Module IN 2018

Introduction to Augmented Reality

Prof. Gudrun Klinker

Coordinate Systems, Transformations and Projective Geometry (2D)

SS 2018

Literature

- T. Akenine-Möller et al., Real-Time Rendering, 3rd edition. AK Peters, Ltd, 2008 (ISBN: 978-1-56881-424-7)
- The OpenGl Programming Guide - The Redbook; http://www.opengl.org/documentation/red_book/
- Pustka, Huber, Bauer and Klinker: *Spatial Relationship Patterns: Elements of Reusable Tracking and Calibration Systems*, ISMAR 06, Oct. 2006. AWARD.
- Huber, Pustka, Keitler, Echtler and Klinker: *A System Architecture for Ubiquitous Tracking Environments*, to be presented at ISMAR 07, Nov. 2007.
- Huber, Becker and Klinker: *Location aware computing using RFID infrastructure*. Int. J. Autonomous and Adaptive Communications Systems, Vol. 3, No. 1, 2010.

- Several dissertations (Fachgebiet Augmented Reality: http://campar.in.tum.de/Chair/ResearchAr)

Agenda

1. Scene Graph
2. 2D Transformations
3. Projective Geometry (2D)
4. 3D Transformations
5. Spatial Relationship Graphs (SRGs), Data Flow Networks (DFNs), and Spatial Relationship Patterns

1. Scene Graph

1.1 Motivation

1.2 Example

1.3 Most important Components of a Scene Graph

1.4 Rendering

1.5 Coordinate Systems

1.1 Motivation

The world consists of many

- People (users): Hands, eyes, …
- Objects: Subparts, tracked targets, …
- Trackers

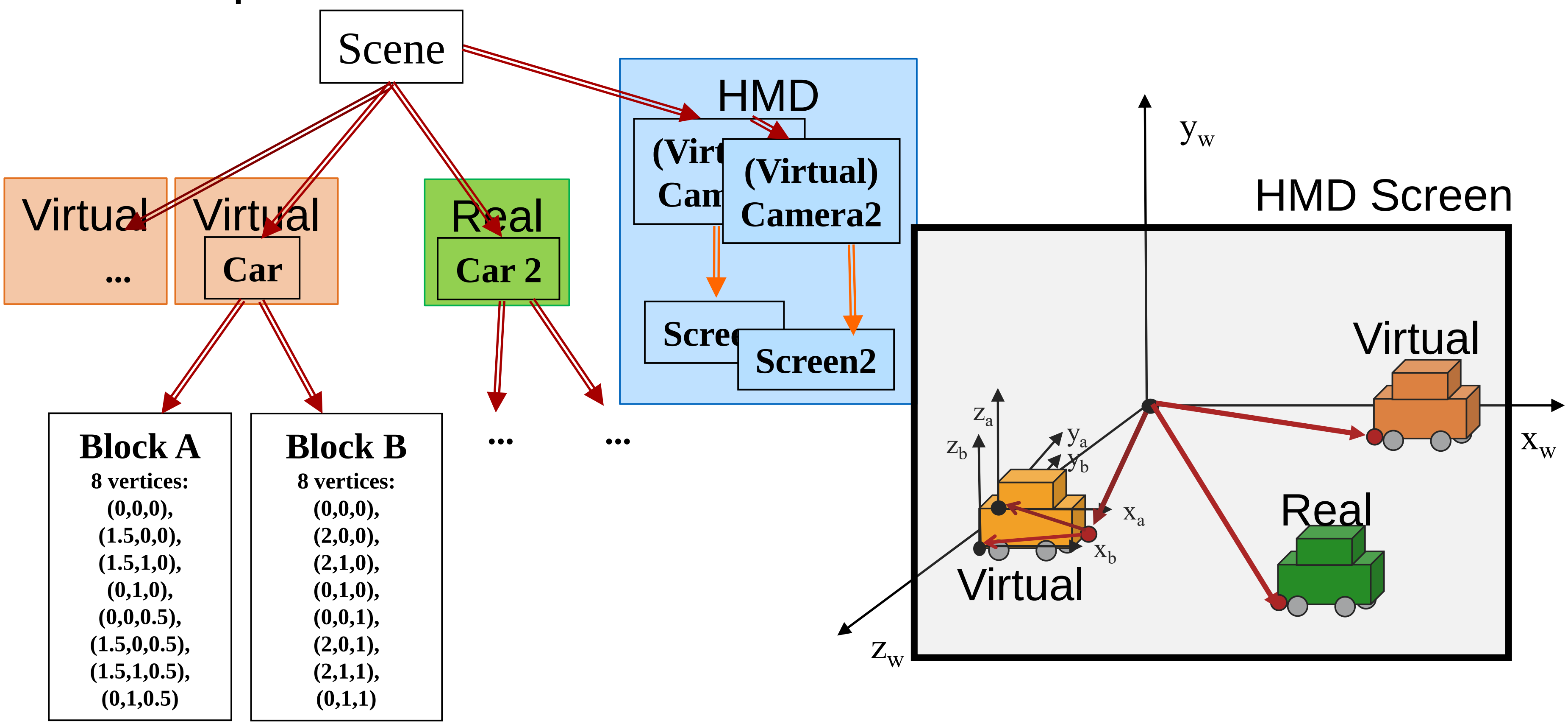
We want to

- Describe objects individually (independently of their current position in the world)
- Replicate objects (or parts) without having to (re)describe all geometric details
- Determine the current pose of an object with respect to many reference systems:
  - Given tracker
  - The world
  - The user
  - A display

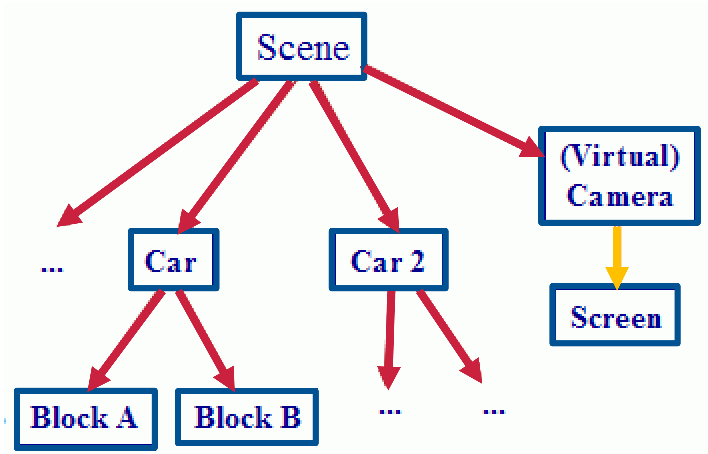
Describe the world as an interrelated network of coordinate systems

1.2 Example

Example scene

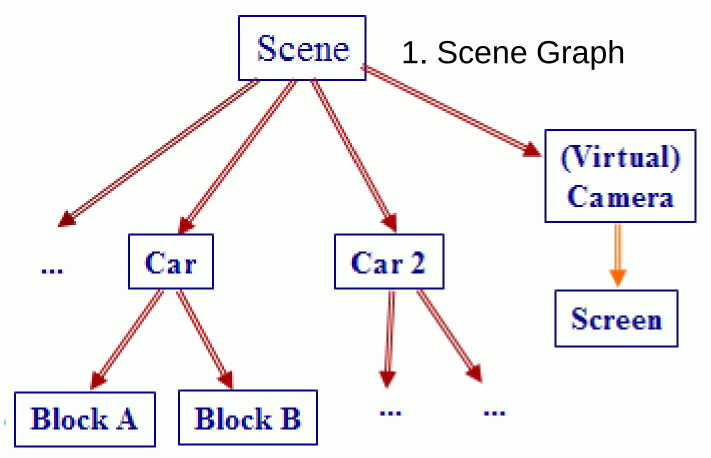


1.3 Most Important Components of a Scene Graph



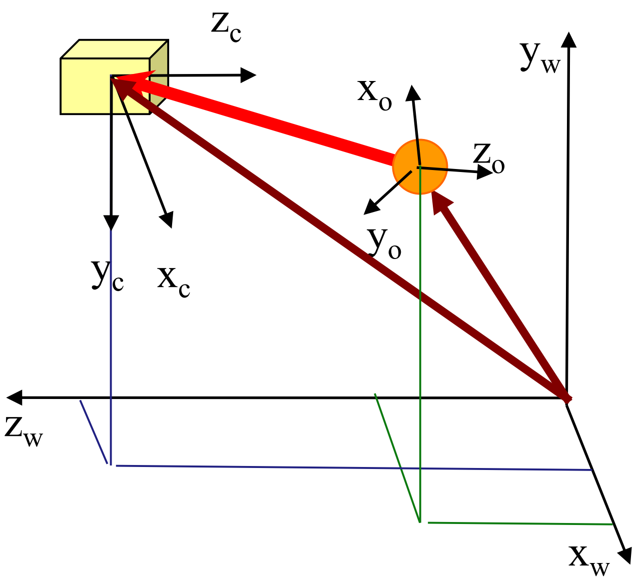
- Nodes
  - Coordinate systems
    - Object parts
    - Groups of objects
    - Scene
    - Camera (eye)
- Directed edges between nodes
  - Geometric transformations
    - Changes in position, orientation, scale, perspective etc. of a node, relative to its predecessor
    - In graphics: typically a tree
    - In AR: can be a true graph (Spatial Relationship Graph, SRG)

1.4 Rendering a Scene Graph



Rendering
= Traversal of the Scene Graph

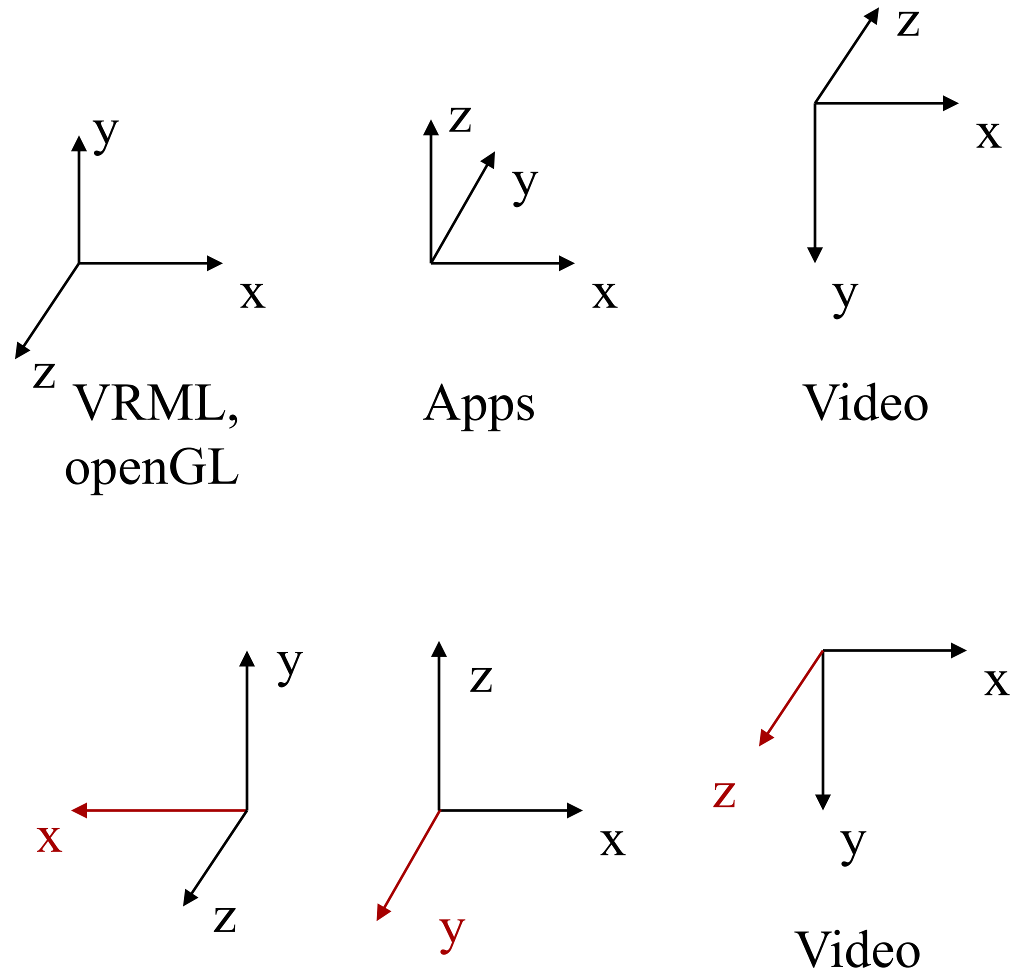
- **Homogeneous coordinates**
  3D Transformations between
  - Object coordinates
  - World coordinates
  - Camera coordinates
- **Geometry Pipeline in OpenGL**



