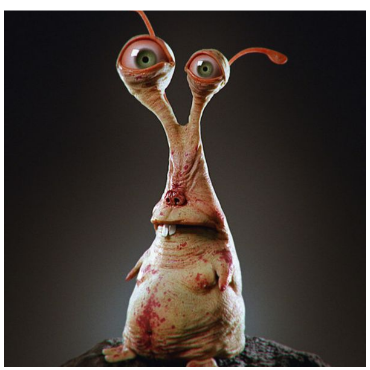
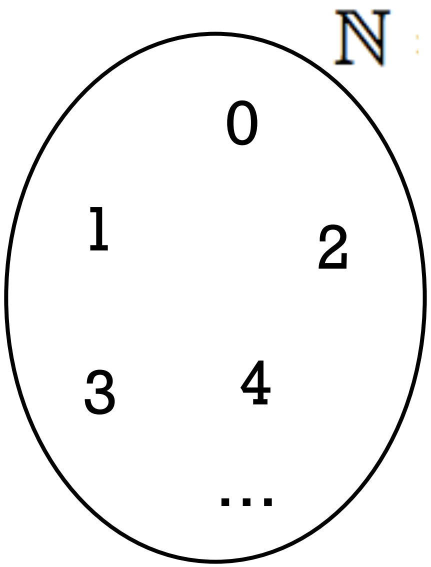
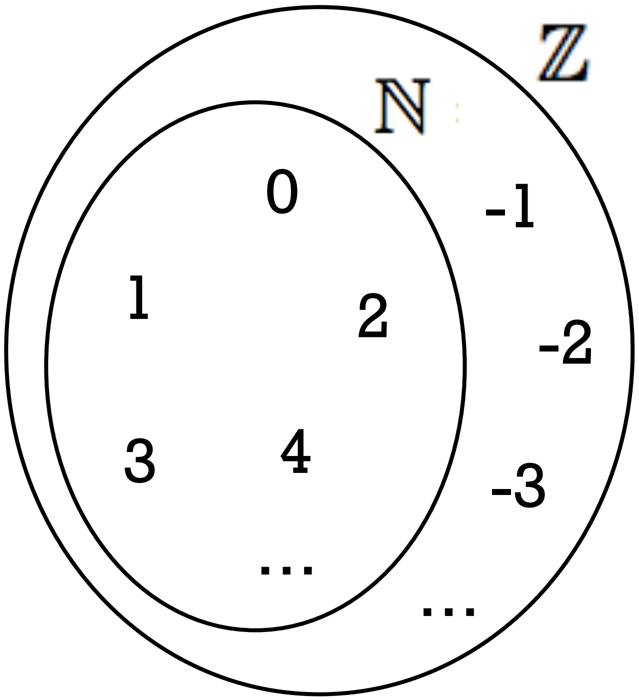
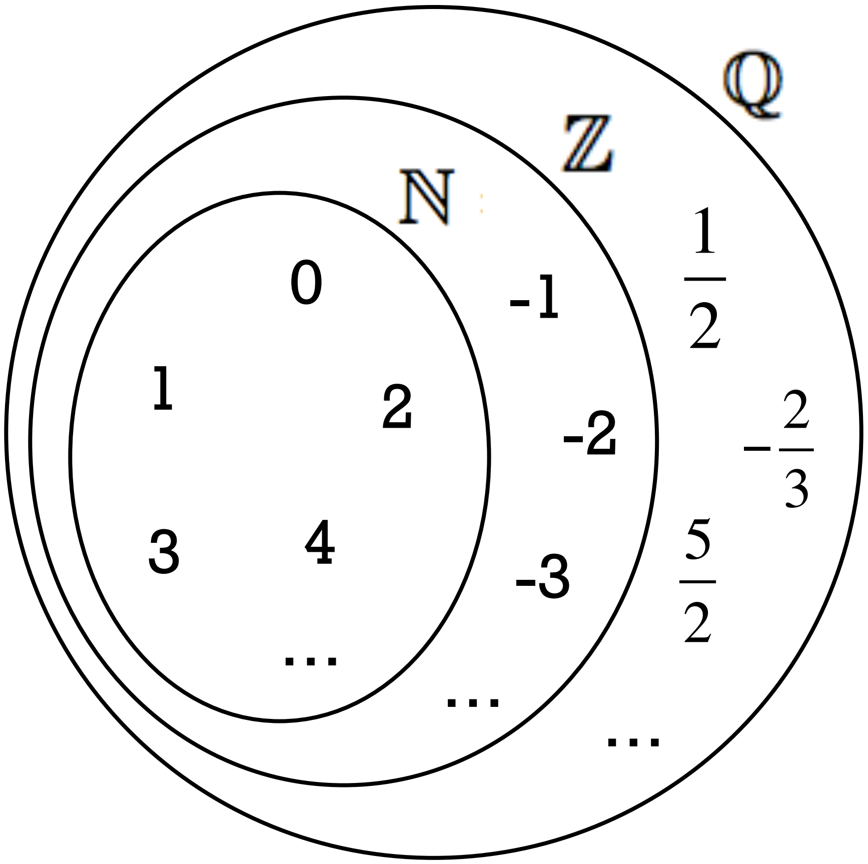
3D Viewing

