

TIEA311

Tietokonegrafiikan perusteet

kevät 2019

(“Principles of Computer Graphics” – Spring 2019)

Copyright and Fair Use Notice:

The lecture videos of this course are made available for registered students only. Please, do not redistribute them for other purposes. Use of auxiliary copyrighted material (academic papers, industrial standards, web pages, videos, and other materials) as a part of this lecture is intended to happen under academic “fair use” to illustrate key points of the subject matter. The lecturer may be contacted for take-down requests or other copyright concerns (email: paavo.j.nieminen@jyu.fi).

TIEA311 Tietokonegrafiikan perusteet – kevät 2019 ("Principles of Computer Graphics" – Spring 2019)

Adapted from: *Wojciech Matusik*, and *Frédo Durand*: 6.837 Computer Graphics. Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, <https://ocw.mit.edu/>.

License: Creative Commons BY-NC-SA

Original license terms apply. Re-arrangement and new content copyright 2017-2019 by *Paavo Nieminen* and *Jarno Kansanaho*

Frontpage of the local course version, held during Spring 2019 at the Faculty of Information technology, University of Jyväskylä:

<http://users.jyu.fi/~nieminen/tgp19/>

TIEA311 - Today in Jyväskylä

Plan for today:

- ▶ Usual warm-up.
- ▶ Continue from yesterday
- ▶ Go through theory
- ▶ Remember to have a break!
- ▶ The teacher will try to remember and make use of the fact that we have groups of 2-3 students with pen and paper.

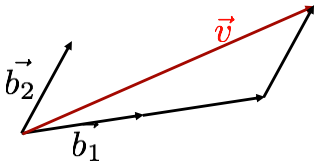
Vectors (linear space)

- Formally, a set of elements equipped with addition and scalar multiplication
 - plus other nice properties
- There is a special element, the zero vector
 - no displacement, no force

Vectors (linear space)

- We can use a **basis** to produce all the vectors in the space:

- Given n basis vectors \vec{b}_i
any vector \vec{v} can be written as
$$\vec{v} = \sum_i c_i \vec{b}_i$$



here:

$$\vec{v} = 2\vec{b}_1 + \vec{b}_2$$

Linear algebra notation

$$\vec{v} = c_1 \vec{b}_1 + c_2 \vec{b}_2 + c_3 \vec{b}_3$$

- can be written as

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

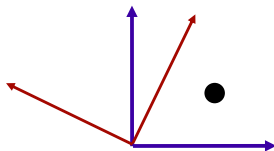
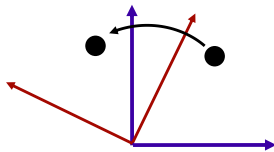
- Nice because it makes the basis (coordinate system) explicit
- Shorthand:

$$\vec{v} = \mathbf{\vec{b}}^t \mathbf{c}$$

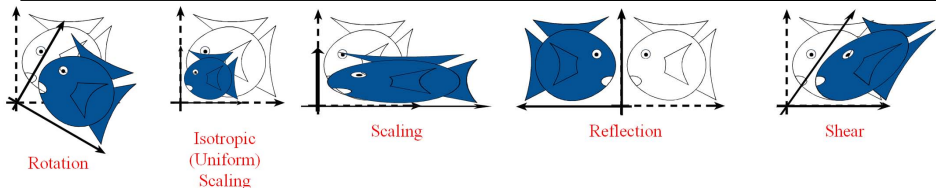
- where bold means triplet, t is transpose

Matrices have two purposes

- (At least for geometry)
- Transform things
 - e.g. rotate the car from facing North to facing East
- Express coordinate system changes
 - e.g. given the driver's location in the coordinate system of the car, express it in the coordinate system of the world



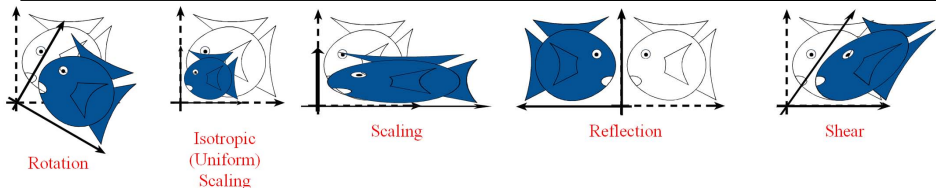
Linear transformation



Courtesy of Prof. Fredo Durand. Used with permission.

