Image Processing and Computer Graphics

Projections and Transformations in OpenGL

Matthias Teschner

Computer Science Department
University of Freiburg

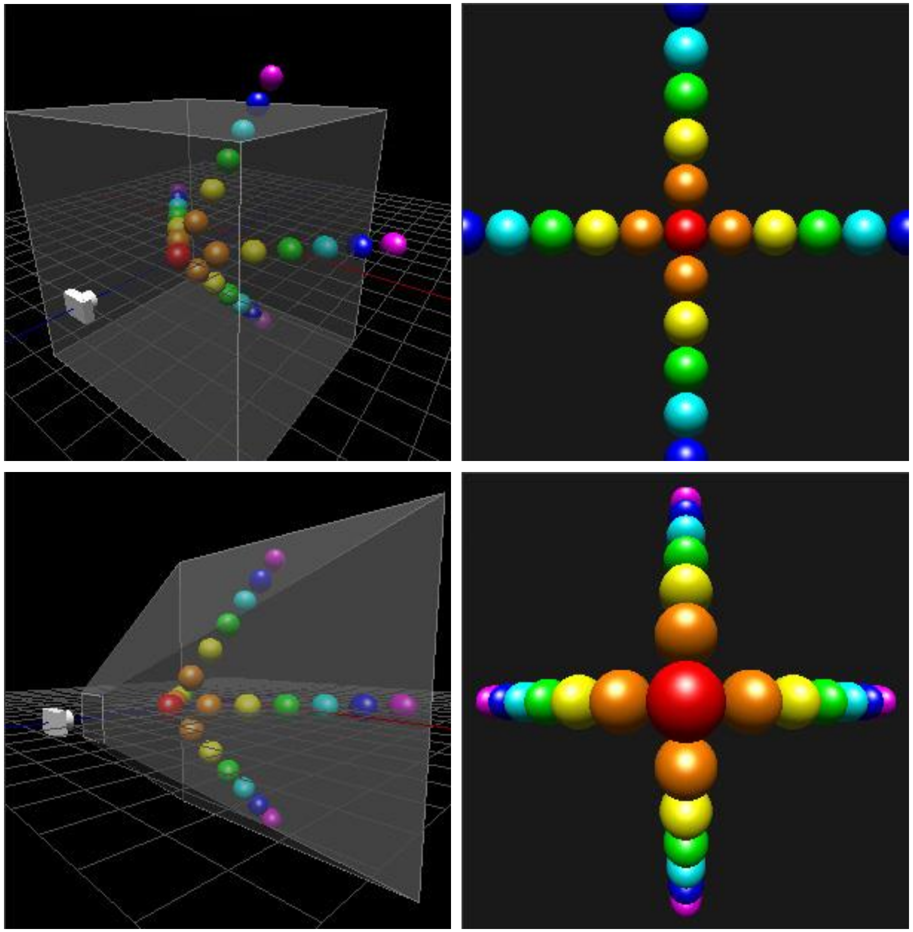
UNI FREIBURG

Motivation

- for the rendering of objects in 3D space,
  a planar view has to be generated
- 3D space is projected onto a 2D plane considering external and internal camera parameters
  - position, orientation, focal length
- in homogeneous notation, 3D projections can be represented with a 4x4 transformation matrix

Examples

- left images
  - 3D scene with a view volume
- right images
  - projections onto the viewplane
- top-right
  - parallel projection
- top-bottom
  - perspective projection



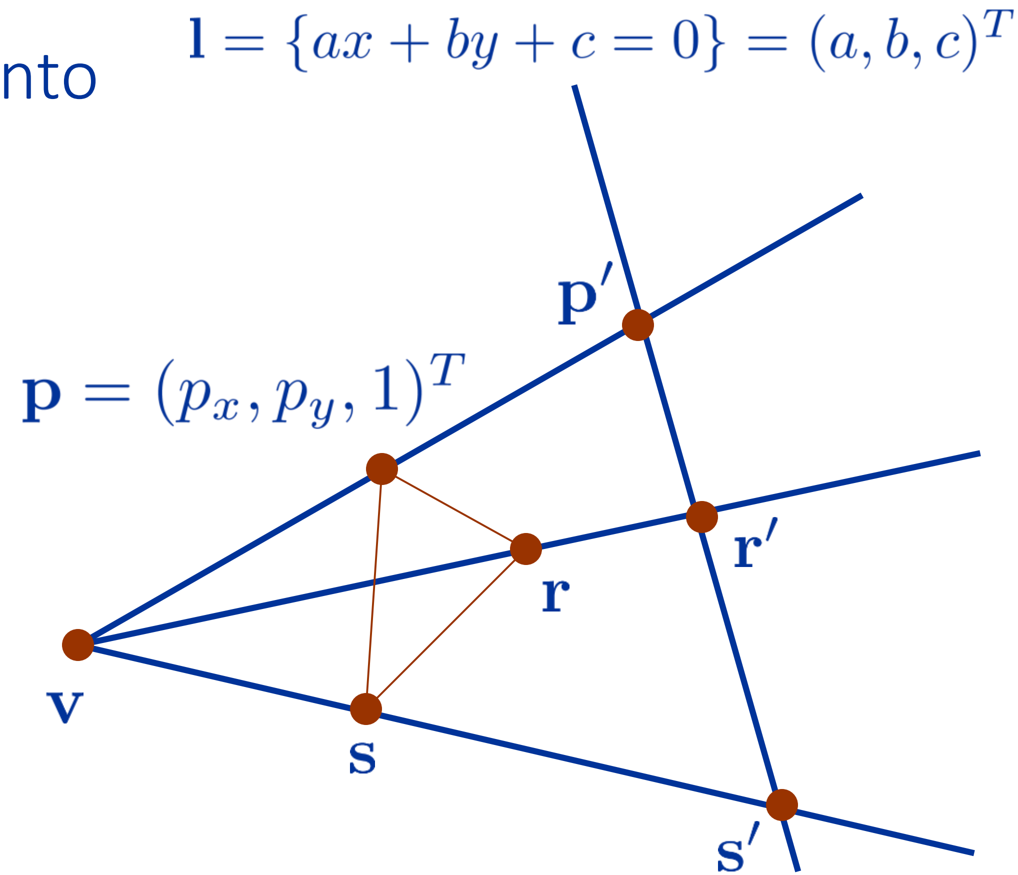
[Song Ho Ahn]

Outline

- 2D projection
- 3D projection
- OpenGL projection matrix
- OpenGL transformation matrices

