Basic Techniques in Computer Graphics

Winter 2017 / 2018

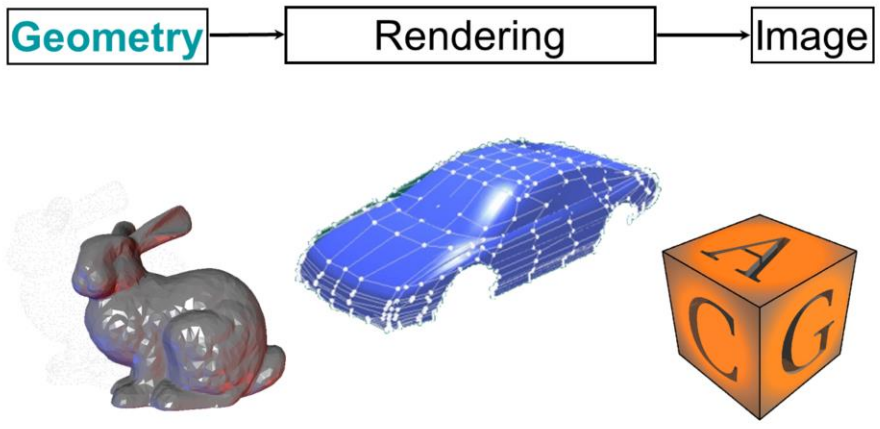
The slide comments are not guaranteed to be complete, they are no alternative to the lectures itself. So go to the lectures and write down your own comments!

Overview

- **Objects in 3D**
- Mathematical Representations
- Transformations

In this lecture, we will give a short overview of different representations for 3D objects. We then formalize the representations (see blackboard) and show how to do transformations with those mathematical representations.

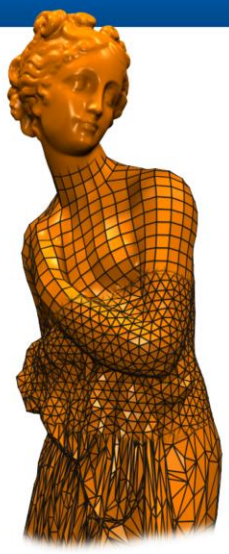
Geometry I



There are different ways to represent the geometry of a scene. From left to right:
Points, Triangles, Freeform Surfaces with control cage, Constructive Solid Geometry.

Objects in 3D

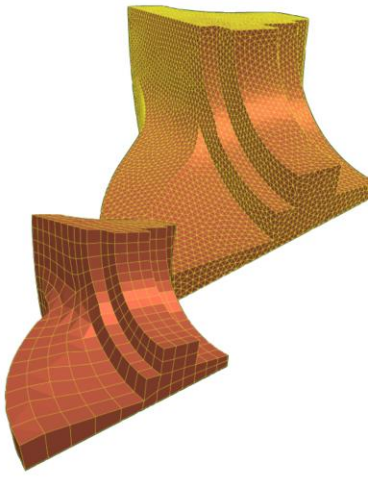
- Polygonal meshes
- Point Clouds
- Constructive Solid Geometry (CSG)
- Volumes



There are various kinds of geometric representations for objects in 3D. Each way has its own advantages and disadvantages and its use strongly depends on the application.
Polygonal meshes are the most important representation, but as meshes get more and more detailed point clouds also have emerged as a important representation.

Objects in 3D

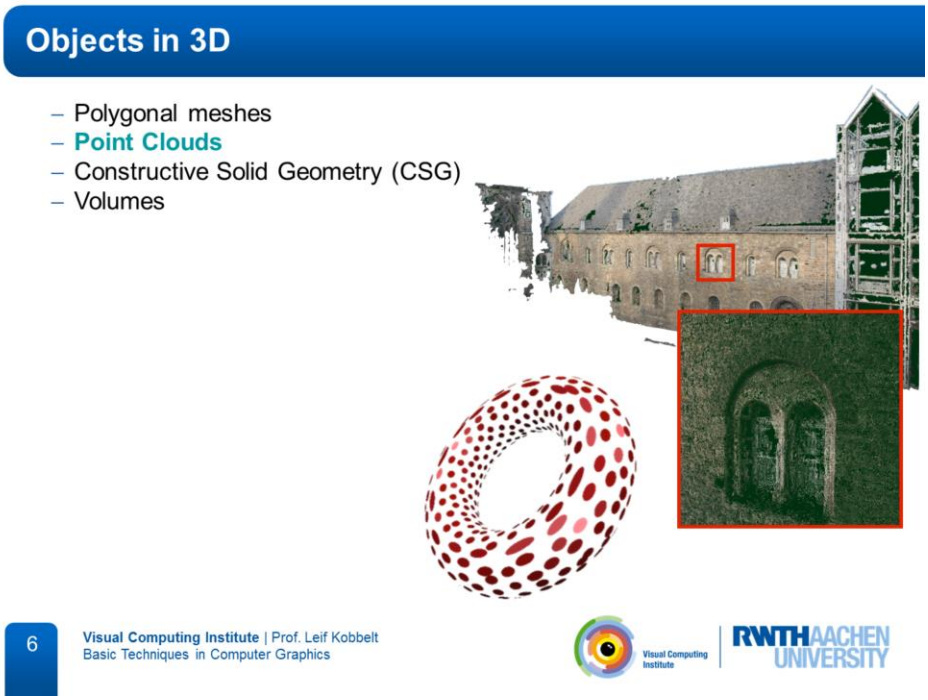
- **Polygonal meshes**
  - **Triangles, Quads, ...**
- Point Clouds
- Constructive Solid Geometry (CSG)
- Volumes



