TIEA311 Tietokonegrafiikan perusteet

kevät 2017

(“Principles of Computer Graphics” – Spring 2017)

TIEA311 Tietokonegrafiikan perusteet – kevät 2017 ("Principles of Computer Graphics" – Spring 2017)

Frontpage of the local course version, held during Spring 2017 at the Faculty of Information technology, University of Jyväskylä: `http://users.jyu.fi/~nieminen/tgp17/`

Assignment 1 etc: how to proceed

- Read instructions
- Start early
- Reflect against the theory slides
- Disregard the “start from scratch” hints! We don’t have time for **that much** pain – we’ll have enough, just **modifying the starter codes**!
- Ask questions when you arrive to useful ones!
- Start early

Recap

- Vectors can be expressed in a basis
  - Keep track of basis with left notation $\vec{v} = \vec{\mathbf{b}}^t \mathbf{c}$
  - Change basis $\vec{v} = \vec{\mathbf{a}}^t M^{-1} \mathbf{c}$
- Points can be expressed in a frame (origin+basis)
  - Keep track of frame with left notation
  - adds a dummy 4th coordinate always 1

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

Linear component

$$\tilde{p} = \tilde{o} + \sum_i c_i \vec{b_i} = \begin{bmatrix} \vec{b_1} & \vec{b_2} & \vec{b_3} & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

$$\Longrightarrow$$

$$\tilde{o} + \sum_i c_i \mathcal{L}(\vec{b_i}) = \begin{bmatrix} \vec{b_1} & \vec{b_2} & \vec{b_3} & \tilde{o} \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & 0 \\ M_{21} & M_{22} & M_{23} & 0 \\ M_{31} & M_{32} & M_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

- Note how we leave the fourth component alone

Translation component

$$\tilde{p} \Rightarrow \tilde{p} + \vec{t}$$

- Express translation vector t in the basis

$$\vec{t} = \begin{bmatrix} \vec{b_1} & \vec{b_2} & \vec{b_3} \end{bmatrix} \begin{bmatrix} M_{14} \\ M_{24} \\ M_{34} \end{bmatrix}$$

Translation

$$\tilde{p} = \tilde{o} + \sum_i c_i \vec{b_i} = \begin{bmatrix} \vec{b_1} & \vec{b_2} & \vec{b_3} & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

$$\Longrightarrow$$

$$\tilde{o} + \vec{t} + \sum_i c_i \vec{b_i} = \begin{bmatrix} \vec{b_1} & \vec{b_2} & \vec{b_3} & \tilde{o} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & M_{14} \\ 0 & 1 & 0 & M_{24} \\ 0 & 0 & 1 & M_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

Full affine expression

$$\tilde{p} = \tilde{o} + \sum_i c_i \vec{b_i} = \begin{bmatrix} \vec{b_1} & \vec{b_2} & \vec{b_3} & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

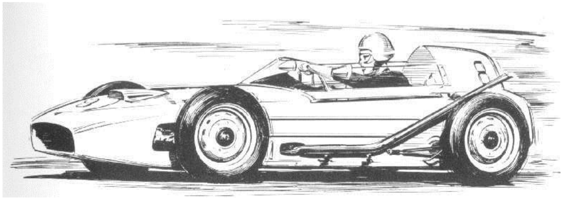
$$\Longrightarrow$$

$$\tilde{o} + \vec{t} + \sum_i c_i \mathcal{L}(\vec{b_i}) = \begin{bmatrix} \vec{b_1} & \vec{b_2} & \vec{b_3} & \tilde{o} \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

Which tells us both how to get a new frame ftM
or how to get the coordinates Mc after transformation

Frames & hierarchical modeling

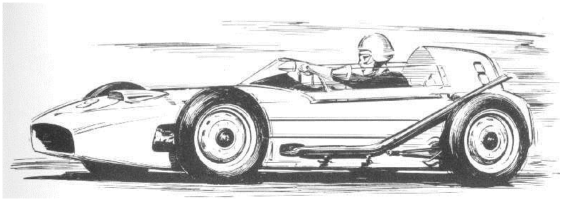
- Many coordinate systems (frames):
  - Camera
  - Static scene
  - car
  - driver
  - arm
  - hand
  - ...



- Need to understand nested transformations

Frames & hierarchical modeling

- Example: what if I rotate the wheel of the moving car:
- frame 1: world
- frame 2: car
- transformation: rotation



Frames & transformations

- Transformation S wrt car frame f

$$\tilde{p} = \vec{\mathbf{f}}^t \mathbf{c} \Rightarrow \vec{\mathbf{f}}^t S \mathbf{c}$$

- how is the world frame a affected by this?
- *we have* $\vec{a}^t = \vec{f}^t A \quad \vec{\mathbf{f}}^t = \vec{\mathbf{a}}^t A^{-1}$
- *which gives* $\vec{a}^t A^{-1} \Rightarrow \vec{a}^t A^{-1} S$

$$\vec{a}^t \Rightarrow \vec{a}^t A^{-1} S A$$

- *i.e. the transformation in a is A-1SA*
- *i.e., from right to left, A takes us from a to f, then we apply S, then we go back to a with A-1*

Regarding the “left notation” and changing frames, the local lecturer of TIEA311 was dealing with something he hadn’t really used before.

