Základy počítačové grafiky
Přednáška 4

Martin Němec

VŠB-TU Ostrava

2024

Opakování

Aktuálně:

- Základní vykreslování objektů v OpenGL (VAO+VBO).
- Kompilace a vytváření jednoduchým shaderů.
- Transformace modelů ve 3D.
- Měli byste mít funkční základní vykreslování.
- Zdrojový kód by měl být vhodně objektově upraven.

Opakování

- Ptejte se prosím přímo na cvičeních, od toho je máme.
- Na cvičení se snažte danou část zprovoznit (pro rotaci tělesa stačilo zkopírovat 5 řádků kódu).
- Pokud se to "rozjelo" následuje rozšiřování, refactoring, testování, pochopení teorie, kódu atd.
- Nepřehledný kód znepříjemňuje další práci.
- Pokud lze něco kontrolovat, udělejte to (kompilace shaderu, existence proměnné v shaderu atd.).

Refactoring is the process of restructuring code, while not changing its original functionality.

glGetUniformLocation na https://khronos.org/

glGetUniformLocation Returns the location of a uniform variable
GLint glGetUniformLocation(GLuint program, GLchar *name);

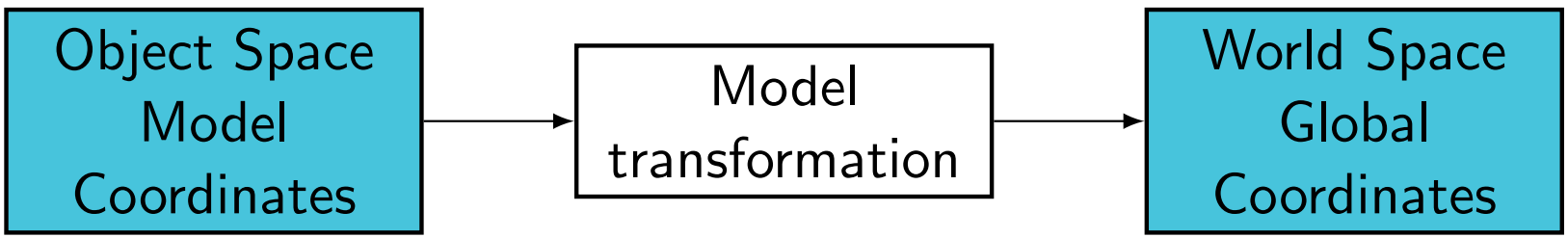
program - Specifies the program object to be queried.
name - Points to a null terminated string containing the name of the uniform variable whose location is to be queried.
Description glGetUniformLocation returns an integer that represents the location of a specific uniform ...
Errors

- GL_INVALID_VALUE is generated if program is not a value generated by OpenGL.
- GL_INVALID_OPERATION is generated if program is not a program object.
- GL_INVALID_OPERATION is generated if program has not been successfully linked.

Associated Gets
glGetActiveUniform, glGetProgram, glGetUniform, glIsProgram

Opakování

- Převod z lokálního do globálního souřadného systému.
- Uniformní proměnná mat4 modelMatrix v GLSL.
- Skládání matic (komutativita).



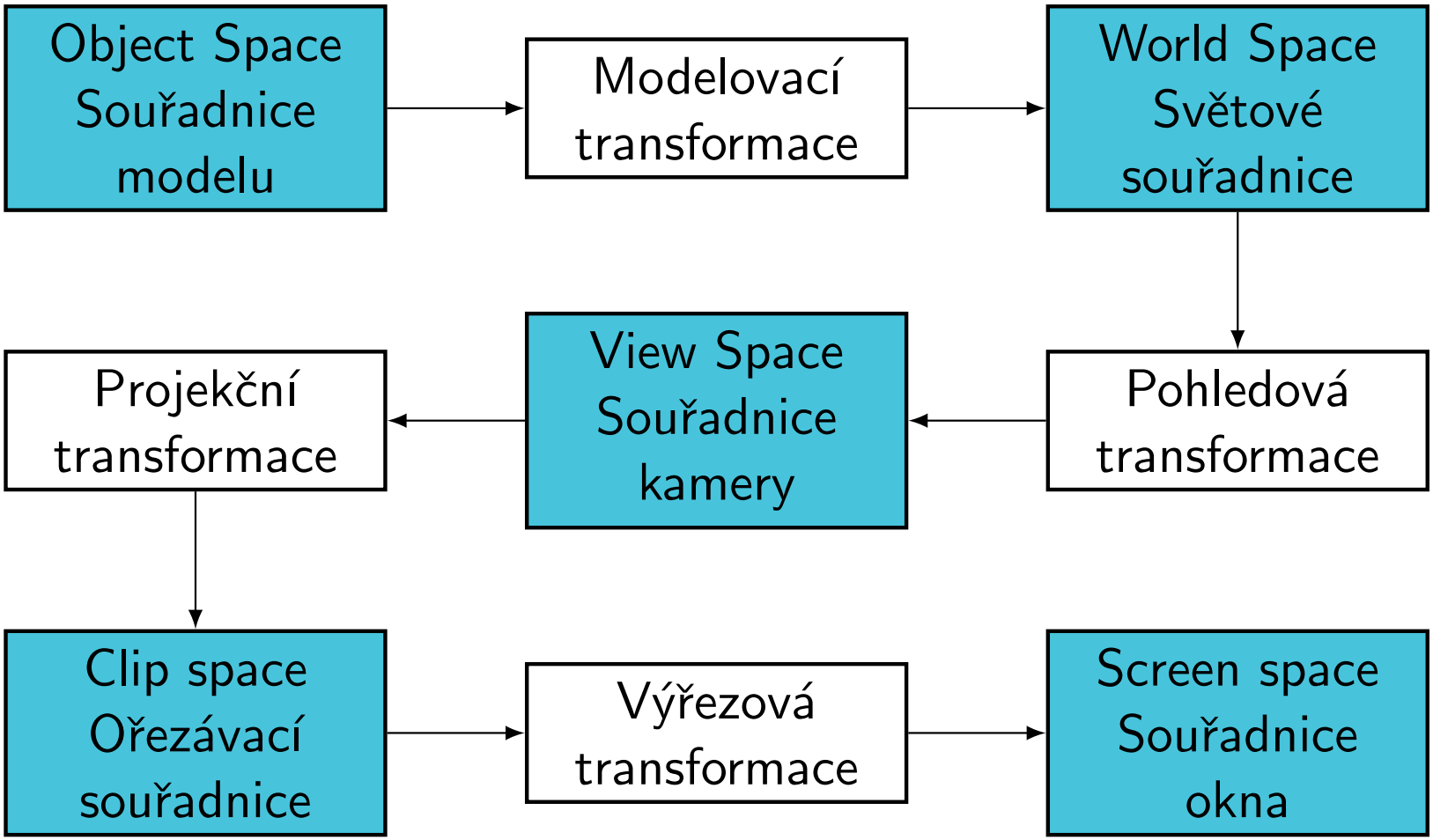
V čem je rozdíl (každý frame rotace o úhel alfa)?

```
alfa+=0.5;
M = rotate(mat4(1.0f),alfa,vec3(0.0f, 1.0f, 0.0f));
```

vs.

