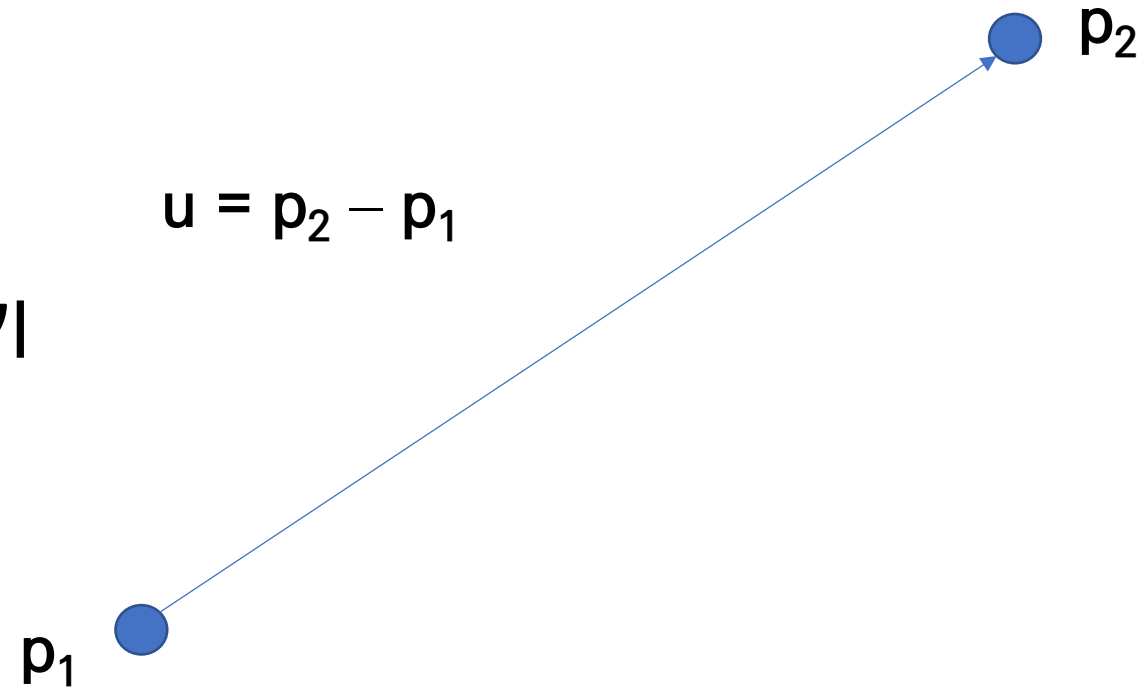
게임 수학 – 강의 5
벡터 연산의 응용

동명대학교 게임공학과
강영민

벡터는 다양한 문제에 활용 가능

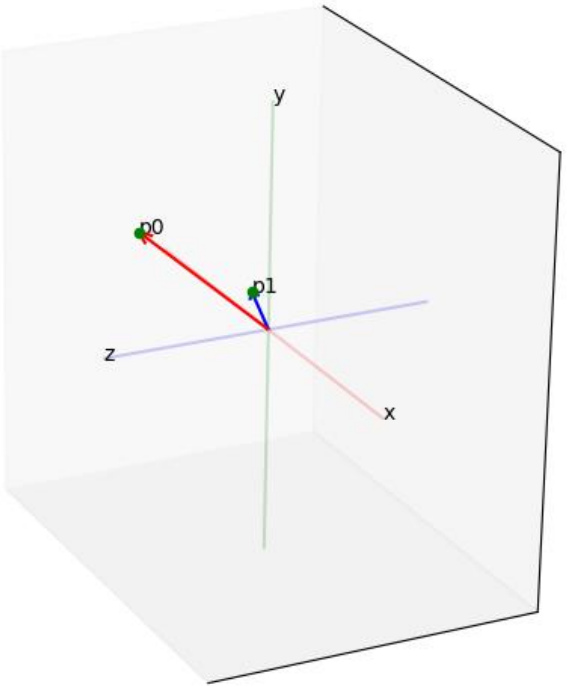
- 기하 객체의 충돌 문제 등에 활용
  - 충돌의 감지: 두 객체 상호간의 거리 문제

두 점 사이의 거리
= 두 점을 잇는 벡터의 크기
= || u ||



두 점 사이의 거리 계산 방법 1/2

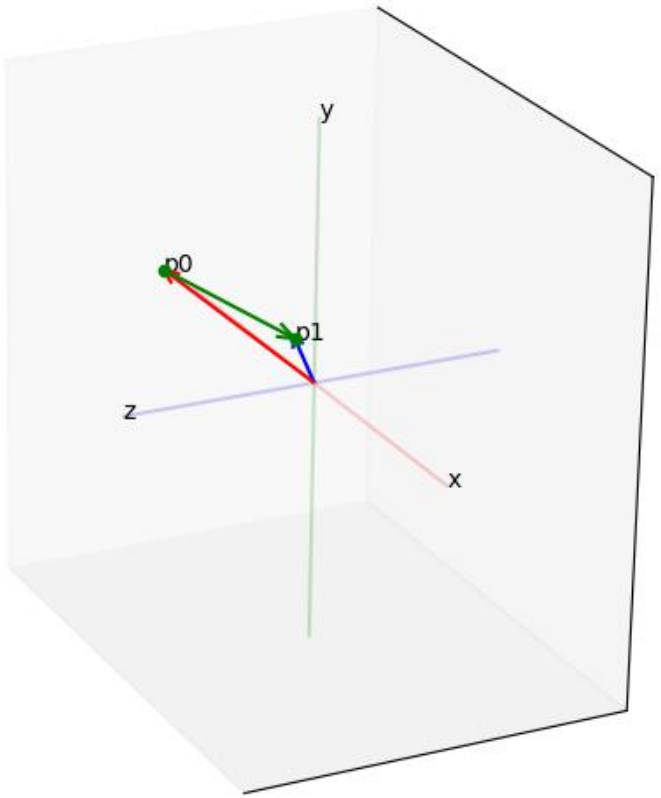
```
p0 = np.array([1, 2, 3])
p1 = np.array([1, 1, 1])

mySpace = axis3d(x=[-3,3], y=[-3, 3], z=[-3, 3])
draw_vec3d(mySpace, p0, color='red')
draw_vec3d(mySpace, p1, color='blue')
draw_points(mySpace, [p0, p1], labels=['p0', 'p1'], color='green')
```



두 점 사이의 거리 계산 방법 2/2

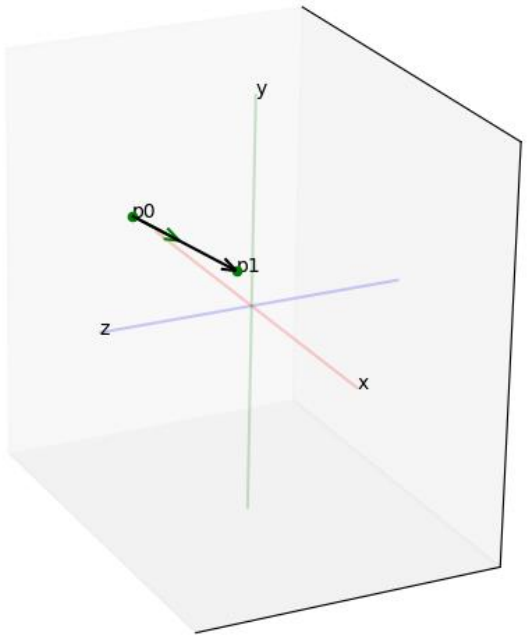
```
p0 = np.array([1, 2, 3])
p1 = np.array([1, 1, 1])

u = p1 - p0

mySpace = axis3d(x=[-3,3], y=[-3, 3], z=[-3, 3])
draw_vec3d(mySpace, p0, color='red')
draw_vec3d(mySpace, p1, color='blue')
draw_points(mySpace, [p0, p1], labels=['p0', 'p1'], color='green')

draw_vec3d(mySpace, u, start_from = p0, color='green')

distance = np.linalg.norm(u)
print(distance)
```

```
2.23606797749979
```



두 점 사이를 잇는 방향 벡터에 거리 곱하기

```
p0 = np.array([1, 2, 3])
p1 = np.array([1, 1, 1])

u = p1 - p0

u_direction = u / np.linalg.norm(u)

mySpace = axis3d(x=[-3,3], y=[-3, 3], z=[-3, 3])
draw_points(mySpace, [p0, p1], labels=['p0', 'p1'], color='green')

draw_vec3d(mySpace, u_direction, start_from = p0, color='green')

distance = np.linalg.norm(u)

long_vec = u_direction * distance

draw_vec3d(mySpace, long_vec, start_from = p0, color='black')
```



