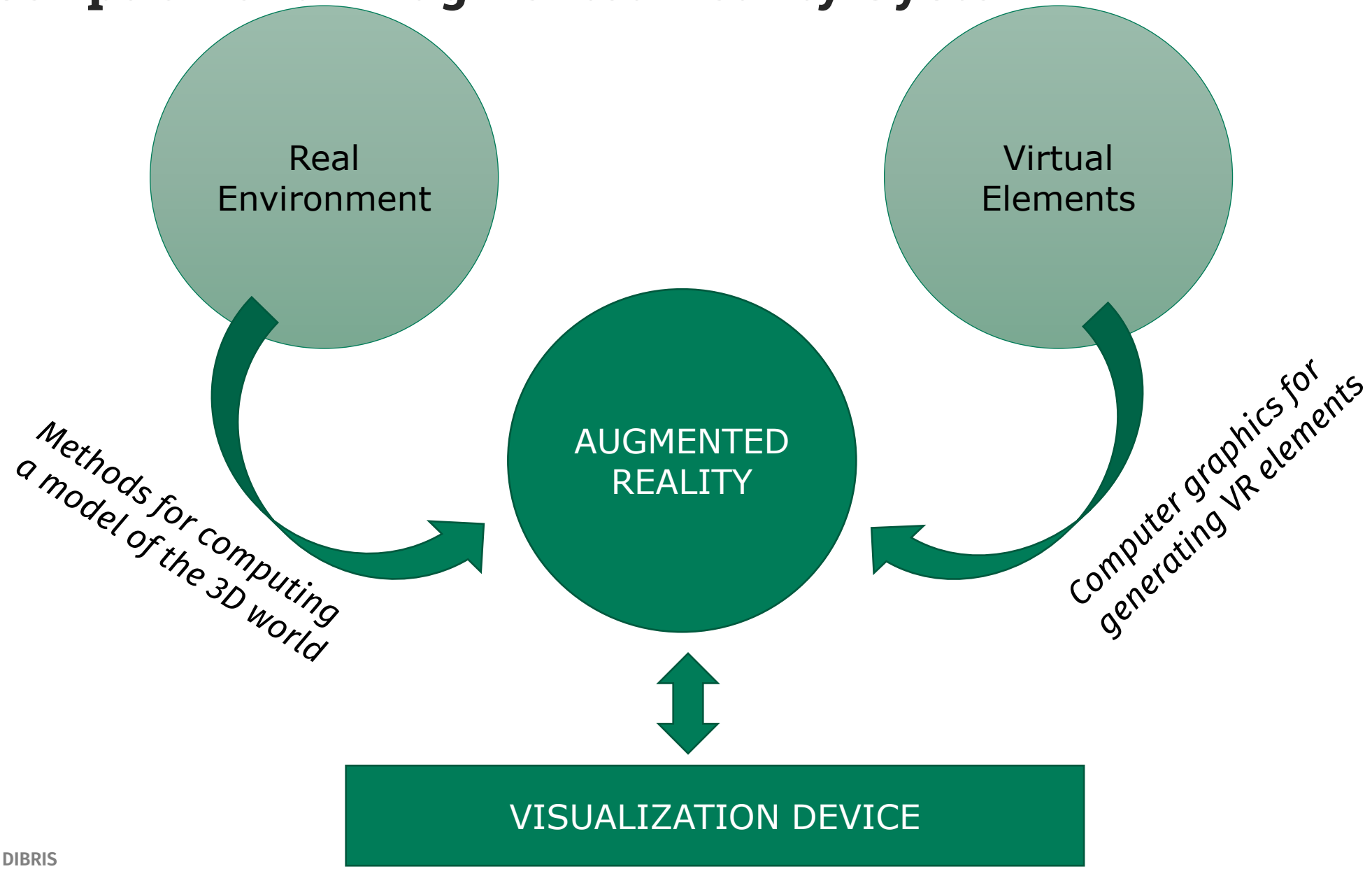
Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Augmented Reality

Lecture 3

Manuela Chessa – manuela.chessa@unige.it
Fabio Solari – fabio.solari@unige.it

Description of an Augmented Reality System



Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Basics of 3D Geometry

The 3D geometry

To describe the geometry of real and virtual worlds, we must define a 3D Euclidian space, with Cartesian coordinates.

Each point P in this space will be described by its Cartesian coordinates:

$$P = (x, y, z)$$

Considering a right-handed coordinate system, whose origin is in (0,0,0), we can express a triangle with the coordinates of its 3 vertices

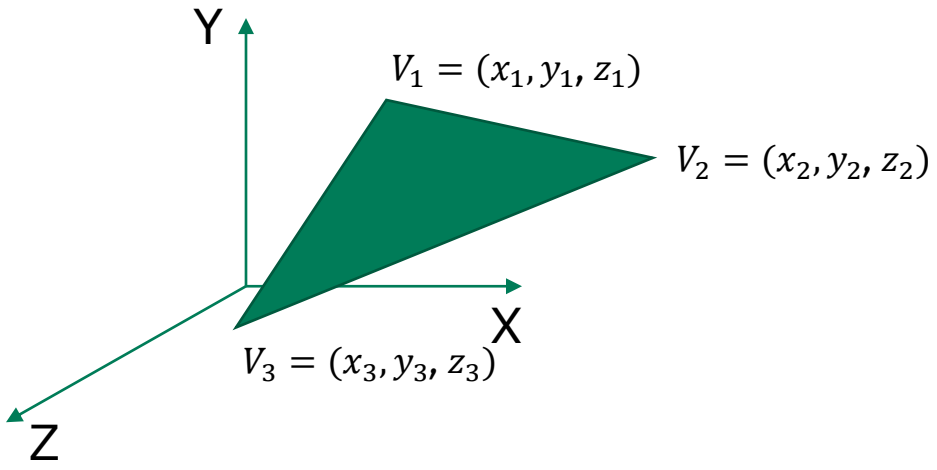
The 3D geometry

The origin of the reference frame is where the X, Y, and Z axis intersect

Figure represents a triangle in a right-handed reference frame

The triangle is defined by 3 vertices

$V_1 = (x_1, y_1, z_1)$
$V_2 = (x_2, y_2, z_2)$
$V_3 = (x_3, y_3, z_3)$



Right-handed vs left-handed reference frame

The right-hand rule (for right-handed coordinates the right thumb points along the z axis in the positive direction and the curl of the fingers represents a motion from the first or x axis to the second or y axis) is in widespread use in physics.

In computer graphics, OpenGL and 3dsMax use a right-handed coordinate system.

Instead, Direct3D, Unity and Unreal use a left-handed coordinate system.

Note that mixing different coordinate system could result in errors and inconsistencies but be prepared to cope with the issue!