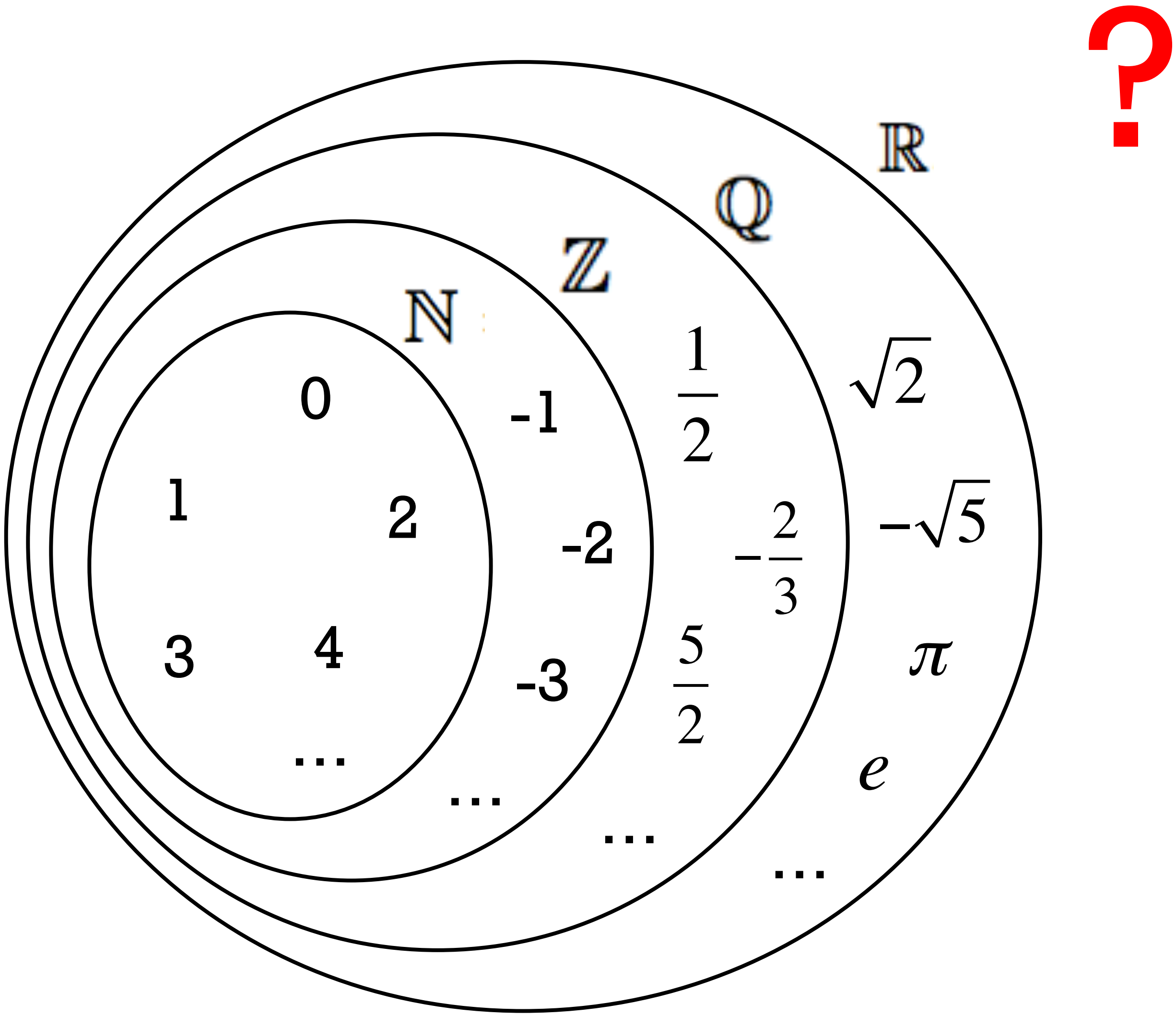
3D Computer Graphics (Lab 3)




ℕ
0
1
2
3
4
...


$\mathbb{Z}$
$\mathbb{N}$
0
1
2
3
4
...
-1
-2
-3
...


$\mathbb{Q}$
$\mathbb{Z}$
$\mathbb{N}$
0
1
2
3
4
...
-1
-2
-3
...
$\frac{1}{2}$
$-\frac{2}{3}$
$\frac{5}{2}$
...


?
ℝ
ℚ
ℤ
ℕ
0
1
2
3
4
...
-1
-2
-3
...
1/2
-2/3
5/2
...
√2
-√5
π
e
...

Complex numbers

A **complex number** is a number of the form

$$a + bi$$

with a and b real numbers and i the **imaginary unit** satisfying $i^2 = -1$.

- Example: $1 + 3i$
- If $a = 0$, the number is called **pure imaginary**. Example: $2i$

The **complex conjugate** of a complex number $z = a + bi$ is given by

$$z^* = a - bi$$

- Example: The complex conjugate of $z = 1 + 3i$ equals

$$z^* = 1 - 3i$$

Complex numbers

The product of two complex numbers $a + bi$ and $c + di$ equals

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i$$

- Proof: $$\begin{aligned}(a + bi)(c + di) &= ac + bci + adi + bdi^2 \\ &= ac + bci + adi - bd \\ &= ac - bd + (bc + ad)i\end{aligned}$$
- Example: The product of $2 + 3i$ and $4 - i$ equals