두 점 사이의 충돌

- 점이 정확이 0의 거리에 있는 것은 드문 일
  - 실제로는 임계 거리 이내로 근접했는지 검사

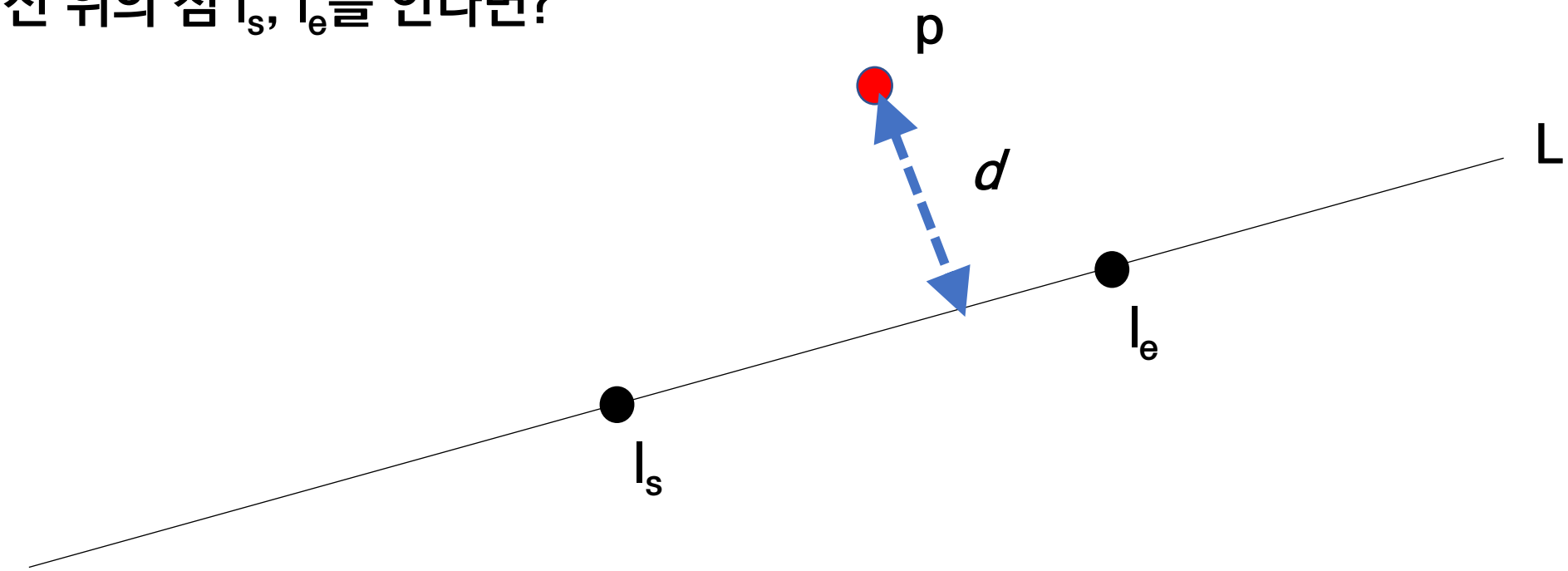
점과 점의 충돌은 두 점이 허용되는 거리 ϵ 이내로 근접했는지를 검사하는 것이다. 충돌 검사의 대상이 되는 두 점을 각각 $\mathbf{P}$과 $\mathbf{Q}$라고 하면, 검사는 다음과 같이 두 점 사이의 거리 $\mathcal{D}(\mathbf{P}, \mathbf{Q})$가 ϵ보다 작은지를 검사하는 것이다.

$$\mathcal{D}(\mathbf{P}, \mathbf{Q})^2 = (\mathbf{P}.x - \mathbf{Q}.x)^2 + (\mathbf{P}.y - \mathbf{Q}.y)^2 + (\mathbf{P}.z - \mathbf{Q}.z)^2 < \epsilon^2$$

이때, $\mathbf{P}_x$, $\mathbf{P}_y$, $\mathbf{P}_z$는 각각 점 P 위치 벡터의 x, y, z 성분이다.

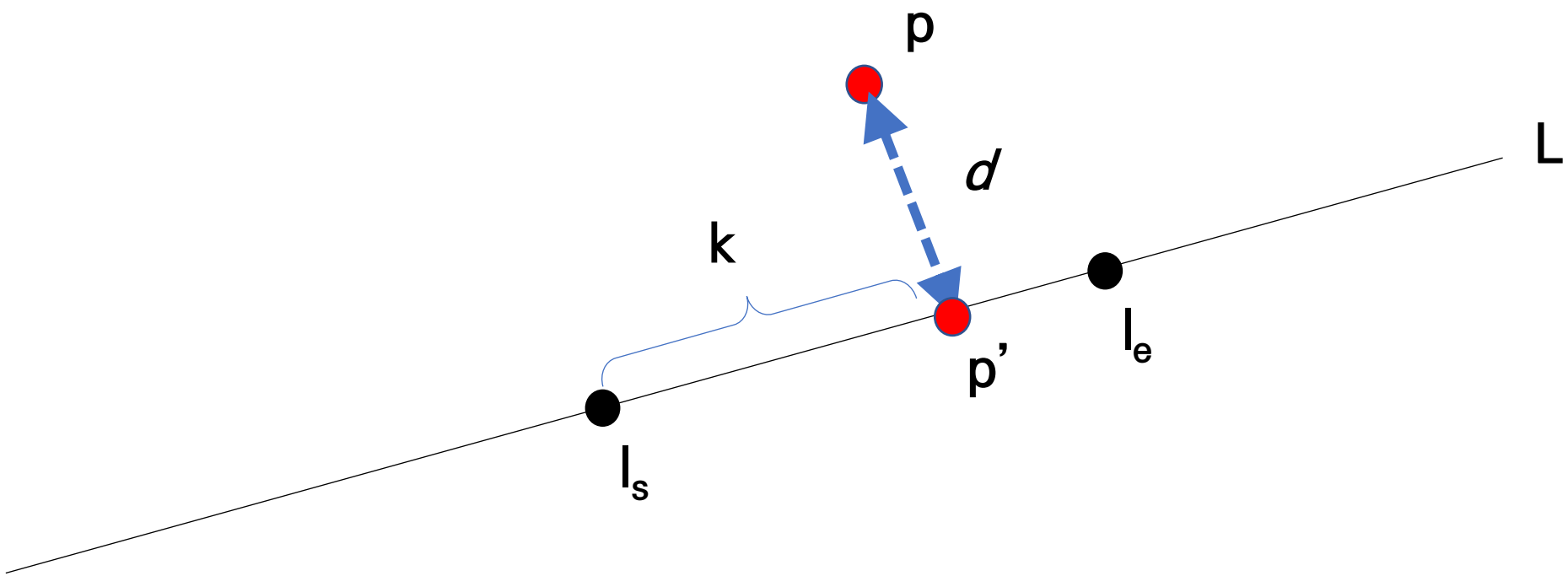
점과 직선의 거리

- 직선 L과 점 p의 거리: *d*
  - 직선 위의 점 l_s, l_e를 안다면?



