CS7.302
Computer Graphics
Module: Introduction

G S Ragavendra P. J. Narayanan

Spring 2025

What is Computer Graphics?

- Techniques (mathematics, physics, algorithms) to generate realistic images on the computer
- How?
  - Representations (“models”) of the world of interest
  - Algorithms to produce ultra-realistic images based on underlying physics
  - Do all of it fast to please users
- **Computational Process:**

  Abstract – Represent – Process – Reproduce
- Digital or computer revolution: Applying this successfully to different areas of human interest

Digitial or Computer Revolution

- Changed the world greatly in the past 20-30 years!
- How? "**Digitize**" different things/concepts/ideas/...
  – Digital preservation, replication, etc., are very cheap
- Initially: Ease tediums or difficulty of activities
  – Aircraft design, payroll, Efficient book-keeping, etc.
- Later: Improve and transform the process
  – Electronic account-books to networked banks to online banking to virtual money to ...
- Enablers: Digital Representation, Efficienty Processing and Manipulation, Quick Communication
- And ... **reversing** the digitization process

Some Computational Processes

- Music: Digitize using microphones and analog-to-digital conversion, process to remove noise, store/transmit as MP3 files, playback using D-to-A and speakers
  – Similarly, Video, Teleconferencing, etc.
- Weather prediction: Capture parameters from locations, apply metereological models, process at different levels of detail, predict
  – Drug design, molecular dynamics, more science
- Computer games: World and its rules set by designer, some aspects controlled by players, interaction with objects according to rules, show results to players
  – Several simulations, Virtual Reality, etc.

What's this?

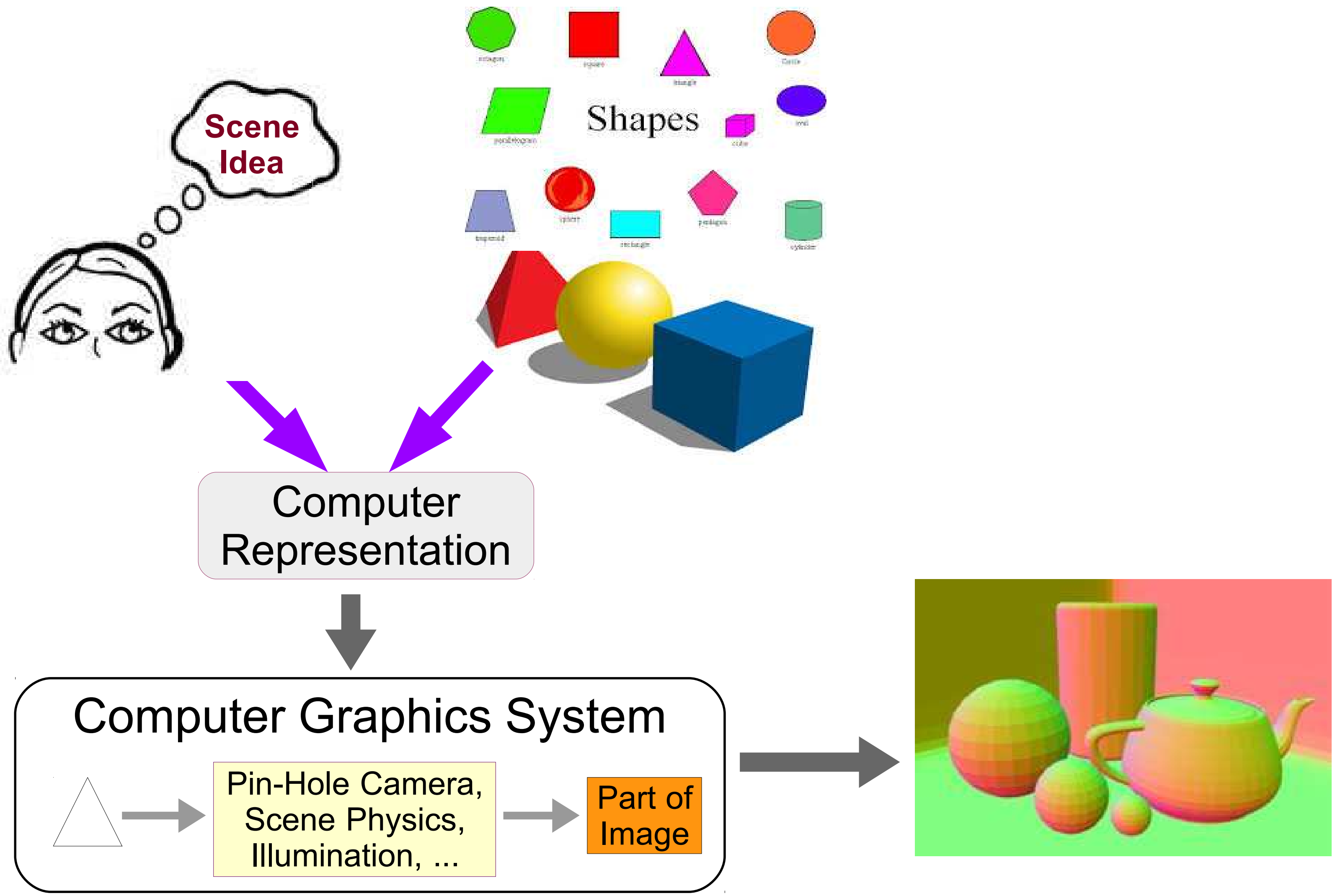
Counter Strike 2

Producing Realistic Images

- Represent physical world using basic "primitives"
  - Basic geometric shapes and objects
  - Efficiency vs utility: Use simple and useful primitives
  - Break up complex scenes into available components
  - Efficiency: **Smallness in size** and **ease of operation**
- Process of image generation from the representation
  - Ape the best that we know: Human eyes
  - (Digital) Cameras approximate the eye in our world
  - Pin-hole camera model approximates the eye conceptually
  - Mathematics of pin-hole cameras known
  - Apply pin-hole camera computationally

- Transform a primitive to a camera image correctly
  - Apply pin-hole camera model on the primitive
  - A series of computations to map primitive to image
  - Paint the picture based on physical properties of objects
- Apply the same to the scene consisting of primitives
  - Evaluate how multiple primitives interact or interfere
  - Paint the physically correct picture of the whole scene
- Do all this efficiently
  - Millions of primitives. High resolution images
  - Complex objects with fine structure and properties
  - Update image fast for application. Real-time for games!

Graphics Process



Some Questions and Concerns

- What geometric primitives to use?
  – Geometric shapes exist. Economy & Utility important
- How do we represent them on the computer?
- How do we create representations of real-world objects?
- How do we manipulate the computer-resident world?
  – Change shape, size, appearance, etc.
- How do we produce images or views of scenes?
  – Realism is important
- How do we do all these efficiently and quickly?

Application Areas

- User interfaces
- Computer aided design (Civil/Mech/VLSI)
- Visualization of scientific & engineering data
- Art
- Virtual Reality
- Entertainment: Great computer games!
- Special effects in movies. **Whole movies themselves!!**
- ...

Quick History

- Whirlwind Computer (1950) from MIT had computer driven CRTs for output.
- SAGE air-defense system (mid 50s) had CRT, lightpen for target identification.
- Ivan Sutherland's Sketchpad (1963): Early interactive graphics system.
- CAD/CAM industry saw the potential of computer graphics in drafting and drawing.
- GE's DAC system (1964), Digitek system, etc.
- Systems were prohibitively expensive and difficult to use.

- Special display processors or image generators were used for high-end graphics.
- Workstations by Silicon Graphics: early eighties.
- Graphics was expensive, escoteric, and hence rare!
- **A parallel:** Computing became “popular” only after mass-produced personal computers became a reality in mid 80s. Before that, bulky, expensive, and rare devices.
- **Circle of Computing Revolution:** *More users* lead to *greater revenues/returns* which affords *more research* which result in *better/cheaper computers* which in turn bring *yet more users*. And this continues!!

Popular Graphics

- Graphics became “popular” only after mass-produced *Graphics Processing Units* (**GPU**s) or *graphics accelerators* came into existence.
- Graphics Accelerators: on board hardware to speed up graphics computations.
- Accelerators were expensive until end nineties!
- Very high end performance is available economically today. Getting part of the CPU chip these days.
- **Computer Games** provide the fuel for fast growth

Graphics Programming

- Device dependent graphics in early days.
- 3D Core Graphics system was specified in SIGGRAPH 77. (Special Interest Group on Graphics)
- GKS (Graphics Kernel System): 2D standard. ANSI standard in 1985.
- GKS-3D: 1988.
- PHIGS: Programmer's Hierarchical Interactive Graphics System. (ANSI 1988)

- **OpenGL**: current ANSI standard.
  - Evolved from SGI's GL (graphics library).
  - Window system independent programming.
  - GLUT (utility toolkit) for the rest.
  - Popular. Many accelerators support it.
- DirectDraw/Direct3D: Microsoft's attempt at it!
- **WebGL:** OpenGL to be used for web programming that is now gaining popularity
- **OpenGL ES:** Slightly reduced version for mobile devices, which will be the prime computing platform
- Desirable: High level toolkits.