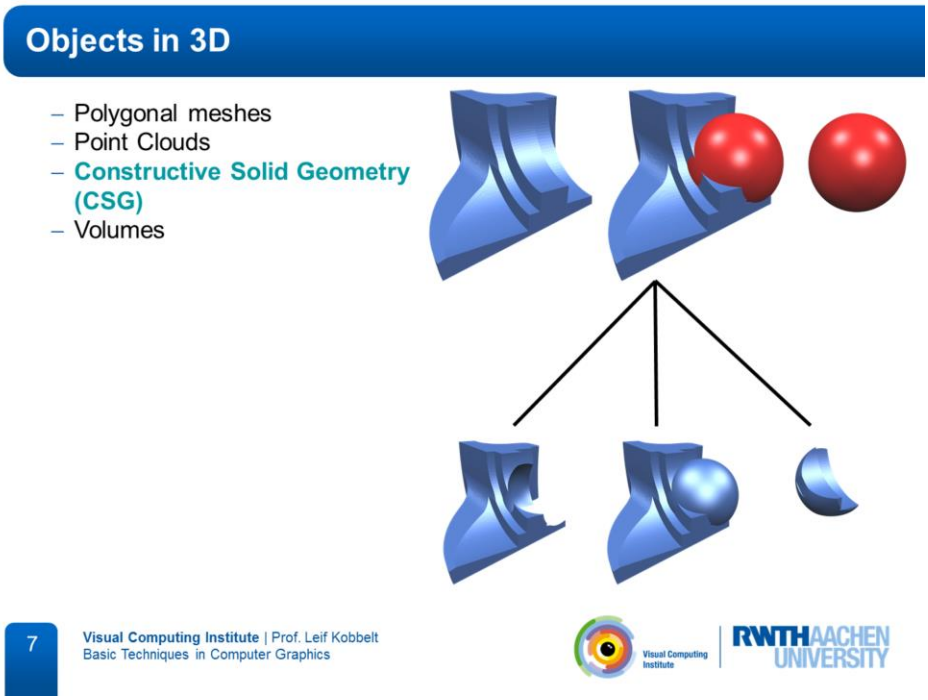
Polygonal meshes are traditionally used to model and render 3D objects. All modern graphic cards are designed to quickly render meshes consisting of triangles (or quads). Using triangles offers many advantages, such as simplified clipping, rasterization, interpolation... .

Quad meshes are mainly used in industrial applications, such as finite element simulations, CAD modeling, movie production, etc. .

In general, we can use all sorts of polygons, but most graphics hardware internally represents the polygons by triangles.



Some data acquisition methods, such as laser scans or Structure-from-motion reconstructions from images, only output point clouds instead of closed surface meshes. Since the output usually contains millions of 3D points, we need efficient ways of drawing such large datasets, as well as techniques to approximate the depicted structures using a few ellipses instead of millions of 3D points (splat rendering).
Also, if the data is very dense, it makes sense to use points instead of triangles since many triangles would project to a single pixel only.



We also want to generate new objects. Here, CSG allows to generate complex models by combining simpler objects. Since CSG models solid objects, we can assign a value to every 3D point, signaling whether the point is inside or outside the object. Compared to polygonal meshes, which only model the surface of objects, cutting out parts of the model with CSG does not result in holes in the model.

Objects in 3D

- Polygonal meshes
- Point Clouds
- Constructive Solid Geometry (CSG)
- **Volumes**

As a further generalization, we can assign any scalar value to a point in 3D space. When rendering, we are interested to display all 3D points with a certain scalar attached to them. This is especially important in the area of medical image processing, where data is usually recorded in volumetric forms (e.g. CT scans) where the scalar value for every volume element has a certain meaning, e.g. bone structure density.