Translations and Transformations

A (virtual) 3D object changes its position, orientation and scale in the (virtual) space, through:

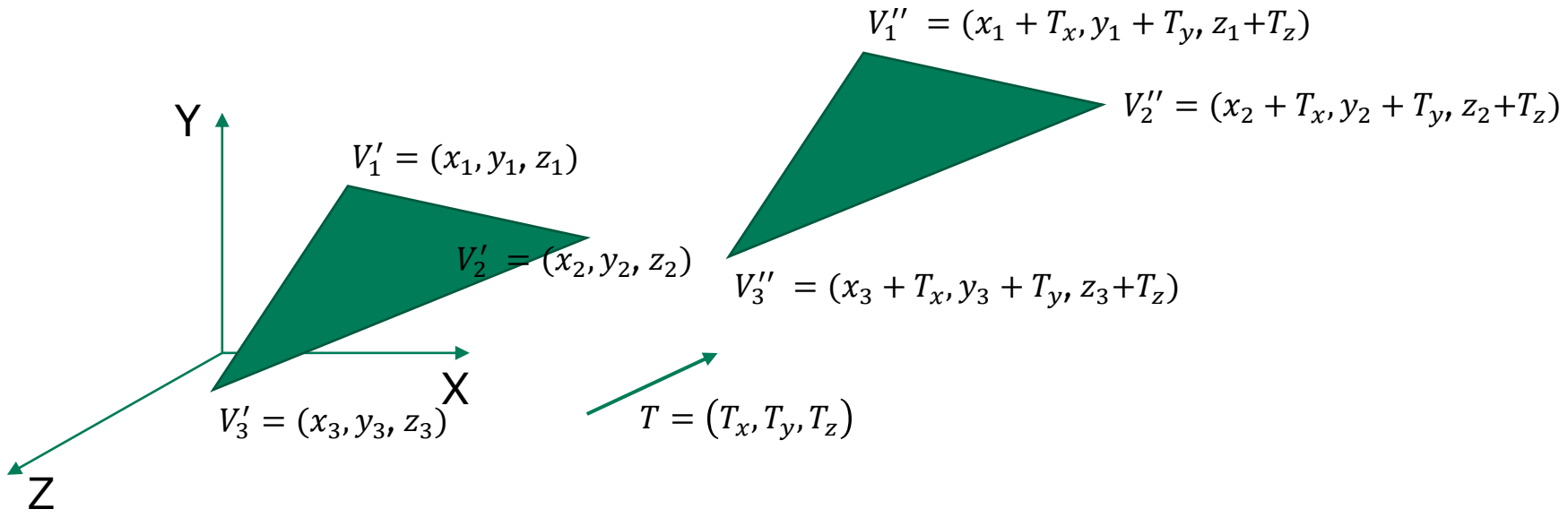
- Translation (or affine transformation): the change of the origin of the object. Affine transformations in 3D cannot be implemented using 3x3 matrices.
- Linear Transformations: straight and parallel lines are preserved and there is no translation (we will define linear transformation better in the next slides)

Translations

A point $P' = (x, y, z)$ translated of the amount $T = (T_x, T_y, T_z)$ moves in the new 3D position

$P'' = (x + T_x, y + T_y, z + T_z)$

Considering a triangle, the same translation is applied to the 3 vertices. Similarly, for a complex 3D shape, translation is applied to all the vertices of the mesh.



Linear Transformations

A linear transformation preserves straight lines and parallel lines. Other properties, such as areas, angles, length of the lines may not be preserved.

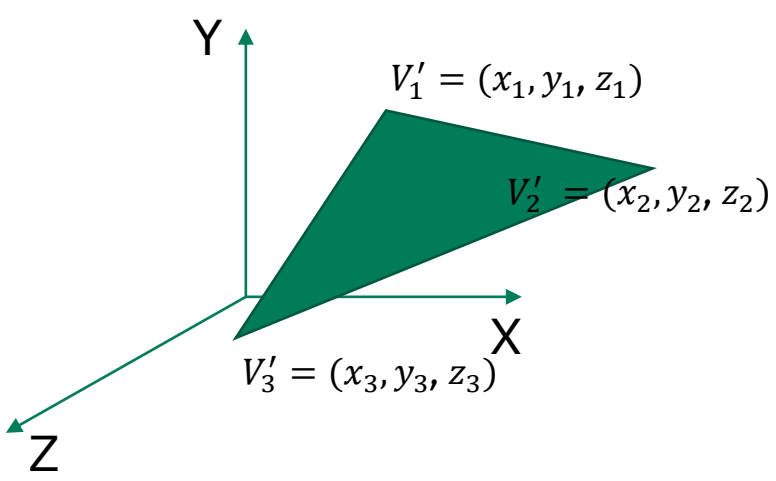
Linear transformation includes:

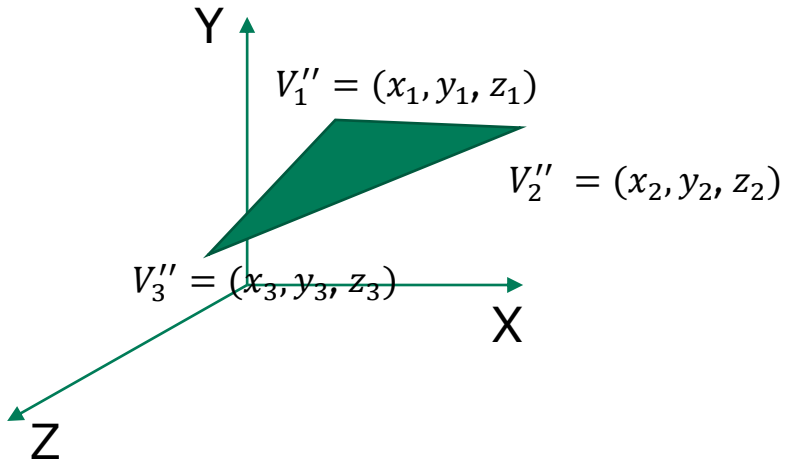
- scaling
- rotation
- orthographic projection
- reflection
- shearing

Linear transformations in 3D can be managed trough 3x3 matrices

Linear transformations - scaling

The simplest scale operation applies a separate scale factor along each cardinal axis. If the scale factors for all axes are equal, then the scale is uniform; otherwise, it is nonuniform.





Linear transformations - scaling

The matrix 3x3 that represents a scaling is:

$$S(k_x, k_y, k_z) = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix}$$

that transforms a point $P' = (x, y, z)$ into a point $P'' = [k_x x, k_y y, k_z z]$ in the following way:

$$[k_x x, k_y y, k_z z] = \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_z \end{bmatrix} [x, y, z]$$

Linear transformations - rotation

Rotation is the operation that changes the orientation of 3D (virtual) objects.

We have different possible rotation representations

The Euler theorem tells us that we need at least three numbers to represent an arbitrary rotation in the 3D space.

Linear transformations - rotation

Some three-number representations:

- ZYZ Euler angles
- ZYX Euler angles (roll, pitch, yaw)
- Axis angle

One four-number representation:

- quaternions

Linear transformations – rotation with Euler angles

Rotation in 3D are not trivial, for this reason we will start considering the 2D case, and we will extend to the 3D case later.

