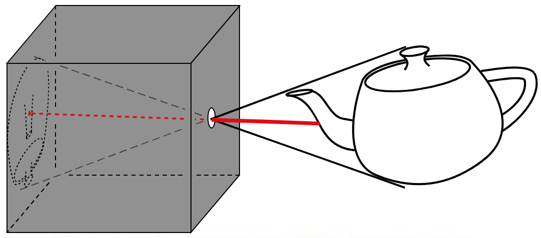
The Graphics Pipeline

One for all and all for one

Elmar Eisemann
Delft University of Technology (TU Delft)

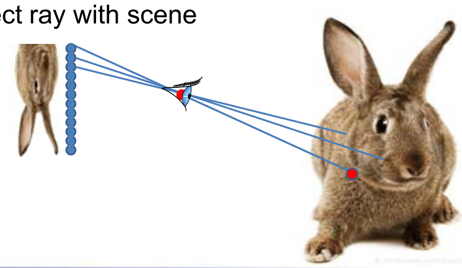
Pinhole camera

- Box with hole
- Perfect image for "point-sized" hole



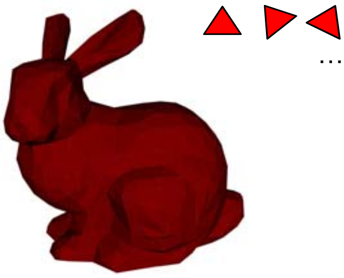
Virtual Camera

- Take a pixel on the image in the virtual world
- Compute ray through pixel and camera center
- Intersect ray with scene



Scene Rerpresentation

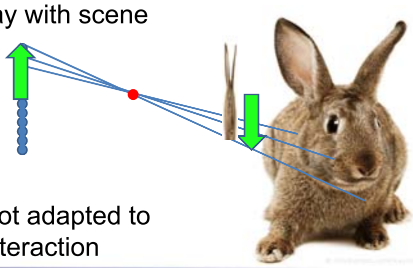
- Models are typically lists of triangles



Easy to compute ray intersection

Virtual Camera

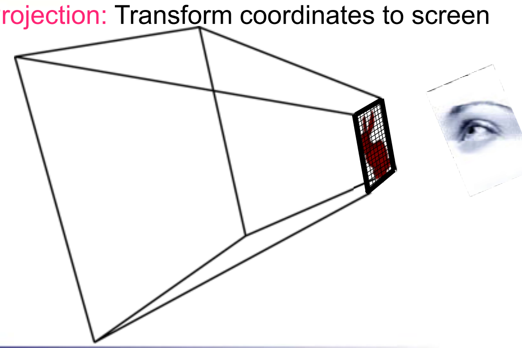
- Take a pixel on the image in the virtual world
- Compute ray through pixel and camera center
- Intersect ray with scene



Currently not adapted to
real-time interaction

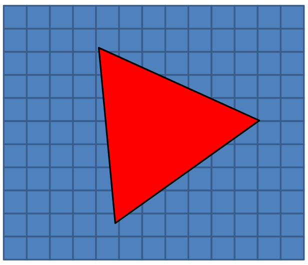
Simplified Graphics Pipeline

- Projection: Transform coordinates to screen

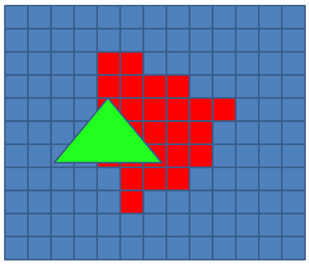


Simplified Graphics Pipeline

- Rasterization: Fill screen pixels

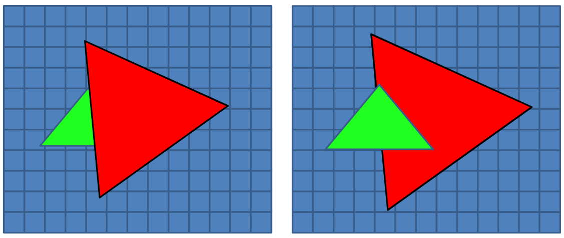


Simplified Graphics Pipeline



Simplified Graphics Pipeline

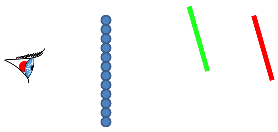
- Catch: Triangle drawing order changes result