1.5 Different Coordinate Systems

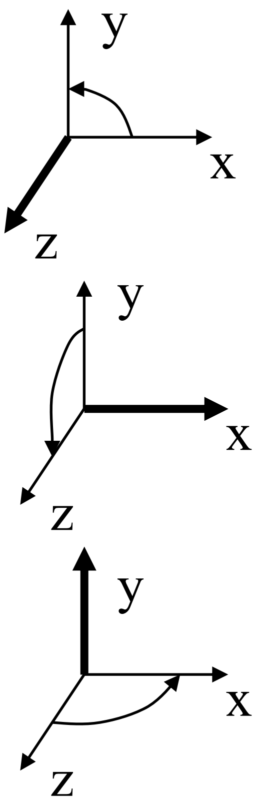
- Righthanded Systems

- Lefthanded Systems



Righthanded Rotations

- Around z: x $\rightarrow$ y
- Around x: y $\rightarrow$ z
- Around y: z $\rightarrow$ x



Counterclockwise rotation around respective axis

Agenda

1. Scene Graph
2. 2D Transformations
3. Projective Geometry (2D)
4. 3D Transformations
5. Spatial Relationship Graphs (SRGs), Data Flow Networks (DFNs), and Spatial Relationship Patterns

2. 2D Transformations

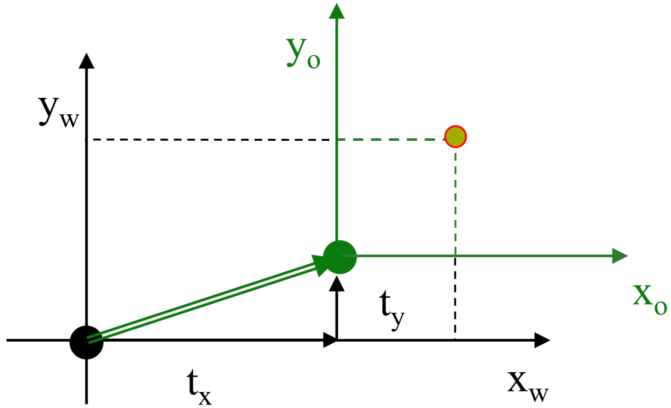
→ 2.1 Principles

2.2 Homogeneous Coordinates

2.1 Principles

- Translation:

$$x_w = x_o + t_x$$
$$y_w = y_o + t_y$$

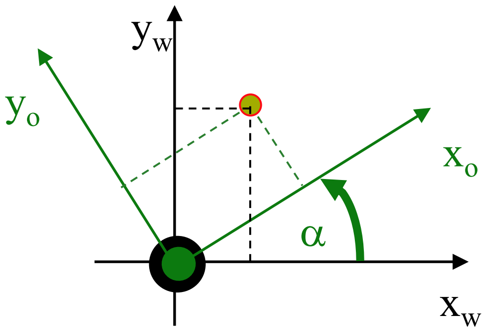


- Scaling:

$$x_w = s_x * x_o$$
$$y_w = s_y * y_o$$

- Rotation:

$$x_w = \cos\alpha * x_o - \sin\alpha * y_o$$
$$y_w = \sin\alpha * x_o + \cos\alpha * y_o$$



2. 2D Transformations

2.1 Principles

→ 2.2 Homogeneous Coordinates

2.2 Homogeneous Coordinates

Homogeneous Coordinates

- Translation:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x + wt_x \\ y + wt_y \\ w \end{bmatrix}$$

- Scaling:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ w \end{bmatrix}$$

- Rotation:

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} \cos\alpha\ x - \sin\alpha\ y \\ \sin\alpha\ x + \cos\alpha\ y \\ w \end{bmatrix}$$

Agenda

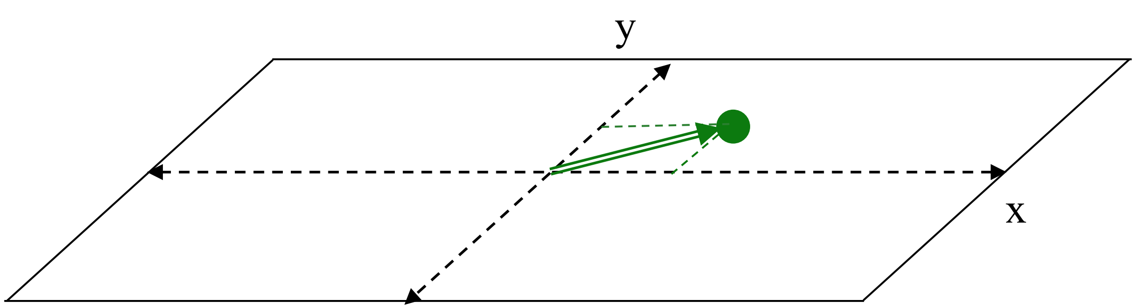
1. Scene Graph
2. 2D Transformations
3. Projective Geometry (2D)
4. 3D Transformations
5. Spatial Relationship Graphs (SRGs), Data Flow Networks (DFNs), and Spatial Relationship Patterns

3. Projective Geometry (2D)

- 3.1 Motivation
- 3.2 Points
- 3.3 Lines
- 3.4 Consequences

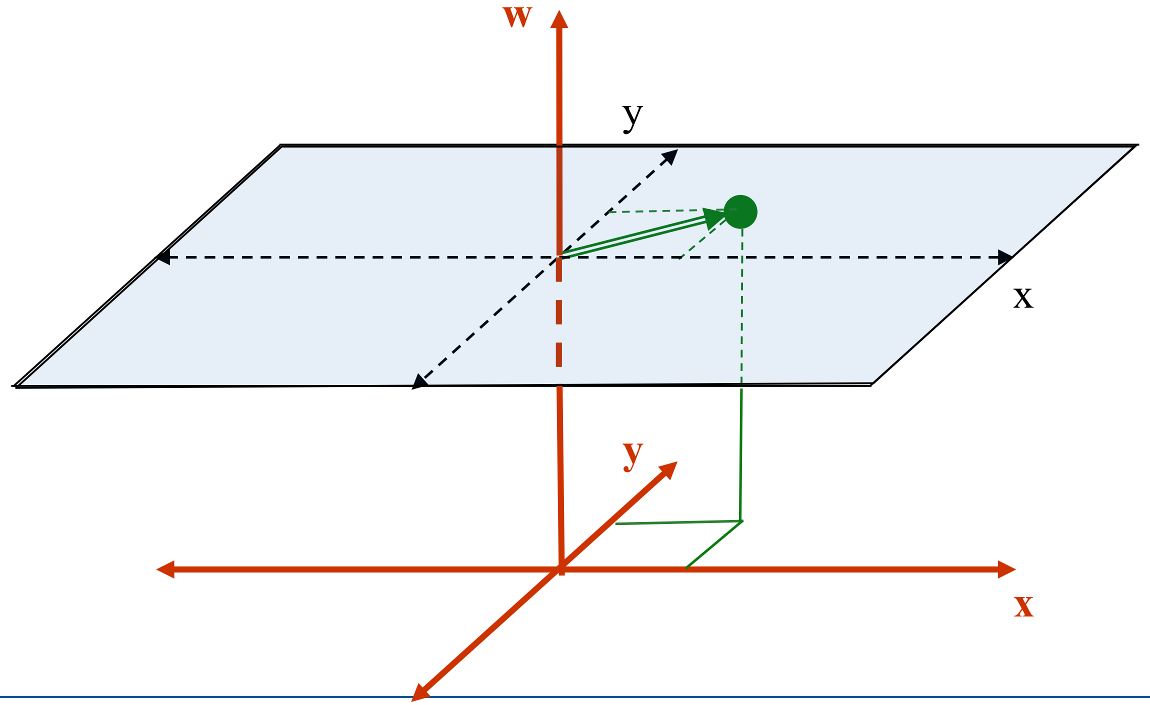
3.1 Motivation

- Why use 3x3 matrix for 2D geometry?
- What does the 3rd dimension **w** mean?
- Why does it get rid of vector addition (for translation)?



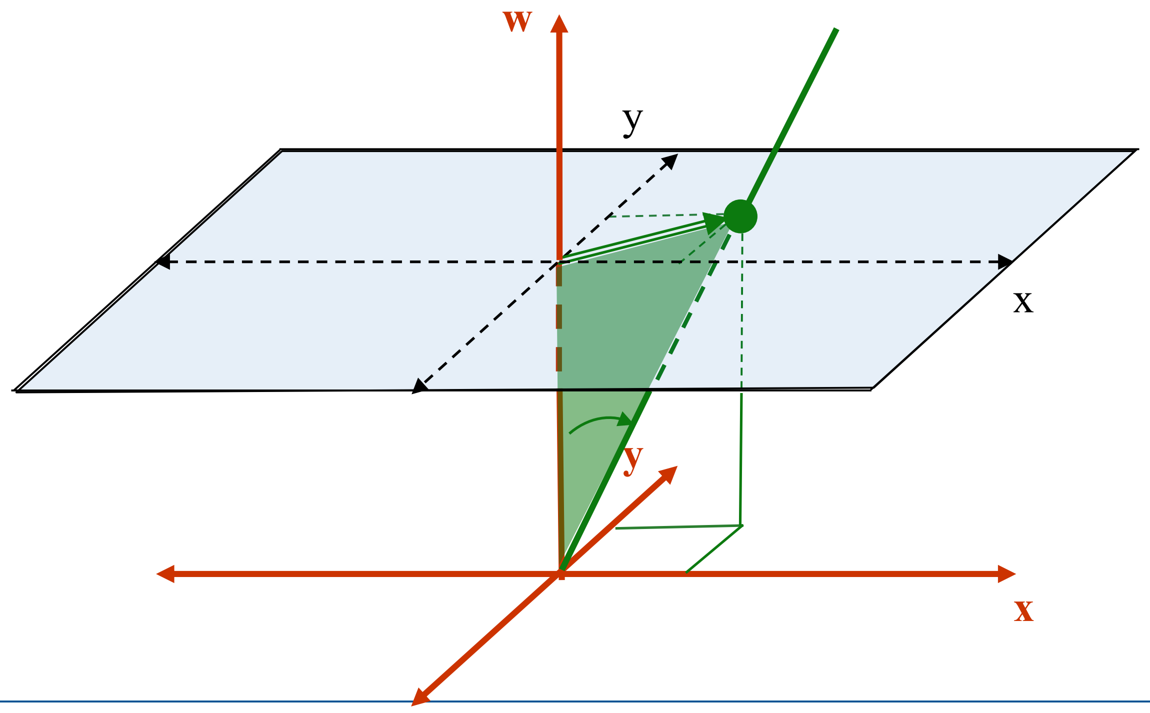
3.1 Motivation

- Why use 3x3 matrix for 2D geometry?
- What does the 3rd dimension **w** mean?
- Why does it get rid of vector addition (for translation)?



3.1 Motivation

- Why use 3x3 matrix for 2D geometry?
- What does the 3rd dimension **w** mean?
- Why does it get rid of vector addition (for translation)?



