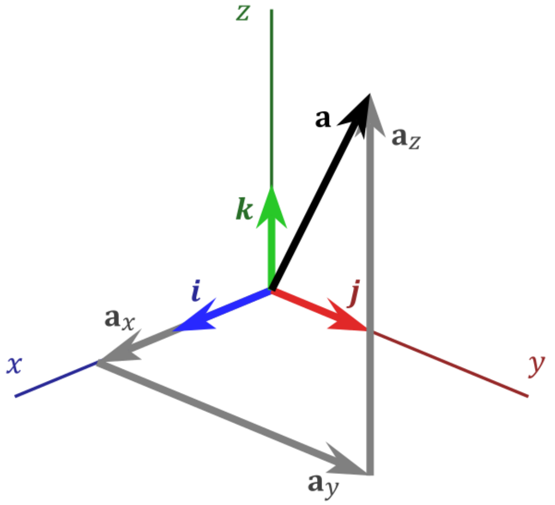
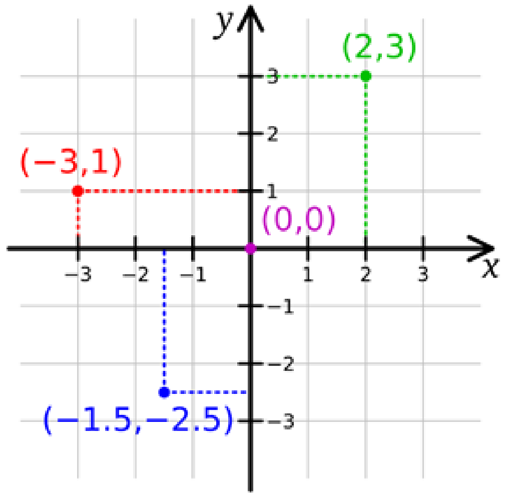
Computational geometry
III – Geometric linear algebra

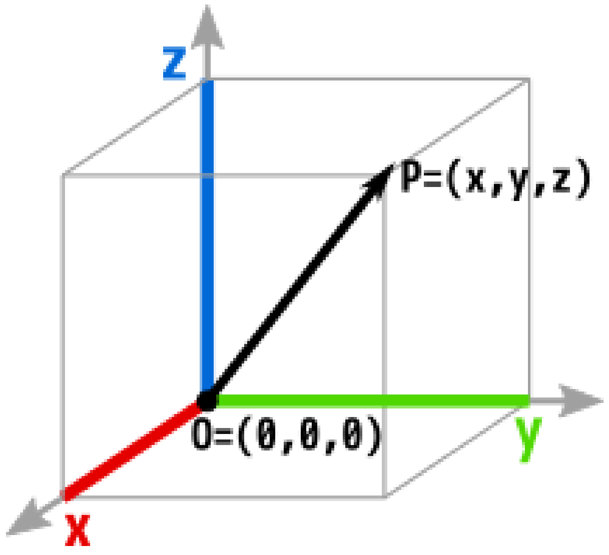


Stefan BORNHOFEN

Cartesian Coordinate System

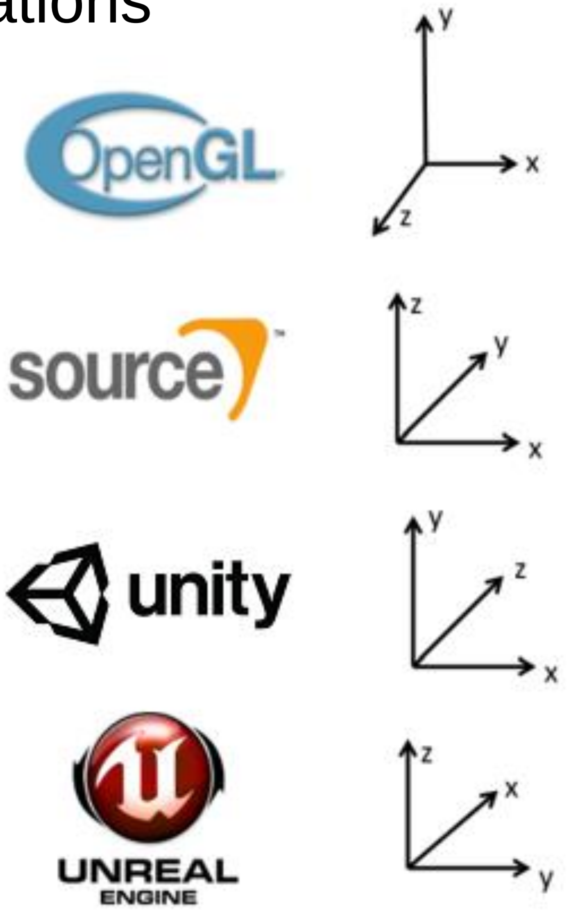
- Each point is uniquely defined by a set of numerical values
- One single origin or center = 0
- Pairwise pependicular axes going through the center





Cartesian Coordinate system

- There is no standard for the axis orientations
- Many different concepts are in use
- In this course, we adopt
  the OpenGL coordinate system:
  - X axis parallel to the « ground »
  - Y axis pointing « up »
  - Z axis pointing « toward us »

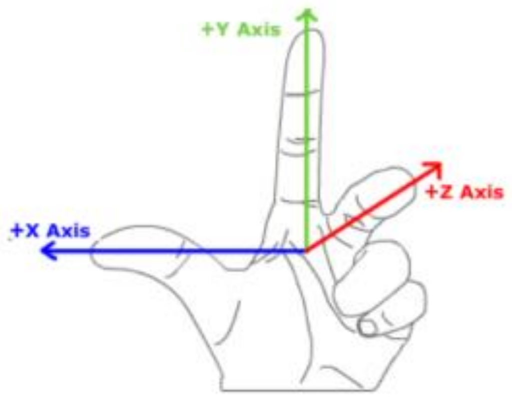


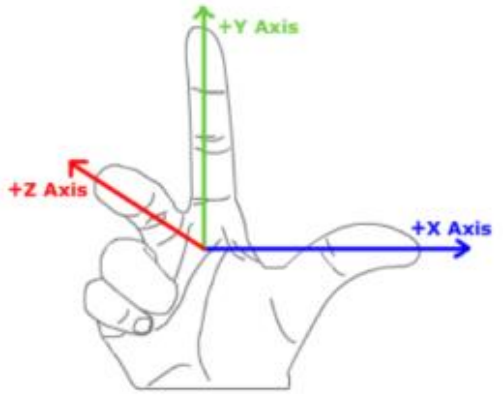
Cartesian Coordinate system

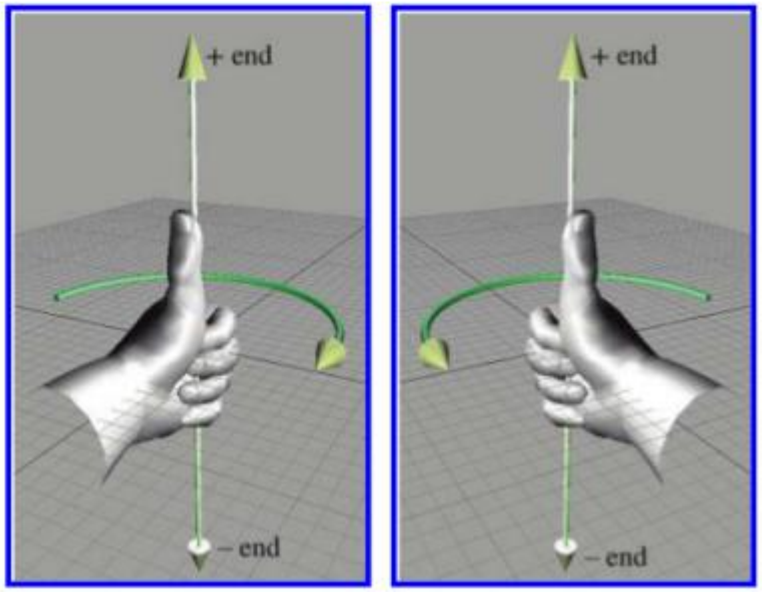
Coordinate systems can be **left-handed** or **right-handed**.

Left handed: positive rotations are clockwise
Right handed: positive rotations are counter-clockwise

Interchanging the labels of any two axes reverses the handedness.
Reversing the direction of one axis (or of all three axes) also reverses the handedness.







Vector

- A vector is an element of a vector space.
- A vector contains the same number of elements than the vector space has dimensions.

