Študijný materiál k 1. a 2. cvičeniu

z predmetu Fyzikálne základy počítačových hier

OpenGL

OpenGL (*Open Graphics Library*) je priemyselný štandard špecifikujúci viacplatformové rozhranie (API) k akcelerovaným grafickým kartám respektíve celým grafickým subsystémom. Slúži na tvorbu aplikácií pracujúcich predovšetkým s trojrozmernou počítačovou grafikou prekresľovanou v reálnom čase. Používa sa pri tvorbe počítačových hier, CAD programov, aplikácií virtuálnej reality alebo pre vedecko-technické vizualizácie.

Ide o zbierku funkcií a tried (ale aj iných programov), ktoré určujú akým spôsobom sa majú funkcie knižníc volá zo zdrojového kódu programu. API funkcie sú programové celky, ktoré programátor volá namiesto vlastného naprogramovania. Knižnica OpenGL obsahuje viac ako 500 rôznych príkazov, ktoré sa používajú na zadefinovanie objektov, obrázkov a operácií, ktoré sú potrebné pri vytváraní interaktívnych trojrozmerných počítačových grafických aplikácií.

OpenGL je navrhnuté ako efektívne, hardvérovo nezávislé rozhranie ktoré môže byť implementované na mnohých rôznych typoch grafického hardvéru, systémoch alebo v rámci softvéru, nezávisle od operačného alebo okenného systému počítača. OpenGL však nezahŕňa funkcie na vykonávanie úloh okien alebo spracovanie používateľského vstupu; namiesto toho musí aplikácia po použiť možnosti, ktoré poskytuje okenný systém daného operačného systému.

OpenGL tiež neumožňuje priame načítavanie obrázkov (ako sú napríklad súbory JPEG). Namiesto toho je potrebné vytvoriť vlastné trojrozmerné objekty z malej množiny geometricky jednoduchých objektov tzv. "geometrických primitív" ako sú body, čiary, trojuholníky a plochy.

FreeGLUT

FreeGLUT je voľne šíriteľná verzia *OpenGL Utility Toolkit (GLUT)*. Je to vysokoúrovňová nadstavba, ktorá nás odbremeňuje od zdĺhavého programovania, umožňuje tvorbu okien v danom prostredí a ich ovládanie. Programy tvorené pomocou tejto knižnice sú použiteľné nezávisle od operačných systémov.

Na cvičení budeme používať:

Jazyk programovania: C

Kompilátor: gcc

Kompilovanie použitím kompilátora gcc s prilinkovaním potrebných knižníc (OpenGL, GLUT, GL Utility Toolkit) je nasledovné:

```
gcc -Wall mojprogram.c -lfreeglut -lopengl32 -lglu32 -o i.x
```

Budeme používať nasledovné funkcie:

FreeGLUT:

`glutInit(&argc, argv);`
inicializácia knižnice GLUT (napr. vytvorenie okna)

`glutCreateWindow("Ja som okno");`
vytvorenie okna s názvom "Ja som okno"

`glutDisplayFunc(procedura);`
„callback“ zobrazenia aktuálneho okna; prekreslenie scény, napr. zmena veľkosti okna, posunutie objektov, a i....,

`glutMainLoop();`
vstup do procesného cyklu GLUT udalostí, cyklenie časti kódu

`glutInitDisplayMode(GLUT_DOUBLE);`
inicializácia zobrazovacieho módu, GLUT_DOUBLE - nastavenie „double buffering“

`glutSwapBuffers();`
prehodenie informácií medzi dvoma buffermi (zo zadného prípravného buffera do predného vykresľovacieho buffera, potrebné na vykreslenie scény

`glutInitWindowSize(int width, int height);`
počiatočný rozmer okna v pixeloch

`glutInitWindowPosition(int x, int y);`
počiatočná poloha okna, súradnice ľavého horného rohu okna aj s dekoráciou, v pixeloch

`glutReshapeFunc(void (*func)(int vyska, int sirka));`
funkcia volajúca „callback“ funkciu zabezpečujúcu správne (nedeformované) zobrazenie premietanej scény pri zmene veľkosti okna (napr. zachovanie pomerov strán pri kolmej projekcii). „Reshape callback“ je spustený pri zmene veľkosti okna. Parametre vyska a sirka „callback“-u špecifikujú novú veľkosť okna.

