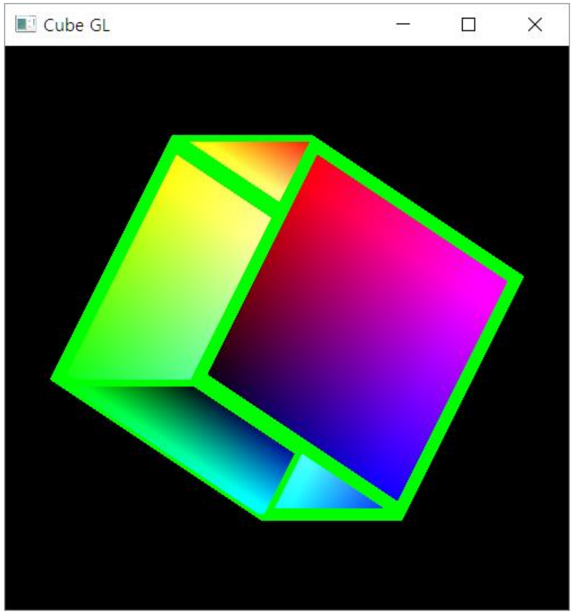
Chapter 3: Geometric Objects and Transformations

Sang Il Park

Dept. of Software

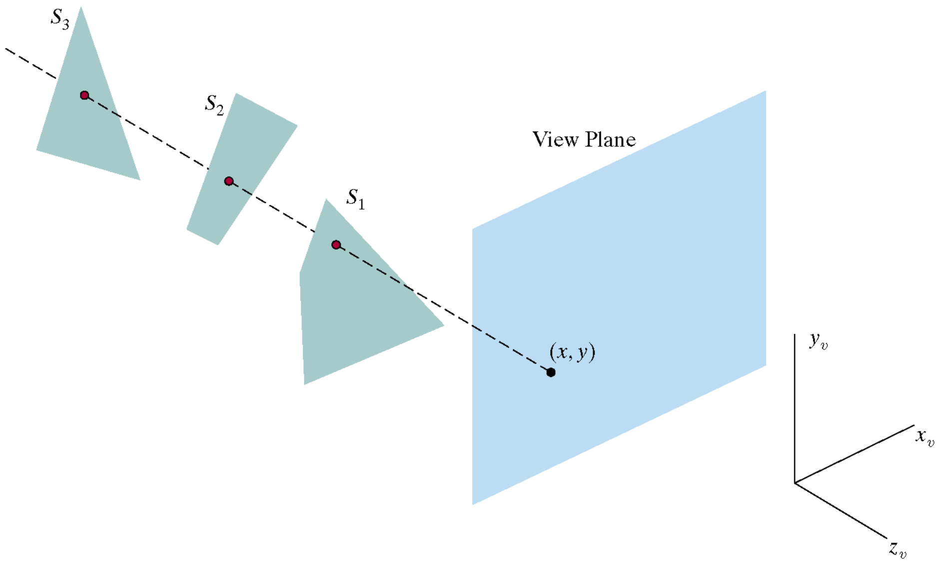
One little problem

- Incorrect Depth Handling



Depth-Buffer (Z-Buffer)

- ***Z-Buffer*** has memory corresponding to each pixel location for storing the current depth value (distance from view plane)
  - Useful for determining whether a new drawing pixel is visible or not
    - Is ***visible*** when the its distance is closer than the previously stored one
    - Is ***not visible*** when it is farer



Enabling the Depth Buffer in OPENGL

- On Initialization :

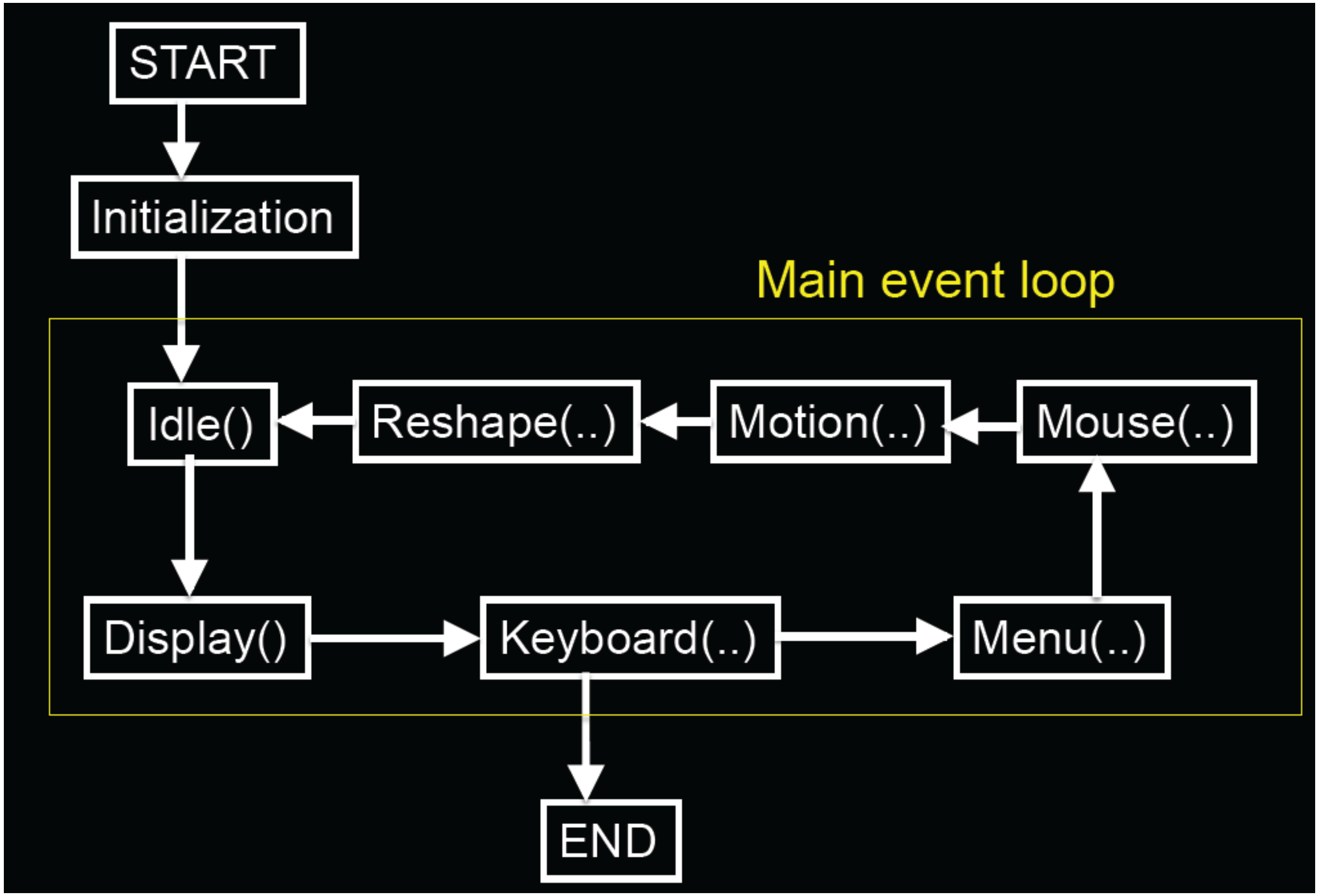
```
glutInitDisplayMode(GLUT_DOUBLE |
                    GLUT_RGBA |
                    GLUT_DEPTH);
```

- On Drawing :

```
glClear (GL_COLOR_BUFFER_BIT |
         GL_DEPTH_BUFFER_BIT);
glEnable(GL_DEPTH_TEST);
```

Interaction: Callbacks

GLUT Program with Callbacks



Event Types

- Window: resize, expose, iconify
- Mouse: click one or more buttons
- Motion: move mouse
- Keyboard: press or release a key
- Idle: non-event
  - Define what should be done if no other event is in queue

Callbacks

- Programming interface for event-driven input
- Define a *callback function* for each type of event the graphics system recognizes
- This user-supplied function is executed when the event occurs
- GLUT example: `glutMouseFunc(mymouse)`

mouse callback function

GLUT callbacks

GLUT recognizes a subset of the events recognized by any particular window system (Windows, X, Macintosh)