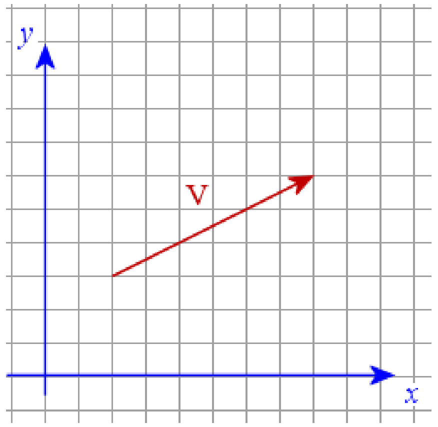
$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \qquad \begin{aligned} a_1 &= a_x = 1 \\ a_2 &= a_y = 2 \end{aligned}$$

$$\mathbf{b} = \begin{bmatrix} 3 \\ 4 \\ 5 \end{bmatrix} \qquad \begin{aligned} b_1 &= b_x = 3 \\ b_2 &= b_y = 4 \\ b_3 &= b_z = 5 \end{aligned}$$

$$\mathbf{c} = \begin{bmatrix} 6 \\ 7 \\ 8 \\ 9 \end{bmatrix} \qquad \begin{aligned} c_1 &= c_x = 6 \\ c_2 &= c_y = 7 \\ c_3 &= c_z = 8 \\ c_4 &= c_w = 9 \end{aligned}$$

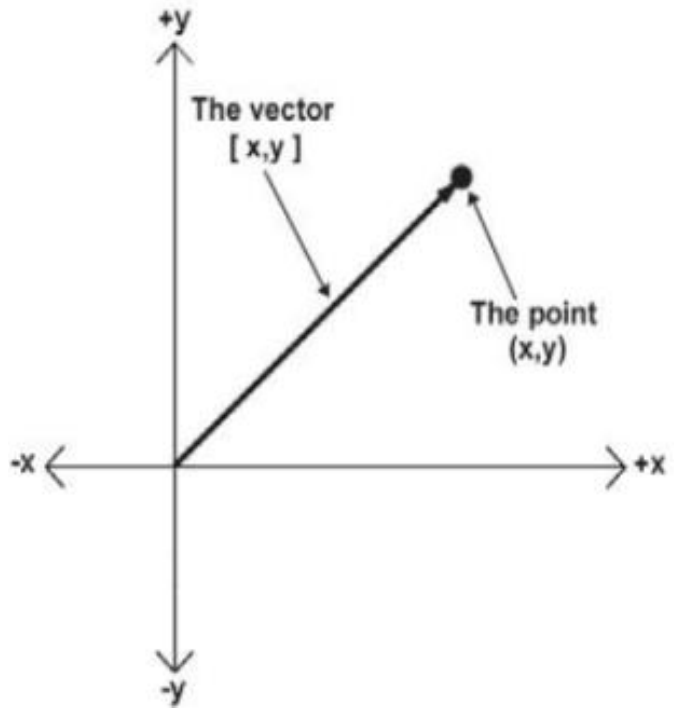
A vector represents a displacement.

A vector has a direction, and can be depicted by an arrow.



Point vs. Vector

- A point is a **position** in space
- A vector is a **displacement** not necessarily from the origin



Don’t miss the subtle difference.

Vector operations

- Magnitude
- Normalization
- Negation
- Scalar Multiplication
- Addition/Substraction
- Dot product
- Cross product

Vector operations

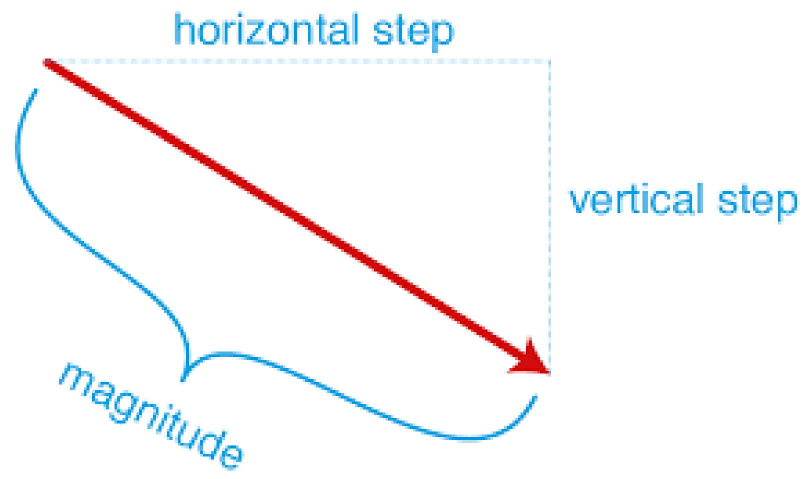
Magnitude

$$2\text{D: } |\mathbf{v}| = \sqrt{x^2 + y^2}$$
$$3\text{D: } |\mathbf{v}| = \sqrt{x^2 + y^2 + z^2}$$

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} {v_i}^2} = \sqrt{{v_1}^2 + {v_2}^2 + \cdots + {v_{n-1}}^2 + {v_n}^2}$$

Geometric interpretation

The “air-line distance” of the displacement.



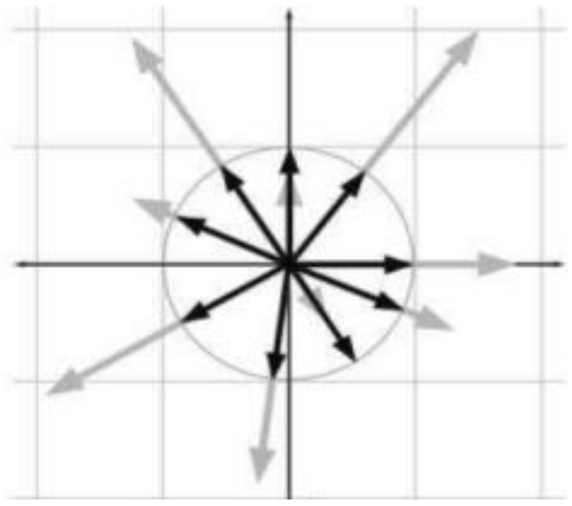
Vector operations

Normalization

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|}$$

Geometric interpretation

- Sometimes we are only interested in the direction
- You obtain a unit vector (magnitude = 1)
- All unit vectors starting from the origin touch the surface of the unit sphere (3D).



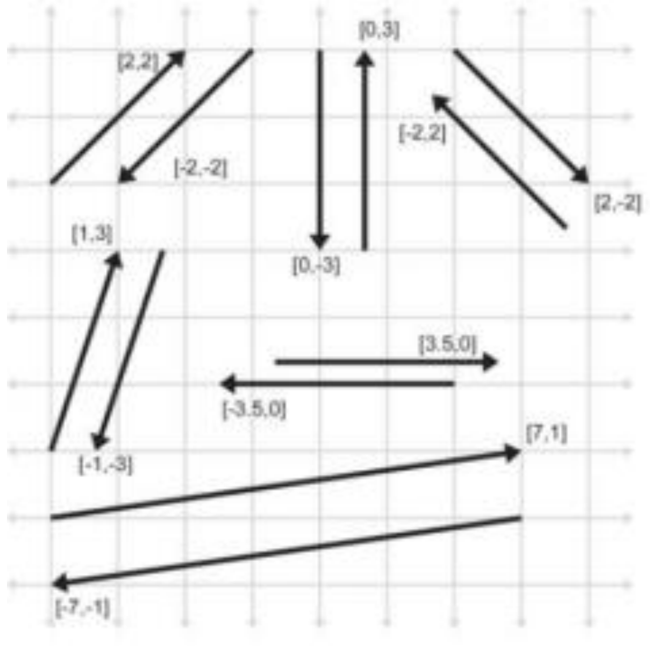
Vector operations

Negation

$$-\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} -a_1 \\ -a_2 \\ \vdots \\ -a_{n-1} \\ -a_n \end{bmatrix}$$

Geometric interpretation

Negating a vector gives a vector with the same magnitude but the opposite direction.



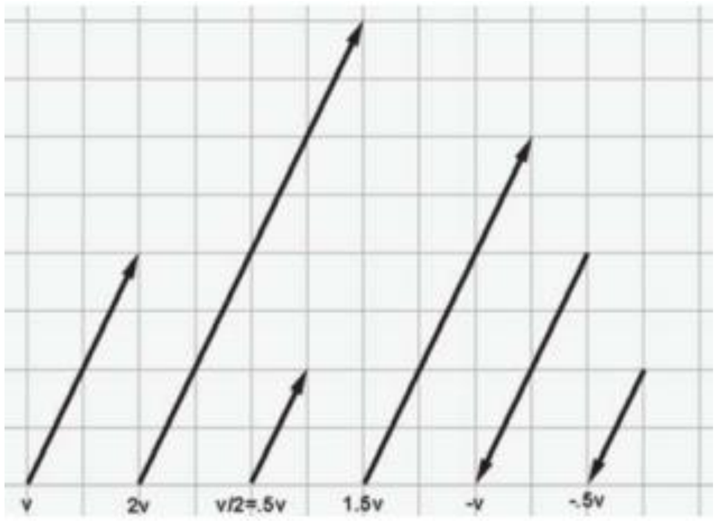
Vector operations

Scalar multiplication

$$k\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} k = \begin{bmatrix} ka_1 \\ ka_2 \\ \vdots \\ ka_{n-1} \\ ka_n \end{bmatrix}$$

Geometric interpretation

You get a parallel vector with a different length, and the same or the opposite direction.