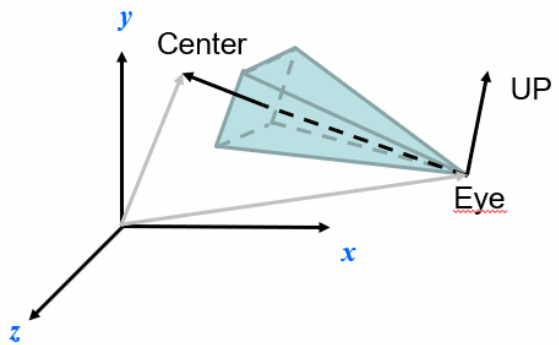
```
M = rotate(M, 0.5f, vec3(0.0f, 1.0f, 0.0f));
```

Aktuálně


Object Space
Souřadnice
modelu
Modelovací
transformace
World Space
Světové
souřadnice
Projekční
transformace
View Space
Souřadnice
kamery
Pohledová
transformace
Clip space
Ořezávací
souřadnice
Výřezová
transformace
Screen space
Souřadnice
okna

Kamera

- Umíme transformovat objekty ve scéně, potřebovali bychom něco, co bude umožňovat zadat odkud a kam se díváme.
- V základní poloze je kamera v počátku, otočená směrem k ose $-z$, horní část je dána směrem osy y (ověřte).
- LookAt umožňuje nastavit pohled kamery libovolně.
- Chceme se procházet a rozhlížet ve scéně.
- Kamera bude definována svou pozici, směrem kam se kouká a ještě vektorem udávající směr kamery nahoru.



Kamera

Můžeme použít funkci **lookAt** z knihovny GLM.

- Eye – definuje bod ve kterém bude kamera umístěna.
- Center – definuje bod, kterým se kamera dívá.
- Up – definuje kterým směrem bude natočen 3D svět v kameře.

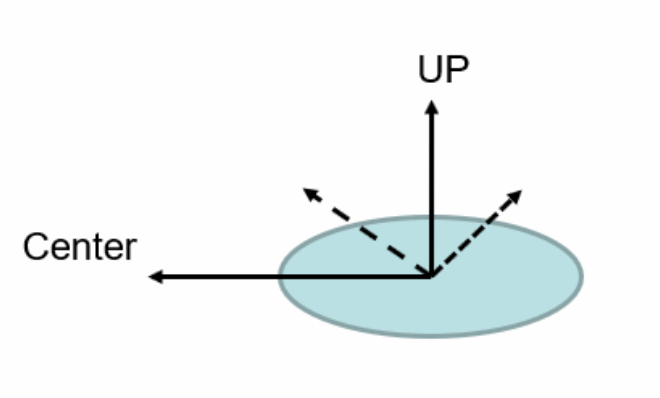
glm::mat4 camera=glm::lookAt(eye, center, up);

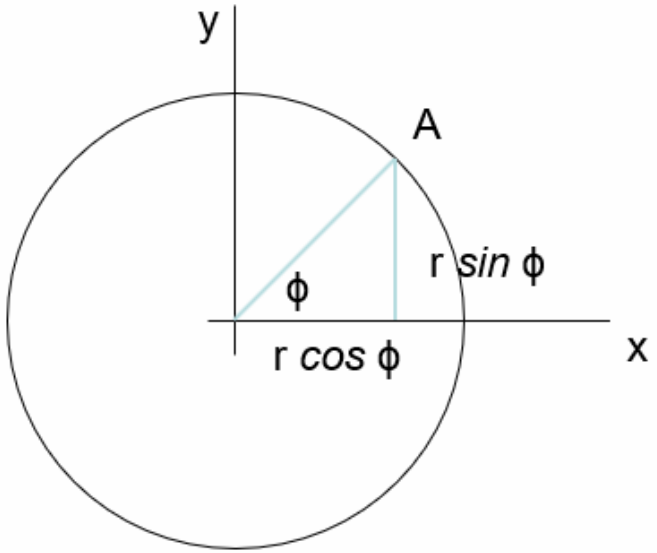
$$
\begin{aligned}
v &= |(center - eye)| \\
r &= |(v \times up)| \\
u &= |(r \times v)|
\end{aligned}
$$

$$
L = A \cdot B\,, A = \begin{bmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ -v_x & -v_y & -v_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, B = \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Polární souřadnice vs. kartézské

Jak určit bod na který se kamera dívá?



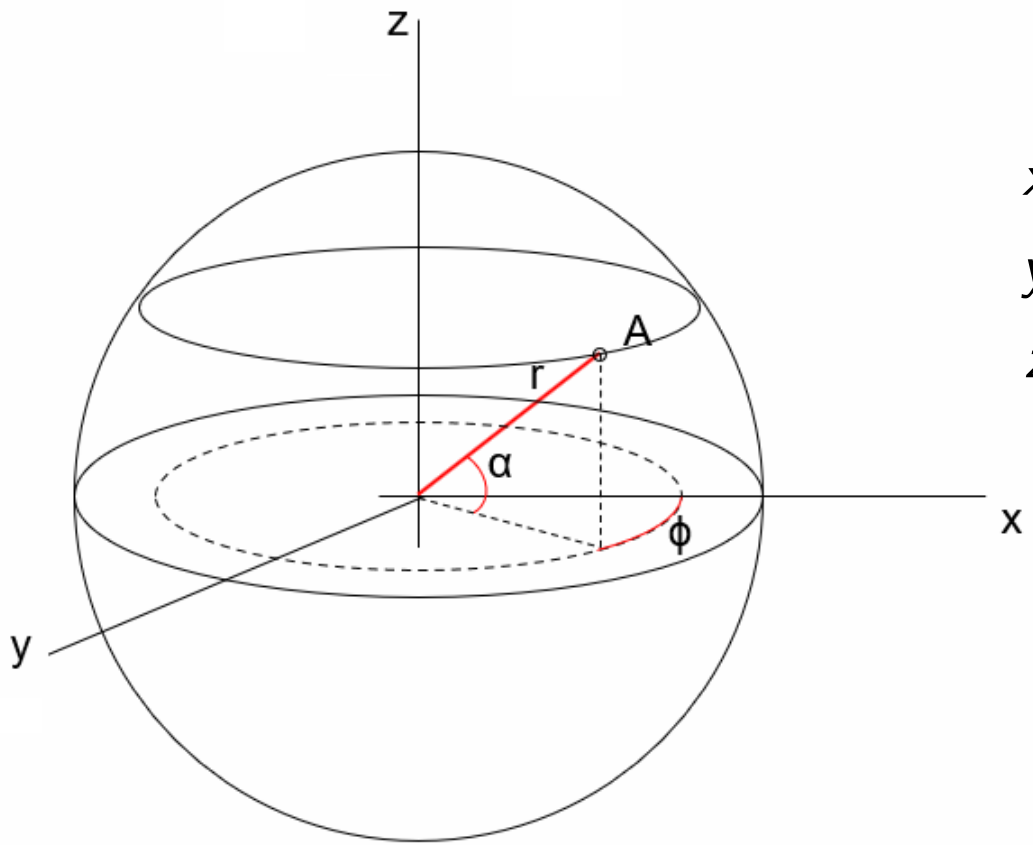


$$x = r \cdot \cos\phi$$

$$y = r \cdot \sin\phi$$

Sférický souřadný systém

Jak určit bod na který se kamera dívá ve 3D?