Overview

- Objects in 3D
- **Mathematical Representations**
- Transformations

Points

Definition:

A point p in a 3-dimensional space is defined according to a basis $B = \{b_1, b_2, b_3\}$ of the vector space $\mathbb{R}^3$ as a linear combination of the basis vectors

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix}_B = (x, y, z)_B^T = \mathrm{x} \cdot b_1 + y \cdot b_2 + z \cdot b_3$$

with $b_1, b_2, b_3 \in \mathbb{R}^3, x, y, z \in \mathbb{R}^3$.

The most simplest form of 3D objects we will talk about in the lecture are points. Points are defined relative to a basis B of the 3D vector space R3 (remember your linear algebra course). Points are 0-dimensional objects. Notice that the definition above can easily be extended to arbitrary dimensions.

Points

Standard / canonical basis:

$$\varepsilon = \{e_1 = (1\,,0\,,0)^T, e_2 = (0\,,1\,,0)^T, e_3 = (0\,,0\,,1)^T\}$$

Simplified notation:

$$p = \begin{pmatrix} x \\ y \\ z \end{pmatrix}_B = \begin{pmatrix} x \\ y \\ z \end{pmatrix}_\varepsilon$$

For the sake of brevity, we will omit the symbol for the base when using the standard basis of R3.

Product Operators

inner / dot product

- $< \mathrm{p}, \mathrm{q} > = \mathrm{p}^T \mathrm{q} \in \mathbb{R}$

cross product

- $\mathrm{p} \times \mathrm{q} \in \mathbb{R}^3$

outer product

- $\mathrm{p}\mathrm{q}^T \in \mathbb{R}^{3\times 3}$

Dot Product

- Vectors are the difference between two points $\mathbf{v} = \mathbf{q} - \mathbf{p}$
- Euclidean **length** of a vector

$$\|v = (x, y, z)^T\| = \|v\|_2 = \sqrt{x^2 + y^2 + z^2}$$

- inner product / dot product of two vectors

$$\langle u, v \rangle = u^T \cdot v = u_x v_x + u_y v_y + u_z v_z$$

- **angle** between two vectors

$$\cos \alpha = \frac{u^T \cdot v}{\|u\|_2 \cdot \|v\|_2}$$

In German, the dot product is also called „Skalarprodukt“ (scalar product).

Cross Product

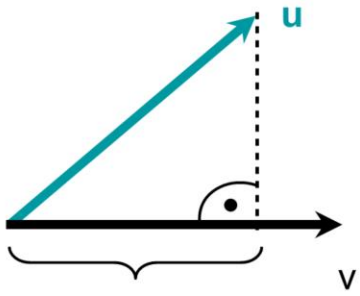
- generate orthogonal basis
- **right** hand rule

Outer Product

- e.g. matrix for orthogonal projection

Vectors

Projection of vector **u** onto vector **v** :



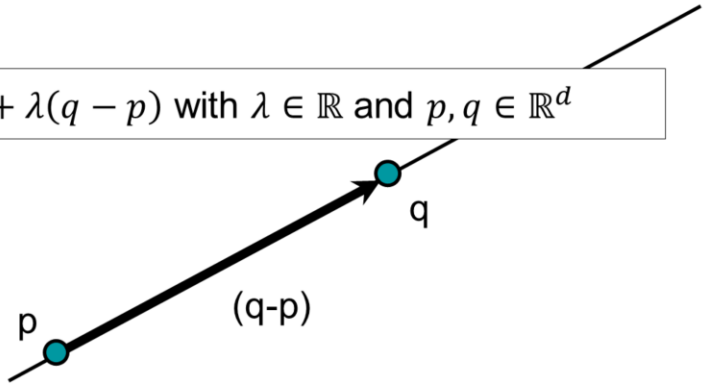
$$\frac{u^T \cdot v}{\|v\|_2}$$

$$\frac{u^T \cdot v}{\|v\|_2} \cdot \frac{v}{\|v\|_2}$$

Lines

- Lines are defined by two points **p**, **q**
- Lines are 1-dimensional objects
- **explicit representation**

$$p + \lambda(q - p) \text{ with } \lambda \in \mathbb{R} \text{ and } p, q \in \mathbb{R}^d$$



We can define a point on the line by a parameter λ : L(λ) = p + λ r, where we start at point p and then take λ steps into the direction r.
In some cases, it is convenient to express a line by two points p and q instead of a point and a direction, where (q-p) gives the direction of the line.

