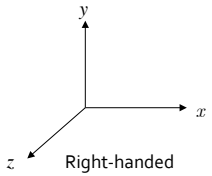
EECS 487: Interactive Computer Graphics

Lecture 11:

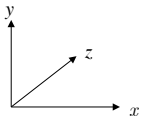
- 3D Transforms
- Rodrigues Formula
- Change of Basis

3D Transformations

Coordinate systems:



Right-handed



Left-handed

We will use a right-handed system

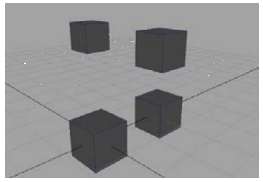
Generally 2D transforms extends to 3D in a straightforward manner: just add on the z-dimension to points, vectors, and matrices

Only rotation is a bit complicated . . .

3D Transforms

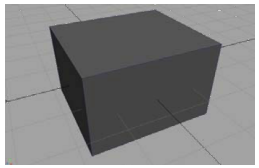
Translation

$$\mathbf{p}' = \mathbf{T}\mathbf{p} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+t_x \\ y+t_y \\ z+t_z \\ 1 \end{bmatrix}$$



Scaling

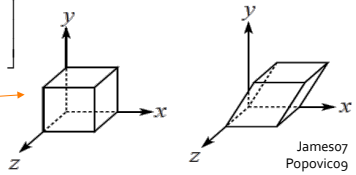
$$\mathbf{p}' = \mathbf{S}\mathbf{p} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ 1 \end{bmatrix}$$



Shear

$$\mathbf{p}' = \mathbf{H}\mathbf{p} = \begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+h_{xy}y+h_{xz}z \\ h_{yx}x+y+h_{yz}z \\ h_{zx}x+h_{zy}y+z \\ 1 \end{bmatrix}$$

shear x by y



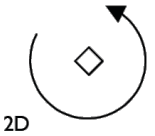
James07
Popovic09

3D Rotation

More involved than translation or scaling

Also more complex than in 2D

- 2D: only need to specify amount of rotation
- 3D: need amount and axis of rotation
  - 3D rotation is about a vector, not just about a point
    - there are many more 3D rotations about a point than in 2D: rotations can cover the whole 3D space around a given point, not just a plane around the point



2D

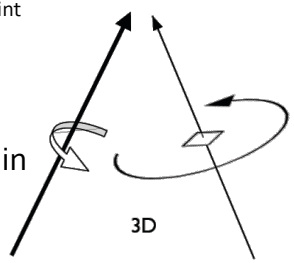
A 2D rotation implicitly rotates about z

3D rotations are orthonormal:

- $\mathbf{R}^{-1}=\mathbf{R}^T$; $|\mathbf{R}| = 1 \neq -1$

Preserve lengths and distance to origin

3D rotations do not commute!

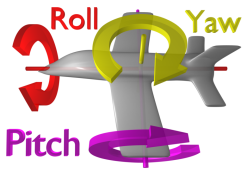


3D

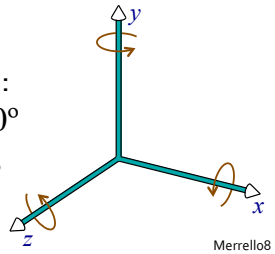
O'Brien08, James07

Axis Aligned 3D Rotation

Axis of rotation	Direction of positive rotation	Common names
x	y to z	lean/pitch/tilt
y	z to x	turn/yaw/ heading/pan
z	x to y	roll/bank



Positive rotation is counter-clockwise: looking down one coordinate axis, a 90° counter-clockwise rotation transforms one positive axis into another