2D rotations

Let's consider a point $p = (x, y)$ in the 2D Euclidean space.

Translations are as we have seen in the more general 3D case; linear transformations are expressed through 2x2 matrices M.

We must ensure that the model does not become distorted. This is achieved by ensuring that M satisfies the following rules:

1. No stretching of axes.
2. No shearing.
3. No mirror images.

2D rotations

Given the linear transformation matrix $M = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$, the previous rules are satisfied if:

1. the columns of M must have unit length $m_{11}^2 + m_{21}^2 = 1$ and $m_{12}^2 + m_{22}^2 = 1$
2. the coordinate axes must remain perpendicular (Otherwise, shearing occurs) $m_{11}m_{12} + m_{21}m_{22} = 0$
3. the determinant of M is positive. After satisfying the first two rules, the only possible remaining determinants are 1 (the normal case) and –1 (the mirror-image case). Thus, the rule implies that:

$$\det(M) = m_{11}m_{22} - m_{12}m_{21} = 1$$

2D rotations

The full range of rotations in 2D is obtained defining the following matrix

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

with θ ranging from 0 to 2π.

A point $p = (x, y)$ is then transformed into a point $p' = (x\cos\theta - y\sin\theta\,, x\sin\theta + y\cos\theta)$

3D rotations with Euler angles

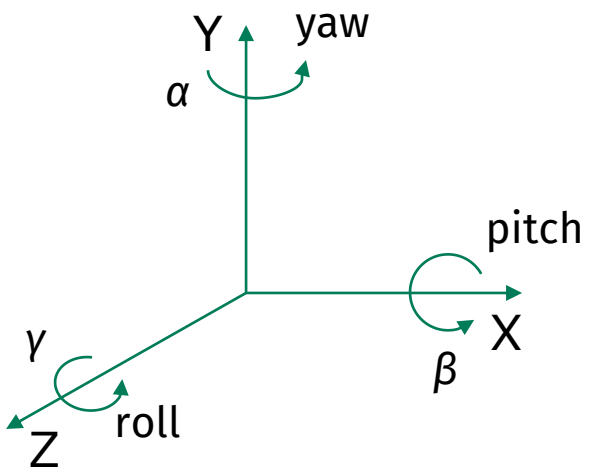
We can easily extend the matrix M to the 3D case:

$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

but we must consider the 3 constraints (slides 15 and 16) to have a valid rotation matrix.

3D rotations with Euler angles

One of the simplest ways to parameterize 3D rotations is to define them as 2D-like transformations, in terms of yaw, pitch and roll



3D rotations – pitch around X axis

The pitch, i.e., the counter-clockwise rotation around the X axis, is the defined by:

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

3D rotations – yaw around Y axis

The yaw, i.e., the counter-clockwise rotation around the Y axis, is the defined by:

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

3D rotations – roll around Z axis

The roll, i.e., the counter-clockwise rotation around the Y axis, is the defined by:

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3D rotations with Euler angles

Rotations around the axis can be concatenated, through matrix multiplication.

E.g.: a pitch of β followed by a roll of γ and then by a yaw of α can be obtained as follows

$$R = R_y(\alpha)R_z(\gamma)R_x(\beta)$$

Note that the operations are not commutative, so the order is important.

Note also that multiplications are backwards. This is motivated by the fact that multiplications are applied on the left.

3D rotations with Euler angles- exercise

Compute the multiplication to show that 3D rotations are not commutative.

Example, show that a pitch of β followed by a roll of γ and then by a yaw of α is different from a roll of γ, followed by a yaw of α and then by a pitch of β

Homogeneous coordinates

Following the previous definitions of translation and rotation, given a 3D point $P' = (x, y, z)$, the point P'' obtained applying a rotation R followed by a translation T will be obtained with

$$P'' = RP' + T$$

It would be convenient to apply both rotation and translation together in a single operation. Although there is no way to form a single 3 by 3 matrix to accomplish both operations, it can be done by increasing the matrix dimensions by one.

Homogeneous coordinates

We can define the homogeneous transformation matrix 4x4 T in the following way:

$$T = \begin{bmatrix} & R & & T \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where R is the 3x3 matrix (like in slide 20, 21, 22) and T is the 3x1 translation (slide 8)

Being T a 4x4 matrix, we must add a dimension to the 3D coordinates.

$P = (x, y, z, w)$ where w is 0 for vectors, 1 for points

So a point will be expressed in the following way:

$$P = (x, y, z, 1)$$

Homogeneous coordinates

With homogeneous transformations and the transformation matrix 4x4, transformations can be combined through matrix multiplication

To invert a transformation T we must compute the inverse T^{-1}

Note that TT^{-1} is the identity matrix

3D rotations – some considerations

Rotations are expressed by a matrix 3x3, so 9 entries and only 3 independent variables.

There is no simple parameterization

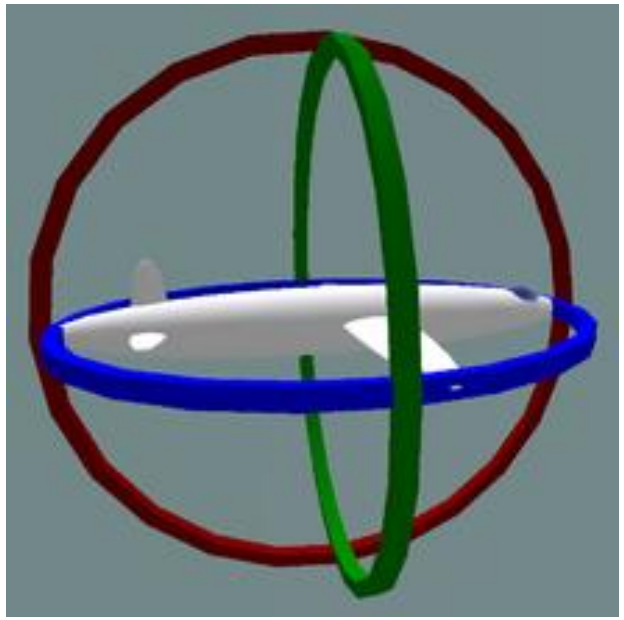
As we have seen, the axis of rotation changes (i.e., applying the rotations on different orders gives different results)

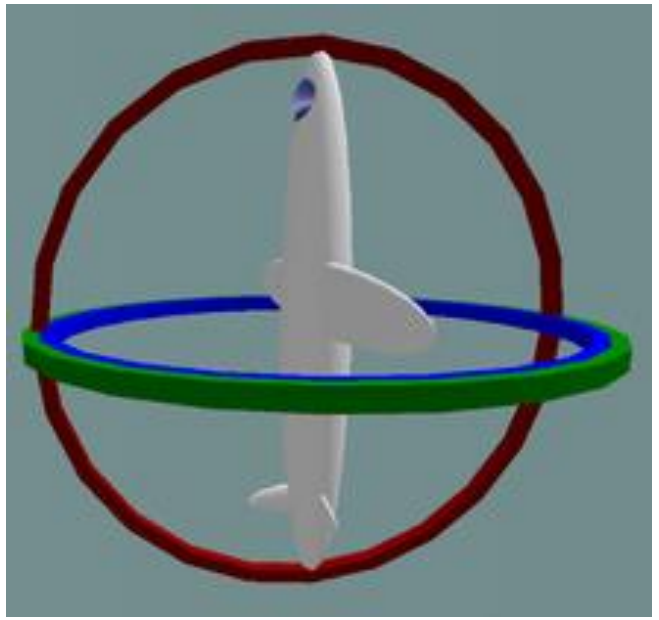
The operations are not commutative.