- Transformation \mathcal{L} of the vector space

Linear transformation



Courtesy of Prof. Fredo Durand. Used with permission.

- Transformation \mathcal{L} of the vector space so that

$$\mathcal{L}(\vec{v} + \vec{u}) = \mathcal{L}(\vec{v}) + \mathcal{L}(\vec{u})$$

$$\mathcal{L}(\alpha\vec{v}) = \alpha\mathcal{L}(\vec{v})$$

- Note that it implies $\mathcal{L}(\vec{0}) = \vec{0}$
- Notation $\vec{v} \Rightarrow \mathcal{L}(\vec{v})$ for transformations

Matrix notation

- Linearity implies

$$\mathcal{L}(\vec{v}) = \mathcal{L}\left(\sum_i c_i \vec{b}_i\right) = \quad ?$$

Matrix notation

- Linearity implies

$$\mathcal{L}(\vec{v}) = \mathcal{L}\left(\sum_i c_i \vec{b}_i\right) = \sum_i c_i \mathcal{L}(\vec{b}_i)$$

- i.e. we only need to know the basis transformation
- or in algebra notation

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} \Rightarrow \begin{bmatrix} \mathcal{L}(\vec{b}_1) & \mathcal{L}(\vec{b}_2) & \mathcal{L}(\vec{b}_3) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Algebra notation

- The $\mathcal{L}(\vec{b}_i)$ are also vectors of the space
- They can be expressed in the basis

...

Algebra notation

- The $\mathcal{L}(\vec{b}_i)$ are also vectors of the space
- They can be expressed in the basis for example:

$$\mathcal{L}(\vec{b}_1) = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} M_{1,1} \\ M_{2,1} \\ M_{3,1} \end{bmatrix}$$

- which gives us

$$\begin{bmatrix} \mathcal{L}(\vec{b}_1) & \mathcal{L}(\vec{b}_2) & \mathcal{L}(\vec{b}_3) \end{bmatrix} = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{bmatrix}$$

Recap, matrix notation

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

- Given the coordinates \mathbf{c} in basis $\vec{\mathbf{b}}$
the transformed vector has coordinates $M\mathbf{c}$ in $\vec{\mathbf{b}}$

Example 1

Just one example of a linear transformation matrix useful in graphics – Counter-clockwise rotation of θ radians around the z -axis (pointing towards the viewer when right-handed coordinates are used):

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

How to remember / understand this? Take a pen, draw a unit circle on the xy -plane, and recall basic trigonometry from school times (or Wikipedia. . .)! The “canonical” basis vectors $[1, 0, 0]^t$ and $[0, 1, 0]^t$ must rotate along the unit circle to the expected new positions.

Really, do it, if you haven't already!

Example 2

Another example of a linear transformation matrix useful in graphics – Scaling of axes:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

How to remember / understand this? Take a pen, draw a unit box or some other simple shape, and see how different values of s_x, s_y , and s_z make isotropic and anisotropic scalings.

Really, do it, if you haven't already!

Further Examples

Look at the implementation of the `Matrix3f` class in our example codes, found in the files `Matrix3f.cpp` and `Matrix3f.h`

Make sure you **understand the implementation** of rotation and scaling matrices, and the overloaded **operators** for matrix-vector multiplication and matrix-matrix multiplication.

Really, do it, if you haven't already!

Also, see what else the class provides and how it all looks in C++. How is the code split in the header (.h) and implementation (.cpp). **Learn to use your IDE to navigate** the files easily!

Example: Inverse transforms

`Matrix3f.cpp` implements determining (by computing the “determinant”) if an **inverse matrix** exists, and a formula for inverting an invertible matrix. It is important to understand the *concept* of the inverse transform ($M^{-1}M = MM^{-1} = I$). For most of our graphics transforms, we know the inverses explicitly (understand and verify):

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

BTW: The last identity means that the inverse **in this case** is the transpose of the original. How do we get that? (1) By symmetry properties of the trigonometric functions (draw it to believe) but also (2) they teach us in linear algebra courses that **this is true for any real-valued matrix that is “orthonormal”, i.e., keeps orthogonality and distances the same**. This makes some inverses in graphics and other computation tasks trivial, maximally accurate, and blazingly fast!