Course Content

- 2D & 3D Graphics: Concepts, Mathematics, Hierarchical Modelling, Algorithms. Practice in OpenGL.
- Representation: Lines & Curves, Surfaces, Solids.
- Drawing algorithms: Primitives, visibility, efficienty
- Lighting and Shading: Simluating the physics of image generation
- Ray Tracing: If we get time

Background Required

- Good programming skills in C/C++.
- Geometry: Points, vectors, matrices, transformations, etc.
- Data structures.
- Java for Web or Mobile graphics
- **Good imagination. Ability to visualize in 3D**

Text Books and Reference

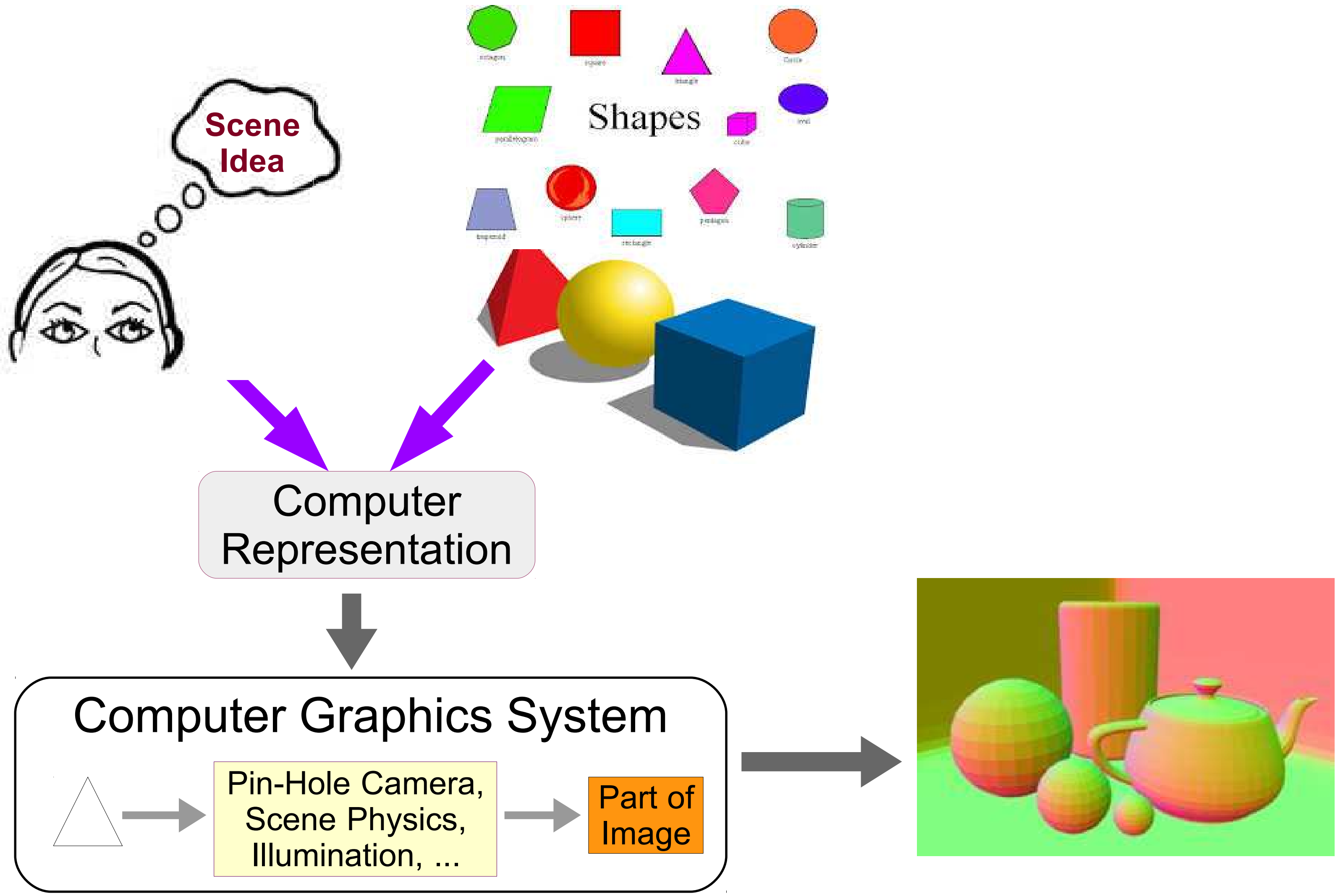
- Interactive Computer Graphics by Edward Angel and Shreiner
- Computer Graphics with OpenGL by Hearn and Baker, Third edition. Indian Edition available.

- Computer Graphics: Principles & Practice by Foley, van Dam, Feiner, Hughes. Indian Edition available.
- OpenGL Programming Guide by Neider, et. al.

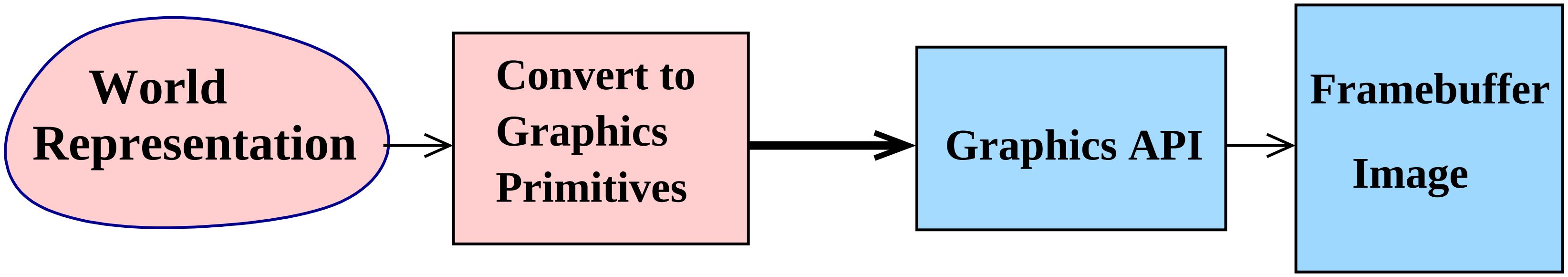
Course Management

- Homework assignments, Programming assignments, lab test, mid-term test, final exam
- Weightages of different components:
  $\approx$ 40% for the final exam
  $\approx$ 50% for the assignments **Subject to change**
  $\approx$ 10% for quiz
- This course involves a lot of fun programming! Enjoy it!
- Several students do much more in assignments than asked for. Let your creative juices flow!!