Lines

- **explicit representation**

$$p + \lambda(q - p) \text{ with } \lambda \in \mathbb{R} \text{ and } p, q \in \mathbb{R}^d$$

- **linear interpolation**

$$p + \lambda(q - p) = (1 - \lambda)p + \lambda q$$

  - all points between p and q have λ from $[0,1]$

- How to determine if point is on the line?
  - ➢ **Implicit representation**

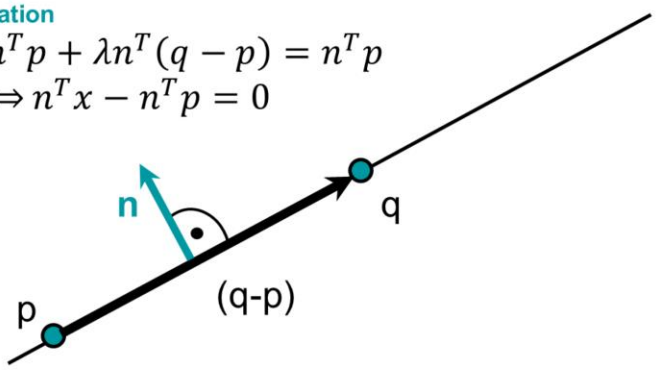
Another nice representation is to denote all points on the line through p and q by a linear combination of the two points. This representation is conceptionally more simple since we only use 3D points to compute a new point instead of a combination of points and lines.
Since the weights (1-λ) and λ add up to 1 = 1 - λ + λ, the formula above is an affine combination of 3D points.
All representations shown so far are parametric representation, since we can generate points on the lines using a (single) parameter.

Lines

- here in 2D
- normal **n** orthogonal to $(q-p)$, i.e $n^T(q-p)=0$
- point x on line : $x = p + \lambda(q-p)$

- **implicit representation**

$$n^T x = n^T p + \lambda n^T (q-p) = n^T p$$
$$\Leftrightarrow n^T x - n^T p = 0$$



Yet another representation for lines is their implicit representation. Here, a function L(x) is defined such that a point x is on the line iff L(x)=0. While this representation makes it hard to generate points on the line, using the implicit representation it is very easy to test whether a point x is on the line by evaluating L(x).

A common implicit representation for lines is given on the slide. Since $n^T p$ is constant, we can precompute its value.

Notice that this representation has another useful property: If L(x) > 0, L(x) gives the distance of the point to the line in the direction of the normal vector (or a multiple of the length of the normal if it does not have unit length).

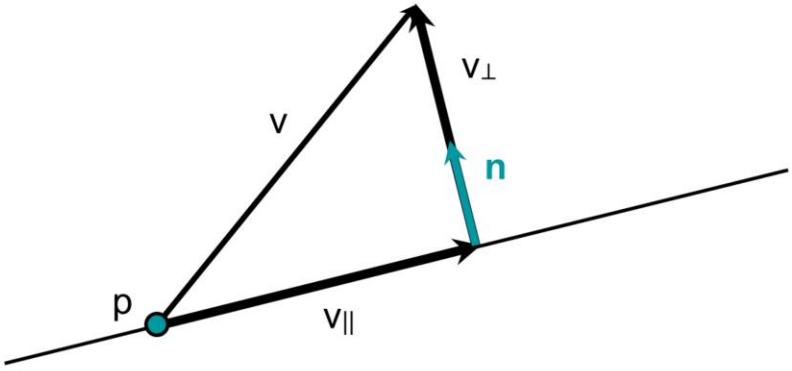
Here, we have only given the implicit representation

of a line in two dimensions. How can this representation be extended to three dimensions?

Lines

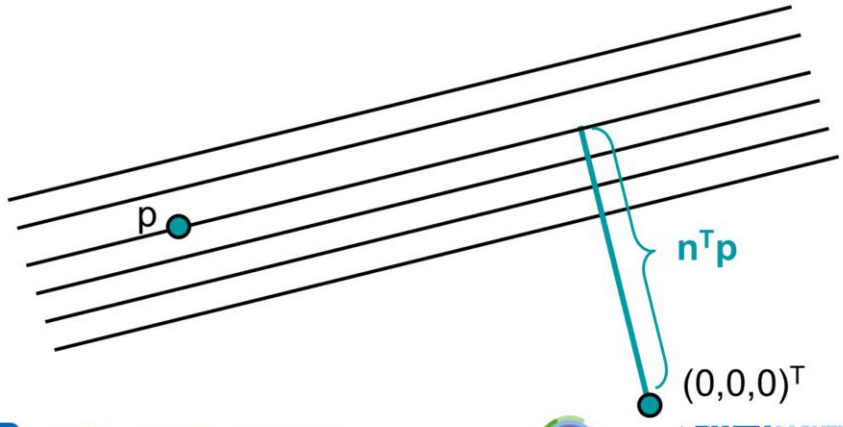
geometric intepretation:

- decompose v into $v = v_{||} + v_{\perp}$
- for points on line $v_{\perp} = 0$



Lines

- decompose v into $v = v_{||} + v_{\perp}$
- for points on line $v_{\perp} = 0$
- holds for set of parallel lines



For all points x on a certain line, the value of $n^T x$ is constant.