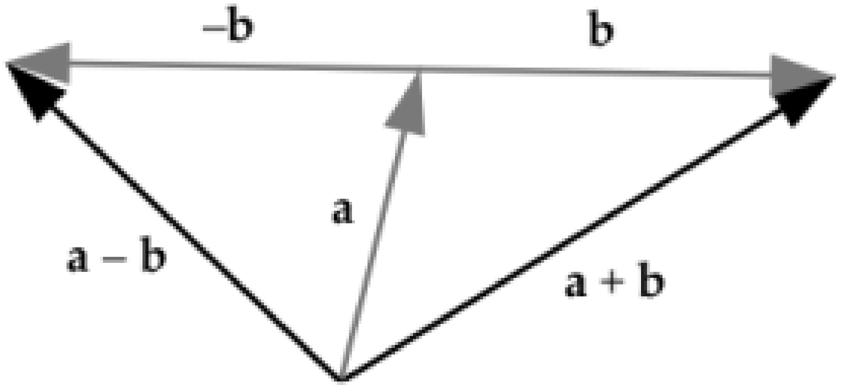
Vector operations

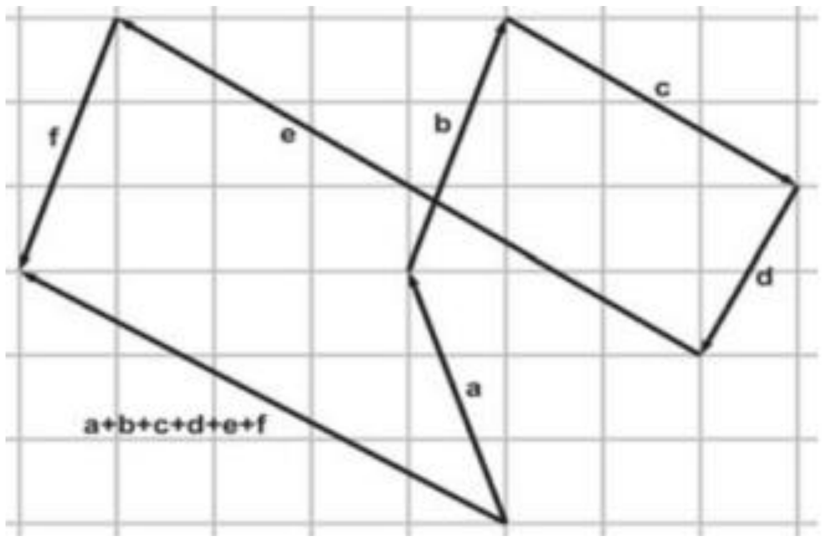
$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \vdots \\ a_{n-1} + b_{n-1} \\ a_n + b_n \end{bmatrix}$$

Addition/Substraction

Geometric interpretation

Multiple additions/substrations represent a sequence of displacements.





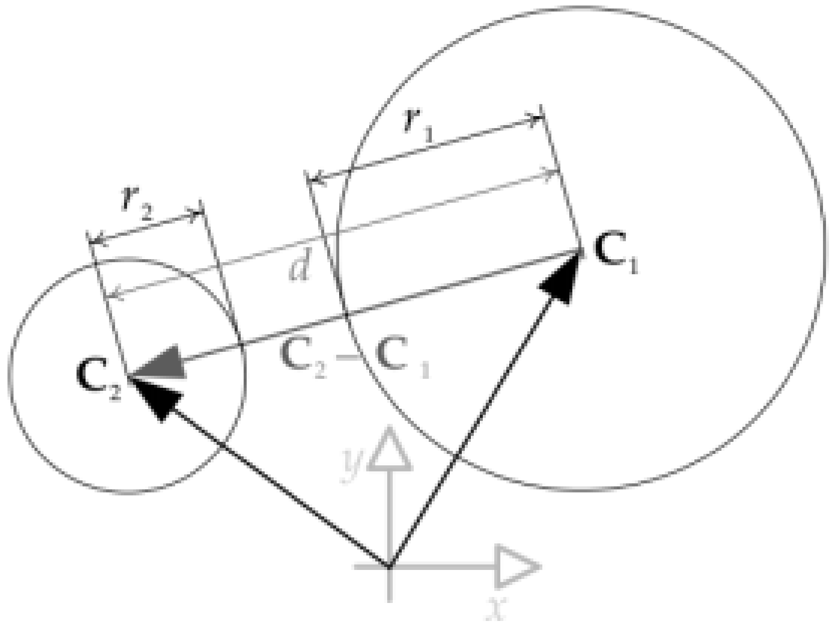
Question

Given two spheres

c_1 et c_2 : sphere center

r_1 et r_2 : sphere radius

Do the spheres intersect?



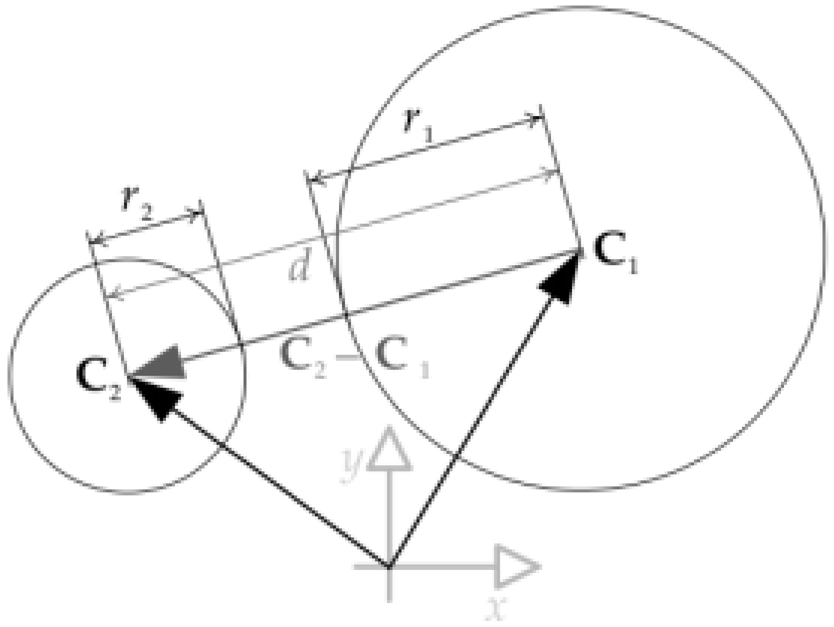
Question

Given two spheres

c_1 et c_2 : sphere center

r_1 et r_2 : sphere radius

Do the spheres intersect?



There is intersection iff $\|\mathbf{c}_2 - \mathbf{c}_1\| < (r_1 + r_2)$

Vector operations

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

Dot product

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

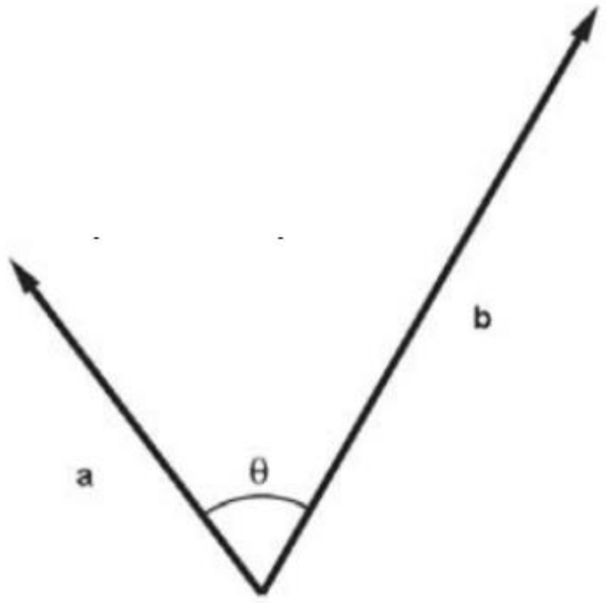
Geometric interpretation

Mesure the « similarity » of two directions

a.b > 0 => angle < 90°, both vectors have more or less the same direction

a.b = 0 => angle = 90°, the vectors are perpendicular

a.b < 0 => angle > 90°: the vectors have more or less opposite directions



Vector operations

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

Dot product

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos \theta$$

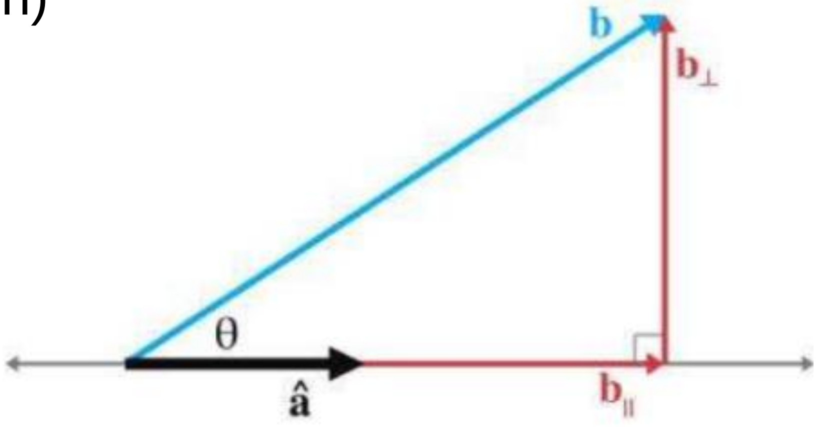
Geometric interpretation (II)

Given two vectors **â** (unit vector) and **b**, then **â.b** represents the signed length of the *projection* of **b** on the line defined by the direction of **â.**