$$x = r \cdot \sin\alpha \cdot \cos\phi$$

$$y = r \cdot \sin\alpha \cdot \sin\phi$$

$$z = r \cdot \cos\alpha$$

Základní pohyb kamery

Průchod scénou pomocí kláves (např. WSAD nebo šipky).

```
camera->toFront();
//eye+=(glm::normalize(target));

void Camera::toLeft(){
    eye+=(glm::normalize(glm::cross(target,up)));
}
```

Promítání

Promítání je převod z 3D do 2D.

- V geometrii nejprve volíme promítací metodu a potom v této zobrazujeme objekty. V počítačové grafice je to naopak.

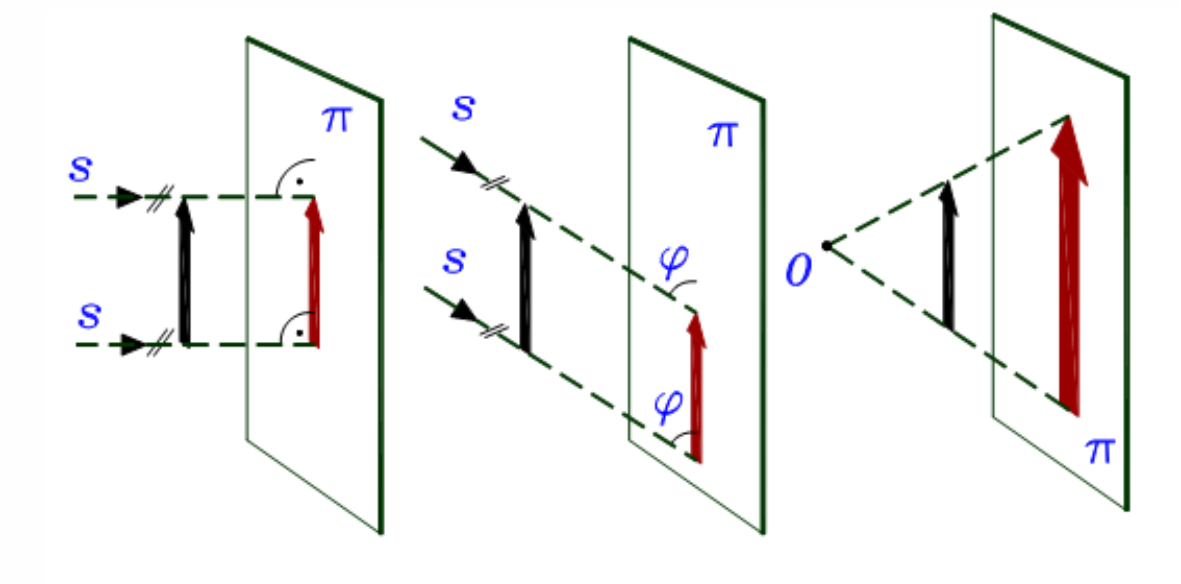
Čím je promítání definováno ?

- Promítací paprsky - polopřímka vycházející z promítacího bodu, směr závisí na typu promítání.
- Průmětna (viewing plane) - plocha v prostoru, na kterou dopadají promítací paprsky (paprsky vytvářejí průmět). Průmětnou nemusí být pouze rovina (polokoule, NURBS plocha apod.)