Planes

- Planes are defined by three points **a**, **b**, **c**
- Planes are 2-dimensional objects
- explicit representation

$$p + \lambda(b - a) + \mu(c - a) \text{ with } \lambda, \mu \in \mathbb{R} \text{ and } \mathrm{a}, \mathrm{b}, \mathrm{c} \in \mathbb{R}^3$$

For representing planes in 3D space, we need three points. Note the similarity to the parametric representation for lines.

Barycentric Coordinates

- **barycentric coordinates:**

$$a + \lambda(b - a) + \mu(c - a) = (1 - \lambda - \mu)a + \lambda b + \mu c$$
$$= \alpha a + \beta b + \gamma c$$

- obviously: $\alpha + \beta + \gamma = 1$
- computes center of mass (barycenter):

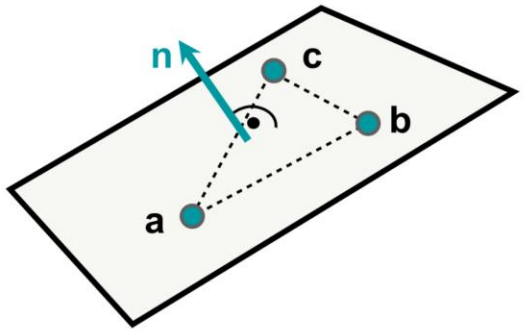
$$\alpha a + \beta b + \gamma c \triangleq \text{points} \cdot \text{mass weights}$$

Similarly, we can express all points on the plane as an affine combination of three points lying in the plane.
α, β, γ are called the barycentric coordinates of the point x = αa + βb + γc. In a physical interpretation of the barycentric coordinates, the barycenter x corresponds to the center of gravity of the three points a, b and c with masses α, β and γ.

Planes

- **implicit representation** similar to lines

$$n^T x - n^T a = n^T x - d = 0$$



Again, we are also interested in the implicit representation for planes. Similar to 2D lines, we use the normal of the plane to express the distance of a point from the plane. The derivation of the implicit representation is similar to the derivation for the implicit representation of 2D lines.

Polygons

- In CG we want to render polygons!
- Especially interested in polygons **in a plane**
  - easier to handle
- Important polygons:
  - Triangles
  - Quads

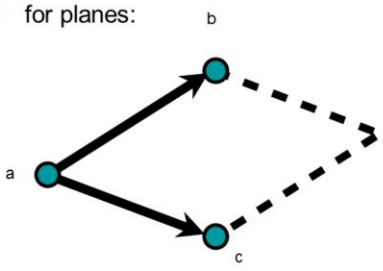
Polygons

- for lines:

points between p and q:

p + λ(q-p), λ from [0,1]

- for planes:



a + λ(b-a) + μ(c-a)

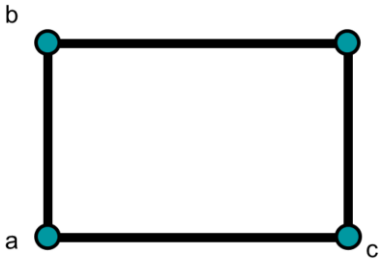
λ, μ from [0,1]

→ parallelogram

Any point on a line can be represented as p + λ(q-p), where the points between p and q have values from [0,1].
Similarly, all points inside a parallelogram defined by a, b and c can be represented as a + λ(b-a) + μ (c-a) for λ, μ from [0,1].
λ = μ = 0 is the point a
λ = μ = 1 is the unnamed point on the right of the parallelogram

Quads

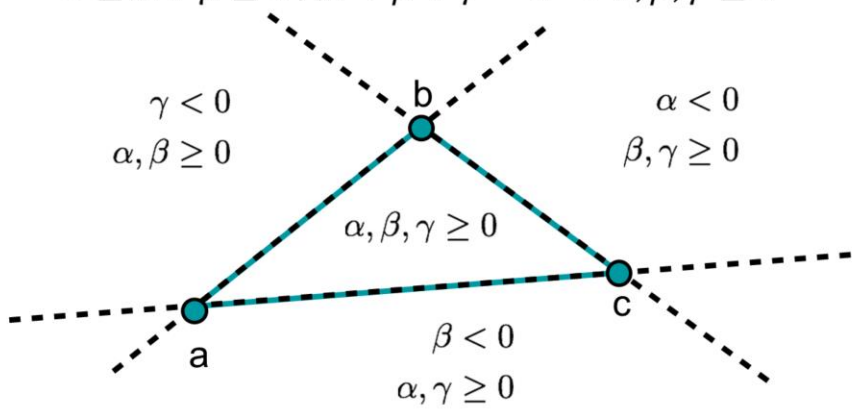
- Special case of parallelogram with $(b-a)^T(c-a) = 0$
- often used for post-processing / computation
- often more natural symmetries



In terms of barycentric coordinates, $0 \leq \alpha \leq 1$ and $0 \leq \gamma \leq 1$ holds for all points inside the quadrilateral.

