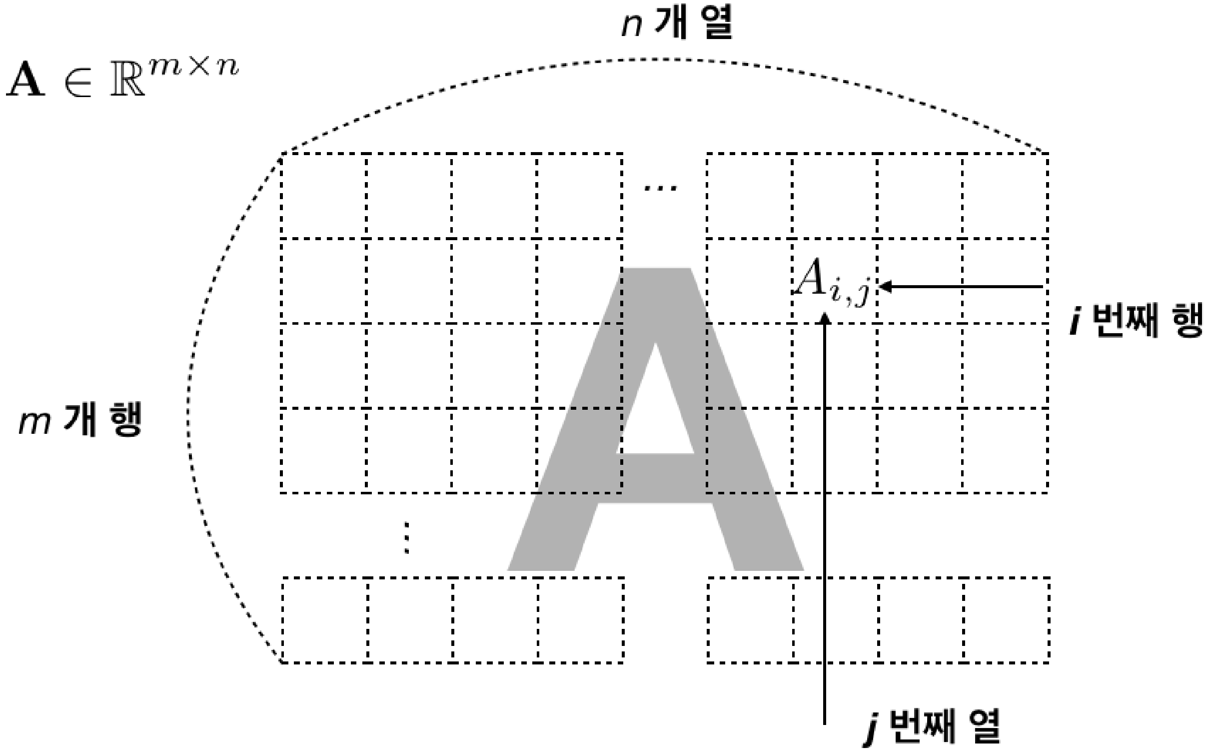
게임 수학 – 강의 6
행렬의 기본 연산 이해

동명대학교 게임공학과

강영민

행렬 데이터의 이해

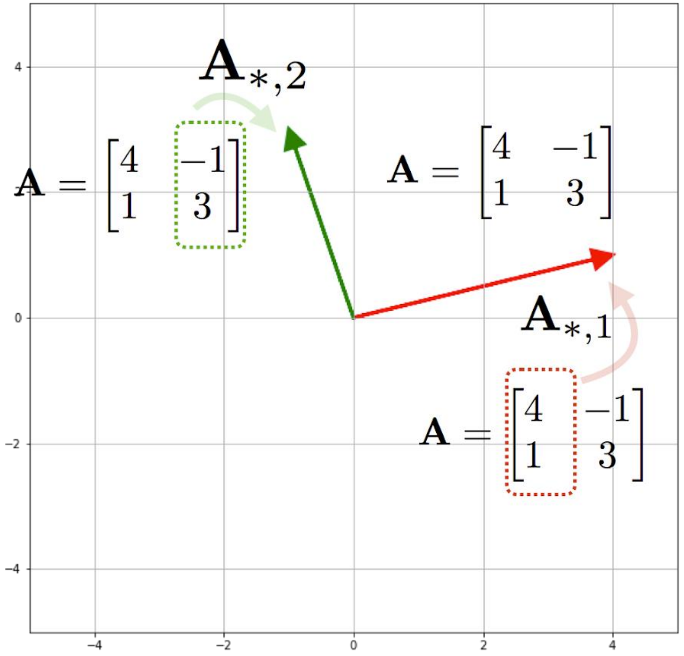
- 행렬은 2차원으로 배열된 숫자



$$\mathbf{A} \in \mathbb{R}^{m \times n}$$

행렬의 가시화

- 가시화하기 쉬운 행렬
  - 2차원 공간에 그려질 수 있는 행렬
    - 2x2 행렬
    - 2개의 2차원 벡터가 존재
- 행렬의 가시화
  - 2차원 벡터들을 2차원 공간에 그림



행렬은
여러 벡터가 모여 있는 것으로
이해할 수 있다.

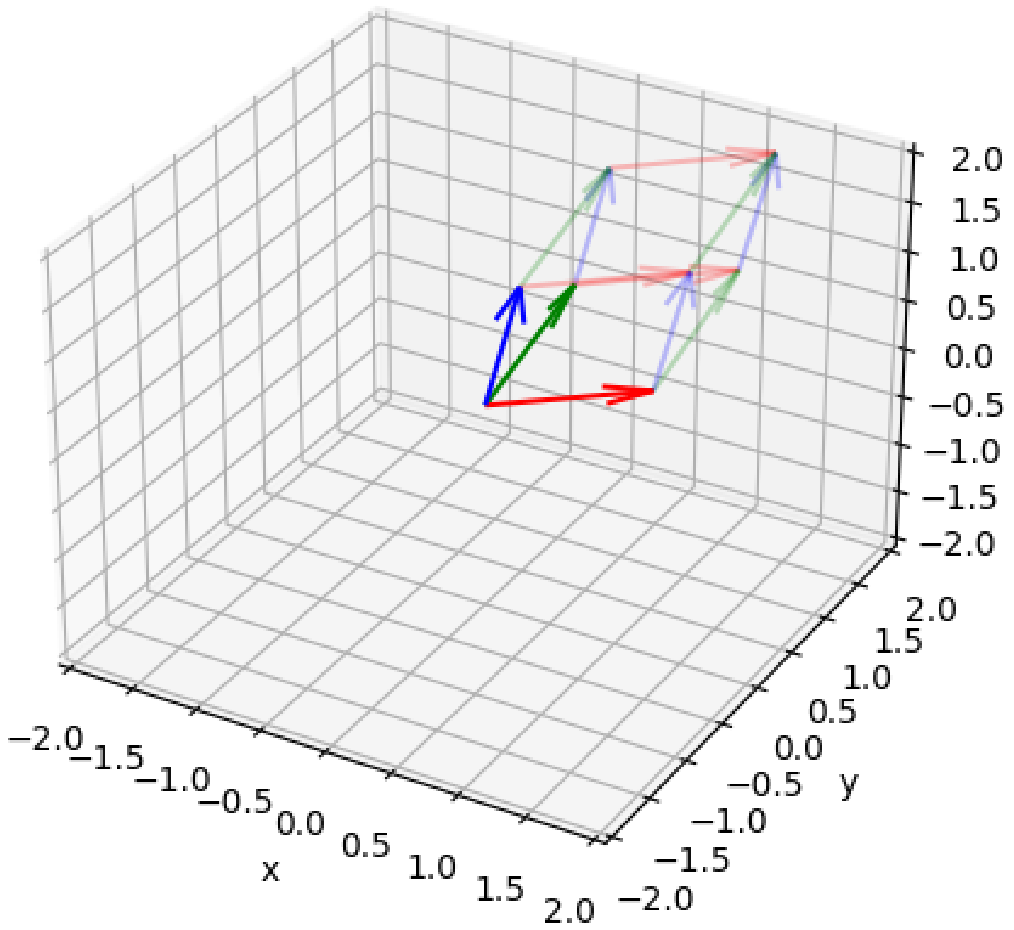
행렬의 모습을 가시화하는 것은
이들 벡터를 각각 그리면 된다.

