CS3500
Computer Graphics
Module: History, 2D Graphics

P. J. Narayanan

Spring 2009

Course Content

- 2D Graphics: Concepts, Mathematics, Algorithms. Practice in OpenGL.
- 3D Graphics: Concepts, Mathematics, Hierarchical Modelling. Practice in OpenGL.
- Representation: Lines & Curves, Surfaces, Solids. Direct, implicit, parametric.
- Visible Surface Determination.
- Lighting and Shading.
- Ray Tracing.

Background Required

- Good programming skills in C or C++/
- Knowledge of geometry: Points, vectors, transformations, etc.
- Data structures.
- Good imagination and visualization.

Text Books and Reference

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

Course Management

- Homework assignments, Programming assignments, lab test, mid-term tests, final exam
- Weightages of different components:
  25-35% for the two tests.
  25-35% for the final exam.
  20-30% for programming assignments, lab test
  10% for the rest (Written assignments, etc.)

What is Computer Graphics?

- Create drawings, pictures, and images by the computer.
- Generate and manipulate representations and “models” required for this.
- Create algorithms to produce ultra-realistic images.
- Do all these very fast.

Application Areas

- User interfaces
- Computer aided design (Civil/Mech/VLSI)
- Visualization of scientific & engineering data
- Art
- Virtual Reality
- Entertainment: Great computer games!

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.
- Graphics workstations by Silicon Graphics came out in early eighties.

Popular Graphics

- Graphics became "popular" only after mass-produced personal computers became a reality in mid to late eighties.
- Graphics Accelerators: on board hardware to speed up graphics computations.
- Accelerators were expensive until mid nineties!
- Very high end performance is available economically today.

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)

- Open GL: 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!
- Java3D: Java-based toolkit.
- Desirable: High level toolkits.

Graphics Output Devices

- Printers: character, line, page, dot-matrix, ink-jet, laser. Monochrome or Colour.
- Plotters: Single pen, multipen.
- CRT (Cathode Ray Tube): Colour/mono.
- LCD displays: Colour/mono.
- Projectors: CRT/LCD etc. Colour/mono.
- Film: Photographic film recorders.

Graphics Input Devices

- Keyboard, mouse: To control display.
- Joystick: Position control.
- Pad: Line/curve/position input.
- Scanner: Pictures to digital images.
- Camera: World to digital images.

How graphics?

- How can a computer "draw" pictures?
- How do **we** draw pictures? Using a number of strokes of a brush/pencil.
- This was the early model for graphics: called **Vector Graphics**.
- Intuitive. Maps directly to plotters, CRT.

Raster Graphics

- CRT became the most popular graphics device.
- CRTs are best used in “raster scanning” mode.
- A 2D image can be displayed directly.

Vector and Raster Graphics

- Vector graphics: Like how we draw. Think in terms of strokes.
- Raster graphics: Discretize the image to a grid and fine which pixels are to be coloured with what.

Comparison: Vector Graphics

- Intuitive as we think in terms of strokes.
- Accurate: No approximations involved.
- Refresh Speed: Depends on scene complexity
- Shading is difficult to achieve
- Expensive.

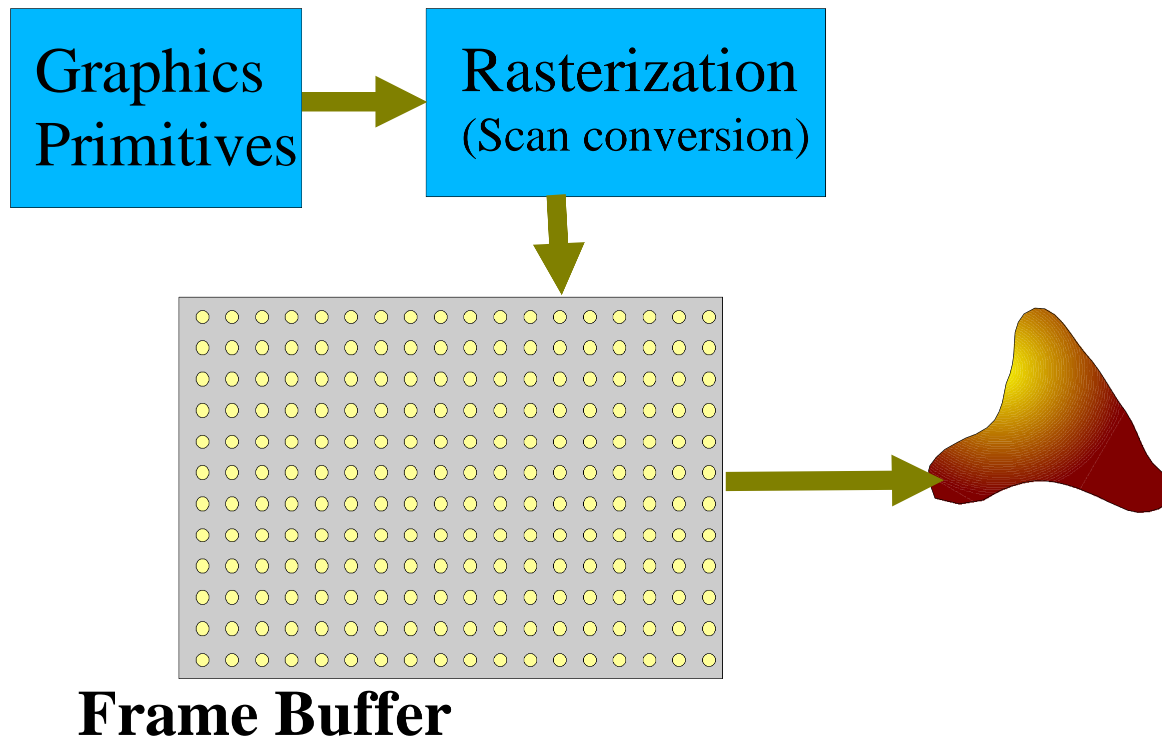
Comparison: Raster Graphics

- Refresh Speed: Independent of scene complexity
- Shading is not difficult
- Inexpensive
- Additional step of "**rasterization**" to convert from strokes to pixels.
- Aliasing effects: Only a discrete approximation

Raster Graphics Process

- Primitives are rasterized to the framebuffer.
- Framebuffer is raster scanned continuously and automatically to the output device.

System Architecture



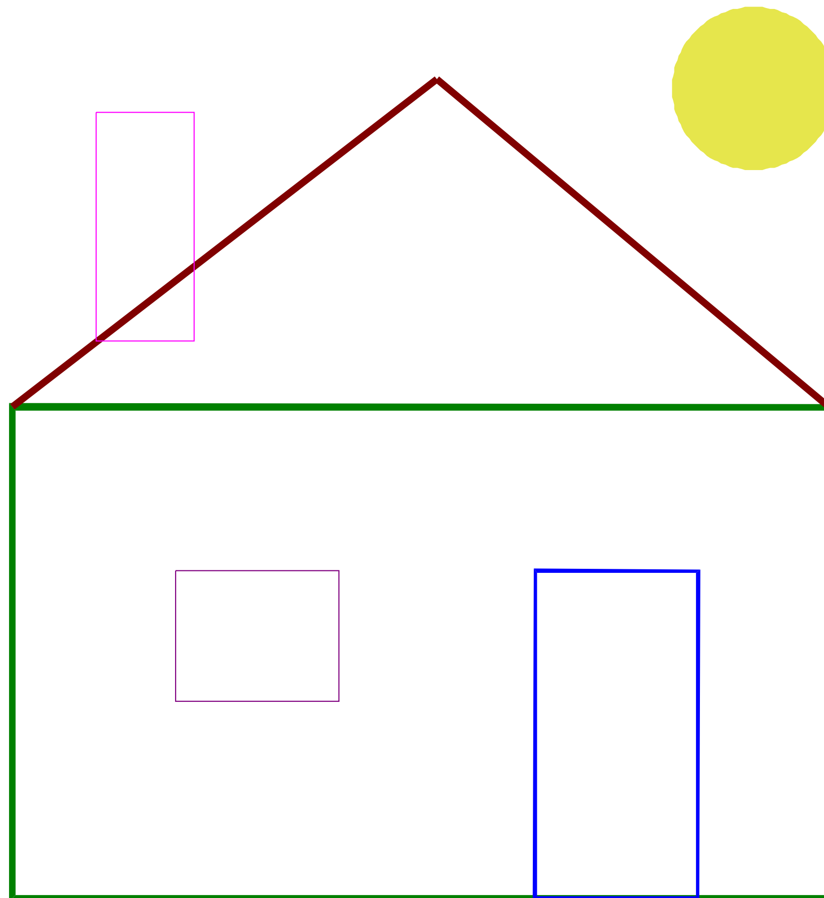
How to Draw A House?