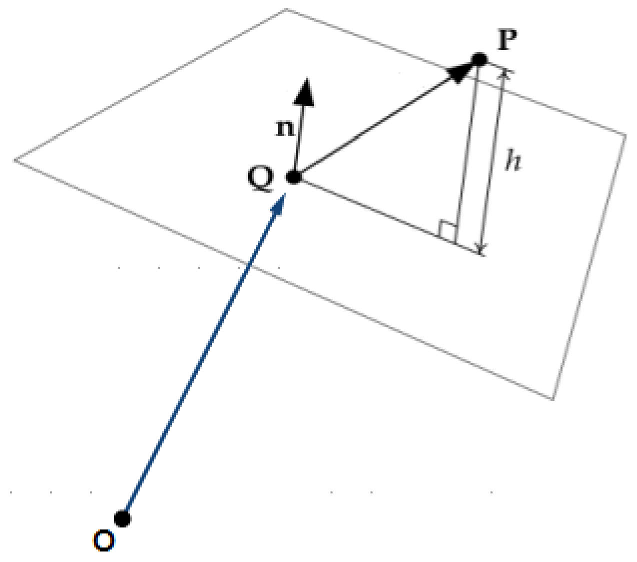
You can then split **b** into two components **b** = **b**$_{\parallel}$ + **b**$_{\perp}$ with

- **b**$_{\parallel}$ parallel to **â** (the projection)
- **b**$_{\perp}$ perpendicular to **â**

b$_{\parallel}$=(**â** · **b**) **â**, and **b**$_{\perp}$=**b** – **b**$_{\parallel}$



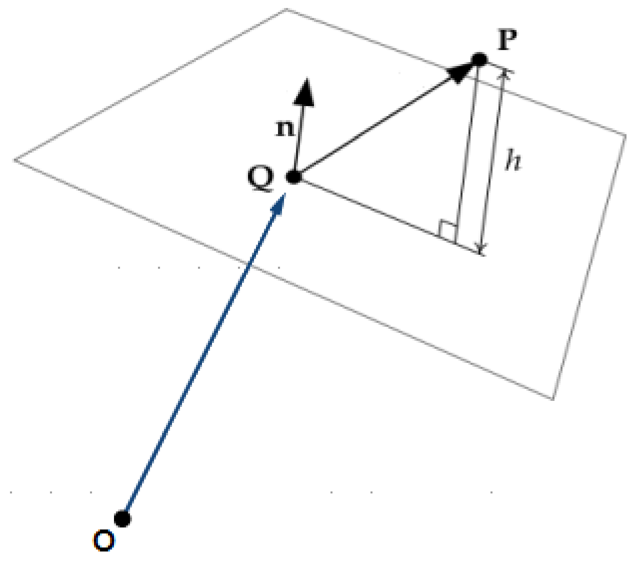
Question



A plane is defined by a point Q on the plane, and a unit vector n being perpendicular to the plane.

Find the vertical height h of a point P above a plane.

Question



A plane is defined by a point Q on the plane, and a unit vector n being perpendicular to the plane.

Find the vertical height h of a point P above a plane.

The solution is h = ||(P – Q) . n||

Vector operations

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} y_1 z_2 - z_1 y_2 \\ z_1 x_2 - x_1 z_2 \\ x_1 y_2 - y_1 x_2 \end{bmatrix}$$

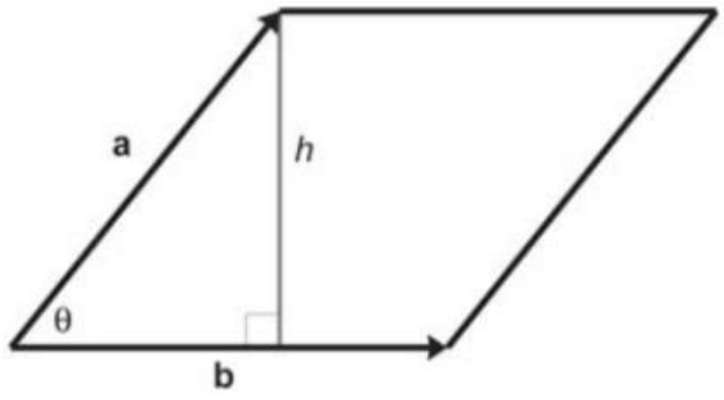
$$\|\mathbf{a} \times \mathbf{b}\| = \|\mathbf{a}\| \|\mathbf{b}\| \sin\theta$$

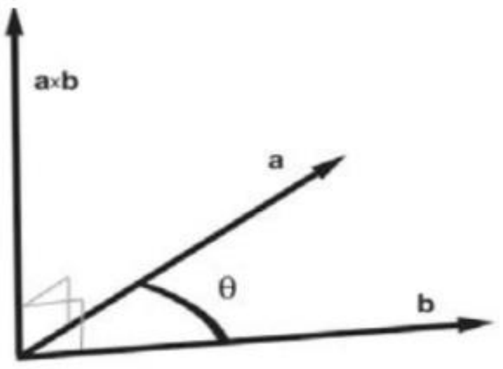
Cross product

- 3D vectors only
- The result is a vector
- a x b = – (b x a)

Geometric interpretation

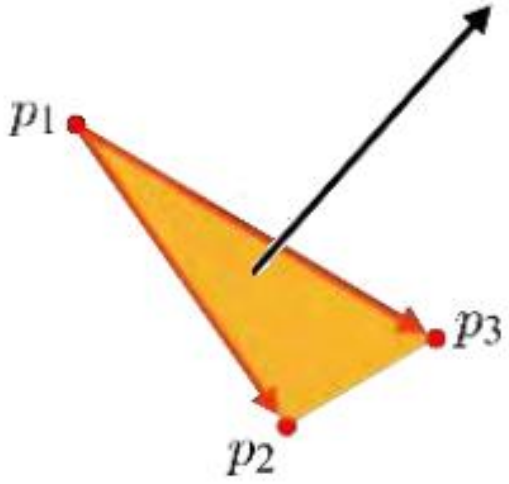
- The cross product vector is perpendicular to the two other ones.
- The length of the new vector is equal to the area enclosed by the parallelogram formed between the two vectors.





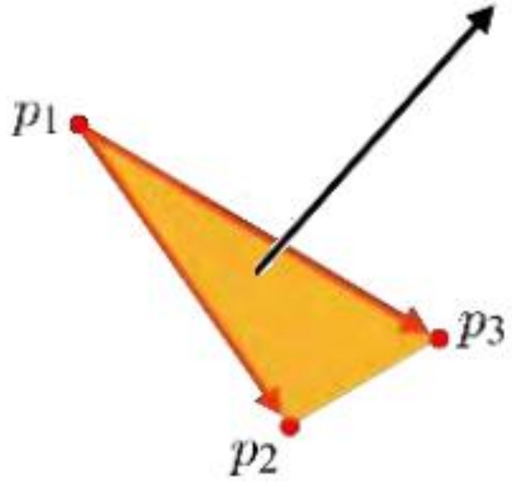
Question

Find a vector which is perpendicular to the surface of a triangle defined by the points p1, p2 and p3.



Question

Find a vector which is perpendicular to the surface of a triangle defined by the points p1, p2 and p3.



The solution is v = (p2-p1) x (p3-p1)

Exercise

Implement and test the 3D vector class « vec3 » having at least the following methods.

- Magnitude v.length() : real
- Normalization v.normalize() : vec3
- Negation v.neg() : vec3
- Scalar Multiplication v.mul(k:real) : vec3
- Addition/Substraction v.add(w:vec3), v.sub(w:vec3) : vec3
- Dot product v.dot(w:vec3) : real
- Cross product v.cross(w:vec3) : vec3
- Output v.print()

Matrix

- A rectangular array or table of numbers arranged in rows and columns
- Can be also be seen as a collection of line or column vectors
- We are mostly intersted in square matrices (2x2, 3x3 and 4x4)

$$
\begin{array}{c|cccc|}
 & 1 & 2 & \dots & n \\
1 & a_{11} & a_{12} & \dots & a_{1n} \\
2 & a_{21} & a_{22} & \dots & a_{2n} \\
3 & a_{31} & a_{32} & \dots & a_{3n} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
m & a_{m1} & a_{m2} & \dots & a_{mn}
\end{array}
$$

Matrices are VERY important for 3D computer graphics

As we will see, matrices can represent many kinds of 3D manipulations (rotate, scale, project, etc.).

Matrix operations

- Matrix-Vector Multiplication
- Scalar Multiplication
- Matrix-Matrix Multiplication
- Inverse

Matrix operations

Matrix-Vector Multiplication

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} xm_{11} + ym_{12} + zm_{13} \\ xm_{21} + ym_{22} + zm_{23} \\ xm_{31} + ym_{32} + zm_{33} \end{bmatrix}$$