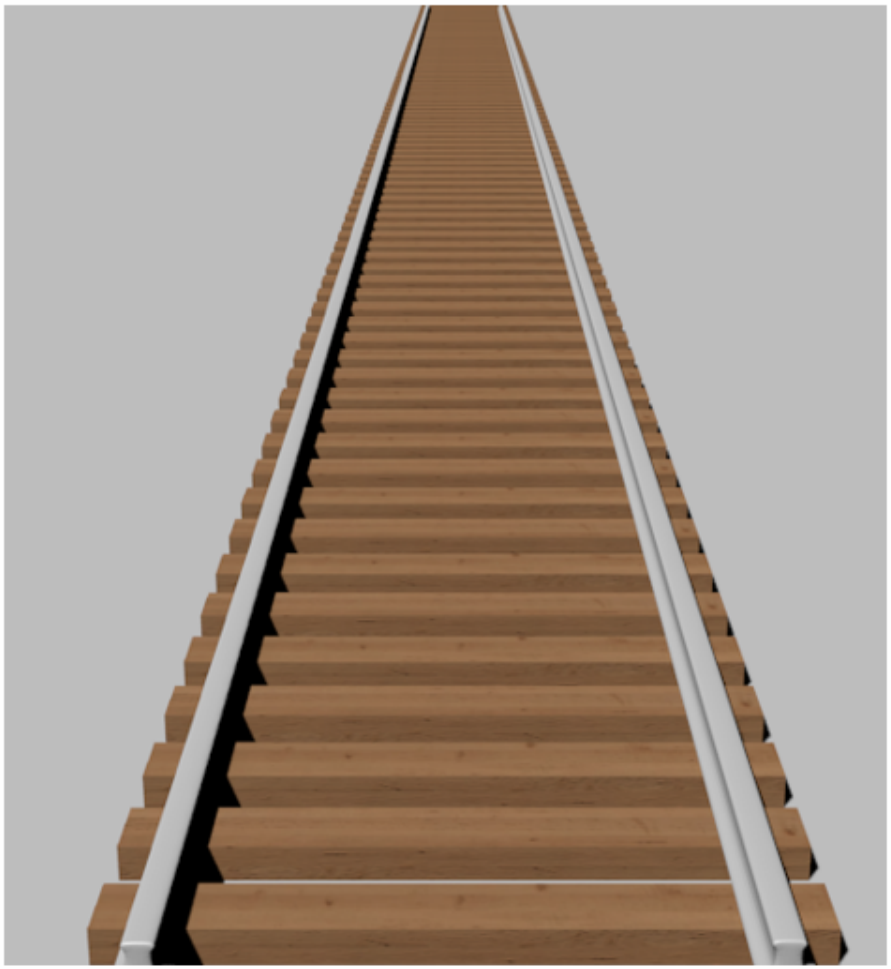
Klasifikace promítacích metod

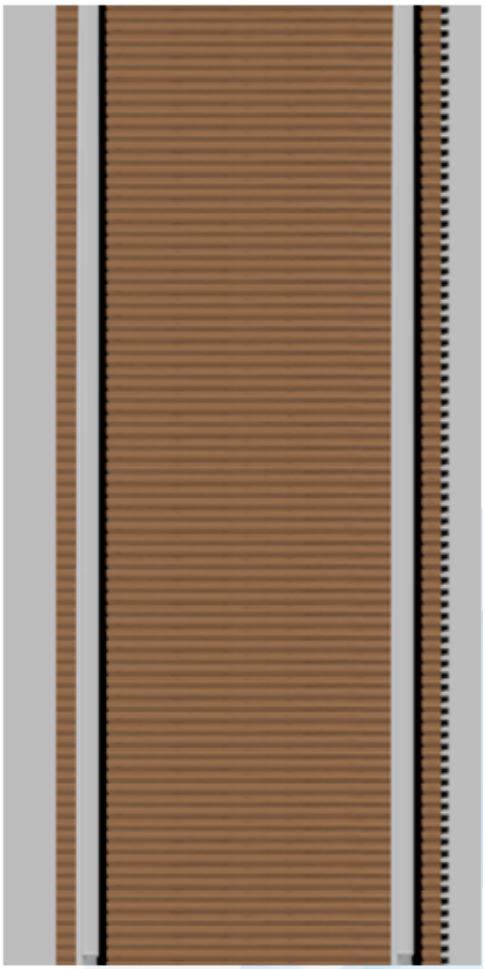
V geometrii (deskriptivní geometri) několik typů promítání. V PG rozlišujeme dva základní typy:

- rovnoběžné (paralelní, orthographic projections) - (pravoúhlé, kosoúhlé)
- středové (perspektivní, perspective projections)