점과 직선의 거리

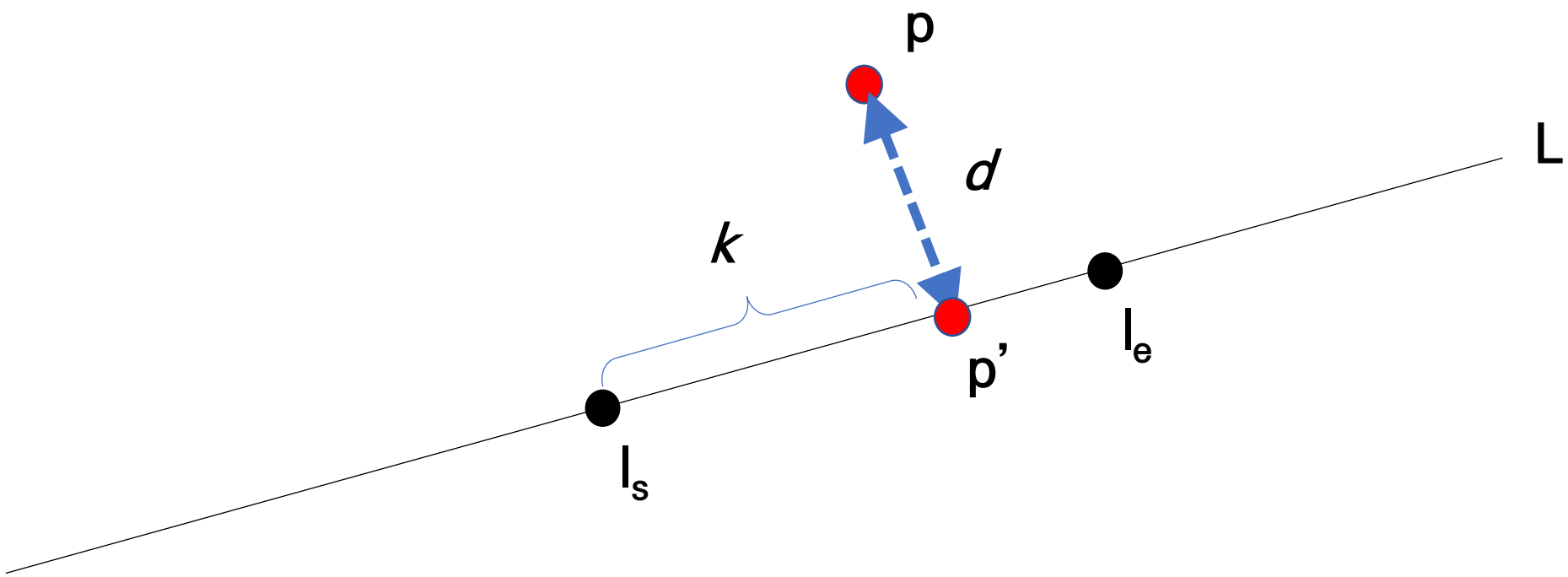
- 점 p에서 직선 L에 내린 수선의 발 p’를 구하자



l_s에서 p’까지의 거리 k는 l_s에서 p로 가는 벡터가 직선 L에 내린 그림자의 길이
→ 내적으로 구할 수 있음

점과 직선의 거리

- 점 p에서 직선 L에 내린 수선의 발 p’를 구하자

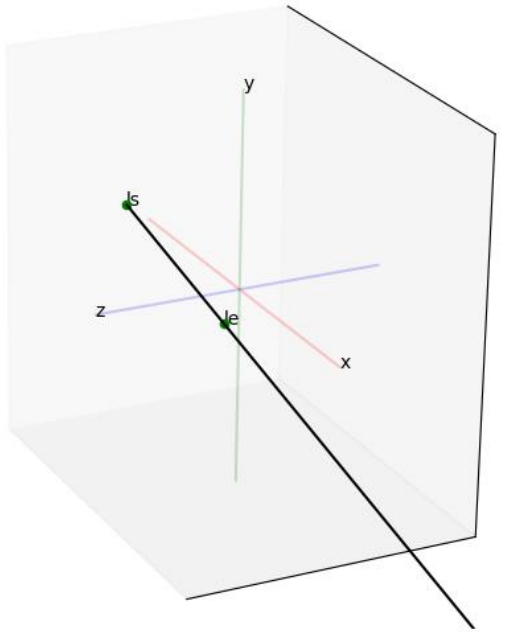


$u = p - l_s$
$v = l_e - l_s$

$\Rightarrow$ $k = u \cdot v / ||v||$ $\Rightarrow$ $p' = l_s + k\, v / ||v||$

구현 – 두 점을 지나는 직선

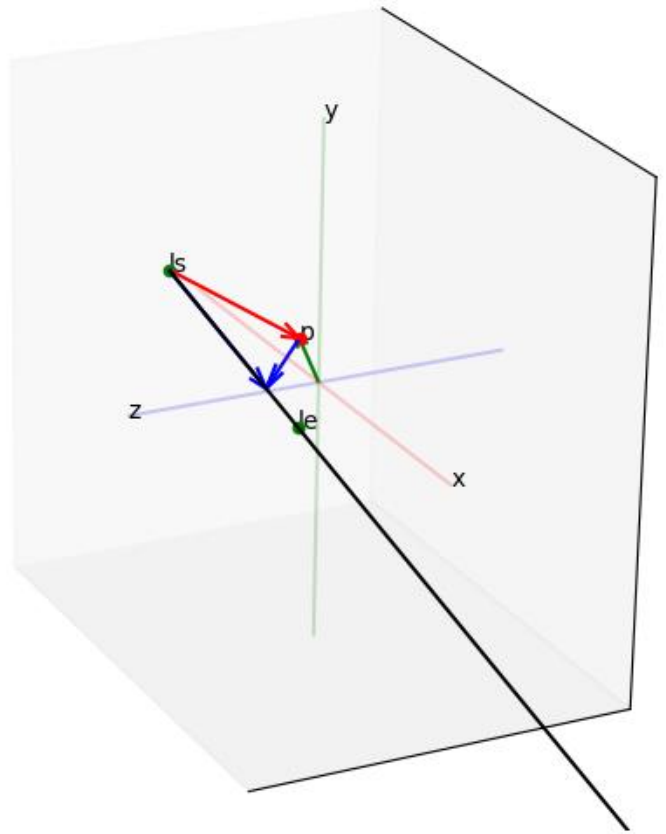
```python
#직선을 생성해 보자

ls = np.array([1, 2, 3])
le = np.array([1, 0, 1])

mySpace = axis3d(x=[-3,3], y=[-3, 3], z=[-3, 3])
draw_points(mySpace, [ls, le], labels=['ls', 'le'], color='green')

line = le - ls
draw_vec3d(mySpace, 10*line, start_from = ls, color='black')
```



구현 – 두 점을 지나는 직선과 어떤 점 p 사이의 거리

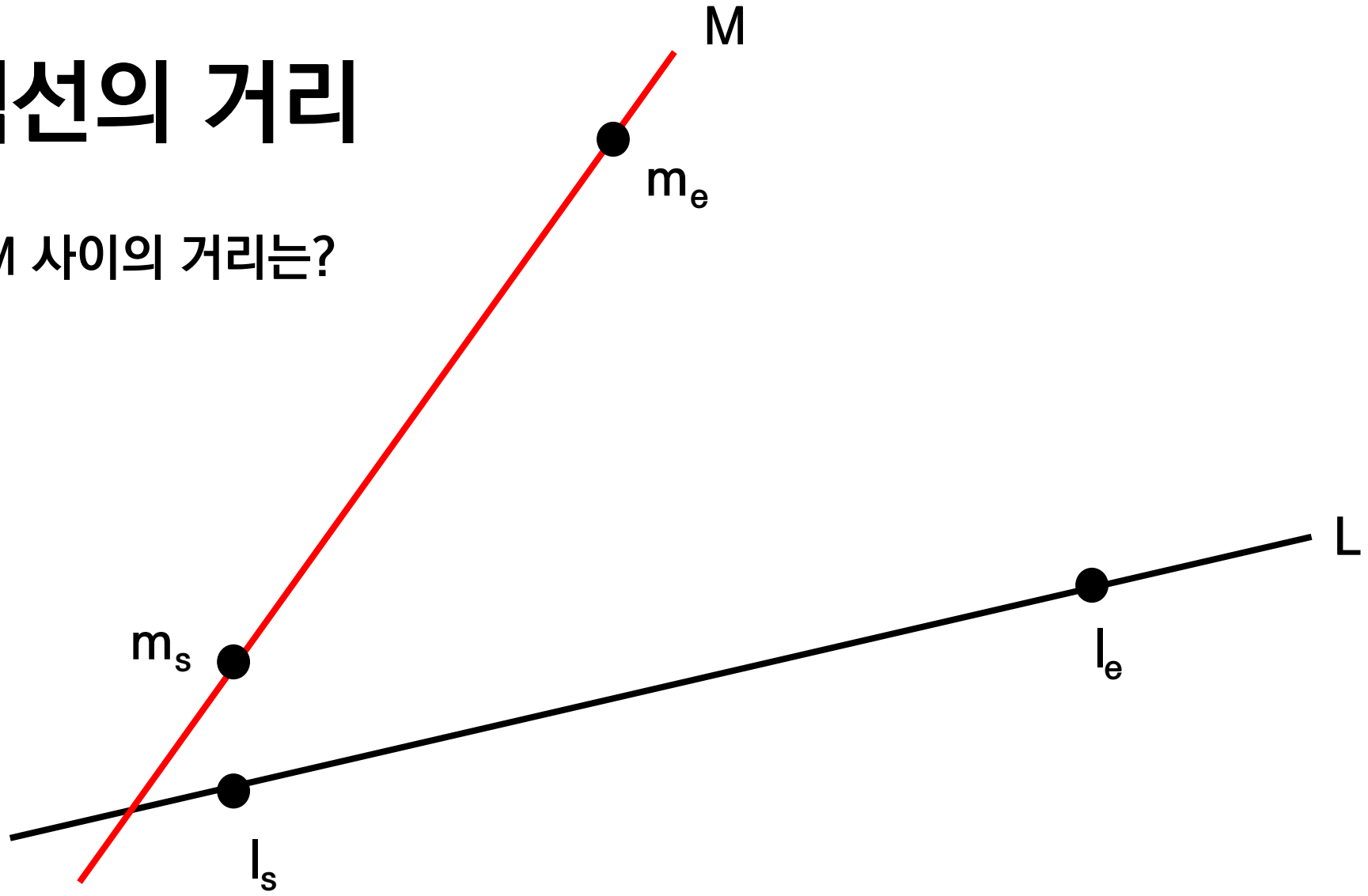
```python
ls = np.array([1, 2, 3])
le = np.array([1, 0, 1])
p = np.array([1, 1, 1])
# 직선 위의 점  ls 에서 점 p로 가는 벡터를 생성해 보자
v = p - ls

# 이 벡터가 ls->le 벡터에 떨어뜨리는 그림자의 길이를 계산하자 : k
u = le - ls
u_norm = np.linalg.norm(u)
u_direction = u / u_norm

k = v.dot(u)/u_norm

#수선의 발
p_proj = ls + u_direction * k

v_p2p_proj = p_proj - p


mySpace = axis3d(x=[-3,3], y=[-3, 3], z=[-3, 3])
draw_points(mySpace, [ls, le], labels=['ls', 'le'], color='green')
draw_points(mySpace, [p], labels=['p'], color='red')

draw_vec3d(mySpace, p, color='green')
draw_vec3d(mySpace, v, start_from = ls, color='red')

draw_vec3d(mySpace, u_direction*k, start_from = ls, color='blue')
draw_vec3d(mySpace, v_p2p_proj, start_from = p, color='blue')

line = le - ls
draw_vec3d(mySpace, 10*line, start_from = ls, color='black')

distance = np.linalg.norm(v_p2p_proj)
print(distance)
```

```
0.7071067811865476
```