Geometric interpretation

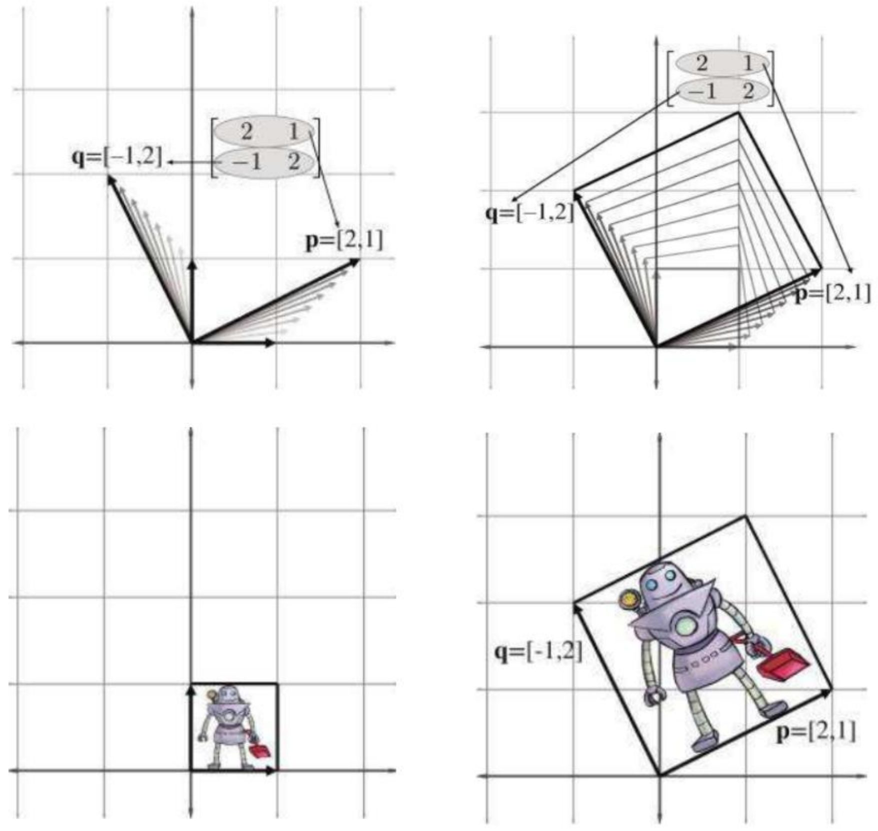
- Transforms a vector into another vector
- The origin is never affected
- Possible to alter lengths, surfaces, angles, volumes, but no displacement

The application of a matrix to a 3D vector is called a linear transformation.

Matrix operations

To get an idea of the linear transformation, you can look at the columns of the matrix. They display its effect on the basis vectors.

$$\mathbf{M} = \begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix}$$



Special Linear Transformations

- Rotation
- Scaling
- Orthographic Projection

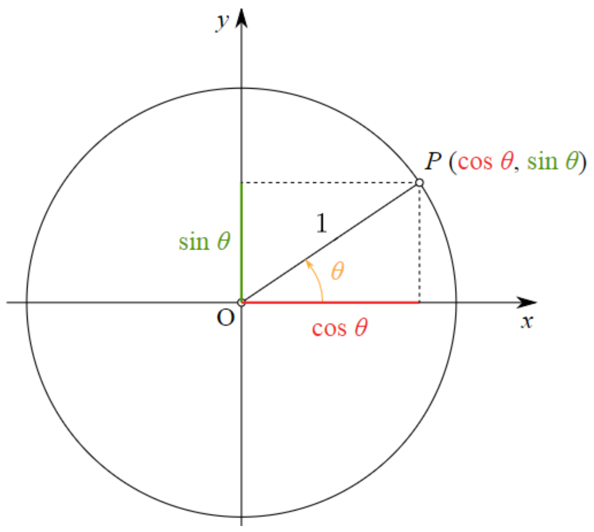
Rotation

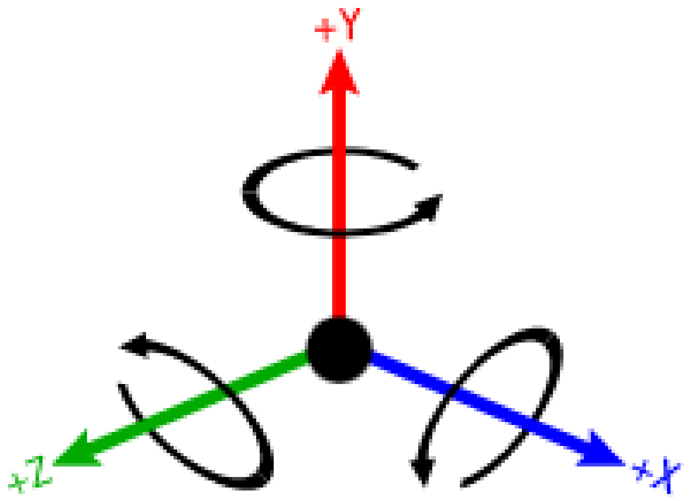
Elementary rotations (counter-clockwise around the basic axes):

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

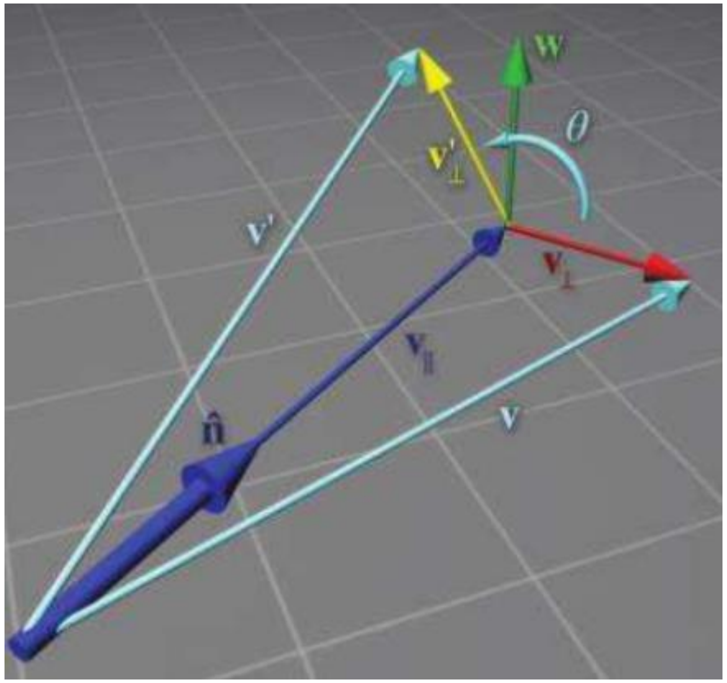
$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$





Rotation

It is also possible to define a matrix which represents the rotation around an arbitrary axis:



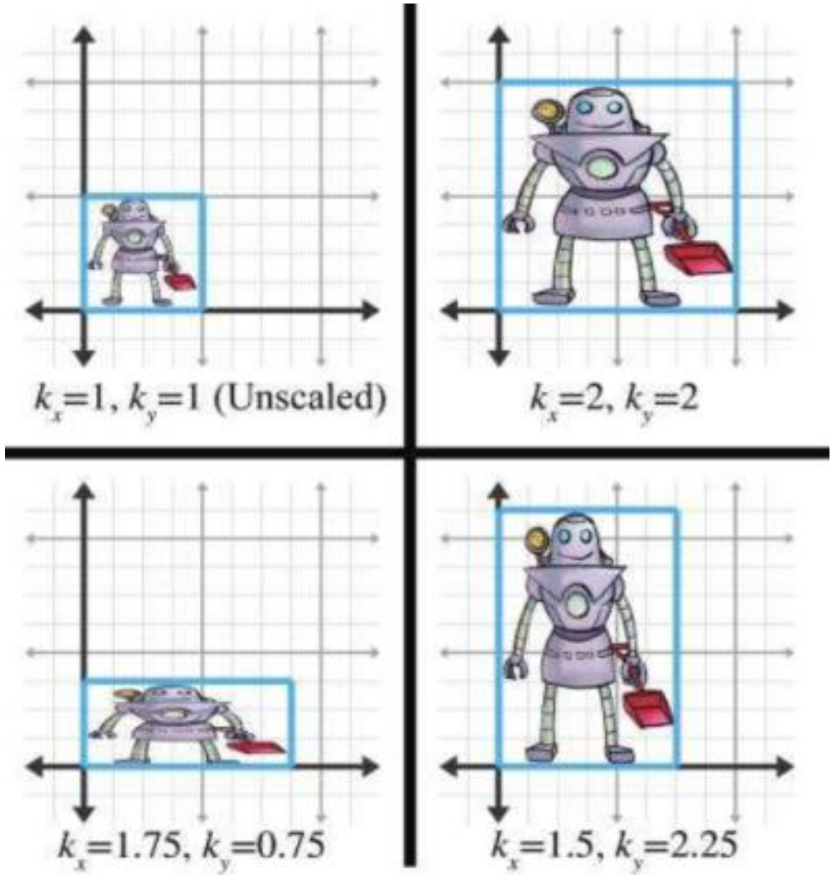
$$\mathbf{R}(\hat{\mathbf{n}}, \theta) =$$

$$\begin{bmatrix} {n_x}^2(1-\cos\theta)+\cos\theta & n_x n_y(1-\cos\theta)+n_z\sin\theta & n_x n_z(1-\cos\theta)-n_y\sin\theta \\ n_x n_y(1-\cos\theta)-n_z\sin\theta & {n_y}^2(1-\cos\theta)+\cos\theta & n_y n_z(1-\cos\theta)+n_x\sin\theta \\ n_x n_z(1-\cos\theta)+n_y\sin\theta & n_y n_z(1-\cos\theta)-n_x\sin\theta & {n_z}^2(1-\cos\theta)+\cos\theta \end{bmatrix}$$

Scale

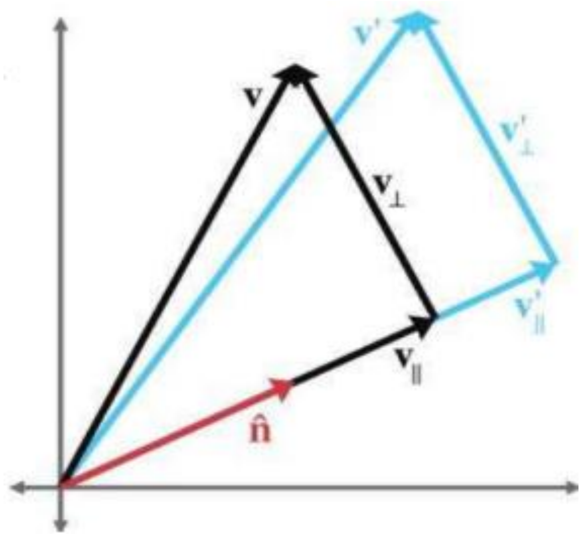
Magnifies or demagnifies in the direction of the three axes.