왼쪽의 두 화살표가 바로
우리가 처음으로
행렬의 모양을 눈으로
확인할 수 있는 이미지이다.

이런 행렬이 무슨 놀라운 일을 하는지는 나중에 알아보자

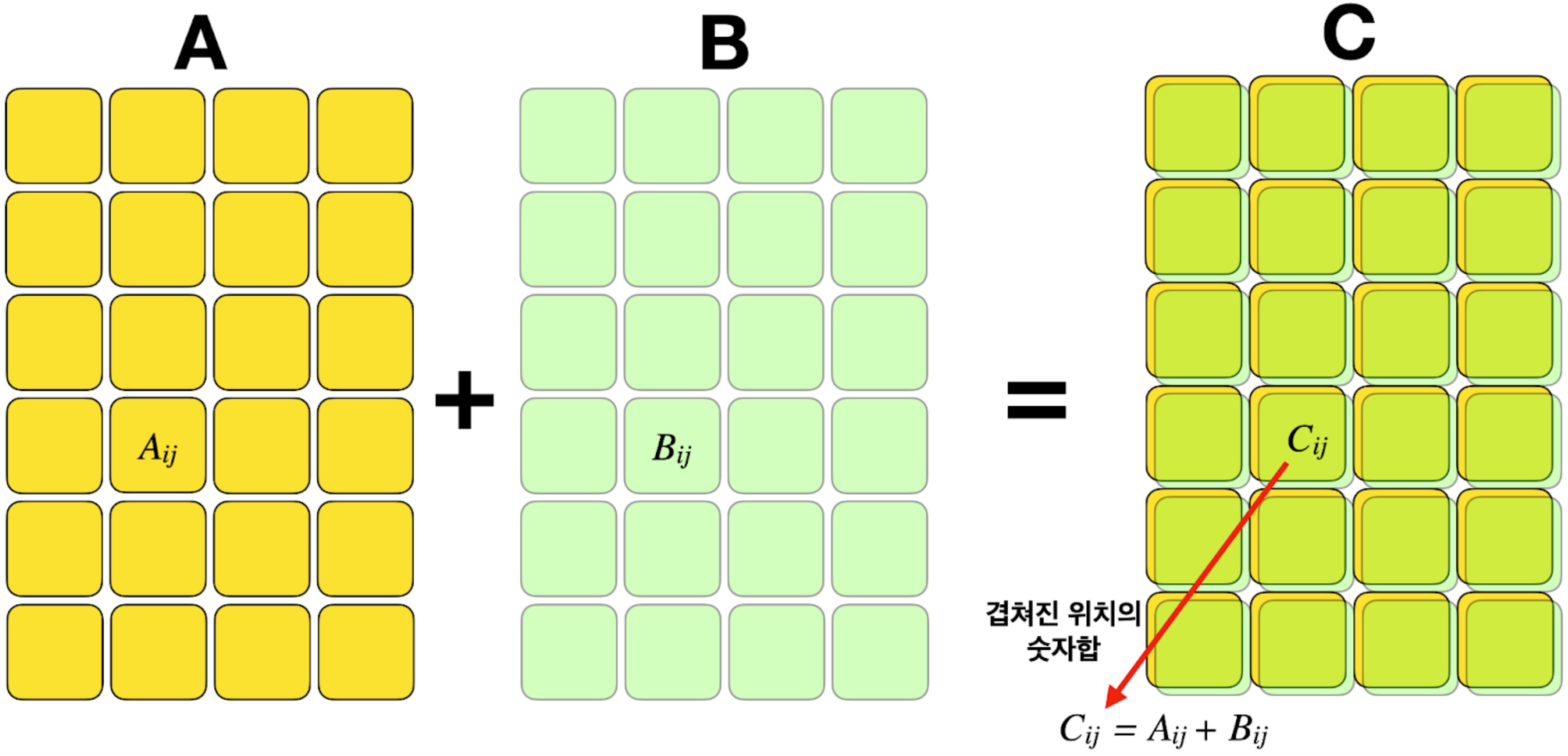
행렬의 가시화

```
mat = np.array([
                    [1, -0.1, 0.2],
                    [0.5,  1.4, 0.11],
                    [0.2,  0.3, 1.2],
               ])

visMat33(mat, x=[-2,2], y=[-2,2], z=[-2,2])
```



행렬의 덧셈과 뺄셈

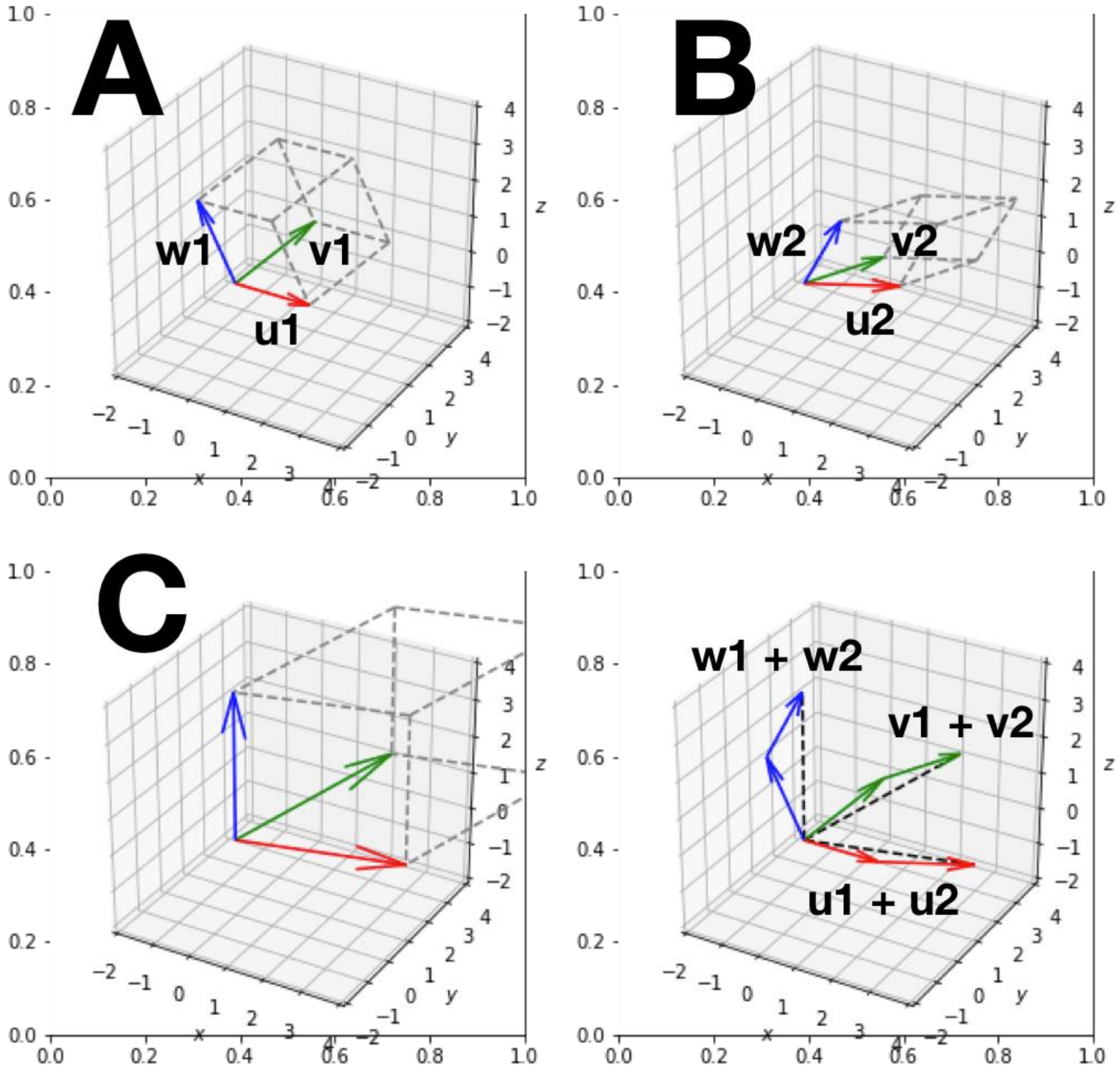
- 기하 객체의 충돌 문제 등에 활용
    - 충돌의 감지: 두 객체 상호간의 거리 문제



