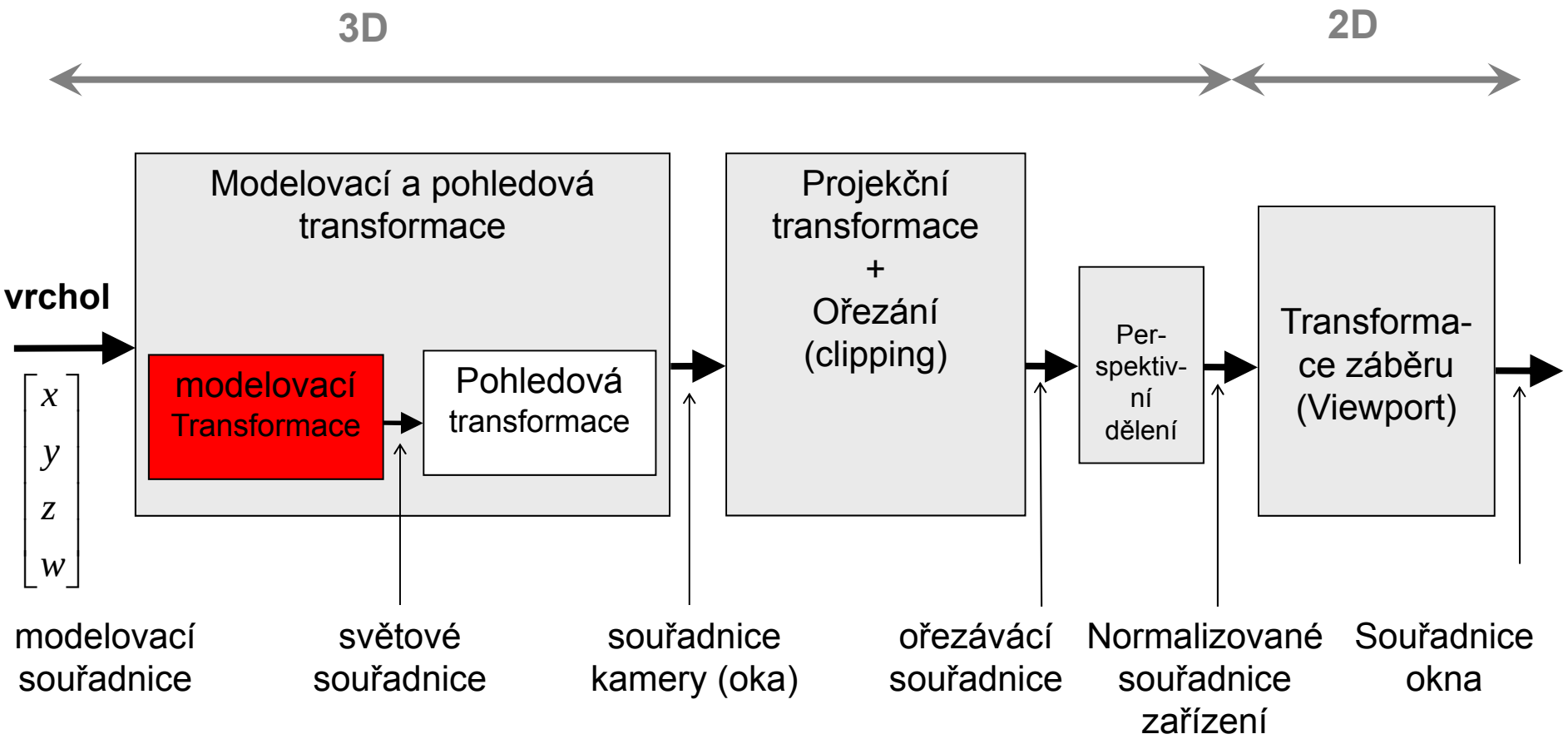
Transformace (v OpenGL) – příklady a knihovna GLM

Petr Felkel, Jaroslav Sloup
Katedra počítačové grafiky a interakce, ČVUT FEL
místnost KN:E-413 (Karlovo náměstí, budova E)
E-mail: felkel@fel.cvut.cz