`glutPostRedisplay();`
označí aktuálne okno ako kandidáta na aktualizáciu, zabezpečí vykreslenie aktualizovaného obrazu (napr. pri zmene súradníc)

`glutTimerFunc(icaskrok, aktualizuj, 0);`
nastaví, že sa po uplynutí časového kroku (v milisekundách) volá callback, v tomto prípade `aktualizuj()`, týmto spôsobom funguje ako „refresh“ po definovanom čase
(Ak použijeme ako posledný argument premennú, napr. ihod, predstavuje táto počet vykreslení a v prípade potreby sa dá pomocou tejto premennej definovať napríklad zmena rýchlosti posunu pri použití dvoch rôznych funkcií aktualizuj, ktoré budú

zabezpečovať prekreslenie scény s rozdielnou hodnotou posunu, pričom každá z nich bude volaná pre iné hodnoty premennej ihod (napr. pre ihod<20 bude volaná funkcia aktualizuj1 s posunom 0.05 a pre ihod>=20 bude použitá funkcia aktualizuj2 s nastaveným posunom na 0.5)

OpenGL:

```
glClear(GL_COLOR_BUFFER_BIT);
```
resetovanie farebných bufferov na prednastavenú farbu pomocou glClearColor()

```
glClearColor(0.0, 1.0, 0.0, 0.0);
```
nastavenie premazávacej farby farebného buffera, RGB v intervale (0, 1)

```
glColor3f(GLfloat R, GLfloat G, GLfloat B);
```
definovanie farby výplne plošného objektu

```
glViewport(0, 0, sirka, vyska);
```
povie knižnici, aký fyzický priestor má naozaj k dispozícii – na akom bode začína, akú má šírku a výšku v pixeloch.

```
glBegin(GL_TRIANGLES);
```
začiatok špecifikácie objektov na scéne, objekty – „primitives“ – vytvárané z vrcholov špecifikovaných medzi glBegin a glEnd;
GL_LINES – spájané sú dva vrcholy – čiara
GL_TRIANGLES – spájané sú tri vrcholy – plný trojuholník
GL_QUADS - spájané sú štyri vrcholy – plný štvoruholník

```
glEnd();
```
ukončenie kreslenia objektov na scéne

```
glVertex2f();
```
definovanie súradníc (x, y) vrcholu

```
glMatrixMode();
```
špecifikuje, ktorá matica bude objektom ďalšieho spracovania pomocou maticových operácií (čoho sa budú týkať transformácie)

```
GL_MODELVIEW;
```
– modelová matica v „local space“, teda nasledujúce nastavenia sa týkajú objektov, ktoré budeme do scény vkladať

```
GL_PROJECTION;
```
– projekčná matica vo „view space“, teda nasledujúce nastavenia sa týkajú spôsobu, ako na scénu pozeráme (kamera)

```
glLoadIdentity();
```
nahradí aktuálnu maticu jednotkovou (resetovanie transformačnej matice).
(ekvivalentné volaniu glLoadMatrix s jednotkovou maticou).