행렬의 덧셈과 뺄셈

- 행렬 덧셈의 이해

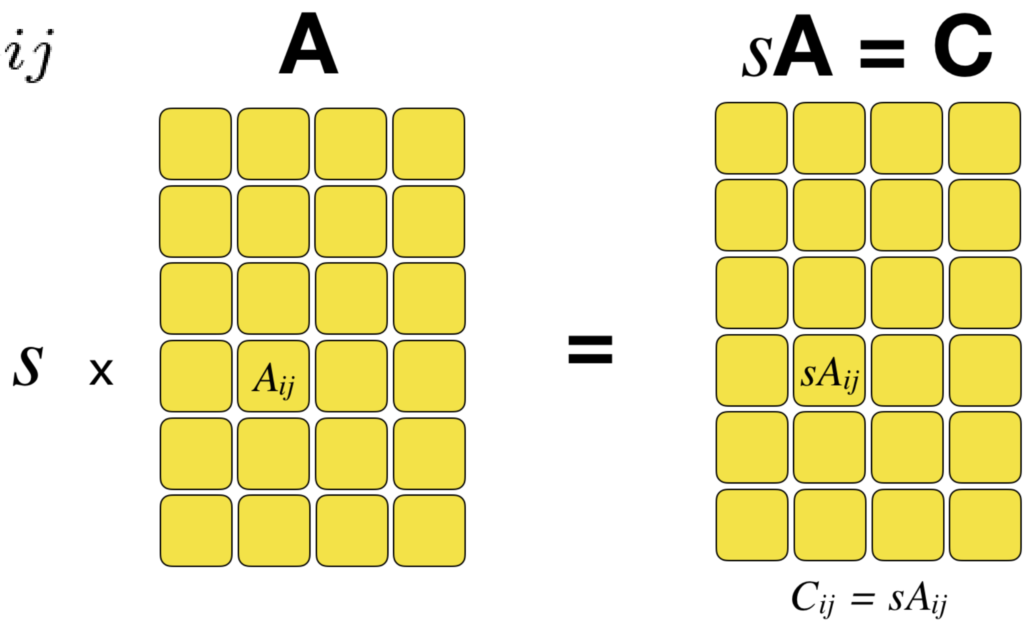
C = A + B



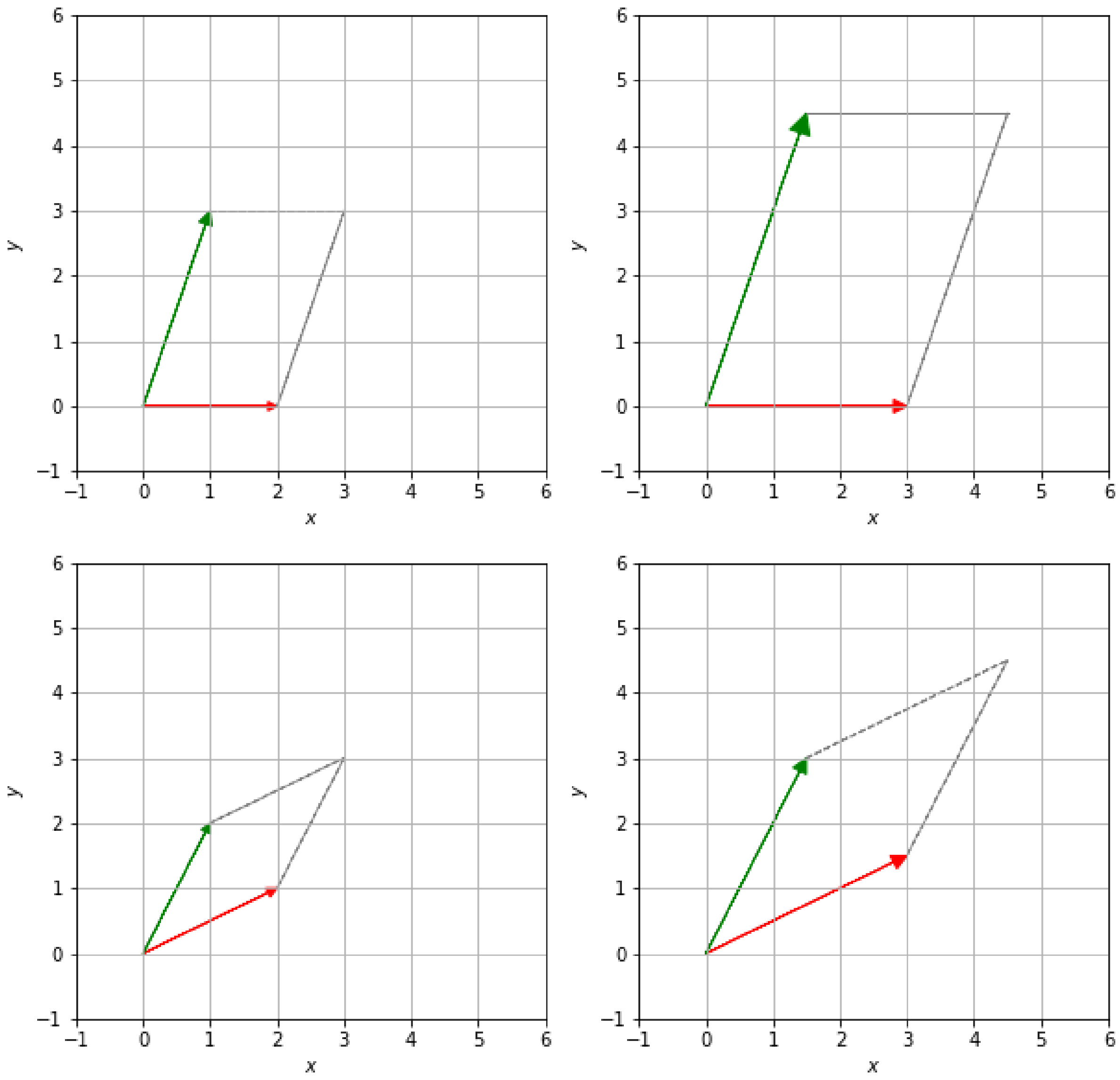
행렬의 스칼라 곱

- 각 성분을 스칼라倍 한다

$$s\mathbf{A} = \mathbf{C} \Rightarrow C_{ij} = sA_{ij}$$



스칼라 곱의 가시화



행렬의 곱

- 곱이 가능한 조건이 존재

$$\mathbf{A} \in \mathbb{R}^{l \times m}$$
$$\mathbf{B} \in \mathbb{R}^{m \times n}$$

이때 두 행렬의 곱 $\mathbf{C}$는 $\mathbb{R}^{l \times n}$ 집합에 속하며 각 원소는 다음과 같이 계산한다.

