VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Jan Morávek

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

INTERAKTIVNÍ WEBOVÉ APLIKACE PRO PODPORU VÝUKY 3D POČÍTAČOVÉ GRAFIKY
INTERACTIVE WEB APPLICATIONS SUPPORTING EDUCATION OF 3D GRAPHICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Jan Morávek

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Mgr. Pavel Rajmic, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Jan Morávek ***ID:*** 175362
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Interaktivní webové aplikace pro podporu výuky 3D počítačové grafiky

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte teorii k níže konkretizovaným aplikacím, graficky je navrhněte, implementujte a otestujte. Kombinací HTML a JavaScriptu vytvořte tři webové aplikace. Pro 3D modelování je vhodné využít knihoven jako jsou WebGL nebo P5.
Aplikace se tématicky týkají vektorové trojrozměrné grafiky, konkrétně budou zaměřeny na:
1) parametrickou konstrukci Bézierova plátu,
2) geometrické transformace 3D objektů,
3) projekce objektů na průmětnu, role kamery.
Zaměřte se především na názornou podobu, interaktivitu a funkčnost pro
potřebu výuky. Každou aplikaci vložte do HTML stránky, která bude obsahovat i stručný souhrn teorie.

DOPORUČENÁ LITERATURA:

[1] Beneš, B.; Sochor, J.; Felkel, P.; Žára, J.: Moderní počítačová grafika. Computer Press, Brno, 2005.

[2] Piegl, L.,; Tiller, W.: The NURBS Book. Druhé vydání. Springer, 1997

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: doc. Mgr. Pavel Rajmic, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Tato diplomová práce se zabývá počítačovou 3D grafikou a implementací výukových aplikací v jazyce JavaScript. Mezi probírané oblasti práce patří transformace objektů, Beziérovy pláty a role kamery ve scéně. V práci je popsán teoretický základ těchto oblastí a následně se práce věnuje vytvořeným výukovým aplikacím. Práce obsahuje detailní popis fungování a implementace vytvořených aplikací. V závěru práce jsou zmíněna možná rozšíření těchto aplikací.

KLÍČOVÁ SLOVA

2D, 3D, počítačová grafika, grafika, křivky, plochy, transformace, pláty, projekce, kamera, JavaScript, THREE.js

ABSTRACT

This diploma thesis is focused on computer 3D graphics and the implementation of educational applications in JavaScript language. Discussed topics of computer graphics include object transformations, Bezier patches and the role of the camera in the scene. The thesis describes the basic theory of these areas and educational applications. The thesis also includes the detailed description of the features and the implementation of the created applications. In the end of the thesis possible extensions are discussed.

KEYWORDS

2D, 3D, computer graphics, graphics, curves, surfaces, transformations, patches, projection, camera, JavaScript, THREE.js

MORÁVEK, Jan. *Interaktivní webové aplikace pro podporu výuky 3D počítačové grafiky*. Brno, 2021, 70 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Mgr. Pavel Rajmic, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Interaktivní webové aplikace pro podporu výuky 3D počítačové grafiky“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Mgr. Pavlu Rajmicovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Úvod

S rozvojem technologií v současné době se objevují nové možnosti interaktivní výuky. Jedním z přístupů je vytvoření interaktivní webové aplikace, na které si může student vyzkoušet probíranou látku a tím jí i lépe porozumět. Tato práce je zaměřena právě na vytvoření tří výukových webových aplikací v jazyce JavaScript. Jednotlivé aplikace jsou tématicky zaměřeny na vybrané oblasti 3D grafiky a ukazují názorné příklady v těchto oblastech. Konkrétně jsou zaměřeny na parametrické konstrukce Beziérova plátu, geometrické transformace 3D objektů a role kamery při perspektivní projekci objektů na průmětnu. Výsledné aplikace budou sloužit jako interaktivní nástroje pro výuku a budou součástí výukového materiálu na FEKT VUT.

První kapitola je zaměřena na teoretický úvod do 3D grafiky. V sekci 1.1 jsou nejdříve rozebrány základy křivek ve 2D, tedy jaká různá vyjádření můžeme pro křivky používat, jaké jsou jejich požadované vlastnosti a jak můžeme křivky dělit. Následně na křivky navazuje sekce 1.2, která popisované křivky ve 2D rozšiřuje na popis ploch ve 3D. V této části jsou popsány i základy parametrické konstrukce Beziérova plátu, tedy teoretické základy pro druhou aplikaci.