Questions he had to **ask himself**: How is the math being used here, conceptually? How are the computations done? How does this relate to what he had learned before (on previous instantiations of the present local course and on Linear Algebra of the math dept.)?

Specifically, the order of A and A^{-1} and what it means to “move from a frame to another” were puzzling. How should we interpret “moving” from $\vec{\mathbf{f}}$ to $\vec{\mathbf{a}}$?

Were the slides correct (always possible to contain mistakes)? And assuming they were (which is more likely), what part of the concept did he not yet fully understand?

So… what did the lecturer have to do in order to understand?

(guesses, anyone?)

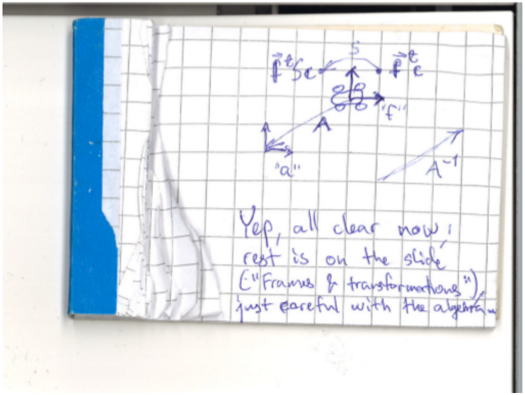
Take a **paper** and a **pen**, and use a **simple, concrete example** to **verify** that the equations **match the mental image**.

This time, it turns out that also the mental image needed to be adjusted (not much, but a little). This is called **learning**. It is painful, takes time, requires necessary tools (perhaps unique for everyone?), and then rewards.

Basic stuff. On the following slides, some scribbles from along the way.

Example: Back and forth between frames?

Pen and paper to help the brain (world + car + local origins and basis vectors + a point rotating in car frame):

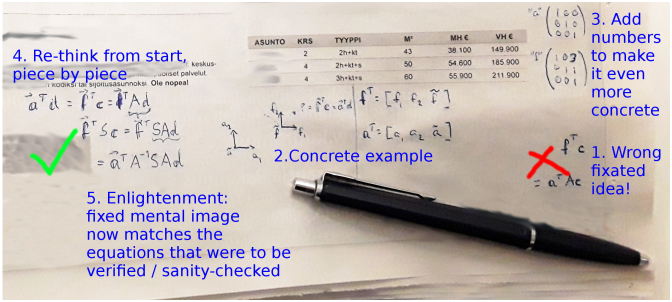


Yep, all clear now!
rest is on the slide
("Frames & transformations"),
just careful with the algebra..

Well, not yet enough.. provided only a momentary enlightenment that faded away overnight. . . followed an old fixation in thinking.

Example: Back and forth between frames!

More paper with whitespace, possibly same pen (very nice Ballograf one), re-start after thinking carefully about the slides covered on previous lectures:


4. Re-think from start, piece by piece
$\vec{a}^T d = \vec{f}^T c = \vec{f}^T A d$
$\vec{f}^T S c = \vec{f}^T S A d$
$= \vec{a}^T A^{-1} S A d$
5. Enlightenment: fixed mental image now matches the equations that were to be verified / sanity-checked
2.Concrete example
$f^T = [f_1 \; f_2 \; \tilde{f}]$
$a^T = [a_1 \; a_2 \; \tilde{a}]$
3. Add numbers to make it even more concrete
1. Wrong fixated idea!
$f^T c$
$= a^T A c$

Finally, a corrected mental model of what is “a frame”, and “keeping track of the frame” as defined by the OCW slides. You **must** do this kind of stuff by **yourself** – in your **own way**!

(If you learn without, I think you have **superpowers** and should **go fight hostile aliens**, not waste those powers on IT studies)

Frames & transformations

- Transformation S wrt car frame f

$$\tilde{p} = \vec{\mathbf{f}}^t \mathbf{c} \Rightarrow \vec{\mathbf{f}}^t S \mathbf{c}$$

- how is the world frame a affected by this?
- *we have* $\vec{a}^t = \vec{f}^t A \quad \vec{\mathbf{f}}^t = \vec{\mathbf{a}}^t A^{-1}$
- *which gives* $\vec{a}^t A^{-1} \Rightarrow \vec{a}^t A^{-1} S$

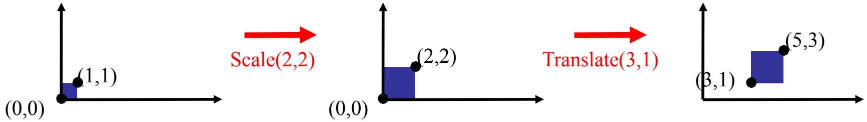
$$\vec{a}^t \Rightarrow \vec{a}^t A^{-1} S A$$

- *i.e. the transformation in a is A-1SA*
- *i.e., from right to left, A takes us from a to f, then we apply S, then we go back to a with A-1*

Questions?

How are transforms combined?