$$(2 + 3i)(4 - i) = (8+3) + (12 - 2)i = 11 + 10i$$

Use of complex numbers

- Equation

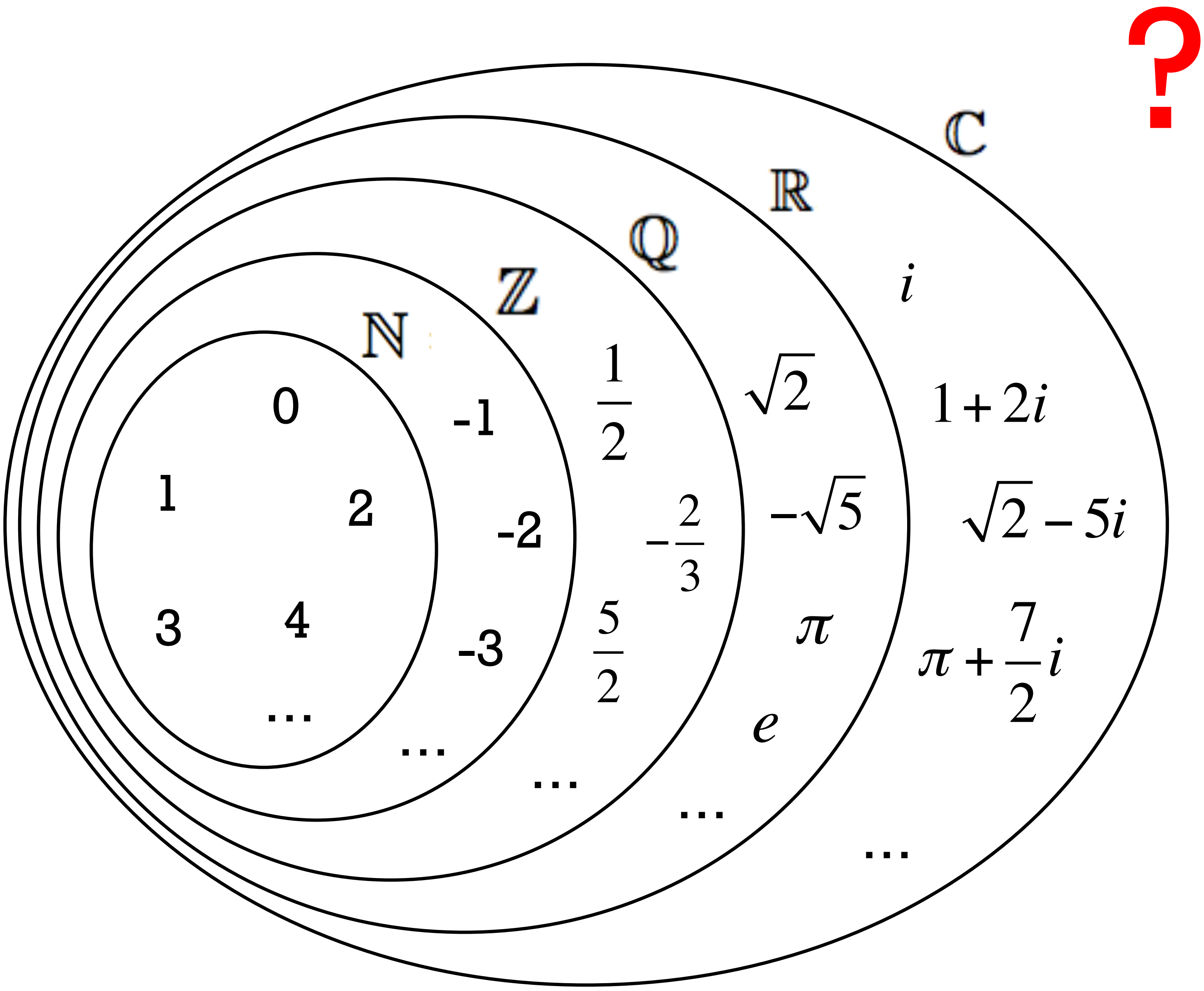
$$(x+1)^2 = -4$$

has no real solution.

But it has two complex numbers as solution: -1 + 2i and -1 – 2i.

Proof for -1 + 2i

$$\begin{aligned}(x+1)^2 &= (-1+2i+1)^2 \\ &= (2i)^2 \\ &= 4i^2 \\ &= 4(-1) \\ &= -4\end{aligned}$$


?
$\mathbb{C}$
$\mathbb{R}$
$\mathbb{Q}$
$\mathbb{Z}$
$\mathbb{N}$
0
1
2
3
4
...
-1
-2
-3
...
$\frac{1}{2}$
$-\frac{2}{3}$
$\frac{5}{2}$
...
$\sqrt{2}$
$-\sqrt{5}$
π
e
...
i
$1+2i$
$\sqrt{2}-5i$
$\pi+\frac{7}{2}i$
...

Quaternions

A quaternion is a number of the form

$$a + bi + cj + dk$$

with a, b, c and d real numbers and i, j and k having the following properties.

x	**i**	**j**	**k**
i	-1	k	-j
j	-k	-1	i
k	j	-i	-1

- Example: $1 + 3i - 2j + 5k$
- If $a = 0$, the number is called pure imaginary. Example: $i - 2j + k$

Quaternions

The conjugate of a quaternion $q = a + bi + cj + dk$ is given by

$$q^* = a - bi - cj - dk$$

- Example: The conjugate of $q = 1 + 2i - 3j + 4k$ equals

$$q^* = 1 - 2i + 3j - 4k$$

The product of two quaternions

$$a_1 + b_1 i + c_1 j + d_1 k \quad \text{and} \quad a_2 + b_2 i + c_2 j + d_2 k$$

equals

$$(a_1 + b_1 i + c_1 j + d_1 k)\,(a_2 + b_2 i + c_2 j + d_2 k) =$$

$$(a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2) + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2)\,i +$$

$$(a_1 c_2 - b_1 d_2 + c_1 a_2 + d_1 b_2)\,j + (a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2)\,k$$

Quaternions

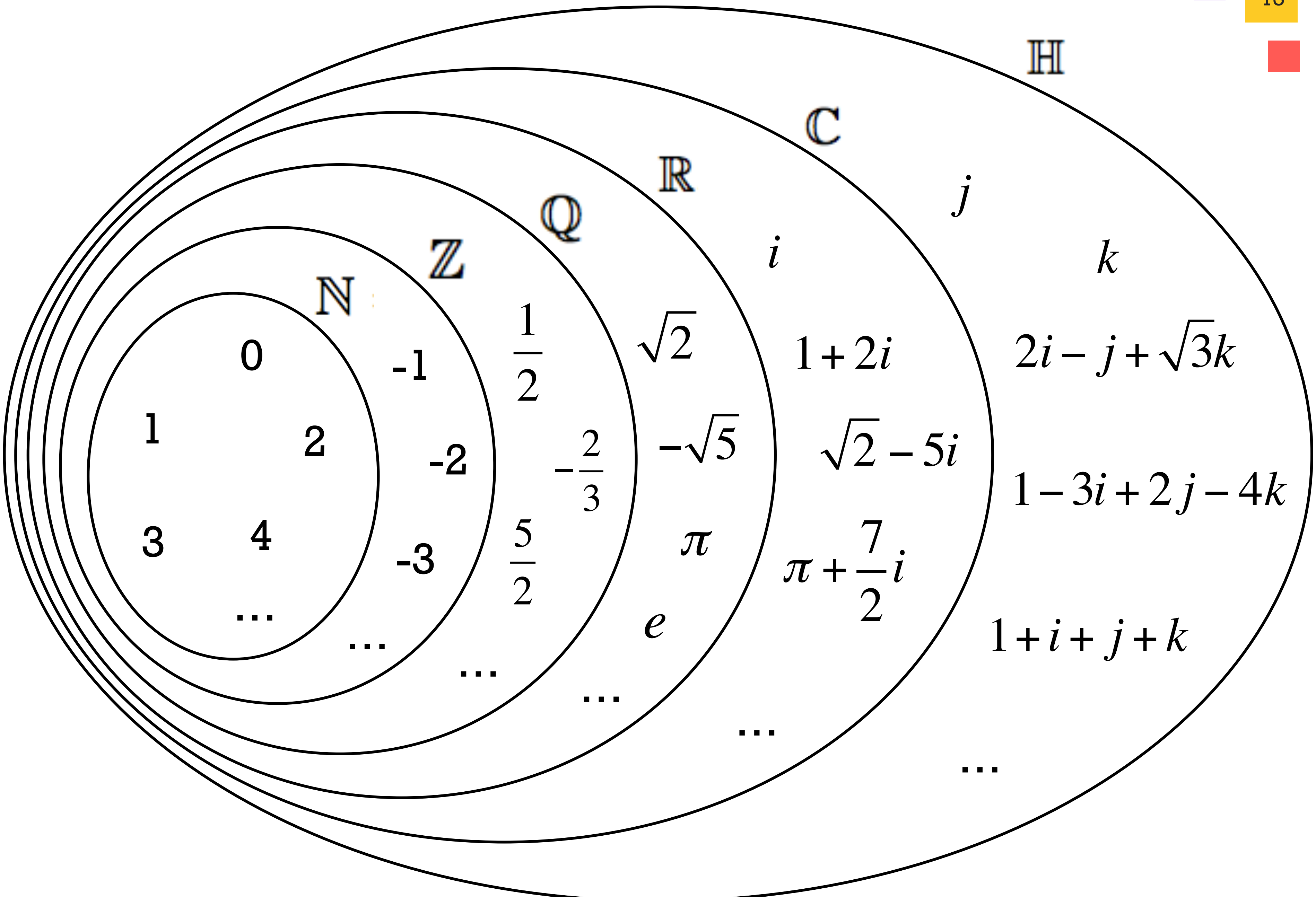
- Proof:

$$(a_1 + b_1 i + c_1 j + d_1 k)\ (a_2 + b_2 i + c_2 j + d_2 k) \ =$$

$$a_1a_2 + a_1b_2 i + a_1c_2 j + a_1d_2 k + b_1a_2 i + b_1b_2 i^2 + b_1c_2 ij + b_1d_2 ik +$$
$$c_1a_2 j + c_1b_2 ji + c_1c_2 j^2 + c_1d_2 jk + d_1a_2 k + d_1b_2 ki + d_1c_2 kj + d_1d_2 k^2 =$$

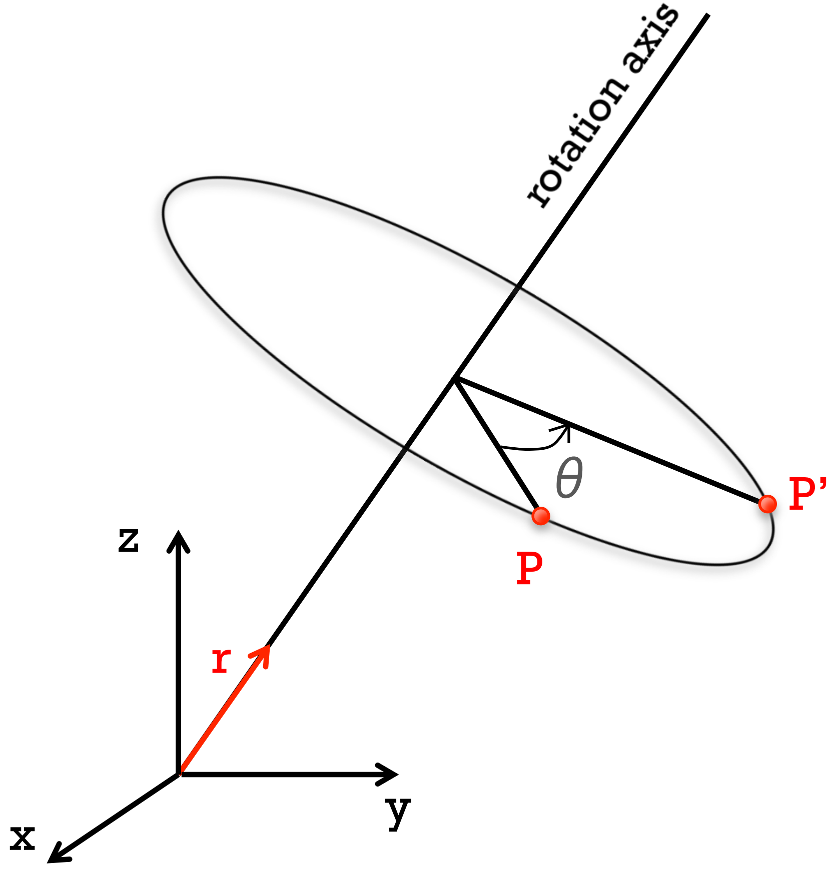
$$a_1a_2 + a_1b_2 i + a_1c_2 j + a_1d_2 k + b_1a_2 i + b_1b_2(-1) + b_1c_2 k + b_1d_2(-j) +$$
$$c_1a_2 j + c_1b_2(-k) + c_1c_2(-1) + c_1d_2 i + d_1a_2 k + d_1b_2 j + d_1c_2(-i) + d_1d_2(-1) =$$

$$(a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) \ + \ (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\, i \ +$$
$$(a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)\, j + \ (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)\, k$$


$\mathbb{H}$
$\mathbb{C}$
$\mathbb{R}$
$\mathbb{Q}$
$\mathbb{Z}$
$\mathbb{N}$
0
1
2
3
4
...
-1
-2
-3
...
$\frac{1}{2}$
$-\frac{2}{3}$
$\frac{5}{2}$
...
$\sqrt{2}$
$-\sqrt{5}$
π
e
...
i
$1+2i$
$\sqrt{2}-5i$
$\pi+\frac{7}{2}i$
...
j
k
$2i-j+\sqrt{3}k$
$1-3i+2j-4k$
$1+i+j+k$
...

Use of quaternions

Rotations in 3D around an arbitrary axis.



$r = (r_x, r_y, r_z)$ with $|r| = 1$

θ

$P = (x, y, z)$

$P' = ?$

Use of quaternions

1. Use the coordinates of P to define a pure imaginary quaternion p

$$p = xi + yj + zk$$

2. Use the rotation data to define a so-called *rotation quaternion* q

$$q = \cos(\theta/2) + \sin(\theta/2)r_x i + \sin(\theta/2)r_y j + \sin(\theta/2)r_z k$$

3. Compute the quaternion

$$p' = q.p.q^*$$

p' is a pure imaginary quaternion which contains the coordinates of the point P' we are searching for!

Example

Rotate the point (0,1,0) by 90° around the z-axis.

r = (0,0,1) θ = 90° P = (0,1,0)

1. Use the coordinates of P to define a pure imaginary quaternion p

p = 0i + 1j +0k = j

2. Use the rotation data to define a so-called *rotation quaternion* q

$$q = \cos(\theta/2) + \sin(\theta/2)r_x i + \sin(\theta/2)r_y j + \sin(\theta/2)r_z k$$

$$= \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}0i + \frac{\sqrt{2}}{2}0j + \frac{\sqrt{2}}{2}1k = \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}k$$

3. Compute the quaternion

$$p' = q.p.q^* = \left(\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}k\right)j\left(\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}k\right)$$

Example

$$p' = q.p.q^* = \left(\frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}k\right) j \left(\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}k\right)$$

$$= \left(\frac{\sqrt{2}}{2}j + \frac{\sqrt{2}}{2}kj\right)\left(\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}k\right)$$

$$= \left(\frac{\sqrt{2}}{2}j - \frac{\sqrt{2}}{2}i\right)\left(\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2}k\right)$$

$$= \frac{1}{2}j - \frac{1}{2}jk - \frac{1}{2}i + \frac{1}{2}ik$$

$$= \frac{1}{2}j - \frac{1}{2}i - \frac{1}{2}i - \frac{1}{2}j \quad = \quad -i \quad = \quad -i + 0j + 0k$$

p' is a pure imaginary quaternion which contains the coordinates of the point P' we are searching for.