3.1 Motivation

Use of projective geometry

- 2D: homogeneous transformations of points and lines
  → 3x3 homogeneous matrices
- 3D: homogeneous transformations of points, lines and planes
  → 4x4 matrices
- 3D-2D projections
  → 3x4 matrices

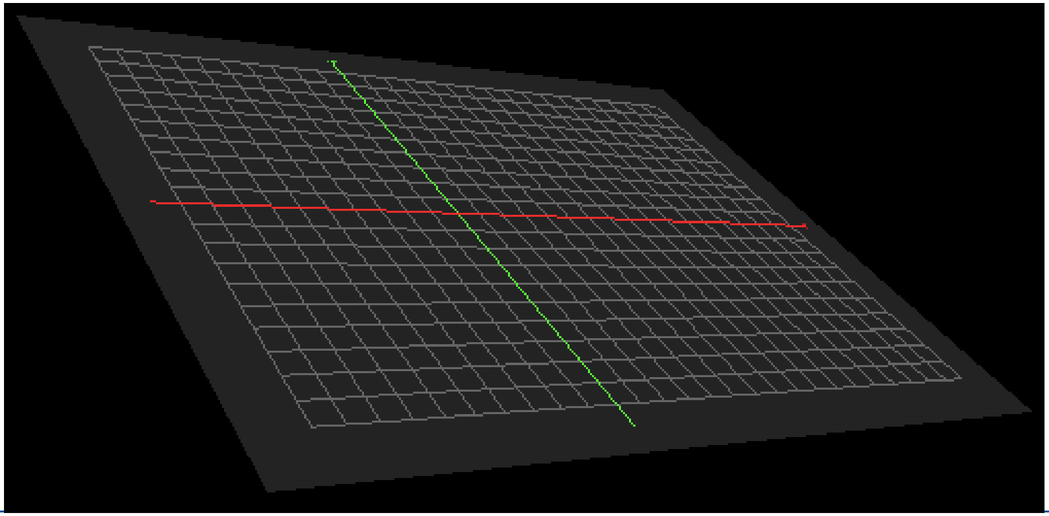
Transformations:

- Euclidean
- Affine
- Projective

3.1 Motivation

Important properties

- Geometric distortion due to perspective projection
- *Invariant*:
  - Straight lines
- Not invariant:
  - Angles
  - Parallel lines



3. Projective Geometry (2D)

3.1 Motivation

→ 3.2 Points

3.3 Lines

3.4 Consequences

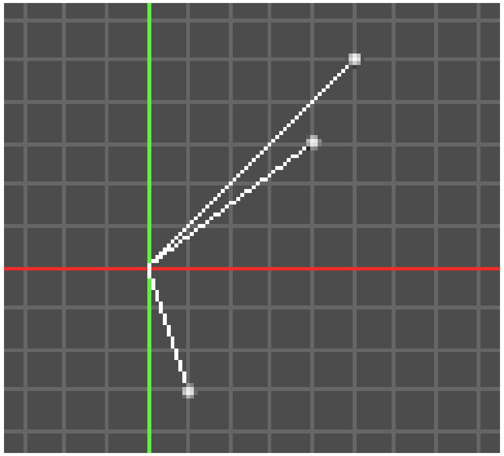
3.2 Points

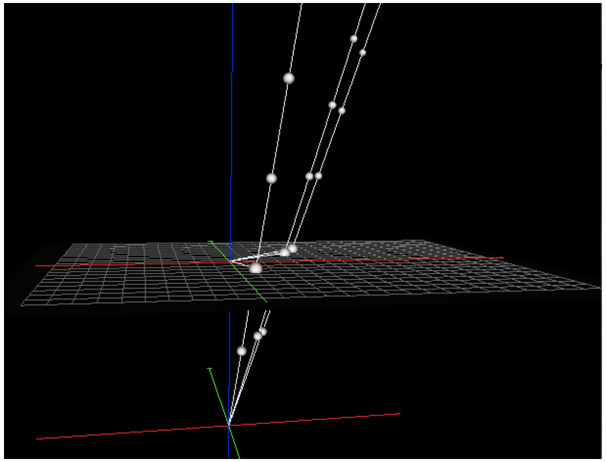
Inhomogeneous notation in R^2

$$P = (x, \quad y)^{\mathrm{T}}$$

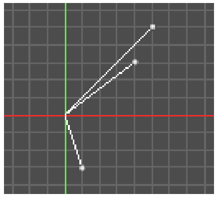
Homogeneous notation in P^2 (*projective space*)

$$\mathbf{x} = (wx, \quad wy, \quad w)^{\mathrm{T}}$$
$$= w\,(x, y, 1)^T$$

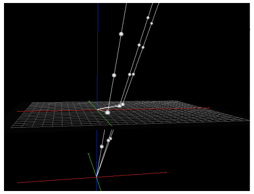




3.2 Points (Examples)



$\mathbb{R}^2$	$\mathbb{P}^2$
$P_1 = (0.4, \quad 0.3)^T$	$\mathbf{x_1} = (0.4w, \quad 0.3w, \quad w)^T$
	$= (0.4, \quad 0.3, \quad 1.0)^T$
	$= (0.8, \quad 0.6, \quad 2.0)^T$
$P_2 = (0.1, \quad -0.3)^T$	$\mathbf{x_2} = (0.3, \quad -0.9, \quad 3.0)^T$
$P_3 = (0.5, \quad 0.5)^T$	$\mathbf{x_3} = (0.5, \quad 0.5, \quad 1.0)^T$



3. Projective Geometry (2D)

3.1 Motivation

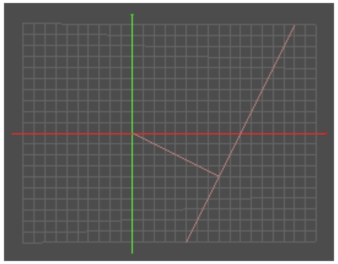
3.2 Points

→ 3.3 Lines

3.4 Consequences

3.3 Lines

- Line equation: $$\mathrm{ax} + \mathrm{by} + \mathrm{c} = 0$$
- Line normal: $$\mathbf{n} = (a\ b)/|\mathbf{n}|$$

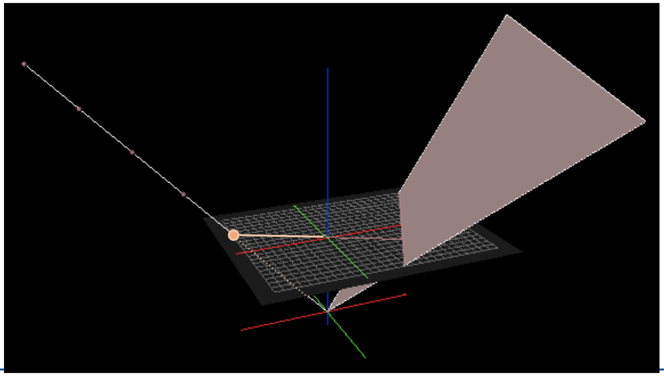


- Homogeneous line notation in $\mathbb{P}^2$ (projective space):

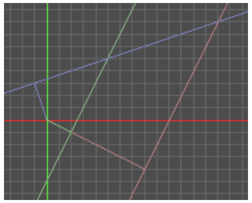
$$\mathbf{l} = k(a\ b\ c)^{\mathrm{T}}$$

$$w(x\ y\ 1) \bullet k \begin{pmatrix} a \\ b \\ c \end{pmatrix} = 0$$

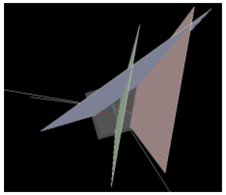
$$\mathbf{x}^{\mathrm{T}}\mathbf{l} = 0$$



3.3 Lines (Examples)



$\mathbb{R}^2$	$\mathbb{P}^2$
$l_1: 2x - y - 2 = 0$	$\mathbf{l}_1 = (2, \; -1, \; -2)^T = (-1, \; 0.5, \; 1.0)^T$
$l_2: 2x - y - 0.5 = 0$	$\mathbf{l}_2 = (2, \; -1, \; -0.5)^T = (-4, \; 2, \; 1)^T$
$l_3: x - 3y + 1 = 0$	$\mathbf{l}_3 = (1, \; -3, \; 1)^T$



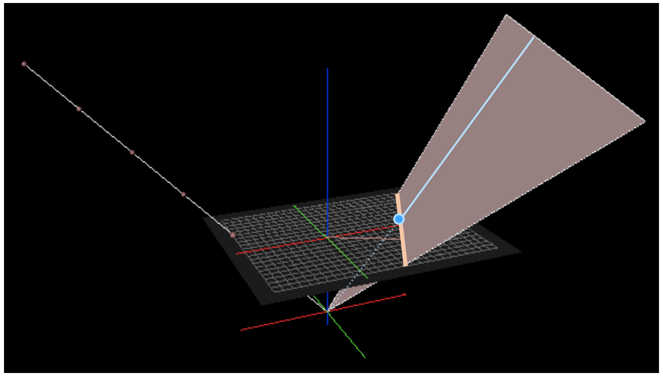
3. Projective Geometry (2D)

3.1 Motivation
3.2 Points
3.3 Lines
→ 3.4 Consequences

3.4 Consequences

Duality of points and lines

$$ax + by + c = 0$$



$\mathbf{x}^T\mathbf{l} = 0$

Point vector $\mathbf{x} = w(x\ y\ 1)^T$
Line vector $\mathbf{l} = k(a\ b\ c)^T$

$\mathbf{x}^T\mathbf{l} = 0$ Point x lies on line l

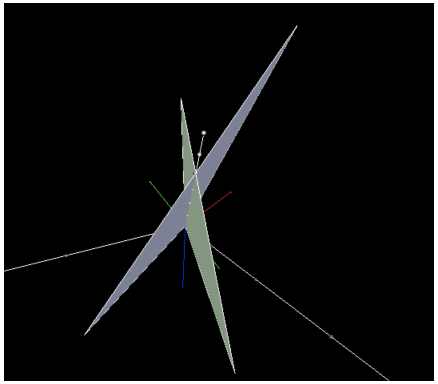
$\mathbf{l}^T\mathbf{x} = 0$ Line l goes through point x

- Point vectors **x** and (normals to) lines **l** are perpendicular: $\mathbf{x}^T\mathbf{l} = \mathbf{l}^T\mathbf{x} = 0$

3.4 Consequences

Concise formulations

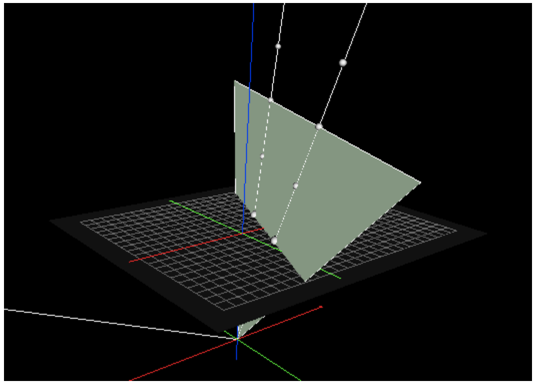
- Intersection of 2 lines $\mathbf{l_1}$,$\mathbf{l_2}$ is a point $\mathbf{x}$:



→ $\mathbf{x}$ is cross product of $\mathbf{l_1}$ and $\mathbf{l_2}$.

$$\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2$$

- Connection between 2 points $\mathbf{x_1}$,$\mathbf{x_2}$ is a line $\mathbf{l}$:



→ $\mathbf{l}$ is cross product of $\mathbf{x_1}$ and $\mathbf{x_2}$.

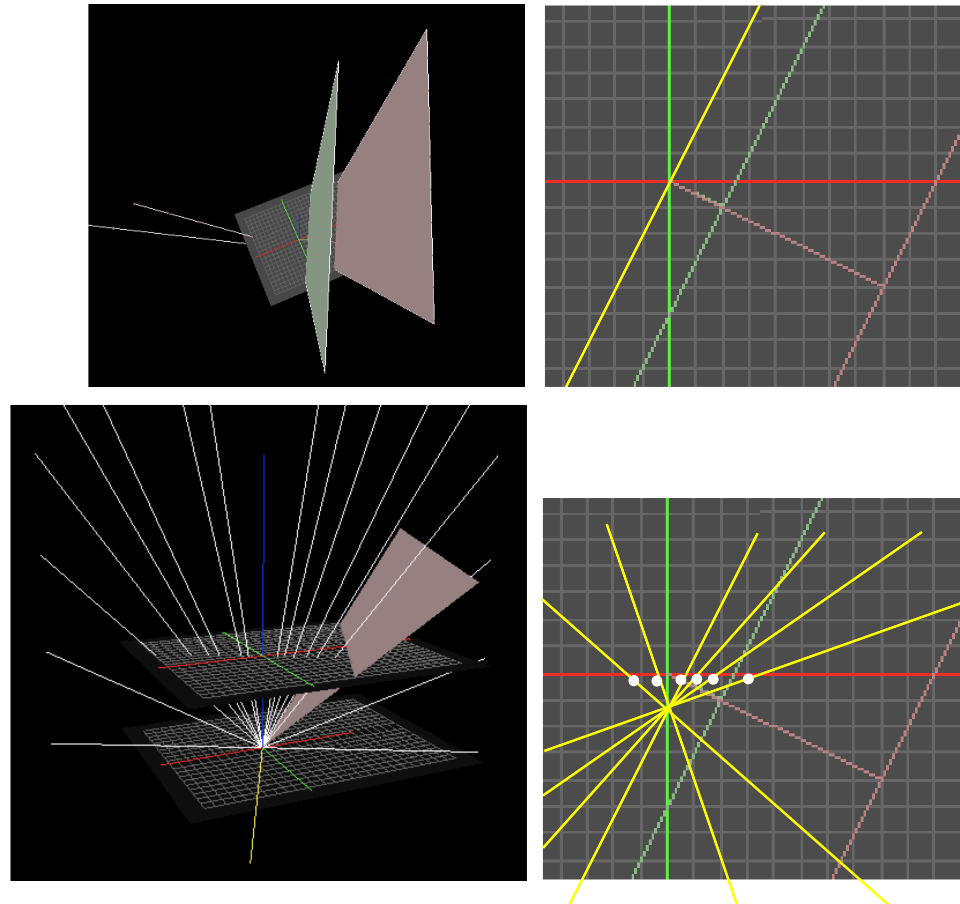
$$\mathbf{l} = \mathbf{x}_1 \times \mathbf{x}_2$$