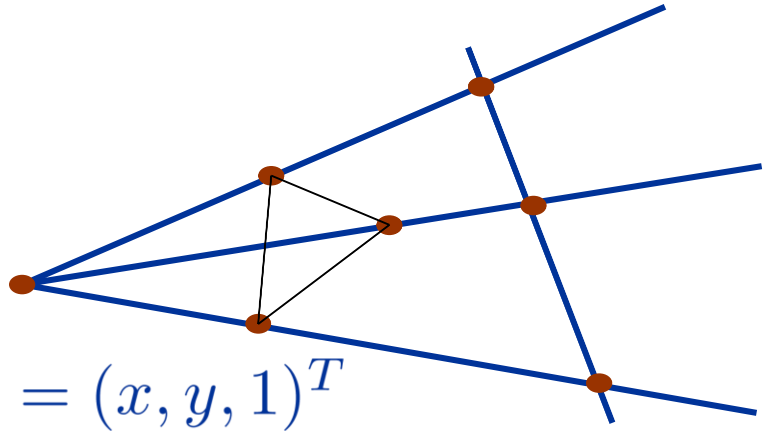
Projection in 2D

- a 2D projection from **v** onto **l** maps a point **p** onto **p'**
- **p'** is the intersection of the line through **p** and **v** with line **l**
- **v** is the viewpoint, center of perspectivity
- **l** is the viewline
- the line through **p** and **v** is a projector
- **v** is not on the line **l**, **p** $\neq$ **v**



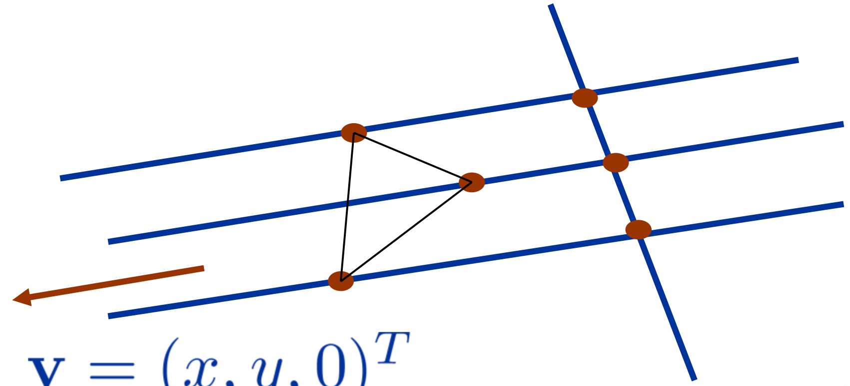
Projection in 2D

- if the homogeneous component of the viewpoint **v** is not equal to zero, we have a perspective projection
  - projectors are not parallel
- if **v** is at infinity, we have a parallel projection
  - projectors are parallel



$$\mathbf{v} = (x, y, 1)^T$$

perspective projection



$$\mathbf{v} = (x, y, 0)^T$$

parallel projection

Classification

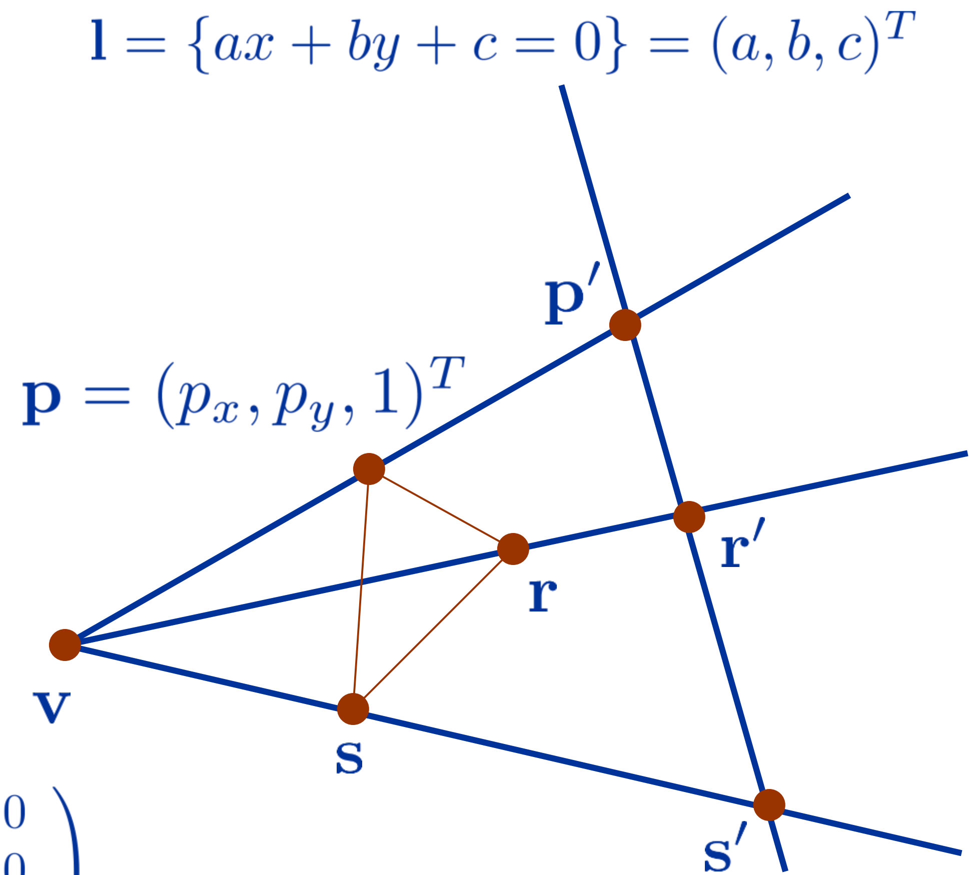
- location of viewpoint and orientation of the viewline determine the type of projection
- parallel (viewpoint at infinity, parallel projectors)
  - orthographic (viewline orthogonal to the projectors)
  - oblique (viewline not orthogonal to the projectors)
- perspective (non-parallel projectors)
  - one-point
    (viewline intersects one principal axis,
    i.e. viewline is parallel to a principal axis, one vanishing point)
  - two-point
    (viewline intersects two principal axis, two vanishing points)

General Case

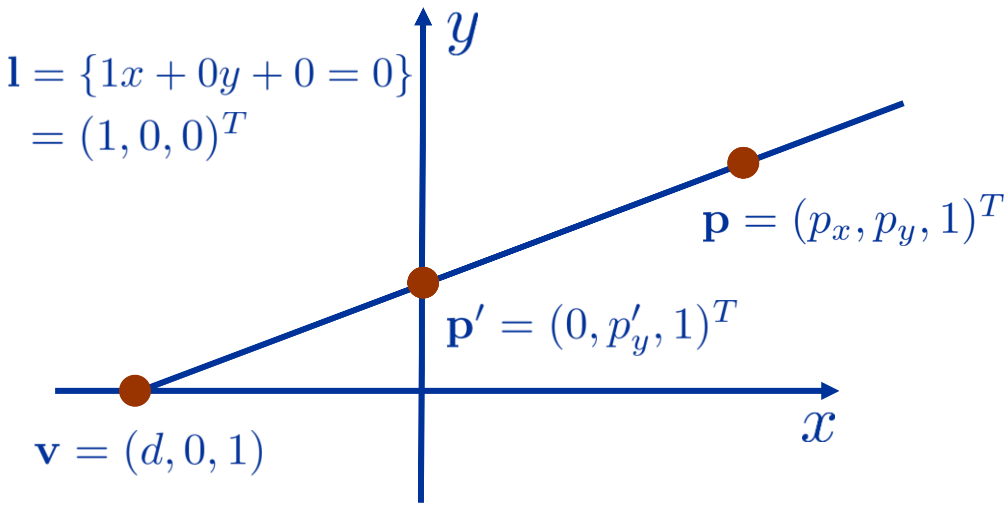
- a 2D projection is represented by matrix $\mathbf{M} = \mathbf{v}\mathbf{l}^T - (\mathbf{l} \cdot \mathbf{v})\mathbf{I}_3$

$$\mathbf{v}\mathbf{l}^T = \begin{pmatrix} v_x a & v_x b & v_x c \\ v_y a & v_y b & v_y c \\ v_w a & v_w b & v_w c \end{pmatrix}$$

$$(\mathbf{l} \cdot \mathbf{v})\mathbf{I} = (a v_x + b v_y + c v_w) \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Example