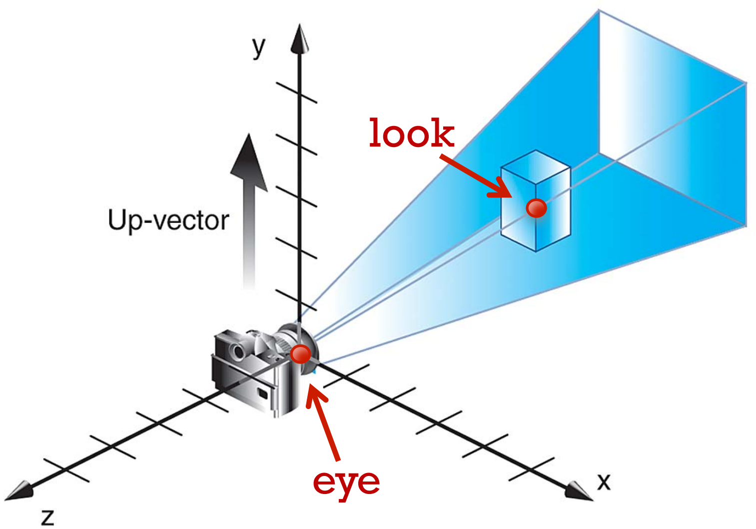
$\boxed{P' = (-1, 0, 0)}$

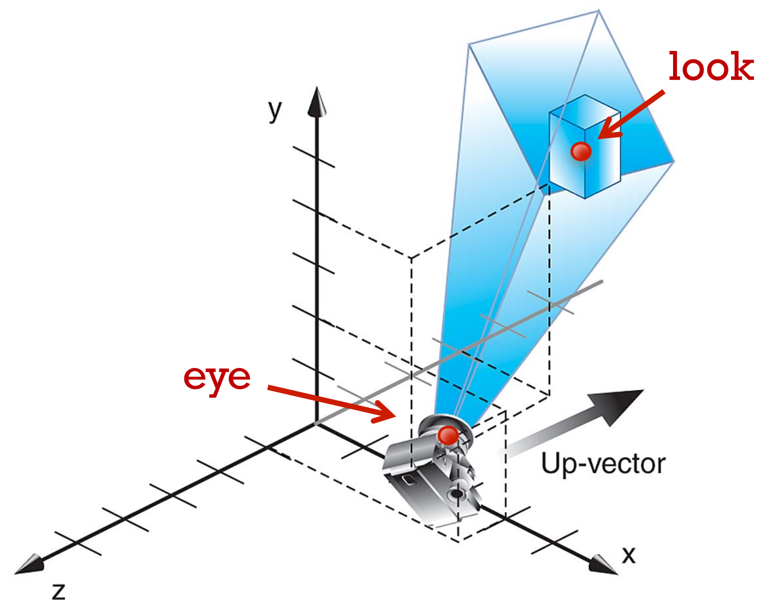
Determine the rotation direction with the right hand rule.

Camera position and orientation

Positioning and aiming the camera

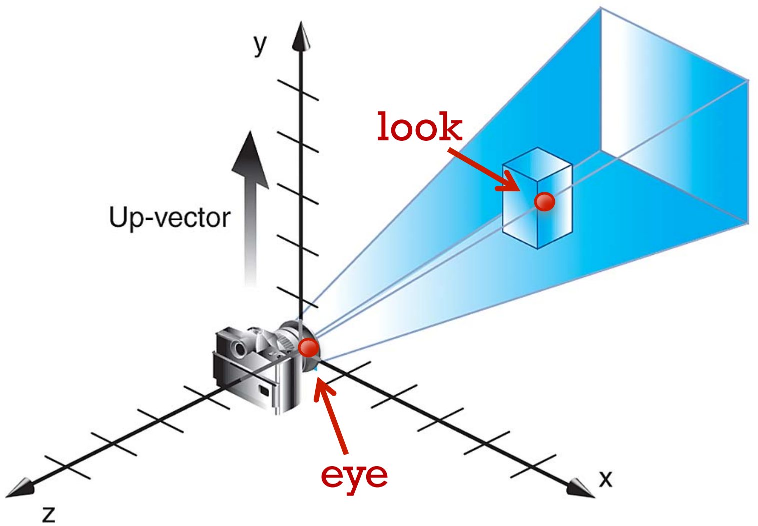
- An *eye* point indicating the location of the camera.
- A reference point (*look*) indicating the point the camera is aimed at.
- An *up vector* indicating the upwards direction of the camera.



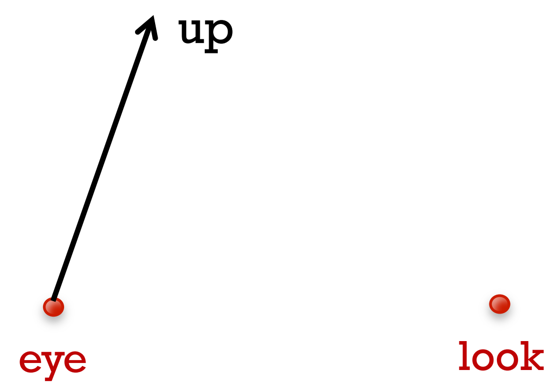


- OpenGL: gluLookAt(eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z);

Positioning and aiming the camera



Assume the user provides …



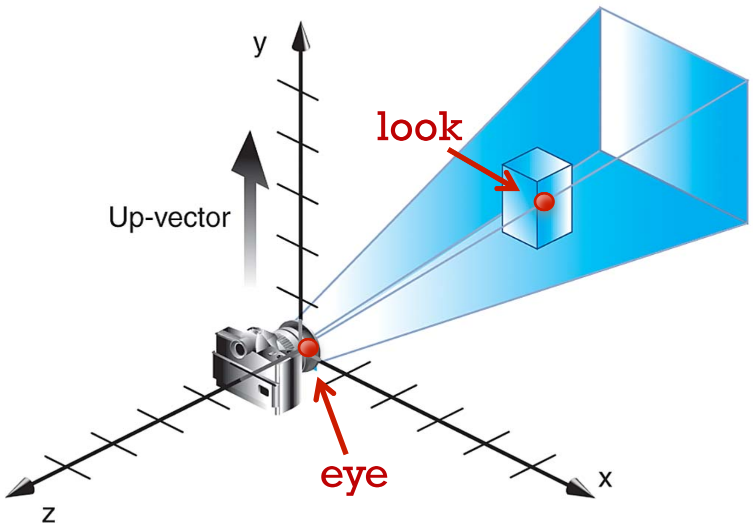

```
gluLookAt( eye.x, eye.y, eye.z,
           look.x, look.y, look.z,
           up.x, up.y, up.z);
```