Barycentric Coordinates for Triangles

$$0 \leq \lambda + \mu \leq 1 \wedge \alpha + \beta + \gamma = 1 \Rightarrow \alpha, \beta, \gamma \geq 0$$



barycentric coordinates unique iff triangle non-degenerate

Given three points a, b and c, all points with non-negative barycentric coordinates are inside the triangle defined by a, b and c. The edges of the triangle are the regions where at least one of the barycentric coordinates is 0.
A triangle defines the split of a plane into 7 sectors, where every sector is defined by the sign of the barycentric coordinates (if the three points are on one line, the triangle gets degenerated).

Overview

- Objects in 3D
- Mathematical Representations
- **Transformations**

Now that we know what points, lines, planes, triangles and quads are, we can continue by explaining how to transform that static geometry.

Linear Maps

Definition:

A mapping $L: \mathbb{R}^3 \longrightarrow \mathbb{R}^3$ is linear

$$:\Leftrightarrow$$

$$L(p+q) = L(p) + L(q)$$
$$L(\lambda p) = \lambda L(p)$$
$$\forall p, q \in \mathbb{R}^3, \forall \lambda \in \mathbb{R}$$

An important family of transformations are linear maps. The definition of a linear mapping with its two properties (additivity (= linearity) and homogeneity) can be seen above.
Since we are mostly using linear geometry (geometric objects formed by a linear combination of points), the linearity of the transformations and the linearity of the geometry combine nicely as we can compute the transformed object again as a linear combination of the transformed points.

Linear Maps

Matrix representation:

For each linear map L ∃ matrix $M \in \mathbb{R}^{3\times3}$ $s.t.$
$L(p) = Mp, \forall p \in \mathbb{R}^3$

Furthermore, linear maps have the important property that they can be represented by matrices. Therefore, applying a linear map reduces to a matrix multiplication, which is algorithmically very easy to evaluate.

Linear Maps

$$
\begin{aligned}
L(p) &= L(xe_1 + ye_2 + ze_3) \\
&= xL(e_1) + yL(e_2) + zL(e_3) \\
&= xf_1 + yf_2 + zf_3 \text{ with } f_i := L(e_i)
\end{aligned}
$$

$$
\Longrightarrow M = [f_1|f_2|f_3] = \begin{bmatrix} f_{1,x} & f_{2,x} & f_{3,x} \\ f_{1,y} & f_{2,y} & f_{3,y} \\ f_{1,z} & f_{2,z} & f_{3,z} \end{bmatrix}
$$

M is completely determined by $L(e_i)$.

Given a linear map, computing its matrix representation is straightforward. Remember that every point in 3D can be written as a linear combination of the basis element of some basis for the 3-dimensional space (here we use the standard basis e1 = [1,0,0]t, e2 = [0,1,0] t, e3 = [0,0,1] t). Due to the Additivity and Homogeneity properties of linear maps, we can therefore express the transformed point as a linear combination of the transformed basis elements.
We then obtain the matrix M corresponding to the linear map L by inserting the transformed basis vectors into the columns of M. Notice that this implies that the matrix M is completely determined by the images of the basis vectors.

Example: Rotation around X-axis

$$f_1 = e_1 = [1,0,0]^T$$
$$f_2 = [0, \cos\alpha, \sin\alpha]^T$$
$$f_3 = [0, -\sin\alpha, \cos\alpha]^T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

Consider the rotation around the x-axis as an example. Since we rotate around the x-axis, the x-value of a transformed point does not change, thus the x-axis is mapped onto itself. The other two axes are transformed according to the well-known formula for 2D rotation (cf. the definition of the sine and cosine on the unit circle). The corresponding transformation matrix can then be obtained as described on the previous slide.