Scale then Translate



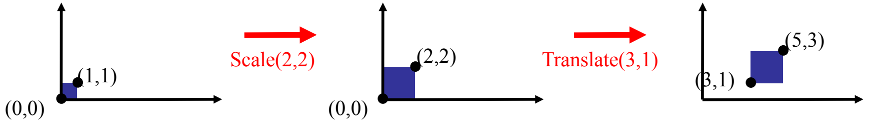
Use matrix multiplication: p' = T (S p) = TS p

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

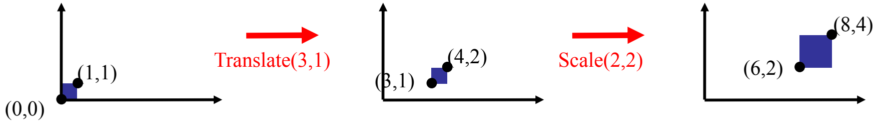
Caution: matrix multiplication is NOT commutative!

Non-commutative Composition

Scale then Translate: $p' = T(Sp) = TSp$



Translate then Scale: $p' = S(Tp) = STp$



Non-commutative Composition

Scale then Translate: p' = T (S p) = TS p

$$TS = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Translate then Scale: p' = S (T p) = ST p

$$ST = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

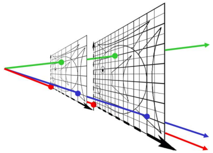
Questions?

Plan

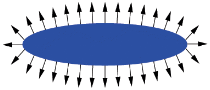
- Vectors

- Points

- **Homogenous coordinates**



- Normals



Forward reference and eye

- The fourth coordinate is useful for perspective projection
- Called homogenous coordinates

Homogeneous Coordinates

- Add an extra dimension (same as frames)
  - in 2D, we use 3-vectors and 3 x 3 matrices
  - In 3D, we use 4-vectors and 4 x 4 matrices
- The extra coordinate is now an **arbitrary** value, w
  - You can think of it as "scale," or "weight"
  - For all transformations except perspective, you can just set *w*=1 and not worry about it

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Projective Equivalence

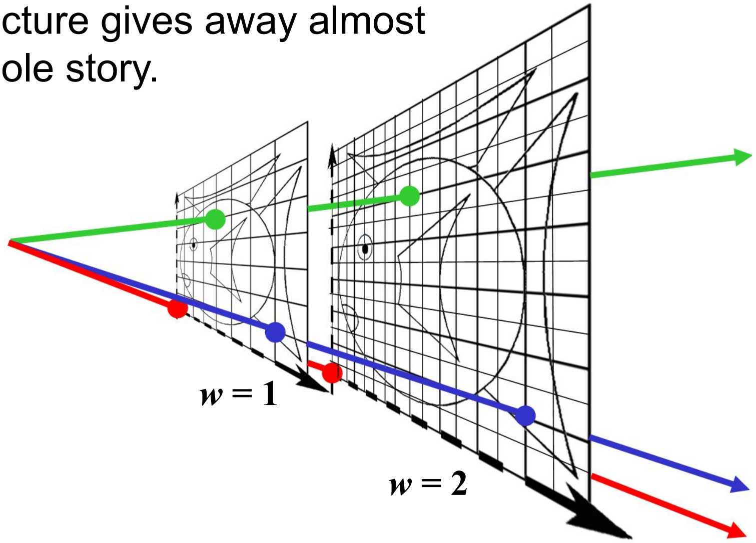
- All non-zero scalar multiples of a point are considered identical
- to get the equivalent Euclidean point, divide by *w*

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az \\ aw \end{bmatrix} \overset{w\,!=0}{=} \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$$

$$a \;\; != 0$$

Why bother with extra coord?

- This picture gives away almost the whole story.



BEWARE: A very lost lecturer babbling BS!

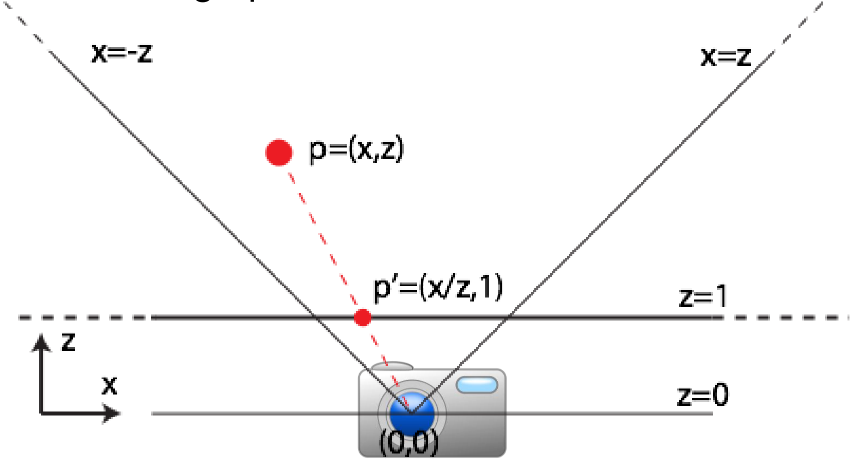
For the perspective projection part (next slides up to the SIGGRAPH paper title), the Finnish spoken “explanation” on the 7th lecture of TIEA311 Spring 2017 (Mon, Jan 30) is **mostly rubbish**. The slides, in turn, are **brilliant** in explaining the perspective projection, but your lecturer hit his own internal limit of continuous explanation capacity, destroying the clarity.

Flatland reference was OK, and all the “floating” at the distance of $1\ldots$ but it is of course the z (“depth”) coordinate that divides the other 2 coordinates in the 3D perspective projection (and the only other, x, coordinate in the 2D conceptual example shown here). It happens conveniently via copying the z coordinate to w (with the matrix given) and then coming back to the Euclidean equivalent coordinates $w = 1$ by division by w, which has been made equal to z by the copy!