As for ray tracing: need the closest triangle

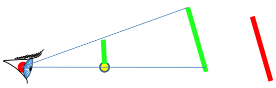
Simplified Graphics Pipeline

- Depth Test: Avoid sorting!
- Store a depth in each pixel



Simplified Graphics Pipeline

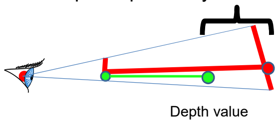
- Depth Test: Avoid sorting!
- Store a depth in each pixel



Simplified Graphics Pipeline

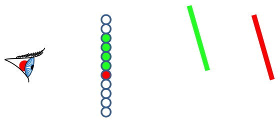
- Depth Test: Avoid sorting!
- Store a depth in each pixel

Compare new distance to stored distance
Update pixel only if new distance is nearer



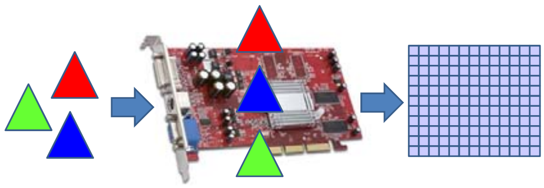
Simplified Graphics Pipeline

- Depth Test: Avoid sorting!
- Store a depth in each pixel



Simplified Graphics Pipeline

- Highly parallelizable ➡GPUs



128 processors working in parallel (in 2007)
...soon thousands and more...

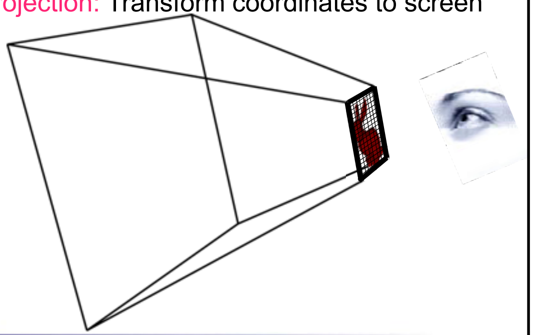
Questions?

Today: Graphics Pipeline

- What did a $400 GPU ever do for me?

Simplified Graphics Pipeline

- Projection: Transform coordinates to screen



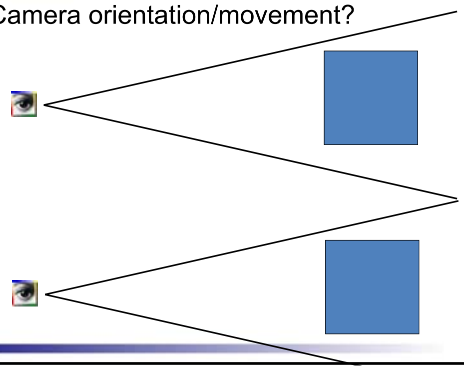
Graphics Pipeline Overview

- Camera Movement/Orientation
- Object Movement/Orientation
- Projection
- Rasterization

How to perform operations in “practice”?

Virtual Camera Model

- Camera orientation/movement?

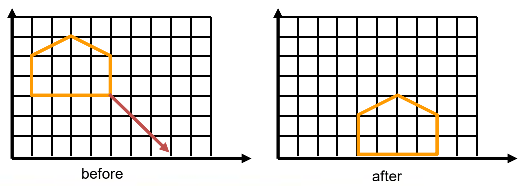


Movement and Orientation in 2D

- Starting in 2D
  - Simpler to represent

Translations

- Simple Modification :
  - $x' = x + t_x$
  - $y' = y + t_y$

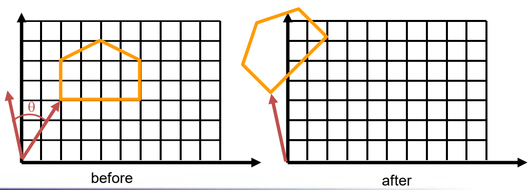


Translation

- Is a sum of vectors: P' = P + T

Rotation

- Rotation in 2D :
  - $x' = \cos\theta\ x - \sin\theta\ y$
  - $y' = \sin\theta\ x + \cos\theta\ y$



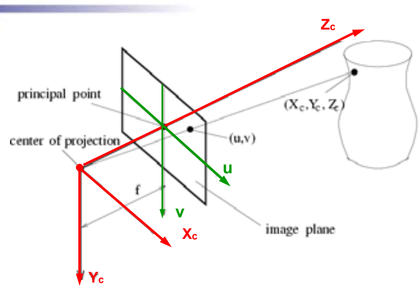
Rotation

- Is a matrix multiplication:

P'=**R**P

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Perspective Projection



$$u = f\,\frac{X}{Z} \qquad v = f\,\frac{Y}{Z}$$

Virtual Camera Model

Projecting a scene point with the camera:
- Apply camera position (adding an offset)
- Apply rotation (matrix multiplication)
- Apply projection (non-linear scaling)

Our camera starts to become complicated and not well adapted to a hardware solution…

There has to be a better way…

What we want:

- Notation simple, concise
- Unification
  - Translation, rotation
- Make concatenation of operations possible

And if I am allowed to dream:
- Allow projection operation too
- Keep everything linear

Dreams can come true!