- **glutDisplayFunc**
- **glutMouseFunc**
- **glutReshapeFunc**
- **glutKeyboardFunc**
- **glutIdleFunc**
- **glutMotionFunc, glutPassiveMotionFunc**

Types of Callbacks

- Display ( ) : when window must be drawn
- Idle ( ) : when no other events to be handled
- Keyboard (unsigned char key, int x, int y) : key pressed
- Menu (...) : after selection from menu
- Mouse (int button, int state, int x, int y) : mouse button
- Motion (...) : mouse movement
- Reshape (int w, int h) : window resize
- Any callback can be NULL

GLUT Event Loop

- Recall that the last line in **`main.c`** for a program using GLUT must be

  `glutMainLoop();`

which puts the program in an infinite event loop

- In each pass through the event loop, GLUT
  - looks at the events in the queue
  - for each event in the queue, GLUT executes the appropriate callback function if one is defined
  - if no callback is defined for the event, the event is ignored

Example : Idling Callback

- Idling Callback is useful for defining a periodic work
  - Ex.) making an animation (rotating a cube and so on)

- How to use:
  1. Registering your function as an Idling Callback Function

```
glutIdleFunc ( myIdle );
```

  2. Implement what you want to do repeatedly

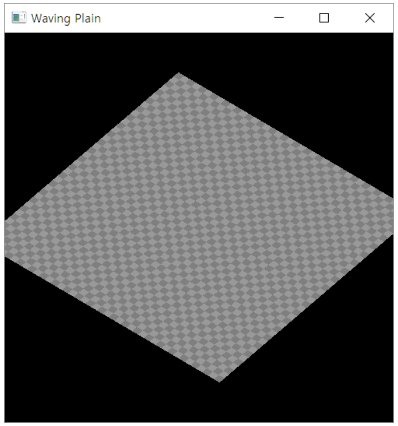
```
void myIdle()
{
        // do some periodic job
        Sleep(16);
        glutPostRedisplay();
}
```

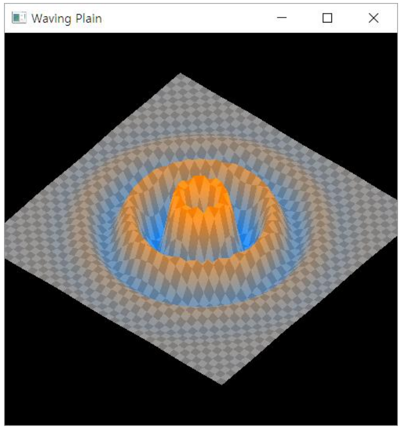
Wait for 16 mille-sec.
(Ensuring 60 FPS)

Invoking redrawing

Homework: A Waving Plain

- Condition:
  - Draw a checker pattern plain (create a “MyPlain” class)
  - Draw waving pattern when [w] key is pressed using only shader
  - Start rotating and “WAVING” when space bar is pressed
  - Increase or decrease the Number of Division when pressing [1] or [2] key.
  - Stop when space bar is pressed again.
  - Quit when [q] is pressed





How to submit:

- Submit your codes to blackboard
  - .cpp file + .h file + vshader.glsl + fshader.glsl

- Also include your report about
  - Ideas
    - Especially how you can get the waving patterns
  - Describe any errors and resolutions you tried and found

- Due date: 9/26(Monday) PM 23:59

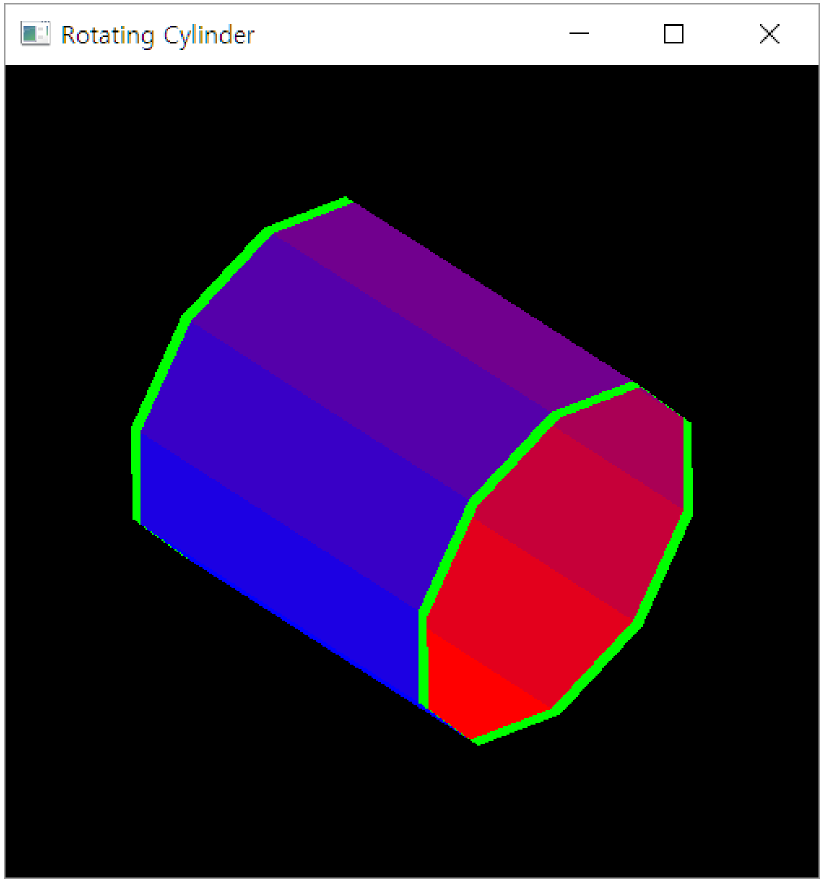
Hint for the homework: Organizing into Classes

- Make MyColorCube Class to increase the readability and portability of the code :