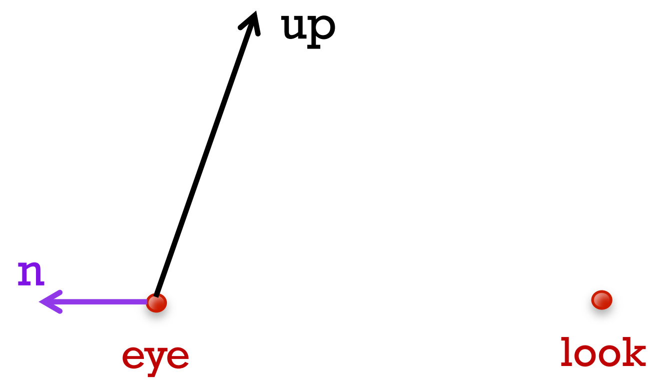
What's wrong?

Data inconsistent!

Solution?

Positioning and aiming the camera

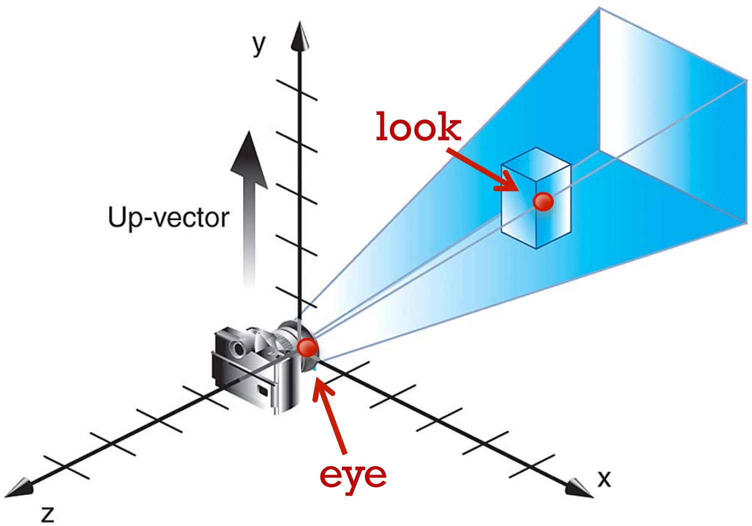


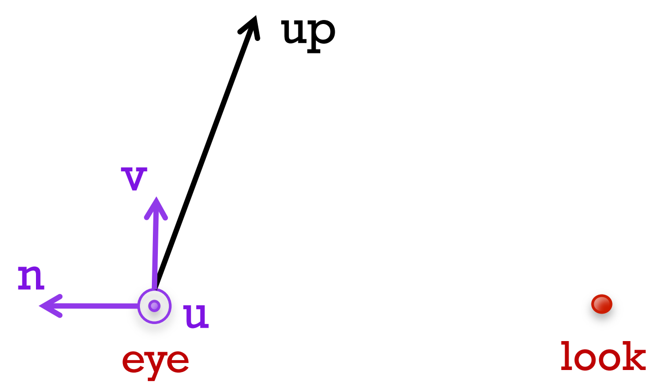


(vector from look to eye) $n = eye - look$

(normalizing) $n = n / |n|$

Positioning and aiming the camera

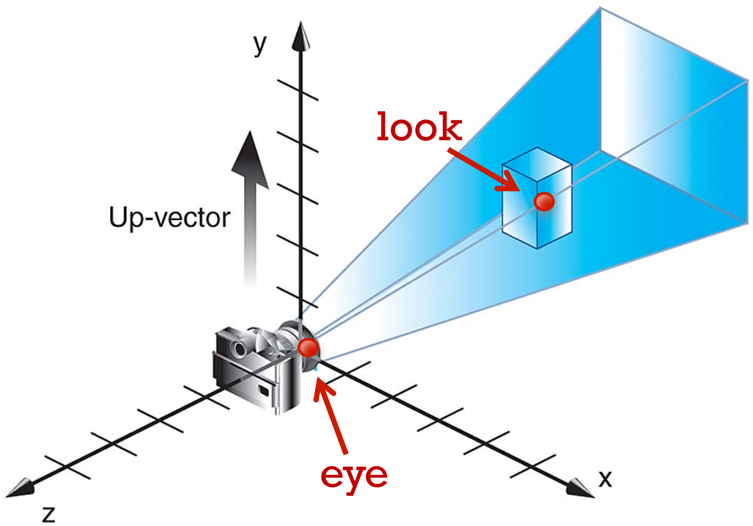


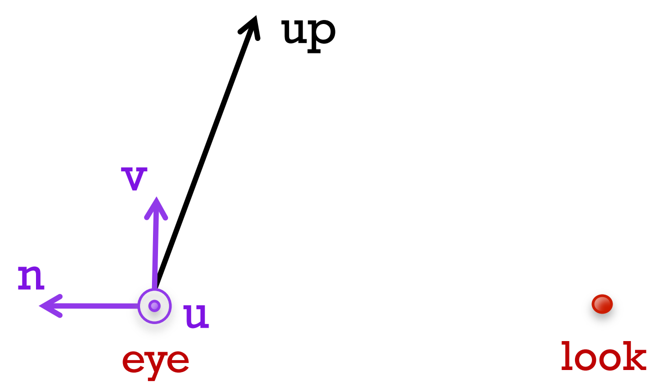


(vector from look to eye)	$n = eye - look$
(normalizing)	$n = n / \lvert n \rvert$
(vector pointing out of the slide)	$u = up \times n$
(normalizing)	$u = u / \lvert n \rvert$
(true upwards direction)	$v = n \times u$
(normalizing)	$v = v / \lvert v \rvert$

Redundant!