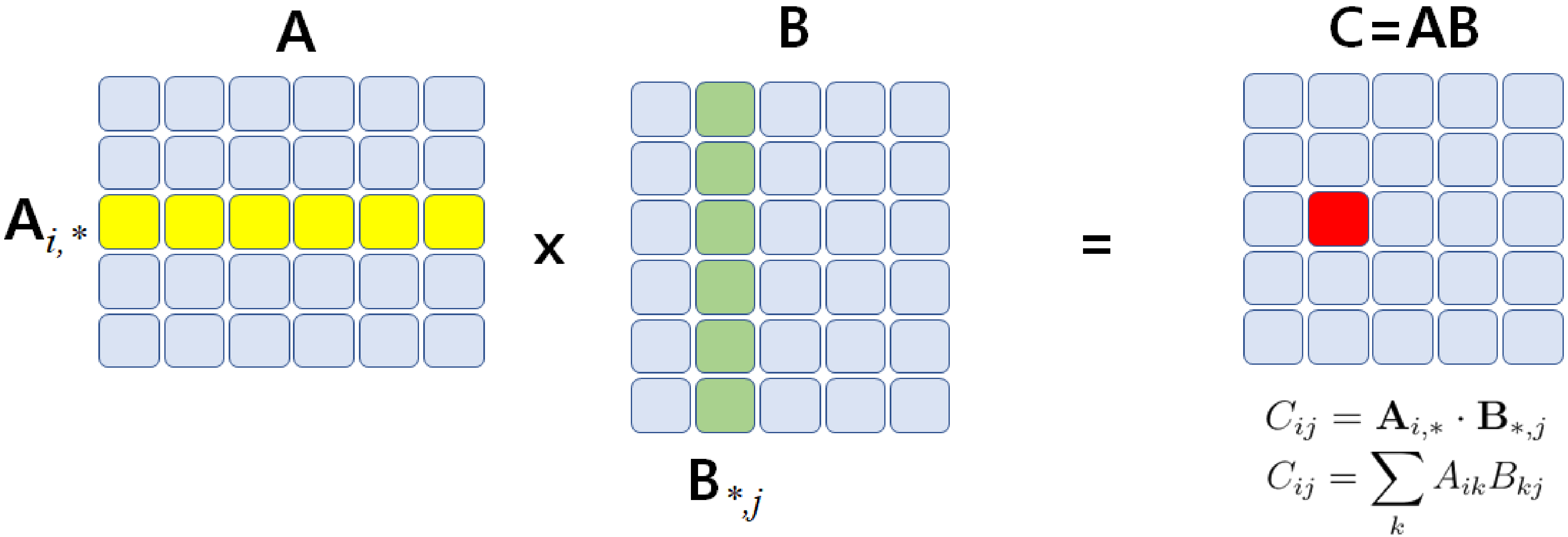
$$\mathbf{C} = \mathbf{AB}$$

$$C_{i,j} = \sum_{k=1}^{m} A_{i,k} B_{k,j} = A_{i,1}B_{1,j} + A_{i,2}B_{2,j} + A_{i,3}B_{3,j} + \cdots + A_{i,m}B_{m,j}$$

행렬 곱의 시각적 표현

- 행렬곱은 두 행렬의 행벡터와 열벡터의 내적으로 구성

$$C_{i,j} = \mathbf{A}_{i,*}\mathbf{B}_{*,j} = \mathbf{A}_{i,*}^{\mathrm{T}} \cdot \mathbf{B}_{*,j}$$



$$C_{ij} = \mathbf{A}_{i,*} \cdot \mathbf{B}_{*,j}$$

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

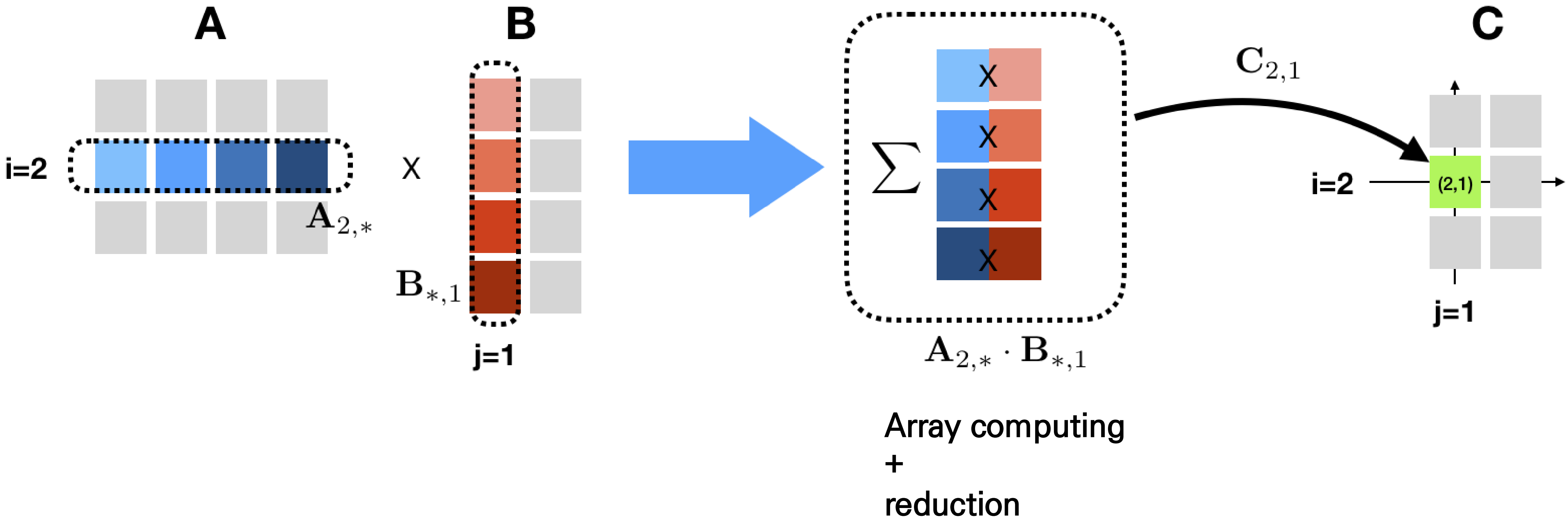
행렬 곱의 계산 – 비효율적인 버전

- 성분별로 계산

```
def mult(A, B): # 아주 단순한 - 그러나 비효율적인 행렬 곱하기
    C = np.zeros((A.shape[0], B.shape[1]), dtype=float)
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            for k in range(A.shape[1]):
                C[i, j] += A[i, k] * B[k, j]
    return C
```

효율적인 행렬곱 – 배열 컴퓨팅 사용

- Numpy는 효율적인 배열 컴퓨팅 제공



주의: *는 아다마르 곱

- A, B가 넘파이 배열로 표현된 행렬일 때,
  - A * B 는 행렬의 곱을 계산하지 못 함

```
A = np.array([[2, 1],
              [0, 3],], dtype=float)
B = np.array([[-2, 1],
              [1, 2],], dtype=float)
C = mult(A, B)
print(C)
print(A*B)
```

```
[[-3.  4.]
 [ 3.  6.]]

[[-4.  1.]
 [ 0.  6.]]
```