Try to decipher it from the slides – we’ll start the next lecture by correcting the story.

Perspective in 2D

- Camera at origin, looking along *z*, 90 degree f.o.v., “image plane” at *z*=1

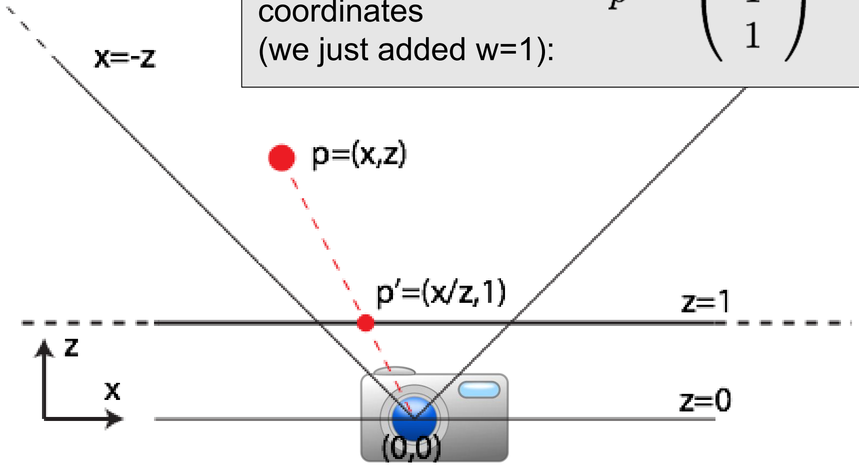


This image is in the public domain.
Source: http://openclipart.org/detail/34051/digicam-by-thesaurus.

Perspective in 2D

The projected point in homogeneous coordinates (we just added w=1):

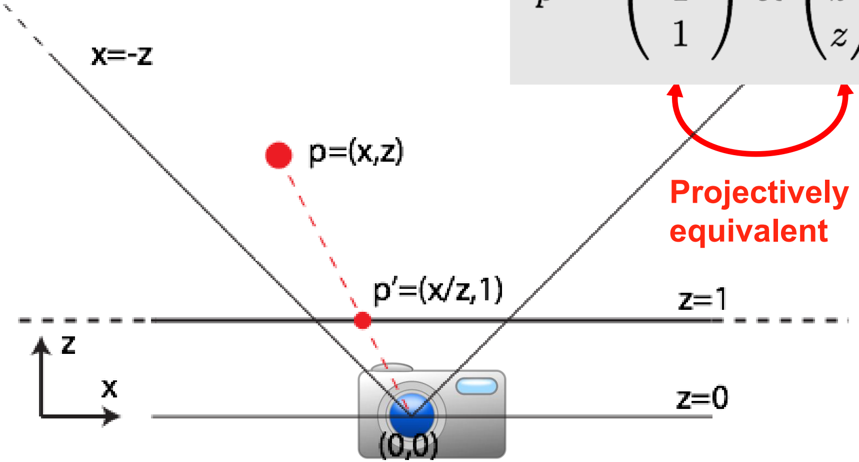
$$p' = \begin{pmatrix} x/z \\ 1 \\ 1 \end{pmatrix}$$



This image is in the public domain.
Source: http://openclipart.org/detail/34051/digicam-by-thesaurus.

Perspective in 2D

$$p' = \begin{pmatrix} x/z \\ 1 \\ 1 \end{pmatrix} \propto \begin{pmatrix} x \\ z \\ z \end{pmatrix}$$

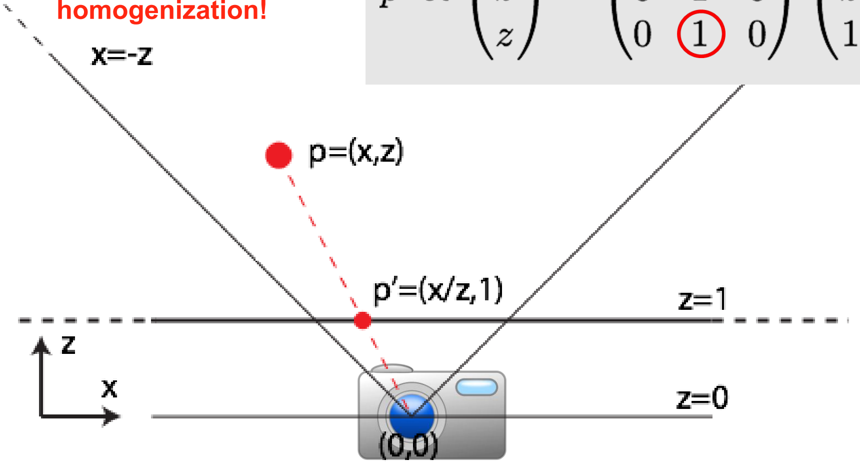


This image is in the public domain.
Source: http://openclipart.org/detail/34051/digicam-by-thesaurus.

Perspective in 2D

We'll just copy z to w, and get the projected point after homogenization!