Graphics Process

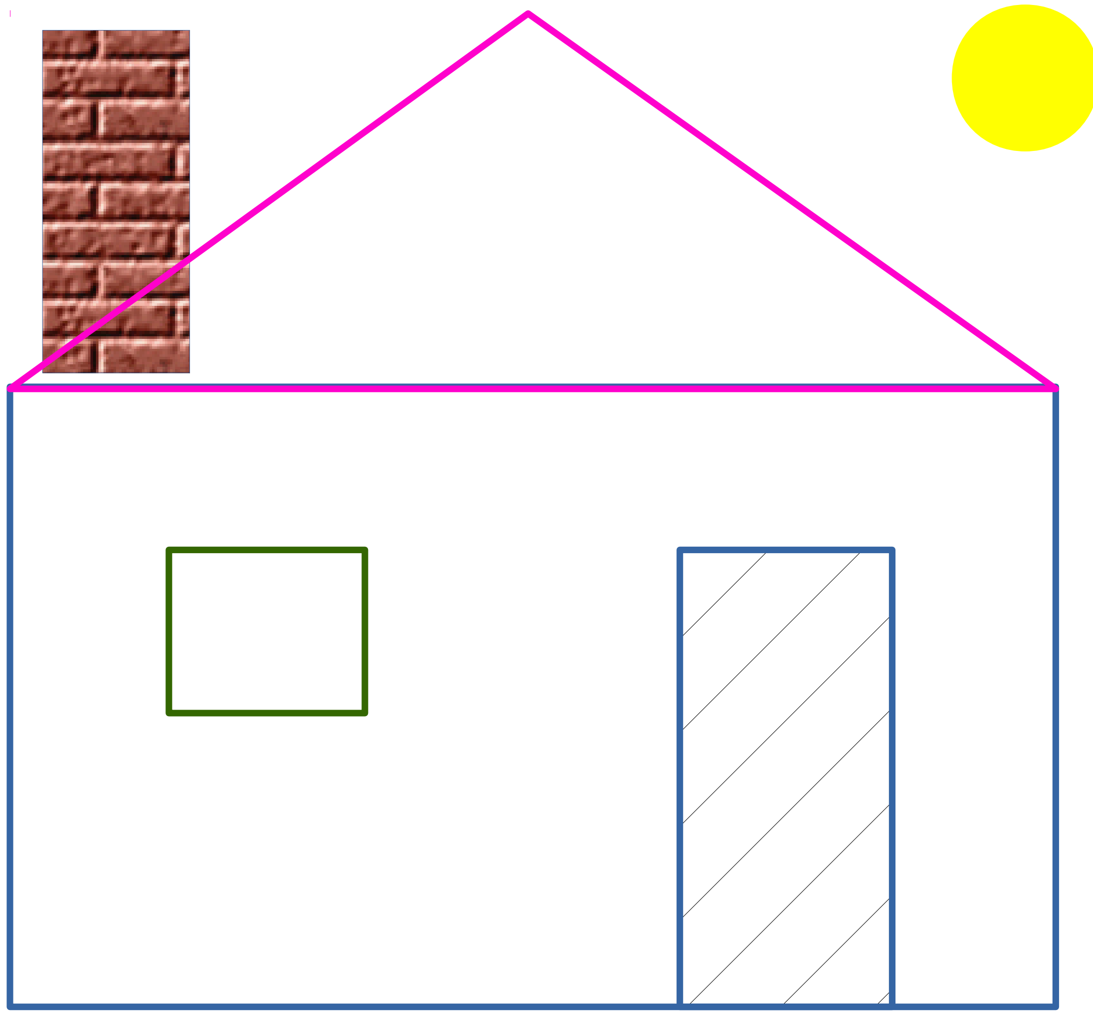


Graphics Process



- **Model** the desired world in your head.
- **Represent** it using natural structures in the program. Convert to standard primitives supported by the API
- **Processing** is done by the API. Converts the primitives in stages and forms an image in the framebuffer
- The image is displayed automatically on the device

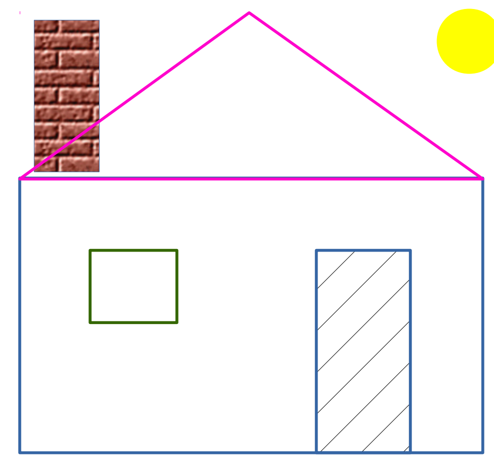
How to Draw A House?



Drawing A House

- Compose using basic shapes

```
// Main part
drawRectangle(v1, v2, v3, v4);
drawTriangle(v2, v3, v5);      // Roof
drawRectangle( ... );          // Door
drawRectangle( ... );        // Window
drawRectangle( ... );       // Chimney
drawCircle( ... );                // Sun
```



- That's all there is, really!

Graphics Primitives

- Graphics is concerned with the **appearance** of the 3D world to a camera
- Only *outer surface* of objects important, not interiors!!
- Hence, uses only 1D and 2D primitives

- Points: 2D or 3D. (x, y) or (x, y, z).
- Lines: specified using end-points
- Triangles/Polygons: specified using vertices
- Why not **circles, ellipses, hyperbolas**?

Graphics Attributes

- Colour, Point width.
- Line width, Line style, Line Colour.
- Fill, Fill Pattern.

- Line: Give two endpoints
- Triangle: Give three vertices
- **Point** is the most basic primitive

Point Representation

- A point is represented using 2 or 3 numbers $(x, y, [z])$ that are the projections on to the respective coordinate axes.
  – Could also be reprsented as a 2 or 3 vector $\mathbf{P}$.
- Fundamental shape-defining primitive in most Graphics APIs. Everything else is built from it!
- Represented using byte, short, int, float, double, etc.
- The scale and unit are application dependent.
  Could be **angstroms** or **lightyears**!
- Points undergo transformations:
  Translations, Rotations, Scaling, Shearing.

3D Coordinates

– Vector $\mathbf{P}$

– Cartesian: (x, y, z)

– Polar: (ρ, θ, ϕ)

– $z =$

  $y =$

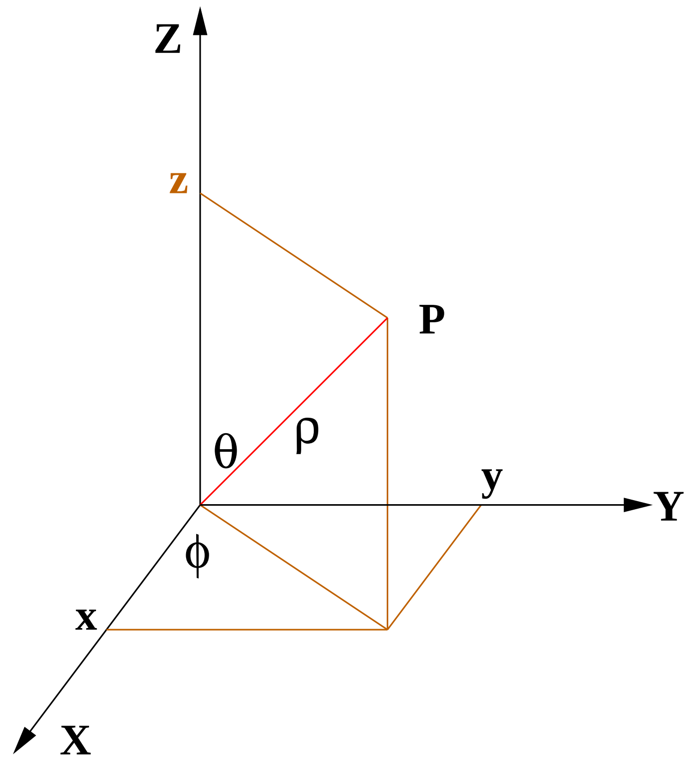
  $x =$

– $\rho =$

  $\phi =$

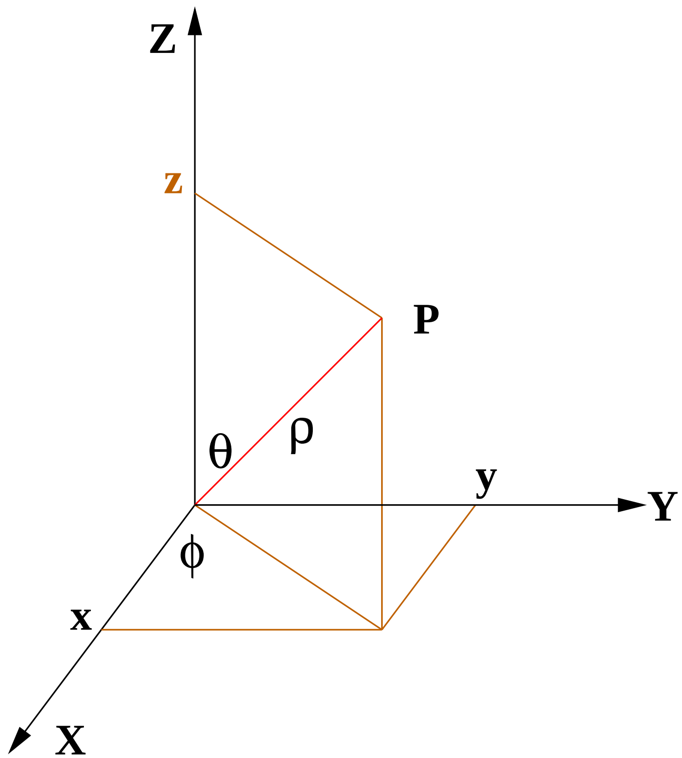
  $\theta =$

– Elevation: θ, Azimuthal: ϕ



3D Coordinates

- Vector $\mathbf{P}$
- Cartesian: (x, y, z)
- Polar: (ρ, θ, ϕ)
- $z = \rho \cos\theta,$
  $y = \rho \sin\theta \sin\phi$
  $x = \rho \sin\theta \cos\phi$
- $\rho^2 = x^2 + y^2 + z^2,$
  $\phi = \tan^{-1}(y/x),$
  $\theta = \tan^{-1}(\sqrt{x^2 + y^2}/z)$
- Elevation: θ, Azimuthal: ϕ



Translation

- Translate a point $\mathbf{P} = (x, y, [z])$ by $(a, b, [c])$.
- Point's coordinates become $\mathbf{P}' = (?, ?, ?)$.
- In vector form, $\mathbf{P}' = ?$.