- Compose out of basic shapes

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

- As simple as that! Now you know everything!

Resulting House



2D Graphics Primitives

- Point.
- Line, Line strip, Line loop.
- Triangle, Triangle strip, Triangle Fan, Quad, Quad strip, Polygon.
- Why not circles, ellipses, hyperbolas?

Graphics Attributes

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

Points

- Specified by coordinates (x, y).
- Respectively the projections onto the X and Y axes.
- Axes need not be orthogonal, though that is more common.

Points in OpenGL

```
glBegin(GL_POINTS);
glVertex2i(10, 20);
glVertex2f(30.0, 40.0);
glVertex2u(5, 7);
glVertex2d(-15.0, 17.0);
glEnd();
```

Lines

- Specified by the end points.

```
glBegin(GL_LINES);
glVertex2i(10, 20);
glVertex2f(30.0, 40.0);
glVertex2u(5, 7);
glVertex2d(-15.0, 17.0);
glEnd();
```

- GL_LINE_STRIP and GL_LINE_LOOP.

Triangles

- Three vertices make a triangle

```
glBegin(GL_TRIANGLES);
glVertex2i(10, 20);
glVertex2f(30.0, 40.0);
glVertex2u(5, 7);
glEnd();
```

Triangle Strips

- Three vertices make a triangle

```
glBegin(GL_TRIANGLE_STRIP);
glVertex2i(10, 20);
glVertex2f(30.0, 40.0);
glVertex2u(5, 7);
glVertex2d(-15.0, 17.0);
glEnd();
```

- $\mathbf{N-2}$ polygons from $\mathbf{N}$ vertices.

Polygons

- A polygon is made up of a number of vertices!

```
glBegin(GL_POLYGON);
glVertex2i(10, 20);
glVertex2f(30.0, 40.0);
glVertex2u(5, 7);
glVertex2d(-15.0, 17.0);
glEnd();
```

Specifying Colours

- Done using **Red**, **Green**, and **Blue** values.
- Each from 0.0 (none) to 1.0 (full).
- **R-G-B** form the **additive primary colours**. Mix them in different proportions to get other colours.
- OpenGL remembers a current colour with which everything is painted.

A Yellow Polygon

- In OpenGL, glColor3f(r, g, b);

```
glColor3f(1.0, 1.0, 0.0);
glBegin(GL_POLYGON);
glVertex2i(10, 20);
glVertex2f(30.0, 40.0);
glVertex2u(5, 7);
glVertex2d(-15.0, 17.0);
glEnd();
```

A Multicolour Line

```
glBegin(GL_LINES);
glColor3f(1.0, 0.0, 0.0);
glVertex2d(13.0, 15.0);
glColor3f(0.0, 0.0, 1.0);
glVertex2f(30.0, 40.0);
glEnd();
```

- Will draw a line that smoothly changes from red to blue.
- Similarly for polygons.

Filled and Unfilled Polygon

- Polygon consists of the edges only or also the interior?
- Polygon has 2 faces: front and back. (More about it later)
- Each can be one of: **GL_POINT, GL_LINE, GL_FILL**.
- glPolygonMode(GL_FRONT, GL_FILL);
- Colour interpolation for the whole triangle.

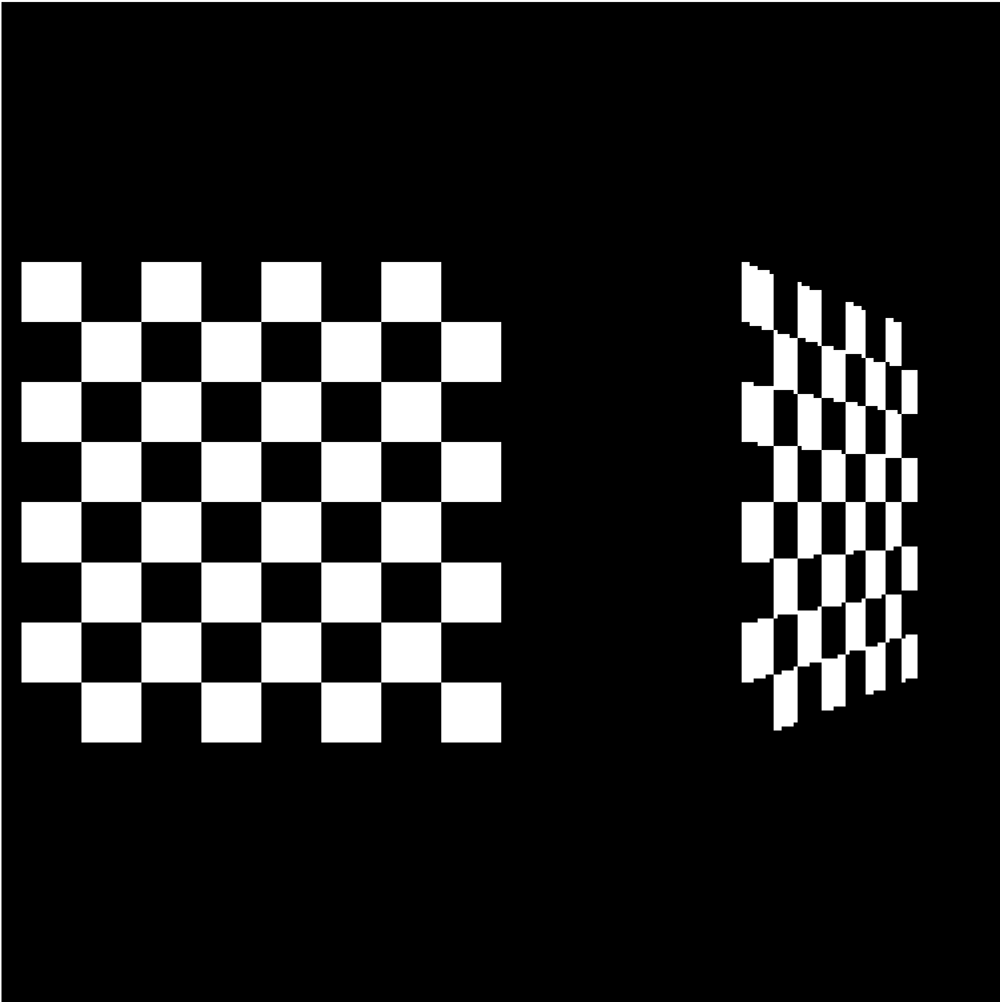
Textures

- Graphics systems allow one to *paste* a picture (called a **texture**) on a polygon.
- Define an image as the current texture. (gif, jpg, png, etc.)
- Texture coordinates *(s, t)* from 0.0 to 1.0 for the image.
- Specify texture coordinates for each polygon vertex.
- The picture gets **“pinned”** to the polygon verices, stretching or compressing as necessary.