Coordinate System Transformation

- $\{p_1, p_2, p_3\} \rightarrow \{q_1, q_2, q_3\}$???

$$
\begin{array}{lll}
[p_1|p_2|p_3] & : & p_i \longrightarrow e_i \\
[q_1|q_2|q_3] & : & q_i \longrightarrow e_i \\
[q_1|q_2|q_3]^{-1} \cdot [p_1|p_2|p_3] & : & p_i \longrightarrow q_i
\end{array}
$$

- p_i orthonormal basis:

$$[\, p_1 \mid p_2 \mid p_3 \,]^{-1} = [\, p_1 \mid p_2 \mid p_3 \,]^T$$

An important case for linear maps is when the matrix M corresponding to a linear map L has full rank (rank 3 for linear maps in 3D). In this case, the columns of M are linear independent. Since the columns of M correspond to the transformed basis vectors, these transformed vectors also form a basis for the 3-dimensional space. In this case, we can regard L as a basis change from the standard basis {e1, e2, e3} to a new basis {p1, p2, p3}.
If we want to do a basis change between two arbitrary bases {p1, p2, p3} and {q1, q2, q3}, we first compute the transformation matrices from the standard matrix to both new bases. Since those matrices have full rank, they are invertible and we can compute the basis change between pi and qi by first transforming the pi to ei and then the ei to qi.

Notice the important case where a basis pi is an orthonormal basis (all basis vectors have unit length and are pairwise orthogonal to each other) for which the inverse of the basis change matrix is identical to its transpose.

Special Linear Maps

- **Scaling**
- Shearing
- Rotation

$$\begin{bmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & \gamma \end{bmatrix}$$

An arbitrary 3x3 matrix will not necessarily have a geometric meaning. But there are some 3x3 matrices that are geometrically meaningfull, such a scaling, shearing and rotations.

Special Linear Maps

- Scaling
- **Shearing**
- Rotation

$$\begin{bmatrix} 1 & 0 & s \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix}$$

This is an example for a shearing matrix, which performs a shearing in the x- and y-direction.

Special Linear Maps

- Scaling
- Shearing
- Rotation
  - around axis n
  - by angle α

$$R_{n,\alpha} = \cos\alpha \cdot I + (1 - \cos\alpha) \cdot nn^t + \sin\alpha \cdot R_n$$

$$\text{with } \mathrm{R_n} = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix}$$

While rotating around one of the standard axes is simple, the formula to rotate a point around an arbitrary vector / axis n is a little bit more complex. The method for obtaining the corresponding rotation matrix Rn,α is known as Rodgrigues‘ rotation formula. I is the identity matrix and Rn * x is the matrix formulation of the cross product between the vectors n and x (i.e. Rn * x = n × x). A good exercise is to try to derive the formula yourself. Hint: Try to define a coordinate system relative to the axis n which you then rotate.

Affine Maps

- Translation is not linear, but affine
  - (image of origin)
- Affine map = linear map + translation
  A: $p \rightarrow L(p) + t$
- Matrix representation ?
  - → *extended coordinates*

Remember the homogeneity property of linear maps: L(a*p) = a*L(p)
As a consequence, L(0*p) = 0*L(p) = [0,0,0]t, linear maps leave the origin of the coordinate system unchanged and therefore cannot represent translations. Unfortunately, translation is an important transformation in computer graphics since we want to be able to move objects through a scene.
We therefore have to use affine maps, which include both linear maps and translations. Using only 3 coordinates (x,y,z), we obviously cannot represent an affine map by a matrix due to the translational part. Since we are still interested in such a representation (due to its ease of use), we therefore represent points and vectors by extended coordinates.

Points vs. Vectors

- Tupel representation ambigous
- vector = point – $[0\ 0\ 0]^T$
- Meaningful operations:
  - vector = vector + vector
  - point = point + vector
  - vector = point - point
  - ??? = point + point

Let us first look at points and vectors in R3 (3-dimensional space). Here, both vectors and points have the same representation as 3-dimensional elements, but different geometric meanings. A point is a position in space while a vector is an offset / direction between two positions. Therefore, not all combinations of vectors and points are geometrically meaningful.

Extended Coordinates

- Point: $[x\ y\ z\ \mathbf{1}]^T$
- Vector: $[x\ y\ z\ \mathbf{0}]^T$
- An operation is valid iff the result's last component is either **0** or **1**