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



Scale

It is also possible to scale in the direction of an arbitrary axis.



$$\mathbf{S}(\hat{\mathbf{n}}, k) =$$

$$\begin{bmatrix} 1+(k-1)\,{n_x}^2 & (k-1)\,n_x n_y & (k-1)\,n_x n_z \\ (k-1)\,n_x n_y & 1+(k-1)\,{n_y}^2 & (k-1)\,n_y n_z \\ (k-1)\,n_x n_z & (k-1)\,n_y n_z & 1+(k-1)\,{n_z}^2 \end{bmatrix}$$

Orthographic Projection

All points are flattened and projected on a plane going through the origin.

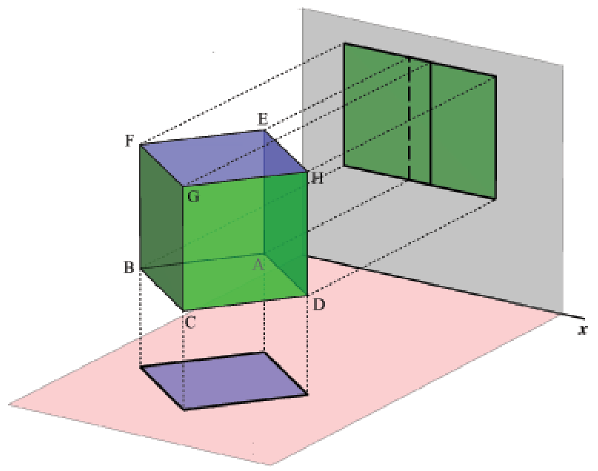
Orthographic projection can be seen as a scaling with a factor *k* = 0 in a given direction.

Elementary projections:

$$\mathbf{P}_{xy} = \mathbf{S}\left(\begin{bmatrix}0 & 0 & 1\end{bmatrix}, 0\right) = \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 0\end{bmatrix}$$

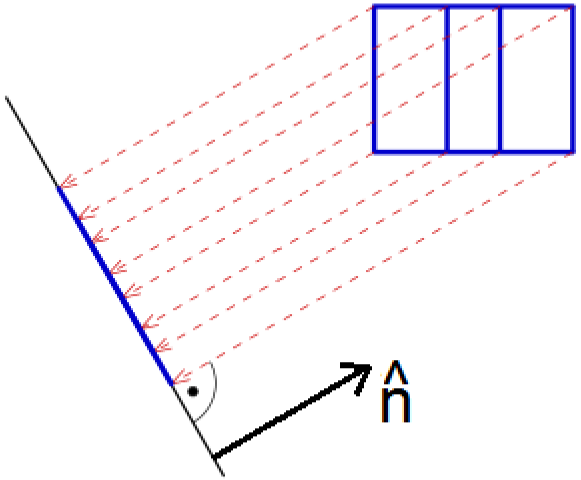
$$\mathbf{P}_{xz} = \mathbf{S}\left(\begin{bmatrix}0 & 1 & 0\end{bmatrix}, 0\right) = \begin{bmatrix}1 & 0 & 0\\0 & 0 & 0\\0 & 0 & 1\end{bmatrix}$$

$$\mathbf{P}_{yz} = \mathbf{S}\left(\begin{bmatrix}1 & 0 & 0\end{bmatrix}, 0\right) = \begin{bmatrix}0 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}$$



Orthographic Projection

It is also possible to project on an arbitrary plane (going through the origin).



$$\mathbf{P}(\hat{\mathbf{n}}) = \mathbf{S}(\hat{\mathbf{n}}, 0) =$$

$$\begin{bmatrix} 1 - {n_x}^2 & -n_x n_y & -n_x n_z \\ -n_x n_y & 1 - {n_y}^2 & -n_y n_z \\ -n_x n_z & -n_y n_z & 1 - {n_z}^2 \end{bmatrix}$$

Matrix operations

Scalar Multiplication

$$k\mathbf{M} = k\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \\ m_{41} & m_{42} & m_{43} \end{bmatrix} = \begin{bmatrix} km_{11} & km_{12} & km_{13} \\ km_{21} & km_{22} & km_{23} \\ km_{31} & km_{32} & km_{33} \\ km_{41} & km_{42} & km_{43} \end{bmatrix}$$

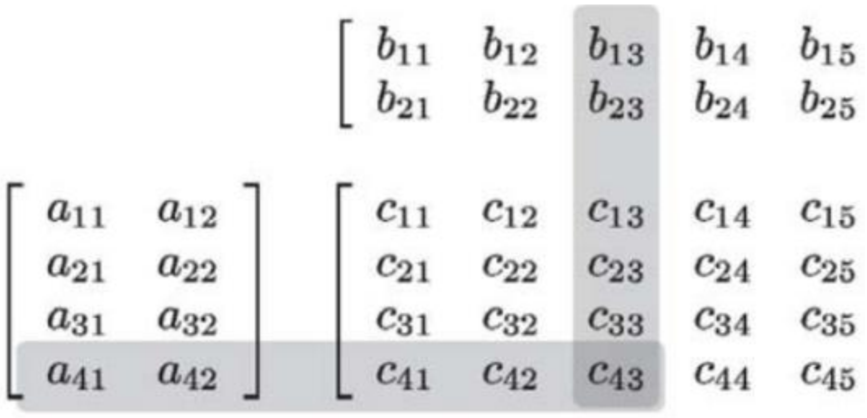
Geometric interpretation

Adds a scale effect to the matrix.

Matrix operations

Matrix-Matrix Multiplication

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$



b_{11} b_{12} b_{13} b_{14} b_{15}
b_{21} b_{22} b_{23} b_{24} b_{25}
a_{11} a_{12}
a_{21} a_{22}
a_{31} a_{32}
a_{41} a_{42}
c_{11} c_{12} c_{13} c_{14} c_{15}
c_{21} c_{22} c_{23} c_{24} c_{25}
c_{31} c_{32} c_{33} c_{34} c_{35}
c_{41} c_{42} c_{43} c_{44} c_{45}

Geometric interpretation

Combines the effects of two linear transformations on a vector.

Attention: AB≠ BA

Throughout this course, we use the « pre-multiplication » standard from mathematics, i.e. applying A to v, then B, means that we have to compute BAv. (). This is also used in OpenGL.

It is possible to define a « post-multiplication » linear algebra such that we could write more naturally: vAB. This is used in Direct3D. However it implies a lot of rethinking in terms of columns and rows, so we may be less comfortable with this approach.

Inverse

The inverse of a square matrix A, written A^{-1}, is the matrix such that $AA^{-1} = A^{-1}A = I$. Note that $(AB)^{-1} = B^{-1}A^{-1}$

Attention: not all square matrices can be inverted!

There are two main ways to obtain the inverse matrix:

- Gauss-Jordan elimination (more suited for humans)
- Determinant and adjugate matrix (more suited for computers):

$$A^{-1} = \frac{1}{|A|}\operatorname{adj}(A)$$

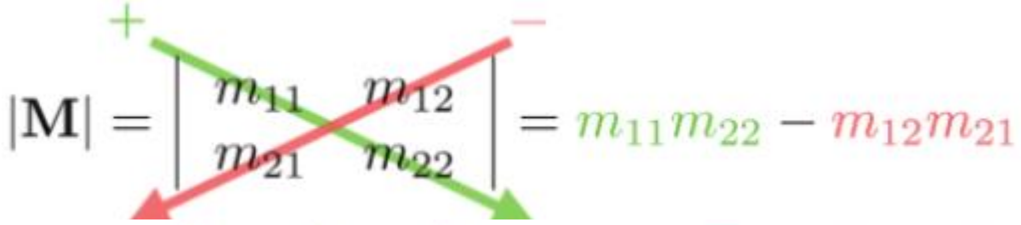
Geometric interpretation

To « undo » a transformation, you apply its inverse.

Determinant

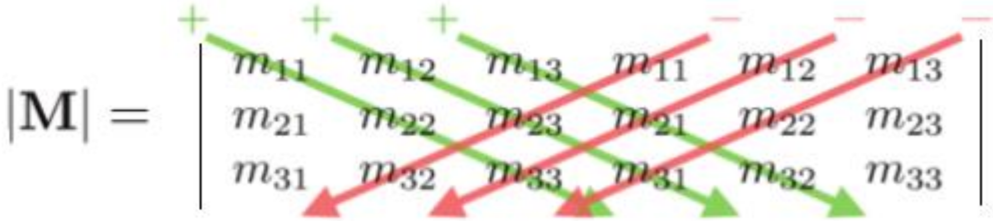
Returns a scalar value for squared matrices.