3D rotations with Euler angles - gimbal lock

Gimbal lock is the loss of one degree of freedom in a three-dimensional, three-gimbal mechanism that occurs when the axes of two of the three gimbals are driven into a parallel configuration, "locking" the system into rotation in a degenerate two-dimensional space.

3D rotations with Euler angles - gimbal lock





Here we have lost one degree of freedom

3D rotations with Euler angles - gimbal lock -exercise

Compute $R = R_x(\beta)\ R_y(\alpha) R_z(\gamma)$ **when** $\alpha = \frac{\pi}{2}$

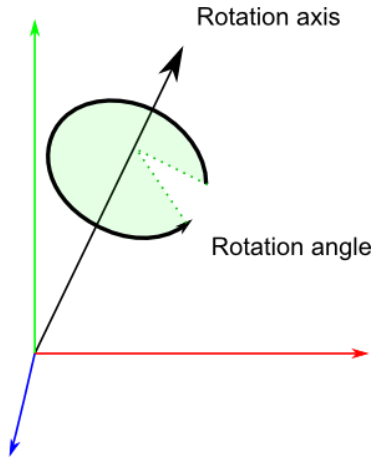
3D rotations – axis-angle representations and quaternions

Given the previous considerations, we need a better way to parameterize rotations.

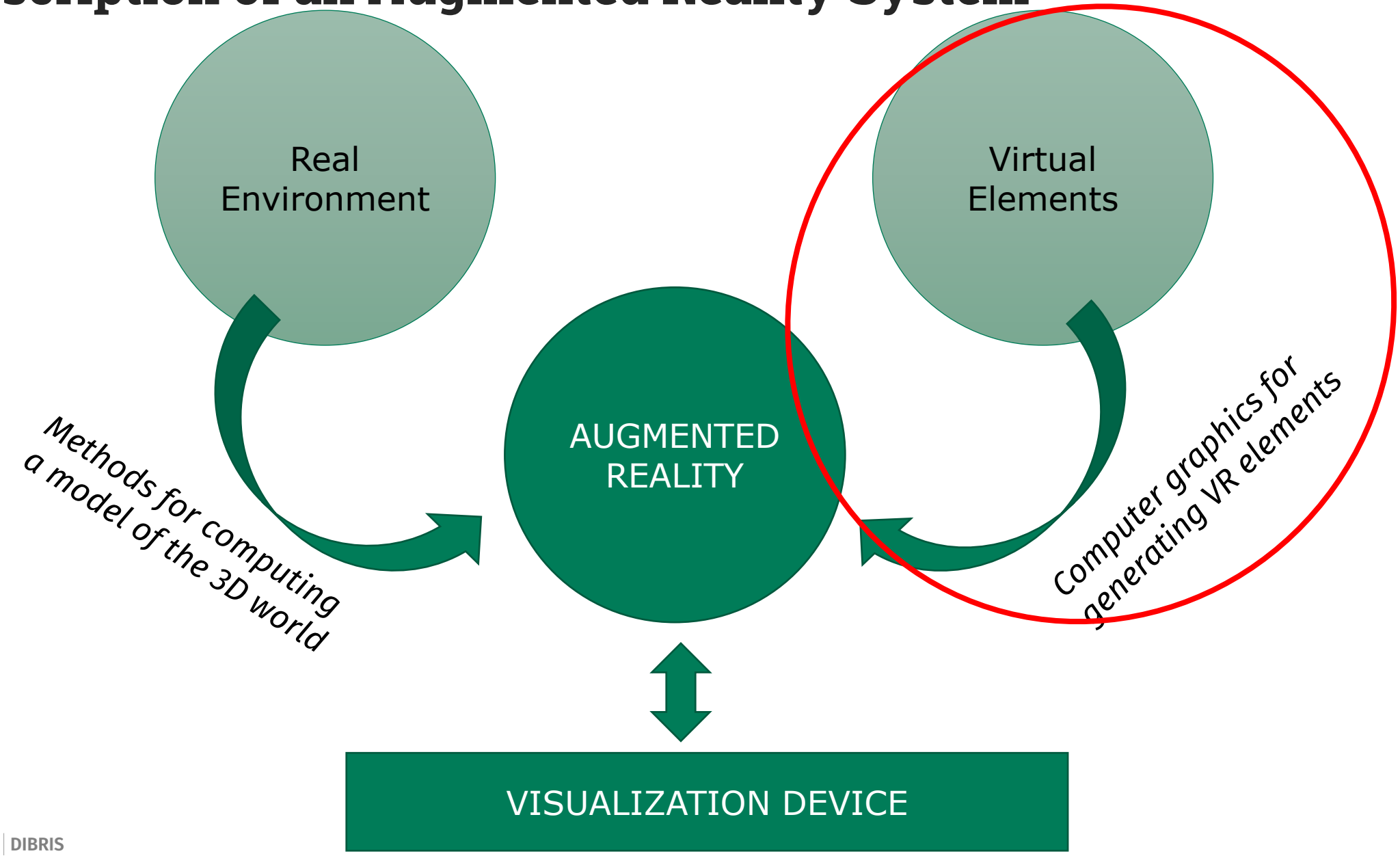
A quaternion is a set of 4 numbers which represents rotation in the following way

A spatial rotation around a fixed point of θ radians around an axis (X, Y, Z) is denoted by the quaternion (C, XS, YS, ZS) where

$C = cos\left(\frac{\theta}{2}\right)$ and $S = sin\left(\frac{\theta}{2}\right)$



Description of an Augmented Reality System



The visualization pipeline (basics)

The visualization pipeline

When defining a (VR) environment, we take into consideration several reference frames:

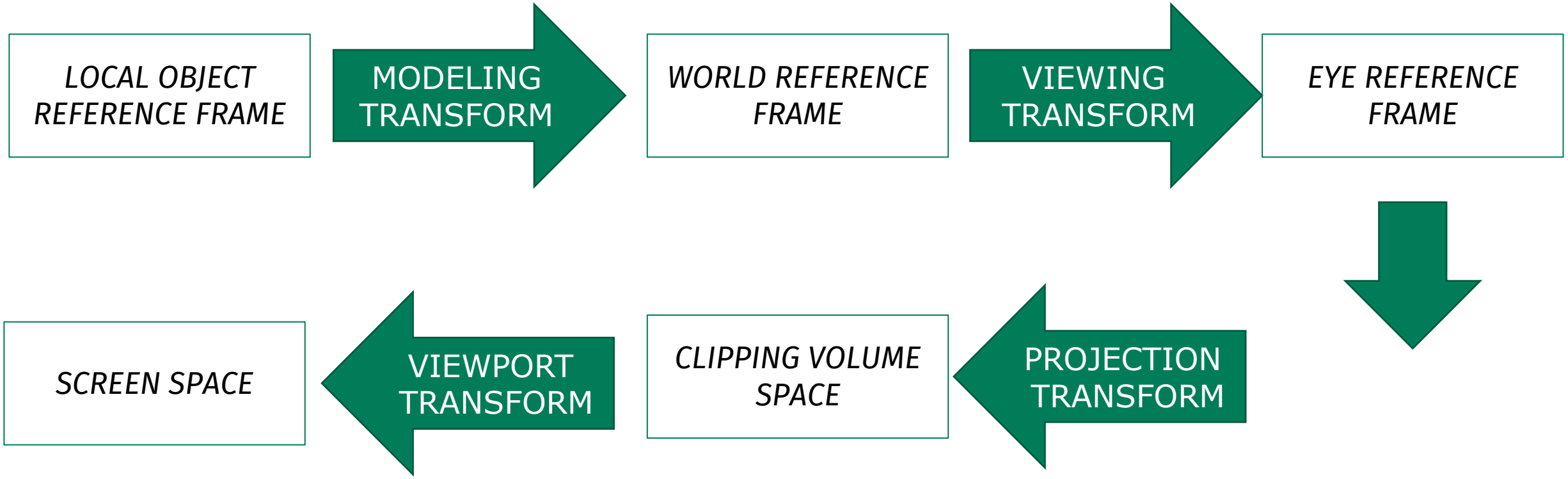
- The 3D local ones of the (virtual) objects in the scene
- The 3D global one of the world
- The 3D ones of the observation points – e.g., the human eyes or the (virtual) cameras
- The 3D one of visualization device – e.g., the display
- The 2D one of the window

The visualization pipeline

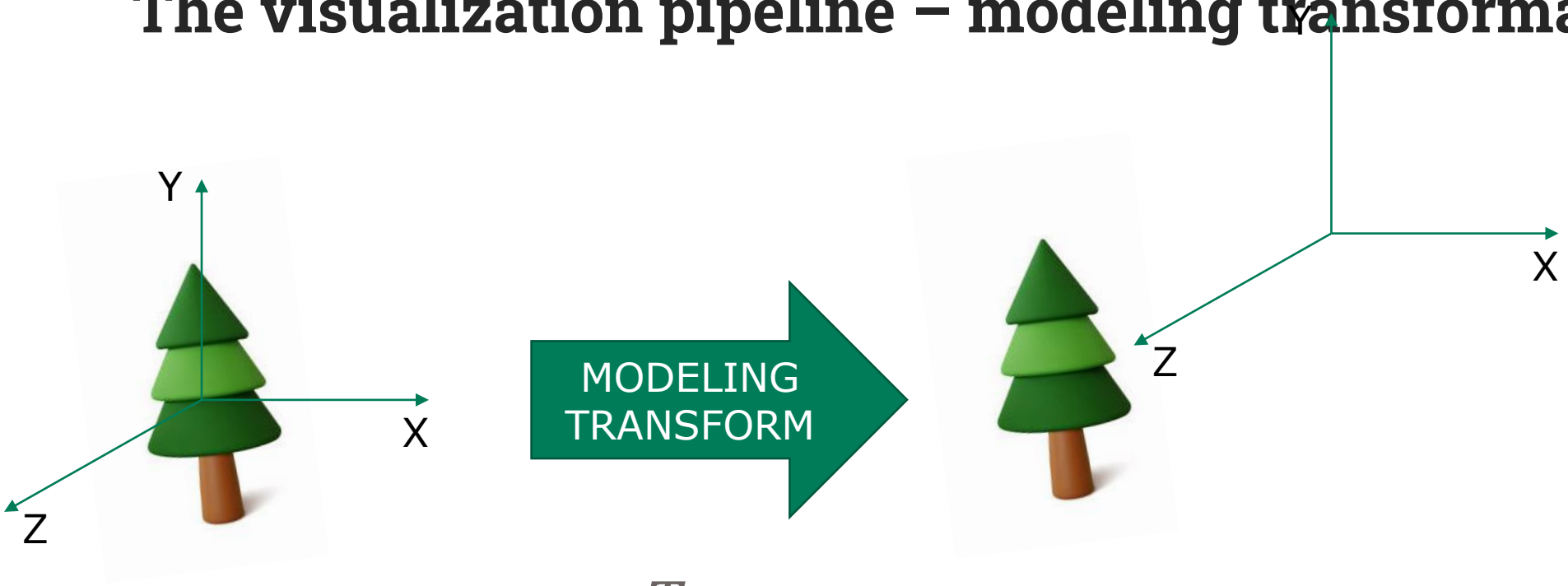
All the reference frames must be linked one with respect to the others.

The visualization pipeline describes all the operations necessary to represent each (virtual) object in the display reference frame, and ultimately in the 2D window reference frame.