Poslední změna: 15.3.2016

Modelovací transformace



Mikropřehled transformací a jejich inverzí

- Následuje přehled základních transformačních matic

  +

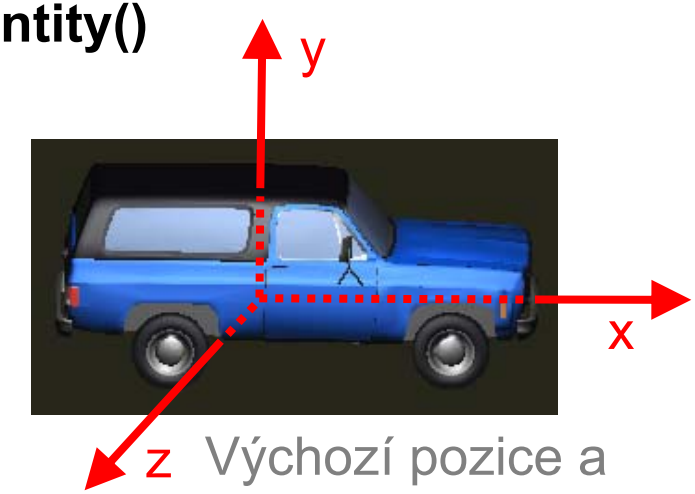
- Způsob jejich vytvoření v knihovně GLM (*OpenGL mathematics*)
  - GLM vychází se syntaxe GLSL
  - Lze ji najít na adrese http://glm.g-truc.net/
  - Knihovna je součástí balíčku PGR-framework.
  - Do verze GLM 0.9.5 používá stupně,
  - Od verze GLM 0.9.6 používá radiany

```
// načtení halviček transformací v knihovně GLM. V PGR frameworku je již uděláno
#include <glm/gtc/matrix_transform.hpp>
```

Identita

Jednotková matice, například funkce M = **Identity()**

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Výchozí pozice a natočení modelu

- Používá se na inicializaci matice v příkazech, které násobí aktuální matici maticí nově definované transformace

```
// vytvoření matice identity pomocí knihovny GLM
glm::mat4 m = glm::mat4(1.0);
```

Matice posunutí (translace)

Matice posunutí (translace) T

- realizuje posunutí dané vektorem $\mathbf{c} = (m_x \quad m_y \quad m_z)^t$
  (resp. posune lokální soustavu souřadnic o stejné hodnoty)

$$T = \begin{bmatrix} 1 & 0 & 0 & \text{mx} \\ 0 & 1 & 0 & \text{my} \\ 0 & 0 & 1 & \text{mz} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{a} \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 & \text{-mx} \\ 0 & 1 & 0 & \text{-my} \\ 0 & 0 & 1 & \text{-mz} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
// posunutí matice o zadaný vektor. Vynásobí matici matrix maticí posunutí o vector
glm::mat4 m = glm::translate( matrix, vector );
```

Posunutí v GLM

```
/* set identity to matrix */
glm::mat4 matrix = glm::mat4(1.0);
passMatrixToVertexShader( matrix );
drawModel();                          /* original model */

 /* add translation to matrix  */
glm::mat4 matrix1 = glm::translate(
  matrix,                             /* matrix to be modified */
  glm::vec3(1.25f, 0.0f, 0.0f));      /* amount of translation */
passMatrixToVertexShader( matrix1 );
drawModel();                          /* ⇒ image 1 */

 /* add another translation to matrix  */
glm::mat4 matrix2 = glm::translate(
  matrix,                             /* matrix to be modified */
  glm::vec3(0.0f, 1.25f, 0.0f));      /* amount of translation */
passMatrixToVertexShader( matrix2 );
drawModel();                          /* ⇒ image 2 */
```

orig. model

image 1

image 2

Matice pro změnu měřítka (škálování)

Matice pro změnu měřítka (škálování)

- Souřadné osy lokální soustavy souřadnic se prodlouží či zkrátí dle **sx**, **sy** a **sz** as nimi se transformuje i přidružený objekt
- Ten se protáhne, smrští, nebo překlopí

$$\mathbf{S} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{a} \quad \mathbf{S}^{-1} = \begin{bmatrix} 1/sx & 0 & 0 & 0 \\ 0 & 1/sy & 0 & 0 \\ 0 & 0 & 1/sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
// matice změny měřítka na jednotlivých osách. Vynásobí matrix škálovací maticí
glm::mat4 m = glm::scale( matrix, vector );
```