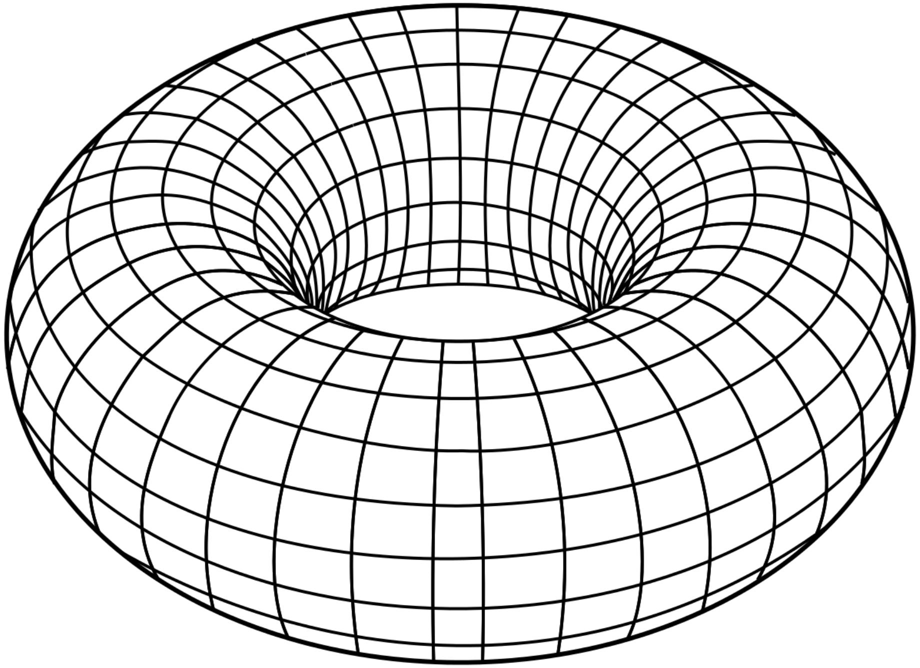
MyColorCube.h

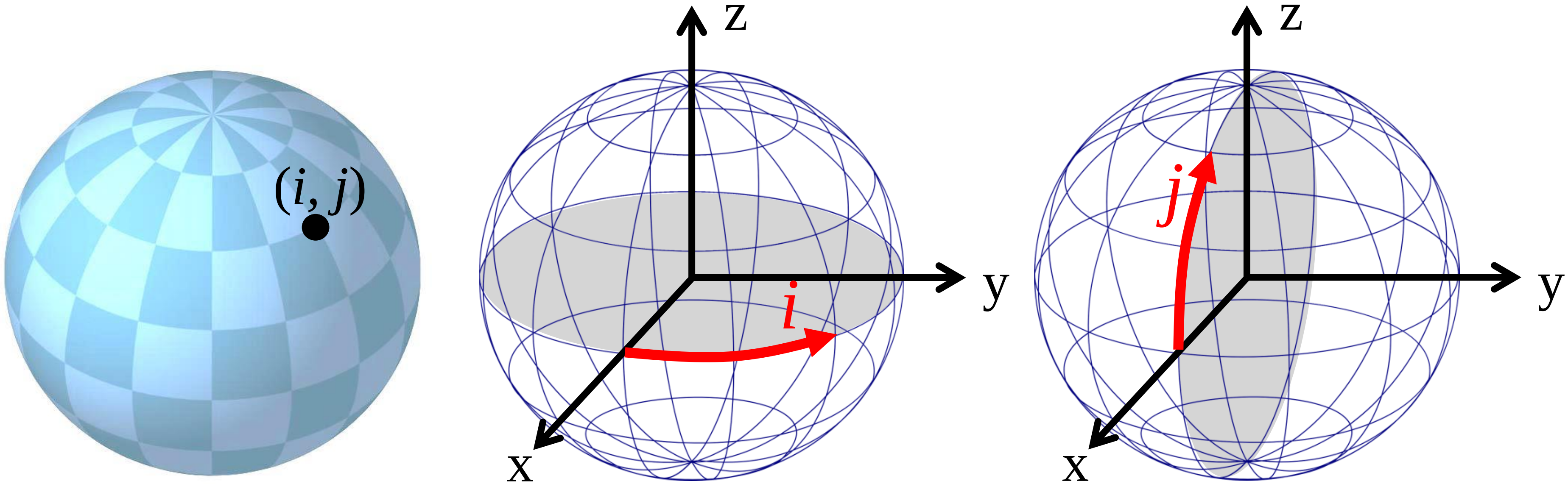
```
class MyColorCube
{
public:

        …

};
```

Coding Practice : A Color Cylinder



Sphere and Torus





(i, j)
z
y
x
i
z
y
x
j

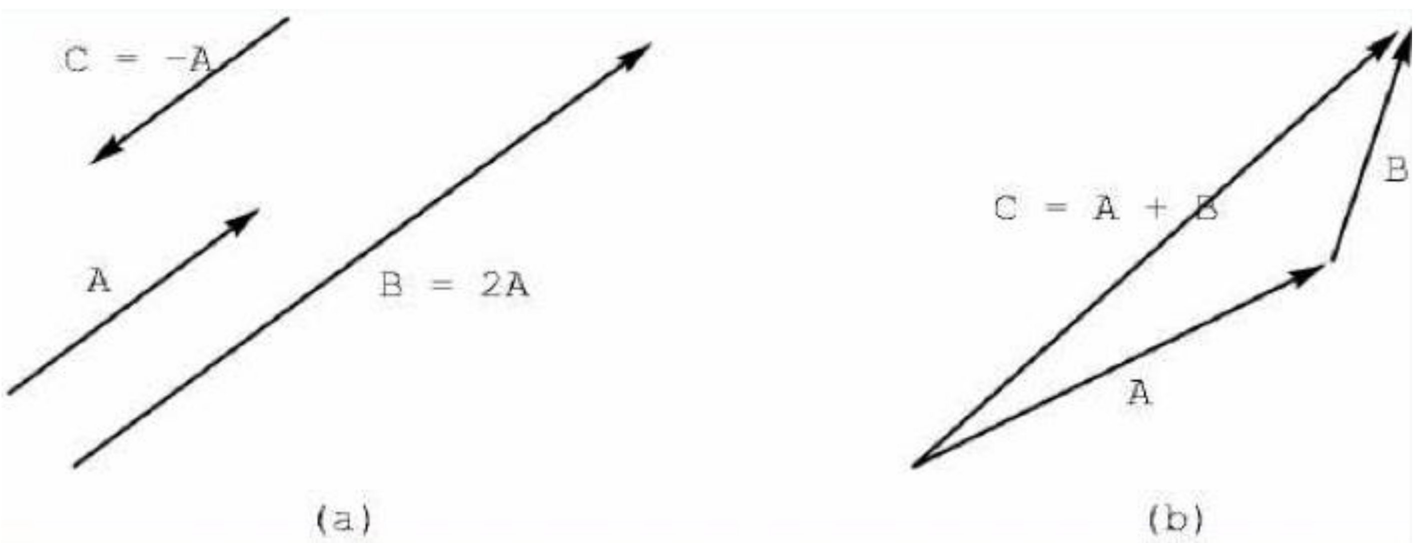
Chapter 3: Geometric Objects and Transformations

Scalars

- Scalars α, β, γ from a *scalar field*
- Operations $\alpha+\beta$, $\alpha\cdot\beta$, 0, 1, $-\alpha$, $(\ )^{-1}$
- "Expected" laws apply
- Examples: rationals or reals with addition and multiplication

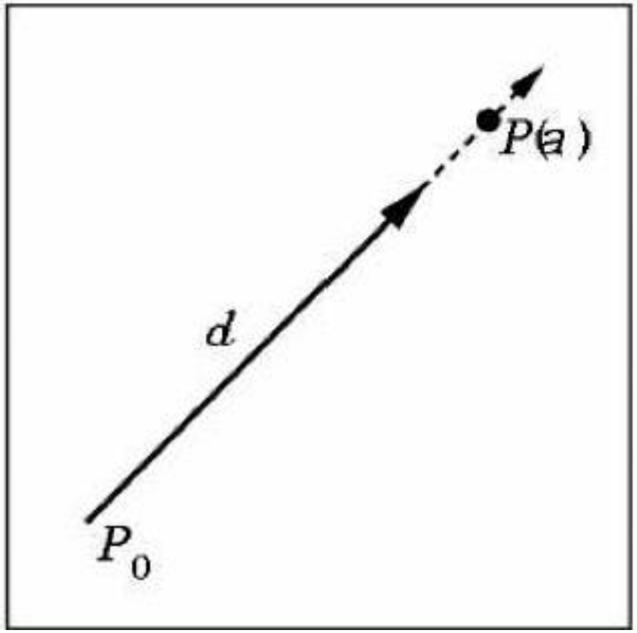
Vectors

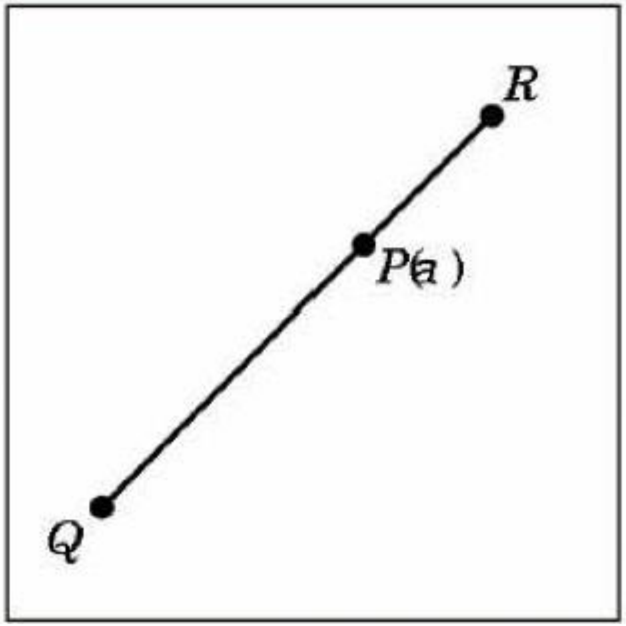
- Vectors *u, v, w* from a *vector space*
- Vector addition *u* + *v* , subtraction *u* - *v*
- Zero vector **0**
- Scalar multiplication α *v*



Lines and line Segments

- Parametric form of line: $\mathbf{P}(\alpha) = \mathbf{P}_o + \alpha\,\mathbf{d}$





- Line segment between *Q* and *R*:

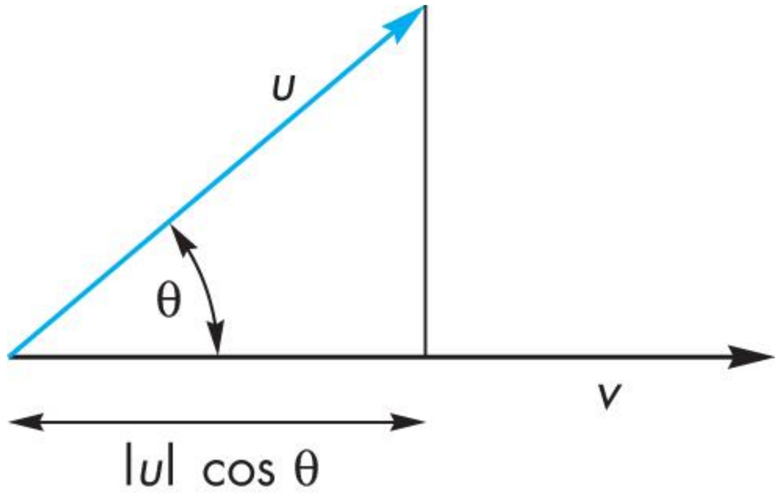
$$\mathbf{P}(\alpha) = (1-\alpha)\mathbf{Q} + \alpha\,\mathbf{R} \quad \textit{for } 0 \le \alpha \le 1$$

Dot Product (Projection)

- Dot product projects one vector onto another vector

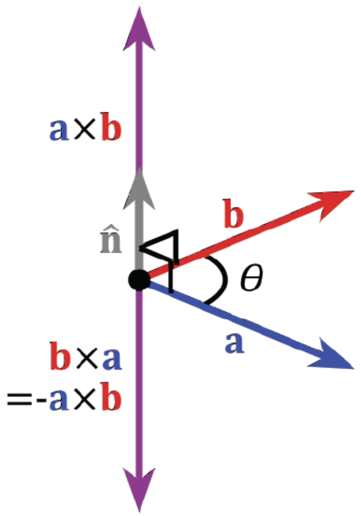
$$\mathbf{u}\cdot\mathbf{v} = \mathbf{u}_1\mathbf{v}_1 + \mathbf{u}_2\mathbf{v}_2 + \mathbf{u}_3\mathbf{v}_3 = |\mathbf{u}||\mathbf{v}|\cos(\theta)$$

$$pr_{\mathbf{v}}\mathbf{u} = (\mathbf{u}\cdot\mathbf{v})\,\mathbf{v}/|\mathbf{v}|^2$$

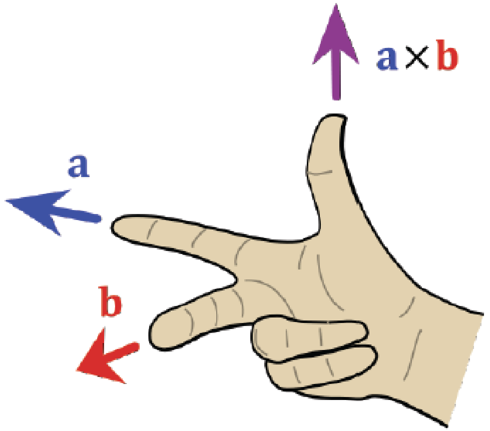


Cross Product

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

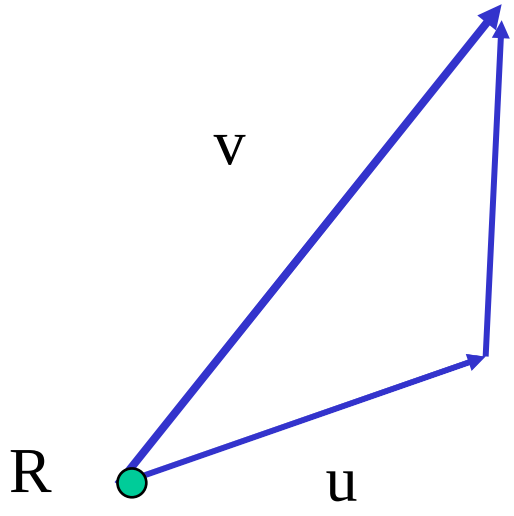


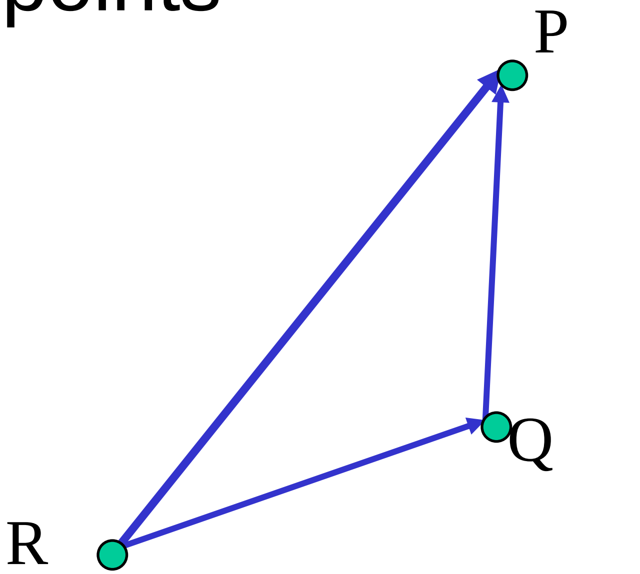
- $|\mathbf{a} \times \mathbf{b}| = |\mathbf{a}||\mathbf{b}||\sin(\theta)|$
- Cross product is perpendicular to both **a** and **b**
- Right-hand rule



Planes

- A plane can be defined by a point and two vectors or by three points



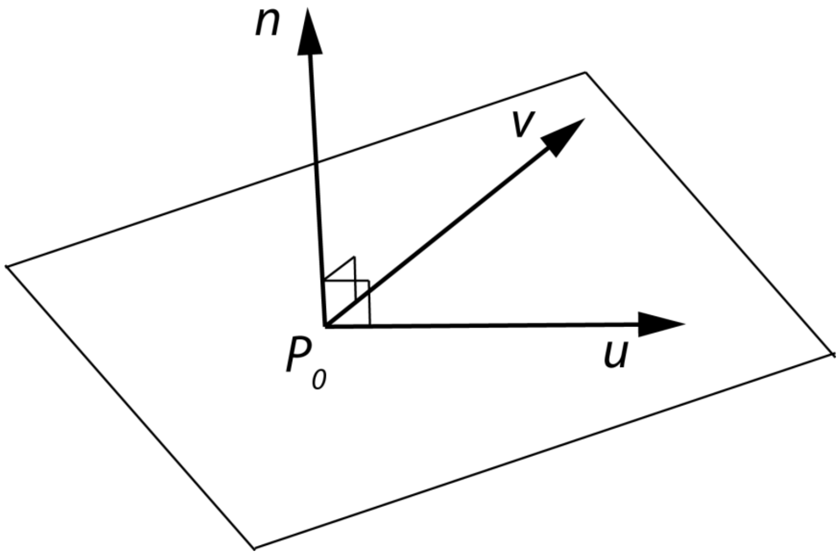


$P(\alpha,\beta)=R+\alpha u+\beta v$

$P(\alpha,\beta)=R+\alpha(Q-R)+\beta(P-Q)$

Planes and normal

- Plane defined by point P_0 and vectors u and v
- u and v should not be parallel
- Parametric form:
  $T(\alpha, \beta) = P_0 + \alpha\, u + \beta\, v$
  (α and β are scalars)