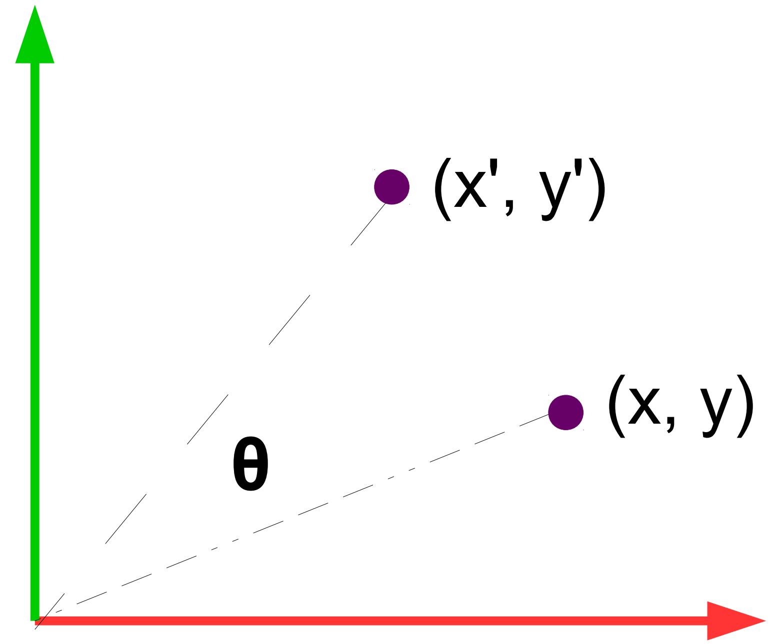
Translation

- Translate a point $\mathbf{P} = (x, y, [z])$ by $(a, b, [c])$.
- Points coordinates become $\mathbf{P}' = (x + a, y + b, [z + c])$.
- In vector form, $\mathbf{P}' = \mathbf{P} + \mathbf{t}$, where $\mathbf{t} = (a, b, [c])$.
- Distances, angles, parallelism are all maintained.

2D Rotation

- Rotate about origin CCW by θ.
- $x' = ?, \quad y' = ?$
- Matrix notation: $\mathbf{P}' = \mathbf{R}\ \mathbf{P}$

$$\begin{bmatrix} x \\ y \end{bmatrix}' = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



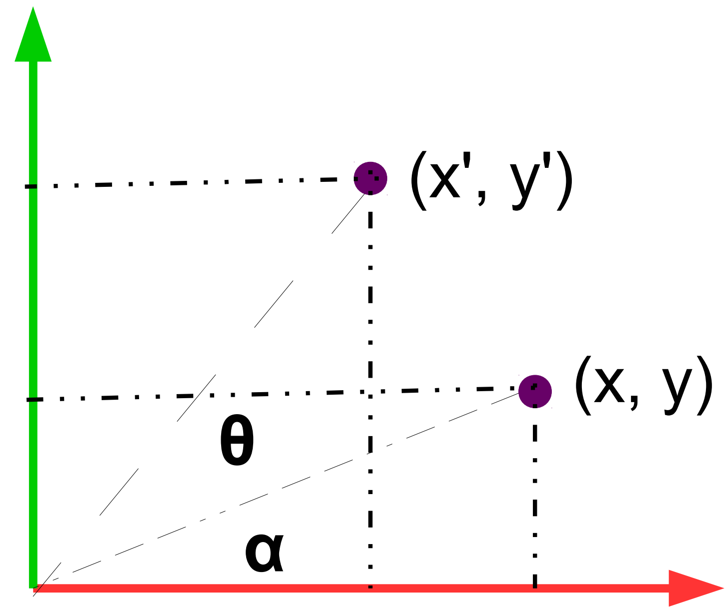
2D Rotation

– Rotate about origin CCW by θ.

– $x' = ?, \quad y' = ?$

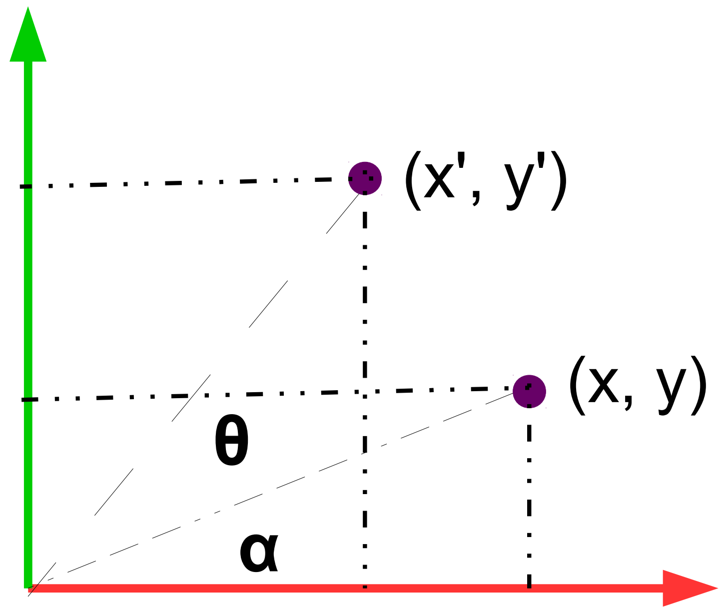
– Matrix notation: $\mathbf{P}' = \mathbf{R}\,\mathbf{P}$

$$\begin{bmatrix} x \\ y \end{bmatrix}' = \begin{bmatrix} ? & ? \\ ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



2D Rotation

- Rotate about origin CCW by θ.
- $x' = x\cos\theta - y\sin\theta$,
  $y' = x\sin\theta + y\cos\theta$.



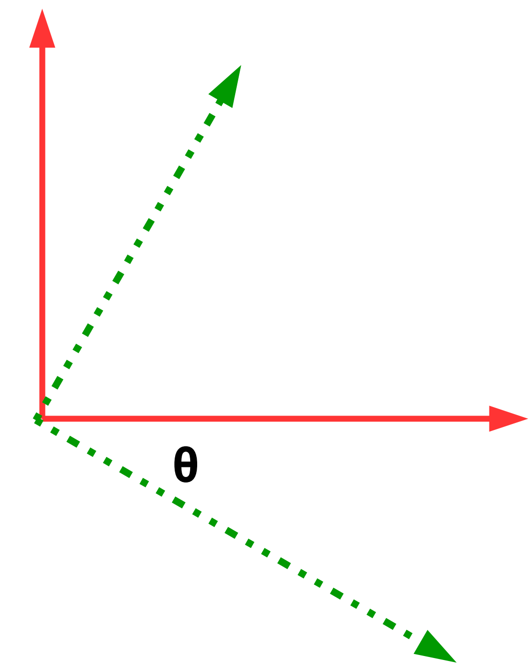
- Matrix notation: $\mathbf{P}' = \mathbf{R}\ \mathbf{P}$

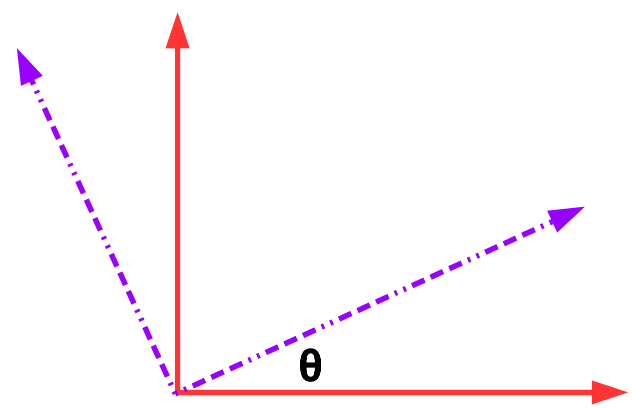
$$\begin{bmatrix} x \\ y \end{bmatrix}' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Rotation: Observations

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

- Orthonormal: $\mathbf{R}^{-1} = \mathbf{R}^{\mathbf{T}}$
- Rows: vectors that **rotate to** coordinate axes
- Cols: vectors coordinate axes **rotate to**
- Invariants: distances, angles, parallelism.