Změna měřítka v GLM

```
/* set identity to matrix */
glm::mat4 matrix = glm::mat4(1.0);
passMatrixToVertexShader( matrix );
drawModel();                                /* original model */

glm::mat4 matrix1 = glm::scale(             /* add scale transform */
  matrix,                                   /* matrix to be modified */
  glm::vec3(1.75f, 1.0f, 1.0f));            /* scale coefficients */
passMatrixToVertexShader( matrix1 );
drawModel();                                /* ⇒ image 1 */

glm::mat4 matrix2 = glm::scale(             /* add another scale transf. */
  matrix,                                   /* matrix to be modified */
  glm::vec3(-1.0f, -2.75f, 1.0f));          /* scale coefficients */
passMatrixToVertexShader( matrix2 );
drawModel();                                /* ⇒ image 2 */
```

orig. model

image 1

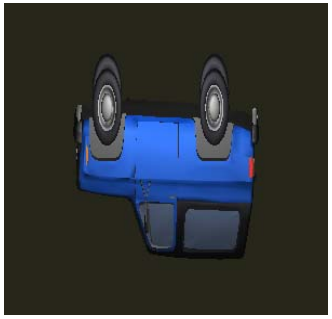
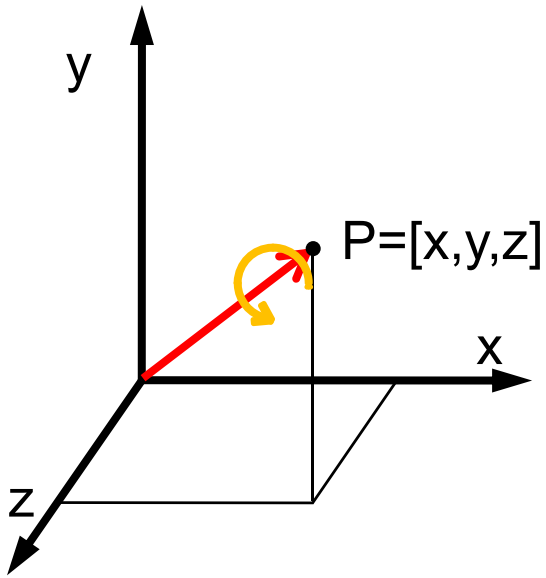


image 2

Rotace kolem obecné osy

- Otočení kolem zadané osy o úhel ve stupních
- Inverzní rotace = rotace okolo stejné osy o opačný úhel $-\theta$



Let $\mathbf{v} = [x, y, z]^T$

$\mathbf{u} = \mathbf{v}/|\mathbf{v}| = [x', y', z']^T$

$$\mathbf{S} = \begin{bmatrix} 0 & -z' & y' \\ z' & 0 & -x' \\ -y' & x' & 0 \end{bmatrix}$$

$$\mathbf{M_R} = \mathbf{u}^T\mathbf{u} + \cos(\text{angle})\,(\mathbf{M_I} - \mathbf{u}^T\mathbf{u}) + \sin(\text{angle})\,\mathbf{S}$$

$$M_R = \begin{bmatrix} m_0 & m_3 & m_6 & 0 \\ m_1 & m_4 & m_7 & 0 \\ m_2 & m_5 & m_8 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

M_I … identity matrix

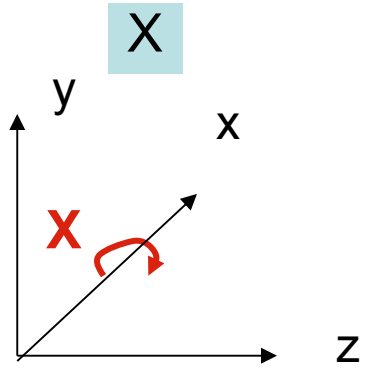
m … coefficients of M_R

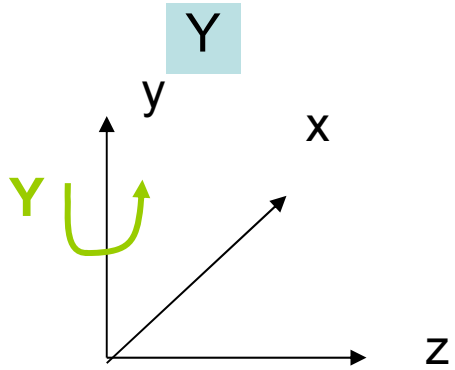
direction of object rotation

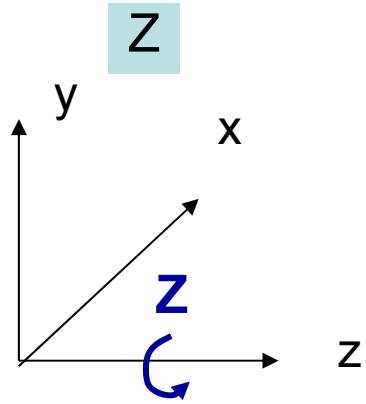
```
// matice rotace o úhel angle kolem  obecné osy axis
// (do GLM 0.9.5 ve stupních / GLM 0.9.6 radianech )
glm::mat4 m = glm::rotate( matrix, angle, axis );
```