- 

- $$\mathbf{M} = \begin{pmatrix} d \\ 0 \\ 1 \end{pmatrix} (1, 0, 0) - \left(\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} d \\ 0 \\ 1 \end{pmatrix} \right) \mathbf{I}_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -d & 0 \\ 1 & 0 & -d \end{pmatrix}$$
- e.g. d=-1, $(1,2)^T$ is mapped to $(0,1)^T$

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

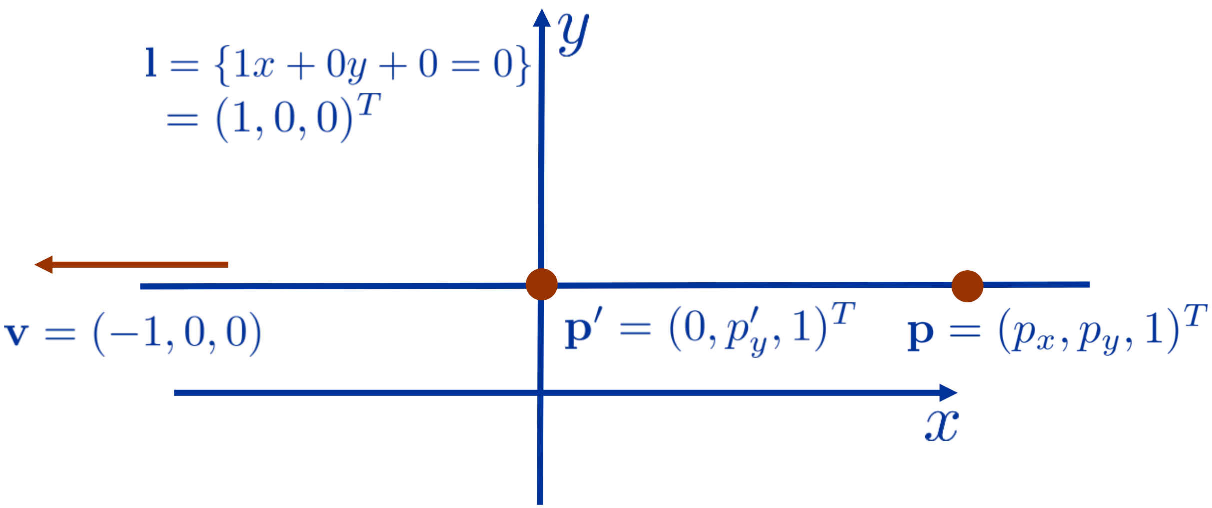
Discussion

- matrices **M** and λ**M** represent the same transformation $(\lambda \mathbf{M}\mathbf{p} = \lambda \mathbf{p}')$
- therefore, $\begin{pmatrix} 0 & 0 & 0 \\ 0 & -d & 0 \\ 1 & 0 & -d \end{pmatrix}$ and $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{d} & 0 & 1 \end{pmatrix}$ represent the same transformation
- $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{d} & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix} = \begin{pmatrix} 0 \\ y \\ -\frac{x}{d} + w \end{pmatrix} \sim \begin{pmatrix} 0 \\ \frac{y}{w - \frac{x}{d}} \end{pmatrix}$
- *x* is mapped to zero, *y* is scaled depending on *x*
- moving *d* to infinity results in parallel projection

$$\lim_{d \to \pm\infty} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{d} & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Discussion

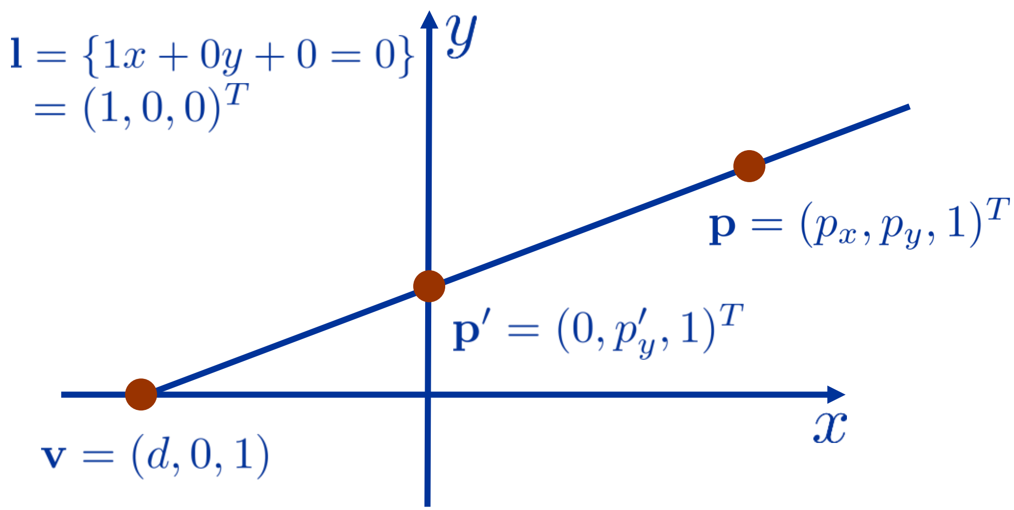
- parallel projection



$$\mathbf{M} = \mathbf{v}\mathbf{l}^T - (\mathbf{l} \cdot \mathbf{v})\mathbf{I}_3$$

$$\mathbf{M} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} (1, 0, 0) - \left(\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} \right) \mathbf{I}_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Discussion



$$p'_x = 0 \qquad \frac{p_y}{p_x - d} = \frac{p'_y}{-d} \Rightarrow p'_y = \frac{-dp_y}{p_x - d} \qquad p_w = 1 \Rightarrow p'_w = p_x - d$$

$$\Rightarrow \mathbf{M} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -d & 0 \\ 1 & 0 & -d \end{pmatrix}$$

maps **p** to $p'_x = 0$

maps **p** to $p'_y = -d\ p_y$

maps **p** with $p_w=1$ to $p'_w = p_x - d$

Discussion

- 2D transformation in homogeneous form

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & t_x \\ m_{21} & m_{22} & t_y \\ w_x & w_y & h \end{pmatrix}$$