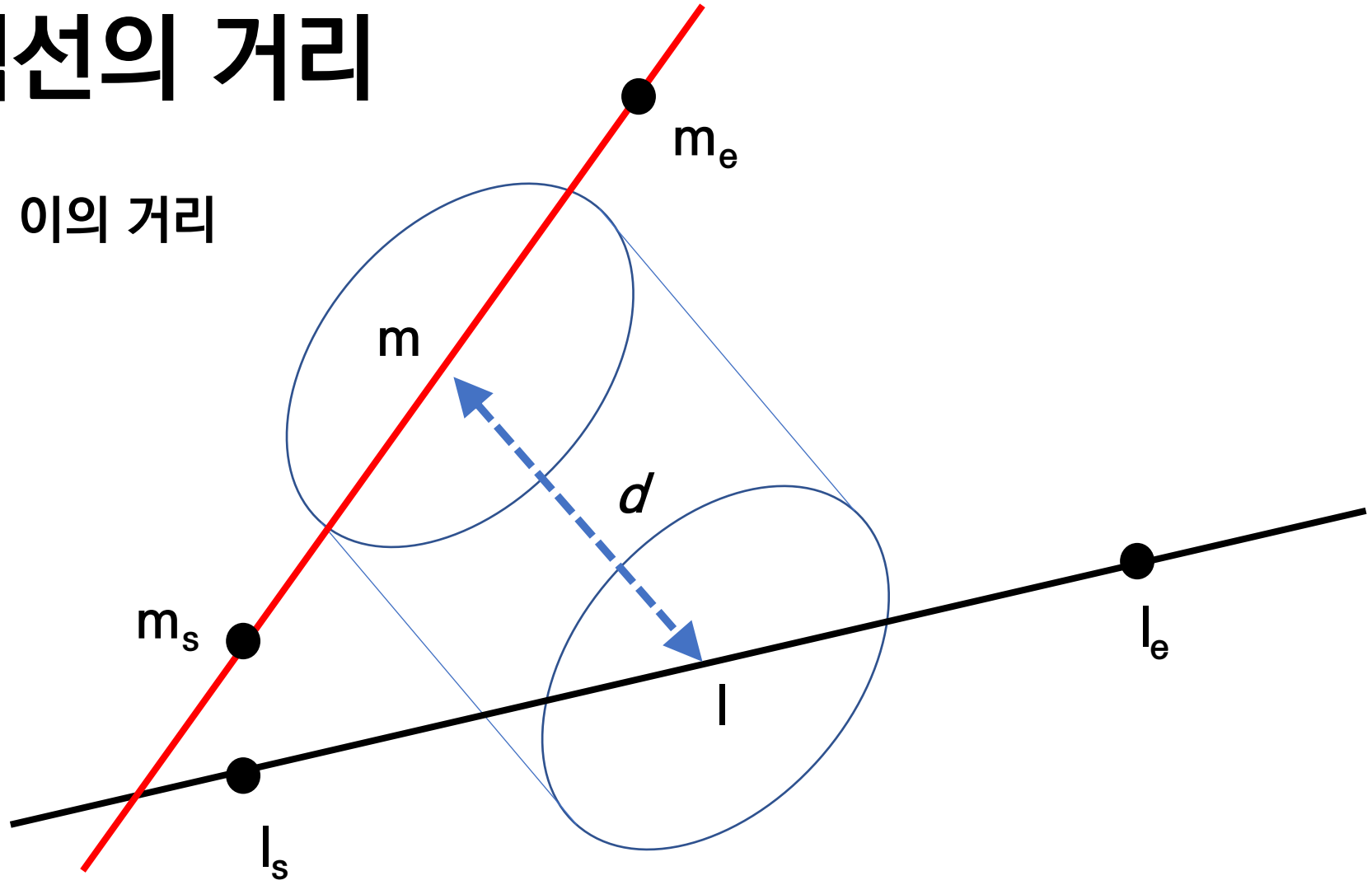
직선과 직선의 거리

두 직선 L과 M 사이의 거리는?