Rotace kolem souřadných os

Speciální případy rotací – rotace podle souřadnicových os







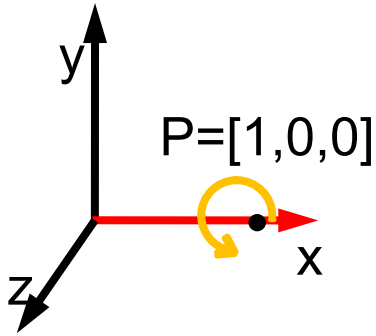
$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Inverzní rotace = rotace okolo stejné osy o opačný úhel $-\theta$

Rotace v GLM

direction of object rotation

orig. model




```
/* set identity to matrix */
glm::mat4 matrix = glm::mat4(1.0);
passMatrixToVertexShader( matrix );
drawModel();                          /* original model */

glm::mat4 matrix1 = glm::rotate( /* add rotation */
  matrix,                             /* matrix to be modified */
  glm::radians(45),                   /* angle in degrees */
  glm::vec3(1.0f, 0.0f, 0.0f));       /* rotation axis */
passMatrixToVertexShader( matrix1 );
drawModel();                          /* ⇒ image 1 */

glm::mat4 matrix2 = glm::rotate( /* add another rotation */
  matrix,                             /* matrix to be modified */
  glm::radians( -45),                 /* angle in degrees */
  glm::vec3(0.0f, 0.0f, 1.0f));       /* rotation axis */
passMatrixToVertexShader( matrix2 );
drawModel();                          /* ⇒ image 2 */
```

image 1

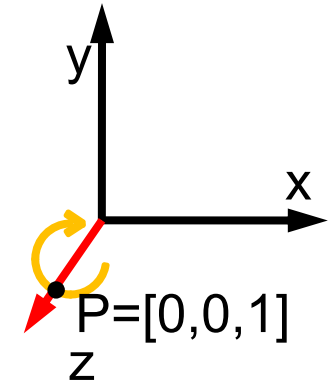


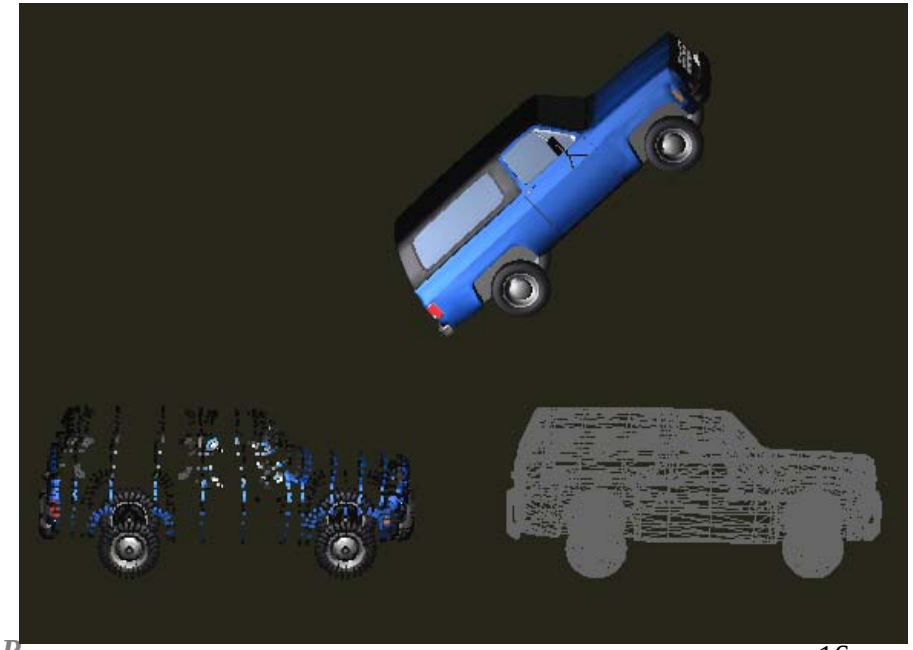
image 2

Rotace není komutativní

Pozor! Záleží na pořadí transformací, neboť
maticové násobení není komutativní!