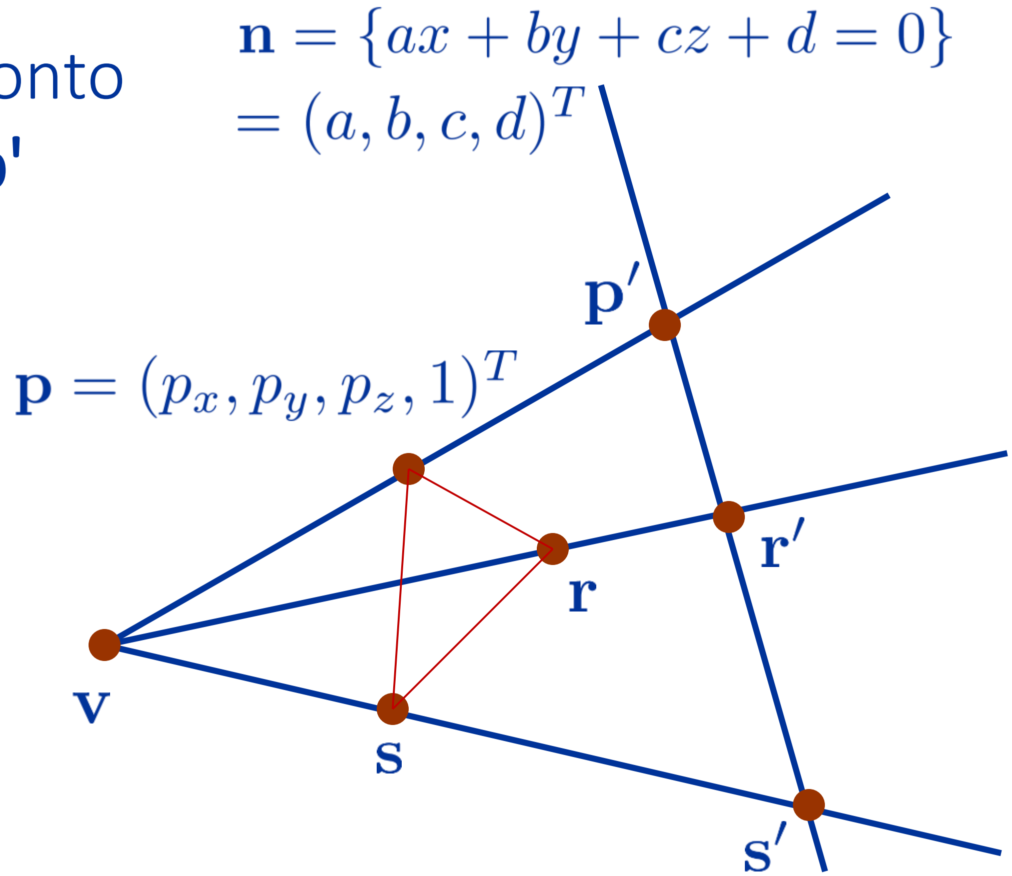
- w_x and w_y map the homogeneous component w of a point to a value w' that depends on x and y
- therefore, the scaling of a point depends on x and / or y
- in perspective 3D projections, this is generally employed to scale the x- and y- component with respect to z, its distance to the viewer

Outline

- 2D projection
- 3D projection
- OpenGL projection matrix
- OpenGL transformation matrices

Projection in 3D

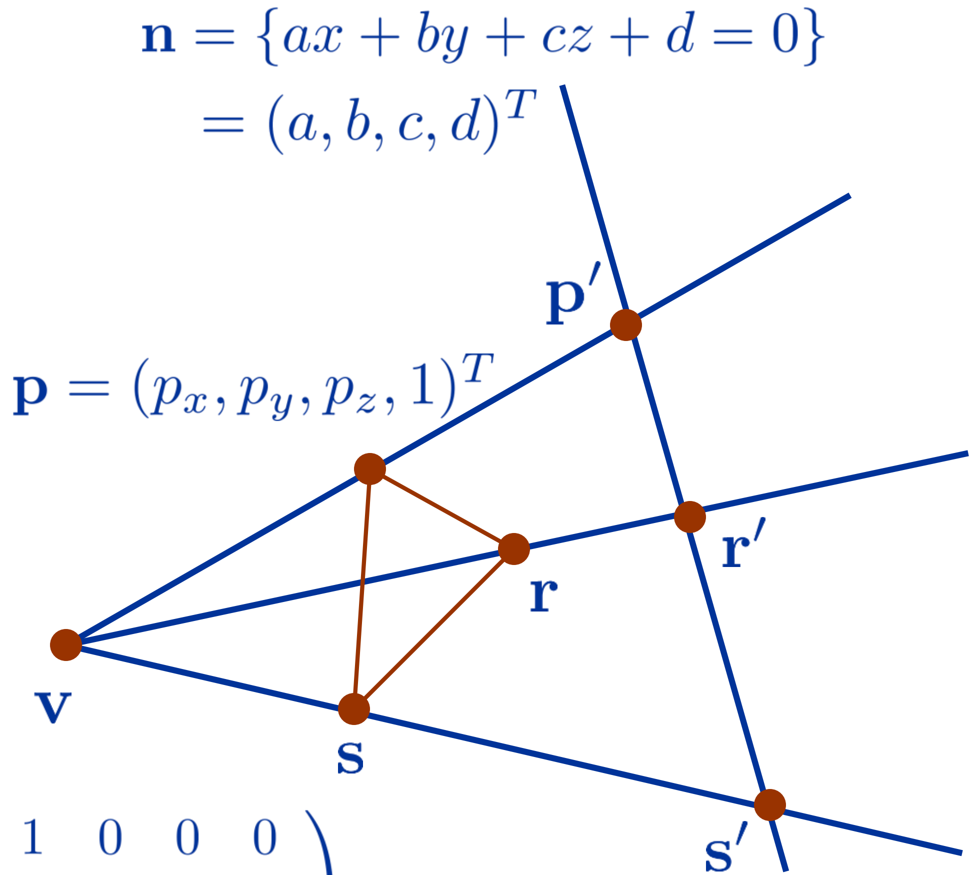
- a 3D projection from **v** onto **n** maps a point **p** onto **p'**
- **p'** is the intersection of the line through **p** and **v** with plane **n**
- **v** is the viewpoint, center of perspectivity
- **n** is the viewplane
- the line through **p** and **v** is a projector
- **v** is not on the plane **n**, **p** $\neq$ **v**