3D Rotations

- Rotation could be about any axis in 3D! What does it mean?
  - Distance of each point to the axis of rotation remains same.
  - Each points moves in a circle on a plan perpendicular to the axis of rotation, with the centre on the axis
- About Z-axis: Just like 2D rotation case. Z-coordinates don't change anyway.
- X-Y coordinates change exactly the same way as in 2D.
- CCW +ve, looking into the **arrowhead:** $\mathbf{R_z}(\theta) = ??$

3D Rotations

- Rotation could be about any axis in 3D!
- About Z-axis: Z-coordinates don't change anyway

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

- CCW +ve; orthonormal; length preserving
- Rows: vectors that rotate onto axes; columns: vectors that axes rotate into....

3D Rotations

$$\mathbf{R_y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \qquad \mathbf{R_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

- CCW +ve; orthonormal
- Rows: vectors that rotate onto axes; columns: vectors that axes rotate into....
- Rotation about an arbitrary axis, for example, $[1, 1, 1]^{\mathsf{T}}$??
  Coming soon ....

Non-uniform Scaling

- Scale along X, Y, Z directions by s, u, and t.
- $x' = s\,x, \;\; y' = u\,y, \;\; z' = t\,z$.
- We are more comfortable with $\mathbf{P}' = \mathbf{S}\ \mathbf{P}$ or

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}' = \begin{bmatrix} s & 0 & 0 \\ 0 & u & 0 \\ 0 & 0 & t \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Invariants: parallelism, ratios of lengths in any direction (Angles also for uniform scaling.)

Shearing

- Add a little bit of x to y or other combinations

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}' = \begin{bmatrix} 1 & x_y & x_z \\ y_x & 1 & y_z \\ z_x & z_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- One of $x_y, x_z, y_x, y_z, z_x, z_y \neq 0$. Rectangles can become parallelograms, but not general quadrilaterals

- Invariants: parallelism, ratios of lengths in any direction.

Reflection

- Negative entries in a matrix indicate reflection.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}' = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- Reflection needn’t be about a coordinate axis alone

Summary of Transformations

- Translation: New coordinates $\mathbf{P}' = \mathbf{P} + \mathbf{t}$
- Rotation: $\mathbf{P}' = \mathbf{R}\ \mathbf{P}$
- Scaling: $\mathbf{P}' = \mathbf{S}\ \mathbf{P}$
- Shearing: $\mathbf{P}' = \mathbf{S_h}\ \mathbf{P}$
- Reflection: $\mathbf{P}' = \mathbf{R_f}\ \mathbf{P}$
- Each is a matrix-vector product, except ....

General Transformation

- Rotation, scaling, shearing, and reflection: **Matrix-vector** product. Vectors get tranformed into other vectors
- Translation alone is a **vector-vector** addition
- Sequence of: Translation, rotation, scaling, translation and rotation: $\mathbf{P'} = \mathbf{R_2}\left[\mathbf{S}\,\mathbf{R_1}\left(\mathbf{P} + \mathbf{t_1}\right) + \mathbf{t_2}\right]$
- If translation is also a matrix-vector product, we can combine above transformations into a single matrix: $\mathbf{P'} = \mathbf{R_2}\,\mathbf{T_2}\,\mathbf{S}\,\mathbf{R_1}\,\mathbf{T_1}\,\mathbf{P} = \mathbf{M}\,\mathbf{P}$
- How? Answer: **homogeneous coordinates**!

Homogeneous Coordinates

- Add a *non-zero scale factor* w to each coordinate.
  A 2D point is represented by a vector $[x \;\; y \;\; w]^{\mathsf{T}}$
- $[x \;\; y \;\; w]^{\mathsf{T}} \equiv (x/w,\; y/w)$.
- Simplest value of w is obviously 1
- Translate $[x \;\; y]^{\mathsf{T}}$ by $[a \;\; b]^{\mathsf{T}}$ to get $[x+a \;\; y+b]^{\mathsf{T}}$

$$\begin{bmatrix} x+a \\ y+b \\ 1 \end{bmatrix} = \begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous Coordinates

- Add a *non-zero scale factor* w to each coordinate.
  A 2D point is represented by a vector $[x \;\; y \;\; w]^\mathsf{T}$
- Translate $[x \;\; y]^\mathsf{T}$ by $[a \;\; b]^\mathsf{T}$ to get $[x+a \;\; y+b]^\mathsf{T}$

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

- Now, translation is also: $\mathbf{P}' = \mathbf{T}\;\mathbf{P}$, a matrix-vector product and a linear operation.

Homogeneous Coordinates

- Add a *non-zero scale factor* w to each coordinate.
  A 2D point is represented by a vector $[x \;\; y \;\; w]^\mathsf{T}$
- $[x \;\; y \;\; w]^\mathsf{T} \equiv (x/w,\; y/w)$.
- Now, translation is also: $\mathbf{P'} = \mathbf{T}\;\mathbf{P}$
- For a point: Rotation followed by translation followed by scaling, followed by another rotation: $\mathbf{P'} = \mathbf{R_2STR_1}\;\mathbf{P}$.
- Similarly for 3D. Points represented by: $[x\; y\; z\; w]^T$.
- All matrices are 3×3 in 2D. Last row is $[0 \;\; 0 \;\; 1]$.
- All matrices are 4×4 in 3D. Last row is $[0 \;\; 0 \;\; 0 \;\; 1]$.

Homogeneous Representation

- Convert to a 4-vector with a scale factor as fourth. $(x, y, z) \equiv [kx \;\; ky \;\; kz \;\; k]^{\mathsf{T}}$ for any $k \neq 0$.

- Translation, rotation, scaling, shearing, etc. become linear operations represented by 4×4 matrices.

- What does $[x \; y \; z \; 0]^{\mathsf{T}}$ mean?

- $[a \; b \; c \; d]^{\mathsf{T}}$ could be a point or a plane. Lines are specified using two such vectors, either as join of two points or intersection of two planes!

Transformation Matrices: Rotations

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

$$\mathbf{R_y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{R_z} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- CCW +ve; orthonormal; length preserving; rows give direction vectors that rotate onto each axis; columns ....

Translation, Scaling, Composite

$$\mathbf{T}(a,b,c) = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{S}(a,b,c) = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- A sequence of transforms can be represented using a composite matrix: $\mathbf{M} = \mathbf{R_x T R_y S T} \cdots$
- Operations are not commutative, but are associative.
- $\mathbf{RT}$ and $\mathbf{TR}$??

Rotation and Translation

- $\mathbf{T}_{4\times4} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$ and $\mathbf{R}_{4\times4} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$
- $\mathbf{T}\,\mathbf{R} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} = ?$
- $\mathbf{R}\,\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = ?$

Rotation and Translation

- $\mathbf{T}_{4\times 4} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$ and $\mathbf{R}_{4\times 4} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$
- $\mathbf{T}\,\mathbf{R} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$
- $\mathbf{R}\,\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{R}\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$
- $\mathbf{T}\mathbf{R} = \mathbf{R}\,\mathbf{T}$ if: (a) $\mathbf{R} = \mathbf{I}$ or (b) $\mathbf{T} = \mathbf{I}(\mathbf{t} = \mathbf{0})$

  or (c) $\mathbf{R}\mathbf{t} = \; ?$

Rotation and Translation

- $T_{4\times4} = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$ and $R_{4\times4} = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}$
- $T\,R = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}$
- $R\,T = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{Rt} \\ \mathbf{0} & 1 \end{bmatrix}$
- $TR = R\,T$ if: (a) $\mathbf{R} = \mathbf{I}$ or (b) $\mathbf{t} = \mathbf{0}$ or (c) $\mathbf{Rt} = \mathbf{t}$
- When is $\mathbf{Rt} = \mathbf{t}$? $\mathbf{t}$ is an eigenvector of $\mathbf{R}$
- **Question:** Are transformations *commutative*?