V další sekci 1.3 jsou vysvětleny základy geometrických transformací 3D objektů, které jsou potřebným základem pro realizaci první aplikace. V poslední části 1.4 kapitoly je popsáno fungování projekce, tedy zobrazování na průmětnu, čemuž se věnuje třetí aplikace.

V další kapitole 2 je popsána technická část práce. Jsou zde krátce zmíněny různé možnosti vytváření grafických aplikací v jazyce JavaScript a využité technologie pro implementaci aplikací. Následuje popis funkcionality aplikací, jaké prvky obsahují a jak se jednotlivé aplikace ovládají. Kapitola je ukončena částí 3.3 s detailním popisem implementace 3.3 jednotlivých aplikací.

V poslední kapitole 4 je obsaženo krátké zhodnocení jednotlivých prvků funkcionality a implementace v aplikacích. Současně s tím jsou popsány návrhy pro případné rozšíření a úpravy aplikací, jak z hlediska přidání funkcionalit, tak z hlediska zlepšení implementace.

1 Základy 3D grafiky

Tato kapitola pojednává o základech reprezentace 3D objektů v počítači. Začátek kapitoly 1.1 se věnuje popisu křivek, jejich vyjádření, vlastnostem a jako příklad jsou popsány Beziérovy křivky. Následuje část 1.2 popisující plochy, jejich vyjádření, vlastnosti a věnuje se Beziérovým plátům. Dále jsou v kapitole 1.3 popsány základní geometrické afinní transformace objektů ve 3D prostoru. Závěrem je v kapitole 1.4 popsána projekce scény.

1.1 Křivky

Křivky jsou základními prvky počítačové grafiky. Jsou běžně využívány pro modelování objektů, vytváření jednotlivých znaků fontu nebo například definici pohybu objektů v animaci. S křivkami se můžeme setkat u 2D i 3D objektů, pro zjednodušení popisu jsou v této kapitole popisovány křivky ve dvou rozměrech.

1.1.1 Vyjádření křivek

Křivku můžeme chápat jako nekonečnou množinu bodů, kde kromě počátečního a koncového bodu má každý bod dva sousední body. V počítači můžeme křivku definovat pomocí geometrických parametrů, respektive pomocí parametrů určité křivky. Pro definici této křivky můžeme využít třech různých vyjádření:

- explicitní vyjádření,
- implicitní vyjádření,
- parametrické vyjádření.

Explicitní vyjádření je možné použít pro křivky, které jsou zároveň funkcemi, tedy pro každou hodnotu x z definičního oboru náleží právě jedna funkční hodnota y. Explicitní tvar křivky můžeme zapsat následujícím zápisem (1.1).

$$y = f(x) \tag{1.1}$$

Z popisu explicitního vyjádření křivky může být zřejmé, že není vhodné pro využití v počítačové grafice, jelikož ve většině případů je naším cílem modelování křivek, které nejsou zároveň funkcemi. [1]

Implicitní vyjádření křivky určuje jednotlivé body na křivce pomocí závislosti neznámých a funkce v předpisu, tedy s pomocí proměnných x a y. Implicitní vyjádření je v mnoha případech možné převést na explicitní, nicméně v oblasti počítačové grafiky nám to nepřinese výhodu. Implicitní vyjádření křivky můžeme zapsat následujícím zápisem (1.2).

$$F(x, y) = 0 \tag{1.2}$$

Implicitní vyjádření v mnoha případech neumožňuje postupný výpočet křivky, tedy pro využití v počítačové grafice je příliš komplikovaný.

Parametrické vyjádření křivky využívá rovnice s parametrem t. Parametr t je proměnný a nabývá určitých hodnot z definovaného intervalu. Interval můžeme volit různě dle modelované křivky. Parametrické vyjádření křivky v trojdimenzionálním prostoru můžeme zapsat pomocí následujícího zápisu (1.3).

$$q(t) = [x(t), y(t)]; t \in [t_{\min}, t_{\max}] \tag{1.3}$$

Parametrické vyjádření křivky je v oblasti počítačové grafiky využíváno nejčastěji pro jeho univerzálnost a jednoduchost. Jsme pomocí něj schopni zapsat různé křivky, i takové, které nejsou zároveň funkcemi. Také body křivky jsou závislé na jedné proměnné, a to parametru t. Pokud za parametr t zvolíme čas, jsme schopni generovat průběh křivky v čase, a to i za předpokladu, že křivka prochází stejnými body v různých časových intervalech. [2][1]

Rozdíl mezi jednotlivými vyjádřeními křivek si můžeme ukázat na příkladu kružnice. Pro kružnici můžeme využít implicitní vyjádření ve tvaru (1.4).