General Case

- a 3D projection is represented by a matrix

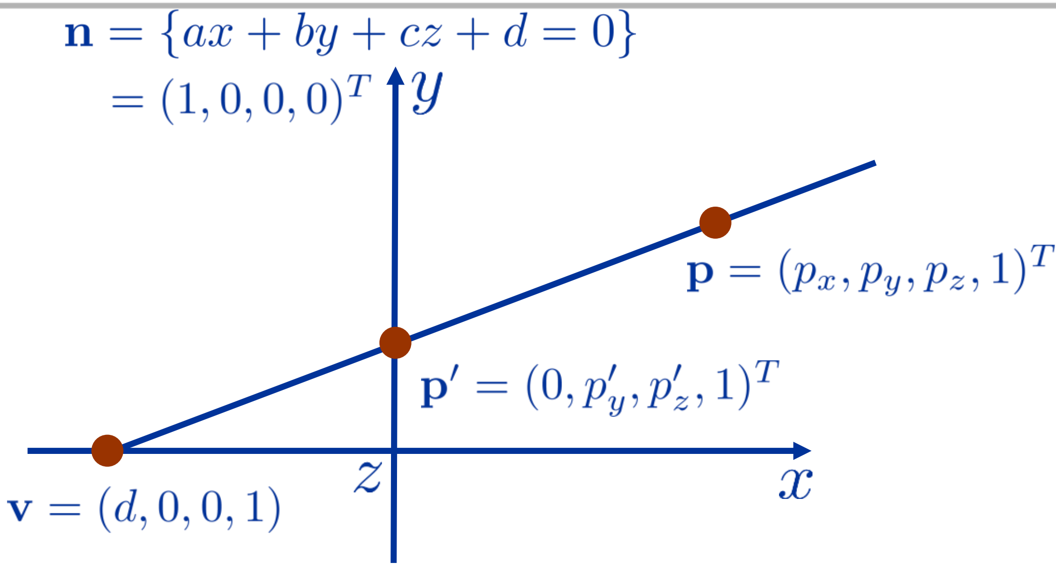
$$\mathbf{M} = \mathbf{v}\mathbf{n}^T - (\mathbf{n} \cdot \mathbf{v})\mathbf{I}_4$$



$$\mathbf{v}\mathbf{n}^T = \begin{pmatrix} v_x a & v_x b & v_x c & v_x d \\ v_y a & v_y b & v_y c & v_y d \\ v_z a & v_z b & v_z c & v_z d \\ v_w a & v_w b & v_w c & v_w d \end{pmatrix}$$

$$(\mathbf{n} \cdot \mathbf{v})\mathbf{I} = (av_x + bv_y + cv_z + dv_w) \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Example

- 

- $$\mathbf{M} = \begin{pmatrix} d \\ 0 \\ 0 \\ 1 \end{pmatrix} (1, 0, 0, 0) - \left(\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} d \\ 0 \\ 0 \\ 1 \end{pmatrix} \right) \mathbf{I}_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -d & 0 & 0 \\ 0 & 0 & -d & 0 \\ 1 & 0 & 0 & -d \end{pmatrix}$$

- e.g. d=-1, $(1,2,0)^T$ is mapped to $(0,1,0)^T$

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

Example

- parallel projection onto the plane *z* = 0 with viewpoint / viewing direction **v** = (0,0,1,0)^T

$$\mathbf{n} = \{0x + 0y + 1z + 0 = 0\}$$
$$\mathbf{v} = (0,0,1,0)^T$$
$$\mathbf{M} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} (0,0,1,0) - \left(\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right) \mathbf{I}_4 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

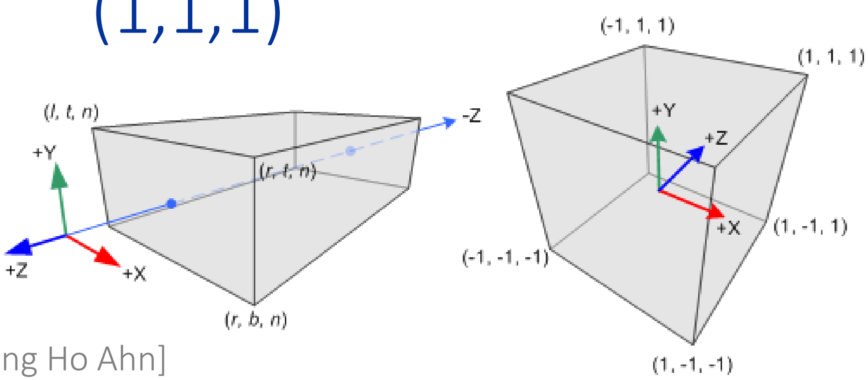
- *x*- and *y*-component are unchanged, *z* is mapped to zero
- remember that **M** and λ**M** with, e. g., λ=-1 represent the same transformation

Outline

- 2D projection
- 3D projection
- OpenGL projection matrix
  - perspective projection
  - parallel projection
- OpenGL transformation matrices

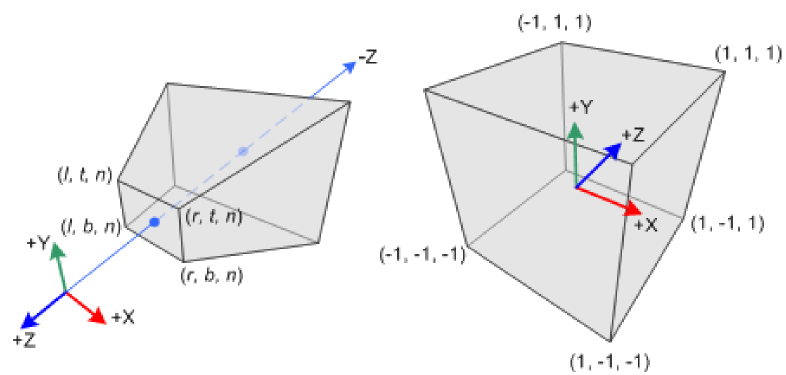
View Volume

- in OpenGL, the projection transformation maps a view volume to the canonical view volume
- the view volume is specified by its boundary
  - left, right, bottom, top, near far
- the canonical view volume is a cube from (-1,-1,-1) to (1,1,1)



[Song Ho Ahn]

this transformation implements orthographic projection



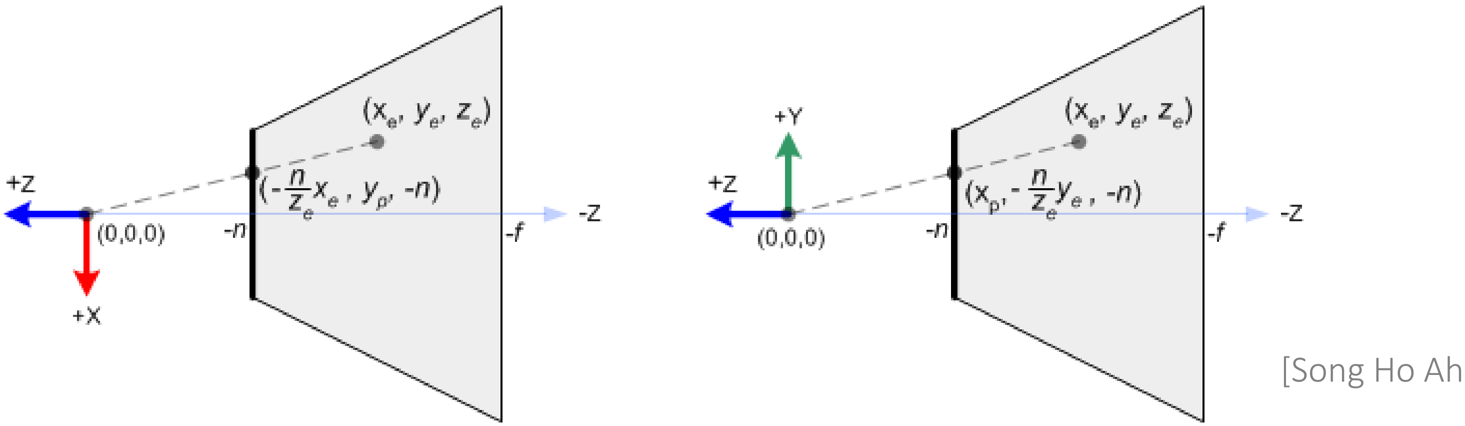
this transformation implements perspective projection