A Textured Triangle

```
// Define the texture using an image
glBegin(GL_POLYGON);
glVertex2i(10, 10); glTexCoord2f(0.0, 0.0);
glVertex2f(20.0, 10.0); glTexCoord2f(0.0, 1.0);
glVertex2u(20, 20); glTexCoord2f(1.0, 1.0);
glEnd();
```

- Colour given by the texture and colour of the polygon can be combined in different ways.



Summary

- Graphics toolkit fills the **Frame Buffer** with a picture. Raster scan hardware displays it automatically.
- Frame Buffer is a 2D array that can be filled by a program.
- Pictures are drawn using basic primitives.
- Basic primitives: **point, line, polygon**.
- Attributes: point/line width, colour.
- Texturing helps bring realism quickly.

2D Transformations

- Translations, Rotations, Scaling, Shearing.
- View 1: a point undergoing these operations and ending up with different coordinates in the same coordinate system.
- View 2: The coordinate frame undergoing such chanages. Points end up with different coordinates.
- A point is represented using 2 numbers (x, y) that are the projections on to the respective coordinate axes.

Translation

- Translate a point by (a, b).
- Points coordinates become $(x + a, y + b)$.
- In vector form, $P' = P + T$.
- Or, coordinate frame translates by $-T$. All points get transformed.
- Distances, angles, parallelism are all maintained.

Rotation

- Rotate a point about origin CCW by angle θ.
- $x' = x\cos\theta - y\sin\theta, \quad y' = x\sin\theta + y\cos\theta$.
- We are more comfortable with $P' = R\,P$ or

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

- Coordinate frame rotating the other way?
- Invariants: distances, angles, parallelism.

Non-uniform Scaling

- Scale along X, Y directions by s and u.
- $x' = s\,x, \;\; y' = u\,y$.
- We are more comfortable with $P' = S\;P$ or

$$\begin{bmatrix} x \\ y \end{bmatrix}' = \begin{bmatrix} s & 0 \\ 0 & u \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Coordinate system undergoing scaling?
- Invariants: parallelism. (Angles also for uniform scaling.)

Shearing

- Add a little bit of x to y (or vice versa).
- $x' = x + t\,y, \quad y' = y$, alternately expressed as

$$\begin{bmatrix} x \\ y \end{bmatrix}' = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Rectangles can become parallelograms.
- Invariants: parallelism.

General Transformation

- Negative entries in a matrix indicate reflection.

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

- A general matrix combines rotation, scaling, shearing, and reflection.
- Translation alone stands out as a vector addition, instead of matrix-vector product.
- Can that be brought on board also?

Uniform Way to Handle All

- **Homogeneous Coordinates**: Add a **scale factor** w to each coordinate. A point becomes $[x, y, w]^{\mathsf{T}}$
- Real coordinates are: $(x/w,\ y/w)$.
- Now, translation is also is: $P' = T\ P$
- For a point: Rotation followed by translation followed by scaling, followed by another rotation: $\mathbf{P}' = \mathbf{R_2STR_1\ P}$.
- All matrices are 3×3. Last row is $[0\ \ 0\ \ 1]$.

2D Transformations: Summary

- Rotation, Translation, Scaling, Shearing, Reflection are all matrix operations using Homogeneous Coordinates.
- Operations can be combined into a composite matrix. Each point can then undergoes one matrix-vector product.
- 2D coordinates have become 3-vectors and the matrices are 3×3, though of a special form.

Objects Away from Origin

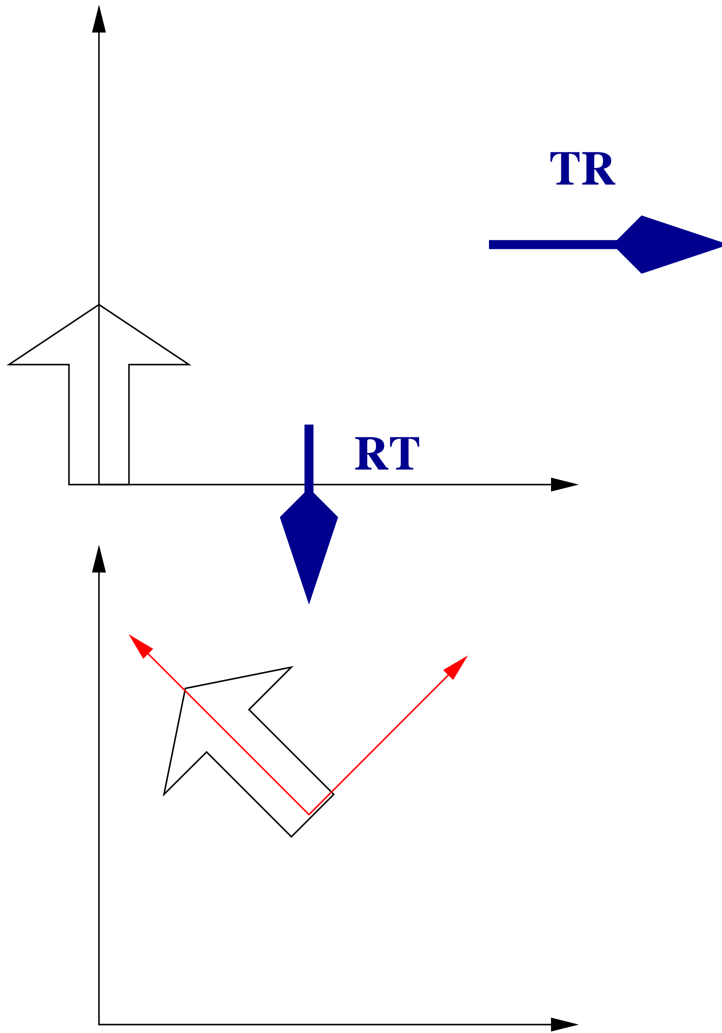
- Rotation: Object seems to “translate” unintentionally when rotated!
- Scaling: Object moves closer to/farther from the origin when rotated.
- These operations are performed **about** a point.
- The transformations were about the origin.

Rotations about Other Points

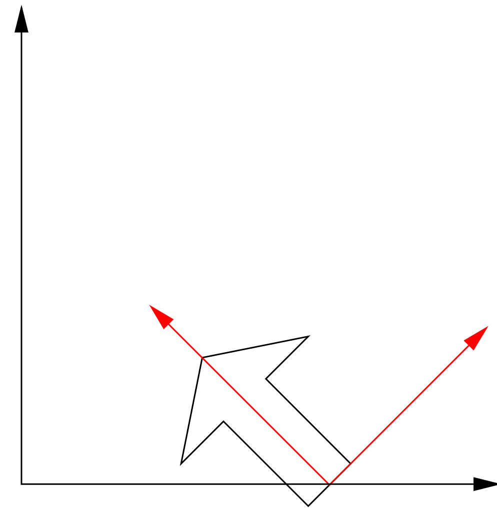
- How about "in place" rotation? That is, rotation about an arbitrary point?
- Idea: Translate to have that point at origin, rotate/scale about origin, translate back!
- $\mathbf{R_C}(\theta) = \mathbf{T}(\mathbf{C})\ \mathbf{R}(\theta)\ \mathbf{T}(-\mathbf{C})$
- Can do any numbers of this.

Composition of Matrices

- A sequence of operations can be reduced to a single, composite, matrix. Saves computations.
- The operations are not **commutative**.
  Example: $R(\pi/4)$ and $T(5,0)$.
- $\mathbf{TR}$ will keep point $(0,0)$ on X axis to $(5,0)$.
  $\mathbf{RT}$ will take it to $(5/\sqrt{2}, 5/\sqrt{2})$.
- Matrix multiplication also is not commutative in general.