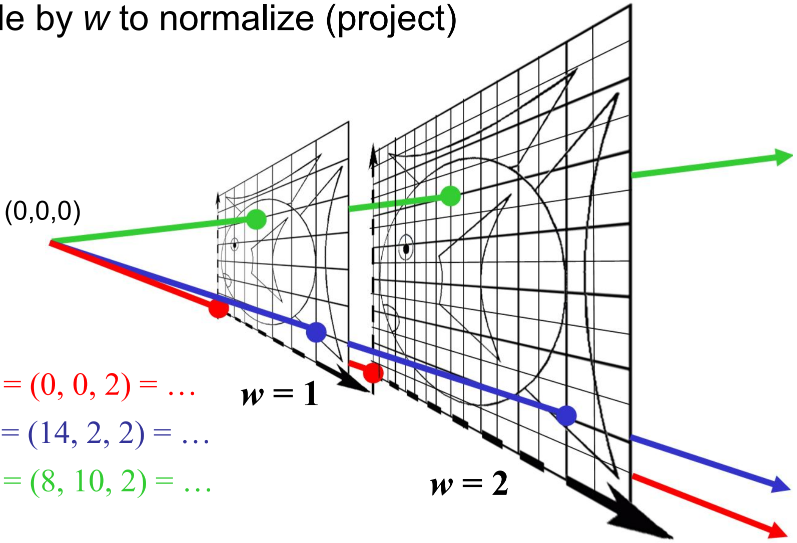
$$p' \propto \begin{pmatrix} x \\ z \\ z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \enclose{circle}{1} & 0 \end{pmatrix} \begin{pmatrix} x \\ z \\ 1 \end{pmatrix}$$



This image is in the public domain.
Source: http://openclipart.org/detail/34051/digicam-by-thesaurus.

Homogeneous Visualization

- Divide by *w* to normalize (project)



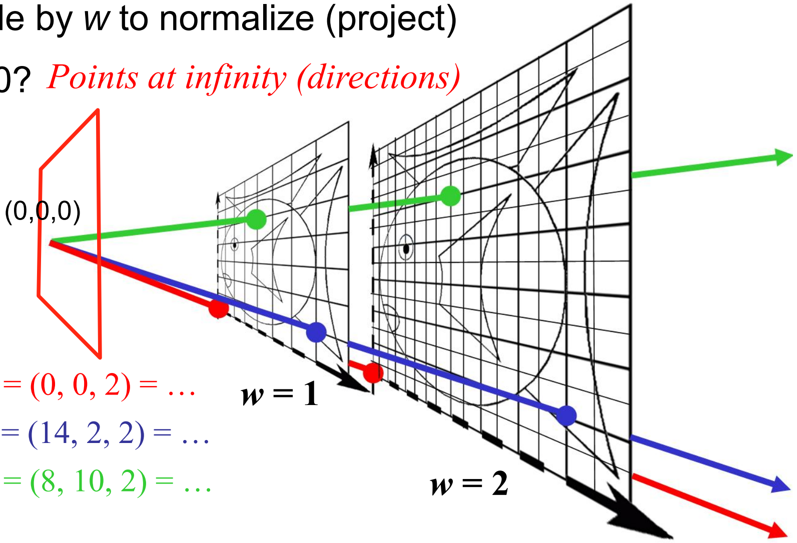
(0, 0, 1) = (0, 0, 2) = …

(7, 1, 1) = (14, 2, 2) = …

(4, 5, 1) = (8, 10, 2) = …

Homogeneous Visualization

- Divide by *w* to normalize (project)
- *w* = 0? *Points at infinity (directions)*



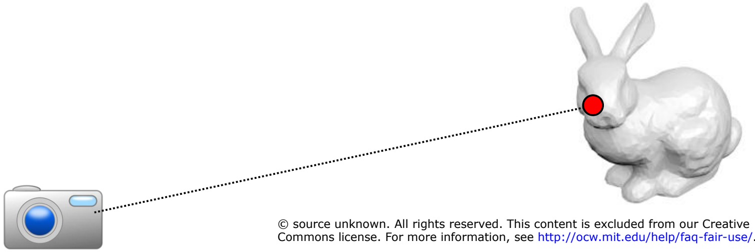
(0, 0, 1) = (0, 0, 2) = …

(7, 1, 1) = (14, 2, 2) = …

(4, 5, 1) = (8, 10, 2) = …

Projective Equivalence – Why?

- For affine transformations, adding w=1 in the end proved to be convenient.
- The real showpiece is **perspective.**



Questions?

Eye candy: photo tourism

- Application of homogenous coordinates
- Goal: given N photos of a scene
  - find where they were taken
  - get 3D geometry for points in the scene

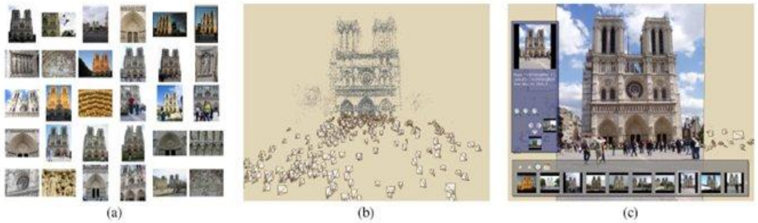


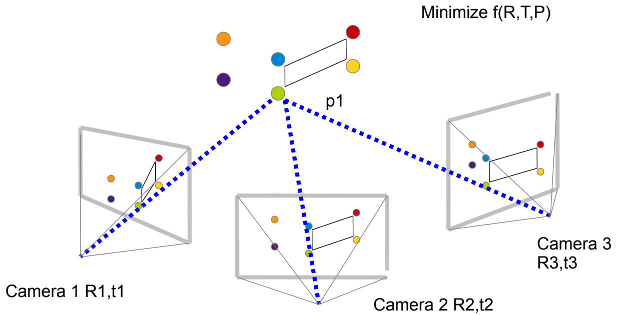
Figure 1: Our system takes unstructured collections of photographs such as those from online image searches (a) and reconstructs 3D points and viewpoints (b) to enable novel ways of browsing the photos (c).