Merrello8
O'Brieno8

Axis-Aligned 3D Rotations

roll

$$\mathbf{R}_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

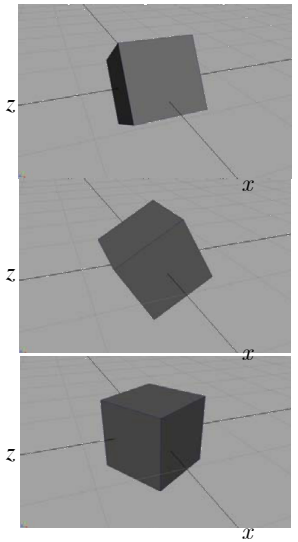
tilt/pitch

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

pan/yaw

$$\mathbf{R}_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

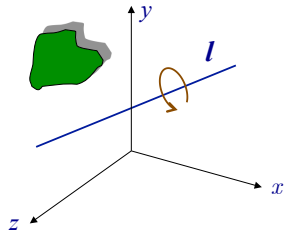
a.k.a. "direction-cosine" matrices



Jamesо7

Rotation About Arbitrary Axis

Rotate θ degrees about some line $\boldsymbol{l}$: $\mathbf{p}' = \mathbf{R}(\theta, \boldsymbol{l})\mathbf{p}$



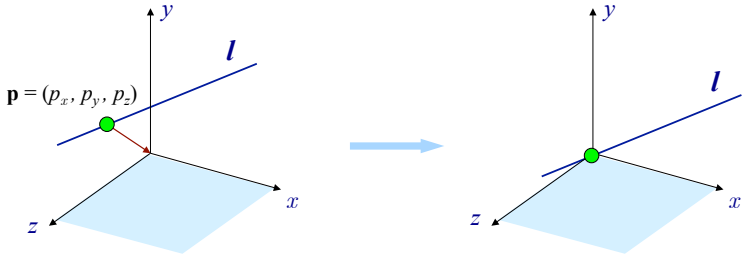
We only know how to rotate about the coordinate axes, so use similar trick as in 2D:

1. translate the rotation axis to the origin
2. rotate the rotation axis around until it coincides with one of the coordinate axes
3. rotate the object, and
4. rotate and translate the rotation axis back

Merrello8

Rotation About Arbitrary Axis

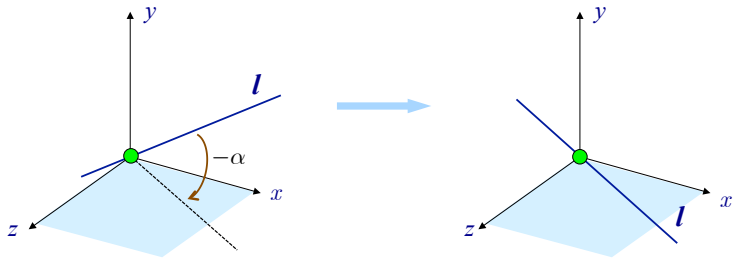
Translate to get a point on $\boldsymbol{l}$ to the origin:



Merrello8

Rotation About Arbitrary Axis

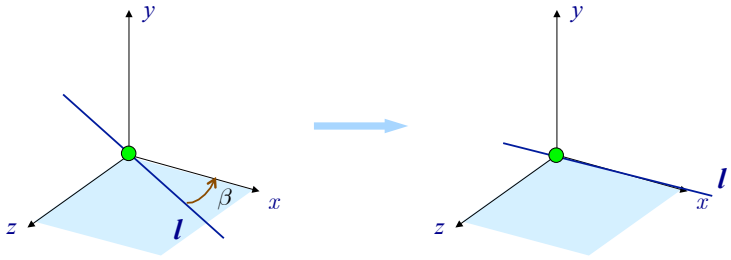
Rotate to get $\boldsymbol{l}$ onto a coordinate plane:
rotate $-\alpha$ about the z-axis to put the line in the xz plane:



$$\mathbf{R}_z(-\alpha)=\begin{bmatrix}\cos(-\alpha) & -\sin(-\alpha) & 0 & 0\\ \sin(-\alpha) & \cos(-\alpha) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}\cos(\alpha) & \sin(\alpha) & 0 & 0\\ -\sin(\alpha) & \cos(\alpha) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Rotation About Arbitrary Axis