$$x^2 + y^2 = r^2 \tag{1.4}$$

Pro rovnici kružnice nejsme schopni vytvořit explicitní zápis, jelikož se nejedná o funkci, respektive jsme schopni vytvořit explicitní zápis pro polovinu kružnice a následně výslednou křivku invertovat podle osy y a tím získat druhou polovinu kružnice. Případně jsme schopni vytvořit explicitní zápis pro obě poloviny kružnice, tedy pro záporná y a kladná y ve tvaru (1.5). [3][2]

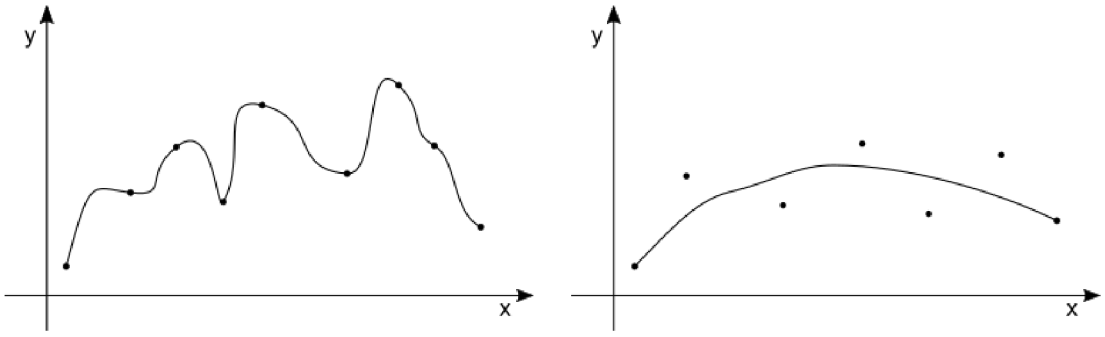
$$y = \sqrt[2]{r^2 - x^2}; y = -\sqrt[2]{r^2 - x^2} \tag{1.5}$$

Pokud bychom chtěli využít výhodného parametrického vyjádření kružnice, tak rovnici můžeme zapsat pro dvojici $[x(t), y(t)]$ ve tvaru (1.6).

$$x(t) = r \cdot \cos(2\pi t); y(t) = r \cdot \sin(2\pi t); t \in [0, 1] \tag{1.6}$$

1.1.2 Dělení křivek

Pro základní dělení křivek využíváme dvou oblastí, a to křivky interpolační a křivky aproximační. Toto rozdělení určuje, zda křivky musí nebo nemusí procházet svými řídícími body. Ukázka obou případů je vidět na obrázku 1.1 . Pro parametrické vyjádření křivek v počítačové grafice se využívá polynomálních křivek, jelikož z nich je možné skládat křivky, jejichž části jsou polynomálními křivkami, tedy tzv. křivky po částech polynomální. [1]



(a) Interpolační křivka. (b) Aproximační křivka.

Obr. 1.1: Ukázka aproximační a interpolační křivky.

Pokud budeme jednotlivé řídící body křivky označovat P_i a samotnou křivku $q(t)$, tak můžeme navazování jednotlivých segmentů křivky označit $q_1(1) = q_2(0)$. Parametricky můžeme využívanou kubiku zapsat následujícím tvarem (1.7).

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \tag{1.7}$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \tag{1.8}$$

Tento tvar můžeme zjednodušit pomocí maticového zápisu (1.10), a to tak, že tvar kubiky v prostoru bude určen parametry matice C a dosazením parametru t získáme bod na křivce v daném časovém okamžiku. Zmíněný maticový tvar rovnice můžeme zapsat v následujícím tvaru (1.9).

$$q(t) = TC = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} \tag{1.9}$$

Maticový zápis (1.9) však není výhodný, a to zejména díky náročné čitelnosti změn parametrů matice C. Pro jednodušší aplikaci změn můžeme matici C rozložit na součin dvou dílčích matic M a G. Přitom matice M bude reprezentovat vlastnosti, které jsou pro definovaný druh křivek společné, a budeme ji nazývat bázová matice. Matice G bude reprezentovat vlastnosti konkrétní definované křivky a nazývá se matice geometrických podmínek. Součinem matic získáme skupinu polynomů, které jsou společné pro definované křivky, jedná se o tzv. polynomální bázi. Takový maticový tvar můžeme zapsat ve tvaru (1.10). [3][2]

$$q(t) = TMG = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \end{bmatrix} \tag{1.10}$$

Beziérovy křivky

Beziérovy křivky jsou základní využívané křivky pro modelování 2D i 3D objektů. Pro tvorbu těchto křivek využíváme Bernsteinových polynomů. Bernsteinovy polynomy n-tého stupně můžeme obecně zapsat tvarem (1.11).