$\Rightarrow$ tj., R.T není totéž co T.R

R.T.[ ]



T.R.[ ]

Rotace následovaná translací

T.R.[ ]

```
/* set identity to matrix */
glm::mat4 matrix = glm::mat4(1.0);
passMatrixToVertexShader( matrix );
drawModel();                          /* original model */

matrix  = glm::translate(            /* add translation */
  matrix,                            /* matrix to be modified */
  glm::vec3(2.5f, 0.0f, 0.0f));      /* amount of translation */

matrix  = glm::rotate(               /* add rotation */
  matrix,                            /* matrix to be modified */
 glm::radians(45),                   /* angle in degrees */
  glm::vec3(0.0f, 0.0f, 1.0f));      /* rotation axis */
passMatrixToVertexShader( matrix );
drawModel();                          /* ⇒ transformed object */
```

filled … transformed object

dotted … original model

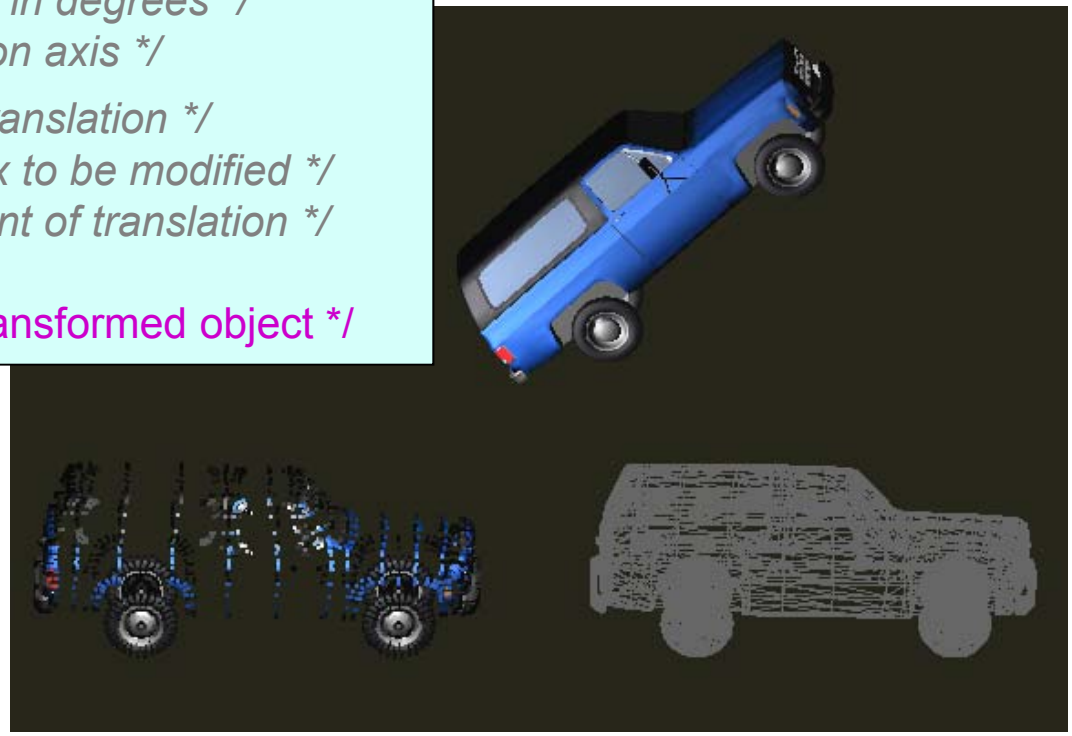
Translace následovaná rotací

R.T.[ ]

```
/* set identity to matrix */
glm::mat4 matrix = glm::mat4(1.0);
passMatrixToVertexShader( matrix );
drawModel();                          /* original model */

matrix  = glm::rotate(                /* add rotation */
  matrix,                             /* matrix to be modified */
  glm::radians(45),                   /* angle in degrees */
  glm::vec3(0.0f, 0.0f, 1.0f));       /* rotation axis */

matrix  = glm::translate(             /* add translation */
  matrix,                             /* matrix to be modified */
  glm::vec3(2.5f, 0.0f, 0.0f));       /* amount of translation */
passMatrixToVertexShader( matrix );
drawModel();                          /* ⇒ transformed object */
```