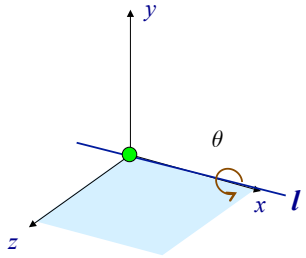
Rotate to get $\boldsymbol{l}$ onto a coordinate axis:
rotate β about the y axis to put line on x axis:



$$\mathbf{R}_y(\beta)=\begin{bmatrix}\cos\beta & 0 & \sin\beta & 0\\ 0 & 1 & 0 & 0\\ -\sin\beta & 0 & \cos\beta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Rotation About Arbitrary Axis

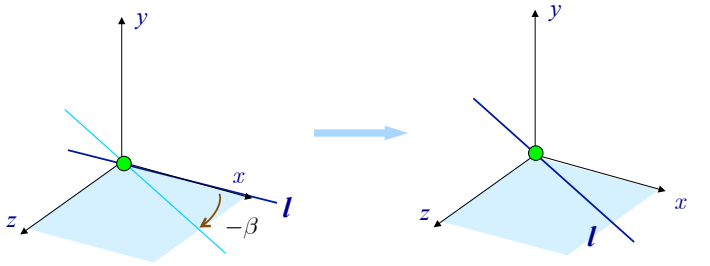
Perform desired rotation (rotate by θ about x-axis)



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

Rotation About Arbitrary Axis

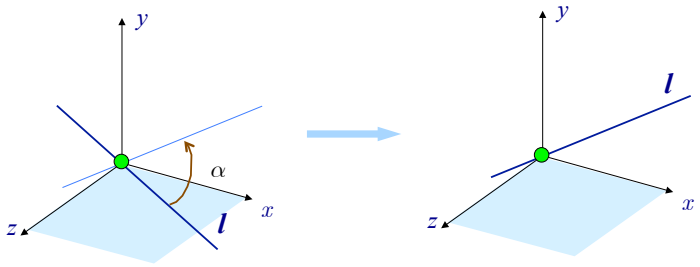
Inverse rotate to get $\boldsymbol{l}$ back into coordinate plane:
rotate $-\beta$ about the y axis to put line back in xz plane:



$$\mathbf{R}_y(-\beta)=\begin{bmatrix}\cos(-\beta) & 0 & \sin(-\beta) & 0\\ 0 & 1 & 0 & 0\\ -\sin(-\beta) & 0 & \cos(-\beta) & 0\\ 0 & 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}\cos\beta & 0 & -\sin\beta & 0\\ 0 & 1 & 0 & 0\\ \sin\beta & 0 & \cos\beta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Rotation About Arbitrary Axis

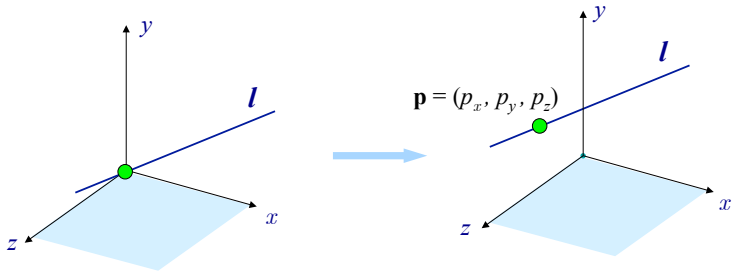
Inverse rotate to take $\boldsymbol{l}$ out of the coordinate plane: rotate α about z:



$$\mathbf{R}_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation About Arbitrary Axis

Invert the initial translation to get the point at the origin back to its original location:



Rotation About Arbitrary Axis

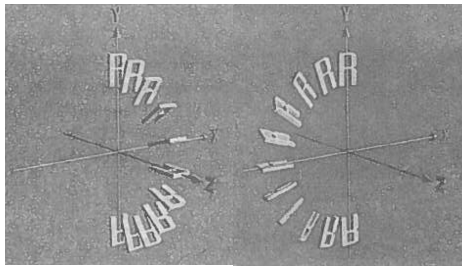
The complete transformation:

$$\mathbf{R}_{arb} = \mathbf{T}(\mathbf{p})\mathbf{R}_z(\alpha)\mathbf{R}_y(-\beta)\mathbf{R}_x(\theta)\mathbf{R}_y(\beta)\,\mathbf{R}_z(-\alpha)\mathbf{T}(-\mathbf{p})$$