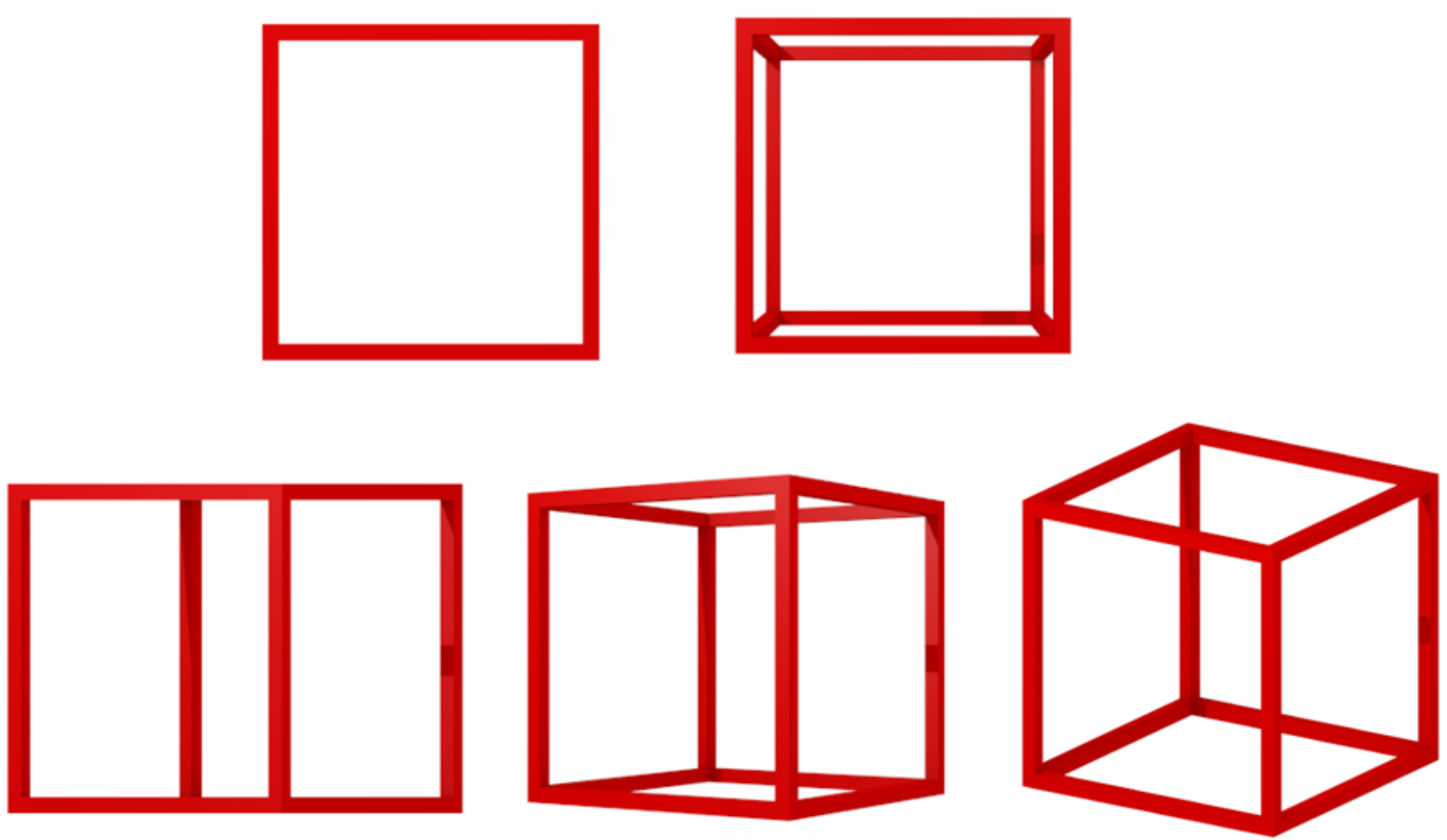


Projekce 3D do 2D



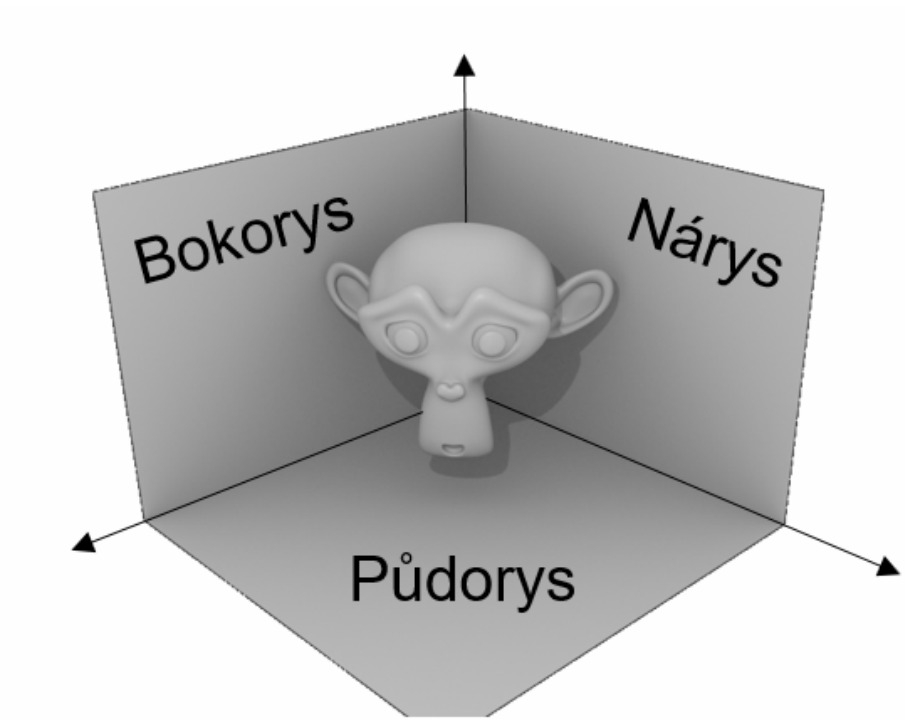


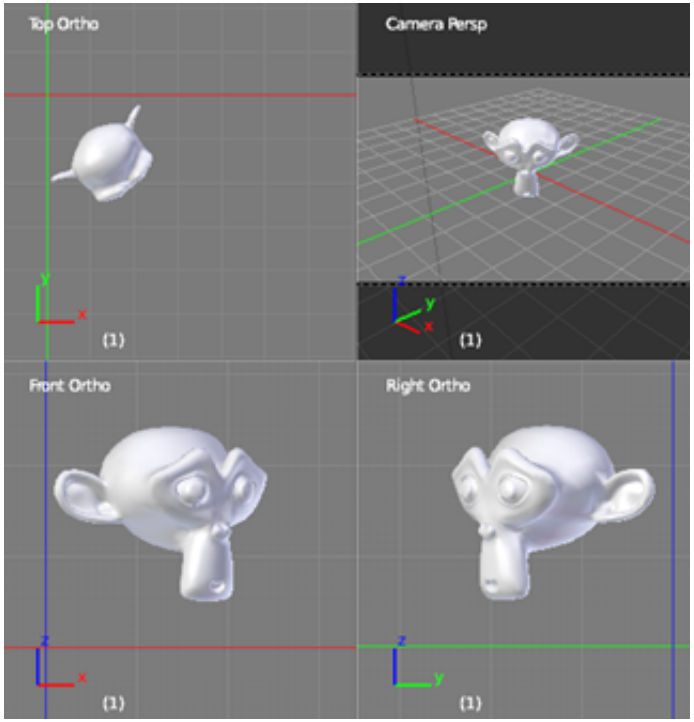
Promítání?



Promítání?

Rovnoběžné pravoúhlé promítání (nárys, půdorys a bokorys).

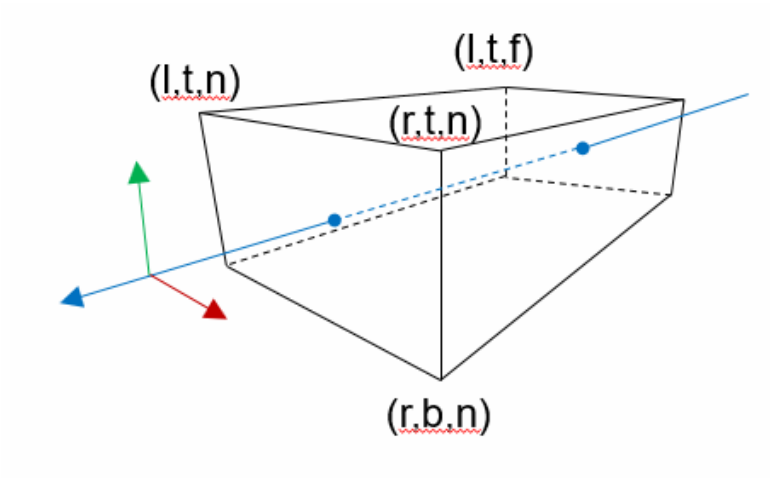


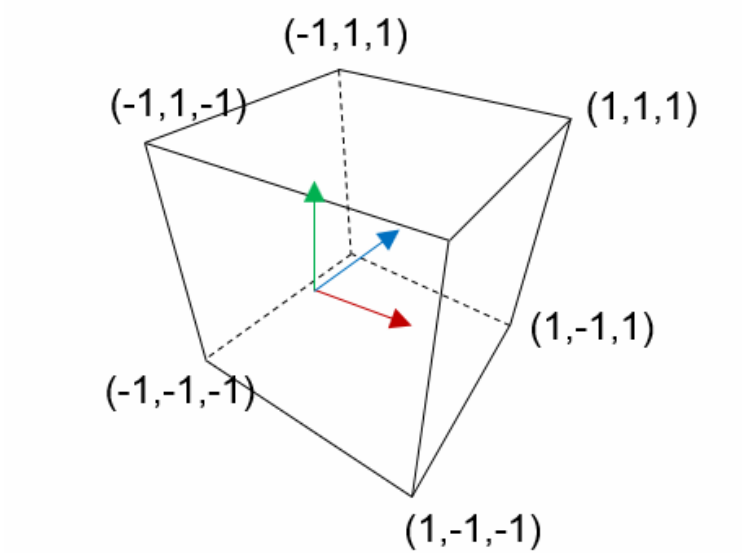


Ortogonální promítání

Scéna je reprezentována jako pravoúhlý hranol.

glm::ortho(left, right, bottom, top, zNear, zFar);
glm::mat4 projMat = glm::ortho(0, 400, 0, 400, -1, 400);



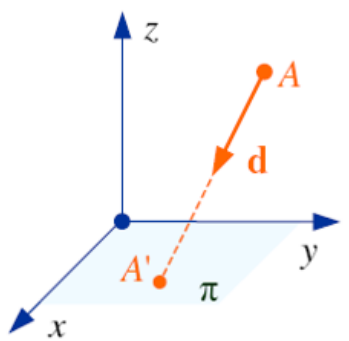


Ortogonální promítání

Matice popisuje rovnoběžné promítání na rovinu $z = 0$ (tedy na rovinu xy). Směr promítacího paprsku je $\vec{d} = (d_x, d_y, d_z)$

$$A = \begin{bmatrix} 1 & 0 & -\frac{d_x}{d_z} & 0 \\ 0 & 1 & -\frac{d_y}{d_z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