RT

TR



Two Views of R and T

- **T R**: Rotate point by $\pi/4$. (stays at $(0, 0)$).
  Translate by $(5, 0)$. (Goes to $(5, 0)$).

- Suppose the coordinate frame first translates by $(5, 0)$ and then rotates by $\pi/4$, what are the new coordinates of a point that was at $(5, 0)$ earlier?

- Point going through changes in a coordinate frame when viewed from right to left.

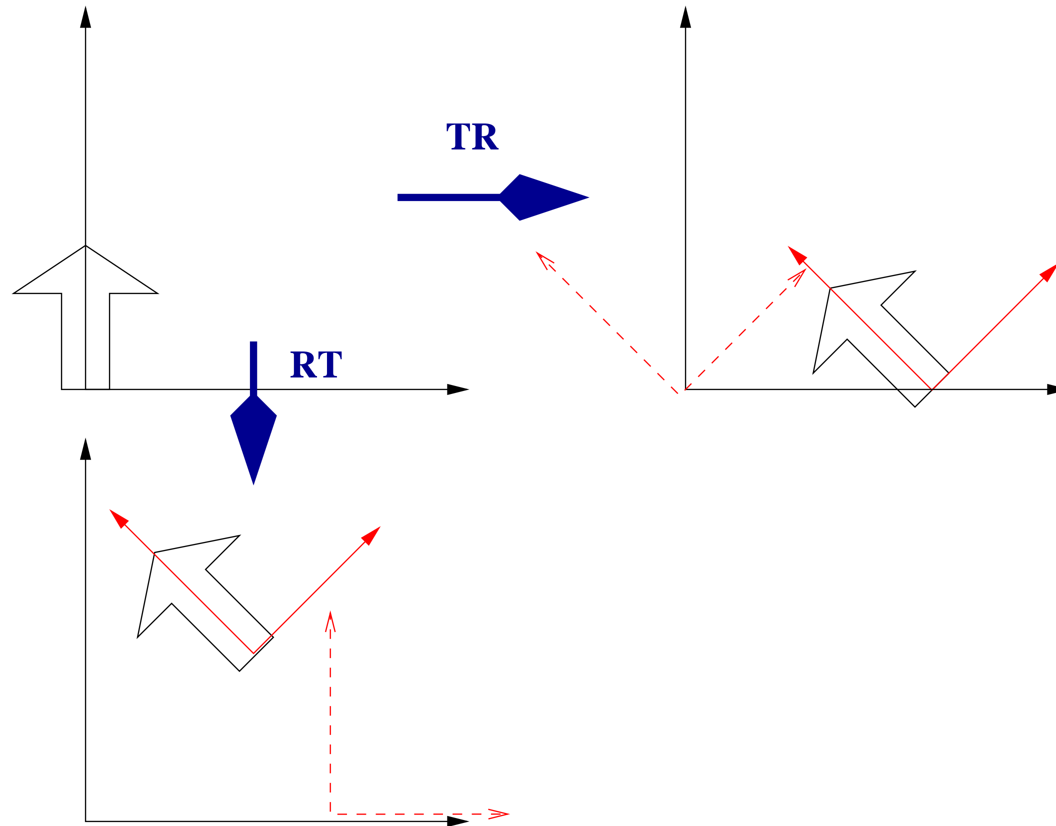
- Coordinate system going through the same transformations when viewed from left to right.

Frame Doing the Opposite?

- Point rotates by $\theta \leftrightarrow$ Coordinate frame rotates by $-\theta$.
- Point translates by $T \leftrightarrow$ Frame translates by $-T$.
- Point scaled by $s \leftrightarrow$ Frame expands by s
- Matrix equation $\mathbf{P}' = \mathbf{MP}$ relates the coordinates in the $\mathbf{XY}$ and the $\mathbf{X'Y'}$ coordinate frames.