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}; t \in [0,1]; i = 0, 1, \dots n \tag{1.11}$$

Mezi důležité vlastnosti Bernsteinových polynomů patří:

- výsledná křivka leží v konvexní obálce svých řídících bodů,
- Bernsteinovy polynomy jsou nezáporné,
- svého maxima na intervalu $[0, 1]$ nabývá v bodě $\frac{i}{n}$,
- součet Bernsteinových polynomů n-tého stupně je 1.

Zde se dostáváme k zápisu Beziérových křivek pomocí Bernsteinových polynomů, které budou definovat váhu jednotlivých řídících bodů, což je vidět v zápisu (1.12). [4][1][2]

$$q_{Bez}(t) = \sum_{k=0}^{n} P_k B_{k,n}(t) \tag{1.12}$$

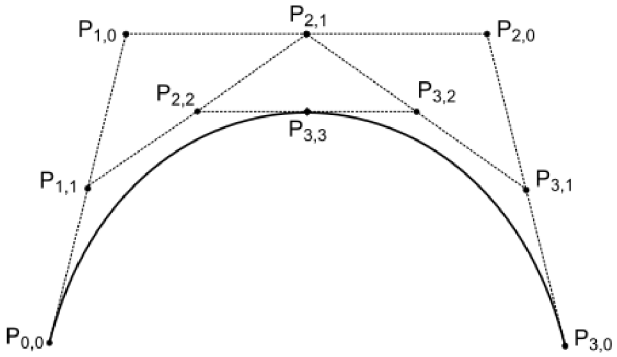
Pokud se vrátíme k maticovému vyjádření (1.11), můžeme říci, že body P_k reprezentují matici G a Bernsteinovy polynomy $B_{k,n}(t)$ reprezentují maticový součin TM. Nejčastěji však využíváme Beziérových křivek třetího stupně, tedy Beziérových kubik, a můžeme je zapsat pomocí maticového zápisu ve tvaru(1.13). [1]

$$q_{\text{bez}}(t) = TMG = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} \tag{1.13}$$

Pro vykreslení Beziérovy křivky můžeme využít přímého postupu, tedy určení bodů na křivce pomocí dosazování časového parametru t a jejich následné spojení. Tento způsob není v praxi příliš vhodný, a to zejména z důvodu časové náročnosti. Lepším a běžně využívaným způsobem je využití algoritmu de Casteljau. Jedná se o rekurzivní algoritmus, který dělí úsečky spojující řídící body na Beziérovy křivky nižšího řádu, tedy využívá matematického vztahu (1.14), který platí pro Beziérovy křivky. [2][4][1]

$$B_{k,n}(t) = (1-t)B_{k,n-1}(t) + tB_{k-1,n-1}(t); t \in [0,1] \tag{1.14}$$

Na vstupu algoritmus zpracovává jednotlivé řídící body P_0 až P_n a časový okamžik t, pro který chceme nalézt bod na křivce. Jednotlivé úsečky řídícího polygonu jsou rozděleny v zadaném poměru, například ve středu, tedy $t = \frac{1}{2}$. Následně jsou vzniklé body propojeny úsečkami a tyto úsečky znovu rozděleny dle parametru t. Algoritmus tento postup rekurzivně opakuje do té doby, než zůstane jeden hledaný bod. Vyhledání bodu na křivce s parametrem $t = \frac{1}{2}$ můžeme vidět na obrázku 1.2. [2]



Obr. 1.2: Hledání bodu na křivce pomocí algoritmu de Casteljau.