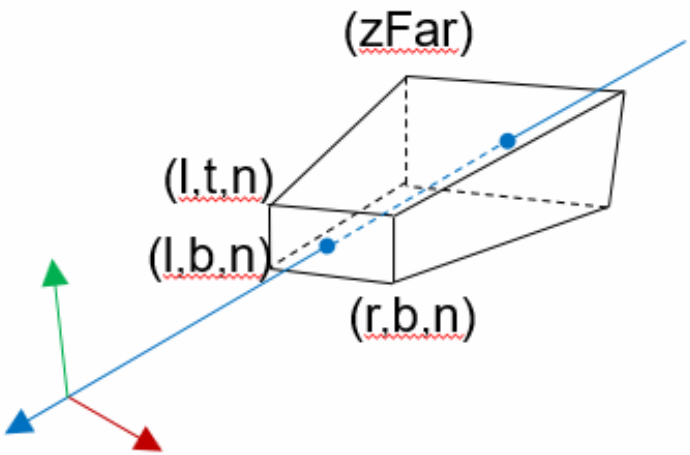
Rovnoběžným přímkám odpovídají opět rovnoběžné přímky, které však nemusí být rovnoběžné s původními přímkami.

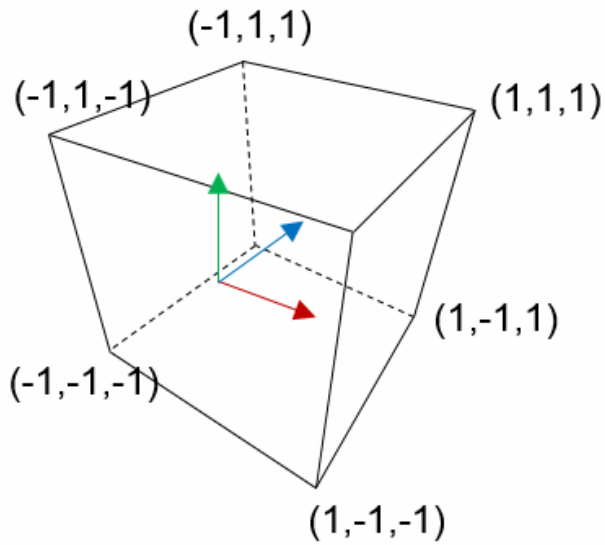


Perspektivní promítání

Scéna je definována jako komolý jehlan.

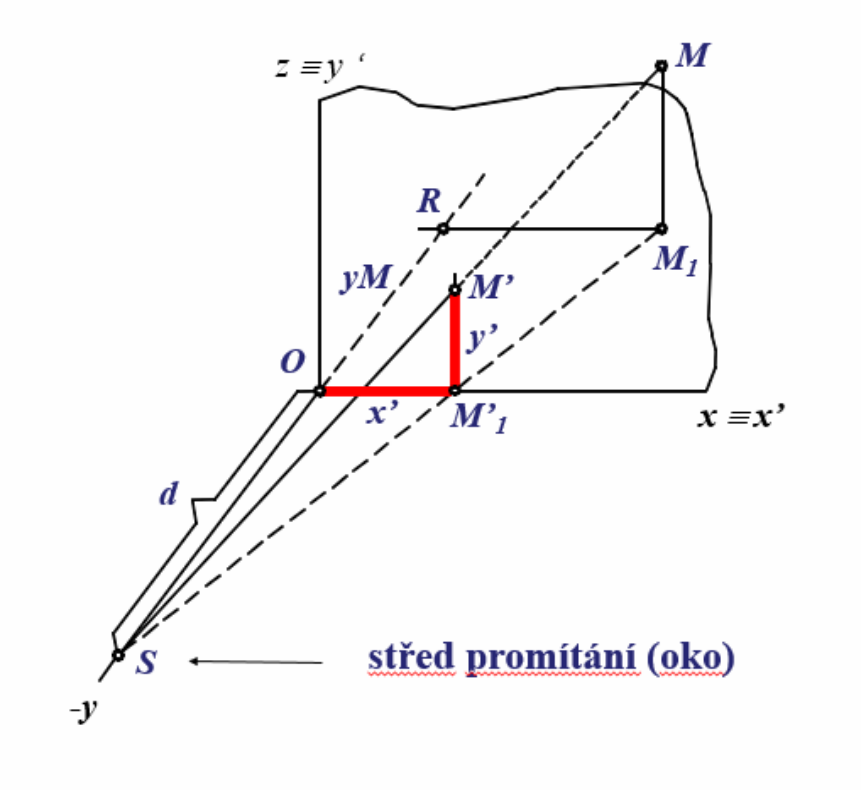
glm::frustum(left, right, bottom, top, zNear, zFar);
glm::perspective(fovy, aspect, zNear, zFar);
projection=perspective(45.0f, 4.0f/3.0f, 0.1f, 100.0f);
//radians or degrees





Perspektivní promítání

Vycházíme z podobnosti trojúhelníků.



$$\frac{x'}{x} = \frac{d}{d+y}$$
$$\frac{y'}{z} = \frac{d}{d+y}$$

$$x' = k \cdot x$$
$$y' = k \cdot z$$
$$k = \frac{d}{d+y}$$

Perspektivní promítání

Matice popisuje projekci ze středu o souřadnicích $(0, 0, f)$ na rovinu $z = 0$ (tedy na rovinu xy).

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{f} & 1 \end{bmatrix}$$

glViewport

void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);

- Část okna, ve které probíhá vykreslování.
- Transformace z NDC (normalizované souřadnice, $(-1, 1)$), na rozsah souřadnic obrazovky.
- Co vše se musí změnit pokud změníme velikost obrazovky?

Křížový odkaz

```
//---- Camera.h ---
// dopredna deklarace
class Shader;
class Camera{
  private:
   Shader* sh;
  public:
   Camera(Shader* s);
   void metoda();
};
```