OpenGL Projection Transform

- the projection transform maps
  - from eye coordinates
  - to clip coordinates (w-component is not necessarily one)
  - to normalized device coordinates NDC
    (x and y are normalized with respect to w,
    w is preserved for further processing)
- the projection transform maps
  - the x-component of a point from (left, right) to (-1, 1)
  - the y-component of a point from (bottom, top) to (-1, 1)
  - the z-component of a point from (near, far) to (-1, 1)
    - in OpenGL, near and far are negative, so the mapping incorporates a reflection (change of right-handed to left-handed)
    - however, in OpenGL functions, usually the negative of near and far is specified which is positive

Perspective Projection

- to obtain x- and y-component of a projected point, the point is first projected onto the near plane (viewplane)

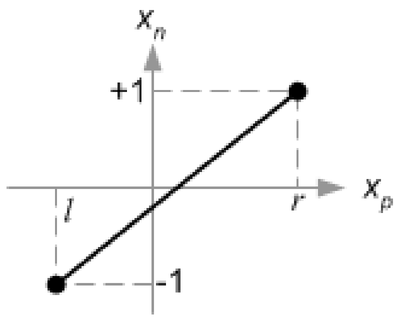


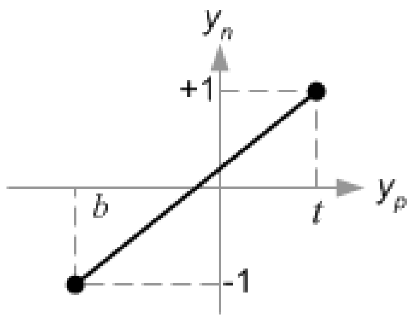
[Song Ho Ahn]

$$\frac{x_p}{x_e} = \frac{-n}{z_e} \Rightarrow x_p = \frac{nx_e}{-z_e} \qquad \frac{y_p}{y_e} = \frac{-n}{z_e} \Rightarrow y_p = \frac{ny_e}{-z_e}$$

- note that n and f denote the negative near and far values

Mapping of x_p and y_p to (-1, 1)





[Song Ho Ahn]

$$x_n = \alpha x_p + \beta$$

$$\alpha = \frac{1-(-1)}{r-l}$$

$$\beta = -\frac{r+l}{r-l}$$

$$x_n = \frac{2x_p}{r-l} - \frac{r+l}{r-l}$$

$$x_n = \frac{1}{-z_e}\left(\frac{2n}{r-l}x_e + \frac{r+l}{r-l}z_e\right)$$

$$y_n = \alpha y_p + \beta$$

$$\alpha = \frac{1-(-1)}{b-t}$$

$$\beta = -\frac{t+b}{t-b}$$

$$y_n = \frac{2y_p}{t-b} - \frac{t+b}{t-b}$$

$$y_n = \frac{1}{-z_e}\left(\frac{2n}{t-b}y_e + \frac{t+b}{t-b}z_e\right)$$

Projection Matrix

- from

$$x_n = \frac{1}{-z_e}\left(\frac{2n}{r-l}x_e + \frac{r+l}{r-l}z_e\right) \qquad y_n = \frac{1}{-z_e}\left(\frac{2n}{t-b}y_e + \frac{t+b}{t-b}z_e\right)$$

- we get

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

clip coordinates

- with

$$\begin{pmatrix} x_n \\ y_n \\ z_n \\ 1 \end{pmatrix} = \begin{pmatrix} x_c/w_c \\ y_c/w_c \\ z_c/w_c \\ w_c/w_c \end{pmatrix}$$

normalized device coordinates

Mapping of z_e to (-1, 1)

- z_e is mapped from (near, far) or (-n, -f) to (-1, 1)
- the transform does not depend on x_e and y_e
- so, we have to solve for A and B in

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix}$$

$$z_n = \frac{z_c}{w_c} = \frac{Az_e + Bw_e}{-z_e}$$

Mapping of z_e to (-1, 1)

- z_e=-n with w_e=1 is mapped to z_n=-1
- z_e=-f with w_e=1 is mapped to z_f=1

$$\Rightarrow A = -\frac{f+n}{f-n} \quad \Rightarrow B = -\frac{2fn}{f-n}$$

- the complete matrix is

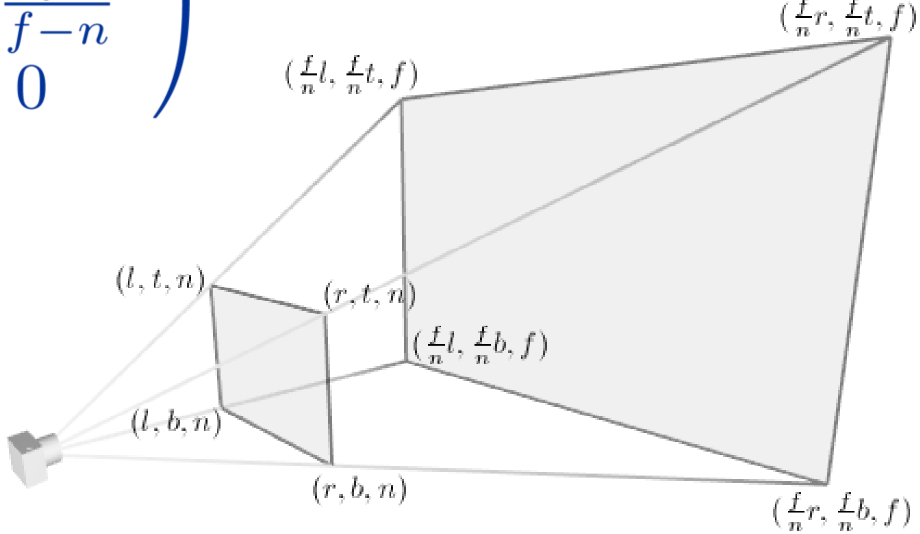
$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Perspective Projection Matrix

- the matrix

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

transforms the view volume, the pyramidal frustum to the canonical view volume



pyramidal frustum

[Song Ho Ahn]

Perspective Projection Matrix

- projection matrix for negated values for n and f (OpenGL)