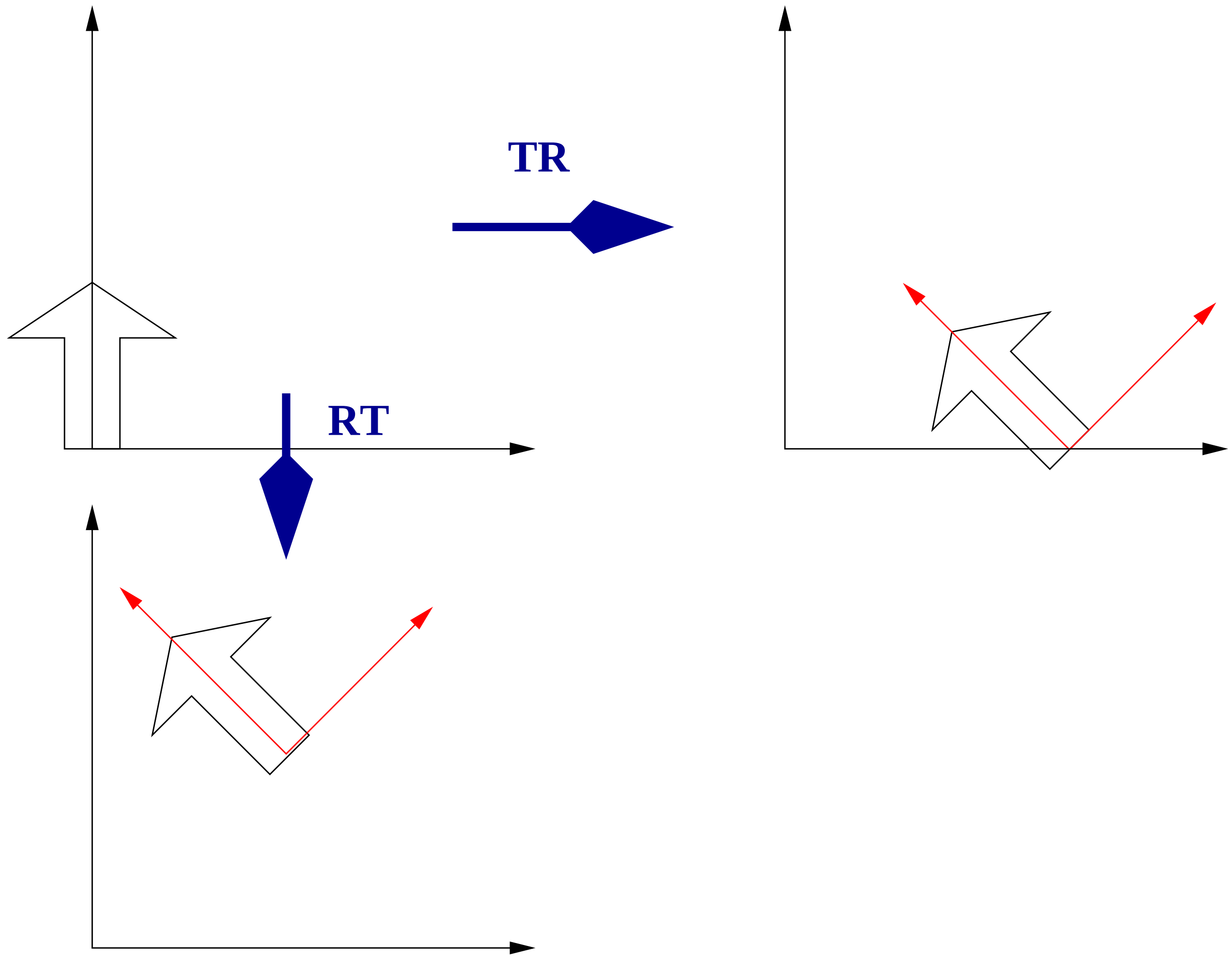
Commutativity

- Translations are commutative: $\mathbf{T_1 T_2 = T_2 T_1}$
- Scaling is commutative: $\mathbf{S_1 S_2 = S_2 S_1}$
- Are rotations commutative? $\mathbf{R_1 R_2 \stackrel{?}{=} R_2 R_1}$
- Rotation and Scaling commute? $\mathbf{SR \stackrel{?}{=} RS}$
- What would be an example?
  Consider the effect on Z-axis of:

Commutativity

- Translations are commutative: $\mathbf{T_1 T_2 = T_2 T_1}$
- Scaling is commutative: $\mathbf{S_1 S_2 = S_2 S_1}$
- Are rotations commutative? $\mathbf{R_1 R_2 \neq R_2 R_1}$
- Rotation and Scaling commute. $\mathbf{SR = RS}$
- Consider the effect on Z-axis of
  $\mathbf{R_x(90) R_y(90)}$ and $\mathbf{R_y(90) R_x(90)}$
- $\mathbf{RT \neq TR}$. (If translation is not parallel to rotation axis)
- Consider: $\mathbf{R}(\pi/4)$ and $T(5, 0)$.
  Where does the origin $(0, 0)$ go in $\mathbf{TR}$ and $\mathbf{RT}$?

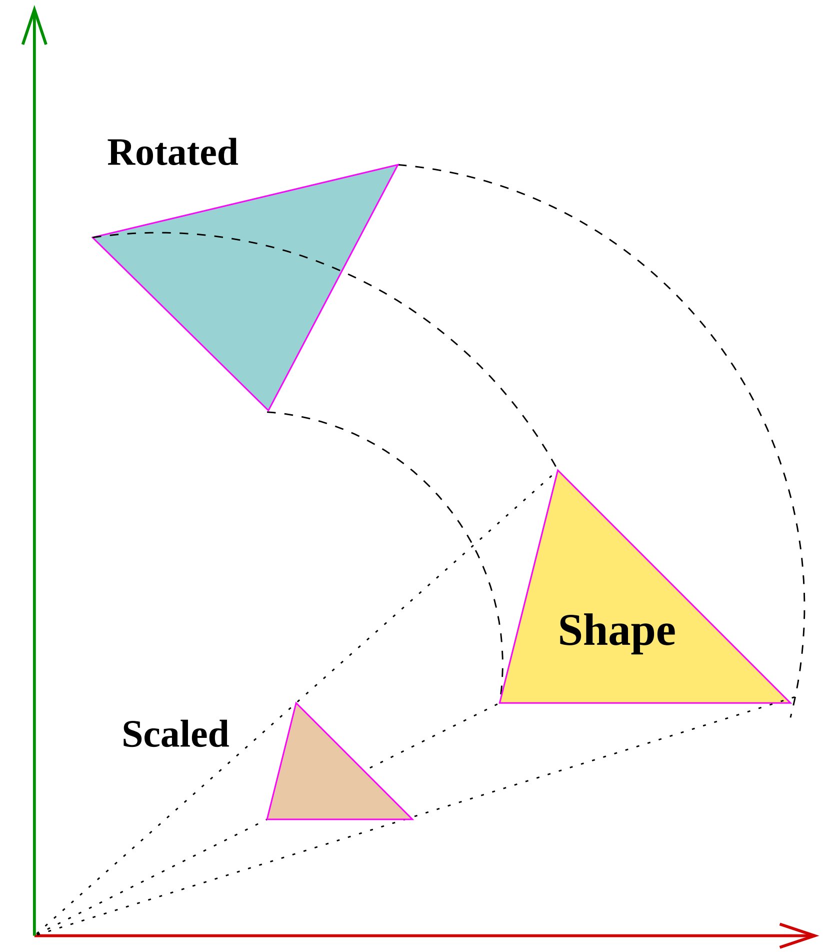
TR and RT



$\mathbf{TR}$ keeps it on X axis to $(5, 0)$. $\mathbf{TR}$ takes it to $(\frac{5}{\sqrt{2}}, \frac{5}{\sqrt{2}})$.

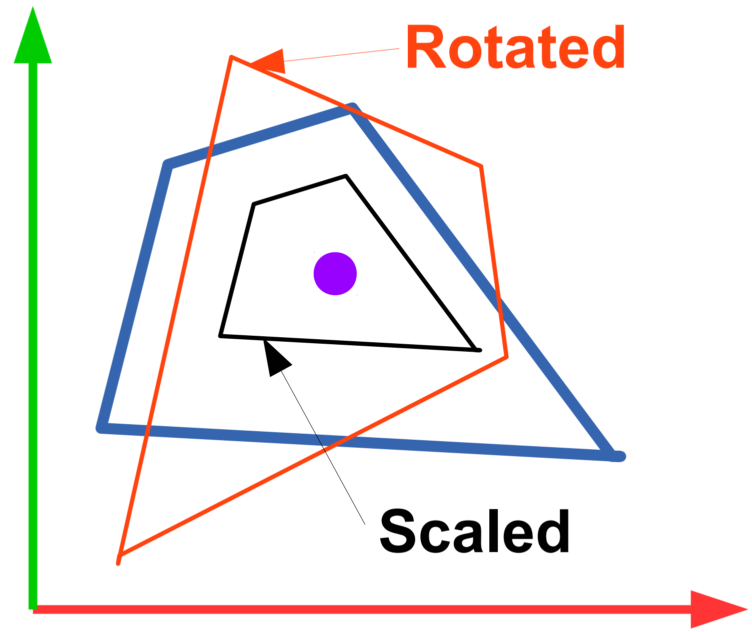
Objects Away from Origin

- Object “translates” when rotated or scaled!!
- Default: Perform these **about the origin**
- How do we transform points “in place”?
- Rotate or scale about the centroid of the object. Or about an arbitrary point
- How?