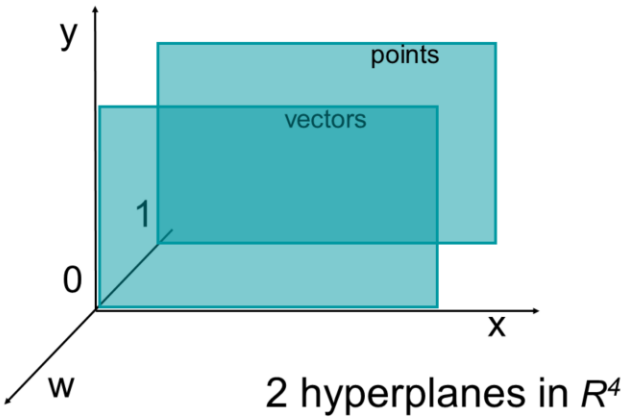
$\sum_i \alpha_i \cdot p_i$ with $\sum_i \alpha_i = 1$ ***affine*** combination

with $\sum_i \alpha_i = 1, \alpha_i \geq 0$ ***convex*** combination

Extended coordinates distinguish between points and vectors by using a fourth coordinate, which is set to 1 for points and 0 for vectors. Therefore, a operation on vectors and points is meaningful only if the fourth component of the result is either 0 (vector) or 1(point) (cf. the operations listed on the previous slide). This implies that a linear combination of points in extended coordinates are only meaningful if the weights sum up to 1 (affine combination).
For a point x given in barycentric α, β, γ coordinates for 3 points p,q,r in extended coordinates (i.e. x = αp + βq + γr) this holds true since the weights of the barycentric coordinates sum up to one, i.e. for the last component of x holds α + β + γ = 1.

Extended Coordinates

- Geometric interpretation:



Geometrically, we can interpret extended coordinates as two hyperplanes in R4, where one plane contains all 3D points and the other plane all 3D lines.

Affine Maps, Extended Coords

- Matrix representation using extended coordinates:

$$A = \left[\begin{array}{ccc|c} & L & & t \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \begin{bmatrix} L_{1,1} & L_{1,2} & L_{1,3} & t_x \\ L_{2,1} & L_{2,2} & L_{2,3} & t_y \\ L_{3,1} & L_{3,2} & L_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A very important property of extended coordinates is that they let us represent affine maps by matrices. The upper 3x3 part of the matrix thereby corresponds to the linear part of the affine map. The translation is represented in extended coordinates in the last row of the matrix. Note that the matrix A corresponding to the affine map does not translate vectors.

Concatenation of Transformations

- Sequence of affine maps: $A_1, A_2, \ldots, A_n$
- Concatenation by matrix multiplication
- Ordering ?
- Coordiate system
  - local coord system (transform camera)
  - global coord system (transform object)

Since we can now represent affine maps as matrices, concatenating a sequence of affine maps can be done by simple matrix multiplication. When doing so, the ordering of the multiplication is important as will be explaining on the next slide. Similarly the transformations that we use depend on our coordinate system. For example, using a local coordinate system for the camera means that we transform the camera while using a global coordinate system means transforming the objects located in the coordinate system. So if you are using affine transformations (or any transformation in that matter) you have to be certain about the coordinate system you are using since this determines the order of operations.

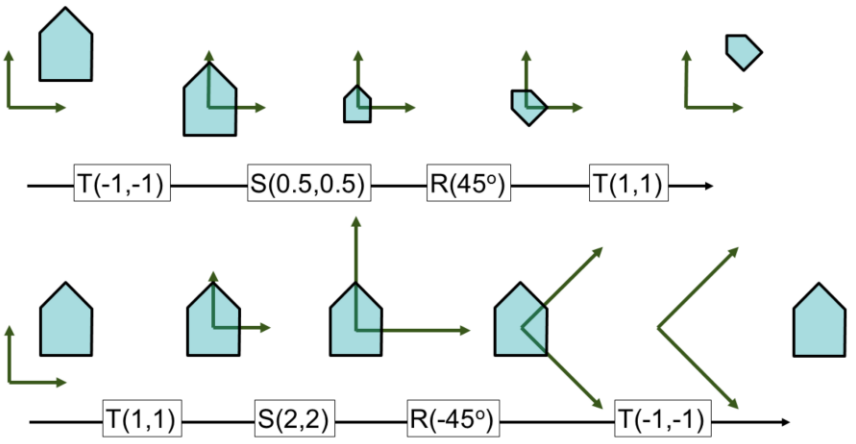
Concatenation of Transformations

- *Example:*
  - rotation and translation
  - translation and rotation
- *Example:* rotation around point *t*:

$$R_{t,n,\alpha} = R_{n,\alpha}(p - t) + t$$
$$= \left(T_t R_{n,\alpha} T_{-t}\right) \cdot p$$

Remember that a rotation is a linear transformation and leaves the origin unaffected. So if we want to rotate around a certain point, we first have to translate the point into the origin, rotate and then move the coordinate system back.

Local vs. Global System



Depending on the coordinate system we have to adjust our transformations to achieve a certain effect (in this case scaling the house).
There are two ways to achieve the same result: Keep the coordinate system fixed and transform the object (top) or keep the object fixed but transform the coordinate system (bottom).