3.4 Consequences

- Intersections of parallel lines

- Cross product $\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2$ lies in plane w=0
  $\rightarrow$ Points at infinity:
  Ideal points (with w=0)

- All possible orientations:
  Ideal line (w=0,y=1)



Agenda

1. Scene Graph
2. 2D Transformations
3. Projective Geometry (2D)
4. 3D Transformations
5. Spatial Relationship Graphs (SRGs), Data Flow Networks (DFNs), and Spatial Relationship Patterns

4. 3D Transformations

→ 4.1 Euclidean Transformations
4.2 Composition of Transformations
4.3 Projections and Viewport Transformations
4.4 Vertex Transformation Stages in OpenGL
4.5 Sample Code (OpenGL)

4.1 Euclidean Transformations

- Translation: glTranslate* (t_x,t_y,t_z)

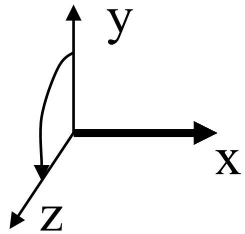
$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x + wt_x \\ y + wt_y \\ z + wt_z \\ w \end{bmatrix}$$

- Scaling: glScale* (s_x,s_y,s_z)

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ w \end{bmatrix}$$

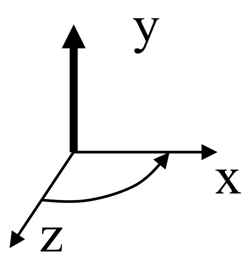
4.1 Euclidean Transformations

- Rotation: glRotate* (a,ex,ey,ez)
  - Around x



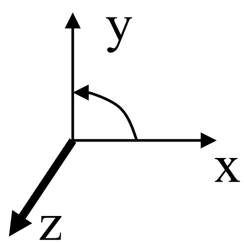
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x \\ \cos\alpha\, y - \sin\alpha\, z \\ \sin\alpha\, y + \cos\alpha\, z \\ w \end{bmatrix}$$

  - Around y



$$\begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \cos\alpha\, x + \sin\alpha\, z \\ y \\ -\sin\alpha\, x + \cos\alpha\, z \\ w \end{bmatrix}$$

  - Around z



$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \cos\alpha\, x - \sin\alpha\, y \\ \sin\alpha\, x + \cos\alpha\, y \\ z \\ w \end{bmatrix}$$

4. 3D Transformations

4.1 Euclidean Transformations

4.2 Composition of Transformations

4.3 Projections and Viewport Transformations

4.4 Vertex Transformation Stages in OpenGL

4.5 Sample Code (OpenGL)

4.2 Composition of Transformations

- Initialize accumulative matrix C_0 with identity matrix I
- Post-multiply accumulative matrix C_i with transformation matrix M_i according to command. $C_{i+1} := C_i M_i$
- Accumulated result represents complete transformation (independent of the number of transformation steps)
- Transformation sequence not commutative!!!

Example: translate1 - scale1 - rotate1 - translate2 - rotate2

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w \end{bmatrix} = I \bullet T_1 \bullet S_1 \bullet R_1 \bullet T_2 \bullet R_2 \bullet \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}$$

4.2 Composition of Transformations

Code-based Interpretation

4.2 Composition of Transformations

Composition of Transformations

Example: translate1 - scale1 - rotate1 - translate2 - rotate2

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w \end{bmatrix} = I \bullet \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}$$

Code-based interpretation

4.2 Composition of Transformations

Composition of Transformations

Example: translate1 - scale1 - rotate1 - translate2 - rotate2

Intrinsic Interpretation

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w \end{bmatrix} = I \bullet T_1 \bullet S_1 \bullet R_1 \bullet T_2 \bullet R_2 \bullet \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}$$

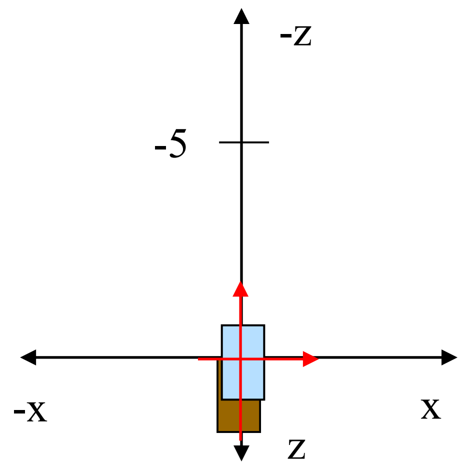
Code-based interpretation

- Fixed on local, mobile (object) coordinate system
- The world turns around the object

4.2 Composition of Transformations

Example

glTranslatef (0.0, 0.0, -5.0); // along z-axis
glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis



Intrinsic Interpretation

Turning the World: „Examine“

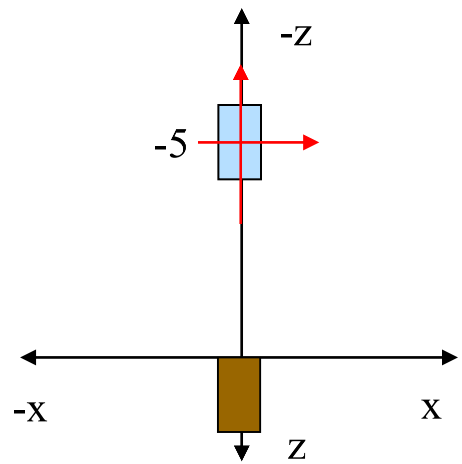
Code-based interpretation (“Examine”)

$$I * T1 * R1 * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

4.2 Composition of Transformations

Example

glTranslatef (0.0, 0.0, -5.0);   // along z-axis
glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis



Intrinsic Interpretation

Turning the World: „Examine“

Code-based interpretation (“Examine”)

$$I * T1 * R1 * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

4.2 Composition of Transformations

Example

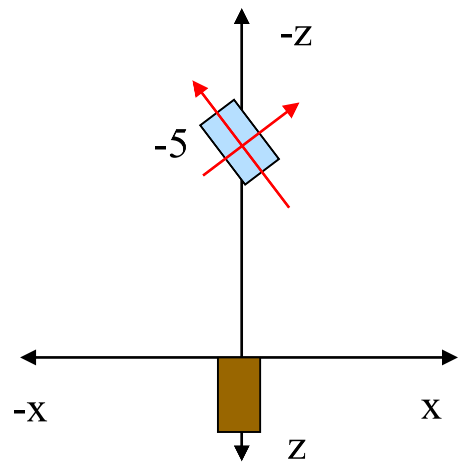
glTranslatef (0.0, 0.0, -5.0);   // along z-axis

glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis

Intrinsic Interpretation

Turning the World: „Examine"

Code-based interpretation (“Examine”)

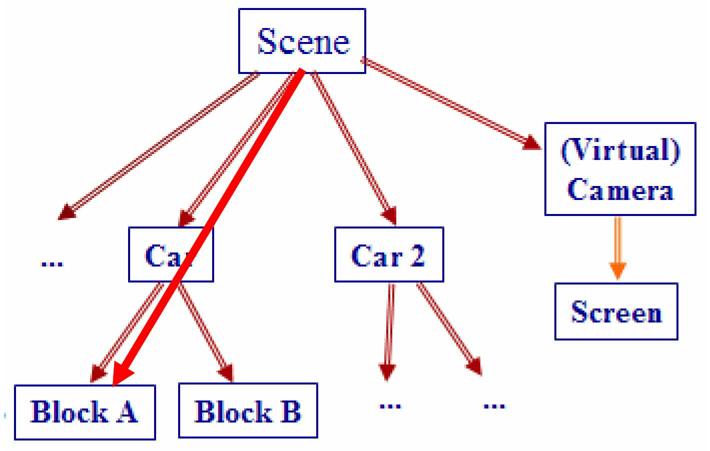


$$I * T1 * R1 * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

4.2 Composition of Transformations

Equivalent concepts:

- Code-based interpretation
- Intrinsic interpretation
- „Examine“ (turning world)
- Top-down traversal of the scene graph



4.2 Composition of Transformations

Mathematical Interpretation

4.2 Composition of Transformations

Composition of Transformations

Example: translate1 - scale1 - rotate1 - translate2 - rotate2

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w \end{bmatrix} = I \bullet \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}$$

Mathematical interpretation: (I (T1 (S1 (R1 (T2 (R2 ● v))))))

4.2 Composition of Transformations

Composition of Transformations

Example: translate1 - scale1 - rotate1 - translate2 - rotate2

Extrinsic Interpretation

$$\begin{bmatrix} x_w \\ y_w \\ z_w \\ w_w \end{bmatrix} = I \bullet T_1 \bullet S_1 \bullet R_1 \bullet T_2 \bullet R_2 \bullet \begin{bmatrix} x_o \\ y_o \\ z_o \\ w_o \end{bmatrix}$$

Mathematical interpretation: (I (T1 (S1 (R1 (T2 (R2 ● v))))))

- Fixed on global, stationary (world) coordinate system
- Object flies through the world

4.2 Composition of Transformations

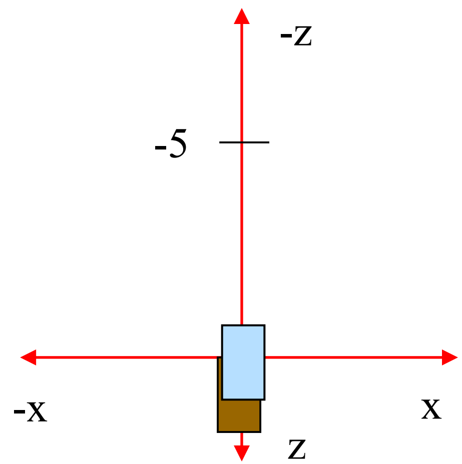
Example

glTranslatef (0.0, 0.0, -5.0); // along z-axis
glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis

Extrinsic Interpretation

Turning the Object: „Fly-Through"

Mathematical interpretation (“Fly-Through”)



$$I * T1 * R1 * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

4.2 Composition of Transformations

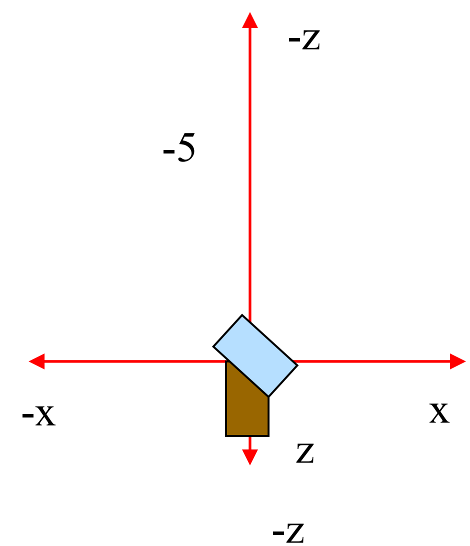
Example

glTranslatef (0.0, 0.0, -5.0);   // along z-axis
glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis

Extrinsic Interpretation

Turning the Object: „Fly-Through“

Mathematical interpretation (“Fly-Through”)

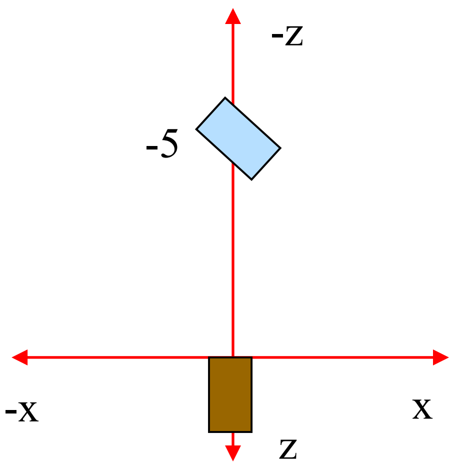


$$I * T1 * R1 * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

4.2 Composition of Transformations

Example

glTranslatef (0.0, 0.0, -5.0);   // along z-axis
glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis



Extrinsic Interpretation

Turning the Object: „Fly-Through“

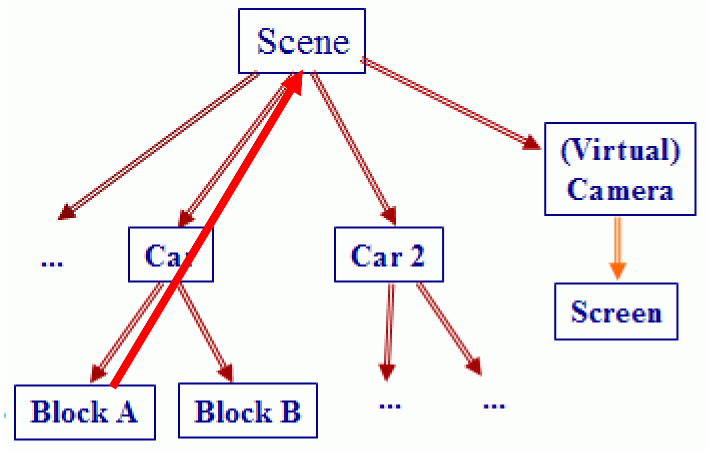
Mathematical interpretation (“Fly-Through”)

$$I * T1 * R1 * \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

4.2 Composition of Transformations

Equivalent concepts:

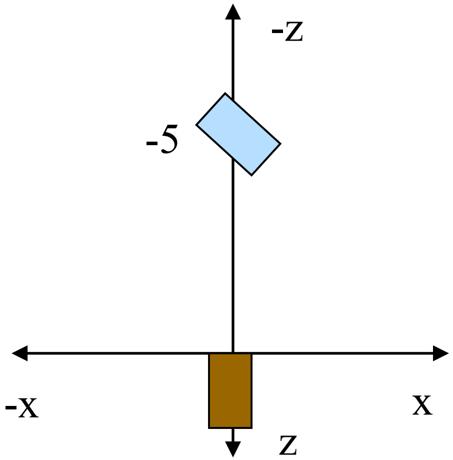
- Mathematical interpretation
- Extrinsic interpretation
- „Fly-through“
- Bottom-up traversal of the scene graph



4.2 Composition of Transformations

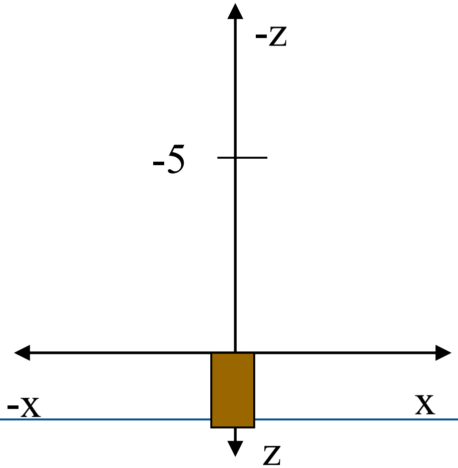
Example

glTranslatef (0.0, 0.0, -5.0);   // along z-axis
glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis

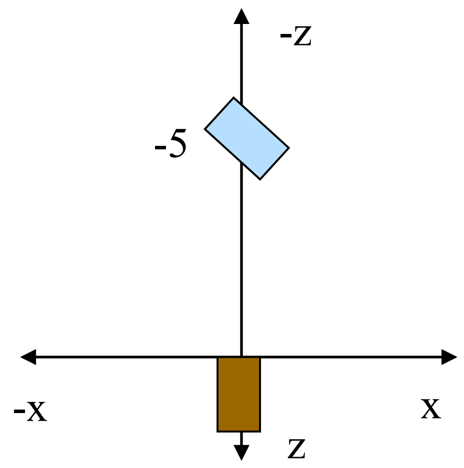


Question:

glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis
glTranslatef (0.0, 0.0, -5.0);   // along z-axis

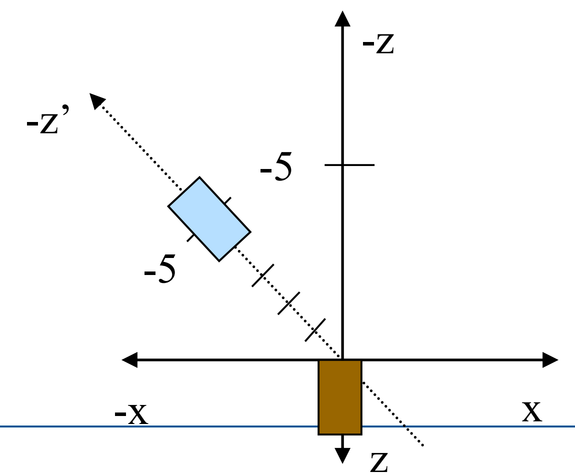


4.2 Composition of Transformations



Example

glTranslatef (0.0, 0.0, -5.0); // along z-axis
glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis



Question:

glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis
glTranslatef (0.0, 0.0, -5.0); // along z-axis

4.2 Composition of Transformations

This is important for reacting to user interactions.

- In a 3D-viewer:
  - Interactive transformations
    R-R-T-T-T-R-T-R-R-T
  - Add a new rotation in “examine mode”
    R-R-T-T-T-R-T-R-R-T-R
  - Add a new rotation in “fly-through mode”
    R-R-R-T-T-T-R-T-R-R-T

4. 3D Transformations

4.1 Euclidean Transformations

4.2 Composition of Transformations

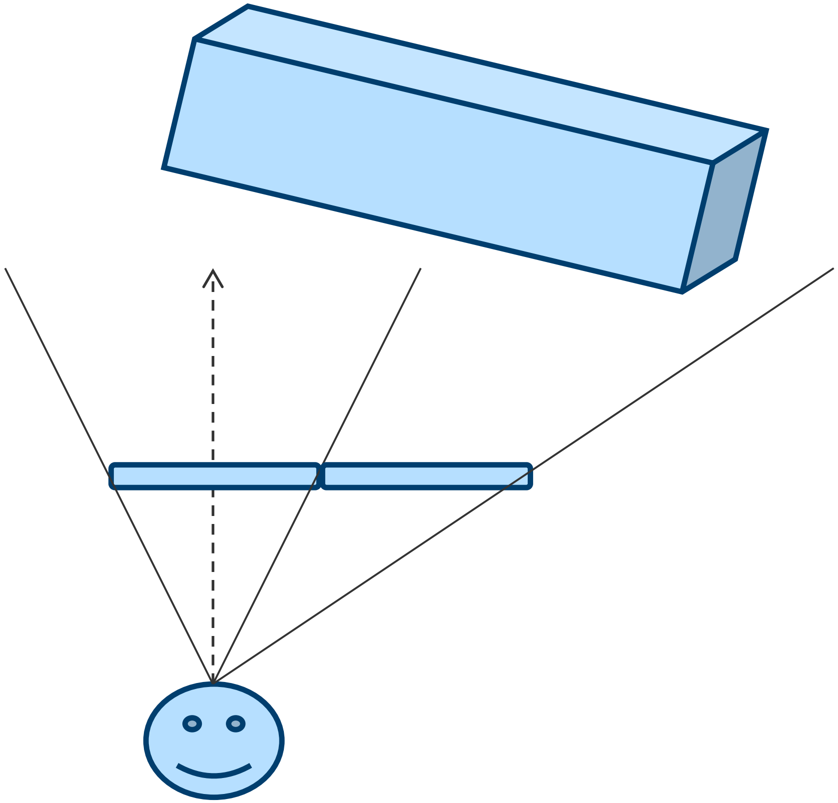
4.3 Projections and Viewport Transformations

4.4 Vertex Transformation Stages in OpenGL

4.5 Sample Code (OpenGL)

4.3 Projections and Viewport Transformations

Viewing frustum

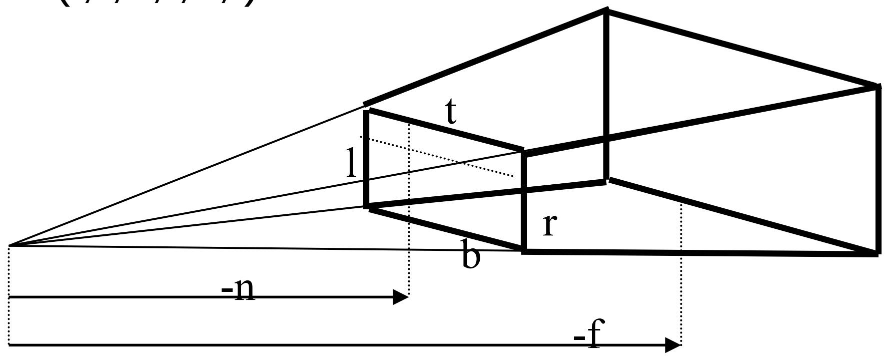


Note: The viewing direction is defined to be perpendicular to the display plane

4.3 Projections and Viewport Transformations

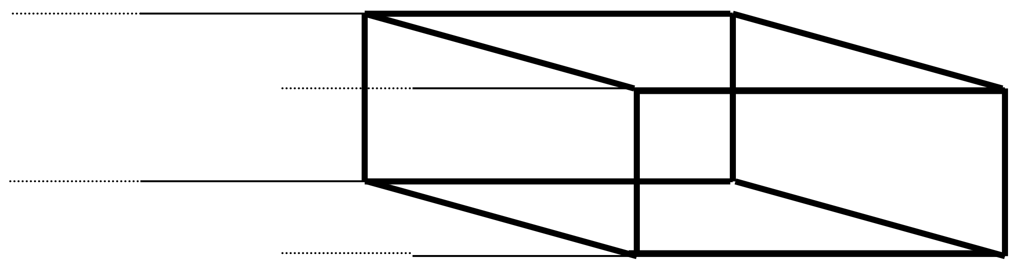
- Perspective Projection: glFrustum* (l,r,b,t,n,f)

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



- Orthographic Projection: glOrtho* (l,r,b,t,n,f)

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

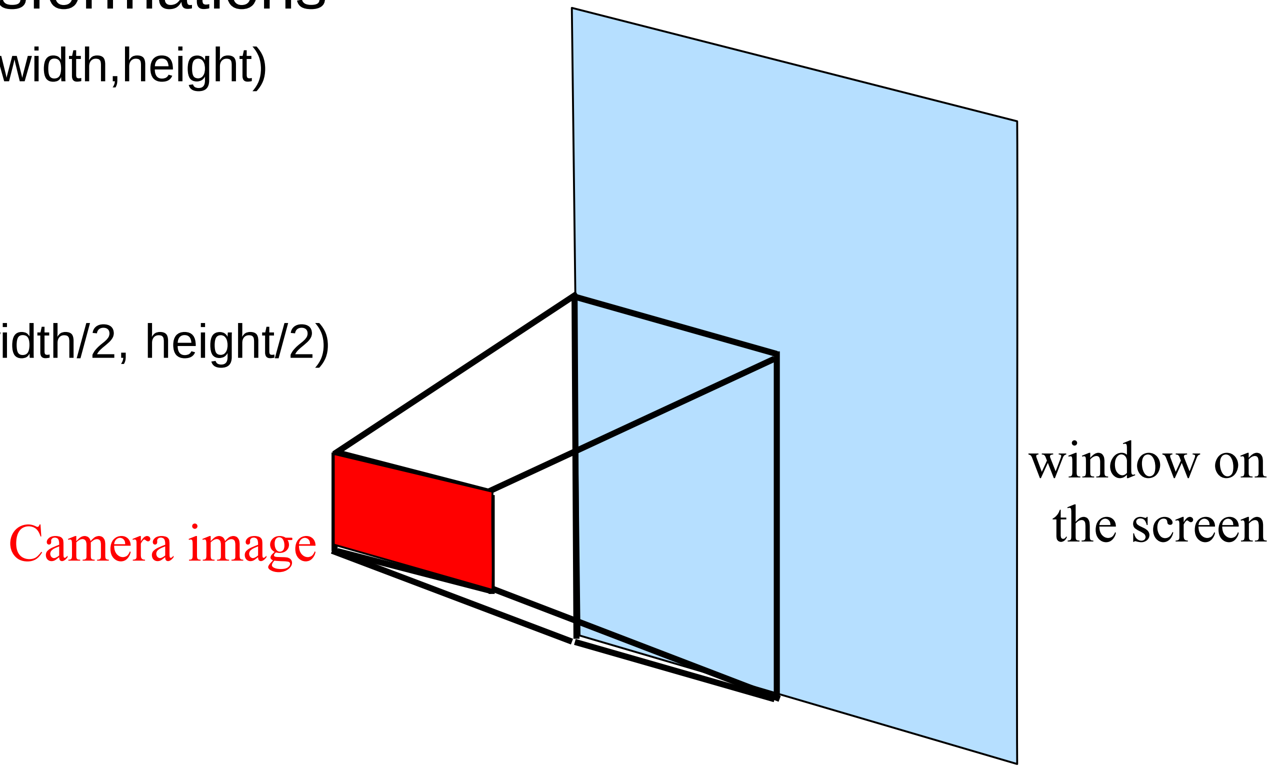


4.3 Projections and Viewport Transformations

Viewport Transformations

- glViewport (x,y,width,height)

- Example:
  Viewport (0,0,width/2, height/2)