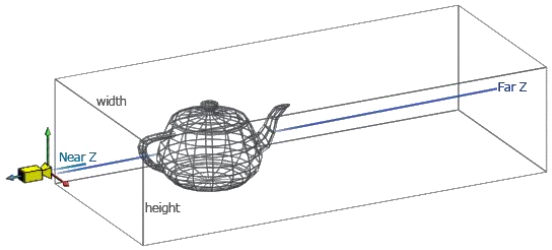
```
glOrtho(left, right, bottom, top, nearVal, farVal);
```

násobí aktuálnu maticu ortografickou maticou. Ekvivalentná operácia ku gluOrtho2D (viď nižšie), avšak tu sú špecifikované aj z-ové súradnice projekčného kvádra (v prípade gluOrtho2D sú tieto explicitne z intervalu (-1,1))

GL Utilities:

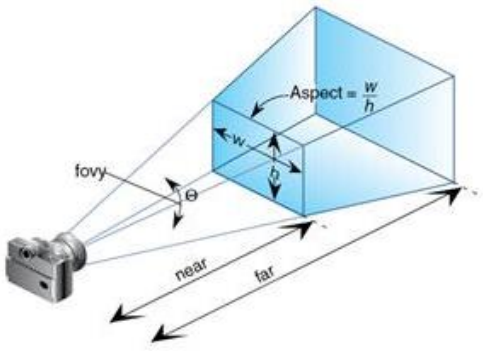
```
gluOrtho2D(left, right, bottom, top);
```

definuje projekčný kváder pri ortografickej (kolmej) projekcii (polohy rovín orezania priestoru scény). Definovanie škálovacej/merítkovej/ortografickej projekčnej matice, ktorá preškáluje súradnice projekčného kvádra na súradnice z intervalu (-1,1).
Left, right – špecifikácia súradníc ľavej a pravej roviny orezania („clipping plane“); bottom, top - špecifikácia súradníc spodnej a hornej roviny orezania. Ekvivalent z knižnice OpenGL: glOrtho.




```
gluPerspective(fovy, aspect, zNear, zFar);
```

definuje „view frustum“ pri perspektívnom zobrazení. Kamera sa umiestni do počiatku súradnicovej sústavy a pozerá sa v smere osi z smerom, ktorým z-ová súradnica klesá. Prvé dva parametre určujú ihlan, ktorý má kamera v zábere. Prvý je uhol medzi hornou a dolnou rovinou a druhý je pomer medzi šírkou a výškou obrazu. Ďalšie dva parametre určujú, odkiaľ pokiaľ kamera vidí.



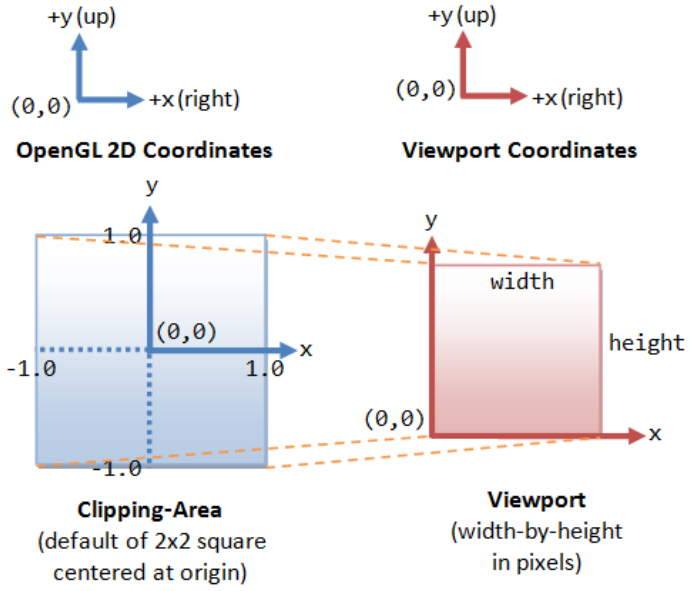
https://stackoverflow.com/questions/8171590/how-to-know-the-plane-size-in-units

```
glTranslate(x, y, z);
```