Positioning and aiming the camera



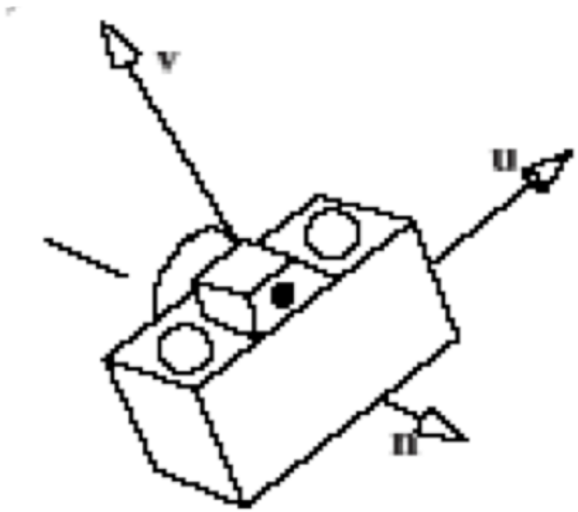


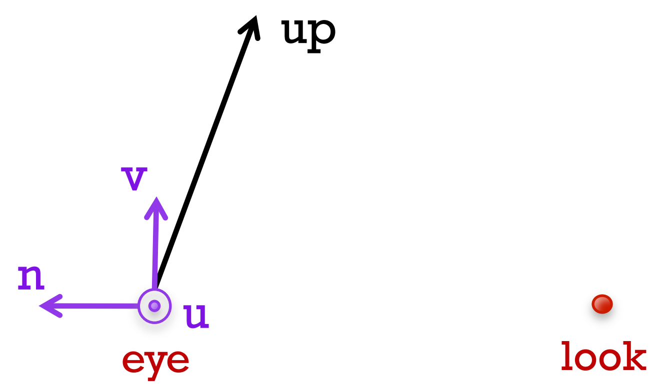
(vector from look to eye)	n = eye – look
(normalizing)	n = n / |n|
(vector pointing out of the slide)	u = up x n
(normalizing)	u = u/ |n|
(true upwards direction)	v = n x u

Result:

a uvn-coordinate system which is attached to the camera.

Positioning and aiming the camera





(vector from look to eye)	n = eye – look
(normalizing)	n = n / |n|
(vector pointing out of the slide)	u = up x n
(normalizing)	u = u/ |n|
(true upwards direction)	v = n x u

Result:

a uvn-coordinate system which is attached to the camera.

The camera

app1.cfg

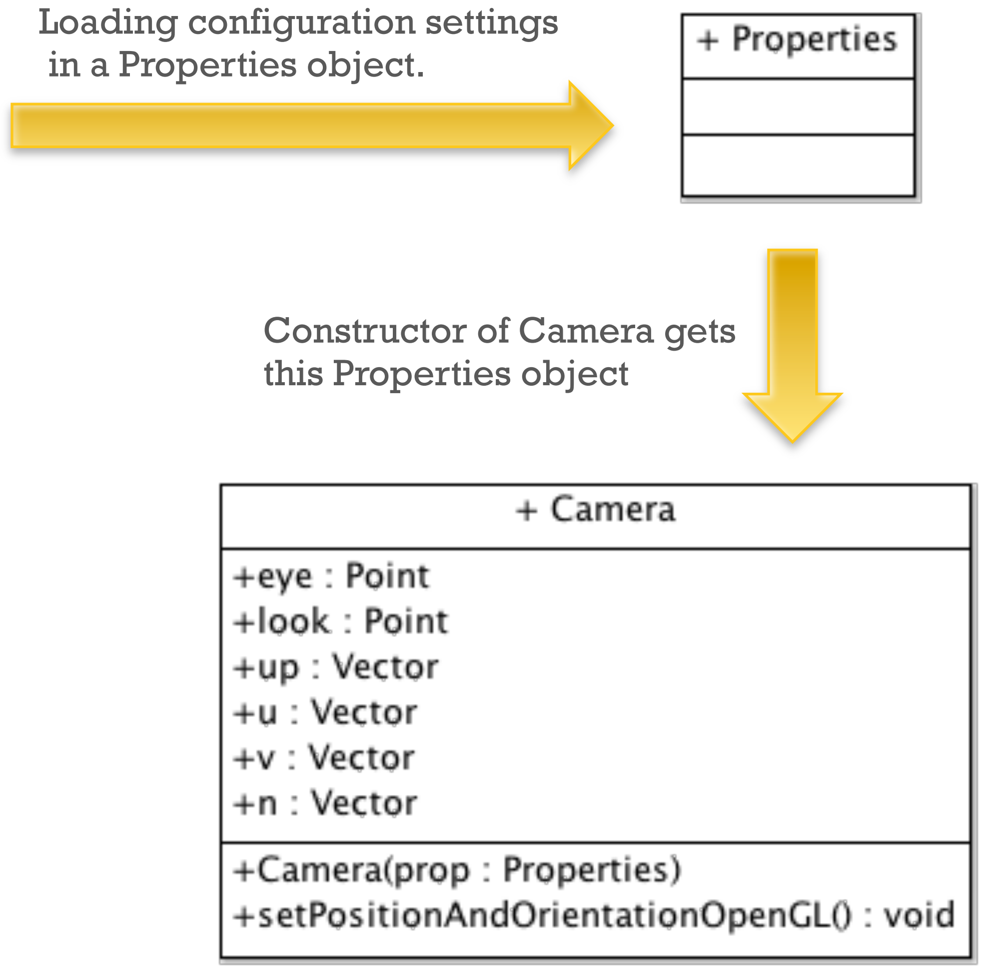
```
scene.file = resources/wineglass.txt

camera.eye.x = 0
camera.eye.y = 0.5
camera.eye.z = 6

camera.look.x = 0
camera.look.y = 0.5
camera.look.z = 0

camera.up.x = 0
camera.up.y = 1
camera.up.z = 0
```

Configuraton file for a particular graphical application



Camera animation

- What happens if the following line of code is executed?

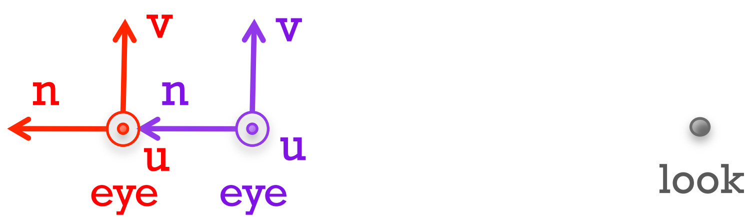
eye = eye + n

The camera moves backward

- Should we change other camera data after moving eye to eye?

no

Moving forward and backward



+ Camera
+eye : Point +look : Point +up : Vector +u : Vector +v : Vector +n : Vector
+Camera(prop : Properties) +setPositionAndOrientationOpenGL() : void +forward() : void +backward() : void

- What happens if the following line of code is executed?