This method specifies rotation about arbitrary axis as rotations about 3 angles, called Euler Angles

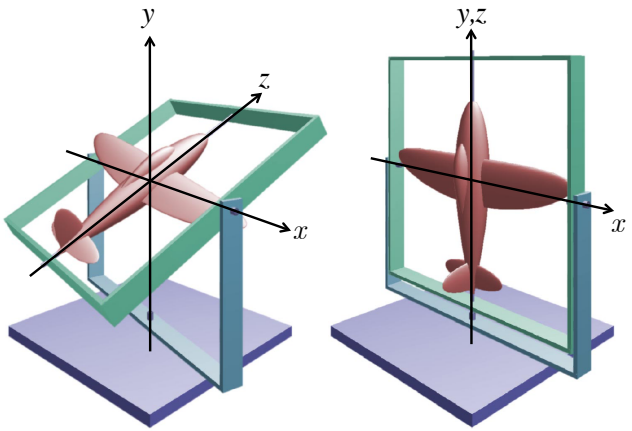
Problems with Euler Angles:

- result is non-unique, order dependent →
- can't add rotations ($90°$ cw + $90°$ ccw = 0, not $\mathbf{I}$), can't compose rotations
- Gimbal lock
- difficult to interpolate angles in animation



Gimbal Lock

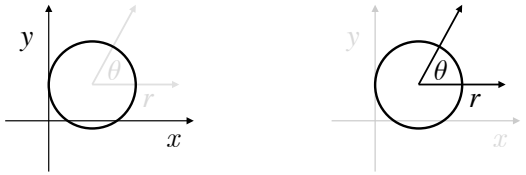
$90°$ rotation in one axis, reduces degree of freedom by one: rotating around y becomes the same as rotating around z



Coordinate Systems

The use of coordinate systems is fundamental to computer graphics

- to describe the locations of points and directions in space
- different ones represent the same point in different ways



- some operations are easier in one system than in another
- multiple coordinate systems make graphics algorithms easier to understand and implement

Chenney

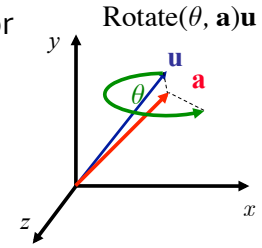
Rodrigues Formula

Direct representation of arbitrary rotation

A.k.a. axis-angle, exponential map, or angular displacement vector

Rotate θ degrees about some axis $\mathbf{a}$

Prevents Gimbal lock (but still has the angles interpolation problem)



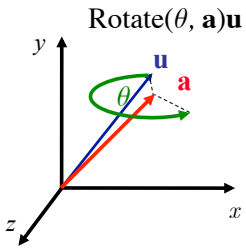
Used in OpenGL `glRotate(`θ`, a`$_x$`, a`$_y$`, a`$_z$`)`

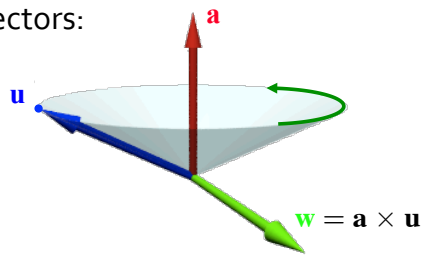
Rodrigues Formula

Look at the problem different: change the basis!