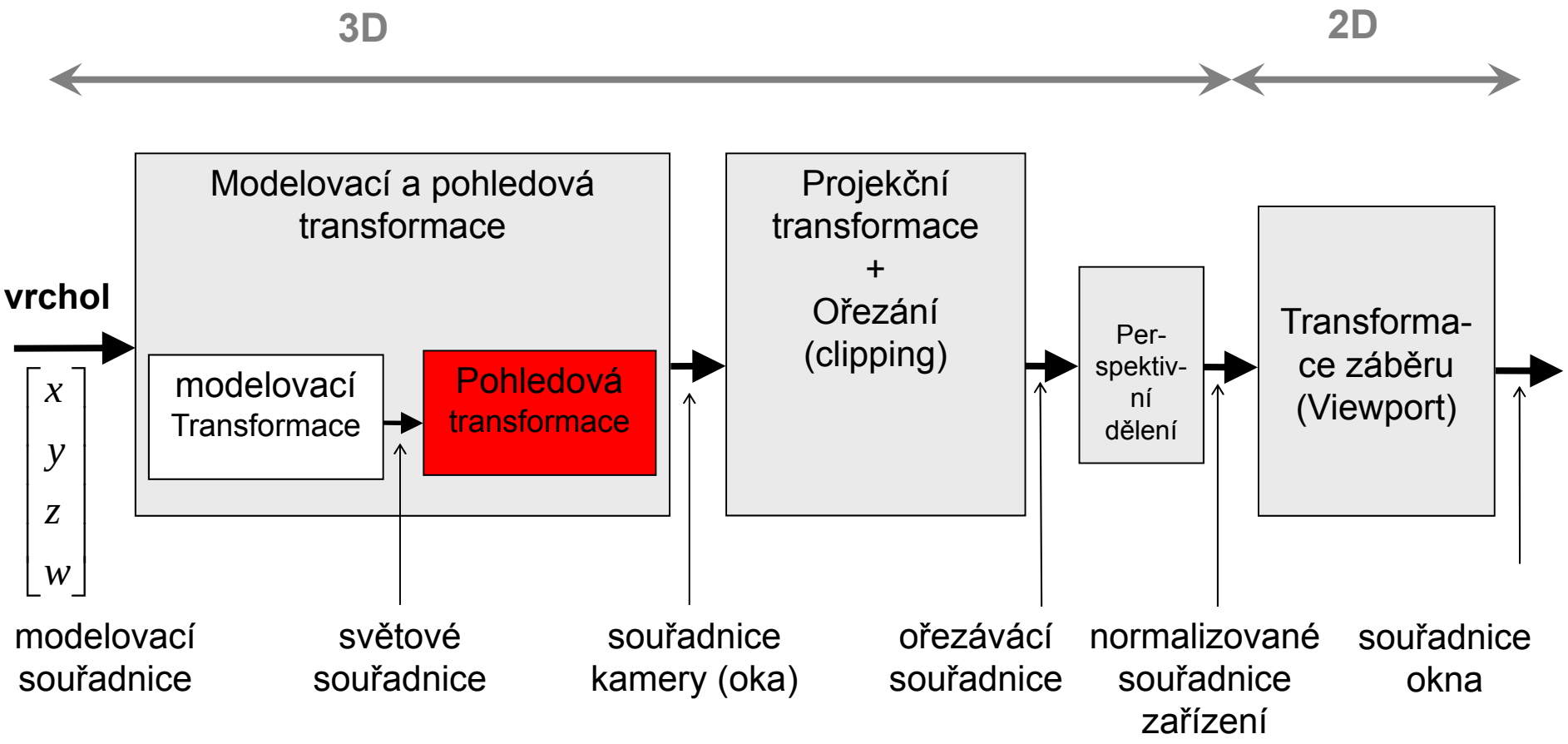
filled … transformed object

dotted … original model



Kód je skoro stejný, jako v předchozím příkladu
Změnilo se pořadí T a R

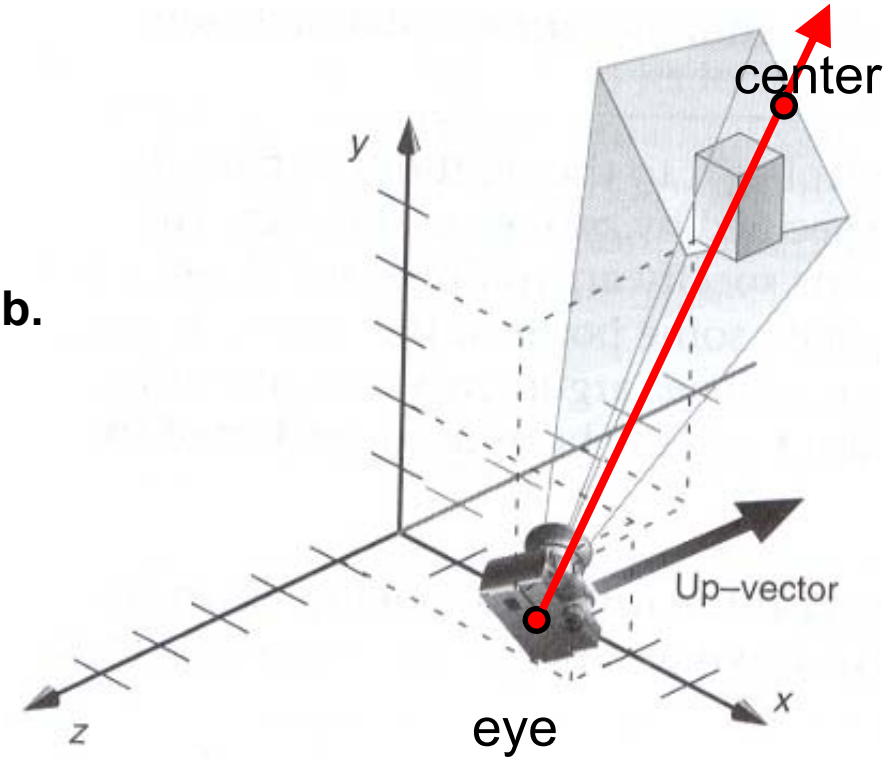
Pohledová transformace



Pohledová transformace - LookAt

```
glm::mat4 glm::lookAt(
        glm::vec3 const & eye,          /* camera position  */
        glm::vec3 const & center,       /* point on a line of sight  */
        glm::vec3 const & up );         /* camera up direction  */
```

- lookAt(…) vytvoří pohledovou matici
- Kameru umístí do
  eye = [eye_X, eye_Y, eye_Z]
- **center = [$center_X$, $center_Y$, $center_Z$] lib.** bod ve směru pohledu
- **up = (up_X, up_Y, up_Z)** směr nahoru – natočení kamery



Pohledová transformace - LookAt

```
glm::mat4 matrix = glm::lookAt(…);
passMatrixToVertexShader( matrix );
drawModel();
```

```
glm::mat4 matrix =
    glm::lookAt(
        glm::vec3(1.0f, 0.0f, 1.0f),
        glm::vec3(0.0f, 0.0f, 0.0f),
        glm::vec3(0.0f, 1.0f, 0.0f)
    );
```

```
glm::mat4 matrix =
    glm::lookAt(
        glm::vec3( 1.0f, 1.0f, 0.0f),
        glm::vec3( 0.0f, 0.0f, 0.0f),
        glm::vec3(-1.0f, 0.0f, 0.0f)
    );
```

```