We will see…

- Graphics Pipeline:
  - Take vertices
  - Multiply with matrix
  - Fill pixels underneath triangle

Homogenous Coordinates - Definition

- Projective geometry
  - Used everywhere (Image, Vision, Robotics…)
- N-D space represented by (N+1)-D coordinates without a zero and a new equivalence relation:

Two points p, q are considered equal
iff
exists a!=0 such that p*a = q

Homogenous Coordinates - Examples

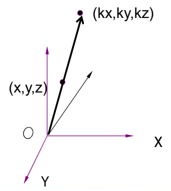
- Let us focus on in the following on a 2D space (having 3D coordinates)
- What does this equivalence relation mean?
- E.g.,
  (1,2,1) = (2,4,2)

Homogenous Coordinates – Affine Space

But why do we have 3 coordinates for 2D?

Two points are the same if they lie on a line through the origin (0,0,0)

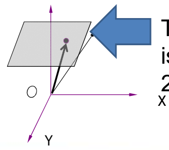
Along a line -> one dimension less



Homogenous Coordinates – Affine Space

What is the relationship between the "standard space" and projective space?

The traditional "affine" 2D space can be embedded via a plane.



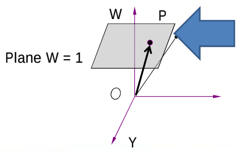
This affine plane is like a standard 2D space

Homogenous Coordinates – Affine Space

Simplest embedding:

We will always set the last coordinate to 1.

To underline this special role, the last coordinate is called W.
e.g., a point is (x,y,w)



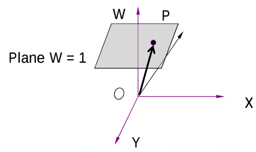
This affine plane is like a standard 2D space

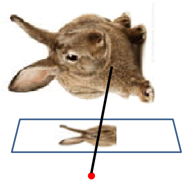
Homogenous Coordinates – Projection

How to find affine points from projective points?

Divide by last coordinate and keep first two:
(2x,2y,2) / 2 = (x,y,1) -> (x,y)

Does this look familiar?

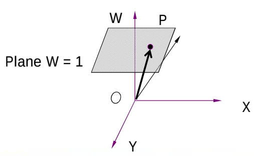




Homogenous Coordinates - Embedding

- How to go from 2D to projective ?

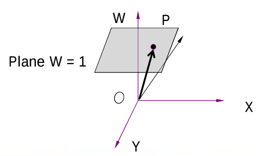
Imagine a mapping (x,y) -> (x,y,1) =(2x,2y,2)



Homogenous Coordinates

Can all points be projected on the plane W=1?

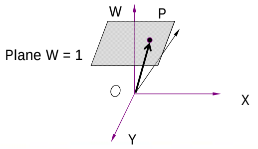
(x,y,0) ?



Homogenous Coordinates

- Let's look at
  P=(x, y, w) with w -> 0

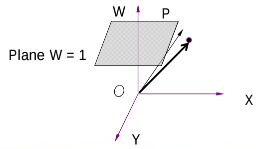
  P=(1,0,1)



Homogenous Coordinates

- Let's look at
  P=(x, y, w) with w -> 0

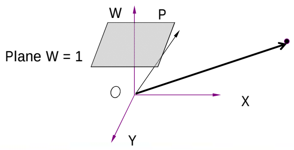
  P=(1,0,0.5)=(2,0,1)



Homogenous Coordinates

- Let's look at
  P=(x, y, w) with w -> 0

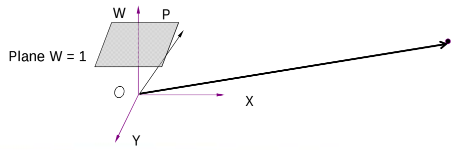
  P=(1,0,1/4)=(4,0,1)



Homogenous Coordinates

- Let's look at
  P=(x, y, w) with w -> 0

  P=(1,0,1/16)=(16,0,1)

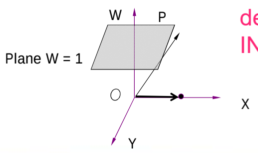


Homogenous Coordinates

- Let's look at
  P=(x, y, w) with w -> 0

  P=(1,0,0)=(1,0,0)

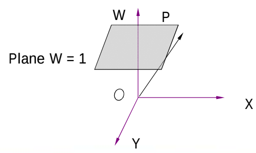
We have a way to describe points at INFINITY!