The visualization pipeline



The visualization pipeline – modeling transformation



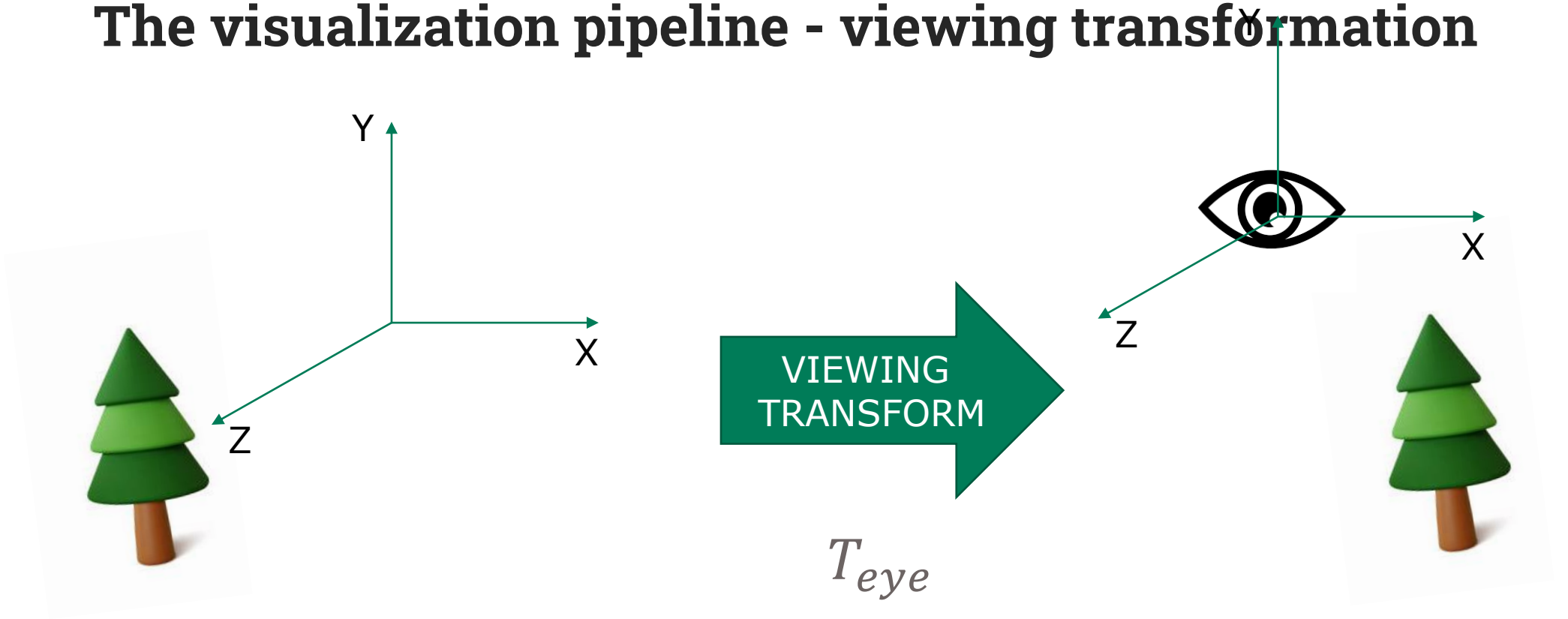
T_{rb}

LOCAL OBJECT REFERENCE FRAME

WORLD REFERENCE FRAME

All the objects are described in a common world reference frame, by applying to each vertex the transformation described by 4x4 matrix T_{rb}

The visualization pipeline - viewing transformation

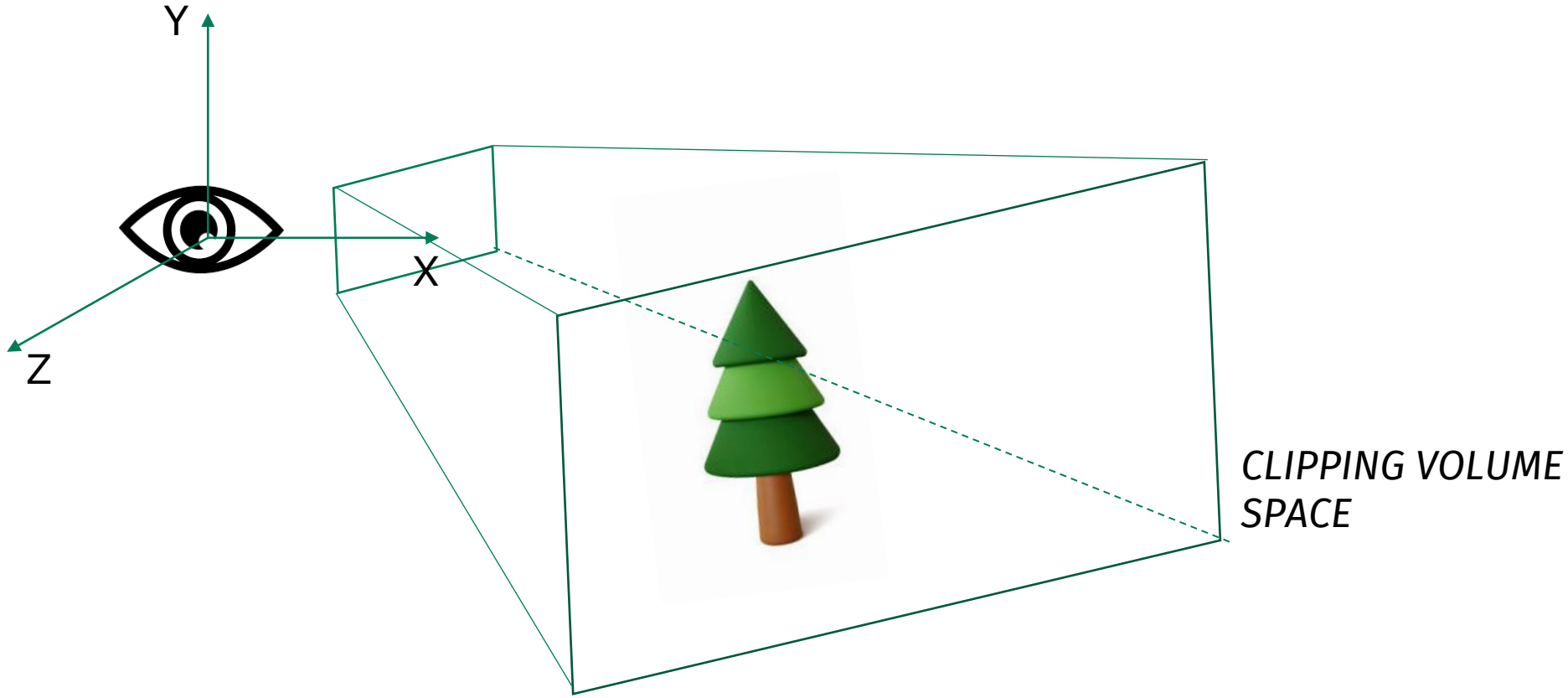


WORLD REFERENCE FRAME

EYE REFERENCE FRAME

All the objects are described in the eye (camera) reference frame, by applying to each vertex the transformation described by 4x4 matrix T_{eye}

The visualization pipeline – projection transform



All the objects are described in the portion of space called projection space or viewing volume. Vertex are still 3D points, but transformed according to the kind of projection

Projection transform – perspective projection

The viewing volume is a frustum (i.e., a truncated pyramid)

This kind of projection is similar to what happens in our eyes and in standard cameras.

An important attribute of the perspective projection, in contrast to the parallel/orthographic projection, is that objects at a larger distance to the viewer or camera are displayed smaller

