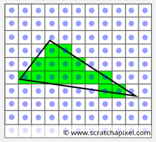


  - conventions differ as to whether (0,0) is at the top left or bottom left
    - most graphics systems (e.g. Java Graphics2D) use top left
    - OpenGL uses bottom left

http://math.hws.edu/graphicsbook/c2/s1.html

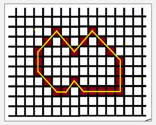
Pixel Coordinates

- pixel coordinates identify a pixel, not a point
  - pixels have area, points do not

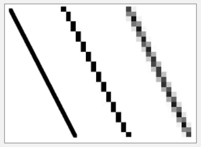


This creates challenges for rasterization.

- determining which pixels to color
  - preserve the correct dimensions
  - avoid gaps or double-colored pixels with adjacent shapes



- "jaggies" (*aliasing*)
  - *antialiasing* techniques try to mitigate the effects by using shades of gray proportional to how much of the pixel's area is covered by the primitive



https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage.html
https://www.youtube.com/watch?app=desktop&v=Pavnw8nhiyl
http://math.hws.edu/graphicsbook/c2/s1.html

Pixel Coordinates

Other drawbacks.

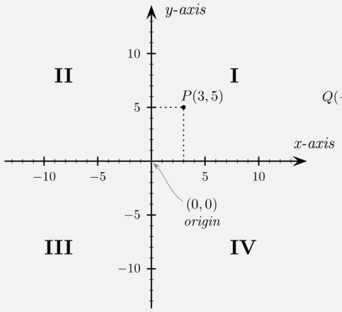
- physical screen pixels are different sizes
  - a 100x100 pixel image will be much smaller on a high-resolution screen than a low-resolution one
- raster images don't scale
  - integer pixel coordinates do not necessarily scale to integer pixel coordinates
  - can suffer from spatial aliasing

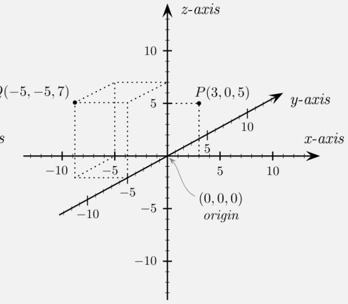
Spatial Aliasing

https://en.wikipedia.org/wiki/Aliasing

Real-Number Cartesian Coordinate Systems

- defined by an *origin* and *n* perpendicular axes





https://commons.wikimedia.org/wiki/File:Cartesian_coordinates_2D.svg
https://commons.wikimedia.org/wiki/File:Cartesian_coordinates_3D.svg

Real-Number Cartesian Coordinate Systems

Advantages over pixel coordinates –

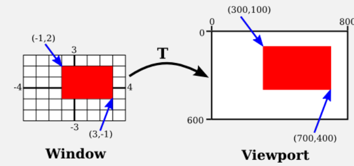
- units can be whatever is convenient rather than being determined by the hardware
- no difficulties with scaling

Coordinate Systems

But the coordinates used in the scene must be transformed into pixel coordinates for display.

world coordinates (real number Cartesian coordinates)

where the scene is defined



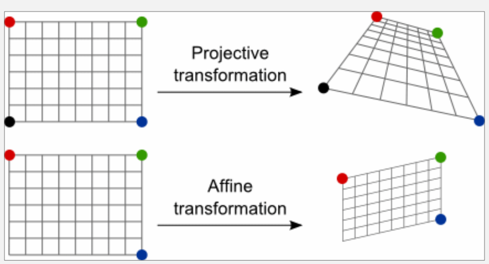
screen coordinates (pixel coordinates)

where things appear on the screen

http://math.hws.edu/graphicsbook/c2/s3.html

Affine Transformations

- an *affine transformation* is a geometric transformation where parallel lines remain parallel
  - lines can be transformed simply by transforming their endpoints



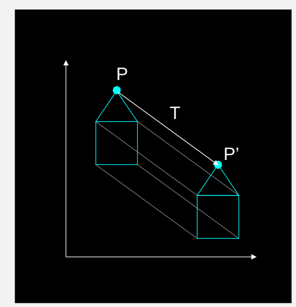
https://www.graphicsmill.com/docs/gm/affine-and-projective-

Translation

- change object position

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

- properties
  - preserves lengths
  - preserves angles

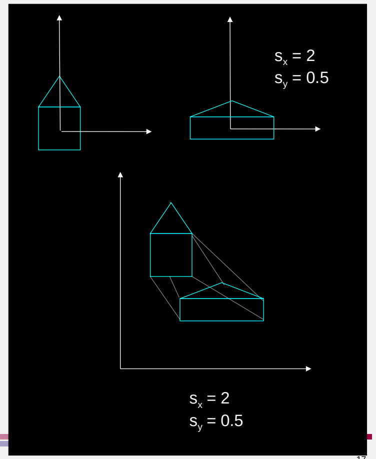


Scaling

- change object size

  $x' = s_x\, x$
  $y' = s_y\, y$

- properties
  - position shifted relative to origin (fixed point)
  - lengths not preserved
  - angles preserved only for uniform scaling



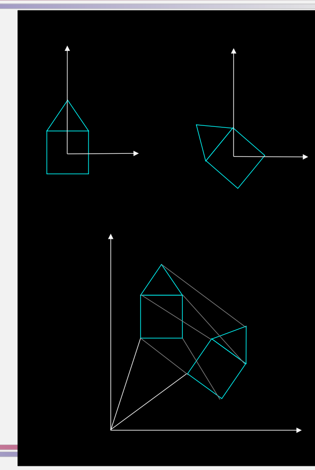
Rotation

- change object orientation

  $x' = x \cos\theta - y \sin\theta$
  $y' = x \sin\theta + y \cos\theta$

  (counterclockwise rotation)

- properties
  - position is shifted relative to origin (pivot point)
  - preserves lengths
  - preserves angles



Shear

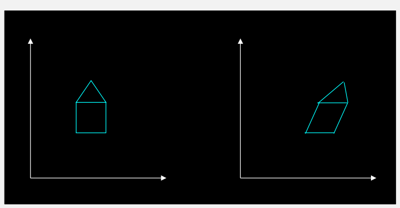
- points are shifted by amount relative to distance from axis

$x' = x + sh_x\, y$
$y' = y$

shear relative to x-axis

$x' = x$
$y' = sh_y\, x + y$

shear relative to y-axis



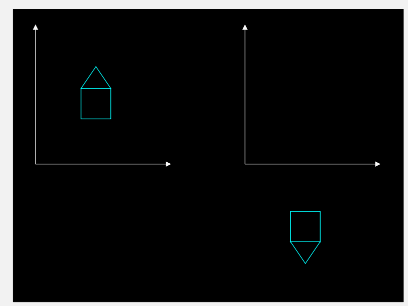
Reflection

$x' = x$
$y' = -y$

reflection about the x-axis

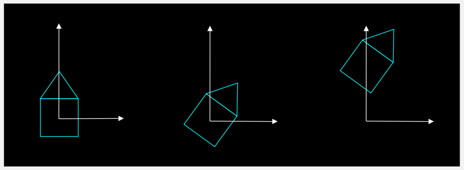
$x' = -x$
$y' = y$

reflection about the y-axis



Combining Transformations

- order matters



rotate, then translate

- in graphics systems, the current transformation applies to everything done after it
  - steps are written in reverse order
  - effect is as if the last transformation is applied first

```
translate(0,10)
rotate(-45)
draw house
```