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

- projection matrix for actual values for n and f

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

UNI FREIBURG

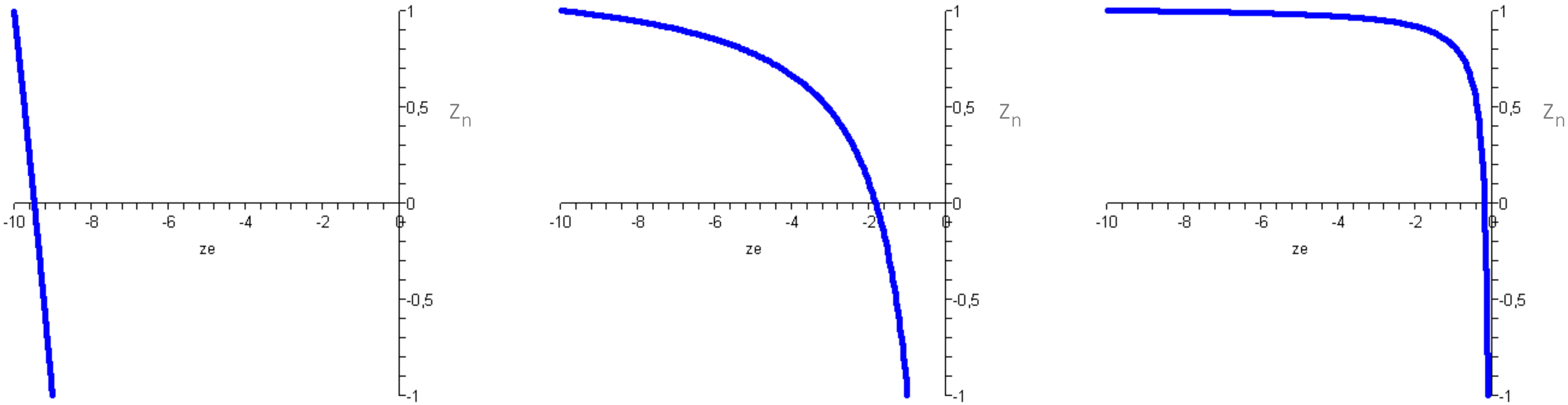
Symmetric Setting

- the matrix simplifies for r = -l and t = -b

$$
\begin{aligned}
r + l &= 0 \\
r - l &= 2r \\
t + b &= 0 \\
t - b &= 2t
\end{aligned}
\quad \Rightarrow \begin{pmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}
$$

Near Plane

- nonlinear mapping of z_e : $z_n = \frac{f+n}{f-n} + \frac{1}{z_e}\frac{2fn}{f-n}$
- varying resolution / accuracy due to fix-point representation of depth values in the depth buffer



$n = 9\ f = 10$ $\quad$ $n = 1\ f = 10$ $\quad$ $n = 0.1\ f = 10$

- do not move the near plane too close to zero

UNI FREIBURG

Far Plane

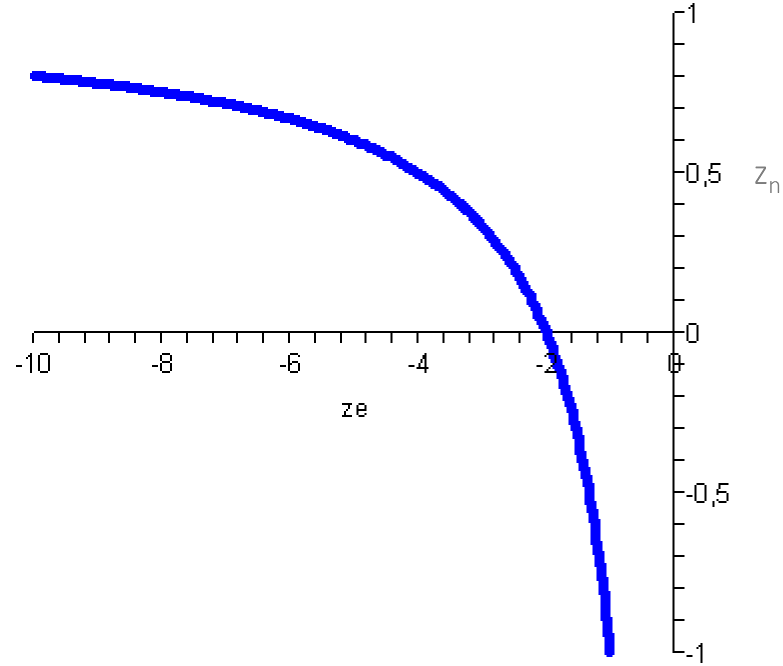
- setting the far plane to infinity is not too critical

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

$$f \rightarrow \infty$$

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

$$z_n = 1 + \frac{2}{z_e}$$



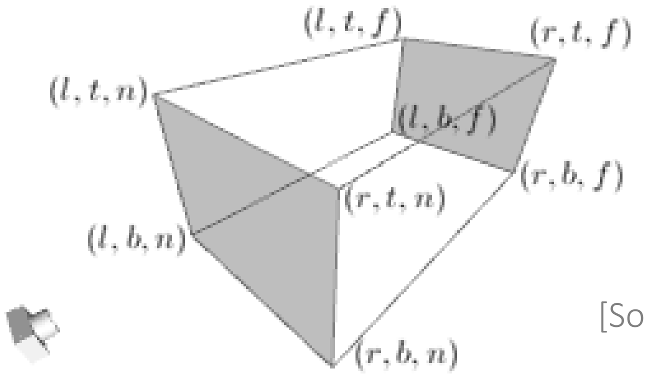
$$n = 1 \quad f = \infty$$

Outline

- 2D projection
- 3D projection
- OpenGL projection matrix
  - perspective projection
  - parallel projection
- OpenGL transformation matrices

Parallel Projection

- the view volume is represented by a cuboid
  - left, right, bottom, top, near, far



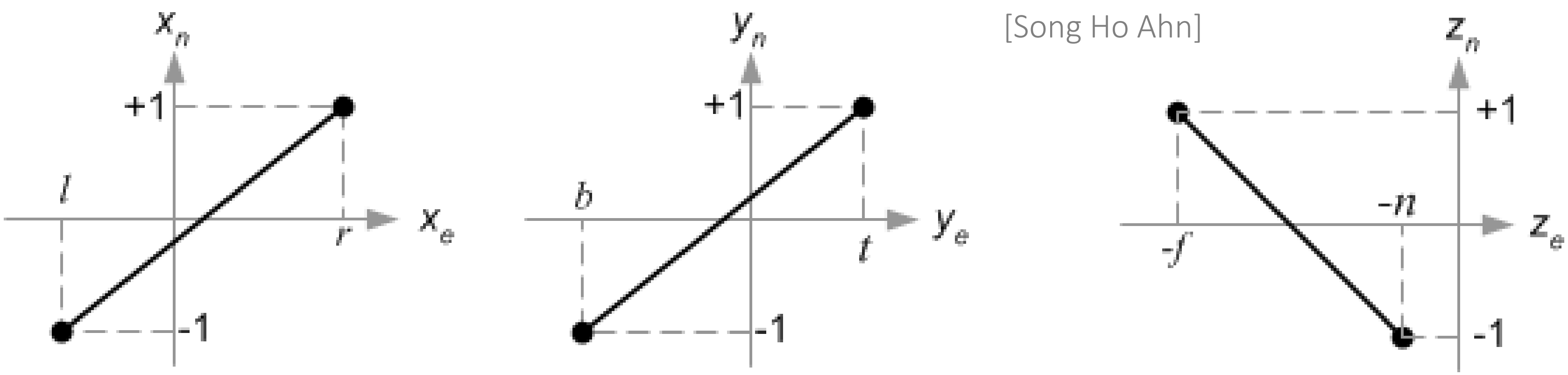
[Song Ho Ahn]