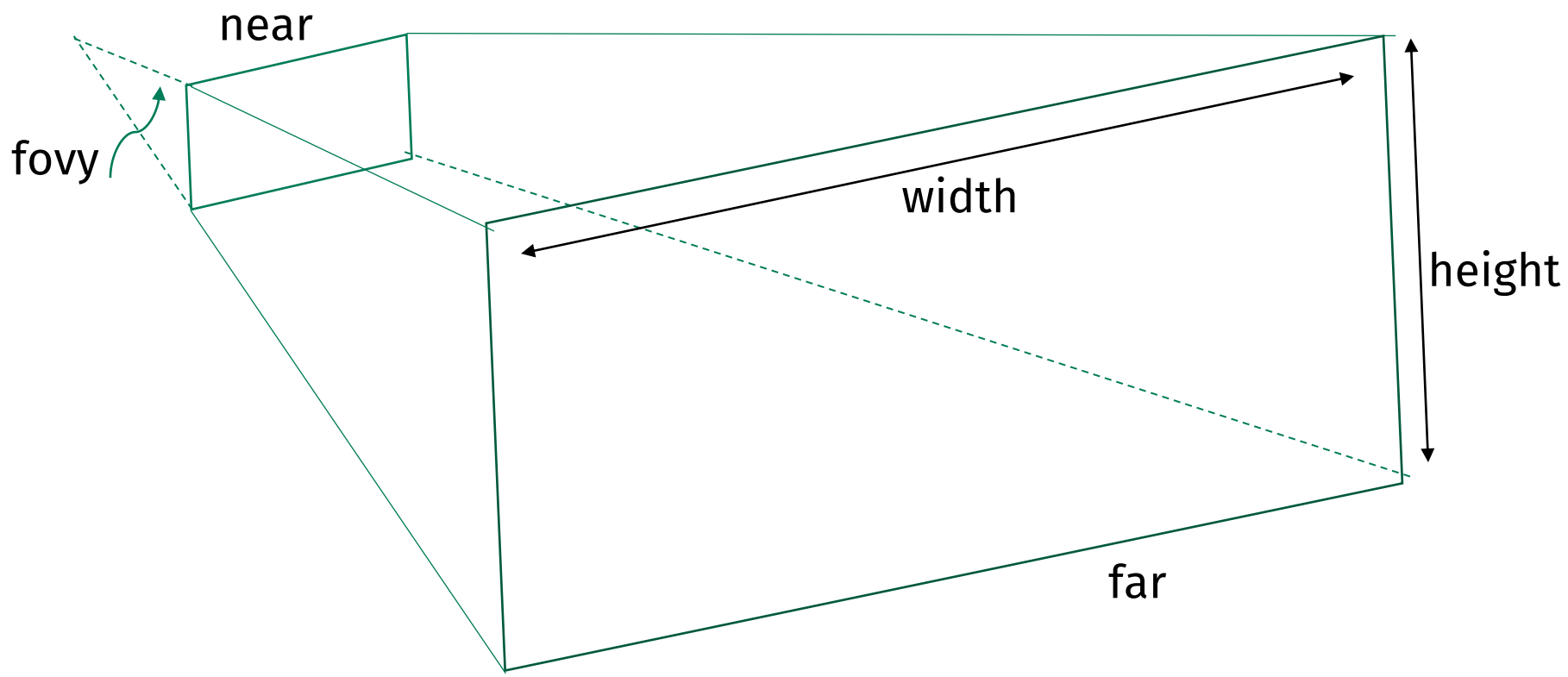
The simplest perspective projection is a mapping that use a pinhole camera model

Projection transform – perspective projection

The frustum of the perspective projection is defined by:

- The near clipping plane
- The far clipping plane
- The vertical field of view (the angle between the upper and lower sides of the viewing frustum)
- The aspect ratio (the aspect ratio of the viewing window - width/height)