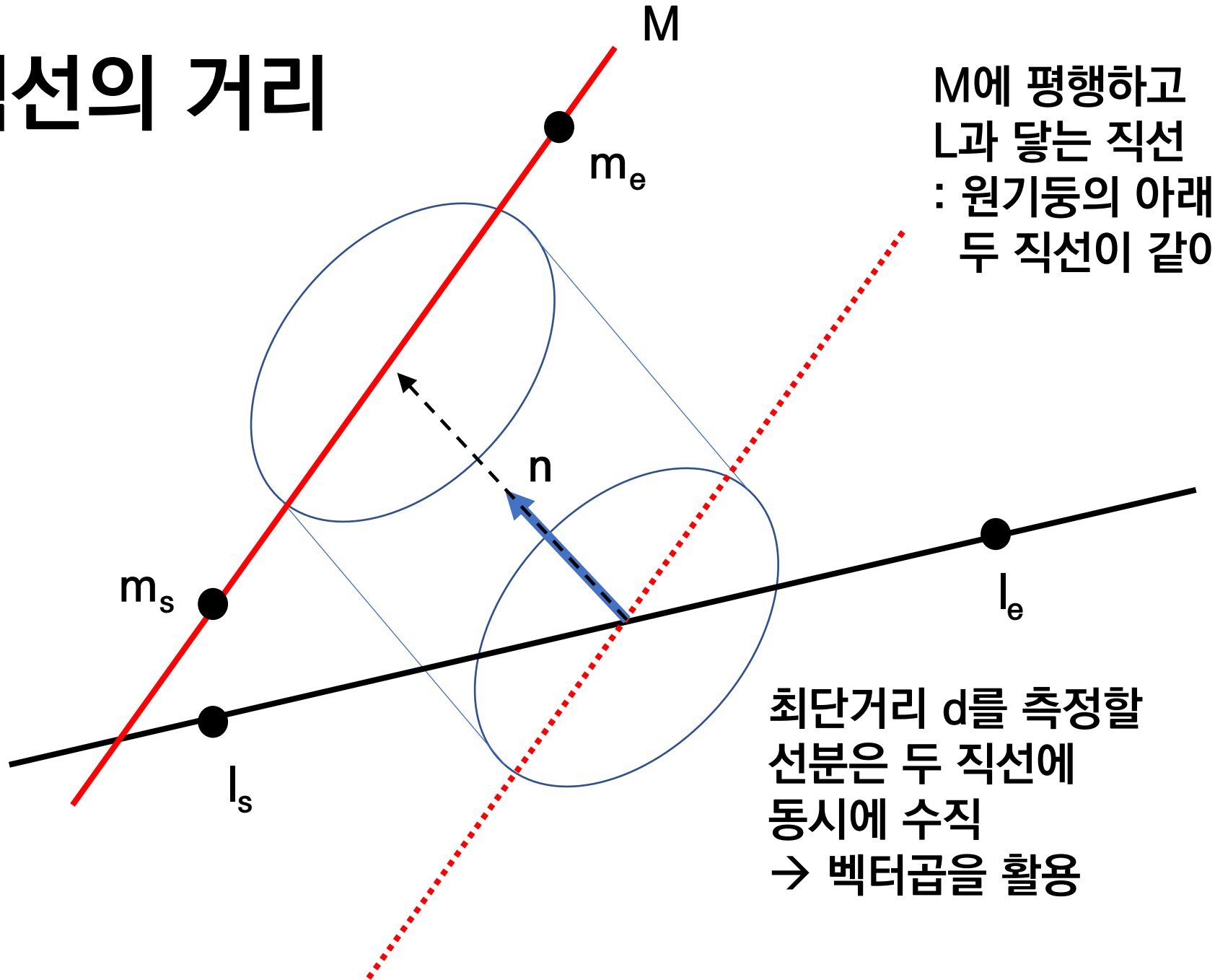


직선과 직선의 거리

최근접점 m, l 이의 거리



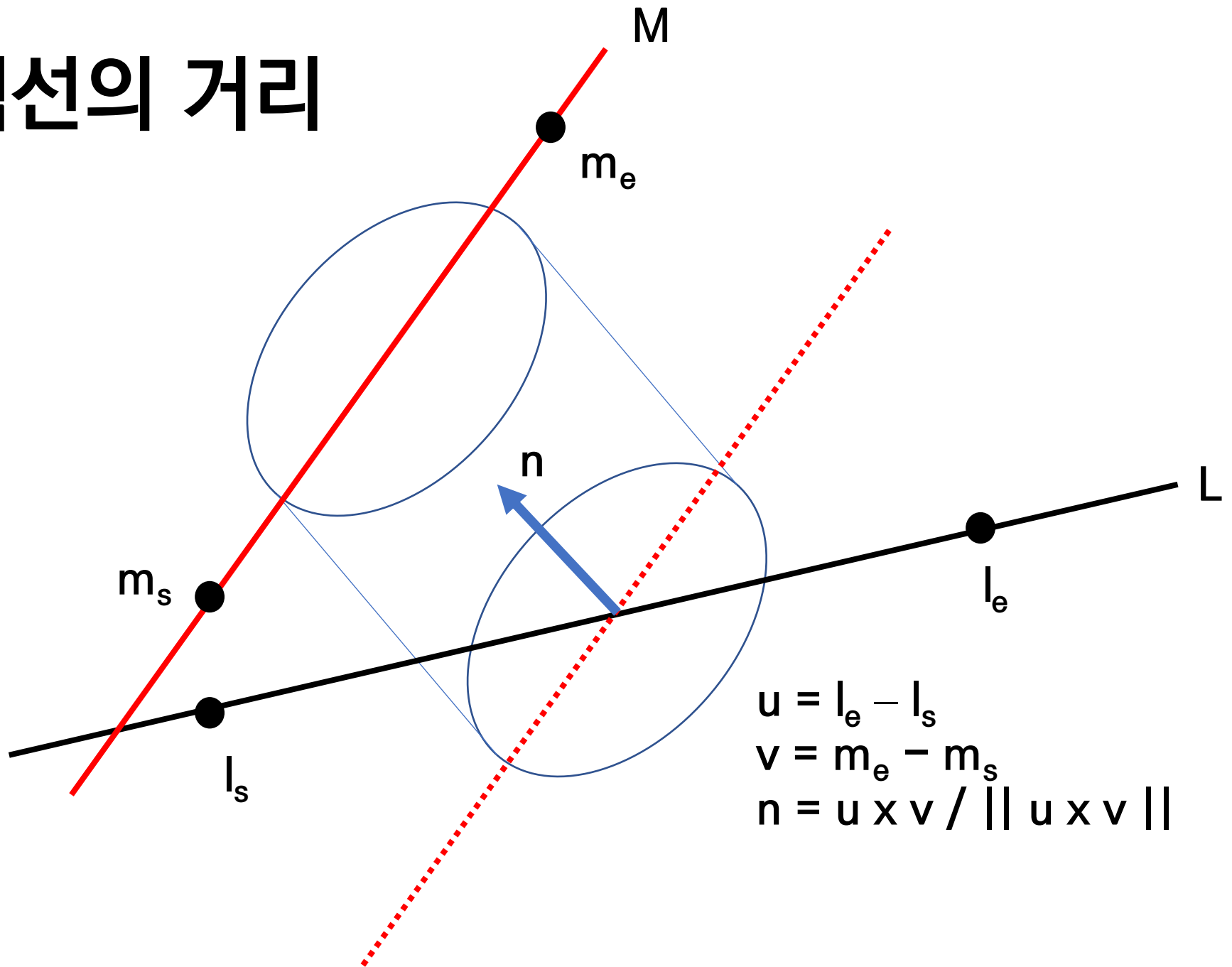
직선과 직선의 거리



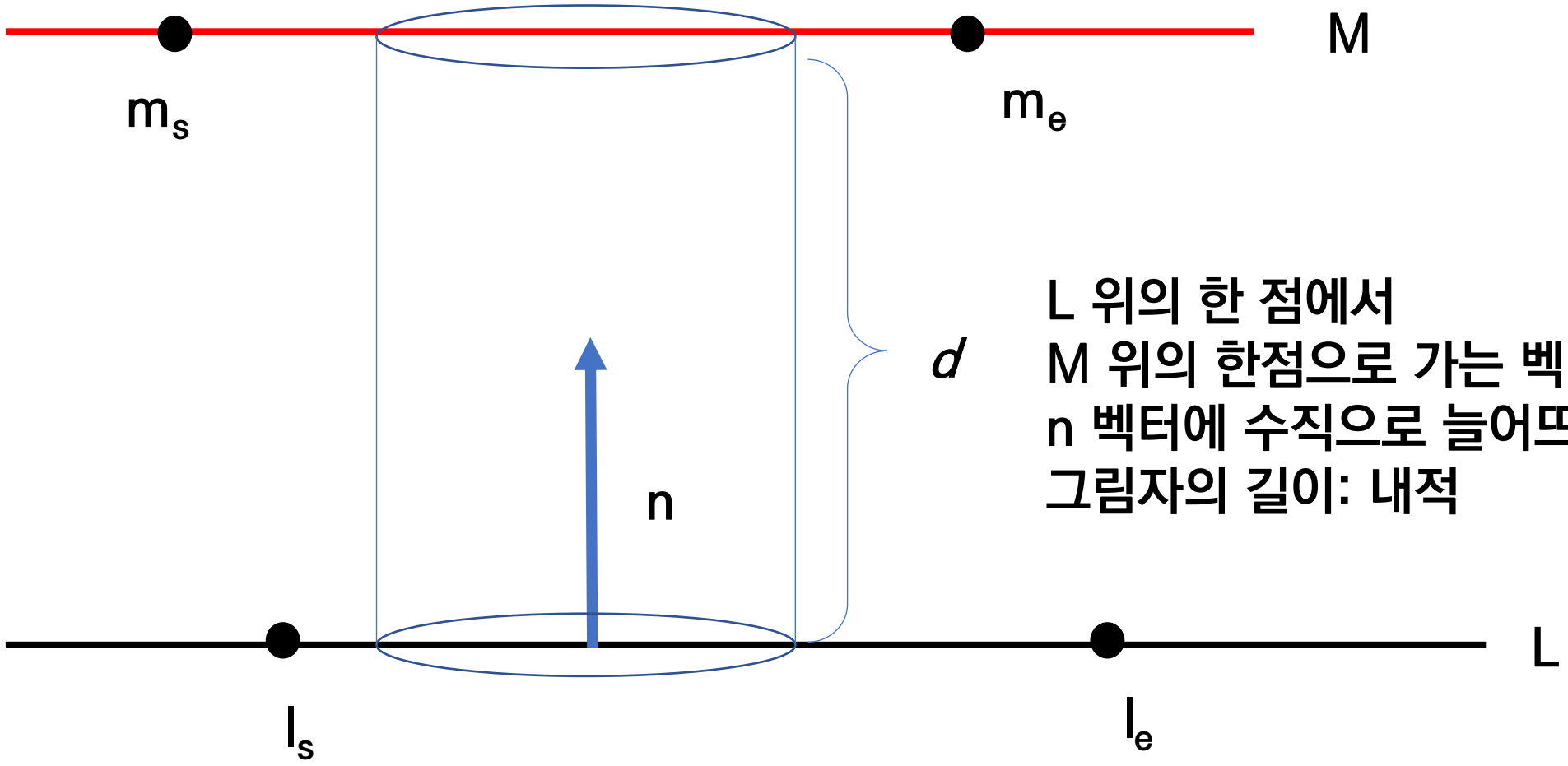
M에 평행하고
L과 닿는 직선
: 원기둥의 아래쪽 면에
두 직선이 같이 놓임