Points at infinity

Choosing a different plane associated with the embedding, changes infinity and much more…

We will focus on an affine space with Plane W=1!
Hence, in our case Infinity always means W=0



Homogeneous Coordinates

- How is all this useful?

Projective geometry makes things simpler!

Overview

- Object Movement/Orientation
- Complex Objects
- Projection

Translations in homog. coordinates

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ w' = \quad w \end{cases}$$

$$\begin{cases} \frac{x'}{w'} = \frac{x}{w} + t_x \\ \frac{y'}{w'} = \frac{y}{w} + t_y \end{cases}$$

What happens to points at infinity?

Rotation in homog. coordinates

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$\begin{cases} x' = \cos\theta x - \sin\theta y \\ y' = \sin\theta x + \cos\theta y \\ w' = \quad w \end{cases}$$

$$\begin{cases} \frac{x'}{w'} = \cos\theta \frac{x}{w} - \sin\theta \frac{y}{w} \\ \frac{y'}{w'} = \sin\theta \frac{x}{w} + \cos\theta \frac{y}{w} \end{cases}$$

What happens to points at infinity?

Transformation Compositing

- All rigid transformations become linear! (and many others too…)
- It is possible to produce linear applications that translate and rotate

M = (Rot)(Trans)

Rotation around point Q

- Rotation around point Q:
  - Translate Q to origin($\mathbf{T}_Q$),
  - Rotate around origin ($\mathbf{R}_\Theta$)
  - Translate back to Q ($\mathbf{T}_{-Q}$).

$$\longrightarrow \quad P' = (\mathbf{T}_{-Q})\mathbf{R}_\Theta \mathbf{T}_Q\, P$$

And in 3 dimensions ?

- The same!
- Add a fourth coordinate, *w*

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- All transformations are 4x4 matrices

Translations in 3D

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ z' = z + wt_z \\ w' = \quad w \end{cases}$$

Rotations in 3D

- Rotation = axis and angle
- Rotations around x,y,z axis are simple
  - A different axis is achieved by combining matrices

Rotation about *x*-axis

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

Axis x not changing

Quick verification : rotation of $\pi/2$
Should change *y* in *z*, and *z* in *-y*

$$R_x(\tfrac{\pi}{2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about *y*-axis

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

Axis y not changing

Quick verification : rotation of $\pi/2$
Should change *z* in *x*, and *x* in *-z*

$$R_y(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about *z*-axis

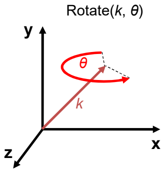
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Axis z not changing

Quick verification : rotation of $\pi/2$
Should change *x* in *y*, and *y* in *-x*

$$R_z(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation



- Around (k_x, k_y, k_z)
  (Rodrigues Formula)

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} k_xk_x(1-c)+c & k_zk_x(1-c)-k_zs & k_xk_z(1-c)+k_ys & 0 \\ k_yk_x(1-c)+k_zs & k_zk_x(1-c)+c & k_yk_z(1-c)-k_xs & 0 \\ k_zk_x(1-c)-k_ys & k_zk_x(1-c)-k_xs & k_zk_z(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

where $c = \cos\theta$ & $s = \sin\theta$

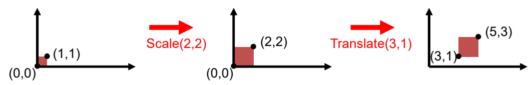
Attention !

- Matrix Multiplication is not commutative
- The order of transformations is important
  - Rotation then translation != transl. then rotation

Source of bugs…

Example

Scaling + Translation

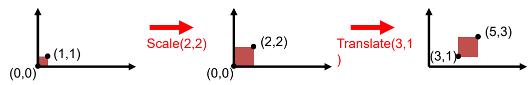


Matrix combination: p' = T (S p) = TS p

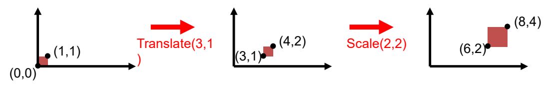
$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

NOT commutative

Scaling + translation: p' = T (S p) = TS p



Translation + scaling: p' = S (T p) = ST p



NOT commutative

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

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

Questions?