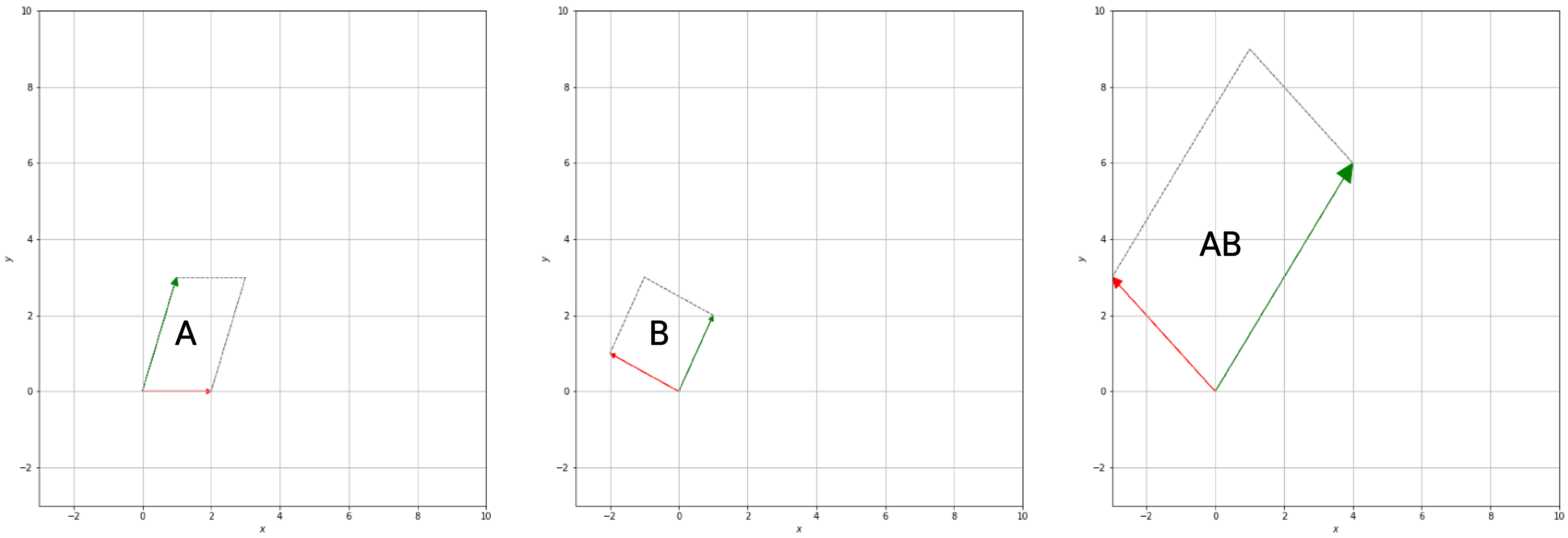
주의: *는 아다마르 곱

- 제대로 된 행렬 곱 계산

```
print(A.dot(B))
print(A @ B)
```

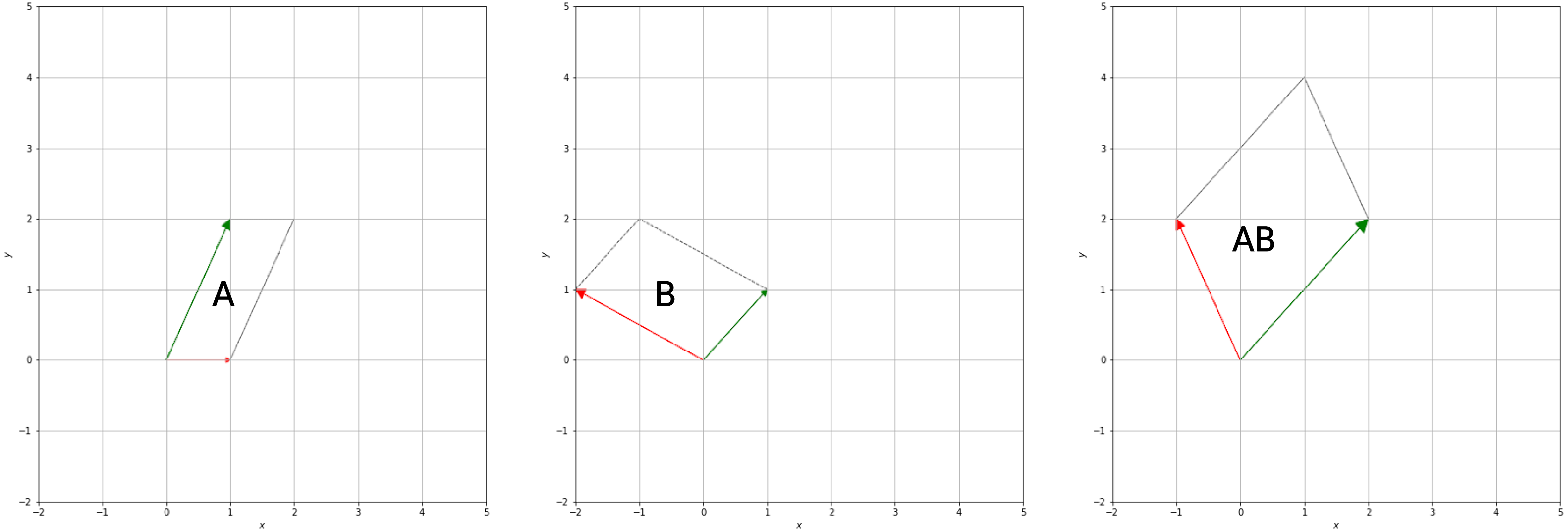
```
[[-3.  4.]
 [ 3.  6.]]

[[-3.  4.]
 [ 3.  6.]]
```

행렬곱의 시각적 확인



무슨 의미인지 파악하기 힘듦

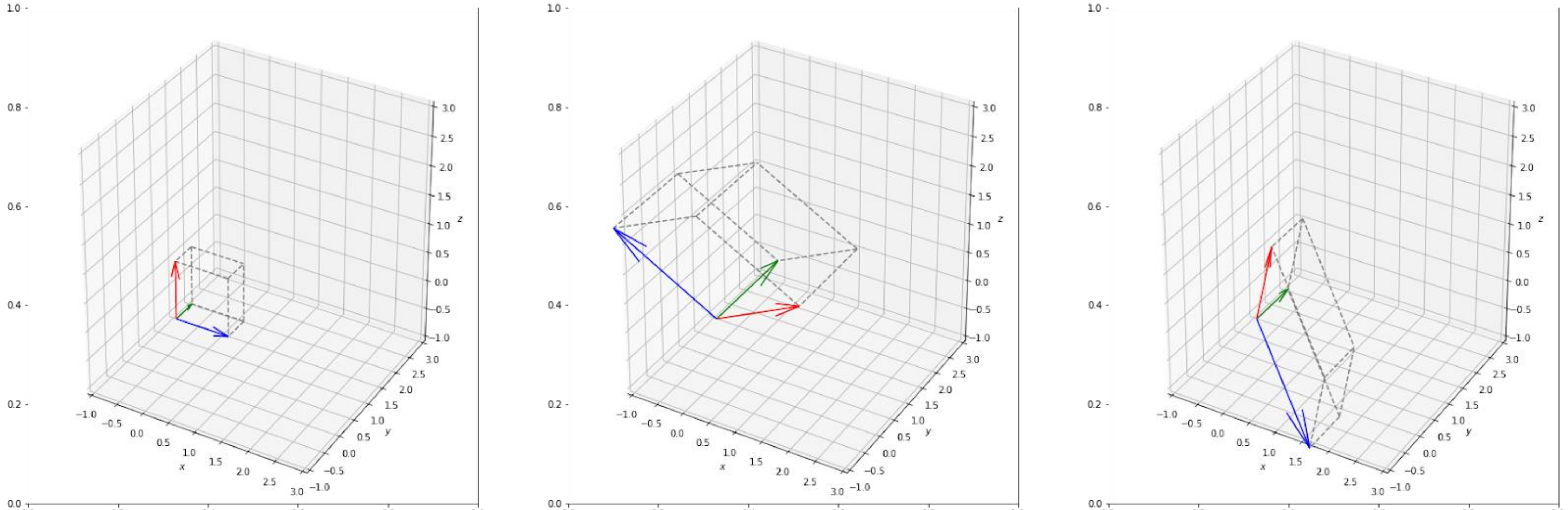
행렬곱의 시각적 확인 2 – 면적을 보자



면적을 살펴 보자: |A|=2 |B| = 3 |AB| = 6   ➔ 2*3 = 6

행렬 곱의 관찰 – 3X3 행렬의 부피 변화

- 부피 곱하기



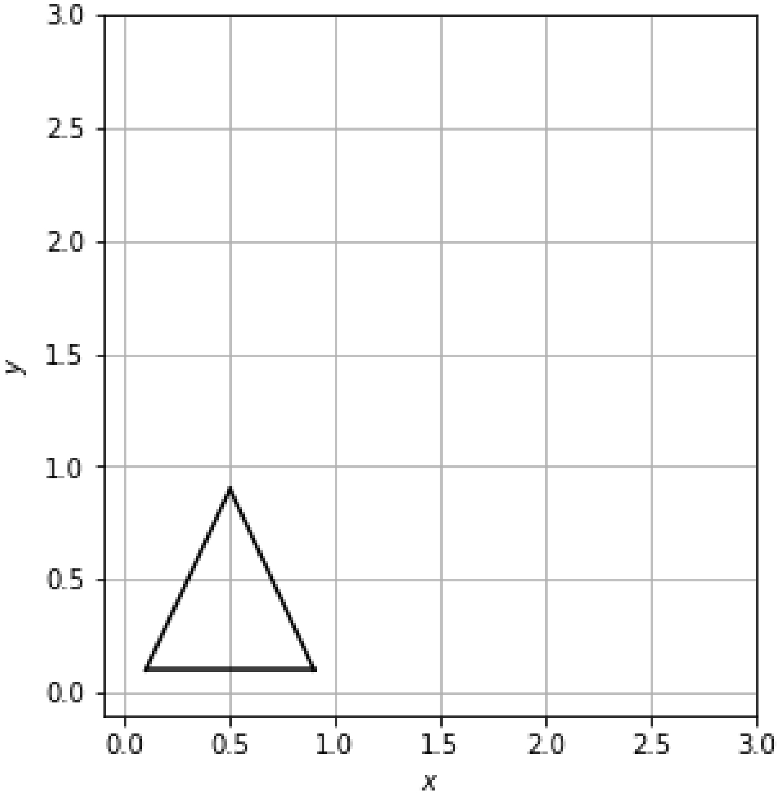
부피를 살펴 보자: |A|=0.5 ➔ |AB| 는 |B| 부피의 반 정도로 줄어든 것처럼 보임