Define a "natural" basis for rotation in terms of three defining vectors:

- the rotation axis **a**, normalized
- the vector being rotated **u**
- a vector perpendicular to both: $\mathbf{w} = \mathbf{a} \times \mathbf{u}$





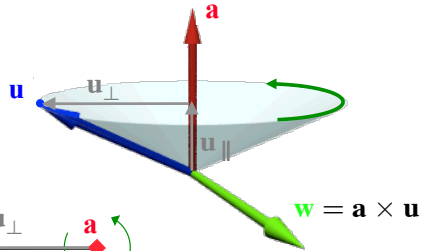
$$\text{Rotate}(\theta, \mathbf{a})\mathbf{u} = \cos\theta\,\mathbf{u} + (1-\cos\theta)\,(\mathbf{a}\cdot\mathbf{u})\mathbf{a} + \sin\theta\,(\mathbf{a}\times\mathbf{u})$$

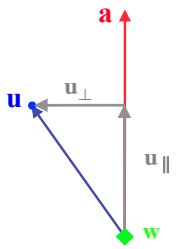
Lozano-Perez&Popovic

Rodrigues Formula

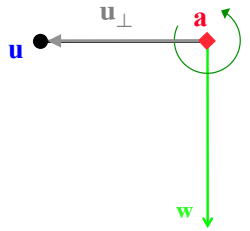
Vector $\mathbf{u}$ has two parts:

- $\mathbf{u}_{\parallel}$ that remains stationary
- $\mathbf{u}_{\perp}$ that rotates like a 2D point





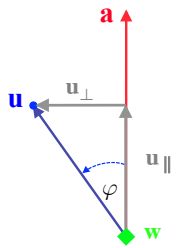
front view



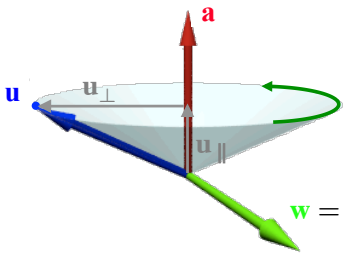
top view

O'Brien

Rodrigues Formula

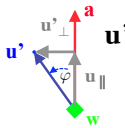


front view



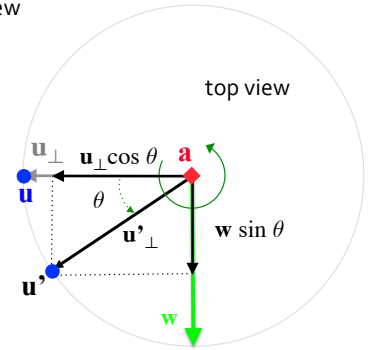
$$|w| = |a \times u|$$
$$= |a||u| \sin\varphi, \ a \text{ normalized}$$
$$= |u| \sin\varphi = |u| \frac{|u_\perp|}{|u|}$$
$$= |u_\perp|$$

Rodrigues Formula



side view

$$u' = \text{Rotate}(\theta, a)u$$
$$= u_\parallel + u'_\perp$$
$$= u_\parallel + u_\perp \cos\theta + w \sin\theta$$



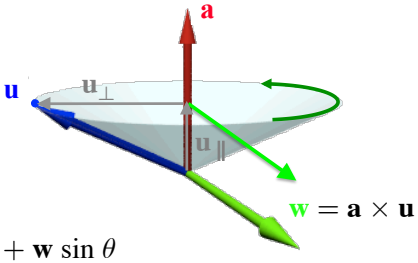
$$u_\perp \cos\theta = u_\perp \left(\frac{u_\perp \cdot u'_\perp}{\|u_\perp\| \|u'_\perp\|} \right)$$
$$= \frac{u_\perp}{\|u_\perp\|} \left(\frac{u_\perp \cdot u'_\perp}{\|u'_\perp\|} \right)$$
$$= \frac{u_\perp}{\|u_\perp\|} \left(\frac{u_\perp}{\|u_\perp\|} \cdot u'_\perp \right)$$
$$= \text{Proj}(u'_\perp) \text{ on } norm(u_\perp)$$

Similarly, Proj($u'_\perp$) on w, given $|w| = |u_\perp|$, is $w \cos(\pi/2 - \theta) = w \sin\theta$