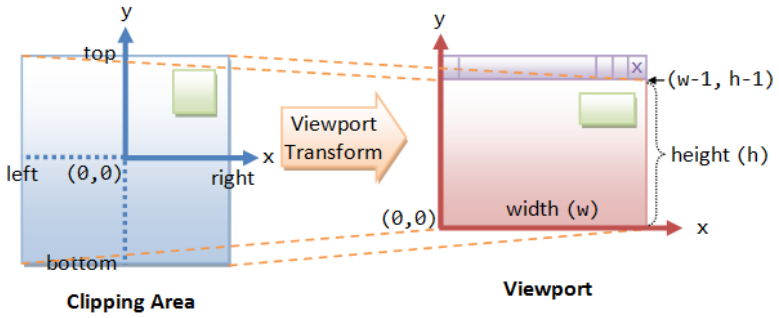
násobenie aktuálnej matice translačnou maticou (maticou posunutia), špecifikácia (x, y, z) súradníc translačného vektora

“Viewport” (výrez) a “clipping area” (oblasť orezania) v OpenGL:

“Viewport” = zobrazená oblasť v okne meraná v pixeloch v súradniciach obrazovky
“Clipping Area” = oblasť videná kamerou meraná v OpenGL súradniciach (-1.0, 1.0, -1.0, 1.0)

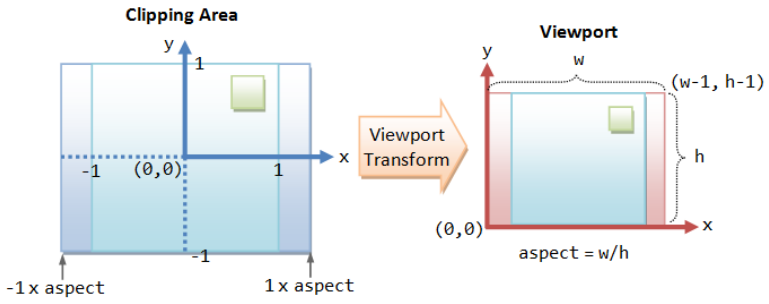


Problém zmeny proporcií objektov pri zmene veľkosti okna:



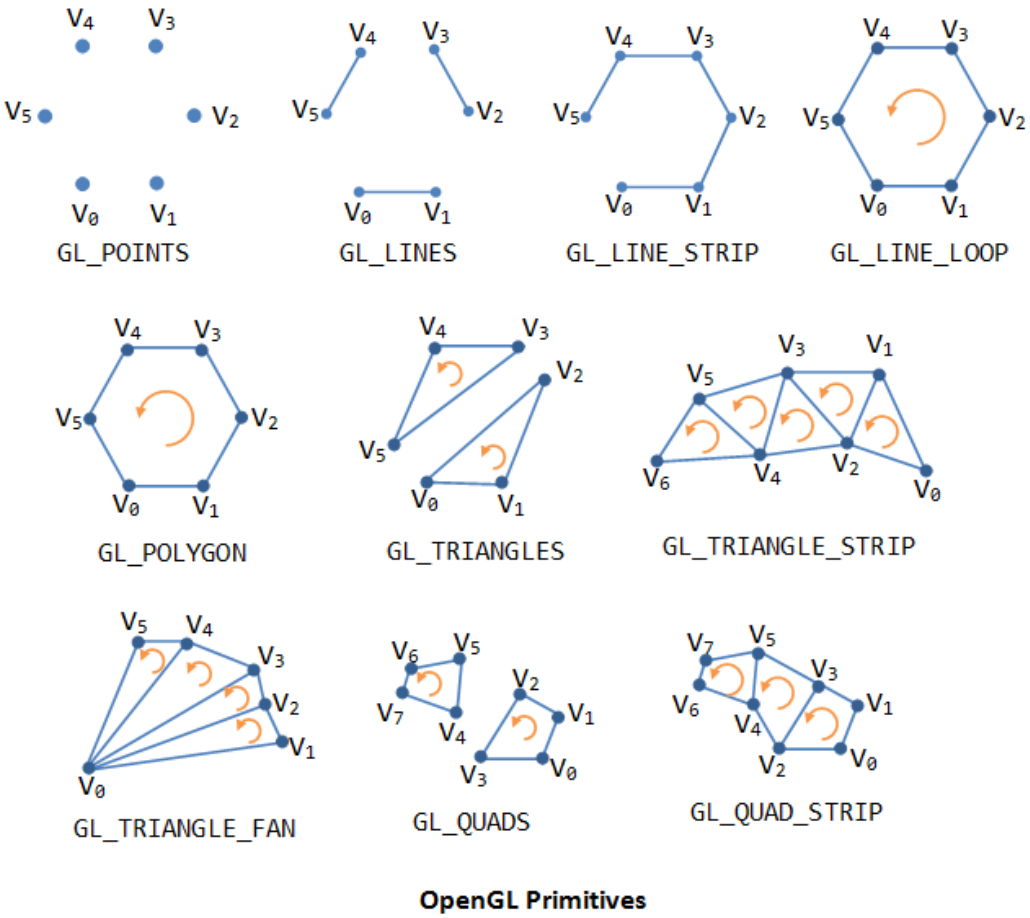
Clipping Area and Viewport: Objects will be distorted if the aspect ratios of the clipping area and viewport are different.