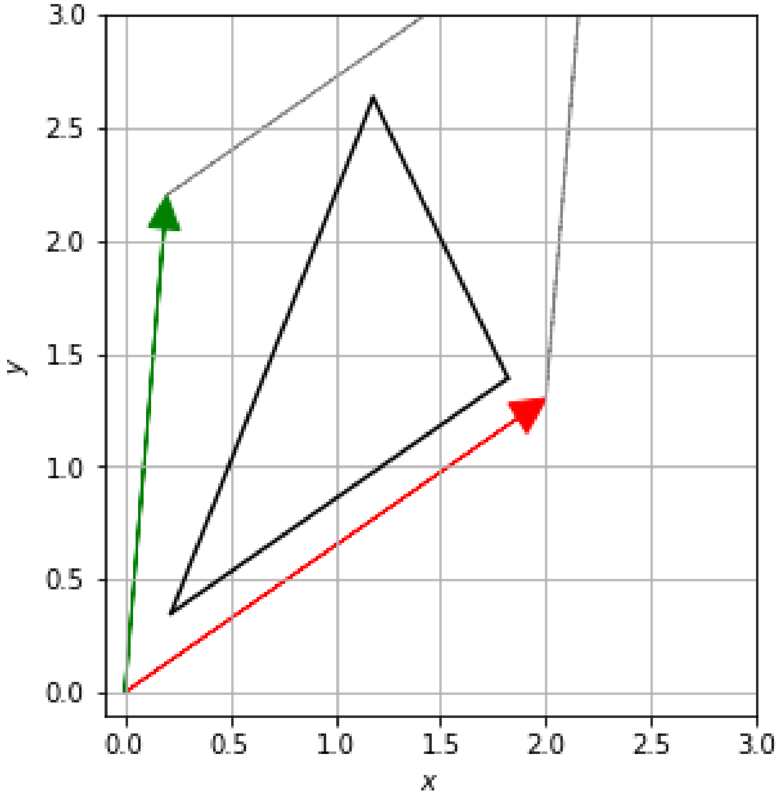
행렬의 크기: 면적, 부피

- 어떤 행렬 A의 부피나 면적 |A|
  - 행렬의 크기
- 정사각 행렬에서만 정의됨
  - 이 값을 행렬식이라고 함
    - det A 혹은 |A|로 표현

행렬의 크기는 어떤 일을 하나

- 행렬은 벡터에 곱해져 새로운 벡터를 생성하는 “변환”의 도구
  - 행렬식을 통해 변환의 결과를 예측할 수 있음



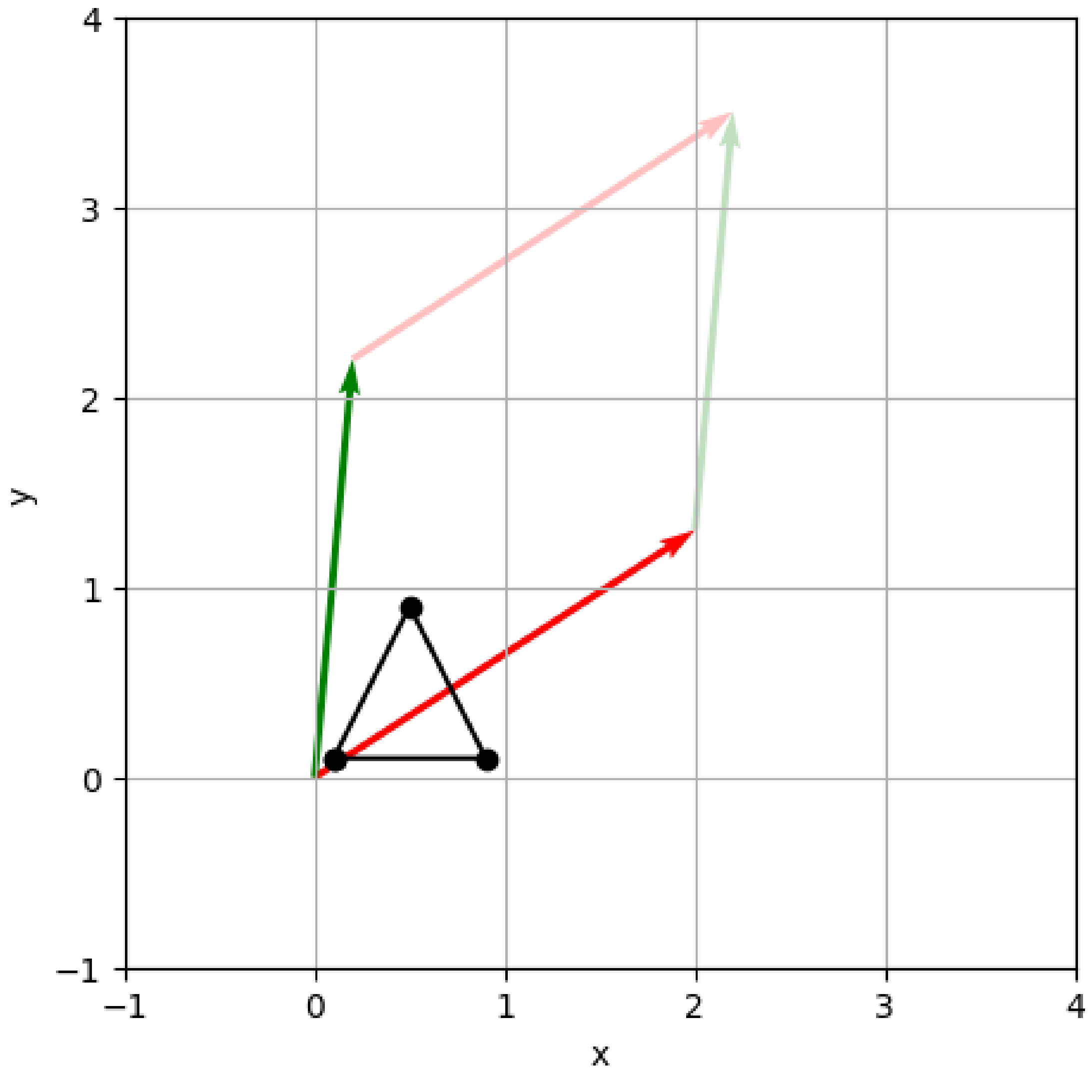


행렬의 크기는 어떤 일을 하나

```
M = np.array([[2.0, 0.2],                   # 행렬 M을 정의한다.
              [1.3, 2.2]])
points = np.array([[0.1, 0.9, 0.5],         # 3개의 좌표로 행렬 구성
                   [0.1, 0.1, 0.9]])
p1 = points[:, 0]                           # 각각의 좌표 p1
p2 = points[:, 1]                           # 각각의 좌표 p2
p3 = points[:, 2]                           # 각각의 좌표 p3
```