Rodrigues Formula

$$u' = u_\parallel + u_\perp \cos\theta + w \sin\theta$$

$$u_\parallel = (a \cdot u)a$$
$$u_\perp = u - u_\parallel = u - (a \cdot u)a$$



$$u' = (a \cdot u)a + (u - (a \cdot u)a) \cos\theta + w \sin\theta$$

$$\text{Rotate}(\theta, a)u = \cos\theta \, u + (1 - \cos\theta)(a \cdot u)a + \sin\theta \, (a \times u)$$

"natural" basis of u rotating around a

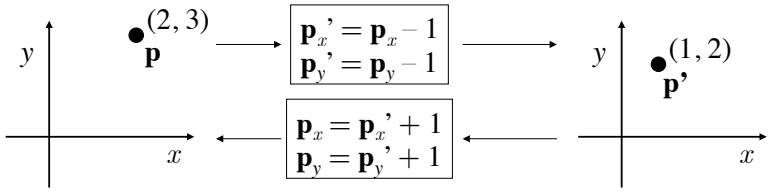
EECS 487: Interactive Computer Graphics

Lecture 11:

- 3D Transforms
- Rodrigues Formula
- Change of Basis

What are Transformations?

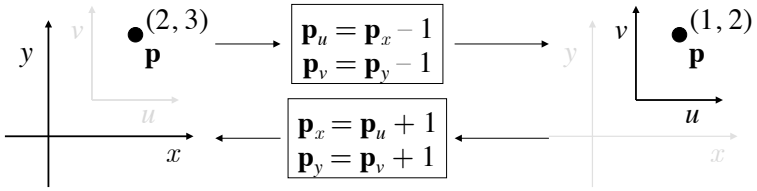
Transformations modify an object's shape and/or location in one coordinate system



This is the common understanding of transformations and one we will generally use

What are Transformations? (Alternate Take)

Transformations convert points between coordinate systems



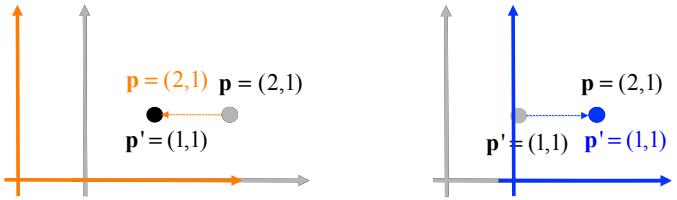
This interpretation sometimes makes graphics algorithms easier to understand and implement

Translation as Change of Basis

E.g., negative translation means either move point backward (**Tp**), or move coordinate system forward ($\mathbf{T}^{-1}\mathbf{I}$)

$\mathbf{p}' = \mathbf{T}\mathbf{p}$
$\mathbf{I}\mathbf{p}' = (\mathbf{T}\mathbf{I})\mathbf{p}$

$\mathbf{p} = \mathbf{T}^{-1}\mathbf{p}'$
$\mathbf{I}\mathbf{p} = (\mathbf{T}^{-1}\mathbf{I})\mathbf{p}'$

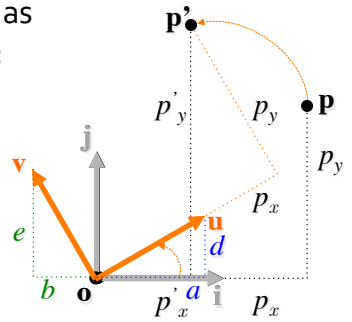


Linear Transform as Change of Basis

Think of a linear transformation as a change of coordinate systems:

$$\mathbf{p}' = \mathbf{M}\mathbf{p} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} = p_x\mathbf{u} + p_y\mathbf{v}$$



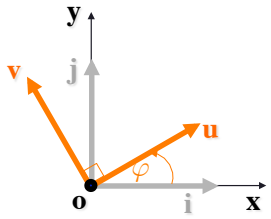
Rotation as Change of Basis

Consider a rotation transform $\mathbf{R}(\varphi, \mathbf{o})$ operating on the standard basis

$$\mathbf{R}(\varphi,\mathbf{o})\mathbf{i} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\varphi \\ \sin\varphi \end{bmatrix} = \mathbf{u}$$

$$\mathbf{R}(\varphi,\mathbf{o})\mathbf{j} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin\varphi \\ \cos\varphi \end{bmatrix} = \mathbf{v}$$

but these are just the columns of $\mathbf{R}$!