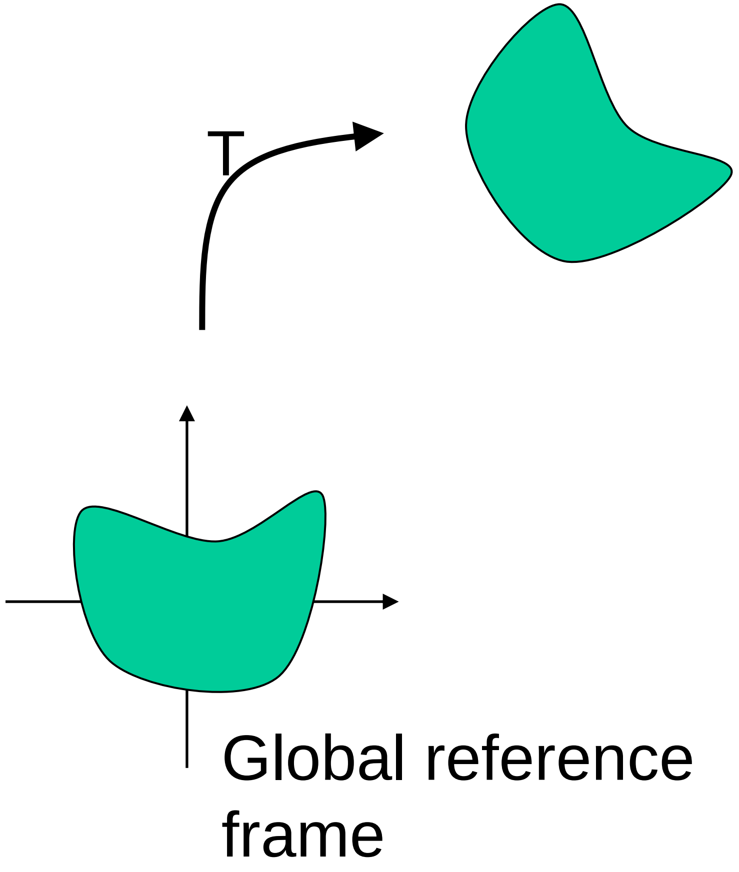


- $n = u \times v / |u \times v|$ is the normal
- $n \cdot (P - P_0) = 0$ if and only if P lies in plane

Geometric Transformations

Transformations

- Linear transformations
- Rigid transformations
- Affine transformations
- Projective transformations



Homogeneous Coordinates

- Any affine transformation between 3D spaces can be represented by a 4x4 matrix

$$T(\mathbf{p}) = \begin{pmatrix} \mathbf{M}_{3\times 3} & \mathbf{T}_{3\times 1} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{p}_{3\times 1} \\ 1 \end{pmatrix}$$

- Affine transformation is *linear* in homogeneous coordinates

Projective Spaces

- Homogeneous coordinates
  - (x, y, z, w) = (x/w, y/w, z/w, 1)
  - Useful for handling perspective projection

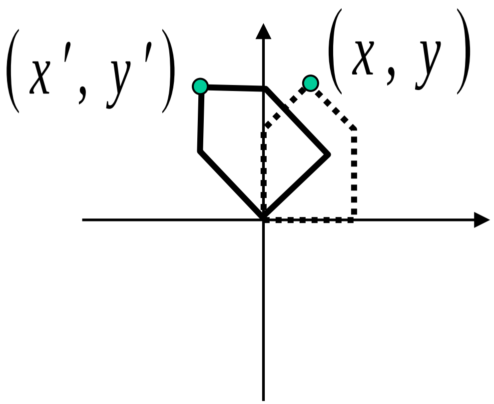
- But, it is algebraically inconsistent !!

$$(1,0,0,1)+(1,1,0,1)=(2,1,0,2)=(1,\frac{1}{2},0,1)$$

$$(1,0,0,1)+(2,2,0,2)=(3,2,0,3)=(1,\frac{2}{3},0,1)$$

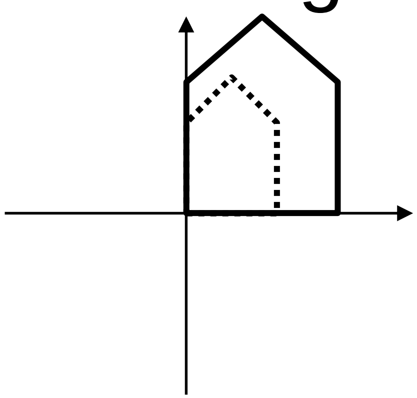
Examples of Affine Transformations

- 2D rotation



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

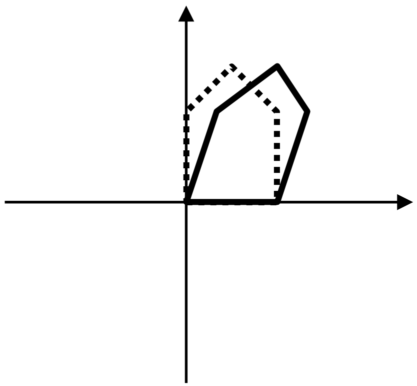
- 2D scaling



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

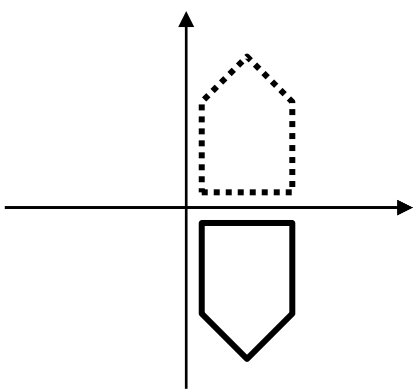
Examples of Affine Transformations

- 2D shear



$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & d & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + dy \\ y \\ 1 \end{pmatrix}$$

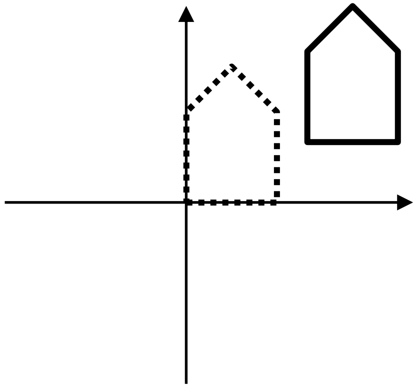
- 2D reflection



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

Examples of Affine Transformations

- 2D translation

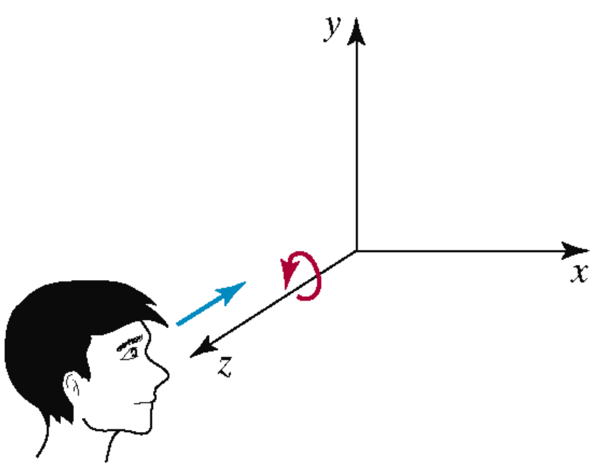


$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

Examples of Affine Transformations

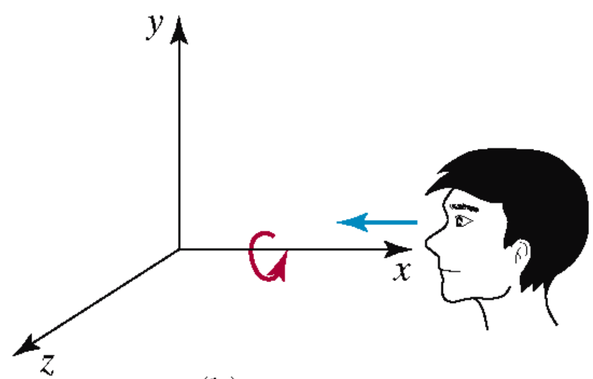
- 3D rotation

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



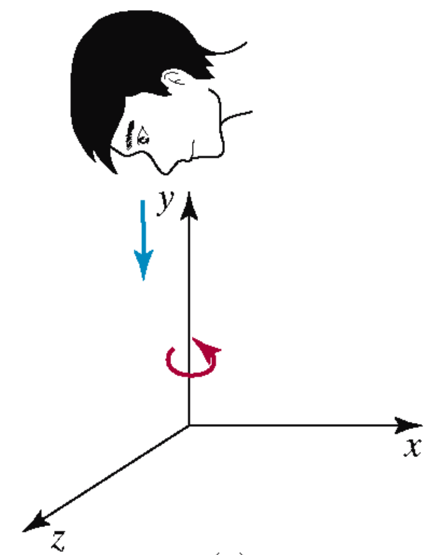
(a)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