최단거리 d를 측정할
선분은 두 직선에
동시에 수직
→ 벡터곱을 활용

직선과 직선의 거리

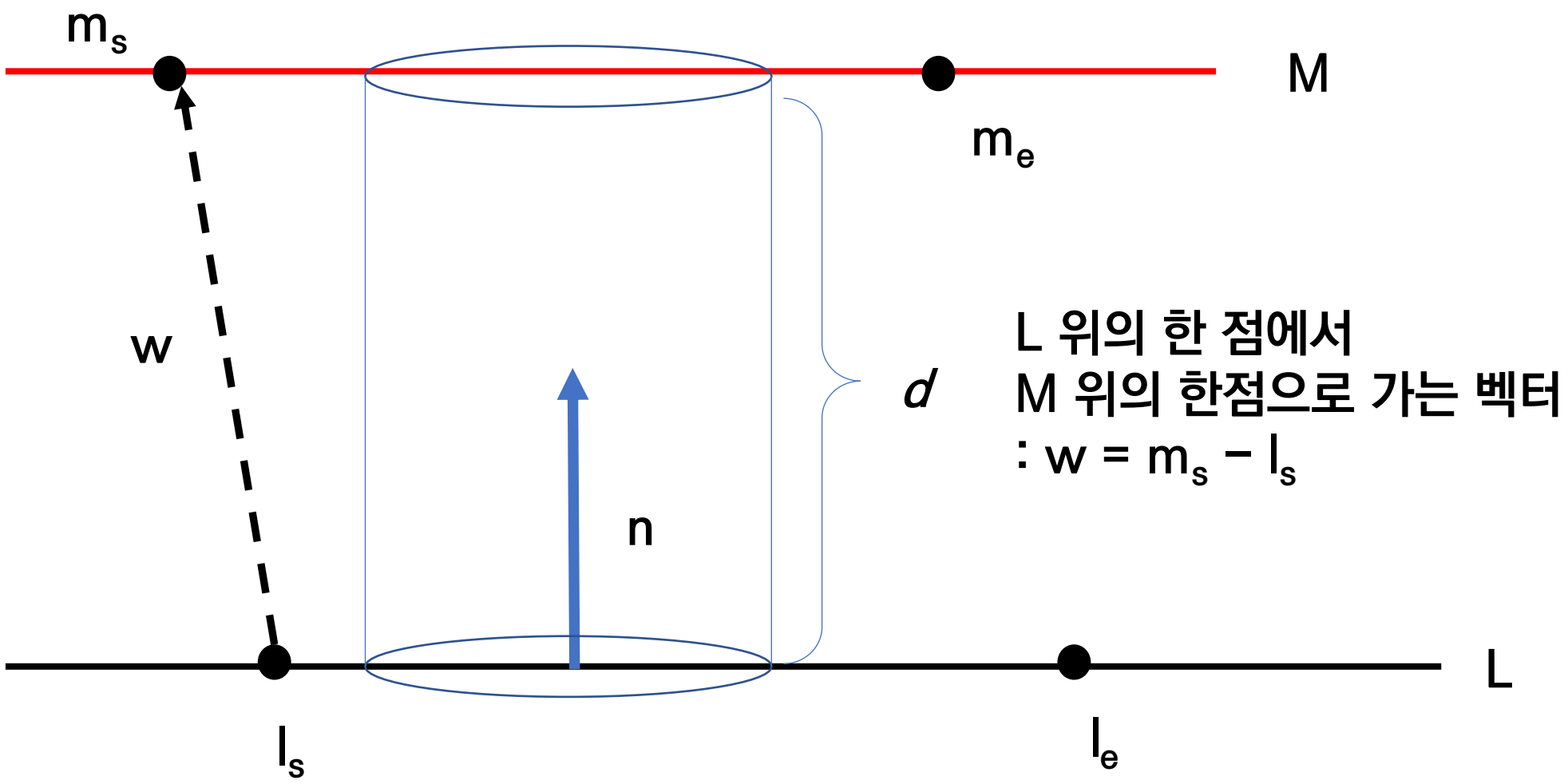


직선과 직선의 거리



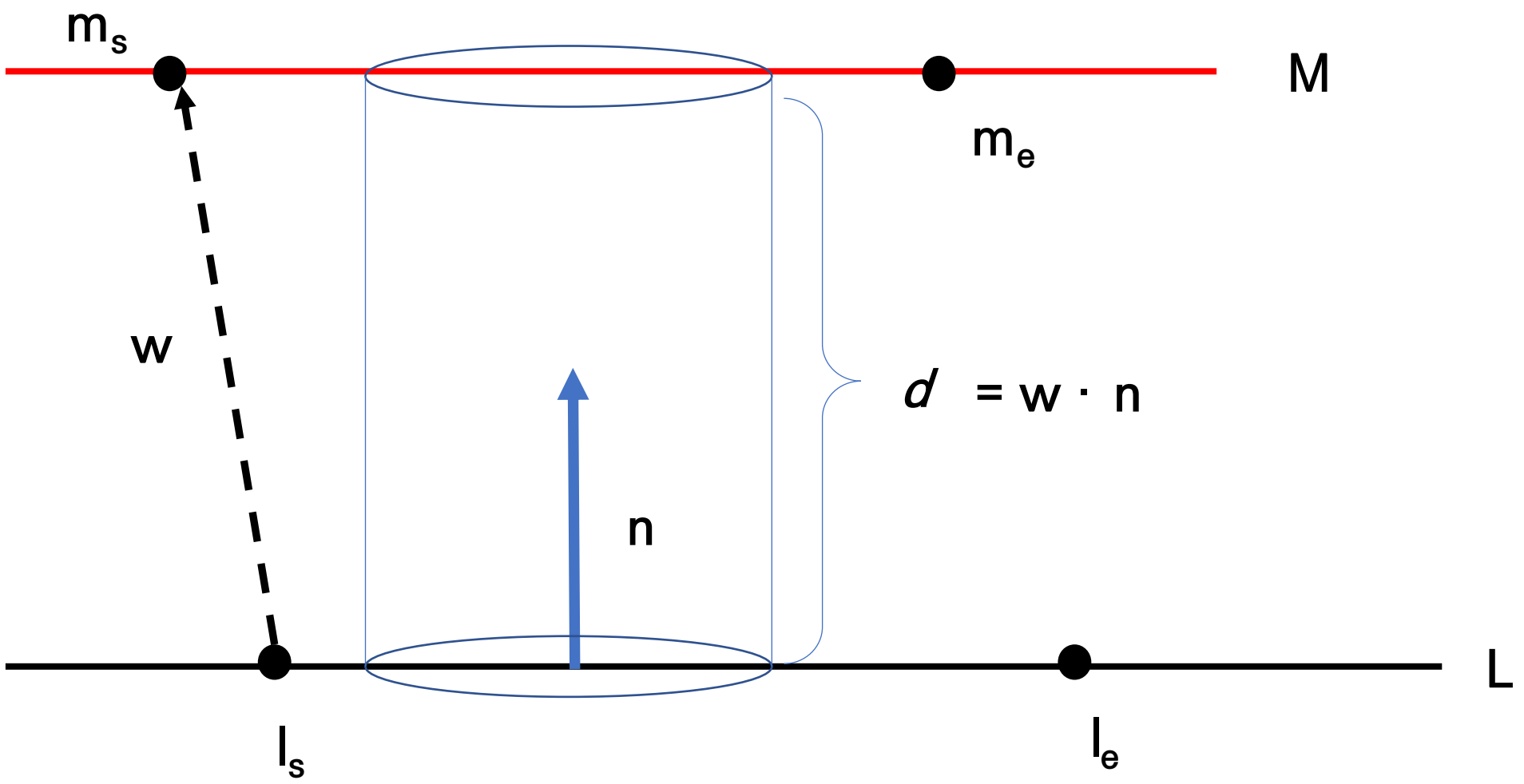
L 위의 한 점에서
M 위의 한점으로 가는 벡터가
n 벡터에 수직으로 늘어뜨리는
그림자의 길이: 내적