4. 3D Transformations

4.1 Euclidean Transformations

4.2 Composition of Transformations

4.3 Projections and Viewport Transformations

→ 4.4 Vertex Transformation Stages in OpenGL

4.5 Sample Code (OpenGL)

4.4 Vertex Transformation Stages in OpenGL

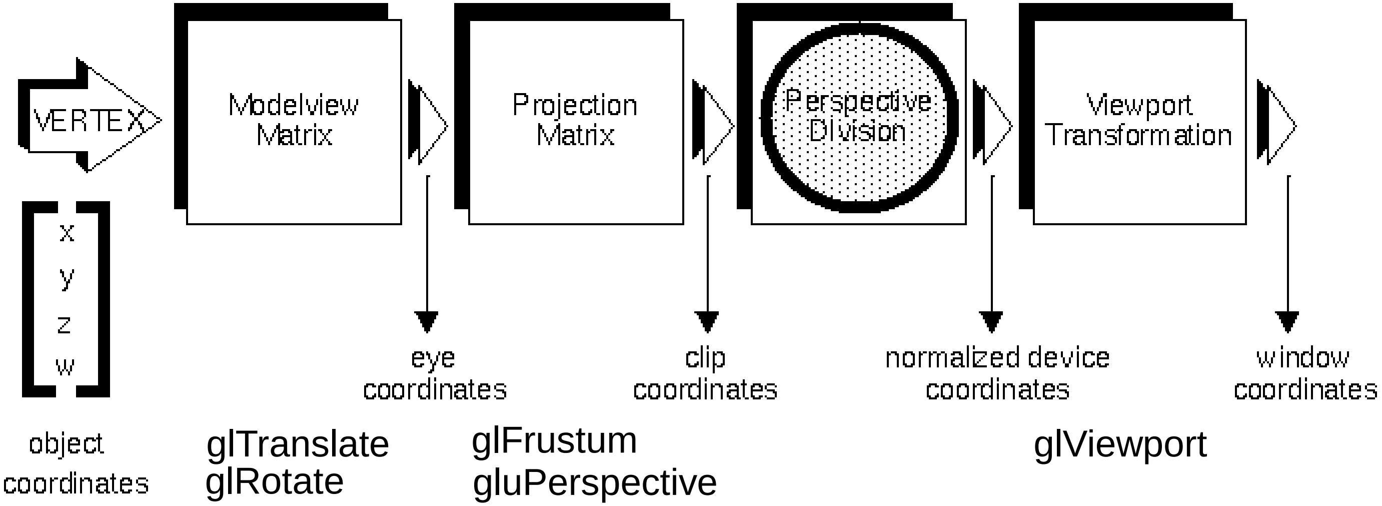
Modeling Transformation

Viewing Transformation

Projection Transformation

Clipping

Viewport Transformation



glTranslate
glRotate
glScale

glFrustum
gluPerspective
glOrtho

glViewport

in the OpenGL server

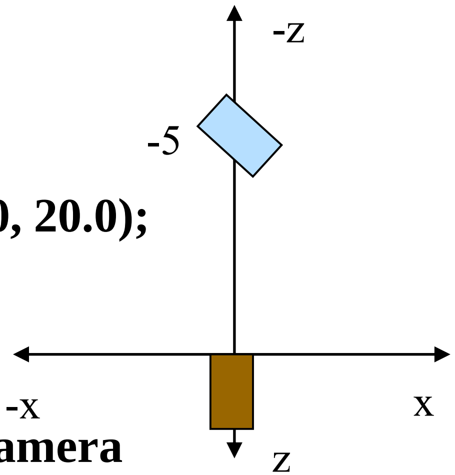
- (generally on the graphics card)...

4. 3D Transformations

4.1 Euclidean Transformations
4.2 Composition of Transformations
4.3 Projections and Viewport Transformations
4.4 Vertex Transformation Stages in OpenGL
→ 4.5 Sample Code (OpenGL)

4.5 Sample Code (OpenGL)

```
// Viewport Transformations
glViewport (0.0, 0.0, w, h);
// Projection Transformations
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glFrustum (l,r,t,b,n,f);   or gluPerspective (60.0, w/h, 1.0, 20.0);
// Viewing Transformations
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
glTranslatef (0.0, 0.0, -5.0);  // move world away from camera
glRotatef (45.0, 0.0, 1.0, 0.0);  // rotate world in front of the camera around y-axis
…
// Modeling Transformations
…
// Draw Object
…
```



Agenda

1. Scene Graph
2. 2D Transformations
3. Projective Geometry (2D)
4. 3D Transformations
5. Spatial Relationship Graphs (SRGs), Data Flow Networks (DFNs), and Spatial Relationship Patterns

5. SRGs, DFNs, Patterns

→ 5.1 Spatial Relationship Graphs (SRG)

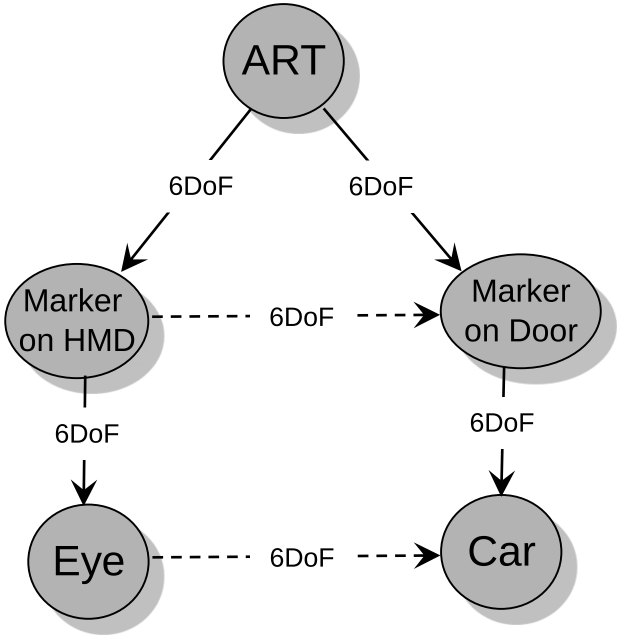
5.2 Compilation into Data Flow Networks

5.3 Spatial Relationship Patterns

5.4 Integration Concept for Applications

5.5 Example: Virtual Window

5.1 Spatial Relationship Graphs (SRG)

3D World	SRG
	

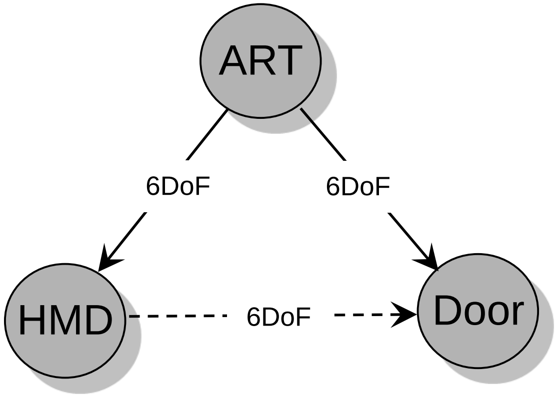
5.1 Spatial Relationship Graphs (SRG)

3D World	SRG
Objects • Things (physical or digital) • Users • Sensors	**Nodes** • Coordinate systems („Pose“)
Spatial relationships	**Edges** • Transformations
Time-dependent relationships • Static • Dynamic	**Properties of edges** • Registation • Calibration • Tracking
Operations • Sensor Fusion	**Properties of paths / cycles** • Graph traversal (one or more parallel paths)

5.1 Spatial Relationship Graphs (SRG)

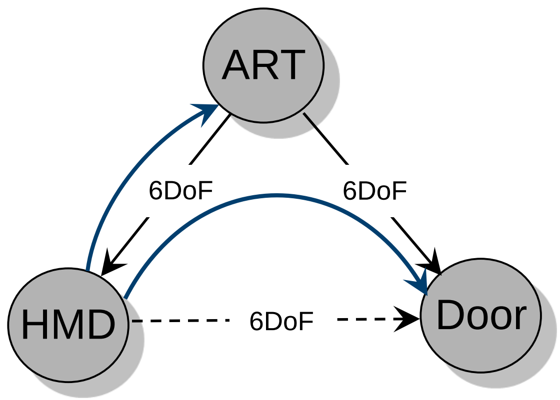
Edge Attributes

- Estimation method (direct ⟶, derived - - - -➝)
- Degrees of freedom (6 DoF, 3 DoF, 2 DoF, …)
- Transformation parameters (pose, translation, rotation, projection 3D → 2D, …)
- Dependence on time (static, dynamic)
- Time stamps
- Synchronization (push, pull)
- Precision, accuracy

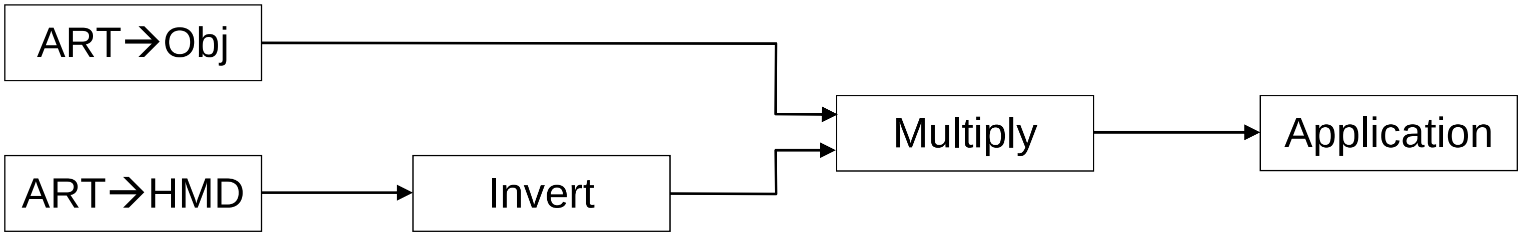


5.2 Compilation into Data Flow Networks

- Spatial Relationship Graph (SRG)

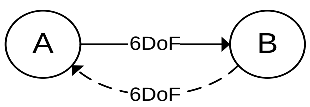


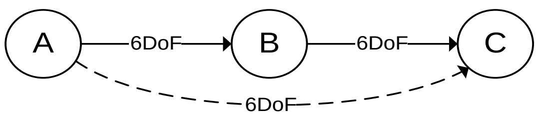
- Data Flow Netzwerk (DFN): SRG edges = DFN nodes