(b)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



(c)

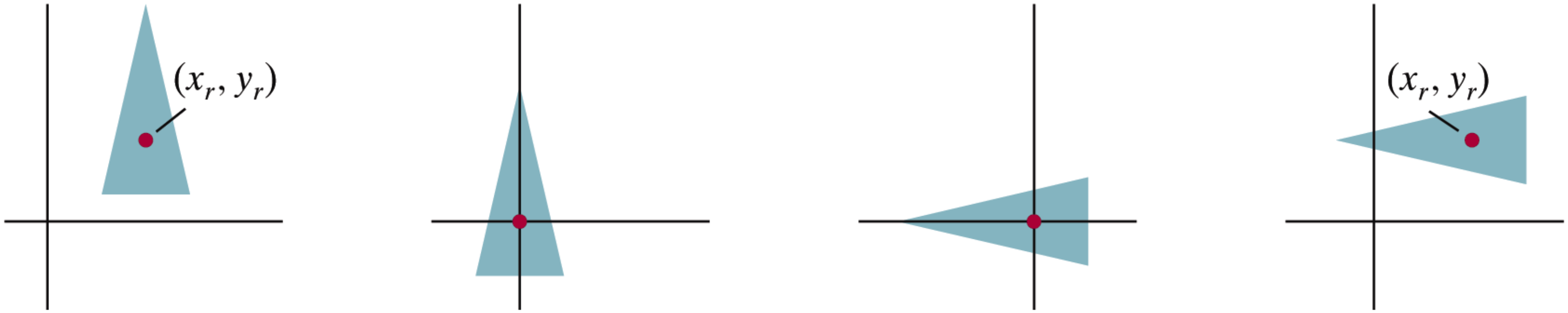
Pivot-Point Rotation

- Rotation with respect to a pivot point (x,y)

$$T(x,y)\cdot R(\theta)\cdot T(-x,-y)$$

$$=\begin{pmatrix}1 & 0 & x\\ 0 & 1 & y\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}1 & 0 & -x\\ 0 & 1 & -y\\ 0 & 0 & 1\end{pmatrix}$$

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



(a) (b) (c) (d)

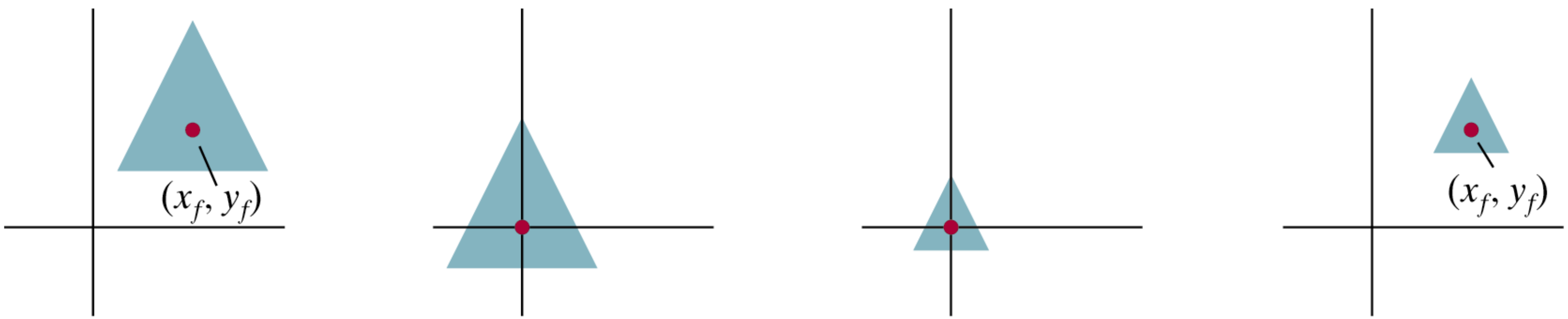
Fixed-Point Scaling

- Scaling with respect to a fixed point (x,y)

$$T(x,y)\cdot S(s_x,s_y)\cdot T(-x,-y)$$

$$=\begin{pmatrix}1 & 0 & x\\ 0 & 1 & y\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}s_x & 0 & 0\\ 0 & s_y & 0\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}1 & 0 & -x\\ 0 & 1 & -y\\ 0 & 0 & 1\end{pmatrix}$$

$$=\begin{pmatrix}s_x & 0 & (1-s_x)\cdot x\\ 0 & s_y & (1-s_y)\cdot y\\ 0 & 0 & 1\end{pmatrix}$$

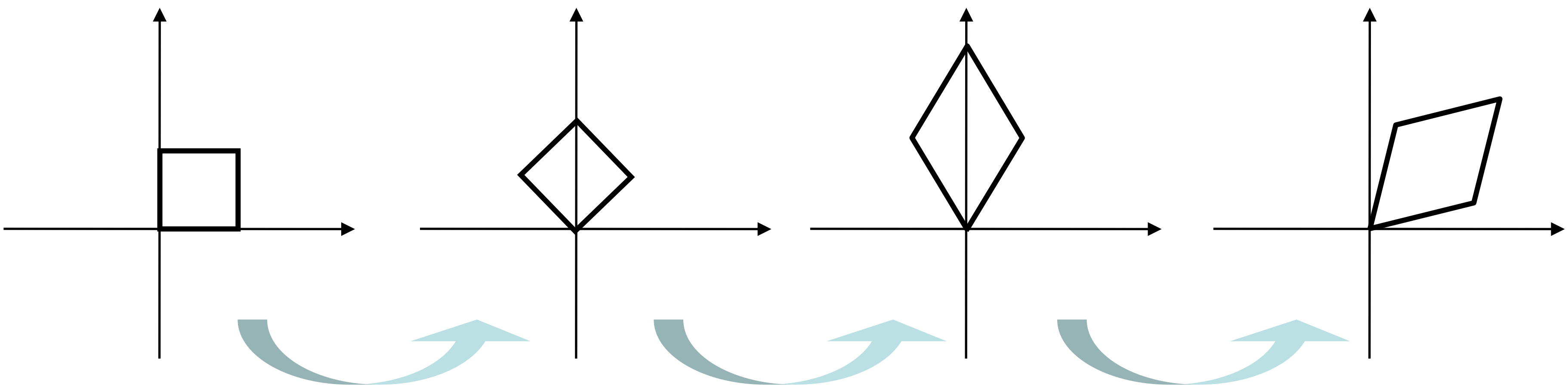


(a) (b) (c) (d)

Scaling Direction

- Scaling along an arbitrary axis

$$R^{-1}(\theta) \cdot S(s_x, s_y) \cdot R(\theta)$$



$R(\theta)$ $\qquad$ $S(s_x, s_y)$ $\qquad$ $R^{-1}(\theta)$

Properties of Affine Transformations

- Any ***affine transformation*** between 3D spaces can be represented as a combination of a ***linear transformation*** followed by ***translation***
- An affine transf. maps ***lines*** to ***lines***
- An affine transf. maps ***parallel lines*** to ***parallel lines***
- An affine transf. preserves ***ratios of distance*** along a line
- An affine transf. does not preserve absolute distances and angles

Rigid Transformations

- A ***rigid transformation*** T is a mapping between affine spaces
  - T maps vectors to vectors, and points to points
  - T preserves distances between all points
  - T preserves cross product for all vectors (to avoid reflection)
- In 3-spaces, T can be represented as

$$T(\mathbf{p}) = \mathbf{R}_{3\times 3}\mathbf{p}_{3\times 1} + \mathbf{T}_{3\times 1}, \quad \text{where}$$

$$\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I} \quad \text{and} \quad \det \mathbf{R} = 1$$

Rigid Body Rotation

- Rigid body transformations allow only rotation and translation

- Rotation matrices form SO(3)
  - Special orthogonal group

$$\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$$ (Distance preserving)

$$\det \mathbf{R} = 1$$ (No reflection)

Rigid Body Rotation

- R is normalized
  - The squares of the elements in any row or column sum to 1

$$R R^T = R^T R = I$$

- R is orthogonal
  - The dot product of any pair of rows or any pair columns is 0