Step 1: point correspondences

- Extract salient points (corners) from images
- Find the same scene point in other images
- To learn how it’s done, take 6.815

Structure from motion

- Given point correspondences
- Unknowns: 3D point location, camera poses
- For each point in each image, write perspective equations



Eye candy: photo tourism

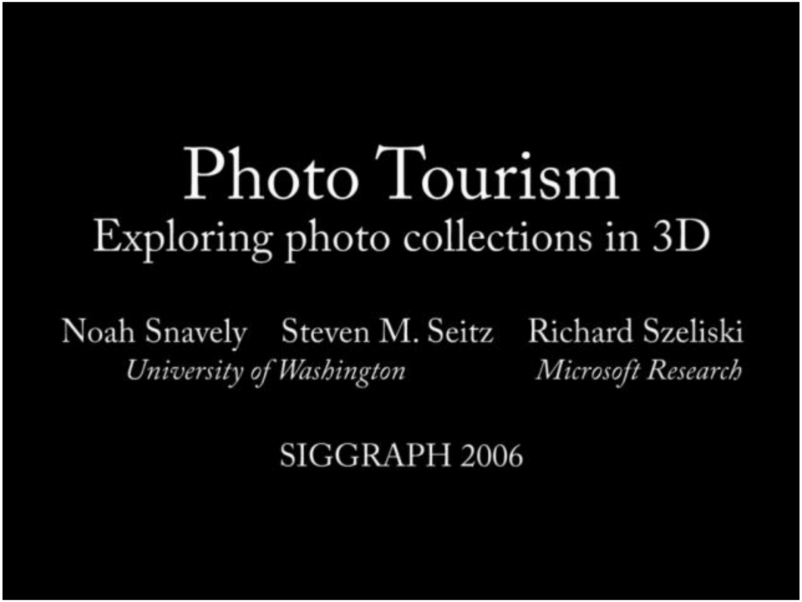


Photo Tourism

Exploring photo collections in 3D

Noah Snavely Steven M. Seitz Richard Szeliski

University of Washington *Microsoft Research*

SIGGRAPH 2006

BS-WARNING OVER: Lecturer woke up from his coma

The rest of the 7th lecture talk was less rubbish.

What happened just before, is a great example of what is likely to happen when you think “this is all very clear, I don’t have to recapitulate this before presenting it to others”

We learn from our mistakes? Again and again.

Carry on. . .

6.837 Computer Graphics Hierarchical Modeling

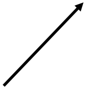
Wojciech Matusik, MIT EECS

Some slides from BarbCutler & Jaakko Lehtinen

Different objects

- **Points**
  - represent locations

- **Vectors**
  - represent movement, force, displacement from A to B



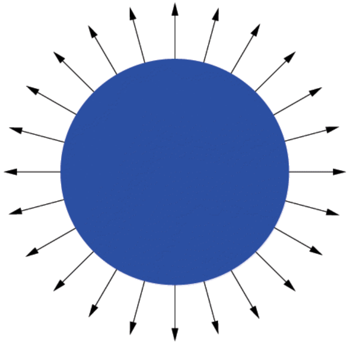
- **Normals**
  - represent orientation, unit length

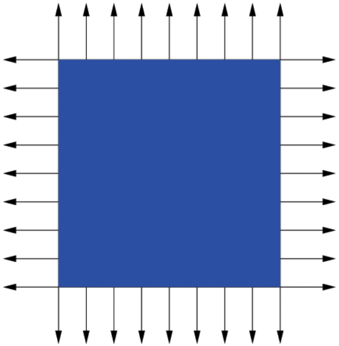
- **Coordinates**
  - numerical representation of the above objects **in a given coordinate system**

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Normal

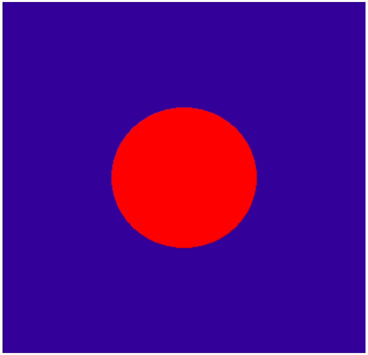
- Surface Normal: unit vector that is locally perpendicular to the surface

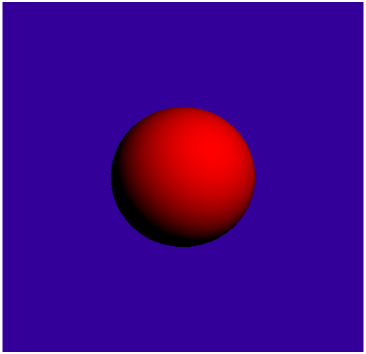




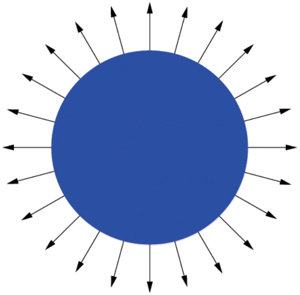
Why is the Normal important?