Riešenie: gluReshapeFunc() - funkcia zabezpečujúca nastavenie veľkosti “clipping area” podľa pomerov strán okna. Pomocou funkcie gluOrtho2D je možné nastaviť oblasť orezania 2D ortografického zobrazenia. Objekty mimo oblasti orezania (zorného poľa) nebudú viditeľné.



Clipping Area and Viewport: same aspect ratio for the clipping area and viewport to ensure that the objects are not distorted.

"Geometric primitives" - základné geometrické prvky v OpenGL:



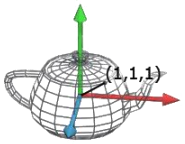
OpenGL Primitives

https://www3.ntu.edu.sg/home/ehchua/programming/opengl/cg_introduction.html

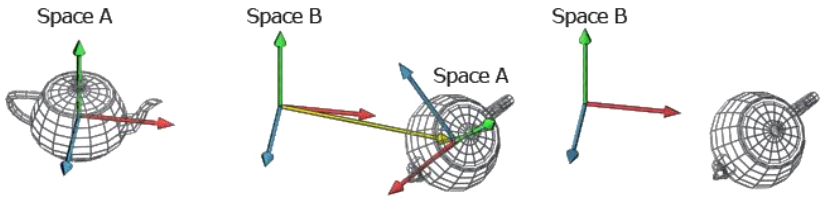
Transformačné matice v OpenGL

„World“, „View“ a „Projection Transformation Matrices“

Každý model vytvorený v OpenGL “žije” vo svojom modelovom priestore (“**Model Space**”) danom pravotočivým súradnicovým systémom so súradnicami x, y, z (1,1,1):



Ak chceme umiestniť objekt do herného sveta, budeme ho musieť presunúť a/alebo otočiť do požadovanej polohy, čím sa objekt umiestni do „**World Space**“. Presúvanie, otáčanie alebo zmenu mierky objektu nazývame **transformácia**. Keď budú všetky objekty transformované do spoločného priestoru („World Space“), ich vrcholy budú teda relatívne vzhľadom na „World Space“. Transformácie vo vektorovom priestore môžeme vidieť jednoducho ako zmenu z jedného priestoru (A) do druhého (B):



Transformácie, ktoré môžeme použiť vo vektorových priestoroch, sú škálovanie (zmena mierky), translácia a rotácia.

Transformáciu z jedného 3D priestoru do druhého vykonávame pomocou násobenia všetkých vektorov maticou 4x4, **tzv. transformačnou maticou**. Ak máme vektory v priestore A a transformácia má popísať novú polohu priestoru A vzhľadom na priestor B, po vynásobení transformačnou maticou budú všetky vektory opísané v priestore B.