- The rows (columns) of R correspond to the vectors of the principle axes of the rotated coordinate frame

3D Rotation About Arbitrary Axis

- Rotate **u** onto the *z*-axis

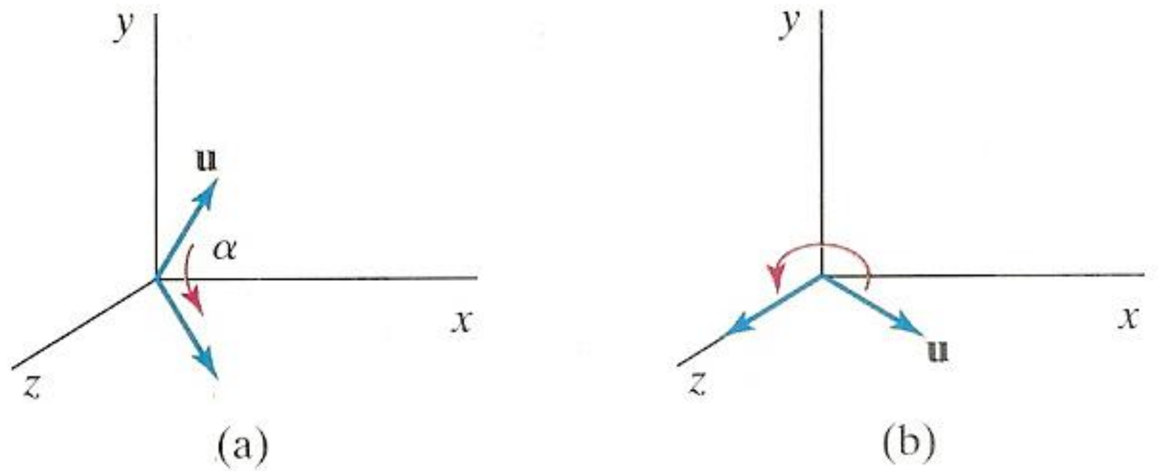


FIGURE 5-45 Unit vector **u** is rotated about the *x* axis to bring it into the *xz* plane (a), then it is rotated around the *y* axis to align it with the *z* axis (b).

3D Rotation About Arbitrary Axis

- Rotate **u** onto the z-axis
  - **u’**: Project **u** onto the yz-plane to compute angle α
  - **u’’**: Rotate **u** about the x-axis by angle α
  - Rotate **u’’** onto the z-asis

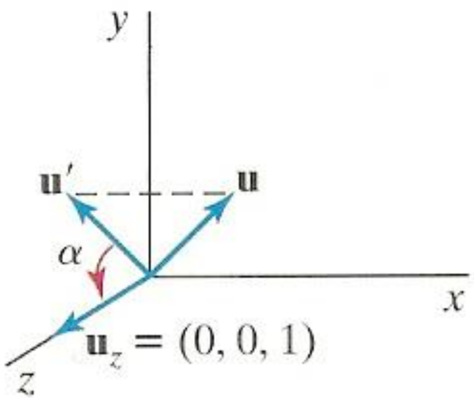


FIGURE 5-46 Rotation of **u** around the x axis into the xz plane is accomplished by rotating $\mathbf{u}'$ (the projection of **u** in the yz plane) through angle α onto the z axis.

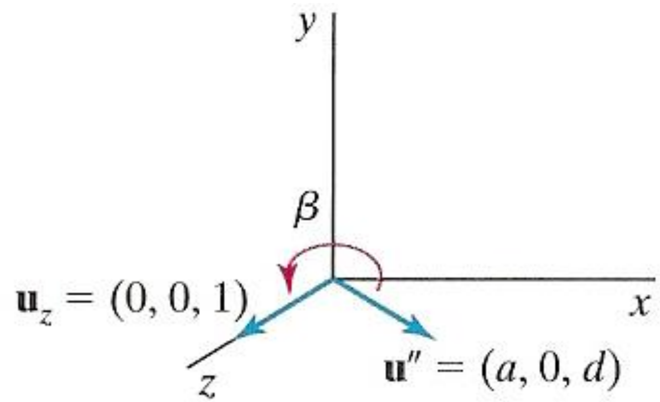


FIGURE 5-47 Rotation of unit vector $\mathbf{u}''$ (vector **u** after rotation into the xz plane) about the y axis. Positive rotation angle β aligns $\mathbf{u}''$ with vector $\mathbf{u}_z$.

3D Rotation About Arbitrary Axis

- Rotate **u'** about the x-axis onto the *z*-axis
  - Let **u**=(a,b,c) and thus **u'**=(0,b,c)
  - Let $\mathbf{u}_z$=(0,0,1)

$$\cos\alpha = \frac{\mathbf{u}'\cdot\mathbf{u}_z}{\|\mathbf{u}'\|\,\|\mathbf{u}_z\|} = \frac{c}{\sqrt{b^2+c^2}}$$

$$\mathbf{u}'\times\mathbf{u}_z = \mathbf{u}_x\|\mathbf{u}'\|\,\|\mathbf{u}_z\|\sin\alpha$$
$$= \mathbf{u}_x\cdot b$$

$$\Rightarrow \sin\alpha = \frac{b}{\|\mathbf{u}'\|\,\|\mathbf{u}_z\|} = \frac{b}{\sqrt{b^2+c^2}}$$

3D Rotation About Arbitrary Axis

- Rotate **u'** about the x-axis onto the *z*-axis
  - Since we know both cos α and sin α, the rotation matrix can be obtained

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \dfrac{c}{\sqrt{b^2+c^2}} & \dfrac{-b}{\sqrt{b^2+c^2}} & 0 \\ 0 & \dfrac{b}{\sqrt{b^2+c^2}} & \dfrac{c}{\sqrt{b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - Or, we can compute the signed angle α

$$\mathrm{atan2}\left(\frac{c}{\sqrt{b^2+c^2}}, \frac{b}{\sqrt{b^2+c^2}}\right)$$

  - Do not use acos() since its domain is limited to [-1,1]

Euler angles

- Arbitrary rotation can be represented by three rotation along x,y,z axis

$$R_{XYZ}(\gamma,\beta,\alpha) = R_z(\alpha)R_y(\beta)R_x(\gamma)$$

$$= \begin{bmatrix} C\alpha C\beta & C\alpha S\beta S\gamma - S\alpha C\gamma & C\alpha S\beta C\gamma + S\alpha S\gamma & 0 \\ S\alpha C\beta & S\alpha S\beta S\gamma + C\alpha C\gamma & S\alpha S\beta C\gamma - C\alpha S\gamma & 0 \\ -S\beta & C\beta S\gamma & C\beta C\gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Gimble