- the projection transform maps the cuboid to the canonical view volume

Mapping of x_e, y_e, z_e to (-1,1)

- all components of a point in eye coordinates are linearly mapped to the range of (-1,1)

[Song Ho Ahn]



$$x_n = \frac{2}{r-l}x_e - \frac{r+l}{r-l} \qquad y_n = \frac{2}{t-b}y_e - \frac{t+b}{t-b} \qquad z_n = -\frac{2}{f-n}z_e - \frac{f+n}{f-n}$$

- linear in x_e, y_e, z_e
- combination of scale and translation

Orthographic Projection Matrix

- general form

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

- simplified form for a symmetric view volume

$$\begin{aligned} r+l &= 0 \\ r-l &= 2r \\ t+b &= 0 \\ t-b &= 2t \end{aligned} \quad \Rightarrow \begin{pmatrix} \frac{1}{r} & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & 0 \\ 0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Outline

- 2D projection
- 3D projection
- OpenGL projection matrix
- OpenGL transformation matrices

OpenGL Matrices

- objects are transformed from object to eye space with the GL_MODELVIEW matrix
  $$\mathbf{GL\_MODELVIEW} \cdot \mathbf{p}$$
- objects are transformed from eye space to clip space with the GL_PROJECTION matrix
  $$\mathbf{GL\_PROJECTION} \cdot \mathbf{GL\_MODELVIEW} \cdot \mathbf{p}$$
- colors are transformed with the color matrix GL_COLOR
- texture coordinates are transformed with the texture matrix GL_TEXTURE

Matrix Stack

- each matrix type has a stack
- the matrix on top of the stack is used

- ```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum(left, right, bottom, top, near, far);
  ```

choose a matrix stack

the top element is replaced with I_4

projection matrix P is generated
the top element on the stack is multiplied with P resulting in $I_4 \cdot P$
  ```

Matrix Stack

- `glMatrixMode(GL_MODELVIEW);` choose a matrix stack
  `glLoadIdentity();` the top element is replaced with I_4

  `glTranslatef(x,y,z);` translation matrix T is generated
  the top element on the stack is multiplied with T resulting in $I_4 \cdot T$

  `glRotatef(alpha,1,0,0);` rotation matrix R is generated
  the top element on the stack is multiplied with R resulting in $I_4 \cdot T \cdot R$

- note that objects are rotated by **R**, followed by the translation **T**

Matrix Stack

- `glMatrixMode(GL_MODELVIEW);` — choose a matrix stack
  `glLoadIdentity();` — the top element is replaced with I_4
  `glTranslatef(x,y,z);` — the top element is $I_4 \cdot T$
  `glRotatef(alpha,1,0,0);` — the top element is $I_4 \cdot T \cdot R$

  `glPush();` — the top element $I_4 \cdot T \cdot R$ is pushed into the stack; the newly generated top element is initialized with $I_4 \cdot T \cdot R$

  `glTranslatef(x,y,z);` — the top element is $I_4 \cdot T \cdot R \cdot T$

  `glPop();` — the top element is replaced by the previously stored element $I_4 \cdot T \cdot R$

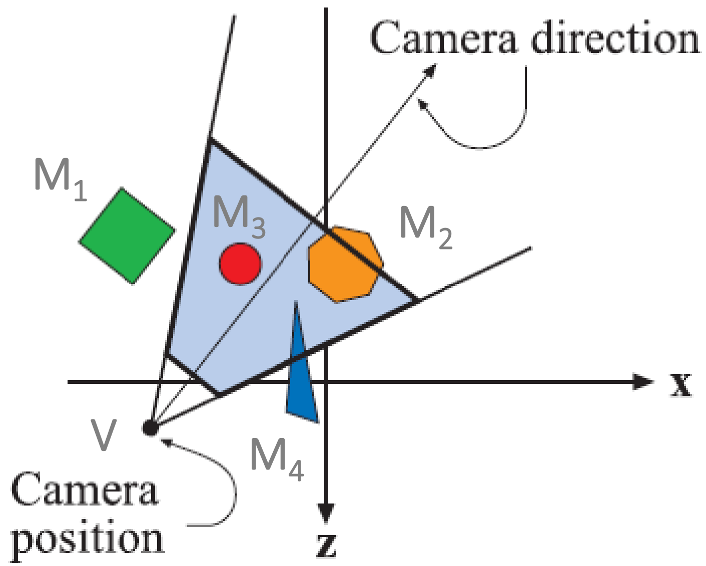
OpenGL Matrix Functions

- `glPushMatrix()`: push the current matrix into the current matrix stack.
- `glPopMatrix()`: pop the current matrix from the current matrix stack.
- `glLoadIdentity()`: set the current matrix to the identity matrix.
- `glLoadMatrix{fd}(`m`)`: replace the current matrix with the matrix *m*.
- `glLoadTransposeMatrix{fd}(`m`)` : replace the current matrix with the row-major ordered matrix *m*.
- `glMultMatrix{fd}(`m`)`: multiply the current matrix by the matrix *m*, and update the result to the current matrix.
- `glMultTransposeMatrix{fd}(`m`)`: multiply the current matrix by the row-major ordered matrix *m*, and update the result to the current matrix.
- `glGetFloatv(GL_MODELVIEW_MATRIX, `m`)`: return 16 values of GL_MODELVIEW matrix to *m*.

- note that OpenGL functions expect column-major matrices in contrast to commonly used row-major matrices

Modelview Example

- objects are transformed with $V^{-1}M$
- $V=T_vR_v$ the camera is oriented and then translated
- $M_{1..4}=T_{1..4}R_{1..4}$ objects are oriented and then translated
- implementation
  - choose the GL_MODELVIEW stack
  - initialize with I_4 — I_4
  - rotate with R_v^{-1} — R_v^{-1}
  - translate with T_v^{-1} — $R_v^{-1} \cdot T_v^{-1} = V^{-1}$
  - push — $R_v^{-1} \cdot T_v^{-1}$
  - translate with T_1 — $R_v^{-1} \cdot T_v^{-1} \cdot T_1$
  - rotate with R_1 — $R_v^{-1} \cdot T_v^{-1} \cdot T_1 \cdot R_1$
  - render object M_1
  - pop — $R_v^{-1} \cdot T_v^{-1}$
  - …



[Akenine-Moeller et al.: Real-time Rendering]

Summary

- 2D projection
- 3D projection
- OpenGL projection matrix
  - perspective projection
  - parallel projection
- OpenGL transformation matrices

UNI FREIBURG

References

- Duncan Marsh: "Applied Geometry for Computer Graphics and CAD", Springer Verlag, Berlin, 2004.
- Song Ho Ahn: "OpenGL", http://www.songho.ca/ .