Image Processing and Computer Graphics

Projections and Transformations in OpenGL

Matthias Teschner

Computer Science Department
University of 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 \( l \), \( p \neq v \)

\[
l = \{ax + by + c = 0\} = (a, b, c)^T
\]

\[
p = (p_x, p_y, 1)^T
\]
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.

$$v = (x, y, 1)^T$$

[Diagram: Perspective projection on the left, parallel projection on the right.]
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
  \[ M = vl^T - (l \cdot v)I_3 \]

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

\[
(l \cdot v)I = (av_x + bv_y + cv_w) \begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

\[
l = \{ax + by + c = 0\} = (a, b, c)^T
\]

![Diagram of projection](image)
Example

- \[ l = \{1x + 0y + 0 = 0\} = (1, 0, 0)^T \]
- \[ \mathbf{p} = (p_x, p_y, 1)^T \]
- \[ \mathbf{p}' = (0, p'_y, 1)^T \]
- \[ \mathbf{v} = (d, 0, 1) \]
- \[ \mathbf{M} = \begin{pmatrix} d \\ 0 \\ 1 \end{pmatrix} (1, 0, 0) - \left( \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} d \\ 0 \\ 1 \end{pmatrix} \right) \]
- \[ \mathbf{I}_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -d & 0 \\ 1 & 0 & -d \end{pmatrix} \]
- 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 $\mathbf{M}$ and $\lambda \mathbf{M}$ 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

\[
\begin{pmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
-\frac{1}{d} & 0 & 1
\end{pmatrix} \begin{pmatrix}
x \\
y \\
w
\end{pmatrix} = \begin{pmatrix}
0 \\
y \\
-\frac{x}{d} + w
\end{pmatrix} \sim \begin{pmatrix}
0 \\
y \\
\frac{y}{w-\frac{x}{d}}
\end{pmatrix}
\]

- $x$ 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

\[ l = \{x + 0y + 0 = 0\} = (1, 0, 0)^T \]

\[ \mathbf{v} = (-1, 0, 0) \quad \mathbf{p'} = (0, p_y, 1)^T \quad \mathbf{p} = (p_x, p_y, 1)^T \]

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

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

\[ 1 = \{1x + 0y + 0 = 0\} = (1, 0, 0)^T \]

\[ \mathbf{p} = (p_x, p_y, 1)^T \]

\[ \mathbf{p}' = (0, p'_y, 1)^T \]

\[ \mathbf{v} = (d, 0, 1) \]

\[ p'_x = 0 \quad \frac{p_y}{p_x - d} = \frac{p'_y}{-d} \Rightarrow p'_y = \frac{-d p_y}{p_x - d} \quad p_w = 1 \Rightarrow p'_w = p_x - d \]

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

- maps \( \mathbf{p} \) to \( p'_x = 0 \)
- maps \( \mathbf{p} \) to \( p'_y = -d p_y \)
- maps \( \mathbf{p} \) with \( p_w = 1 \) to \( p'_w = p_x - d \)


Discussion

- 2D transformation in homogeneous form

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

- \(w_x\) 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 \( \mathbf{v} \) onto \( \mathbf{n} \) maps a point \( \mathbf{p} \) onto \( \mathbf{p}' \)
- \( \mathbf{p}' \) is the intersection of the line through \( \mathbf{p} \) and \( \mathbf{v} \) with plane \( \mathbf{n} \)
- \( \mathbf{v} \) is the viewpoint, center of perspectivity
- \( \mathbf{n} \) is the viewplane
- the line through \( \mathbf{p} \) and \( \mathbf{v} \) is a projector
- \( \mathbf{v} \) is not on the plane \( \mathbf{n} \), \( \mathbf{p} \neq \mathbf{v} \)

\[ \mathbf{n} = \{ax + by + cz + d = 0\} \]
\[ = (a, b, c, d)^T \]

\[ \mathbf{p} = (p_x, p_y, p_z, 1)^T \]
General Case

- a 3D projection is represented by a matrix
  \[ M = vn^T - (n \cdot v)I_4 \]

\[
vn^T = \begin{pmatrix}
v_xa & v_xb & v_xc & v_xd \\
v_ya & v_yb & v_yc & v_yd \\
v_za & v_zb & v_zc & v_zd \\
v_wa & v_wb & v_wc & v_wd \\
\end{pmatrix}
\]

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

\[ n = \{ax + by + cz + d = 0\} = (a, b, c, d)^T \]

\[ p = (p_x, p_y, p_z, 1)^T \]

\[ p' \]

\[ r \]

\[ r' \]

\[ s \]

\[ s' \]
Example

\[ \mathbf{n} = \{ax + by + cz + d = 0\} \]
\[ \mathbf{v} = (d, 0, 0, 1) \]
\[ \mathbf{p} = (p_x, p_y, p_z, 1)^T \]
\[ \mathbf{p}' = (0, p'_y, p'_z, 1)^T \]