Projection transform – perspective projection



Perspective Projection in OpenGL

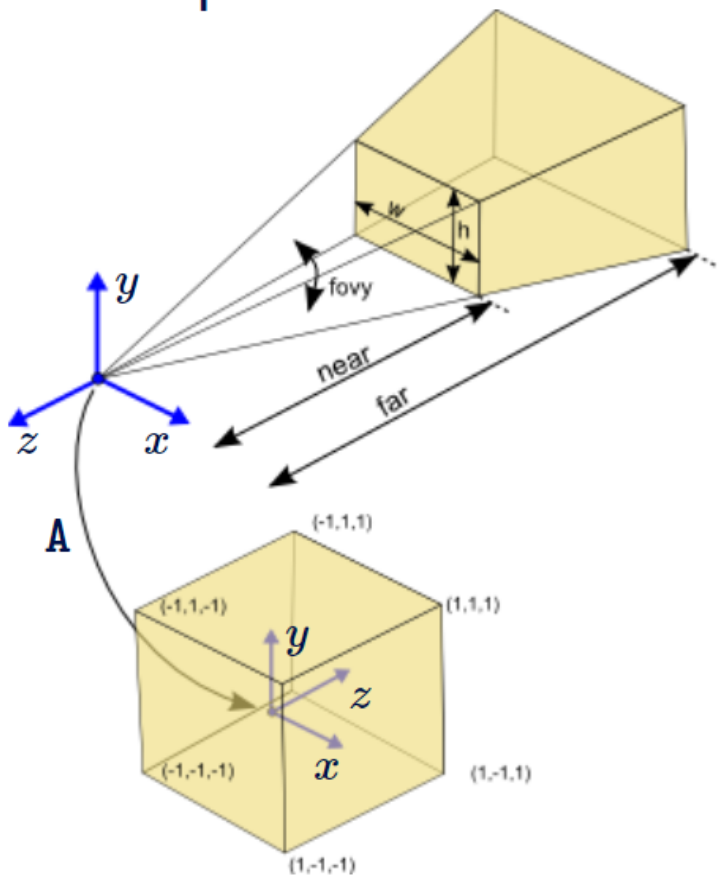
Creating a perspective projection matrix in OpenGL:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fovy, aspect, near, far);
```

$$\mathbf{A} = \begin{bmatrix} \frac{f}{\text{aspect}} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{\text{far}+\text{near}}{\text{near}-\text{far}} & \frac{2*\text{far}*\text{near}}{\text{near}-\text{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

with $f = \text{cotan}(0.5 * \text{fovy})$

and $\text{aspect} = \text{w}/\text{h}$



Projection transform – perspective projection

More at

https://www.mathematik.uni-marburg.de/~thormae/lectures/graphics1/graphics_6_1_eng_web.html#1

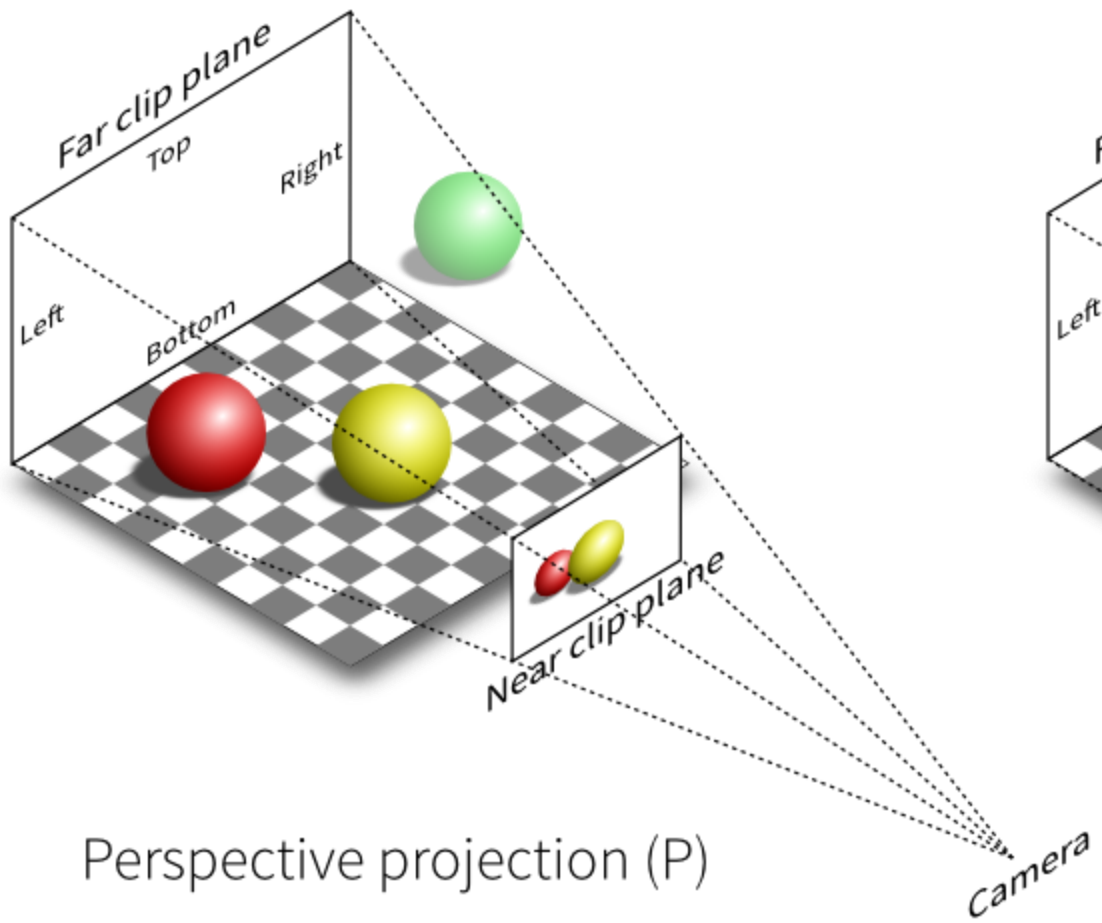
Projection transform – orthographic projection

The viewing volume is a parallelepiped

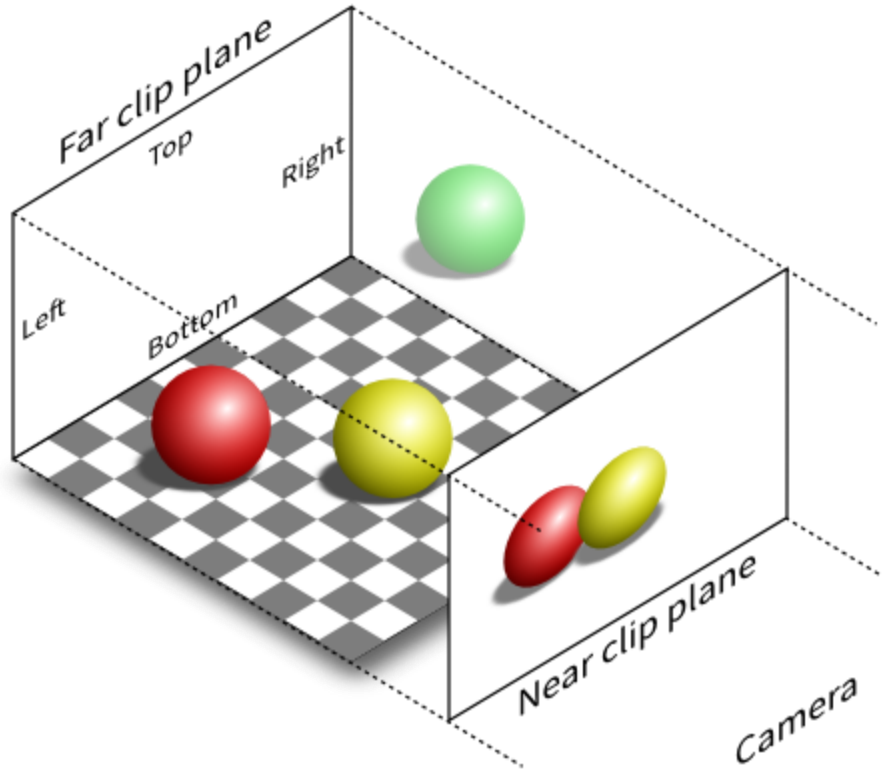
This kind of projection is used in architecture

An important attribute of the parallel/orthographic projection, is that objects' dimension is not affected by distance

Projection transform

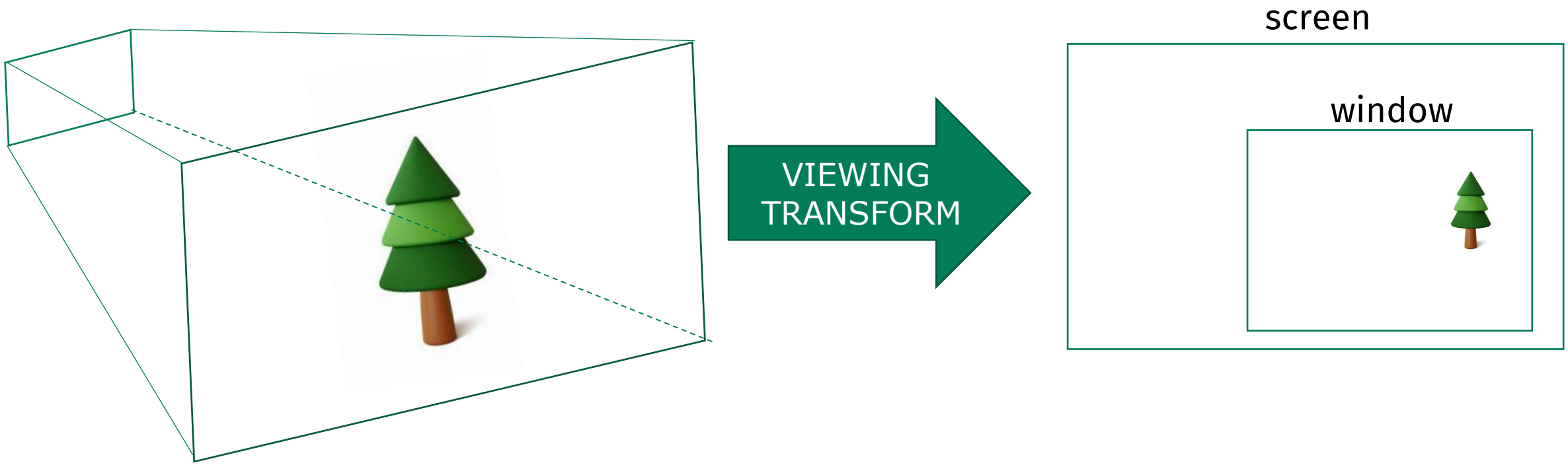


Perspective projection (P)



Orthographic projection (O)

The visualization pipeline – viewport transform



The viewport transform define the viewport, i.e. the portion of the screen where 2D (after projection!) images are shown

UniGe

DIBRIS