1.1.3 Vlastnosti křivek

Jednou ze základních vlastností křivek je třída spojitosti křivky. Třídu spojitosti můžeme určit na základě spojitosti derivací v bodech křivky. Pokud parametricky vyjádřenou křivku zderivujeme po složkách, získáme tečný vektor v bodě křivky. Výpočet tečného vektoru můžeme zapsat vztahem (1.15).

$$\vec{q'}(t_0) = (x'(t_0), y'(t_0)) = \left(\frac{dx(t_0)}{dt}, \frac{dy(t_0)}{dt}\right) \tag{1.15}$$

Třídu spojitosti křivek definujeme v případě skládání komplikovanější křivky z dílčích segmentů, které jsou samostatnými křivkami. Máme-li křivku $q(t)$, můžeme o ní říci, že patří do parametrické třídy spojitosti C^n, pokud má do řádu n (včetně) spojité derivace ve všech bodech. Tedy pokud máme křivku $q(t)$ složenou ze segmentů $q_1(t)$ a $q_2(t)$, mají segmenty třídu spojitosti:

- C^0 pokud je koncový bod segmentu $q_1(t)$ roven počátečnímu bodu segmentu $q_2(t)$,
- C^1 pokud je tečný vektor v koncovém bodu segmentu $q_1(t)$ roven tečnému vektoru v počátečním bodu segmentu $q_2(t)$,
- C^n pokud je n-tá derivace v koncovém bodu segmentu $q_1(t)$ rovna n-té derivaci v počátečním bodu segmentu $q_2(t)$.

Můžeme tedy říci, že čím vyšší třídy spojitosti křivka je, tím více se segmenty blíží stejnému směru. [3][2][4]

Kromě parametrické třídy spojitosti se zároveň pro jednotlivé segmenty určuje geometrická třída spojitosti G^n. Křivka $q(t)$ patří do geometrické třídy spojitosti G^n, pokud má do řádu n (včetně) shodné vektory všech derivací, a to včetně jejich orientace ve všech bodech. Tedy segmenty $q_1(t)$ a $q_2(t)$ mají geometrickou třídu spojitosti:

- G^0 pokud je koncový bod segmentu $q_1(t)$ roven počátečnímu bodu segmentu $q_2(t)$,

- G^n pokud má n-tá derivace v koncovém bodu segmentu $q_1(t)$ shodný vektor včetně jeho orientace s vektorem n-té derivace v počátečním bodu segmentu $q_2(t)$.

V praktickém využití je zaručení geometrické třídy spojitosti jednodušší než zajištění parametrické třídy spojitosti. [4][1][2]

Mezi další požadované vlastnosti křivek patří, aby křivka procházela krajními body svého řídícího polygonu. Dále aby křivka ležela v konvexní obálce svých řídících bodů a aby změna jednoho z řídících bodů křivky ovlivnila pouze okolní část křivky, a ne křivku celou. Křivka by také měla splňovat invarianci k lineárním transformacím a projekcím. Tedy pokud například posuneme řídící body křivky a následně křivku vykreslíme nebo pokud posuneme každý bod křivky, budou výsledné křivky shodné. [1][2]

1.2 Plochy

Základním prvkem generování 3D objektů jsou pláty. Jak již bylo zmíněno v kapitole 1.1 využití explicitního a implicitního vyjádření není vhodné a pro reprezentaci plátů budeme využívat vyjádření parametrické. Oproti kapitole 1.1 budeme pro plochy uvažovat i třetí rozměr.

1.2.1 Vyjádření ploch

Analogicky k zápisu (1.3) můžeme plát $q(u, v)$ parametricky vyjádřit pomocí tří funkcí o dvou proměnných(1.16).

$$q(u, v) = [x(u, v), y(u, v), z(u, v)]; u, v \in [0, 1] \tag{1.16}$$

Normálový vektor, tedy vektor, který je kolmý na tečnou rovinu, můžeme v daném bodě vyjádřit pomocí součinu tečných vektorů $\vec{q'_u}(u_0, v_0)$ a $\vec{q'_v}(u_0, v_0)$, a to tvarem (1.17). [1]

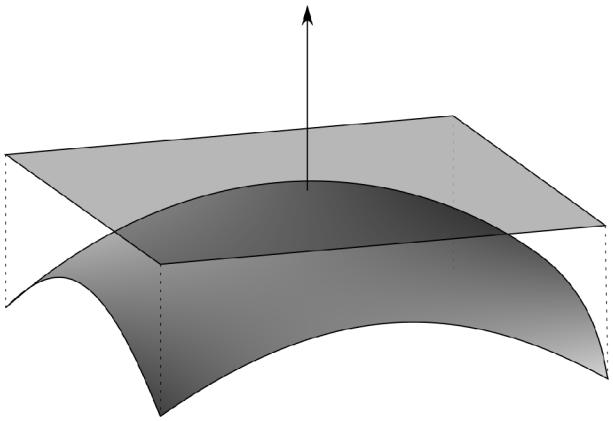
$$\vec{n'_u}(u_0, v_0) = \frac{\vec{q'_u}(u_0, v_0) \times \vec{q'_v}(u_0, v_0)}{|\vec{q'_u}(u_0, v_0) \times \vec{q'_v}(u_0, v_0)|} \tag{1.17}$$