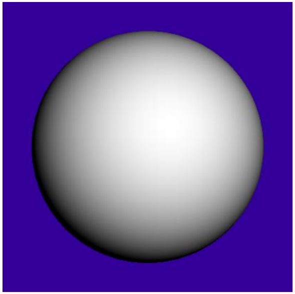
- It's used for shading — makes things look 3D!

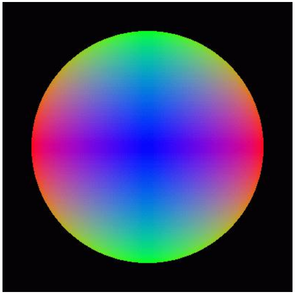

object color only


Diffuse Shading

Visualization of Surface Normal





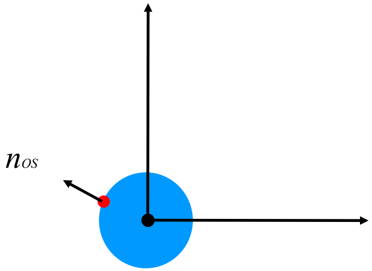


$\pm x$ = Red

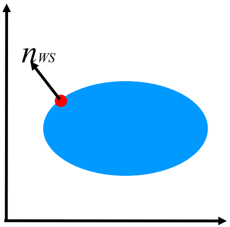
$\pm y$ = Green

$\pm z$ = Blue

How do we transform normals?



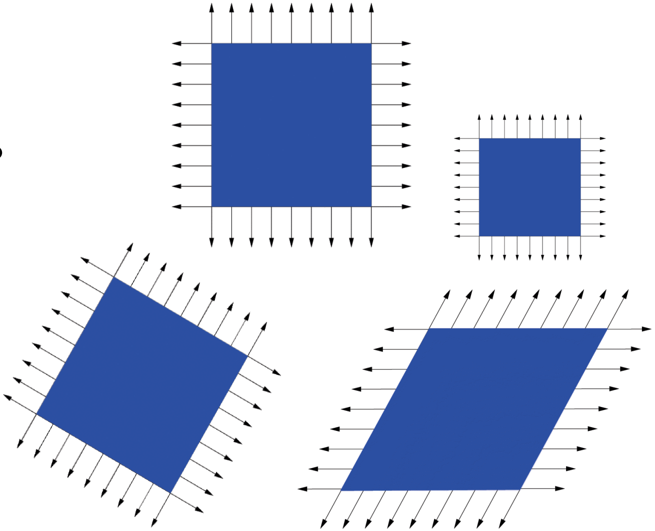
Object Space



World Space

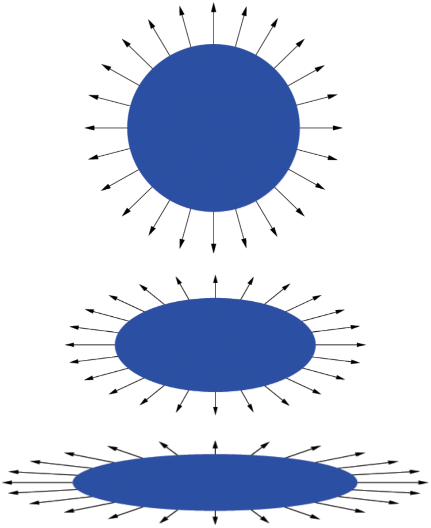
Transform Normal like Object?

- translation?
- rotation?
- isotropic scale?
- scale?
- reflection?
- shear?
- perspective?



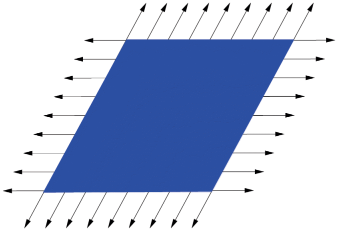
Transform Normal like Object?

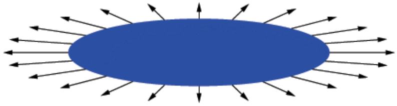
- translation?
- rotation?
- isotropic scale?
- scale?
- reflection?
- shear?
- perspective?