2D:



$$|\mathbf{M}| = \begin{vmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{vmatrix} = m_{11}m_{22} - m_{12}m_{21}$$

3D:



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

$$\begin{aligned} \begin{vmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{vmatrix} &= m_{11}m_{22}m_{33} + m_{12}m_{23}m_{31} + m_{13}m_{21}m_{32} \\ &\quad - m_{13}m_{22}m_{31} - m_{12}m_{21}m_{33} - m_{11}m_{23}m_{32} \\ &= m_{11}(m_{22}m_{33} - m_{23}m_{32}) \\ &\quad + m_{12}(m_{23}m_{31} - m_{21}m_{33}) \\ &\quad + m_{13}(m_{21}m_{32} - m_{22}m_{31}). \end{aligned}$$

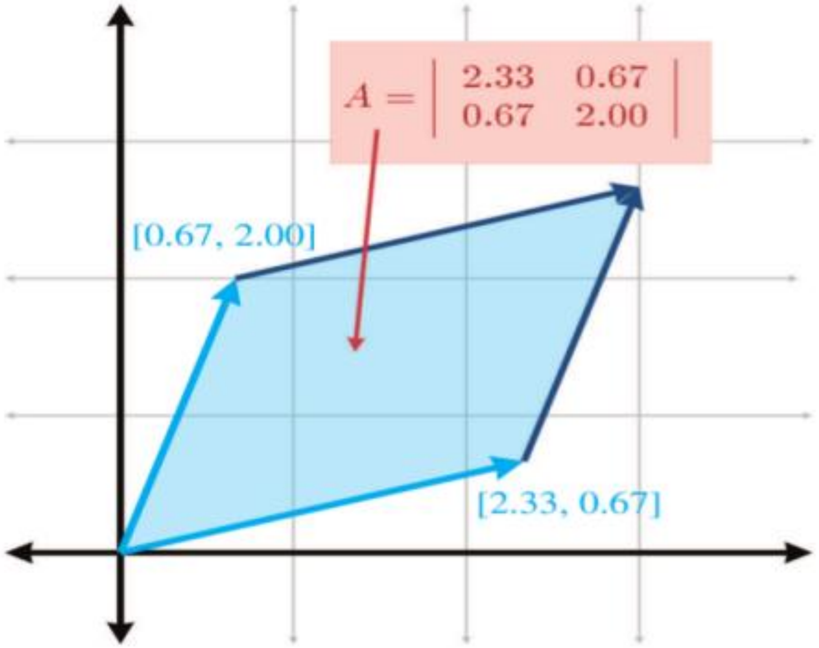
Determinant

Geometric interpretation

|M| is the volume (or surface in 2D) of the rhomboid whose edges are the transformed basis vectors.

Some observations:

- |Rotation| = 1
- |Scale| = k_x*k_y*k_z
- |Orthographic projection| = 0
- M is invertible iff |M|≠ 0



$A = \begin{vmatrix} 2.33 & 0.67 \\ 0.67 & 2.00 \end{vmatrix}$
[0.67, 2.00]
[2.33, 0.67]

Adjugate matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

The adjugate of A is the transpose of the cofactor matrix of A.

$$\operatorname{adj}(\mathbf{A}) = \begin{bmatrix} +\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & +\begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} \\ \\ -\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & +\begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ \\ +\begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & +\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}$$

Affine transformations

- It is very convenient to use one single mathematical structure (**3x3 matrix**) to produce so many different effects.
- However, linear transformations never affect the origin, therefore they cannot represent displacements.
- Also, without displacements we are also unable to rotate around an axis which does not go through the origin, or to project on a plane with does not lie on the origin.
- In order to include these so-called affine transformations, we extend the 3D space to the homogeneous space. Here, linear transformations and translations can be combined in a single **4x4 matrix**.

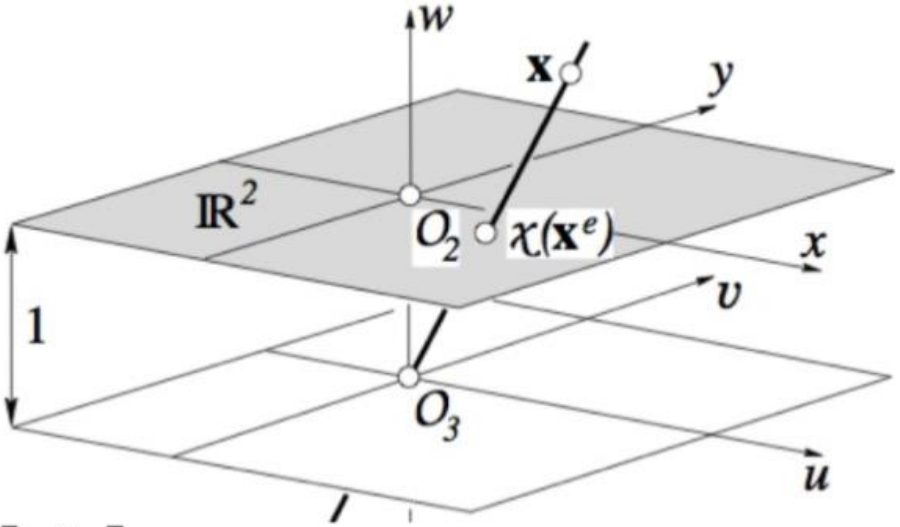
Homogeneous space

- We embed the 3D space into the 4D space
- Cartesian coordinates (*x*,*y,z)* are represented by the *homogeneous coordinates* (*x*,*y*,z,1).
- The homogenous coordinates (*x*,*y*,z,*w≠0*) correspond to the cartesian coordinates (*x*/*w*, *y*/*w*, z/w)
- The homogenous coordinates (*x*,*y*,z,0) lie « at infinity ». They are not affected by translations and correspond to a direction rather than a point.

Homogeneous space

Illustration for the 2D homogenous space.

(The same for the 3D homogenous space with one more dimension.)



$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u/w \\ v/w \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} u/w \\ v/w \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Homogeneous space

In homogeneous space, a 4x4 matrix can describe both linear transformation and translation in 3D cartesian space.

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Linear transformation

Translation

Exercise

Implement and test the classes « vec4 » and « mat4 » having at least the following methods.

vec3

- Convert cartesian to homogenous: v.hom(): vec4

vec4

- Convert homogenous to cartesian: v.cart(): vec3
- Output: v.print()

mat4

- Scalar Multiplication: M.mul(k:real): mat4
- Matrix-Matrix Multiplication: M.mul(N:mat4): mat4
- Matrix-Vector Multiplication M.mul(v:vec4): vec4
- Inverse: M.inv(): mat4 (https://semath.info/src/inverse-cofactor-ex4.html)
- Output: M.print()

Note: we do not need mat3.

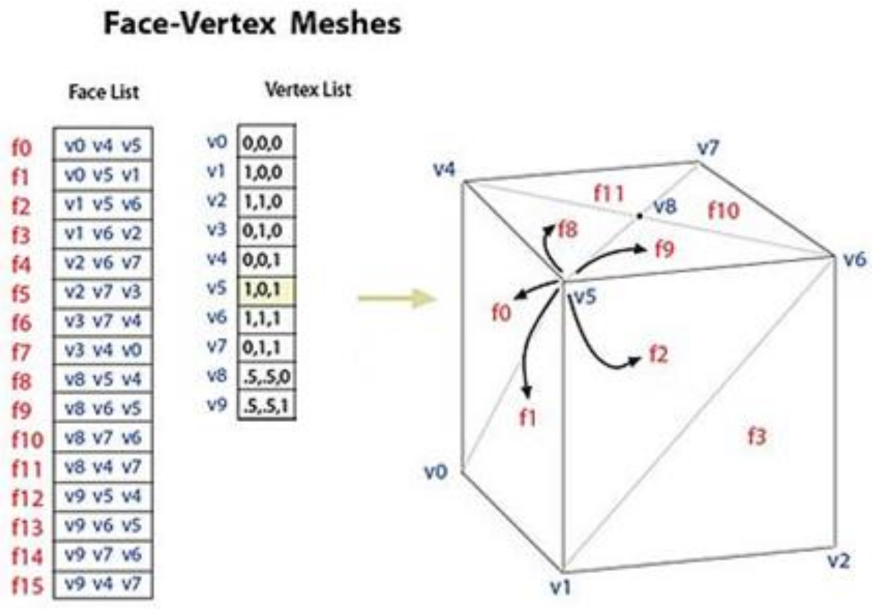
Triangle Mesh

In 3D computer graphics, a polygon mesh is a collection of vertices, edges and faces that defines the shape of a polyhedral object. The faces usually consist of triangles (triangle mesh), quadrilaterals (quads), or other simple convex polygons.

- Vertex: a position
- Edge: a connection between two vertices
- Face: a closed set of edges

Polygon meshes may be represented in a variety of ways, to store the vertex, edge and face data.

In the scope of this course, we will use the most widely used mesh representation: **face-vertex meshes**.