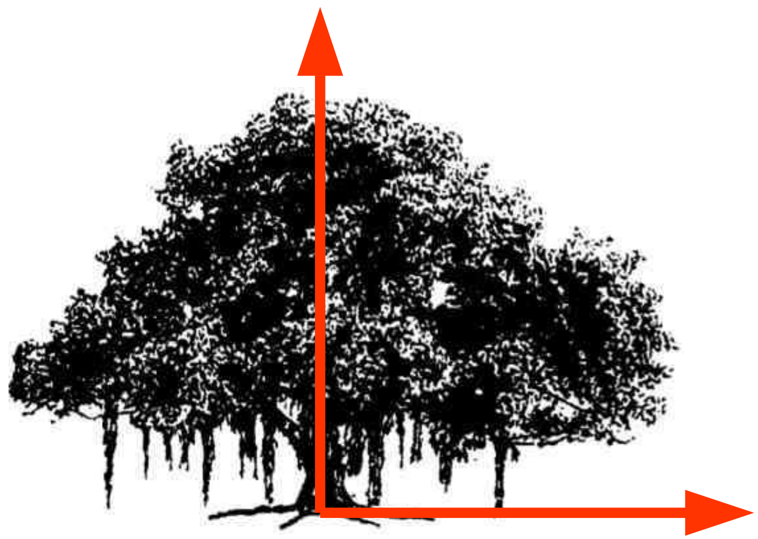
Transformations About A Point

- Rotating about point P
  - Align P with origin
  - Rotate/scale about origin
  - Translate back
- Rotation:
  $\mathbf{R_C}(\theta) = \mathbf{T}(\mathbf{C})\ \mathbf{R}\ \mathbf{T}(-\mathbf{C})$
- Scaling:
  $\mathbf{S_C}() = \mathbf{T}(\mathbf{C})\ \mathbf{S}()\ \mathbf{T}(-\mathbf{C})$
- A transformation $\mathbf{M}$:
  $\mathbf{M_C} = \mathbf{T}(\mathbf{C})\ \mathbf{M}\ \mathbf{T}(-\mathbf{C})$

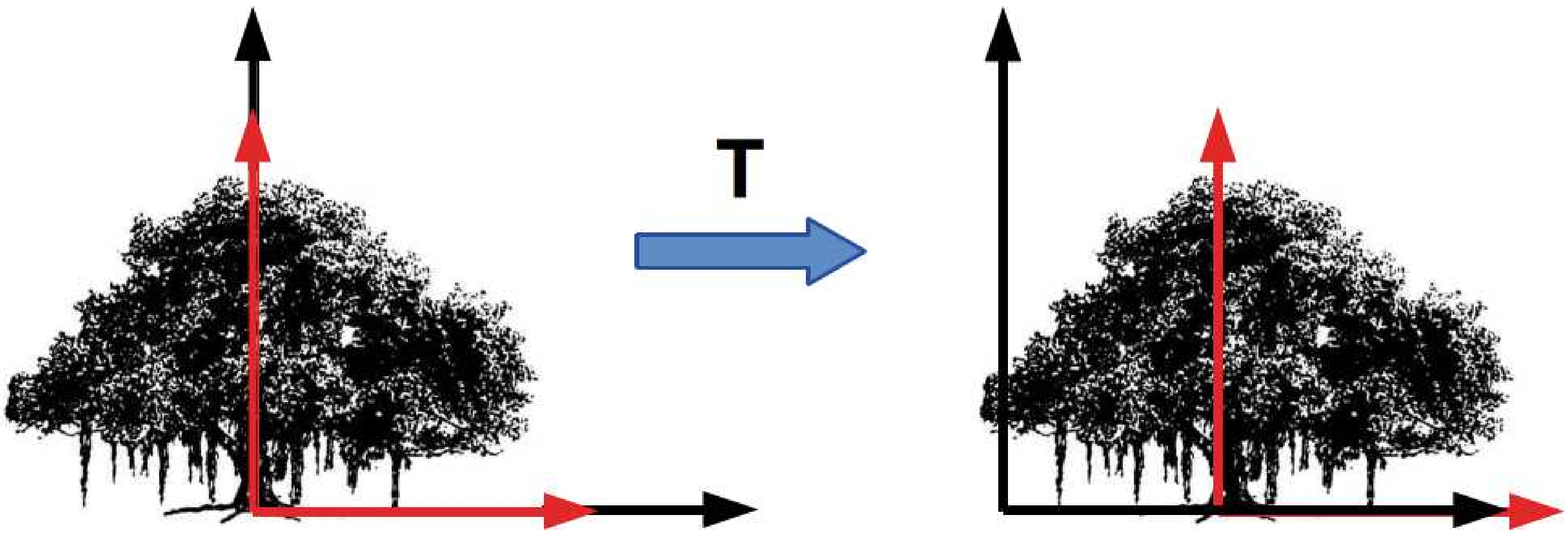


Object

- Object has a coordinate frame of its own.