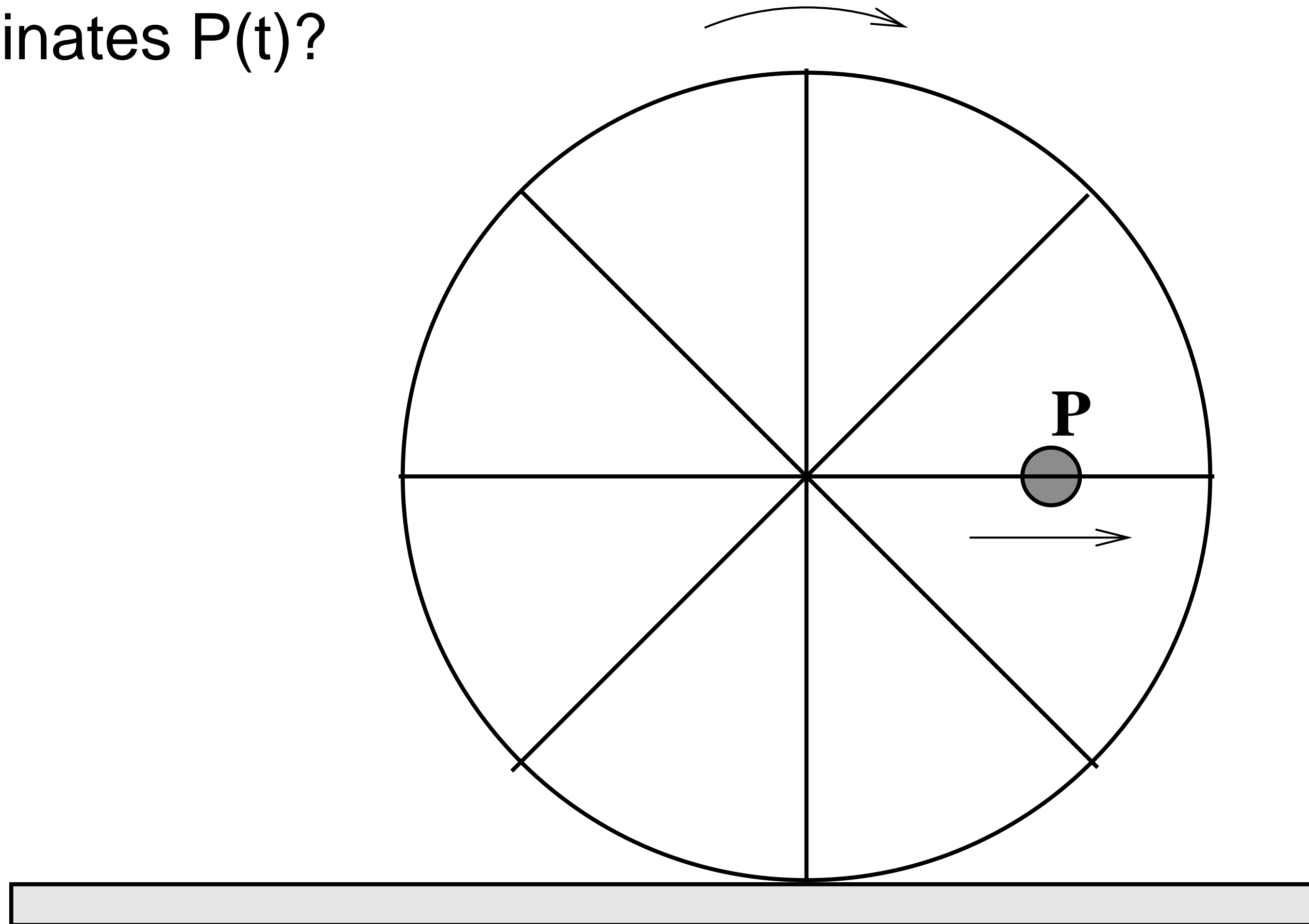
Relating Coordinates in Two Frames

- Combinations of $T(5,0)$ and $R(\pi/4)$

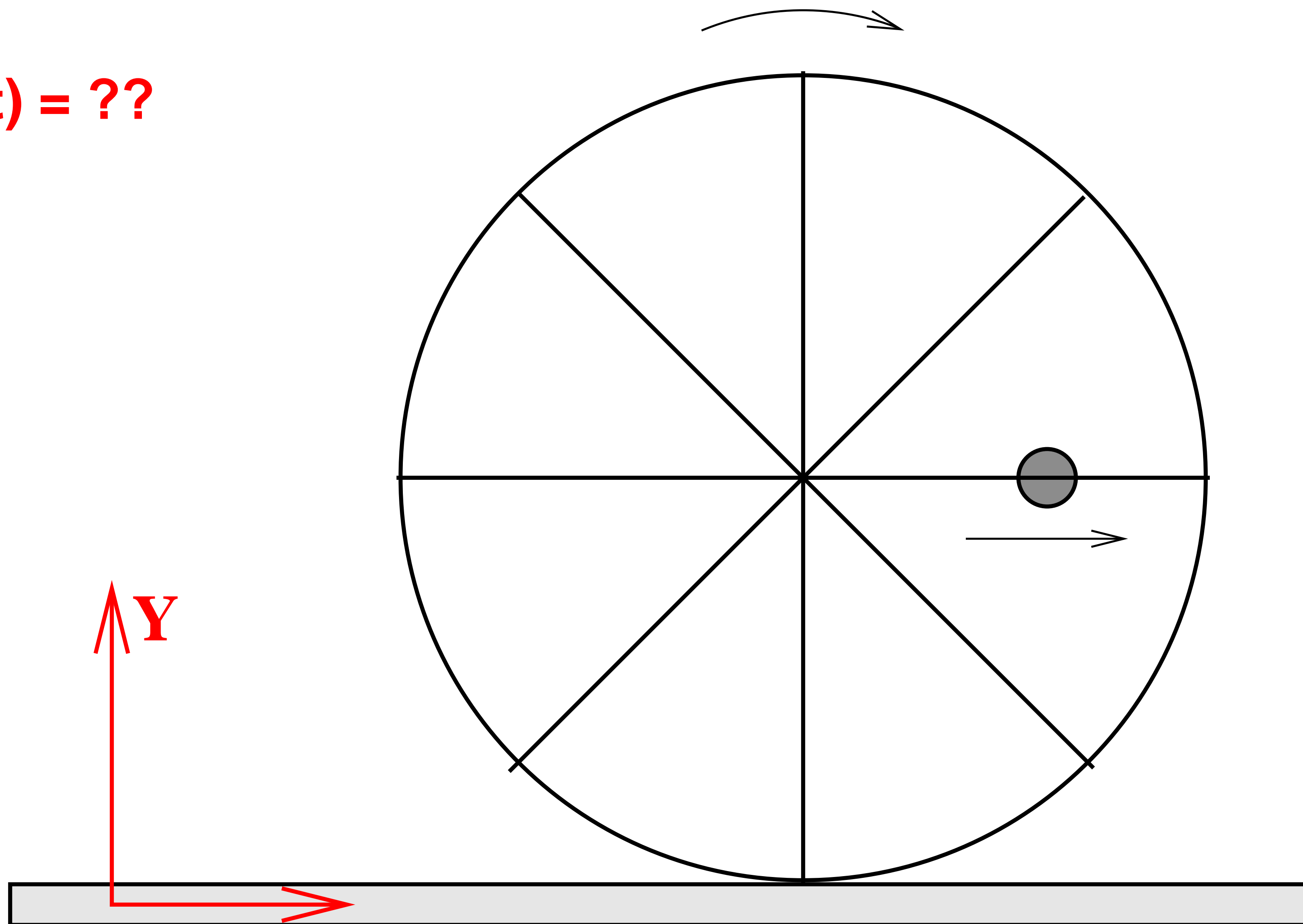


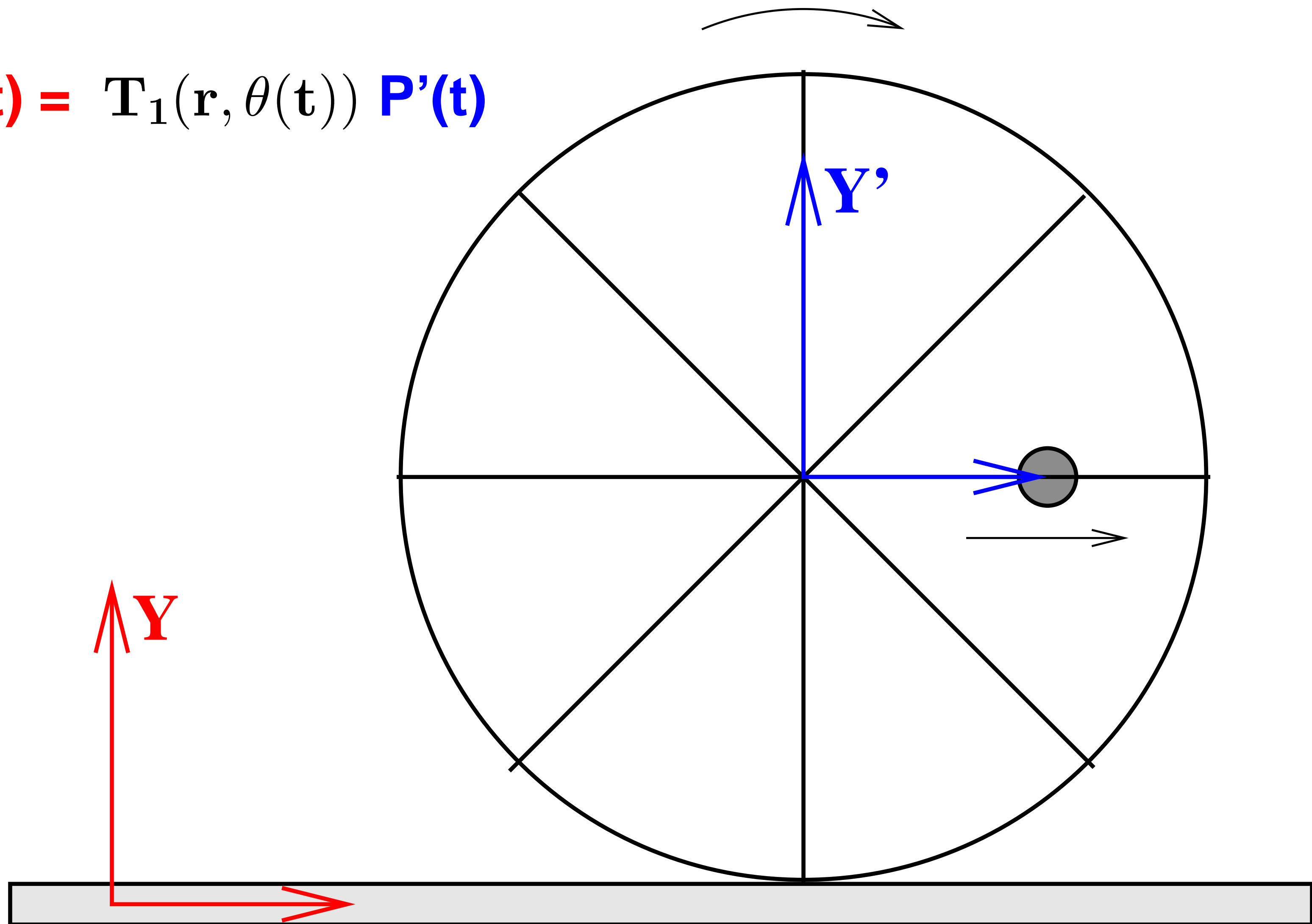
Rolling Wheel

Coordinates P(t)?