- Hardware implementation of Euler angles
- Aircraft, Camera

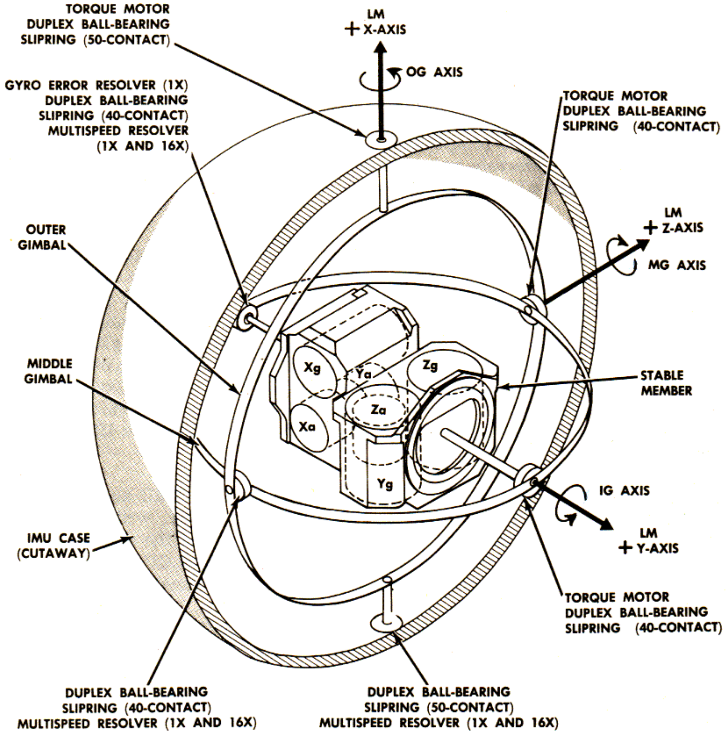
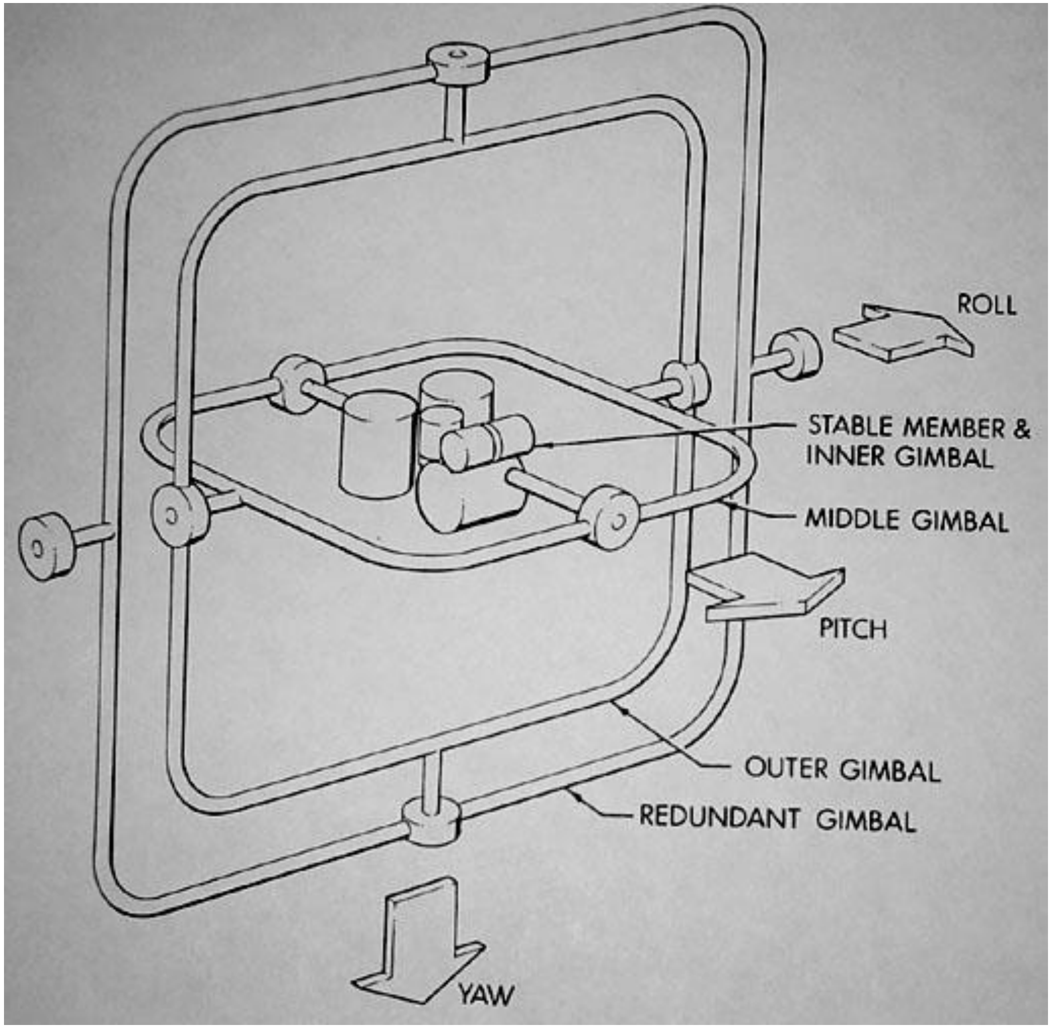


Figure 2.1-24. IMU Gimbal Assembly

Euler Angles

- Rotation about three orthogonal axes
  - 12 combinations
    - XYZ, XYX, XZY, XZX
    - YZX, YZY, YXZ, YXY
    - ZXY, ZXZ, ZYX, ZYZ
- ***Gimble lock***
  - Coincidence of inner most and outmost gimbles' rotation axes
  - Loss of degree of freedom



Euler Angles

- Euler angles are ambiguous
  - Two different Euler angles can represent the same orientation

$$R_1 = (r_x, r_y, r_z) = (\theta, \frac{\pi}{2}, 0) \quad \text{and} \quad R_2 = (0, \frac{\pi}{2}, -\theta)$$

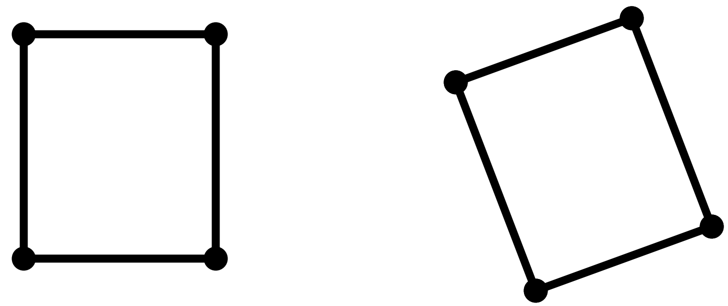
  - This ambiguity brings unexpected results of animation where frames are generated by interpolation.

Taxonomy of Transformations

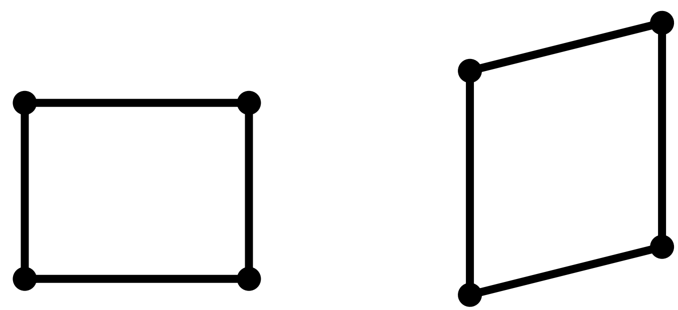
- **Linear** transformations
  - 3x3 matrix
  - Rotation + scaling + shear
- **Rigid** transformations
  - SO(3) for rotation
  - 3D vector for translation
- **Affine** transformation
  - 3x3 matrix + 3D vector or 4x4 homogenous matrix
  - Linear transformation + translation
- **Projective** transformation
  - 4x4 matrix
  - Affine transformation + perspective projection

Taxonomy of Transformations

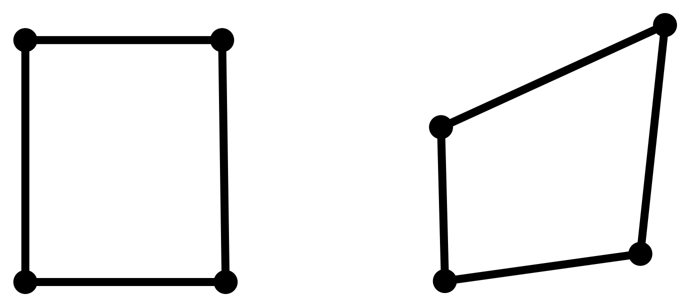
Rigid



Affine



Projective



Composite Transformations

- Composite 2D Translation

$$T = \mathbf{T}(t_{x1}, t_{y1}) \cdot \mathbf{T}(t_{x2}, t_{y2})$$
$$= \mathbf{T}(t_{x1} + t_{x2}, t_{y1} + t_{y2})$$

$$\begin{pmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{pmatrix}$$

Composite Transformations

- Composite 2D Scaling

$$T = \mathbf{S}(s_{x1}, s_{y1}) \cdot \mathbf{S}(s_{x2}, s_{y2})$$
$$= \mathbf{S}(s_{x1}s_{x2}, s_{y1}s_{y2})$$

$$\begin{pmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Composite Transformations

- Composite 2D Rotation

$$T = \mathbf{R}(\theta_2) \cdot \mathbf{R}(\theta_1)$$
$$= \mathbf{R}(\theta_2 + \theta_1)$$

$$\begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta_2+\theta_1) & -\sin(\theta_2+\theta_1) & 0 \\ \sin(\theta_2+\theta_1) & \cos(\theta_2+\theta_1) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$