Overview

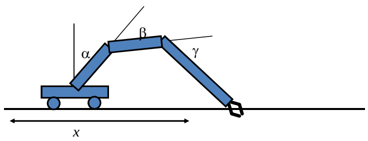
- Object Movement/Orientation
- Complex Objects
- Projection

Complex Objects

- Projective Geometry allows us to describe relationships by concatenating operations together.

Why concatenate operations?

- Often, objects are defined as combinations
    - Examples : robots, cars (wheels)…
- We want a natural behavior:
    - Object stays connected:
        - If you move the arm, the hand should follow



Why concatenate operations?

- Relative coordinates
    - Wheel with respect to car
    - Bolts with respect to wheel
- Like in real life: rarely absolute coordinates!

Why concatenate operations?

- Use relative coordinates:
    - Position of hand with respect to arm
    - Position of arm with respect to body
    - …

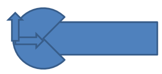
Why concatenate operations?

- We need absolute coordinates to draw:
    - Concatenate and apply complete matrix
        - Example: Draw an arm with hand

The arm consists of two parts:
The arm itself and a hand
Both are designed independently
and placed at the origin as shown below:

Why concatenate operations?

- Drawing first the arm, then the hand, you get:



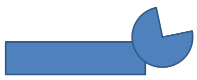
- That does not look right!
- Instead: Produce a matrix to place the origin at the right location and then draw the object

Why concatenate operations?

- Graphics Pipeline:
  - Take vertices
  - multiply with matrix
  - Fill pixels

Why concatenate operations?

- Concatenate and apply matrices
  - Example: Draw an arm with hand



Why concatenate operations?

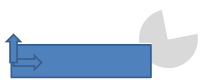
- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)



Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)
  - Rotate for hand (changes matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)
  - Rotate for hand (changes matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)
  - Rotate for hand (changes matrix)
  - Draw hand (transform geometry using the matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)
  - Rotate for hand (changes matrix)
  - Draw hand (transform geometry using the matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - **ROTATE (changes matrix)**
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)
  - Rotate for hand (changes matrix)
  - Draw hand (transform geometry using the matrix)

Why concatenate operations?

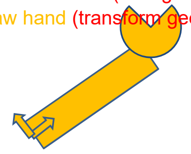
- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - **ROTATE (changes matrix)**
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)
  - Rotate for hand (changes matrix)
  - Draw hand (transform geometry using the matrix)

Why concatenate operations?

- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - **ROTATE (changes matrix)**
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)
  - Rotate for hand (changes matrix)
  - Draw hand (transform geometry using the matrix)

Why concatenate operations?

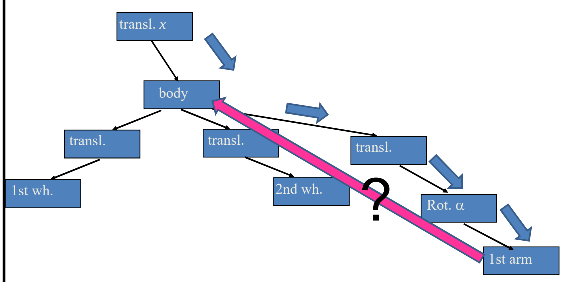
- Concatenate and apply matrices
  - Translation at position of arm (changes matrix)
  - **ROTATE (changes matrix)**
  - Draw arm (transform geometry using the matrix)
  - Translate further to position of hand (changes matrix)
  - Rotate for hand (changes matrix)
  - Draw hand (transform geometry using the matrix)



Why concatenate operations?

- What about even more complex objects?
- For example:
  - A race car…
  - An airplane…

Hierarchical Object Definition



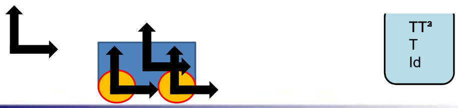
Reuse objects (e.g., only one wheel is stored)

Coordonnées relatives

- How to go back?
  E.g., Body -> left arm -> Body?
- Simple Solution:
- Keep a matrix stack!