Another inverse

The inverse of scaling is easy to figure out:

$$S^{-1}(s_x, s_y, s_z) = S(1/s_x, 1/s_y, 1/s_z) = \begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1/s_z \end{bmatrix}$$

This one is **not** the same as S^T because scaling is not orthonormal (orthogonal yes, but not normal, i.e., it does not preserve lengths).

Remember the supertools

Your Super Tools: the Brain, the Pen and the Paper. Teacher's own example from 2017:

4. Re-think from start, piece by piece

$\vec{a}^T d = \vec{f}^T e = \vec{f}^T A d$

$\vec{f}^T S e = \vec{f}^T S A d$

$= \vec{a}^T A^{-1} S A d$

2. Concrete example

ASUNTO	KRS	TYYPPI	M ²	MH €	VH €
2	2h+kt	43	38 100	149 900	
4	2h+kt+s	50	54 600	185 900	
4	3h+kt+s	60	55 900	211 900	

$f^T = [f_1 \ f_2 \ \tilde{f}]$

$a^T = [a_1 \ a_2 \ \tilde{a}]$

3. Add numbers to make it even more concrete

$\vec{c} = \begin{pmatrix} 100 \\ 0 \\ 10 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

$\vec{f}^T = \begin{pmatrix} 100 \\ 0 \\ 10 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

1. Wrong fixated idea!

~~$f^T c = a^T A c$~~

5. Enlightenment: fixed mental image now matches the equations that were to be verified / sanity-checked

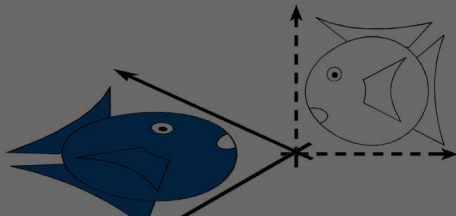
Why do we care

- We like linear algebra
- It's always good to get back to an abstraction that we know and for which smarter people have developed a lot of tools
- But we also need to keep track of what basis/coordinate system we use

Linear Transformations

$$\bullet L(p + q) = L(p) + L(q)$$

$$\bullet L(ap) = a L(p)$$



Translation is not linear:

$$f(p) = p+t$$

$$f(ap) = ap+t \neq a(p+t) = a f(p)$$

$$f(p+q) = p+q+t \neq (p+t)+(q+t) = f(p) + f(q)$$

Affine space

- Points are elements of an affine space
- We denote them with a tilde \tilde{p}

- Affine spaces are an extension of vector spaces

Point-vector operations

- Subtracting points gives a vector

$$\tilde{p} - \tilde{q} = \vec{v}$$

- Adding a vector to a point gives a point

$$\tilde{q} + \vec{v} = \tilde{p}$$

Frames

- A frame is an origin \tilde{o} plus a basis $\vec{\mathbf{b}}$
- We can obtain any point in the space by adding a vector to the origin

$$\tilde{p} = \tilde{o} + \sum_i c_i \vec{b}_i$$

- using the coordinates \mathbf{c} of the vector in $\vec{\mathbf{b}}$

Algebra notation

- We like matrix-vector expressions
- We want to keep track of the frame
- We're going to cheat a little for elegance and decide that 1 times a point is the point

$$\tilde{p} = \tilde{o} + \sum_i c_i \vec{b}_i = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix} = \vec{f}^t \mathbf{c}$$

- \tilde{p} is represented in \vec{f} by 4 coordinate, where the extra dummy coordinate is always 1 (for now)

Further Examples (Affine transforms)

Look at the implementation of the `Matrix4f` class in our example codes, found in the files `Matrix4f.cpp` and `Matrix4f.h`

Really, do it. (And keep doing it all the time!)

Observe how the fourth coordinate is used to implement 3D frames and affine transforms of points. And linear transforms of directions. A straightforward way to do many things is to build a proper 4x4 matrix and then multiply. **Not much code**, actually!

In Assignment 1 you will **avoid a lot of tears** by figuring out how (and when and why) to use the provided constructors and the operator `*` to transform points and frames suitably.

Also, this stuff **is a key thing** in **computer graphics**!

TIEA311 - Today in Jyväskylä

The time allotted for this week's graphics lectures is now over.

Next lecture happens in 6 days and 4 hours.

The teacher will now tell his view about what **could be useful activities** for you during that time period.

→ see lecture video.

Make notes, if you have to.

Even if he forgets to say it, **remember to rest, too!**