glm::mat4 matrix =
    glm::lookAt(
        glm::vec3(-1.0f, -0.5f, 1.0f),
        glm::vec3( 0.0f, 0.0f, 0.0f),
        glm::vec3( 0.0f, 1.0f, 0.0f)
    );
```

Transformace v OpenGL

Lineární složka:
rotace (zkosení)
rotation (shear)

posunutí
translation

změna měřítka
scale

$$v' = [x', y', z', w']^T = M.v = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

transformovaný vrchol

projekce
projection
matice pak není afinní

(příští týden)

vrchol před transformací

vertex to be transformed

[Sloup]

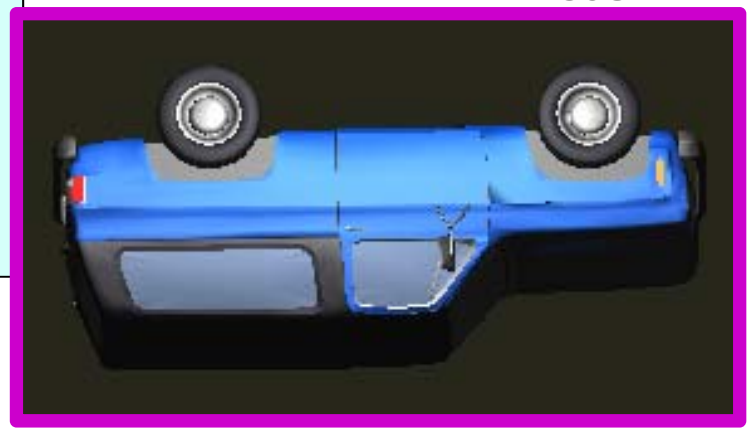
Vytvoření libovolné matice

- Matici můžeme inicializovat polem 16 hodnot (m_0, m_1, ... , m_{15})
- POZOR, z pole se převezmou po sloupcích

```
#include <glm/gtc/type_ptr.hpp>
 /* symmetry with respect to xz-plane */
float initValues[16] = {
            1.0,  0.0, 0.0, 0.0,    // first column
            0.0, -1.0, 0.0, 0.0,    // second column
            0.0,  0.0, 1.0, 0.0,
            0.0,  0.0, 0.0, 1.0 };
glm::mat4 matrix =
            glm::make_mat4(initValues);

passMatrixToVertexShader( matrix );
drawModel();
```

orig. model

transformed model



Obecná transformace pro zobrazení

- Cílem je složit ze tří matic správnou matici:

  M = projection * view * model

- Matici předáme do „Vertex Shaderu“ (uložena po sloupcích)
- „Vertex shader“ provede násobení každého vrcholu

$$v' = [x', y', z', w']^T = M.v = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

souřadnice k zobrazení na obrazové rovině

souřadnice v objektovém prostoru

Skládání transformací = násobení matic

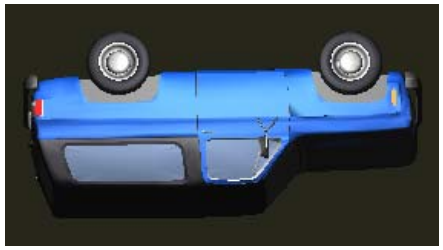
Matice se skládají násobením - přetížený operátor * vynásobí dvě matice M1 a M2 a výsledek uloží do nové matice **matrix**

```
#include <glm/gtc/type_ptr.hpp>
 /* symmetry with respect to xz-plane */
float initValues[16] = {
                1.0,  0.0, 0.0, 0.0,
                0.0, -1.0, 0.0, 0.0,
                0.0,  0.0, 1.0, 0.0,
                0.0,  0.0, 0.0, 1.0 };
glm::mat4 M1 =       glm::make_mat4(initValues);

/* symmetry - yz-plane */
glm::mat4 M2 = glm::mat4(
                glm::vec4(-1.0, 0.0, 0.0, 0.0),
                glm::vec4(0.0, 1.0, 0.0, 0.0,),
                glm::vec4(0.0, 0.0, 1.0, 0.0),
                glm::vec4(0.0, 0.0, 0.0, 1.0) );

glm::mat4  matrix = M1 * M2;
passMatrixToVertexShader( matrix );
drawModel();
```

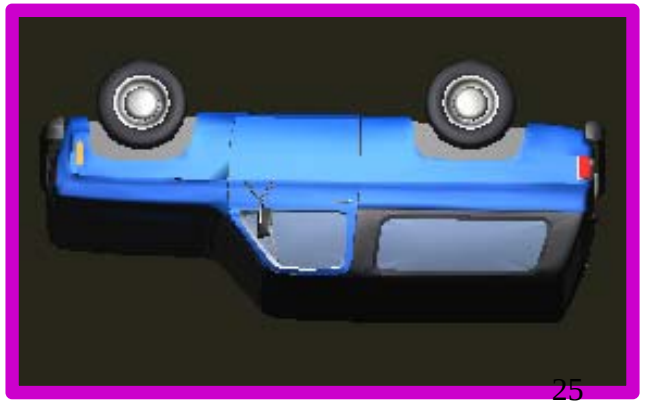
PGR

orig. model

after M1



transformed model



Předání matice do „Vertex Shader“

- Ukázka pro předání všech tří matic a násobení vektoru maticemi přímo na grafické kartě:

```
#version 330

in vec4 position;

uniform mat4 projection ;   // P
uniform mat4 view ;         // E-1, V
uniform mat4 model;         // O, M

void main() {
   vec4 worldPos = model * position;
   vec4 cameraPos = view * worldPos;
   gl_Position = projection * cameraPos;
}
```

Na straně CPU je se matice nahraje na GPU pomocí následujícího kódu

```
projectionMatrixLoc = glGetUniformLocation(theProgram,
   "projection");
float fFrustumScale = 1.0f;
float fzNear = 0.5f; float fzFar = 3.0f;
float matrix[16];
memset(matrix, 0, sizeof(float) * 16);
   matrix[0] = fFrustumScale;
   matrix[5] = fFrustumScale;
   matrix[10] = (fzFar + fzNear) / (fzNear - fzFar);
   matrix[14] = (2 * fzFar * fzNear) / (fzNear - fzFar);
   matrix[11] = -1.0f;
glUseProgram(theProgram);
glUniformMatrix4fv(projectionMatrixLoc , 1, GL_FALSE,
   matrix);
glUseProgram(0);
```