Matrix Stack

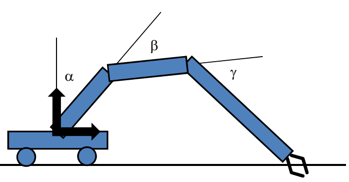
- Keep information about modifications
  - **T** (translation by x) push
  - draw robot body
  - **TT²** (translation to center of 1st wheel) push
  - draw first wheel as circle of center (0,0)
  - return to **T:** pop
  - **TT³** (T³ translation to center of 2nd wheel) push
  - draw second wheel



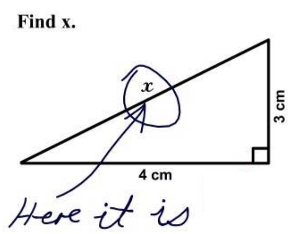
Why concatenate operations?

- Another simple example:

1. Rotation a
2. Translation a
3. Rotation b
4. Translation b
5. Rotation c
6. Translation c



Questions?

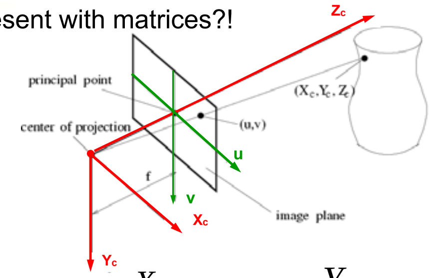


Overview

- Object Movement/Orientation
- Complex Objects
- Projection

Perspective Projection

- Represent with matrices?!



$$u = f\frac{X}{Z} \qquad v = f\frac{Y}{Z}$$

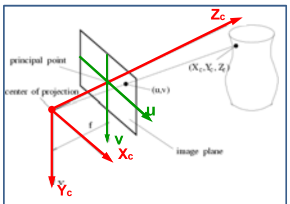
Perspective Projection

- "Homogenize" the points

$$P = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \Rightarrow \tilde{P} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- Look for M such that:

$$M\tilde{P} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix}$$



$$u = f\frac{X}{Z} \qquad v = f\frac{Y}{Z}$$

Perspective Projection

- Hint: Think projective!

$$M\tilde{P} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix} \Rightarrow M\tilde{P} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix}$$

Perspective Projection

- Solution:

$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

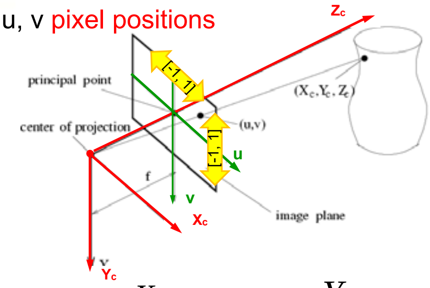
Putting things together: Camera Model

- Projection + Movement + Rotation

$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Adding Image Space to Camera Model

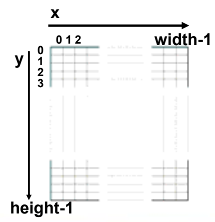
- Make u, v pixel positions



$$u = f\frac{X}{Z} \qquad v = f\frac{Y}{Z}$$

Internal Camera Parameters

- [-1,1]x [-1,1] ➡ [0, width-1]x[0,height-1]
- You know how to do it…



$$\begin{cases} x = k_x u + x_0 \\ y = k_y v + y_0 \end{cases}$$

$$\begin{pmatrix} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Complete Camera Model

- Finally:

$$P = \underbrace{\overbrace{\begin{pmatrix} a_x & 0 & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}}^{\text{image}} \overbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}^{\text{projection}}}_{\begin{pmatrix} a_x & 0 & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}} \overbrace{\begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}}^{\text{orientation/location}}$$

Complete Camera Model

- Finally:

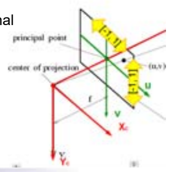
$$P = \begin{pmatrix} a_x & 0 & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \end{pmatrix}}$$

Complete Camera Model

- Finally (notation often used in vision literature):

$$P = \underbrace{\begin{pmatrix} a_x & 0 & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\substack{\text{intrinsic/ internal} \\ \text{parameters}}} \underbrace{\begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \end{pmatrix}}_{\substack{\text{extrinsic / external} \\ \text{parameters}}}$$

Change settings | Move camera



In graphics, the pixel mapping is implicitly handled via an external viewport matrix .

Summary

- First steps into the Graphics Pipeline
  - Homogenous Coordinates
  - Matrix Stacks
  - Camera Model

Thank you very much for your attention!