The column space of a matrix is the coordinate space formed by the columns of the matrix, taken as basis vectors

- e.g., the column space of $\mathbf{R}(\varphi, \mathbf{o})$ is the space formed by vectors $\mathbf{u}$ and $\mathbf{v}$

⇒ any matrix transforms the standard basis vectors into a set of vectors that form the columns of the matrix (expressed in standard basis)

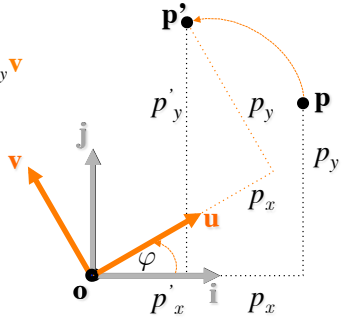
Levoyo8

Rotation as Change of Basis

$$\mathbf{p}' = \mathbf{R}\mathbf{p}$$
$$\mathbf{I}\mathbf{p}' = (\mathbf{R}\mathbf{I})\mathbf{p}$$

$$\mathbf{p}' = \mathbf{R}(\varphi,\mathbf{o})\mathbf{p} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}\begin{bmatrix} p_x \\ p_y \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{u} & \mathbf{v} \end{bmatrix}\begin{bmatrix} p_x \\ p_y \end{bmatrix} = p_x\mathbf{u} + p_y\mathbf{v}$$



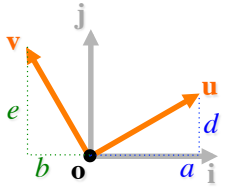
Transformation ⇔ Change of Basis

Transformations operate on the coordinates of point ($\mathbf{p}$) or vector ($\mathbf{v}$) in one coordinate system: $\mathbf{M}\mathbf{p}$ or $\mathbf{M}\mathbf{v}$

Or, transformations convert points or vectors between coordinate systems: from the Cartesian standard space to the column space of $\mathbf{M}$

The column space of a matrix is the coordinate space formed by the columns of the matrix, taken as vectors (expressed in the standard basis)

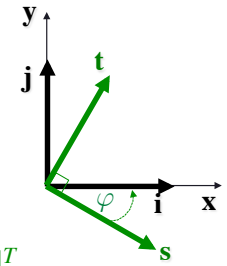
$$\mathbf{M} = \begin{bmatrix} a & b \\ d & e \end{bmatrix},\ \mathbf{u} = \begin{bmatrix} a \\ d \end{bmatrix},\ \mathbf{v} = \begin{bmatrix} b \\ e \end{bmatrix}$$



Inverse Rotation

Let $\mathbf{s}$ and $\mathbf{t}$ be the two rows of $\mathbf{R}(\varphi,\mathbf{o})$:

$$\mathbf{R}(\varphi,\mathbf{o}) = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{matrix} \mathbf{s} \\ \mathbf{t} \end{matrix}$$



$\mathbf{s} = [\cos\varphi \quad -\sin\varphi]^T$ and $\mathbf{t} = [\sin\varphi \quad \cos\varphi]^T$

$$\mathbf{R}(\varphi,\mathbf{o})\mathbf{s} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}\begin{bmatrix} \cos\varphi \\ -\sin\varphi \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \mathbf{i}$$

$$\mathbf{R}(\varphi,\mathbf{o})\mathbf{t} = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}\begin{bmatrix} \sin\varphi \\ \cos\varphi \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \mathbf{j}$$

standard basis!