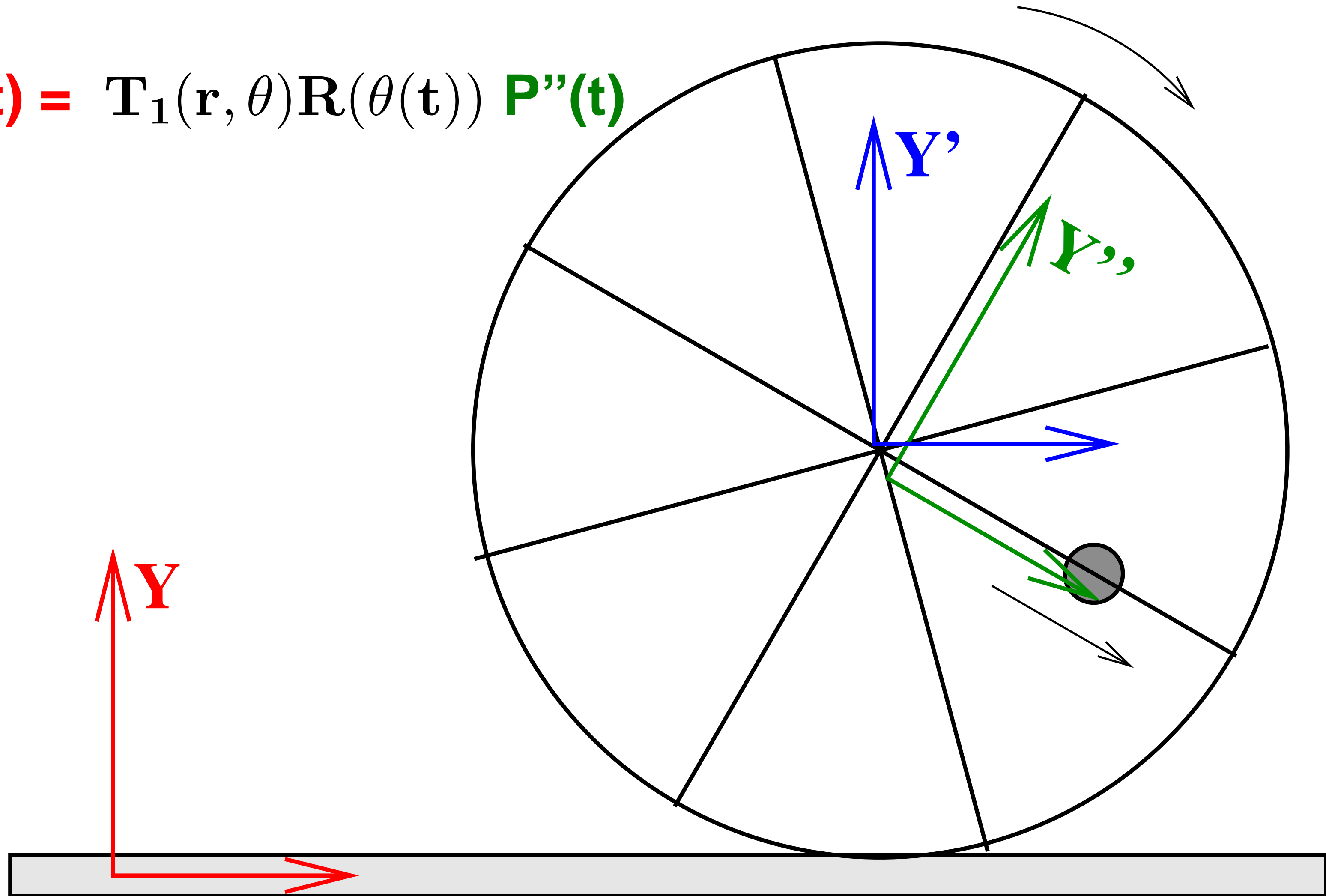


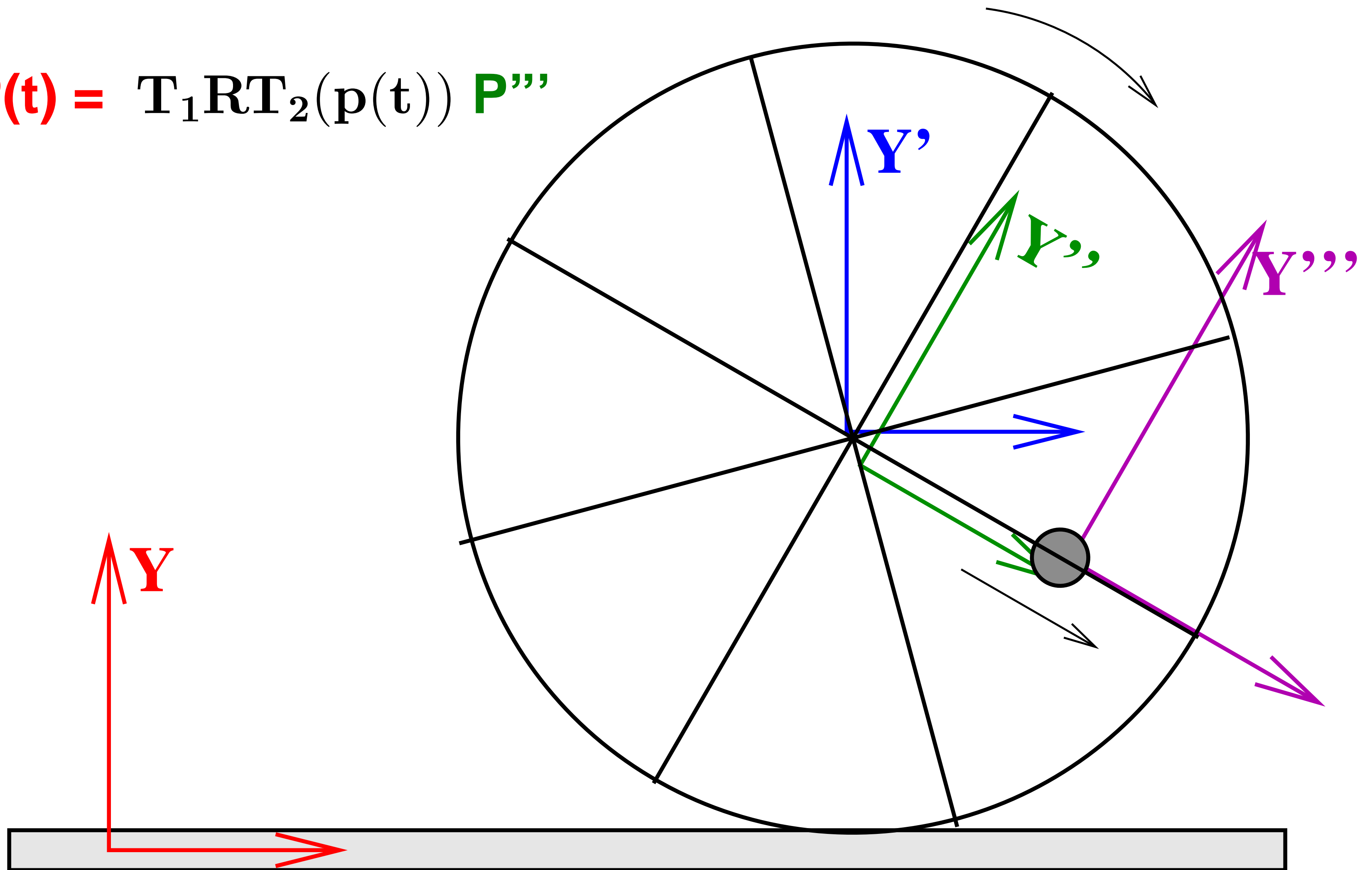
P(t) = ??