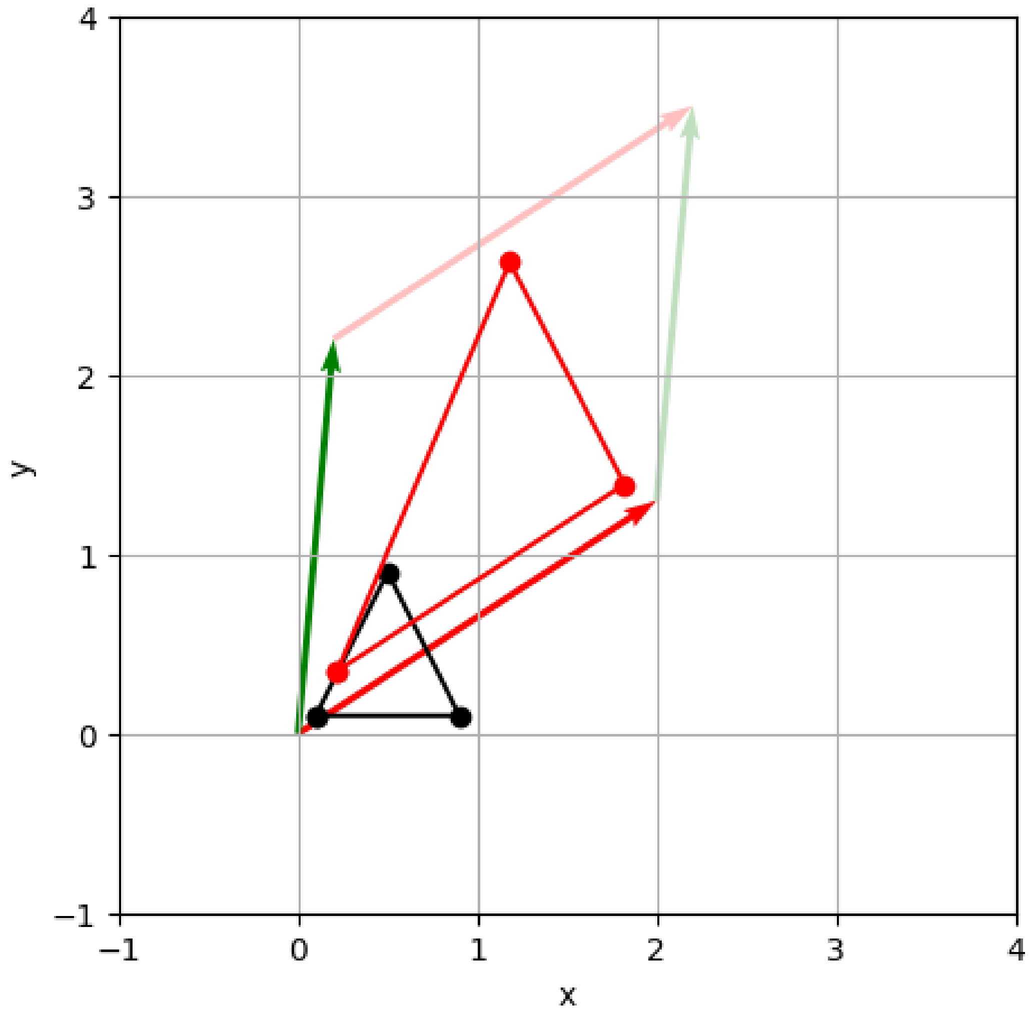
점들과 행렬 그려보기

```
ax = visMat22(M, x=[-1,4], y=[-1, 4])

 # 3개의 열벡터가 표현하는 3개의 좌표
points = np.array([[0.1, 0.9, 0.5],
                   [0.1, 0.1, 0.9]])

drawLineLoop(ax, points)
```



점들과 행렬을 곱해서 가시화

```
ax = visMat22(M, x=[-1,4], y=[-1, 4])

 # 3개의 열벡터가 표현하는 3개의 좌표
points = np.array([[0.1, 0.9, 0.5],
                   [0.1, 0.1, 0.9]])
newPoints = M.dot(points)
drawLineLoop(ax, points)
drawLineLoop(ax, newPoints, color='red')
```



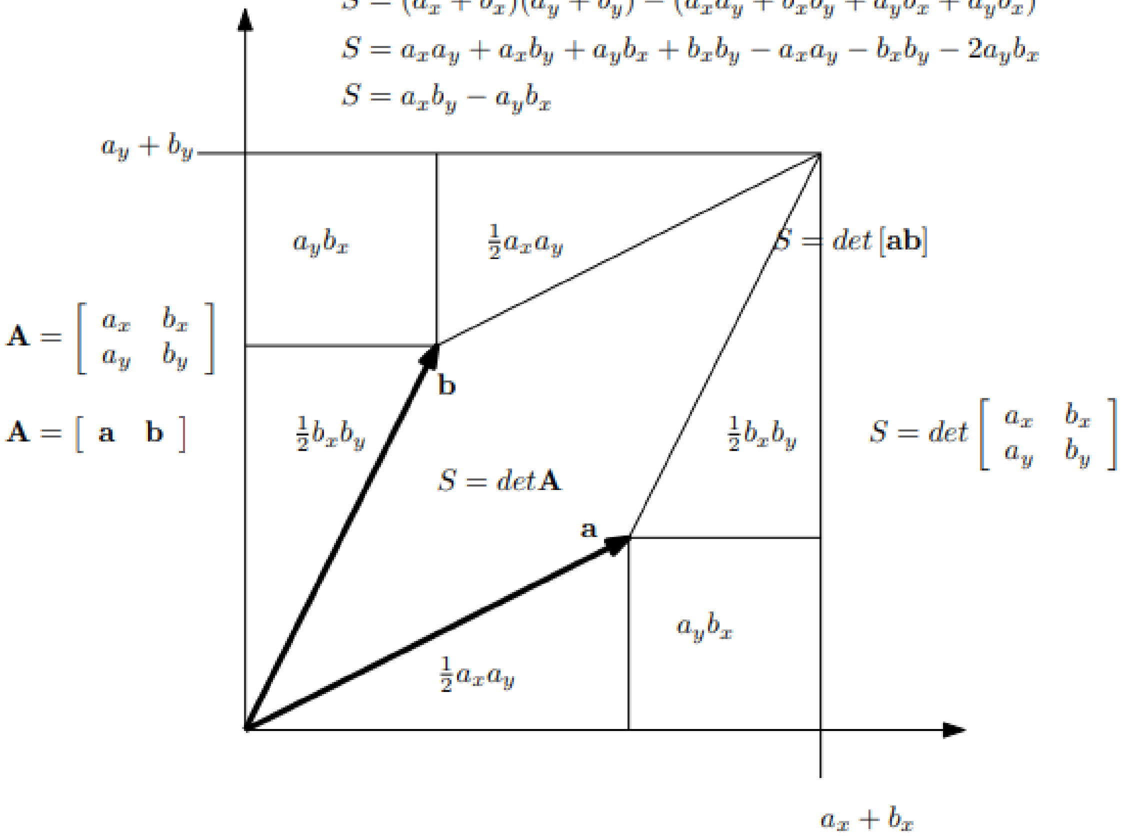
행렬식의 기하적 의미

두 열 벡터 $\mathbf{a} = (a_x a_y)^{\mathrm{T}}$와 $\mathbf{b} = (b_x, b_y)^{\mathrm{T}}$를 열로 하는 행렬 $\mathbf{A}$

$$\mathbf{A} = \begin{bmatrix} a_x & b_x \\ a_y & b_y \end{bmatrix}$$

이 두 벡터를 두 개의 변으로 하는 평행사변형의 넓이가 행렬 $\mathbf{A}$의 행렬식

$$S = (a_x + b_x)(a_y + b_y) - (a_x a_y + b_x b_y + a_y b_x + a_y b_x)$$
$$S = a_x a_y + a_x b_y + a_y b_x + b_x b_y - a_x a_y - b_x b_y - 2a_y b_x$$
$$S = a_x b_y - a_y b_x$$