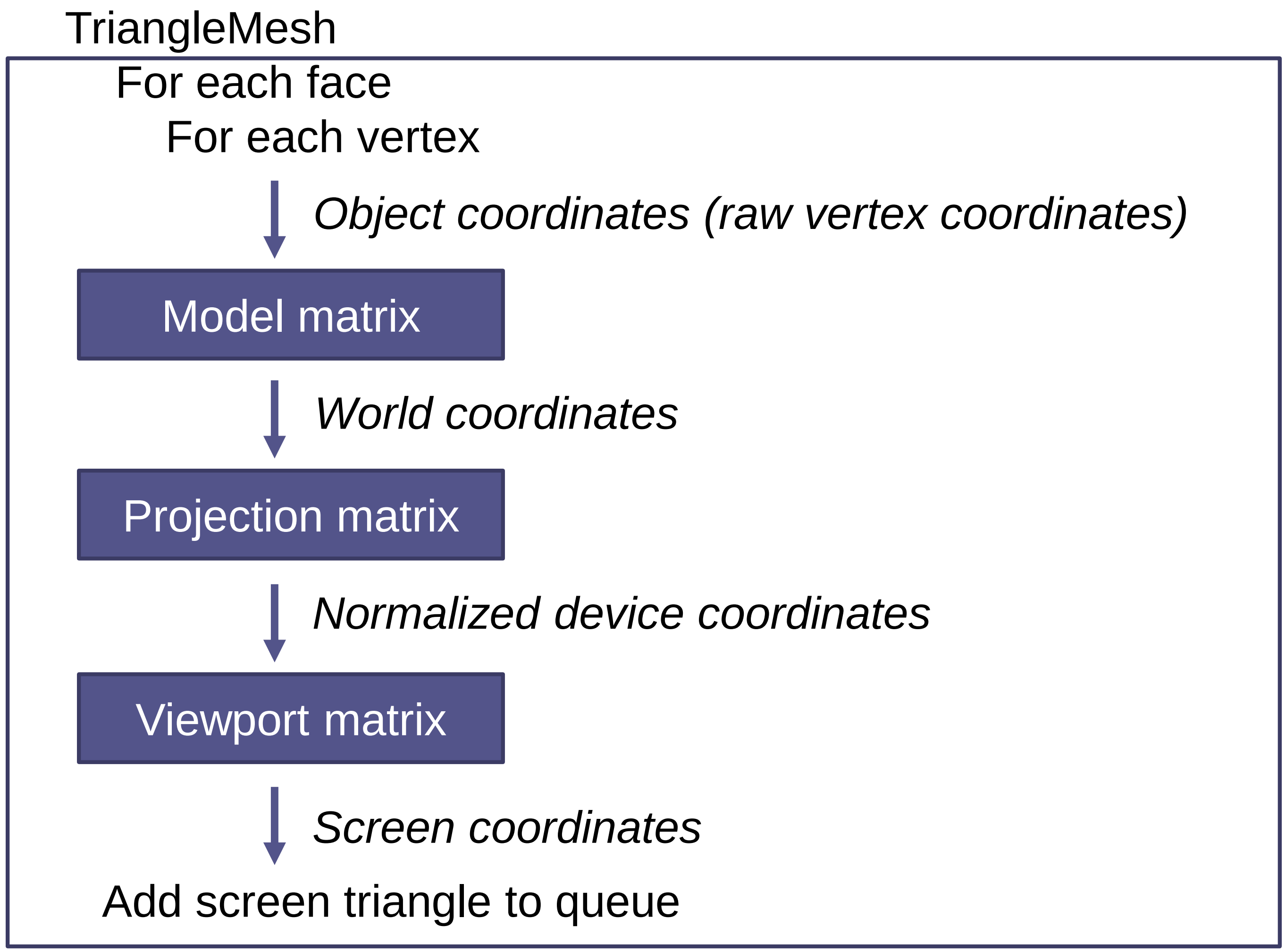
Face List

f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

Vertex List

v0	0,0,0
v1	1,0,0
v2	1,1,0
v3	0,1,0
v4	0,0,1
v5	1,0,1
v6	1,1,1
v7	0,1,1
v8	.5,.5,0
v9	.5,.5,1

A simple rendering pipeline



Render queue in wireframe mode

Model matrix

Specifies the transformation from object coordinates (also called local coordinates) to a common world coordinate system. It pushes the object away from the origin and optionally applies a scale or a rotation to it.

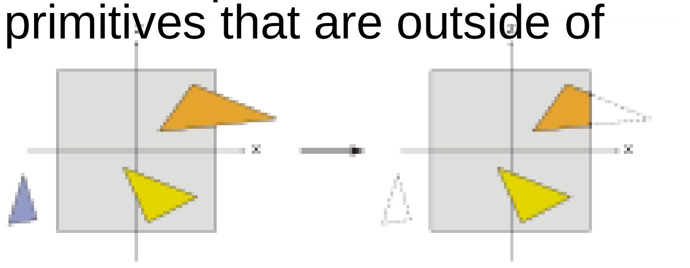
$$\mathrm{M}_{\text{object}\rightarrow\text{world}} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & t_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & t_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{with } \mathrm{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \quad \text{and } \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

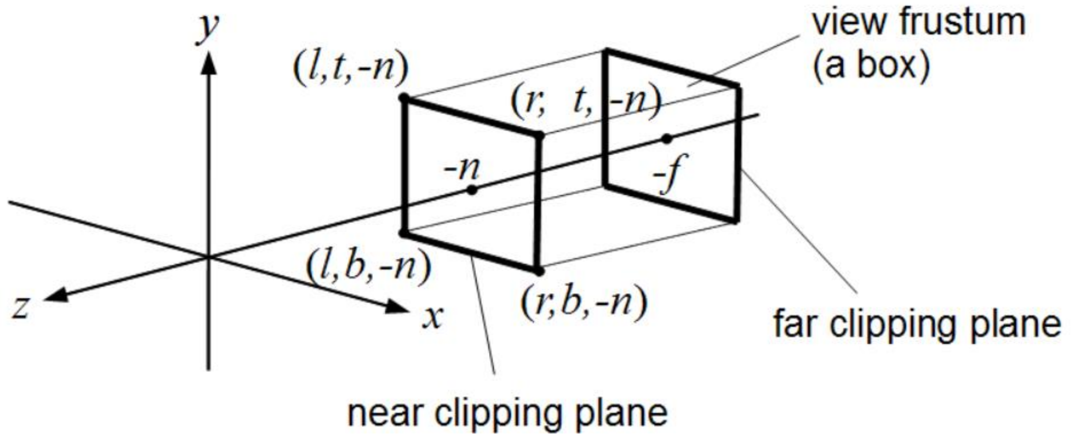
- A is a 3×3 matrix, which represents a linear transformation
- t is a 3D vector, which represents a translation

Projection matrix

For the moment we only consider orthographic projection (without foreshortening). The clipping planes specified by the parameters l, r, t, b, n, f define the **view frustum.**

The result of the projection transformation are **normalized device coordinates**. Their values are between -1 and +1 for all points in the visible part of the scene. All primitives and all parts of primitives that are outside of the box should be clipped away.





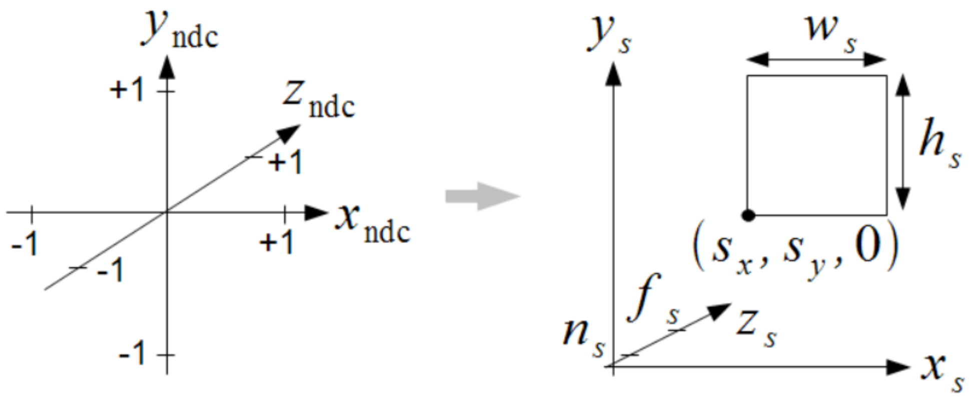
$$M_{\text{projection}} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewport matrix

The viewport is the rectangle of the screen that is rendered.

In normalized device coordinates (ndc), the view volume is a box centered around the origin with the coordinates inside the box between -1 and +1. The box is now mapped to screen coordinates. This is a shift, followed by scaling. The parameters are

- the coordinates sx and sy of the lower, left corner of the viewport
- its width ws and height hs
- the depths ns and fs of the front and near clipping planes. These values are typically between 0 and 1 and used for depth tests.



$$\mathrm{M}_{\text{viewport}} = \begin{bmatrix} \frac{w_s}{2} & 0 & 0 & s_x + \frac{w_s}{2} \\ 0 & \frac{h_s}{2} & 0 & s_y + \frac{h_s}{2} \\ 0 & 0 & \frac{f_s - n_s}{2} & \frac{n_s + f_s}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Important classes

```
class Scene
  projection: mat4 // projection matrix
  viewport: mat4 // viewport matrix

class TriangleMesh
  vertices: List<vec3> // a set of vertex coordinates
  // a set of faces, i.e. three vertex indices
  // forming the corners of a triangle
  // (counter-clockwise order):
  faces: List<int[3]>
  model: mat4 // model matrix

class ScreenTriangle
  v0, v1, v2: vec3
 // render the triangle:
 // (int(v0.x),int(v0.y))
 // (int(v1.x),int(v1.y))
 // (int(v2.x),int(v2.y))
```

Exercise

Implement a wireframe renderer for triangle meshes with orthographic projection. You may ignore the clipping step. Design some simple meshes for your tests. The user can scale, rotate and translate the rendered object.