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

\[ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \end{pmatrix} \]

E. g. \( d = -1 \), \( (1,2,0)^T \) is mapped to \( (0,1,0)^T \)
Example

- parallel projection onto the plane $z = 0$ with viewpoint / viewing direction $\mathbf{v} = (0, 0, 1, 0)^T$

  $\mathbf{n} = \{0x + 0y + 1z + 0 = 0\}$

  $\mathbf{v} = (0, 0, 1, 0)^T$

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

  $I_4 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$

- $x$- and $y$-component are unchanged, $z$ is mapped to zero

- remember that $\mathbf{M}$ and $\lambda \mathbf{M}$ with, e.g., $\lambda = -1$ 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 \((1,1,1)\).

This transformation implements orthographic projection.

This transformation implements perspective projection.

[Song Ho Ahn]
Open GL 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 Open GL, near and far are negative, so the mapping incorporates a reflection (change of right-handed to left-handed)
    - however, in Open GL 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)

\[
\frac{x_p}{x_e} = \frac{-n}{z_e} \Rightarrow x_p = \frac{n x_e}{-z_e} \\
\frac{y_p}{y_e} = \frac{-n}{z_e} \Rightarrow y_p = \frac{n y_e}{-z_e}
\]

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

[Song Ho Ahn]
Mapping of $x_p$ and $y_p$ to (-1, 1)

$x_n = \alpha x_p + \beta$

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

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

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

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

$y_n = \alpha y_p + \beta$

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

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

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

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

[Song Ho Ahn]
**Projection Matrix**

- from

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

- we get

\[
\begin{pmatrix}
x_c \\
y_c \\
z_c \\
w_c
\end{pmatrix} = \begin{pmatrix}
\frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\
0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\
. & . & . & . \\
0 & 0 & -1 & 0
\end{pmatrix} \begin{pmatrix}
x_e \\
y_e \\
z_e \\
w_e
\end{pmatrix} \quad \text{clip coordinates}
\]

- with

\[
\begin{pmatrix}
x_n \\
y_n \\
z_n \\
1
\end{pmatrix} = \begin{pmatrix}
x_c/w_c \\
y_c/w_c \\
z_c/w_c \\
w_c/w_c
\end{pmatrix} \quad \text{normalized device coordinates}
\]
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

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

\[
\tilde{z}_n = \frac{z_c}{w_c} = \frac{A z_e + B w_e}{-z_e}
\]
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

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

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

[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

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

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

\[
\begin{align*}
  r + l &= 0 \\
  r - l &= 2r \\
  t + b &= 0 \\
  t - b &= 2t
\end{align*}
\]

\[
\begin{pmatrix}
  \frac{n}{r} & 0 & 0 & 0 & 0 \\
  0 & \frac{n}{t} & 0 & 0 & 0 \\
  0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} & 0 \\
  0 & 0 & -1 & 0 & 0
\end{pmatrix}
\]
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

- do not move the near plane too close to zero

$n = 9 \ f = 10$

$n = 1 \ f = 10$

$n = 0.1 \ f = 10$
**Far Plane**

- setting the far plane to infinity is not too critical

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

\[f \to \infty\]

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

\[z_n = 1 + \frac{2}{z_e}\]  
\[n = 1 \quad f = \infty\]
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

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

[Song Ho Ahn]
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)

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

- 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**

\[
\begin{align*}
r + l &= 0 \\
r - l &= 2r \\
t + b &= 0 \\
t - b &= 2t
\end{align*}
\]

\[
\Rightarrow \begin{pmatrix}
\frac{1}{r} & 0 & 0 & 0 \\
0 & \frac{1}{t} & 0 & 0 \\
0 & 0 & -\frac{2}{f-n} & -\frac{f+n}{f-n} \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
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
  \[ \text{GL\_MODELVIEW} \cdot p \]

- objects are transformed from eye space to clip space with the GL_PROJECTION matrix
  \[ \text{GL\_PROJECTION} \cdot \text{GL\_MODELVIEW} \cdot p \]

- 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);` 
  `glLoadIdentity();` 
  choose a matrix stack 
  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);
glLoadIdentity();
glTranslatef(x,y,z);
glRotatef(alpha,1,0,0);

choose a matrix stack
the top element is replaced with $I_4$
the top element is $I_4 \cdot T$
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 $V^{-1}M$
- $V = T_v R_v$
- $M_{1..4} = T_{1..4} R_{1..4}$
- implementation
  - choose the GL_MODELVIEW stack
  - initialize with $I_4$
  - rotate with $R_v^{-1}$
  - translate with $T_v^{-1}$
  - push
  - translate with $T_1$
  - rotate with $R_1$
  - render object $M_1$
  - pop
  - ...
Summary

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

- Song Ho Ahn: "OpenGL", http://www.songho.ca/