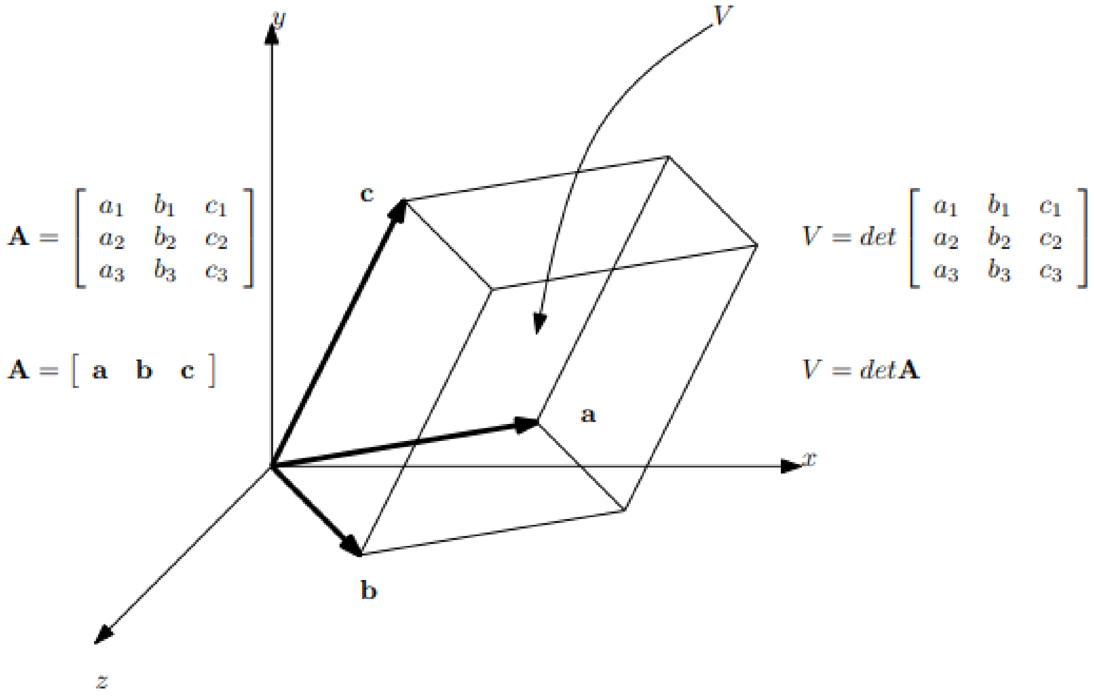


3x3 행렬의 기하적 이해

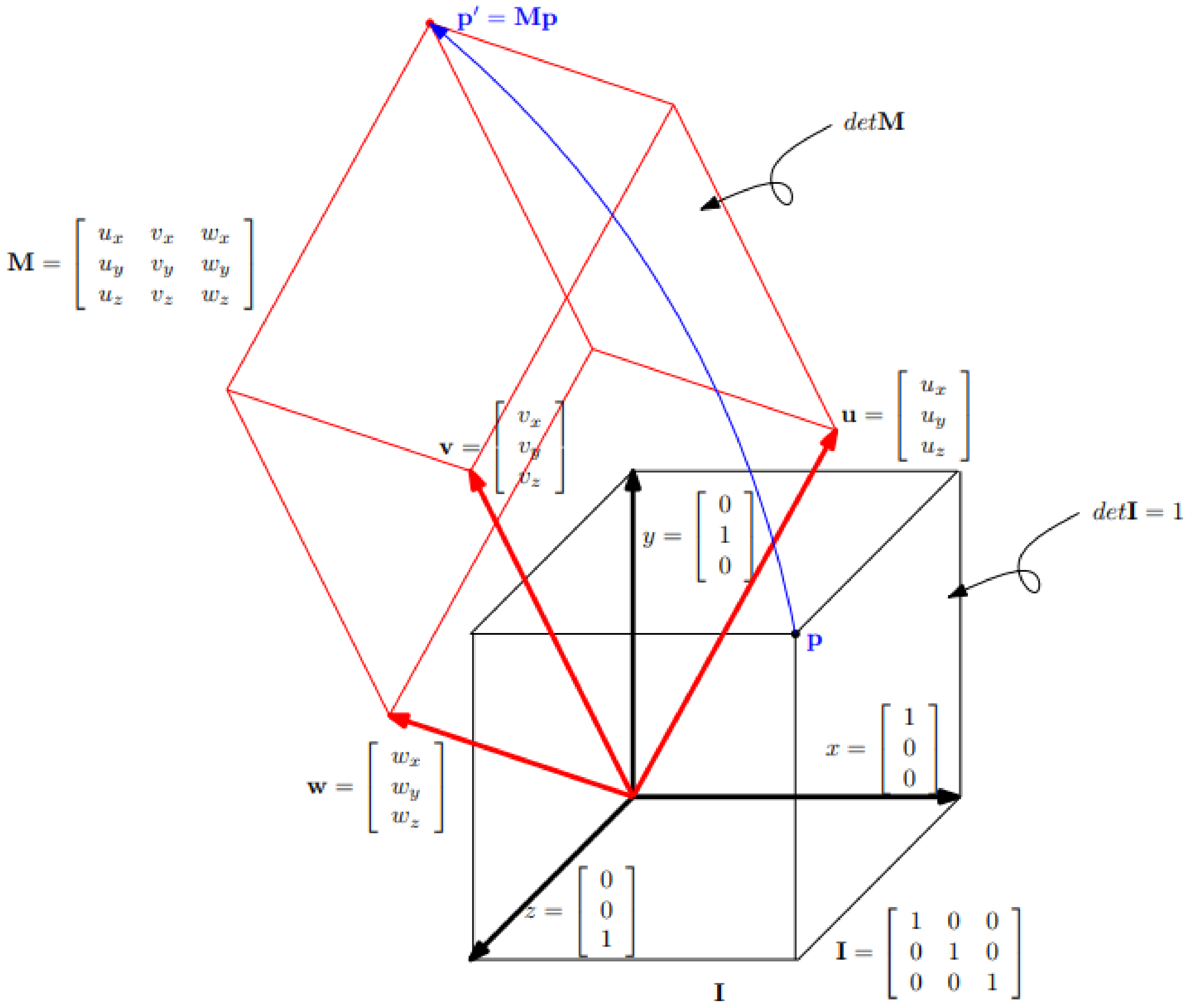
$\mathbf{A} \in \mathbb{R}^{3\times 3}$는 세 개의 벡터 $\mathbf{a}, \mathbf{b}, \mathbf{c}$를 포함

$$\mathbf{A} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} = \begin{bmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{bmatrix}$$

이 세 개의 벡터들이 만드는 평행육면체의 크기가 세 개의 벡터들로 구성된 행렬의 행렬식



변환으로서의 3x3 행렬과 행렬식의 이해



행렬식의 특성

- 몇 가지 기억해 둘 행렬식의 특성

$$|\mathbf{A}| = |\mathbf{A}^{\mathrm{T}}|$$

$$\mathbf{A} \in \mathbb{R}^{n \times n} \Rightarrow |k\mathbf{A}| = k^n|\mathbf{A}|$$

$$|\mathbf{AB}| = |\mathbf{A}||\mathbf{B}|$$

역행렬

- 역행렬은 정방행렬에만 존재
- $\mathbf{A}$의 역행렬이 존재한다면, 이 역행렬을 $\mathbf{A}^{-1}$로 표현
- 역행렬 $\mathbf{A}^{-1}$은 다음과 같은 조건을 만족
  - $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$
  - $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$

- 역행렬이 존재하는 행렬을 가역행렬(invertible matrix)
- 역행렬이 존재하지 않는 행렬은 특이행렬(singular matrix)

- 의사 역행렬(pseudo-inverse)
  - 행렬 $\mathbf{A}$가 정방행렬이 아니고 $\mathbb{R}^{m\times n}$에 속한다고 하자. 다른 어떤 행렬 $\mathbf{B}$가 $\mathbb{R}^{n\times m}$에 속하면, 두 행렬의 곱 $\mathbf{AB}$는 $\mathbb{R}^{m\times m}$에 속하는 정방행렬이 된다. 만약 $\mathbf{AB} = \mathbf{I} \in \mathbb{R}^{n\times n}$이라면, $\mathbf{B}$를 $\mathbf{A}$의 의사 역행렬(pseudo-inverse)라고 한다.

역행렬의 계산

- 역행렬의 계산은 수반행렬(adjoint matrix)를 이용하여 쉽게 정의
  - 행렬 $\mathbf{A}$의 수반행렬: 여인자 C_{ij}를 성분으로 하는 행렬 $\mathbf{C}$의 전치 (transpose)

$$adj\,\mathbf{A} = \begin{pmatrix} C_{11} & C_{21} & \cdots & C_{n1} \\ C_{12} & C_{22} & \cdots & C_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ C_{1n} & C_{2n} & \cdots & C_{nn} \end{pmatrix} = \mathbf{C}^{\mathrm{T}}$$

- 수반행렬을 행렬의 행렬식으로 나누면 역행렬이 된다.

$$\mathbf{A}^{-1} = \frac{adj\,\mathbf{A}}{|\mathbf{A}|} = \frac{1}{|\mathbf{A}|} \begin{pmatrix} C_{11} & C_{21} & \cdots & C_{n1} \\ C_{21} & C_{22} & \cdots & C_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ C_{n1} & C_{2n} & \cdots & C_{nn} \end{pmatrix} = \mathbf{C}^{\mathrm{T}}$$

식은 간단하지만, 여인자를 구하는 재귀호출이 매우 많은 계산을 요구