Ukázku normálového vektoru plátu v jednom bodě můžeme vidět na obrázku 1.3. Normálové vektory jsou důležitou součástí následného generování objektů ve scéně. Jednotlivé tečné vektory pro oba body můžeme vyjádřit pomocí (1.18).

$$\vec{q'_u}(u_0, v_0) = \frac{\partial q(u_0, v_0)}{\partial u} = \left(\frac{\partial x(u_0, v_0)}{\partial u}, \frac{\partial y(u_0, v_0)}{\partial u} \frac{\partial z(u_0, v_0)}{\partial u} \right) \tag{1.18}$$

$$\vec{q'_v}(u_0, v_0) = \frac{\partial q(u_0, v_0)}{\partial v} = \left(\frac{\partial x(u_0, v_0)}{\partial v}, \frac{\partial y(u_0, v_0)}{\partial v} \frac{\partial z(u_0, v_0)}{\partial v} \right) \tag{1.19}$$

Bázové funkce u ploch v počítačové grafice reprezentujeme pomocí polynomů, stejně jako u křivek 1.1, nejčastěji konkrétně polynomem třetího stupně, tedy kubikou. Kubiky oproti polynomům vyššího stupně umožňují rychlejší výpočet a zároveň si zachovávají vhodné vlastnosti pro vzájemné napojování. Analogicky k napojování křivek můžeme komplikovanější generované plochy vytvářet napojováním jednodušších dílčích ploch neboli plátů. [2]



Obr. 1.3: Normálový vektor plátu v jednom bodě.

1.2.2 Vlastnosti ploch

Stejně jako u napojování segmentů křivek a určení jejich třídy spojitosti, můžeme u napojování plátů určit třídu spojitosti. Můžeme říci, že mezi dvěma pláty je napojení:

- C^0 pokud mají společnou stranu, která má třídu spojitosti alespoň C^0,
- C^1 pokud mají společnou stranu, která má třídu spojitosti alespoň C^1, a zároveň mají na této straně shodné příčné parciální derivace ve všech bodech,
- G^1 pokud mají společnou stranu, která má třídu spojitosti alespoň G^1, a zároveň mají parciální derivace ve směru napojení lineárně závislé na spojitě proměnlivém koeficientu k.

Stejně jako u křivek 1.1.3 patří mezi další požadované vlastnosti invariance k lineárním transformacím a projekcím, podmínky konvexní obálky, zachování změny křivky při změně řídícího bodu pouze v oblasti tohoto bodu. [4][2]

1.2.3 Beziérovy pláty

Mezi často používané postupy generování povrchu objektů patří Beziérovy pláty, a to hlavně pro jejich jednoduché a intuitivní modelování. Beziérovy pláty můžeme skládat z jednotlivých Beziérových křivek. Tedy pokud vezmeme zápis Beziérovy

křivky pomocí Bernsteinových polynomů s parametrem u a budeme jednotlivé řídící body chápat jako Beziérovy křivky s parametrem v, můžeme Beziérův plát stupně (m, n) zapsat jako (1.20).

$$q_{\text{Bez}}(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} P_{ij} B_{i,n}(u) B_{j,m}(v) \tag{1.20}$$

Tečné vektory Beziérovy křivky můžeme získat analogicky z rovnice (1.18), stejně tak normálový vektor můžeme získat jako součin tečných vektorů v bodě podle rovnice (1.17). Řídící body P_{00}, P_{01}, P_{10}, P_{11} jsou krajními body řídící sítě, respektive v těchto bodech jsou rohy Beziérova plátu. [4][2][1]

Stejně jako u Beziérových křivek jsou nejčastěji využívány Beziérovy kubiky, tak u Beziérových plátů jsou nejčastěji využívány Beziérovy bikubiky. Tedy Beziérovy pláty třetího stupně, respektive pláty, u kterých platí, že $(m, n) = (3, 3)$. Pokud výpočet délky zjednodušíme na výhodný maticový zápis (1.10), kde matice M bude znovu reprezentovat bázovou matici a matice G bude tvořena souřadnicemi jednotlivých řídících bodů, získáme zápis (1.21). [2]

$$x(u, v) = UMG_x M^T V^T = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} MG_x M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} \tag{1.21}$$