직선과 직선의 거리



L 위의 한 점에서
M 위의 한점으로 가는 벡터
: $w = m_s - l_s$

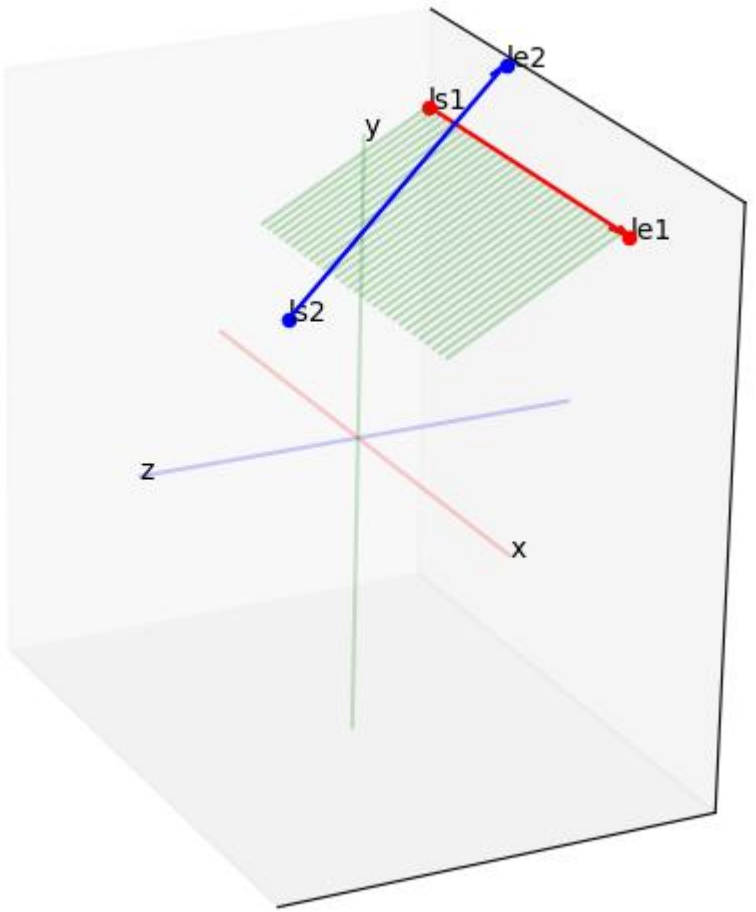
직선과 직선의 거리



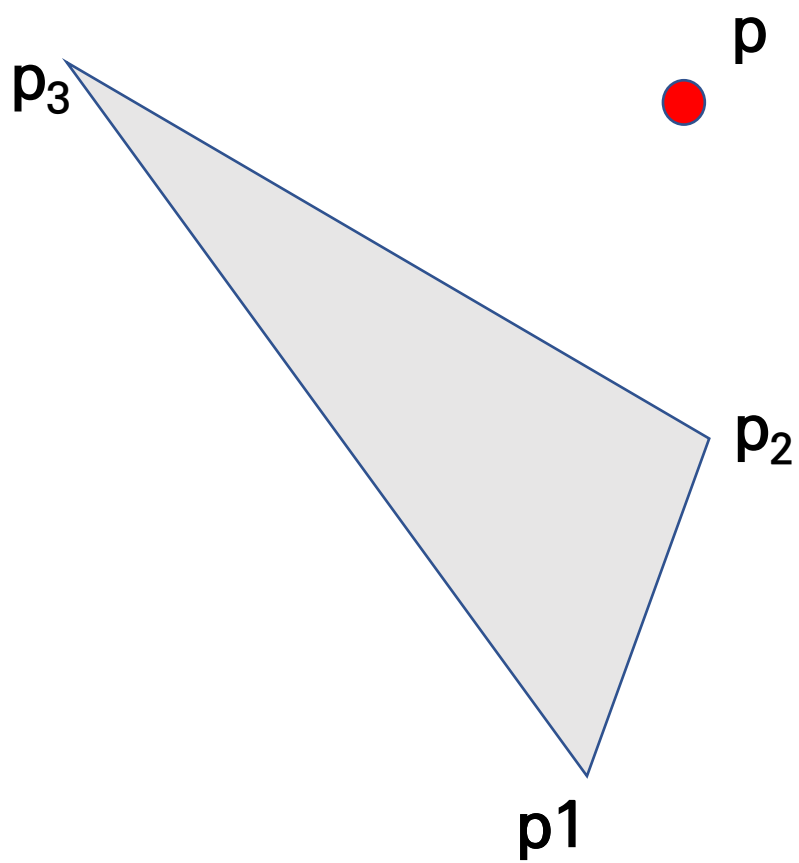
구현 – 두 직선 사이의 거리

```
ls1 = np.array([-3, 2, -3])
le1 = np.array([1, 2, -3])

ls2 = np.array([-3, 0, -1])
le2 = np.array([4, 5, 1])

mySpace = axis3d(x=[-3,3], y=[-3, 3], z=[-3, 3])

line1 = le1 - ls1
line2 = le2 - ls2

draw_vec3d(mySpace, line1, start_from = ls1, color='red')
draw_points(mySpace, [ls1, le1], labels=['ls1', 'le1'], color='red')
draw_vec3d(mySpace, line2, start_from = ls2, color='blue')
draw_points(mySpace, [ls2, le2], labels=['ls2', 'le2'], color='blue')

# 두 벡터에 동시에 수직인 벡터 N을 구하자
N = np.cross(line1, line2)
N_norm = np.linalg.norm(N)
N = N / N_norm

# line1 위에 있는 아무 점이나 선택
p1 = ls1

# line2  위에 있는 아무 점을 선택
p2 = le2

vec = p2 - p1
distance = vec.dot(N)
print(distance)

steps = 30
d = np.linalg.norm(line1)/steps
line_dir = line1 / np.linalg.norm(line1)
for i in range(steps):
    starting_location = ls1 + line_dir * (d * i)
    draw_vec3d(mySpace, distance*N, start_from=starting_location, color='green', alpha=0.25)
```

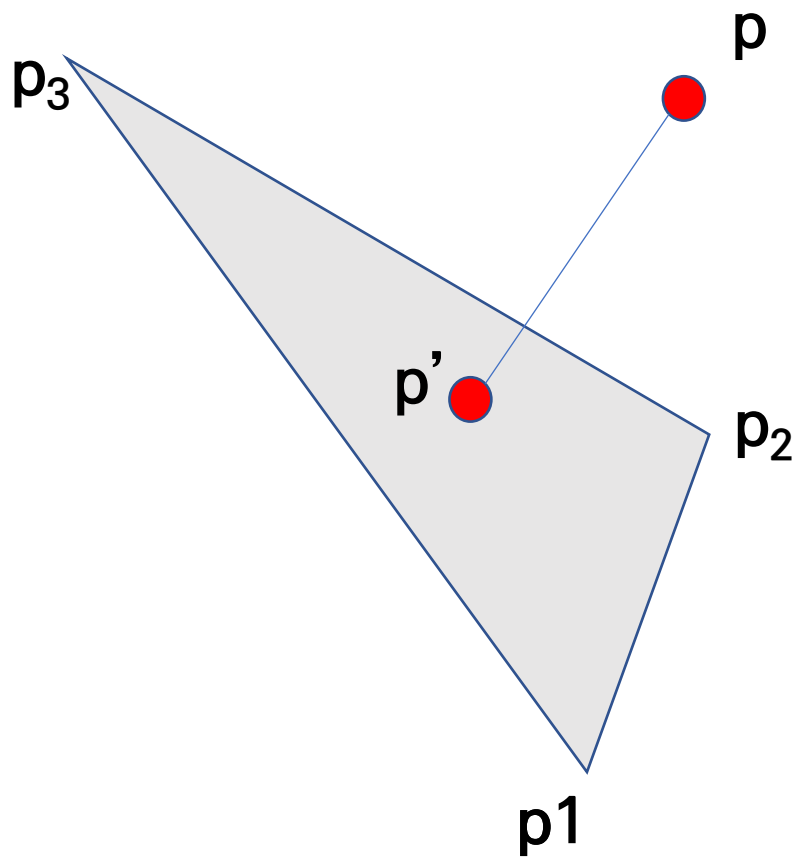
```
2.5997347344787265
```



평면과 점 사이의 거리



평면과 점 사이의 거리: 한 가지 방법

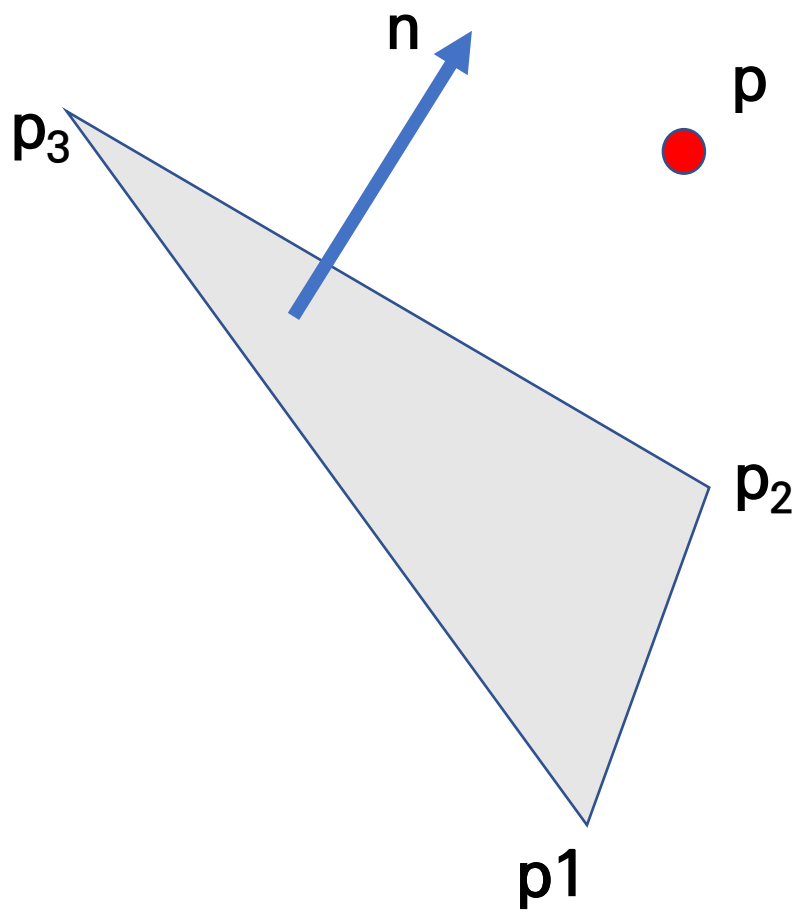


점 p에서 평면 위로 수선의 발을 내린다.
그리고 이 수선의 발 p'와 p의 거리 계산

… 가능하지만,
수선의 발을 구하는 계산 등이 필요

좀 더 간단한 방법은?