P(t) = $\mathbf{T_1}(\mathbf{r}, \theta(\mathbf{t}))$ P'(t)
Y'
Y

$$\mathbf{P(t)} = \mathbf{T_1}(\mathbf{r}, \theta)\mathbf{R}(\theta(\mathbf{t}))\ \mathbf{P''(t)}$$





P(t) = $\mathbf{T_1 R T_2(p(t))}$ P’’’
Y’
Y’’
Y’’’
Y

Final Transformation

- P(t) = $\mathbf{T_1}(\mathbf{r}, \theta(\mathbf{t}))\ \mathbf{R}(\theta(\mathbf{t}))\ \mathbf{T_2}(\mathbf{p}(\mathbf{t}))$ P’’’
- $\mathbf{T_1}(\mathbf{t}) = \mathbf{r}\ \theta(\mathbf{t}) = \mathbf{r}\ \omega\ \mathbf{t}$
- $\mathbf{R}(\theta) = \mathbf{R}(\theta)$, normal rotation matrix.
- $\mathbf{T_2}(\mathbf{t}) = \mathbf{T}(\mathbf{p}(\mathbf{t})) = \mathbf{T}(\mathbf{v}\ \mathbf{t})$
- **P’’’** = $[0, 0, 1]^T$
- Lot simpler than thinking about it all together.

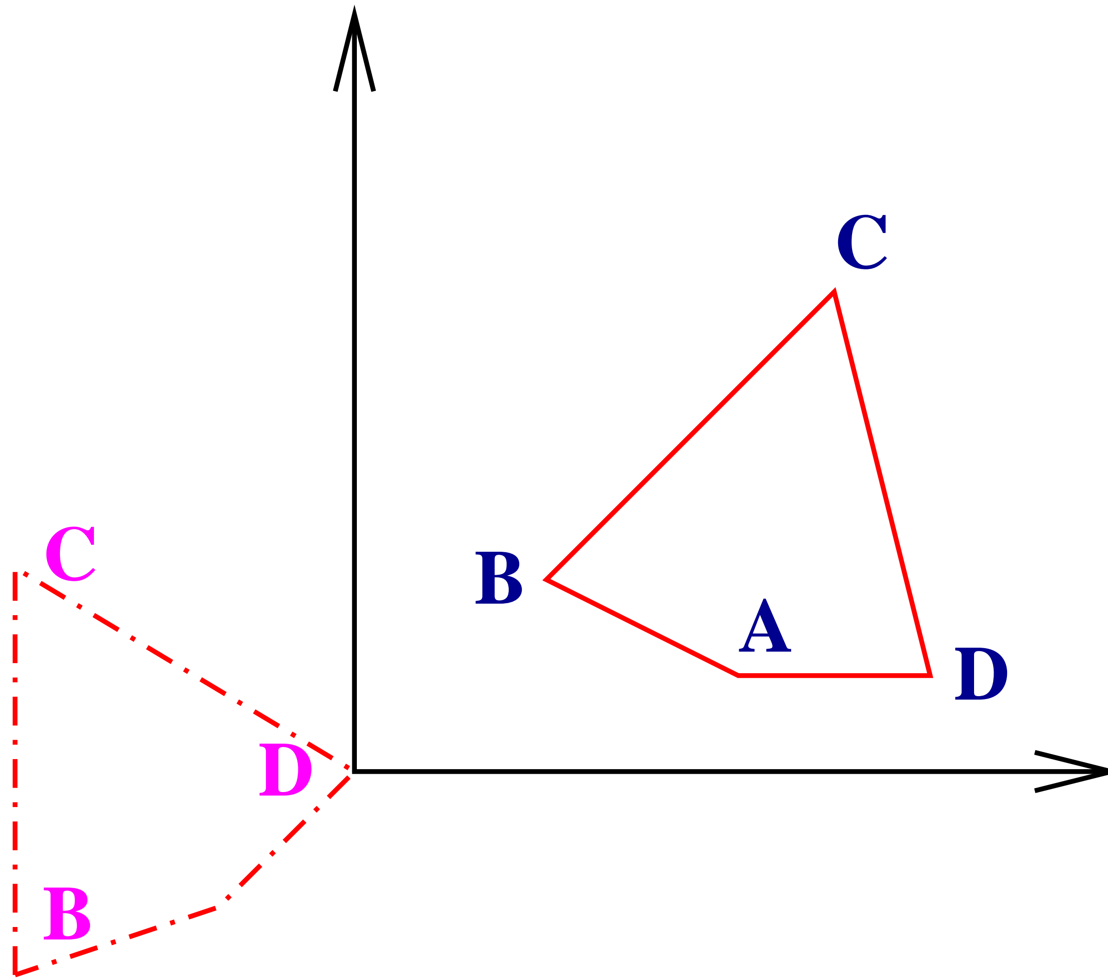
Rotation: Alternate View

- $[\cos\theta \quad -\sin\theta]^{\mathsf{T}}$ is the axis that **rotates to** the X-axis.
- Rows: Direction vectors that rotate and sit on the respective coordinate axes.
- Cols: Direction vectors to which the coordinate axes go after rotation.

Why Alternate Views?

- The combined effects are easy to figure out, given a series of operations.
- Reverse is called for often in Graphics.
- Given the model for a table, what are the transformations that would place it at the center of the table with 37 degrees to the northern wall?
- Such intuitions help in solving this problem.

A Problem



Transformation Computation

- How do we bring **D** to origin and **BC** parallel to the Y axis?
- First translate by $-\mathbf{D}$.
- Rotate such that the unit vector $\vec{\mathbf{u}}$ along **BC** sits on the Y axis. The other vector is orthogonal to it.
- Rotation matrix: $\begin{bmatrix} \vec{\mathbf{u}}_y & -\vec{\mathbf{u}}_x \\ \vec{\mathbf{u}}_x & \vec{\mathbf{u}}_y \end{bmatrix}$ or $\begin{bmatrix} -\vec{\mathbf{u}}_y & \vec{\mathbf{u}}_x \\ \vec{\mathbf{u}}_x & \vec{\mathbf{u}}_y \end{bmatrix}$.
- Difference? Which direction is aligned to the X-axis?

Another Problem

- A clock is hanging from a nail fixed to a flat plate. The plate is being translated with a velocity $\vec{\mathrm{v}}$ and acceleration $\vec{\mathrm{a}}$. The pendulum of the clock swings back and forth with a time period of 5 seconds and a max angle of $\pm\theta$. An ant travels from the bottom tip of the pendulum up to the centre.

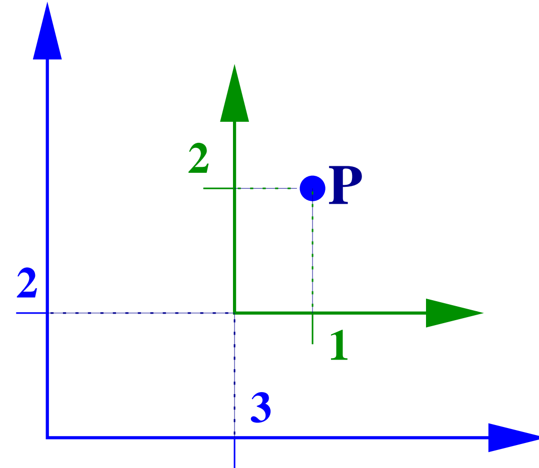
- How do we compute the ant's position with respect to a fixed coordinate system coplanar with the plate?