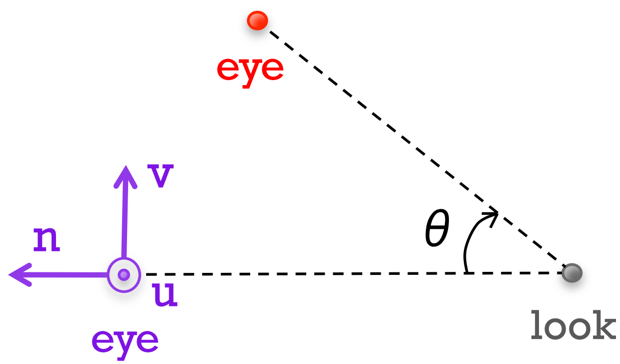
  eye = eye + n — The camera moves backward

- Should we change other camera data after moving eye to eye?

  no

- Which line of code moves the camera forward? eye = eye - n

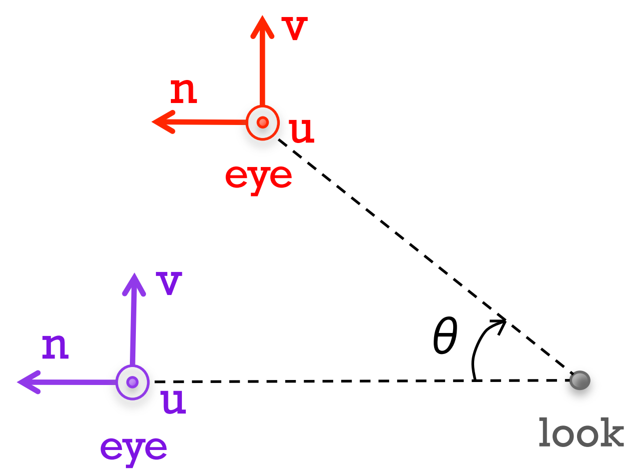
Rotating the camera upwards



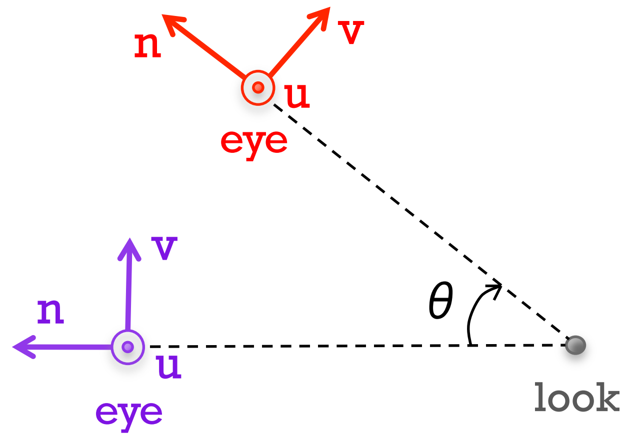
eye = rotate eye by θ around **u** by making use of quaternions

Should we change other camera data after moving eye to eye?

If we only changed eye to eye without changing other camera data, the camera would not face the object anymore. (right figure)



Rotating the camera upwards

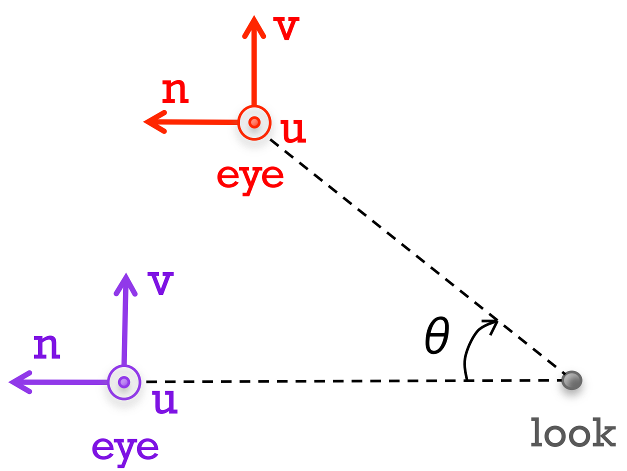


eye = rotate eye by θ around **u**
by making use of quaternions

n = eye – look
n = n / |n|
v = n x u

Should we change other camera data after moving eye to eye?

If we only changed eye to eye without changing other camera data, the camera would not face the object anymore. (right figure)



So yes, v and n should also be updated. (figure above)

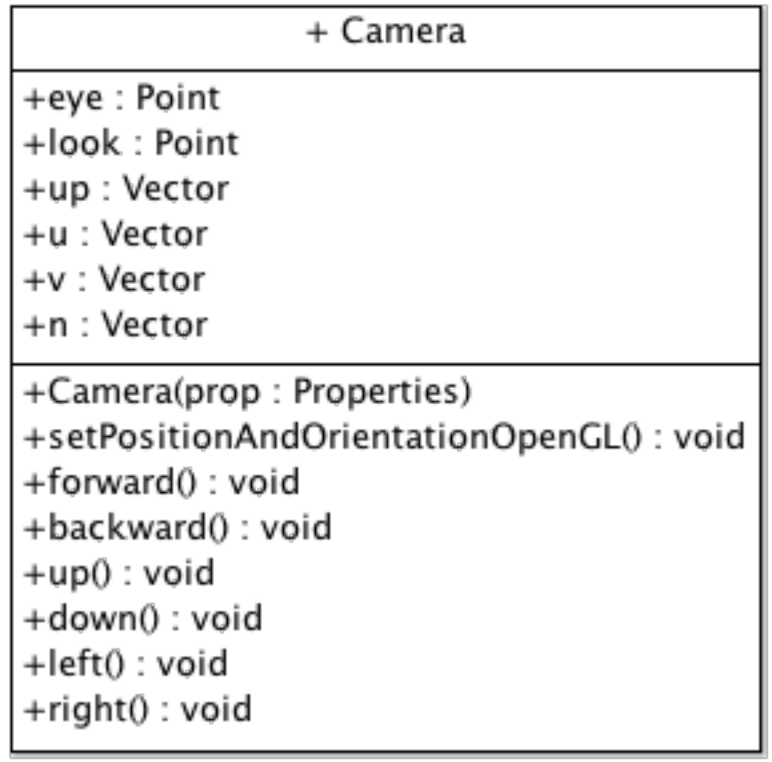
Camera animation

You got all ingredients to make the camera rotate upwards.

Rotating the camera

- downwards,
- to the left and
- to the right

are similar operations which are left for you to figure them out …



Questions?