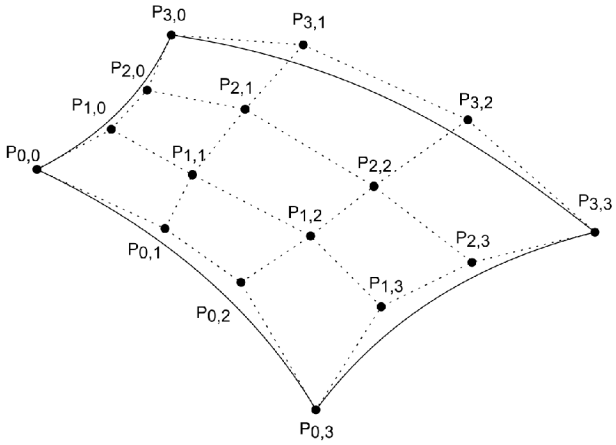
$$y(u, v) = UMG_y M^T V^T = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} MG_y M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} \tag{1.22}$$

$$z(u, v) = UMG_z z M^T V^T = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} MG_z M^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix} \tag{1.23}$$

Vlastnosti Bezierových plátů

Jak bylo zmíněno v části 1.2.2, jednou z požadovaných vlastností křivek je lokalita změn. Právě v této oblasti nemají Beziérovy pláty ideální chování. Pokud změníme polohu jednoho z řídících bodů, dojde ke změně celého plátu. Z tohoto důvodu se běžně pláty dělí na menší dílčí pláty a tím je dopad změny polohy řídícího bodu lokálnější, respektive má vliv pouze na dílčí plát. Následné určení třídy spojitosti dílčích plátů můžeme určit z popisu v části 1.2.2. [4][1]

Beziérovy pláty jsou určeny svým okrajem, který je tvořen Beziérovými křivkami, což můžeme vidět na obrázku 1.4. Toto tvrzení se dá ověřit, pokud si do



Obr. 1.4: Beziérův plát se sítí řídících bodů.

rovnice (1.20) dosadíme postupně za u a v hodnoty 0 a 1, tak dostaneme Beziérovy křivky, které tvoří okraj Beziérova plátu. V sekci 1.1.2 bylo mezi vlastnostmi Bernsteinových polynomů zmíněno, že jejich součet je roven jedné a zároveň jsou nezáporné. Z toho lze odvodit, že Beziérova plocha leží celá ve své konvexní obálce. Analogicky k postupnému hledání bodů na křivce 1.1.2 je možné k hledání bodů na ploše využít algoritmus de Casteljau. [4][1][2]

Polygonizace Beziérových plátů

Polygonizaci chápeme jako převod plátů na síť jednotlivých polygonů. Nejčastěji je plát převáděn na síť trojúhelníků, jelikož jsou planární. Polygonizace je výhodná pro další zpracovávání plátů, například jejich zobrazení. Pro polygonizaci Beziérových bikubických plátů se využívají nejčastěji dva postupy.

Prvním postupem je jednoduché dosazování do rovnice (1.20) za parametry u a v s pevně daným krokem. Tímto postupem dostaneme po každém kroku čtverec, který je určen rohy Beziérova plátu. Jelikož cílem je získat trojúhelníky, můžeme vzniklý čtverec rozdělit na dva potřebné trojúhelníky. Výhodou tohoto postupu je jednoduchost a nevýhodou je nerespektování zakřivení plochy. [1][2]

Druhý postup je rekurzivní a využívá algoritmus de Castlejau, který je popsán v části 1.1.2. Na vstupu vezmeme matici řídících bodů P. Následně pomocí algoritmu de Casteljau pro dělení křivek rozdělíme křivky určené řídícími body, které jsou v řádcích vstupní matice, v bodě $t = \frac{1}{2}$. Tím rozdělíme plát ve směru u na dva dílčí pláty Q_1, Q_2 a získáme jejich matice řídících bodů P_1 a P_2. Následně znovu aplikujeme stejný postup, ale pro křivky určené řídícími body ve sloupcích matic P_1 a P_2 a tím rozdělíme pláty Q_1 a Q_2 ve směru v. Tím získáme čtyři nové pláty a čtyři nové matice řídících bodů, které jsou aproximací původního plátu. Následně můžeme toto dělení rekurzivně opakovat dokud nedostaneme n ploch. [2]

1.3 Geometrické transformace

Mezi nejčastěji používané operace v počítačové grafice můžeme zařadit geometrické transformace. Geometrické transformace můžeme dělit například na lineární a nelineární. Mezi lineární transformace můžeme zařadit běžně používané operace jako je posunutí, změna měřítka, rotace, zkosení. Nelineární transformace můžeme využít při složitějších operacích s objekty. Pro potřebu využití transformací využíváme afinní prostor, který zobecňuje některé vlastnosti Euklidovského prostoru, respektive v afinním prostoru je zachována linearita a dělící poměr.