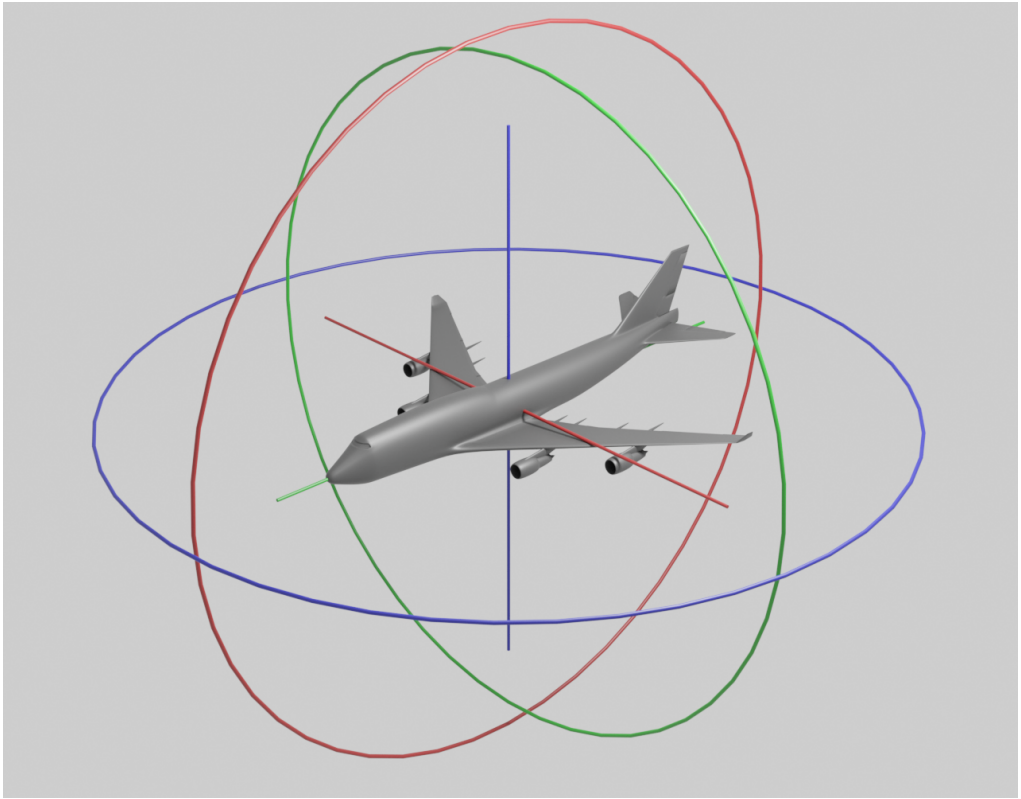
```
//--- Camera.cpp --
#include "Camera.h"
#include "Shader.h"
Camera::Camera(){
    ...
}
```

```
//------- Shader.h ------
// dopredna deklarace
class Camera;
class Shader{
  private:
    Camera* cam;
  public:
    Shader(Camera* c);
    void metoda();
};
```

```
//--- Shader.cpp ---
#include "Shader.h"
#include "Camera.h"
Shader::Shader(){
    ...
}
```

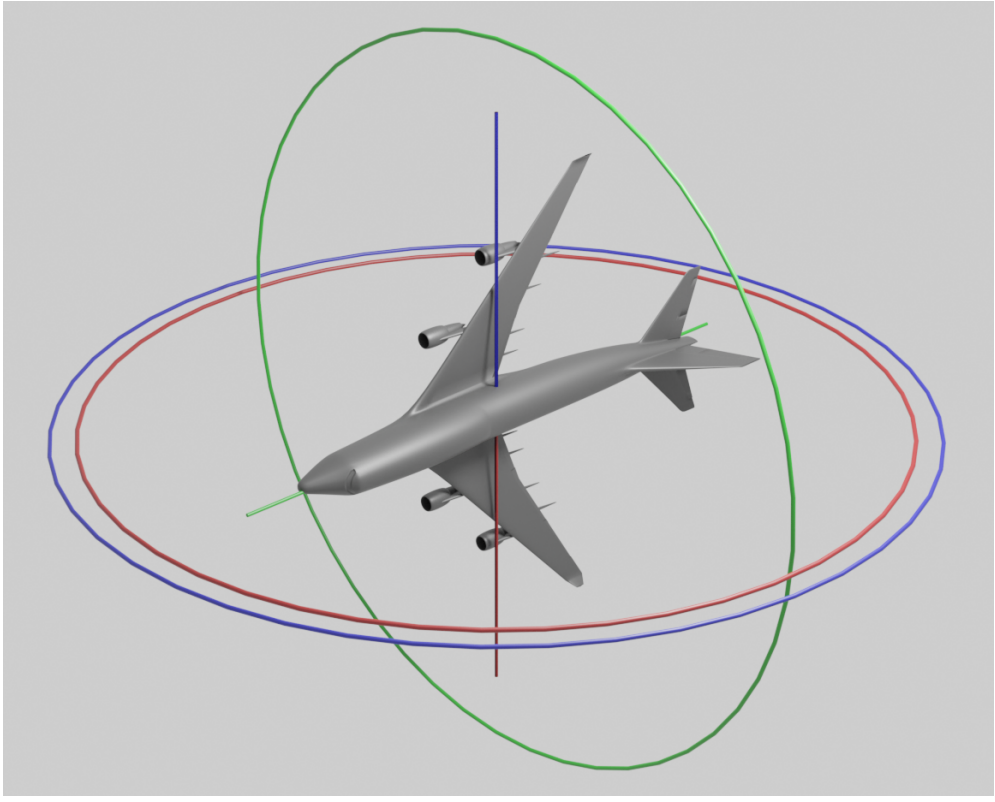
Gimbal lock

Gimbal lock je jev, který vzniká při skládání rotací, pokud se přiblíží osy rotace. Projevuje se tím více, čím se natočení v některé ose blíží 90°.



Gimbal lock

Při natočení kolem osy Y o 90° ztratíme jeden stupeň volnosti.
Zablokováním degeneruje systém do dvourozměrného prostoru.



Rotace

$$X' = \mathbf{R}(\alpha, \beta, \gamma) \cdot X = \mathbf{R}(\alpha) \cdot \mathbf{R}(\beta) \cdot \mathbf{R}(\gamma) \cdot X$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Kvaterniony

Kvaterniony (z lat. quaternion, čtveřice) jsou nekomutativním rozšířením oboru komplexních čísel.