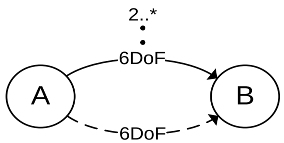


5.3 Spatial Relationship Patterns

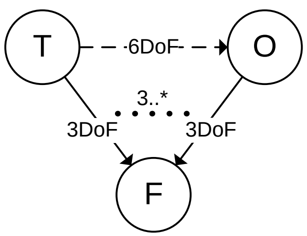
- Inversion Concatenation Sensor Fusion

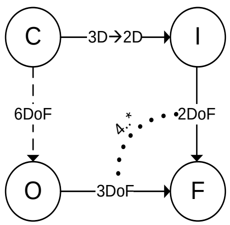


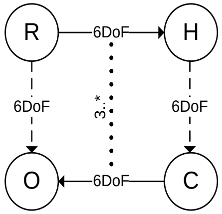




- 3D-3D Pose Estimation 2D-3D Pose Estimation Hand-Eye Calibration

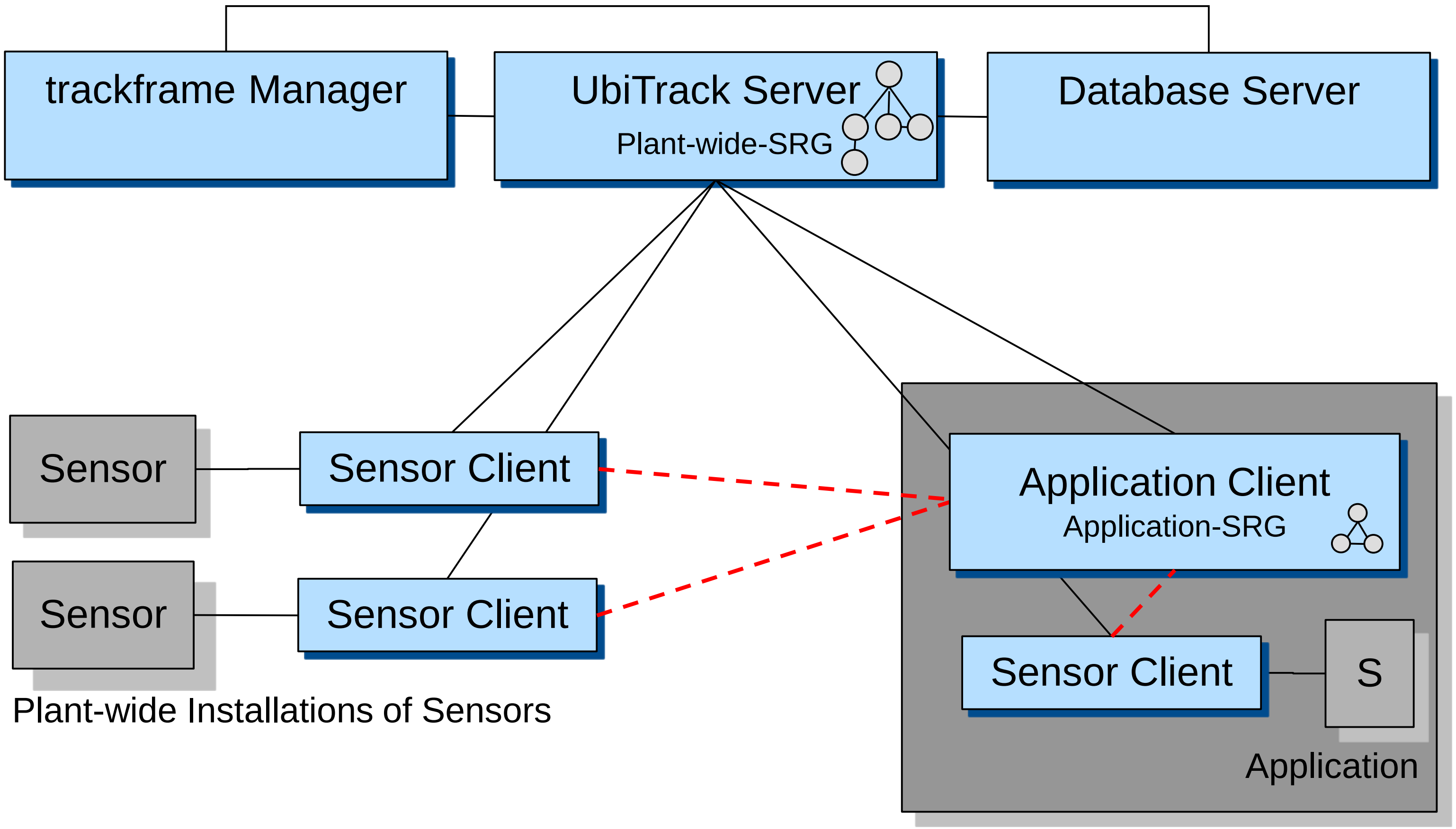






Pustka, Huber, Bauer and Klinker: *Spatial Relationship Patterns: Elements of Reusable Tracking and Calibration Systems*, ISMAR 06, Oct. 2006. AWARD.

5.4 Integration Concept for Applications



Huber, Pustka, Keitler, Echtler and Klinker: *A System Architecture for Ubiquitous Tracking Environments*, to be presented at ISMAR 07, Nov. 2007.

Huber, Becker and Klinker: *Location aware computing using RFID infrastructure*. Int. J. Autonomous and Adaptive Communications Systems, Vol. 3, No. 1, 2010.

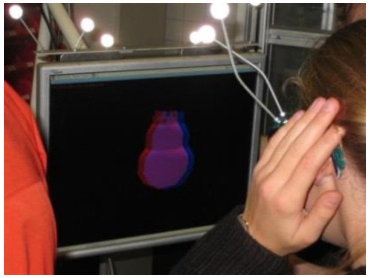
5.5 Example: Virtual Window

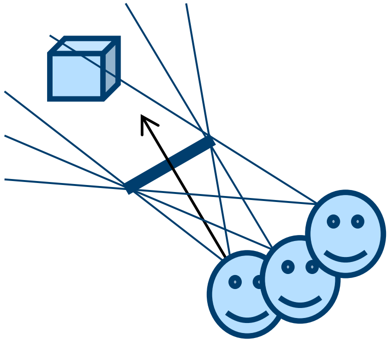
Example: Virtual window embedded into the physical world
[SEP Heuser]

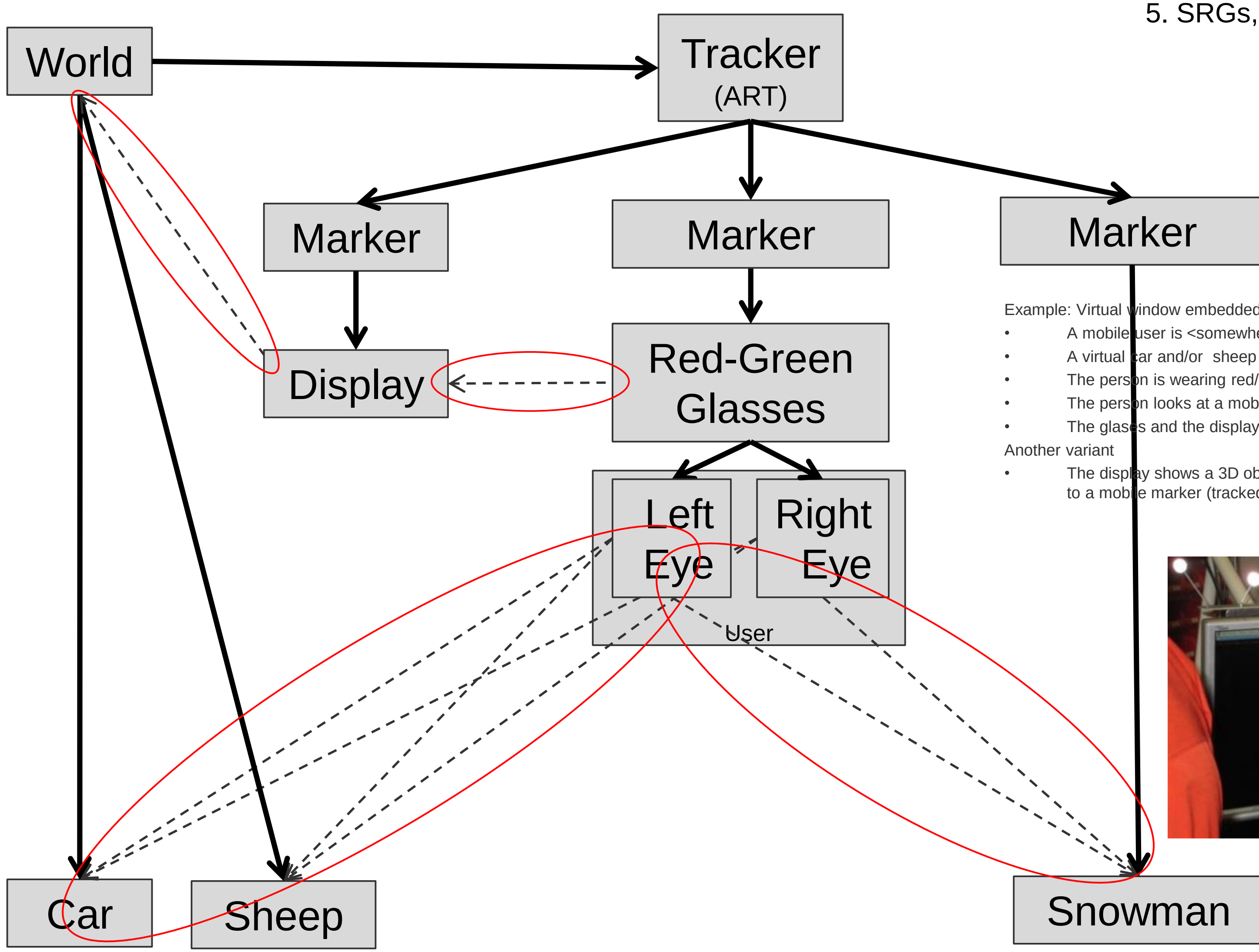
- A person is wearing red/green glasses (tracked)
- The person looks at a mobile stereo display (tracked)
- The display shows a 3D object (car, sheep) positioned at a fixed position in the world
- The user can move around freely, and the display can also be moved. The object is rendered according to the changing viewing frustum (defined by the current poses of the glasses and the display).

Another variant

- The display shows a 3D object (snowman), attached to a mobile marker (tracked)





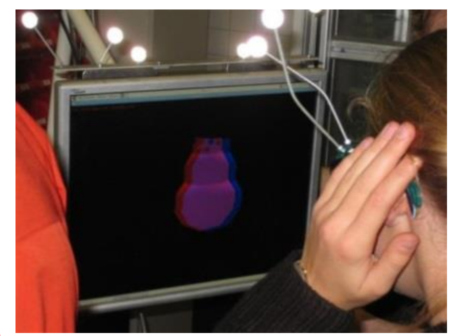


Example: Virtual window embedded into the physical world

- A mobile user is <somewhere> in a physical world
- A virtual car and/or sheep exist in this world
- The person is wearing red/green glasses
- The person looks at a mobile stereo display
- The glasses and the display have markers and are tracked

Another variant

- The display shows a 3D object (snowman), attached to a mobile marker (tracked)



6. Problems to think about

- Why might it be helpful to use a scene graph?
- Isn‘t it very time consuming to use a scene graph in computer graphics?
- Do P1 (1, 0.5, 3.0) and P2 (2, 1, 7) represent the same points in inhomogeneous coordinates?
- Compute the intersection of lines L1 (1,1,1) and L2 (-1,3,2).
- Compute and interpret the intersection of lines L1 (3,4,1) and L2 (6,8,1).
- What are ideal points?
- What is the duality of points and lines? Why is this useful?
- What is the result of applying the following transformations to P (1,1,1):
  glTranslatef (0.0, 0.0, -5.0); // along z-axis
  glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis
- What is the result of applying the following transformations to P (1,1,1):
  glRotatef (45.0, 0.0, 1.0, 0.0); // around y-axis
  glTranslatef (0.0, 0.0, -5.0); // along z-axis
- Should you get the same results? If not, why not?
- Why do we need a spatial relationship graph in AR? Wouldn’t the simpler Scene Graph be enough?

Thank you!

More on Projective Geometry

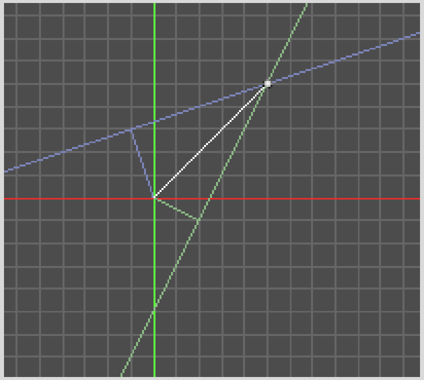
2D Projective Plane
- Comparison -

R^2	P^2
• Degrees of Freedom (*DOF*): 2	• Degrees of Freedom (DOF): 2
• Point P = (x,y)	• Vector **x** = w(x,y,1)^T
• Line l: ax+by+c=0 Normal **n**=(a,b)/|**n**|	• Vector **l** = k(a,b,c)^T
	• *Duality*: $\mathbf{x}^{\mathrm{T}}\mathbf{l} = \mathbf{l}^{\mathrm{T}}\mathbf{x} = 0$

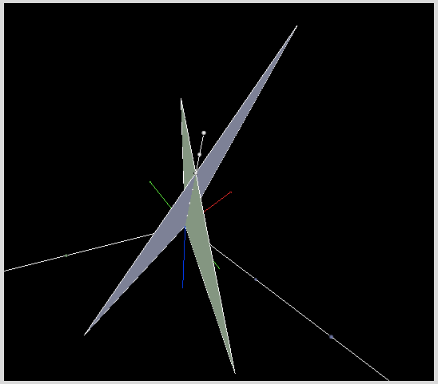
2D Projective Plane
- Intersection of Lines -

- Lines $\mathbf{l_1} = (a_1,b_1,c_1)^T$ and $\mathbf{l_2} = (a_2,b_2,c_2)^T$ intersect at a point $\mathbf{x} = (x,y,w)^T$.
- $\mathbf{x}$ is on $\mathbf{l_1}$ and on $\mathbf{l_2}$: $\mathbf{x}^{\mathrm{T}}\mathbf{l}_1 = 0$, $\mathbf{x}^{\mathrm{T}}\mathbf{l}_2 = 0$
- $\mathbf{x}$ is perpendicular to $\mathbf{l_1}$ and $\mathbf{l_2}$.
- $\mathbf{x}$ is cross product of $\mathbf{l_1}$ and $\mathbf{l_2}$.

$$\mathbf{x} = \mathbf{l_1} \times \mathbf{l_2} = \begin{vmatrix} i & j & k \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{vmatrix} = \begin{pmatrix} b_1c_2 - c_1b_2 \\ c_1a_2 - a_1c_2 \\ a_1b_2 - b_1a_2 \end{pmatrix}$$