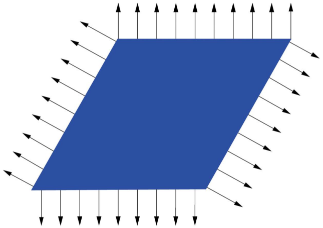
Transformation for shear and scale

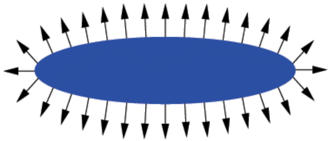
Incorrect
Normal
Transformation



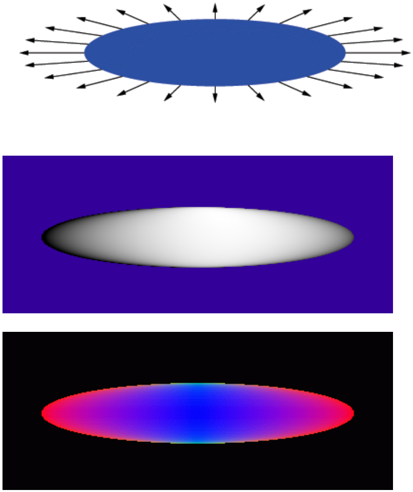


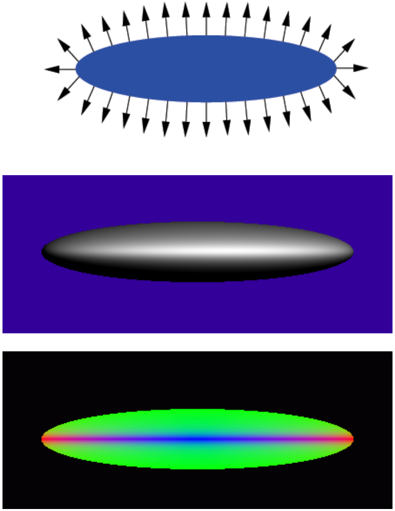
Correct
Normal
Transformation





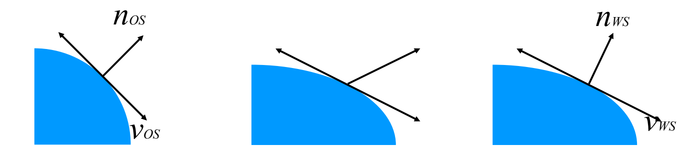
More Normal Visualizations


Incorrect Normal Transformation


Correct Normal Transformation

So how do we do it right?

- Think about transforming the *tangent plane* to the normal, not the normal *vector*



Original Incorrect Correct

Pick any vector *vOS* in the tangent plane, how is it transformed by matrix **M**?

$$v_{WS} = \mathbf{M}\ v_{OS}$$

Transform tangent vector *v*

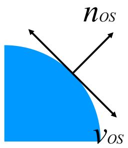
v is perpendicular to normal *n*:

Dot product $n_{OS}^{\mathsf{T}} \, v_{OS} = 0$

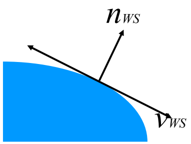
$$n_{OS}^{\mathsf{T}} \, (\mathbf{M}^{-1} \; \mathbf{M}) \; v_{OS} = 0$$

$$(n_{OS}^{\mathsf{T}} \; \mathbf{M}^{-1}) \; (\mathbf{M} \;\; v_{OS}) = 0$$

$$(n_{OS}^{\mathsf{T}} \; \mathbf{M}^{-1}) \; v_{WS} = 0$$



v_{WS} is perpendicular to normal n_{WS}:



$$n_{WS}^{\mathsf{T}} = n_{OS}^{\mathsf{T}} \, (\mathbf{M}^{-1})$$

$$\boxed{n_{WS} = (\mathbf{M}^{-1})^{\mathsf{T}} \, n_{OS}}$$

$$n_{WS}^{\mathsf{T}} \, v_{WS} = 0$$

Digression

$$\boxed{n_{WS} = (\mathbf{M}^{-1})^{\mathsf{T}}\, n_{OS}}$$

- The previous proof is not quite rigorous; first you’d need to prove that tangents indeed transform with **M**.
  - Turns out they do, but we’ll take it on faith here.
  - If you believe that, then the above formula follows.

Comment

- So the correct way to transform normals is:

$$n_{WS} = (\mathbf{M}^{-1})^{\mathsf{T}}\, n_{OS}$$

Sometimes denoted $\mathrm{M}^{-\mathrm{T}}$

- But why did $n_{WS} = \mathbf{M}\, n_{OS}$ work for similitudes?
- Because for similitude / similarity transforms,

$$(\mathbf{M}^{-1})^{\mathsf{T}} = \lambda\, \mathbf{M}$$

- e.g. for orthonormal basis:

$$\mathbf{M}^{-1} = \mathbf{M}^{\mathsf{T}} \quad \text{i.e. } (\mathbf{M}^{-1})^{\mathsf{T}} = \mathbf{M}$$

Connections

- Not part of class, but cool
  - “Covariant”: transformed by the matrix
    - e.g., tangent
  - “Contravariant”: transformed by the inverse transpose
    - e.g., the normal
    - a normal is a “co-vector”
- Google “differential geometry” to find out more

- Further Reading
  - Buss, Chapter 2

- Other Cool Stuff
  - Algebraic Groups
  - http://phototour.cs.washington.edu/
  - http://phototour.cs.washington.edu/findingpaths/
  - Free-form deformation of solid objects
  - Harmonic coordinates for character articulation

Question?

Assignment 0 aftermath: How to do it “right”

- The “right way” to implement OBJ reading?
- In programming, “the **best way**” is somewhat **ill-defined**.
- All you can (should) do is **better than your earlier code**.
- **One measure** of a “**good enough** way” is that the code is successfully used in real products
- But never forget **safety** (as the most important measure), **readability**, **maintainability**, **performance** (which only matters in selected places! Almost never “comes first”!)…
- That said, two alternatives from real software to read OBJ meshes:
  - `https://github.com/openscenegraph/OpenSceneGraph/blob/master/src/osgPlugins/obj/obj.cpp`
  - `https://github.com/davll/ICG_SSAO/tree/master/source/nv`
- (The former is properly open source; about the latter I’m not certain - it seems to originate in some Nvidia SDK …)

And that’s it for today

- The rest on Thursday