Composite Transformations

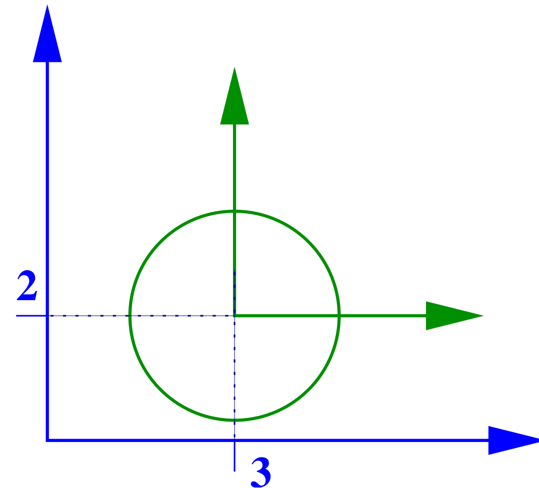
- $\mathbf{T} \cdot \mathbf{R}$: Translation vector appears as last column.
- Translation followed by rotation: more complex.
- $\mathbf{M} = \mathbf{M_1}\ \mathbf{M_2}\ \cdots\ \mathbf{M_k}\ \ \mathbf{P}$.
- Points get transformed and stay in the same coordinate frame when viewed from right to left.
- Coordinate frames get transformed to new ones with subsequent operations expressed in the new frame when viewed from left to right.

Change of Coordinate Frame

- $\mathbf{P_G = (1, 2)}$, $\mathbf{P_B = (4, 4)}$.
- $\mathbf{T(3, 2)}$ aligns Blue to Green.
- $\mathbf{P_B = T(3, 2)\ P_G}$.



- $x^2 + y^2 = r^2$ and $(x - 3)^2 + (y - 2)^2 = r^2$
- $\mathbf{P_B = T(3, 2)\ P_G}$.



Viewport Transformation

- Graphics toolkit like OpenGL computes a standard square image of size -1 to +1 in X and Y.
- The **viewport** transformation maps it to the actual window on screen.
- Map x range from $-1 \cdots 1$ to $0 \cdots W$ and y range from $-1 \cdots 1$ to $0 \cdots H$.
- How do we decompose this into steps?

Viewport Txform: on Points

- Let us look at the $(1,1)$ corner.
- First set the sizes right: $\mathbf{S}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})$
- Translate to have origin at SouthWest corner: $\mathbf{T}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})$
- Overall transformation: $\mathbf{M} = \mathbf{T}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})\ \mathbf{S}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})$

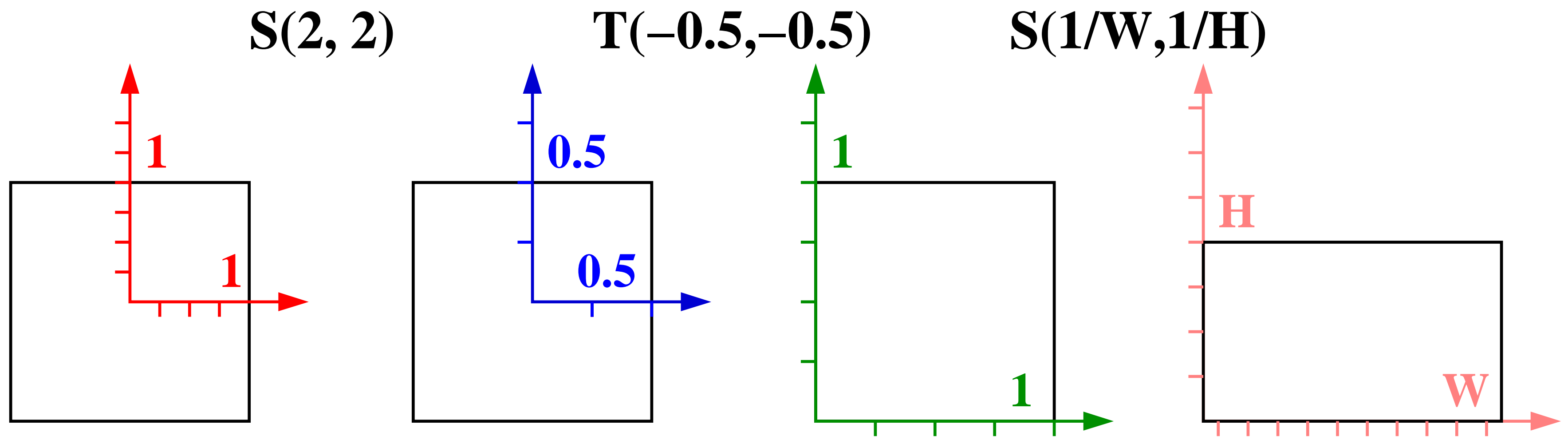
$$\mathbf{M} = \begin{bmatrix} \frac{W}{2} & 0 & \frac{W}{2} \\ 0 & \frac{H}{2} & \frac{H}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Viewport Tx: On Coord Frames-1

- Start with a $W \times H$ window with origint at SW corner
- Scale so that the size is 2×2: $\mathbf{S}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})$
- The centre of the rectange is now at $(1, 1)$.
- Translate the origin to the centre: $T(1, 1)$.
- Overall transformation: $\mathbf{S}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})\ \mathbf{T}(\mathbf{1}, \mathbf{1}) = \mathbf{M}$.
- Can fix centre first, then size: $\mathbf{T}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}})\ \mathbf{S}(\frac{\mathbf{W}}{\mathbf{2}}, \frac{\mathbf{H}}{\mathbf{2}}) = \mathbf{M}$

Viewport Tx: On Coord Frames-2

- `glViewport(0, 0, W, H)`



- $\mathbf{P_C} = \mathbf{P_r}$ and $\mathbf{P_o} = \mathbf{P_W}$. What's the transformation?

- Scale to an expanded frame, translate by $(-\frac{1}{2}, -\frac{1}{2})$, scale

to a shrunken coordinate frame.

- $\mathbf{P_r} = \mathbf{S}(2,2)\,\mathbf{P_b}, \quad \mathbf{P_b} = \mathbf{T}(-\frac{1}{2}, -\frac{1}{2})\,\mathbf{P_g}, \quad \mathbf{P_g} = \mathbf{S}(\frac{1}{W}, \frac{1}{H})\,\mathbf{P_o}$
- $\mathbf{P_W} = \mathbf{S}^{-1}(\frac{1}{W}, \frac{1}{H})\ \mathbf{T}^{-1}(-\frac{1}{2}, -\frac{1}{2})\ \mathbf{S}^{-1}(2,2)\ \mathbf{P_C}$.
- Viewport transform: $\mathbf{S}(W,H)\ \mathbf{T}(0.5,0.5)\ \mathbf{S}(0.5,0.5) = M$

General Viewport Txform

- General command: `glViewport(l, b, r, t)`.
- Translate so the range of x, y is $0 \cdots 2$.
- Scale so x varies from 0 to $(r - l)$ and y varies from 0 to $(t - b)$.
- Translate so x range is l to r and y range is b to t.

- Matrix for this? $\mathbf{T}(l, b)\ \mathbf{S}(\frac{r-l}{2}, \frac{t-b}{2})\ \mathbf{T}(1, 1)$

$$= \begin{bmatrix} \frac{r-l}{2} & 0 & \frac{r+l}{2} \\ 0 & \frac{t-b}{2} & \frac{t+b}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

- $[-1 \;\; -1 \;\; 1]^{\mathsf{T}}$ maps to $[l \;\; b \;\; 1]^{\mathsf{T}}$.

- $[1 \;\; 1 \;\; 1]^{\mathsf{T}}$ maps to $[r \;\; t \;\; 1]^{\mathsf{T}}$.