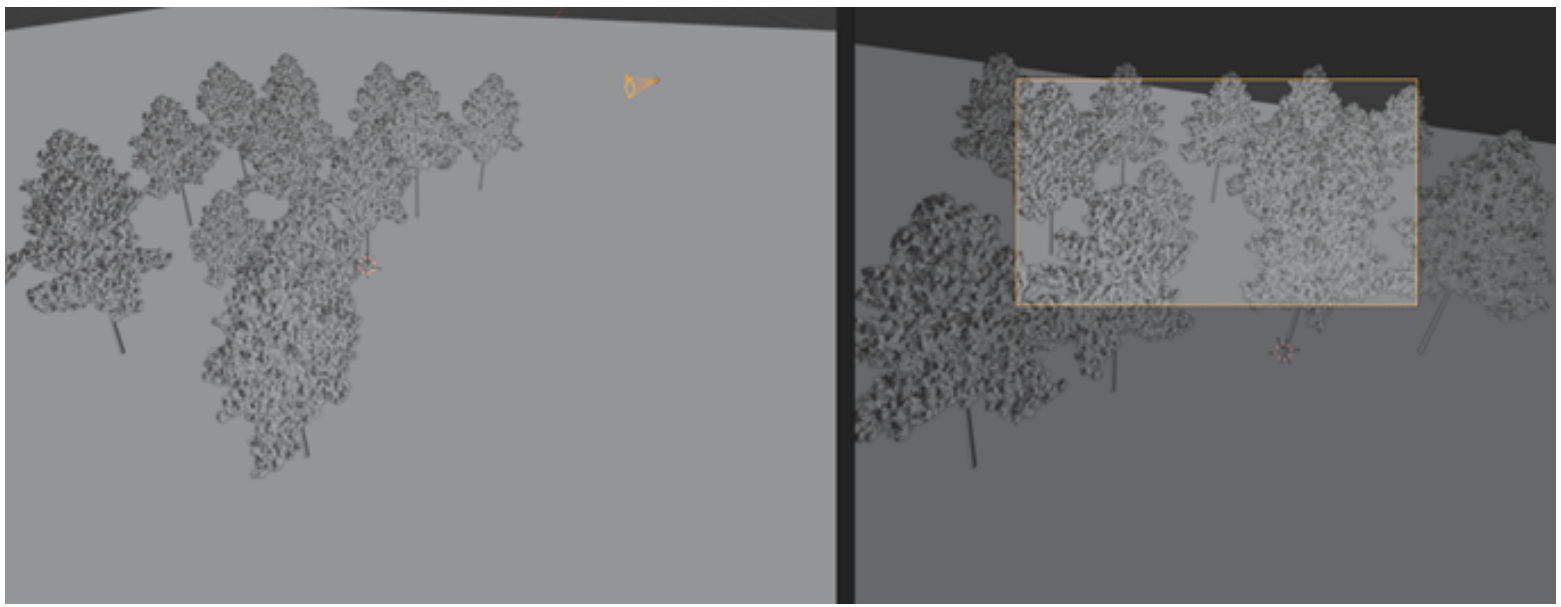
- Autorem byl v roce 1843 sir William Rowan Hamilton (irský matematik) s tím, že když tří rozměry nefunguj ı, tak by mohly čtyři.
- Jednoduchá metodu, jak otáčet tělesa v prostoru.
- Skládají se z jedné reálné a a třech imaginárních složek i, j, k.
- Nepotřebujeme pro každou osu x,y,z rotační matici a jejich konečnou matici.
- Kvaternion má výhodu v tom, že rotovat můžeme kolem osy procházející středem souřadné soustavy.

$$q = [a, \mathbf{v}] = a + bi + cj + dk$$

Cvičení

Vytvořte kameru, která bude obsahovat informace odkud se dívám do scény, kterým směrem a vektor nahoru.

Dále bude kamera obsahovat typ a vlastnosti promítání (perspektivu).

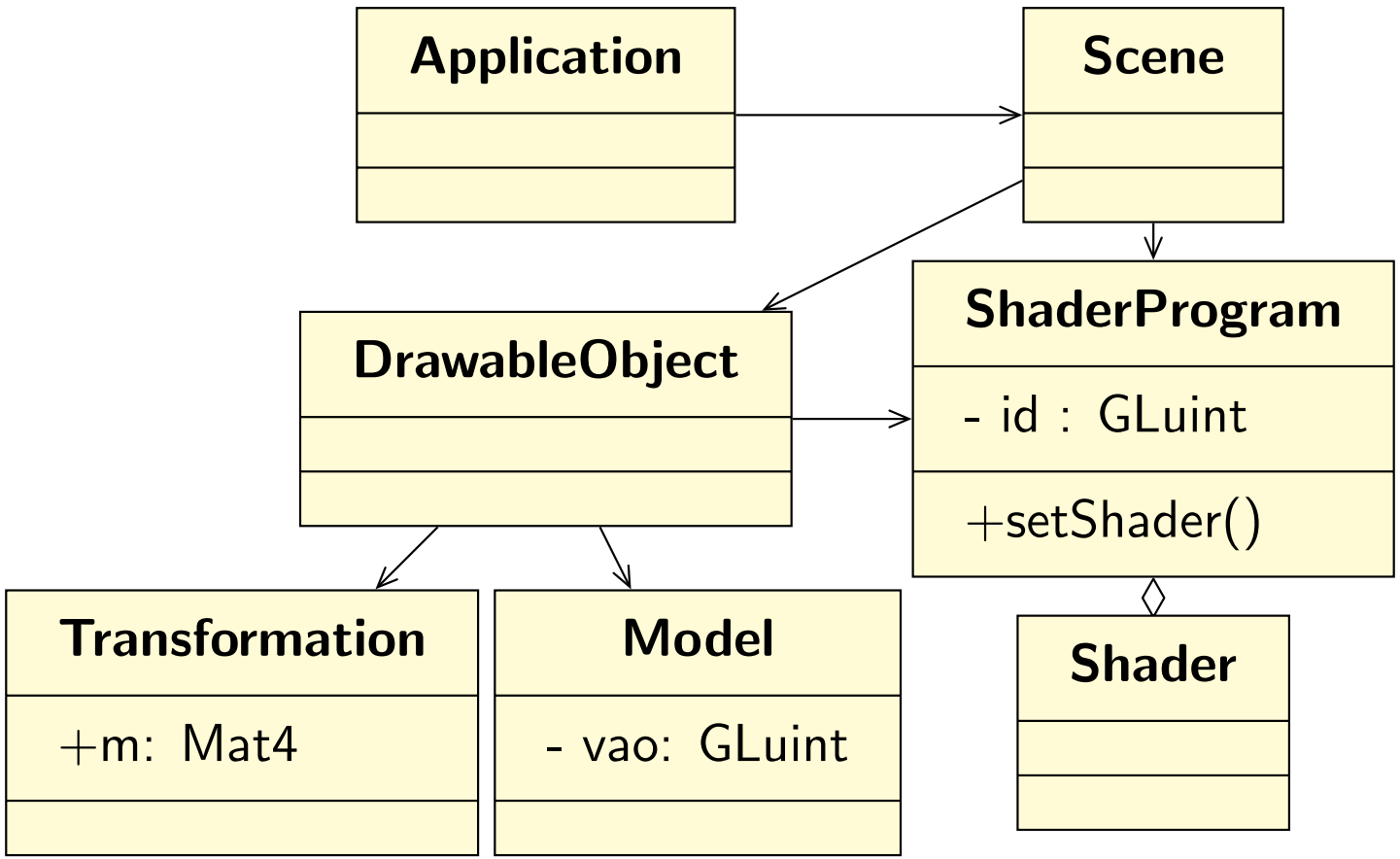


Kamera

```
#version 400
layout(location = 0) in vec3 localPosition;
out vec4 worldPos;
uniform mat4 modelMatrix;
//uniform mat4 viewMatrix;
//uniform mat4 projectionMatrix;

void main(void)
{
  worldPos = modelMatrix * vec4(localPosition, 1.0);
  gl_Position = worldPos;
}
```

Projekt


Application
Scene
DrawableObject
ShaderProgram
- id : GLuint
+setShader()
Transformation
+m: Mat4
Model
- vao: GLuint
Shader

Závěr

Dotazy?