Example



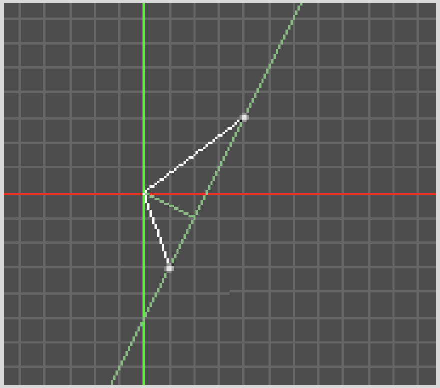
$\mathbb{R}^2$	$\mathbb{P}^2$
$l_2 : 2x - y - 0.5 = 0$	$\mathbf{l_2} = (-4.0, \quad 2.0, \quad 1.0)^T$
$l_3 : x - 3y + 1 = 0$	$\mathbf{l_3} = (1.0, \quad -3.0, \quad 1.0)^T$
$P_3 = (0.5 \quad 0.5)^T$	$\mathbf{x_3} = (0.5, \quad 0.5, \quad 1.0)^T$

$$\begin{pmatrix} -4 \\ 2 \\ 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ -3 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 10 \end{pmatrix}$$

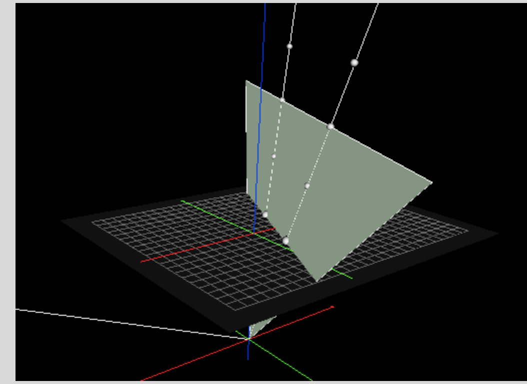
2D Projective Plane
- Line through 2 Points -

- Points $\mathbf{x_1} = (x_1,y_1,w_1)^T$ and $\mathbf{x_2} = (x_2,y_2,w_2)^T$ define a line $\mathbf{l} = (a,b,c)^T$.
- $\mathbf{l}$ goes through $\mathbf{x_1}$ and $\mathbf{x_2}$: $\mathbf{x_1}^T\mathbf{l} = 0$, $\mathbf{x_2}^T\mathbf{l} = 0$
- $\mathbf{l}$ is perpendicular to both vectors.
- $\mathbf{l}$ is cross product of $\mathbf{x_1}$ and $\mathbf{x_2}$.

$$\mathbf{l} = \mathbf{x}_1 \times \mathbf{x}_2$$



Example



$\mathbb{R}^2$ | $\mathbb{P}^2$

$$P_1 = (0.4, \quad 0.3)^T \qquad \mathbf{x}_1 = (0.4, \quad 0.3, \quad 1.0)^T$$

$$P_2 = (0.1, \quad -0.3)^T \qquad \mathbf{x}_2 = (0.1, \quad -0.3, \quad 1.0)^T$$

$$l_2 : 2x - y - 0.5 = 0 \qquad \mathbf{l}_2 = (2.0, \quad -1.0, \quad -0.5)^T$$

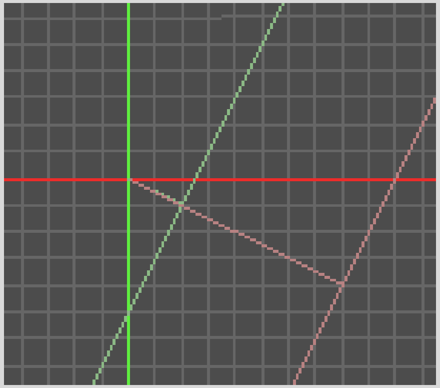
$$\begin{pmatrix} 0.4 \\ 0.3 \\ 1.0 \end{pmatrix} \times \begin{pmatrix} 0.1 \\ -0.3 \\ 1.0 \end{pmatrix} = \begin{pmatrix} 0.6 \\ -0.3 \\ -0.15 \end{pmatrix} = -6.66 \begin{pmatrix} -4 \\ 2 \\ 1 \end{pmatrix}$$

2D Projective Plane
- Ideal Points -

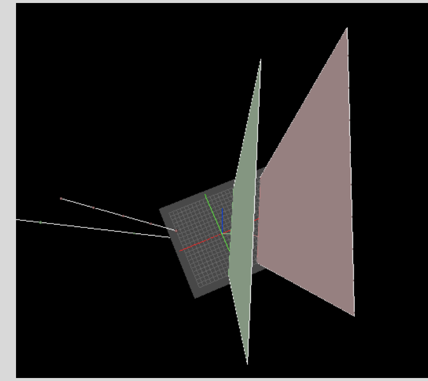
- Intersection of parallel lines $\mathbf{l_1} = (a,b,c)^T$ and $\mathbf{l_2} = (a,b,c')^T$

$$\mathbf{x} = \mathbf{l}_1 \times \mathbf{l}_2 = (c' - c)\begin{pmatrix} b \\ -a \\ 0 \end{pmatrix}$$

- Parallel lines intersect “at infinity”.
- *Ideal points* lie on plane w=0
(*Points at infinity*).



Example



$\mathbb{R}^2$ | $\mathbb{P}^2$

$$l_1 : 2x - y - 2 = 0$$

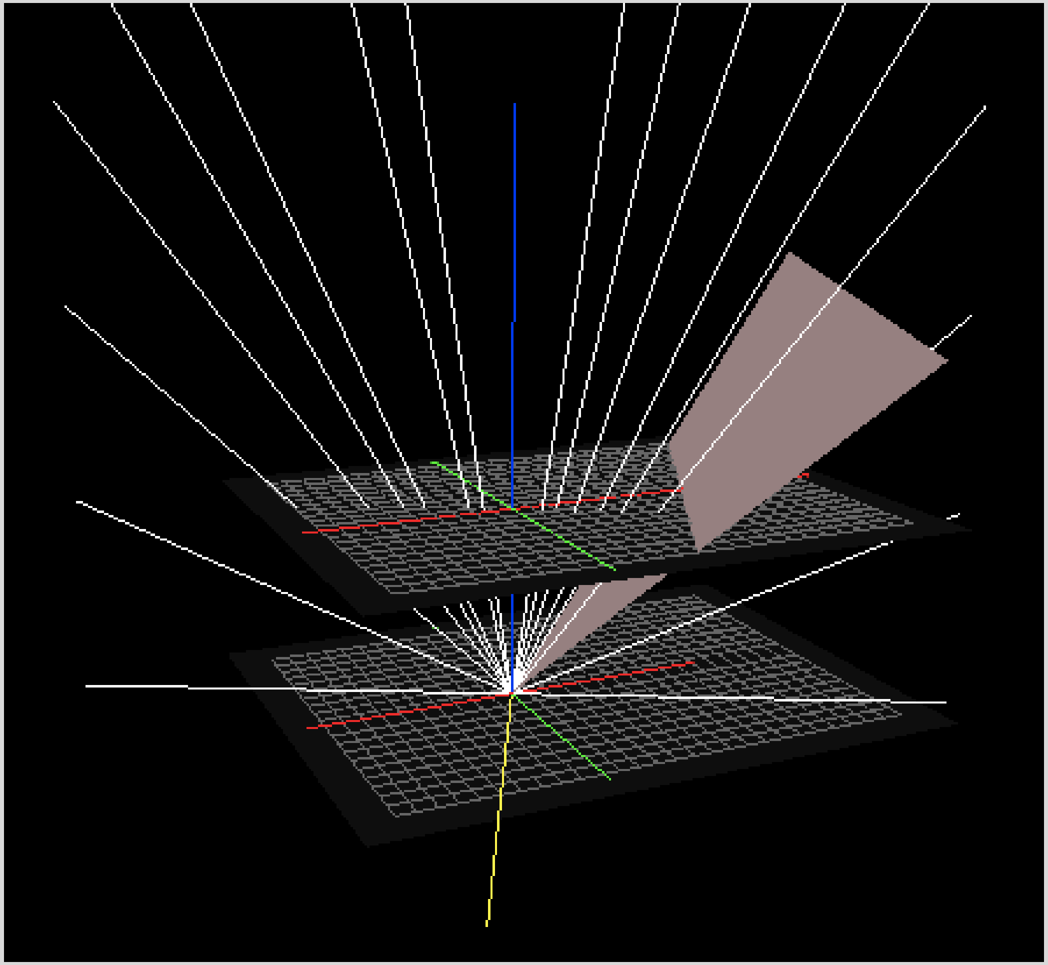
$$\mathbf{l_2} = \begin{pmatrix} 2, & -1, & -2 \end{pmatrix}^{\mathrm{T}}$$

$$l_2 : 2x - y - 0.5 = 0$$

$$\mathbf{l_3} = \begin{pmatrix} 2, & -1, & -0.5 \end{pmatrix}^{\mathrm{T}}$$

$$\begin{pmatrix} 2 \\ -1 \\ -2 \end{pmatrix} \times \begin{pmatrix} 2 \\ -1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} -1.5 \\ -3 \\ 0 \end{pmatrix} = 1.5 \begin{pmatrix} -1 \\ -2 \\ 0 \end{pmatrix}$$

2D Projective Plane
- Ideal Points -

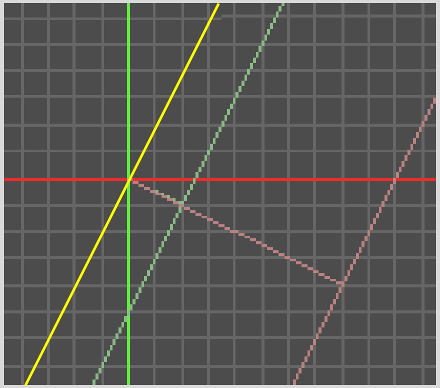


2D Projective Plane
- Points at Infinity -

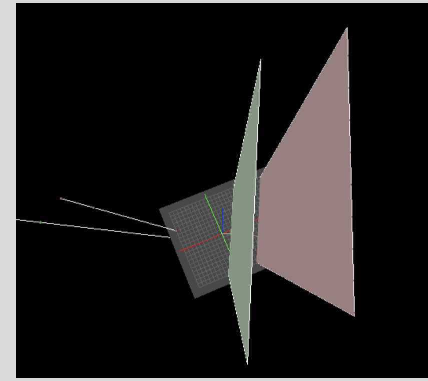
- Set of all ideal points (*points at infinity*):

$$\mathbf{x}_{\mathbf{Id\_i}} = (x_i, y_i, 0)^T$$
$$= s(x_i/y_i, 1, 0)^T$$

i.e.: all ideal points lie in plane, w = 0.

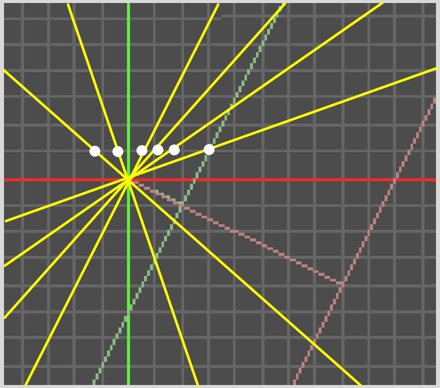


Example



$\mathbb{R}^2$	$\mathbb{P}^2$
$l_1 : 2x - y - 2 = 0$	$\mathbf{l}_2 = (2, \quad -1, \quad -2)^T$
$l_2 : 2x - y - 0.5 = 0$	$\mathbf{l}_3 = (2, \quad -1, \quad -0.5)^T$

$$\begin{pmatrix} 2 \\ -1 \\ -2 \end{pmatrix} \times \begin{pmatrix} 2 \\ -1 \\ -0.5 \end{pmatrix} = \begin{pmatrix} -1.5 \\ -3 \\ 0 \end{pmatrix} = 1.5 \begin{pmatrix} -1 \\ -2 \\ 0 \end{pmatrix} = -3 \begin{pmatrix} 0.5 \\ 1 \\ 0 \end{pmatrix}$$



2D Projective Plane
- Line at Infinity -

- Set of all ideal points (*points at infinity*):
  $\mathbf{x}_{\mathbf{Id\_i}} = (x_i, y_i, 0)^T = s(x_i/y_i, 1, 0)^T$
  i.e.: all ideal points lie in plane, w = 0.
- The *line at infinity* represents all ideal points.

$$\mathbf{l}_\infty = \mathbf{x}_{Id_1} \times \mathbf{x}_{Id_2} = \begin{pmatrix} m_1 \\ 1 \\ 0 \end{pmatrix} \times \begin{pmatrix} m_2 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ m_1 - m_2 \end{pmatrix} = t \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Normal to the plane w=0.
Set of the directions of all lines in the plane.