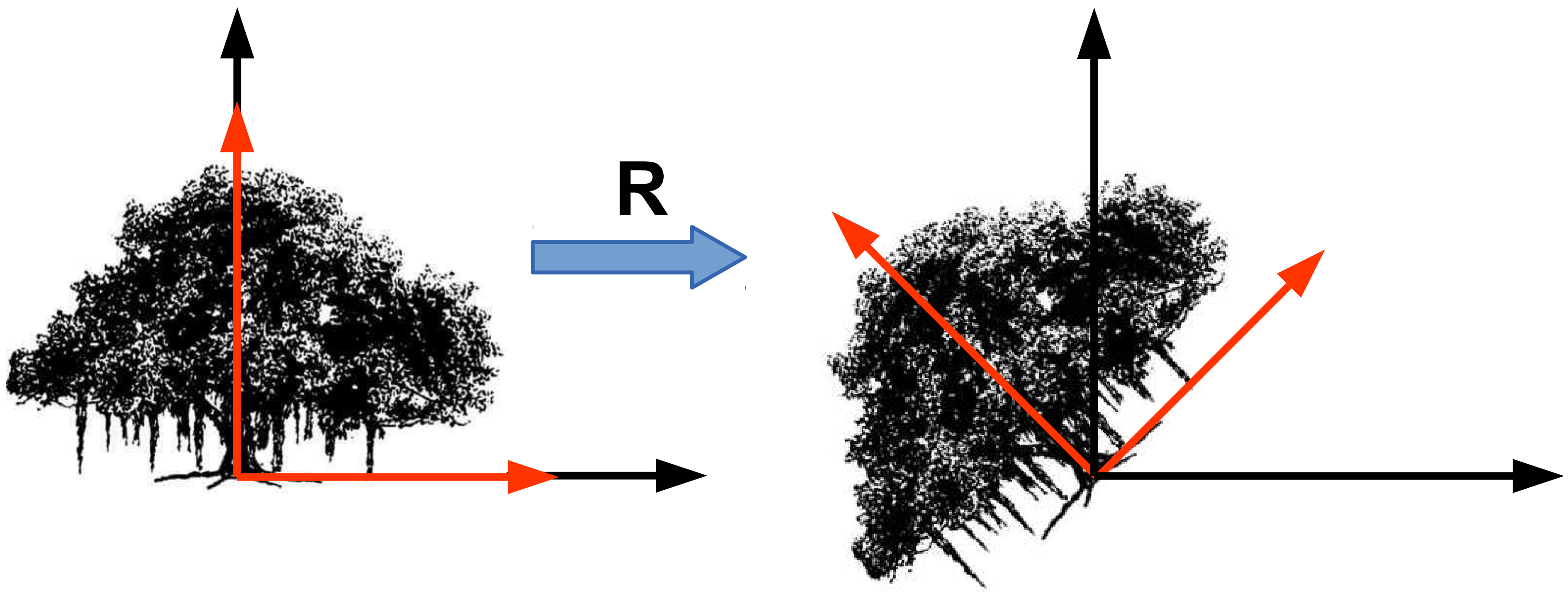
Object and Translation



$$\mathbf{P}' = \mathbf{P}$$

$$\mathbf{P}' = \mathbf{T}\ \mathbf{P}$$

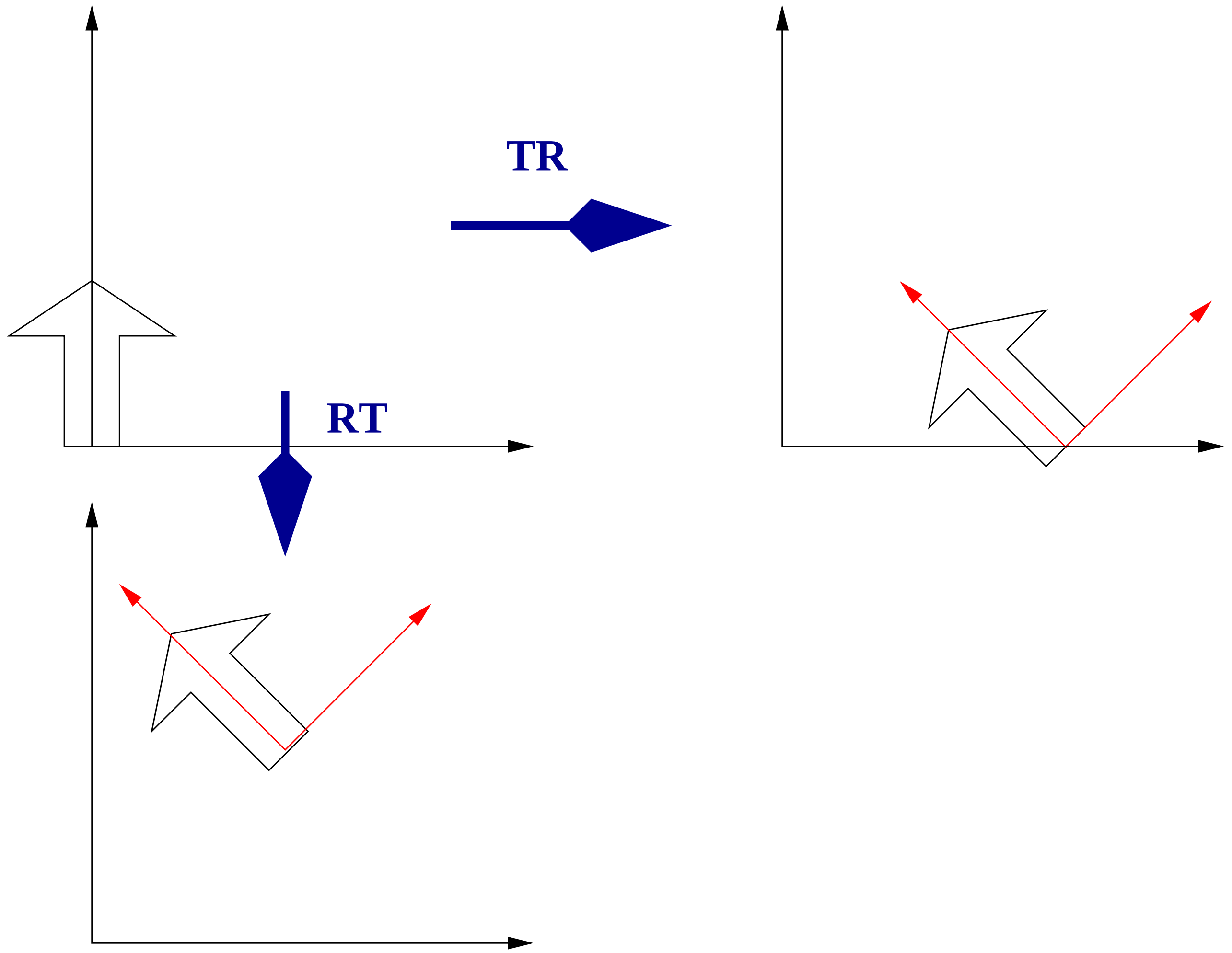
Object, Translation, Rotation



$$\mathbf{P}' = \mathbf{P}$$

$$\mathbf{P}' = \mathbf{R}\ \mathbf{P}$$

Understanding Transformations

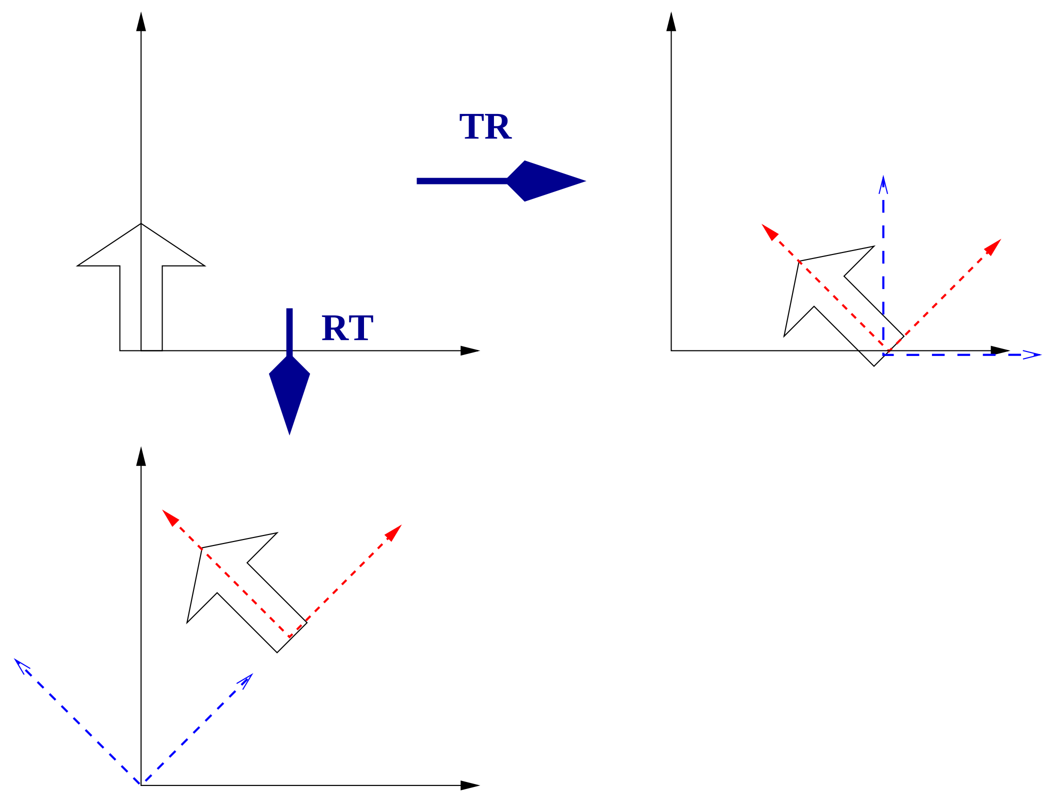


R, T Operations on Points

- **T(5,0) R($\pi/4$)**: Impact on a point:
  - R($\pi/4$): **(Point stays at $(\mathbf{0}, \mathbf{0})$)**
  - T$(5, 0)$: **(Point goes to $(\mathbf{5}, \mathbf{0})$)**
- **R($\pi/4$) T(5,0)**: Impact on the point:
  - T$(5, 0)$: **(Point moves to $(\mathbf{5}, \mathbf{0})$)**
  - R($\pi/4$). **(Point rotates about origin)**
- All points on the shape undergo the same motions and get new coordinates

Relating Coordinate Frames

- $T(5,0)$ and $R(\pi/4)$
- Start: Black axes
  Next: Blue axes
  Last: Red axes



- $\mathbf{P}' = \overset{\textbf{Black}}{|}\ \mathbf{T}\ \overset{\textbf{Blue}}{|}\ \mathbf{R}\ \mathbf{P}$
- $\mathbf{P}' = \overset{\textbf{Black}}{|}\ \mathbf{R}\ \overset{\textbf{Blue}}{|}\ \mathbf{T}\ \mathbf{P}$

R, T Operaions on Frames

- **T(5,0) R($\pi/4$)**: Impact on coordinate frame:
  - T$(5, 0)$: **(Origin shifted to $(\mathbf{5}, \mathbf{0})$)**
  - R($\pi/4$). **(Axes rotated at new origin)**
- **R($\pi/4$) T(5,0)**: Impact on coordinate frame:
  - R($\pi/4$): **(Axes rotate by 45 degrees))**
  - T$(5, 0)$. **(Point moves to $(\mathbf{5}, \mathbf{0})$ in new axes)**
- Frames move around, giving new coordinates to points on objects!!