Inverse Rotation

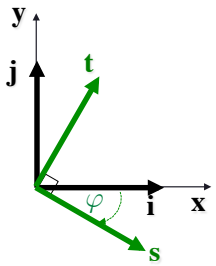
$$\mathbf{R}(\varphi,\mathbf{o})[\ \mathbf{s}\quad \mathbf{t}\ ]$$

$$= \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}$$

$$\therefore \mathbf{R}\mathbf{R}^T = \mathbf{I} = \mathbf{R}\mathbf{R}^{-1}$$

$$\Rightarrow \mathbf{R}^{-1} = \mathbf{R}^T$$



The row space of a matrix is the coordinate space formed by the rows of the matrix, taken as basis vectors

The inverse of an orthogonal matrix transforms the standard basis vectors into the row space of the matrix

$\Rightarrow \mathbf{R}$ is an orthogonal matrix!

Inverse Rotation as Change of Basis

Since the rotation transform is an orthogonal matrix:

$$\mathbf{R}^{-1}(\varphi,\mathbf{o}) = \mathbf{R}^T = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix}$$

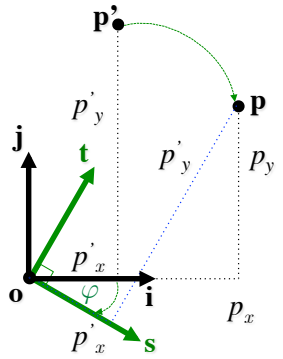
$$\mathbf{p} = \mathbf{R}^{-1}\mathbf{p}'$$

$$\mathbf{I}\mathbf{p} = (\mathbf{R}^{-1}\mathbf{I})\mathbf{p}'$$

$$\mathbf{R}^{-1}(\varphi,\mathbf{o})\mathbf{p}' = \begin{bmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} p'_x \\ p'_y \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{s} & \mathbf{t} \end{bmatrix} \begin{bmatrix} p'_x \\ p'_y \end{bmatrix} = p'_x\mathbf{s} + p'_y\mathbf{t}$$

$$= \mathbf{p}$$



OpenGL 2.1's Current Transformation Matrix

OpenGL 2.1 maintains a current transformation matrix (CTM = **C**)
- **C** defines the current coordinate system
- all geometry is defined in the current coordinate system
- all geometry is transformed by **C**

Transformation matrices are post-multiplied with the CTM
- the last transform specified is the first performed
- **C'** = **CT**

OpenGL 2.1 has four CTMs:
- `GL_MODELVIEW`: contains the composite of modeling and viewing matrices
- `GL_PROJECTION`
- `GL_TEXTURE`: stretching, moving, rotating texture
- `GL_COLOR`: for image processing
- to change a CTM, specify which one with `glMatrixMode(mode)`

Matrix Operations

Specify which matrix:

```
glMatrixMode(mode)
```

Assign matrix (mainly used to load identity):

```
glLoadIdentity()
glLoadMatrix[fd]()
glLoadTransposeMatrix[fd]()
```

Compose matrix, post multiply:

```
glMultMatrix[fd]()
glMultTransposeMatrix[fd]()
```

- used in transforming normals, e.g.

Modeling Transforms

Translate:

```
glTranslate[fd](x,y,z)
```

Rotate:

```
glRotate[fd](angle,x,y,z) // angle in degrees
```

Scale:

```
glScale[fd](x,y,z)
```

Example:	Results
`glMatrixMode(GL_MODELVIEW);`	**C**
`glLoadIdentity();`	**C ← I**
`glScalef(…);`	**C ← IS**
`glRotatef(…);`	**C ← ISR**
`glTranslatef(…);`	**C ← ISRT**
`glVertex3f(…);`	**ISRTv**

Matrix Stack and Stack Operations

The CTM may be pushed onto and popped from a stack

- Modelview stack: ≥ 32 4×4 matrices (e.g., Mac OS X: 40)
- Projection stack: ≥ 2 4×4 matrices (e.g., Mac OS X: 4)

Matrix stack operation:

`glPushMatrix()`: pushes the stack down by one, duplicating the CTM, i.e., after a `glPushMatrix()` call, the matrix on top of the stack is identical to the one below it

`glPopMatrix()`

Useful for hierarchical modeling, when different parts of an object transform in different ways