평면과 점 사이의 거리: 법선 벡터 이용하기

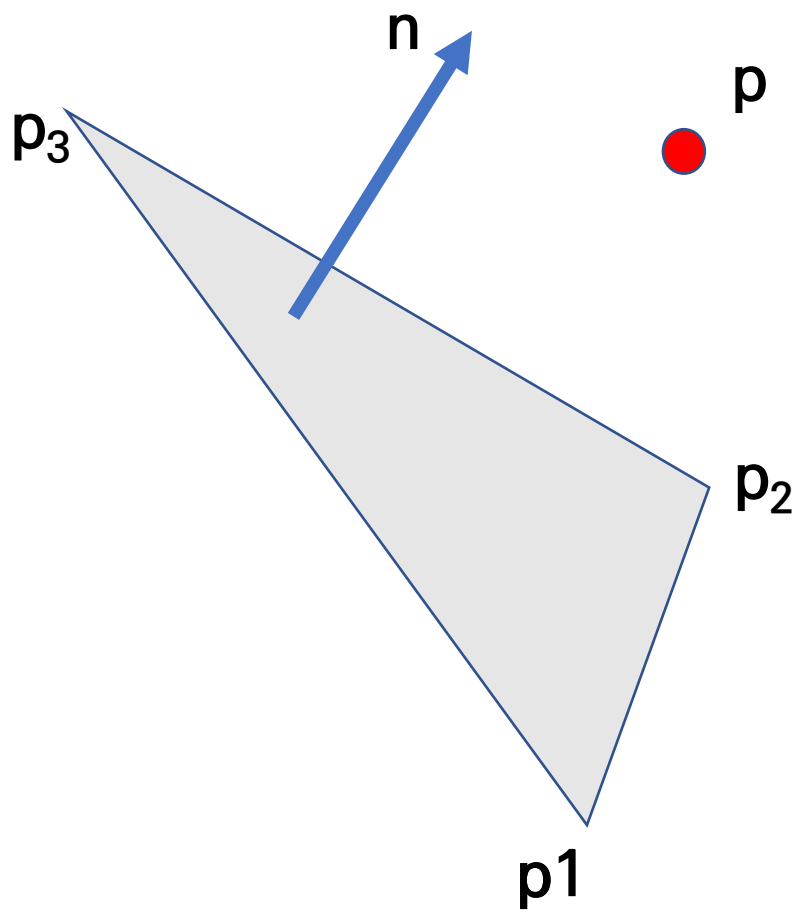


평면의 법선 벡터를 안다면?

점과 평면의 거리는…

평면 위의 한 점에서 점 p로 가는 벡터가
법선에 떨어뜨리는 그림자의 길이 (내적)

평면과 점 사이의 거리: 법선 벡터 이용하기



평면의 법선 벡터
: 평면에 수직인 단위 벡터

평면 위의 두 벡터를 가위곱하여 정규화

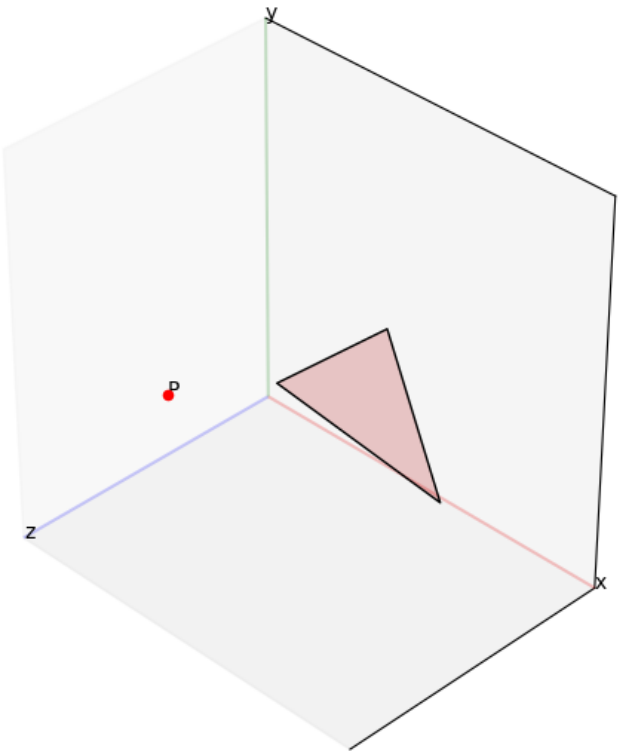
$u = p_2 - p_1$
$v = p_3 - p_1$

$n = u \times v / || u \times v ||$

$d = n \cdot (p - p_1)$

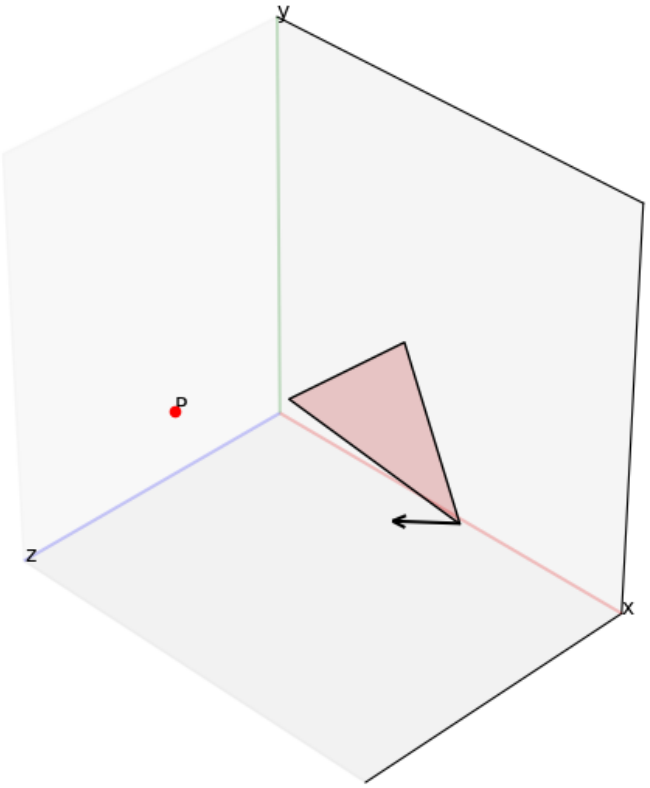
구현 – 삼각형과 한 점

```
[50] # Define the vertices of the triangle
     p0 = np.array([3.2, 1.1, 1.5])
     p1 = np.array([2.2, 2.2, 1.0])
     p2 = np.array([1.1, 1.2, 1.3])

     P = np.array([1.9, 2.5, 4.0])

     triangle = np.array([p0, p1, p2])
     color = np.array([1, 0, 0]) * 0.7

     my_axis = axis3d(x=[0, 4], y=[0,4], z=[0,4])
     draw_polygons(my_axis, [triangle], facecolors=[color], edgecolors='black', alpha=0.2)
     draw_points(my_axis, [P], labels=['P'], color='red')

     setCam(my_axis, [1, 1, 1])
```



구현 – 삼각형과 한 점

```python
# Define the vertices of the triangle
p0 = np.array([3.2, 1.1, 1.5])
p1 = np.array([2.2, 2.2, 1.0])
p2 = np.array([1.1, 1.2, 1.3])

# 법선 벡터를 구하자
u = p1 - p0
v = p2 - p0
N = np.cross(u, v)
N_norm = np.linalg.norm(N)
N = N / N_norm  # 법선 벡터


# 삼각형 위의 임의 점 (p0, p1, or p2)에서  P로 가는 벡터를 구한다
w = P - p2

# 점과 평면의 거리는 이 벡터가 법선벡터에 드리우는 그림자의 길이
distance = w.dot(N)
print(distance)

triangle = np.array([p0, p1, p2])
color = np.array([1, 0, 0]) * 0.7

my_axis = axis3d(x=[0, 4], y=[0,4], z=[0,4])
draw_polygons(my_axis, [triangle], facecolors=[color], edgecolors='black', alpha=0
draw_vec3d(my_axis, N, start_from = p0, color='black')

draw_points(my_axis, [P], labels=['P'], color='red')
setCam(my_axis, [1, 1, 1])
```



구현 – 삼각형과 한 점, 그리고 삼각형의 법선, 그리고 거리

```python
# Define the vertices of the triangle
p0 = np.array([3.2, 1.1, 1.5])
p1 = np.array([2.2, 2.2, 1.0])
p2 = np.array([1.1, 1.2, 1.3])

# 법선 벡터를 구하자
u = p1 - p0
v = p2 - p0
N = np.cross(u, v)
N_norm = np.linalg.norm(N)
N = N / N_norm  # 법선 벡터

# 삼각형 위의 임의 점 (p0, p1, or p2)에서  P로 가는 벡터를 구한다
w = P - p0

# 점과 평면의 거리는 이 벡터가 법선벡터에 드리우는 그림자의 길이
distance = w.dot(N)
print(distance)

triangle = np.array([p0, p1, p2])
displacement = distance * N
triangle2 = np.array([p0 + displacement, p1 + displacement, p2 + displacement])

color = np.array([1, 0, 0]) * 0.7
my_axis = axis3d(x=[0, 4], y=[0,4], z=[0,4])
draw_polygons(my_axis, [triangle], facecolors=[color], edgecolors='black', alpha=0.2)
draw_polygons(my_axis, [triangle2], facecolors=[color], edgecolors='black', alpha=0.2)
draw_vec3d(my_axis, N, start_from = p0, color='black')
draw_vec3d(my_axis, distance * N, start_from = p0, color='green', alpha=0.5)
draw_vec3d(my_axis, w, start_from = p0, color='red')

draw_points(my_axis, [P], labels=['P'], color='red')
setCam(my_axis, [1, 1, 1])
```

```
2.9217486860808863
```