Transformačné matice:

Translačná (4. stĺpec predstavuje súradnice vektora posunutia):

$$\begin{bmatrix} 1 & 0 & 0 & Translation.x \\ 0 & 1 & 0 & Translation.y \\ 0 & 0 & 1 & Translation.z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Škálovacia/merítková:

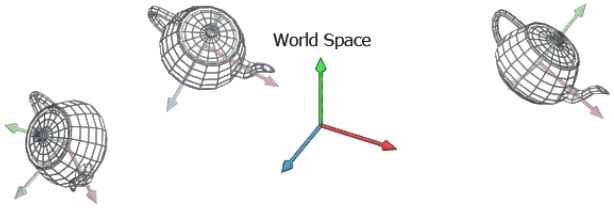
$$\begin{bmatrix} Scale.x & 0 & 0 & 0 \\ 0 & Scale.y & 0 & 0 \\ 0 & 0 & Scale.z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotačná okolo osi x, y, resp. z:

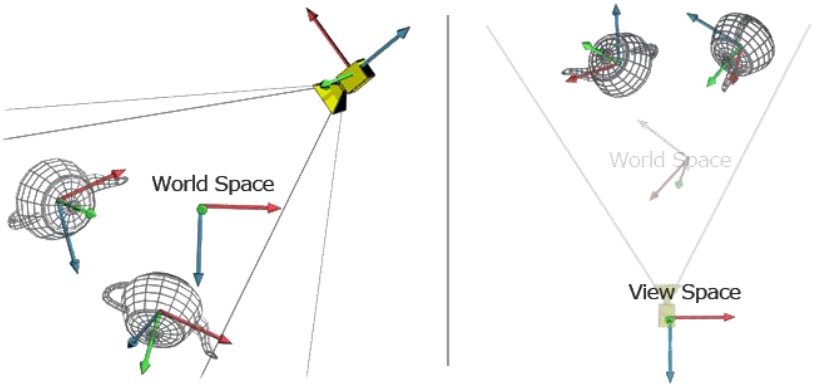
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix}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

Vynásobením matíc jednej po druhej môžeme zreťaziť niekoľko transformácií po sebe. Výsledkom bude jedna matica, ktorá zakóduje výslednú transformáciu.

Majme teraz tri objekty umiestnené vo „World Space“ v rôznych pozíciách a orientáciách podľa obrázku (každý objekt má svoju „Model to World Space“ transformačnú maticu.



„World space“ so všetkými objektami na správnom mieste musíme teraz premietnuť na obrazovku. Zvyčajne sa to robí v dvoch krokoch. Prvý krok presunie všetky objekty do iného priestoru nazývaného „**View Space**“. V druhom kroku sa vykoná skutočná projekcia pomocou **projekčnej matice** do projekčného priestoru (**„Projection (Clip) Space“**).



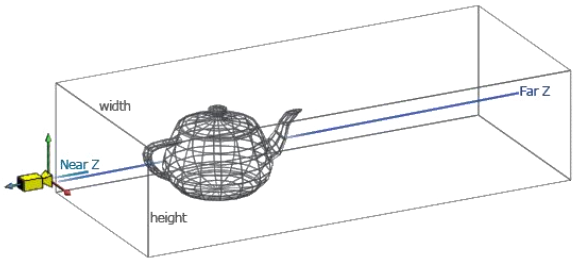
„View Space“ je pomocný priestor, ktorý používame na zjednodušenie matematiky pretože potrebujeme premietať všetky vrcholy na obrazovku kamery, ktorá môže byť ľubovoľne orientovaná v priestore. Matematika sa veľmi zjednoduší, ak kameru umiestnime do počiatku a budeme sledovať jednu z troch osí (os z). Preto „premapujeme“ „World Space“ tak, aby kamera bola v počiatku a pozerala sa dole pozdĺž osi z. Tento priestor sa nazýva „View Space“ (niekedy „Camera Space“) a transformácia, ktorú aplikujeme, presúva všetky vrcholy z „World Space“ do „View Space“. Transformačná maticu pre „View Space“ je teda spojením dvoch transformácií (z „Model“ do „World“ a z „World“ do „View“).

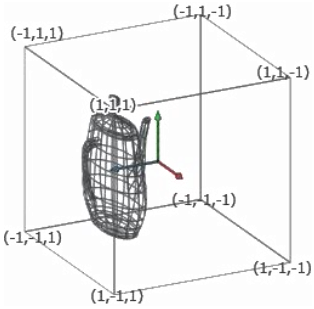
Scéna je teraz v najvhodnejšom priestore pre projekciu, vo „View Space“. Všetko, čo teraz musíme urobiť, je premietnuť ho na pomyselnú obrazovku kamery. Pred vytvorením obrazu sa ešte musíme presunúť do **projekčného priestoru**. Tento priestor predstavuje kváder, ktorého súradnice sú od -1 do 1 pre každú os a je veľmi užitočný na orezávanie (čokoľvek mimo rozsahu 1:-1 je mimo oblasti zobrazenia kamery) a zjednodušuje operáciu sploštenia (na získanie plochého obrazu musíme jednoducho znížiť hodnotu z).

Aby sme prešli z priestoru zobrazenia do projekčného priestoru, potrebujeme ďalšiu maticu „**View to Projection**“, a hodnoty tejto matice závisia od toho, aký typ projekcie chceme vykonať. Dve najpoužívanejšie projekcie sú **ortografická** projekcia a **perspektívna** projekcia.

Na vykonanie **ortografickej projekcie** musíme definovať veľkosť oblasti, ktorú môže kamera vidieť. Tá je zvyčajne definovaná hodnotami šírky a výšky pre os x a y a hodnotami blízkych a vzdialených z pre os z. Pomocou týchto hodnôt môžeme vytvoriť transformačnú maticu, ktorá „premapuje“ oblasť boxu

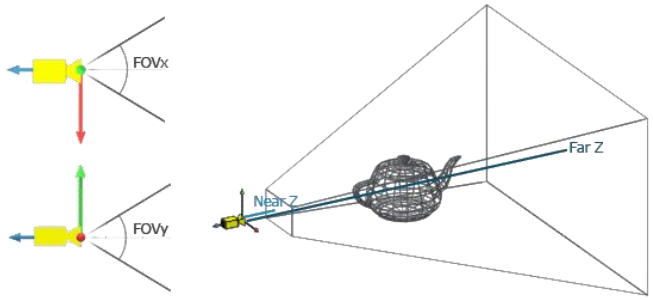
na kocku. Táto matica transformuje vektory z „View Space“ do priestoru „Ortho Projection Space“ (pravotočivý súradnicový systém).

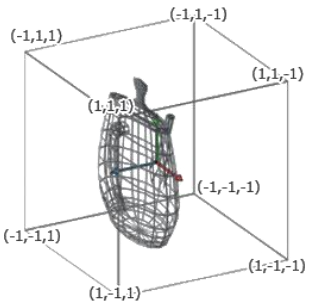




$$\begin{bmatrix} \frac{1}{width} & 0 & 0 & 0 \\ 0 & \frac{1}{height} & 0 & 0 \\ 0 & 0 & -\frac{2}{Z_{far}-Z_{near}} & -\frac{Z_{far}+Z_{near}}{Z_{far}-Z_{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

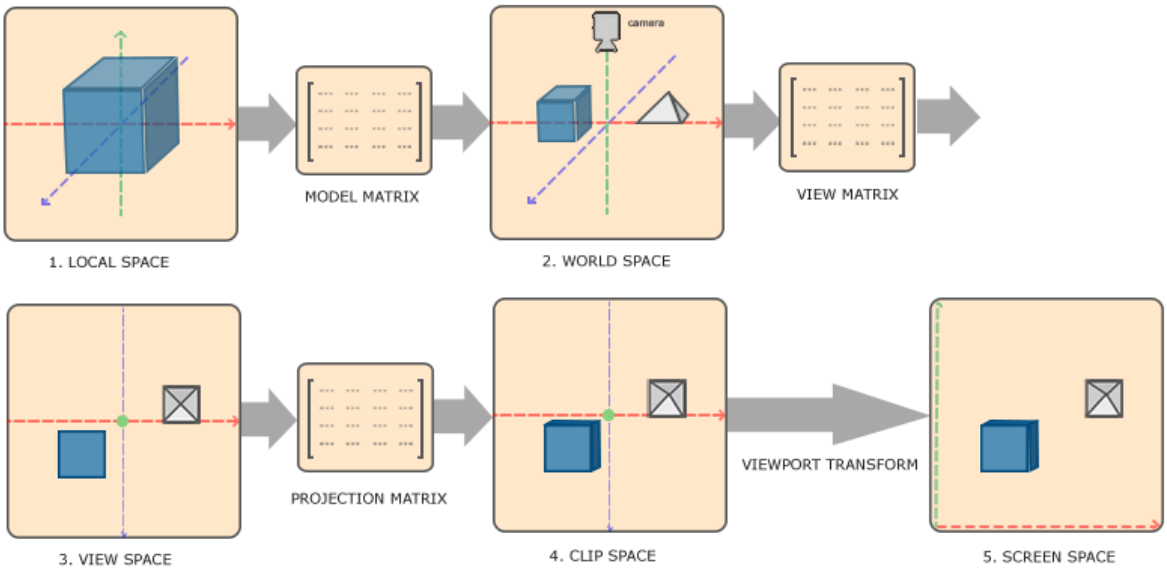
Ďalšou projekciou je perspektívna projekcia, pri ktorej je oblasť zobrazenia zrezaná (frustum), a preto je jej „premapovanie“ o niečo zložitejšie.





$$\begin{bmatrix} \tan^{-1}(\frac{FOVx}{2}) & 0 & 0 & 0 \\ 0 & \tan^{-1}(\frac{FOVy}{2}) & 0 & 0 \\ 0 & 0 & -\frac{Z_{far}+Z_{near}}{Z_{far}-Z_{near}} & -\frac{2\,(Z_{near}\,Z_{far})}{Z_{far}-Z_{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspektívny priestor je posledným krokom reťazca transformácií, o všetko ostatné sa postará GPU (orezanie, „premapovanie“ všetkého z rozsahu -1 až 1 do rozsahu 0 až 1, zmenšenie na šírku a výšku „Viewportu“).



https://learnopengl.com/Getting-started/Coordinate-Systems

Literatúra a užitočné odkazy:

OpenGL:

https://www.opengl.org/

https://www.khronos.org/registry/OpenGL-Refpages/

https://learnopengl.com/book/book_pdf.pdf

https://learnopengl.com/Getting-started/Coordinate-Systems

https://www3.ntu.edu.sg/home/ehchua/programming/opengl/cg_introduction.html

http://www.codinglabs.net/article_world_view_projection_matrix.aspx

John Kessenich, Graham Sellers, Dave Shreiner: *The OpenGL Programming Guide*

Shreiner D. et al.: OpenGL Programming Guide: The O_cial Guide to Learning OpenGL, Version 4.3., Addison-Wesley Professional 2013, 8th edition, isbn 0321773039.

http://nehe.ceske-hry.cz/subdom/nehe/tut_obsah.php

GLUT:

http://freeglut.sourceforge.net/

https://www.opengl.org/resources/libraries/glut/

Fyzika:

Július Cirák a kol.: *Zbierka príkladov a úloh z fyziky*

Elektronické skriptá prof. Balla:

http://kf-lin.elf.stuba.sk/~ballo/fyzika_online/Fyzika%20I/e-fyzika1-frame-2mechanikaHB.htm

Príklady z fyziky:

http://www.ujfi.fei.stuba.sk/fyzika-priklady.php