1.3.1 Homogenní souřadnice

Pro potřebu afinních transformací se využívají homogenní souřadnice. V homogenních souřadnicích reprezentujeme bod jako čtveřici, respektive ke třem kartézským souřadnicím přidáme váhu bodu w. Bod P můžeme tedy s využitím homogenních souřadnic zapsat jako (1.24).

$$P_{\text{kart}} = [x_{\text{kart}}, y_{\text{kart}}, z_{\text{kart}}]^T \Rightarrow P_{\text{hom}} = [x_{\text{hom}}, y_{\text{hom}}, z_{\text{hom}}, w_{\text{hom}}]^T = [x_{\text{kart}}, y_{\text{kart}}, z_{\text{kart}}, 1]^T \tag{1.24}$$

Pro homogenní souřadnice zároveň platí vztah (1.25), pro jeden bod můžeme mít tedy v homogenních souřadnicích nekonečně mnoho vyjádření.

$$x_{\text{kart}} = \frac{x_{\text{hom}}}{w_{\text{hom}}}; \quad y_{\text{kart}} = \frac{y_{\text{hom}}}{w_{\text{hom}}}; \quad z_{\text{kart}} = \frac{z_{\text{hom}}}{w_{\text{hom}}}; \quad w \neq 0 \tag{1.25}$$

Máme-li bod $P' = [x', y', z', w']^T$, který je výsledným bodem po transformaci původního bodu $P = [x, y, z, w]^T$, a matici A, která reprezentuje požadovanou transformaci, můžeme jednotlivé transformace zapsat tvarem (1.26).

$$P' = AP = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{1.26}$$

Velkou výhodou homogenních souřadnic je možnost vyjádření lineárních transformací jednou maticí. Zároveň můžeme jednotlivé afinní transformace skládat prostým násobením matic a inverzní transformace inverzní maticí. [2][5][6][7]

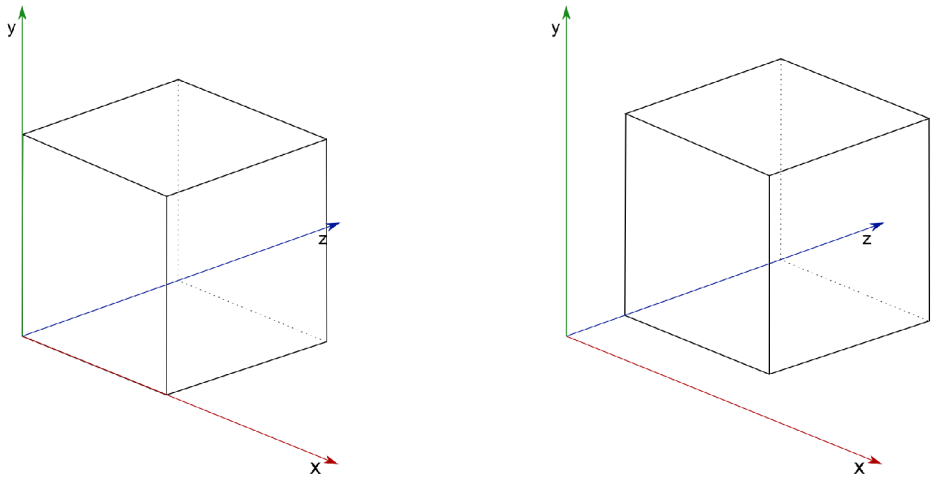
1.3.2 Posunutí

Posunutí, nebo také translace, je definováno vektorem posunutí $\vec{p}$, který je určen rozdílem jednotlivých souřadnic mezi původním a transformovaným bodem (1.27).

$$\vec{p} = [t_x, t_y, t_z] = [x'_{\text{kart}} - x_{\text{kart}}, y'_{\text{kart}} - y_{\text{kart}}, z'_{\text{kart}} - z_{\text{kart}}] \tag{1.27}$$

Ukázku translace pro vektor posunutí $\vec{p} = [0, 0, t_z]$, tedy posunutí po ose z, můžeme vidět na obrázku 1.5b. Transformační matici A a inverzní matici A^{-1} můžeme zapsat vztahem (1.28). [2][1][5][7][6]

$$A = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.28}$$



(a) Původní zobrazení objektu v počátku. (b) Výsledný transformovaný objekt.

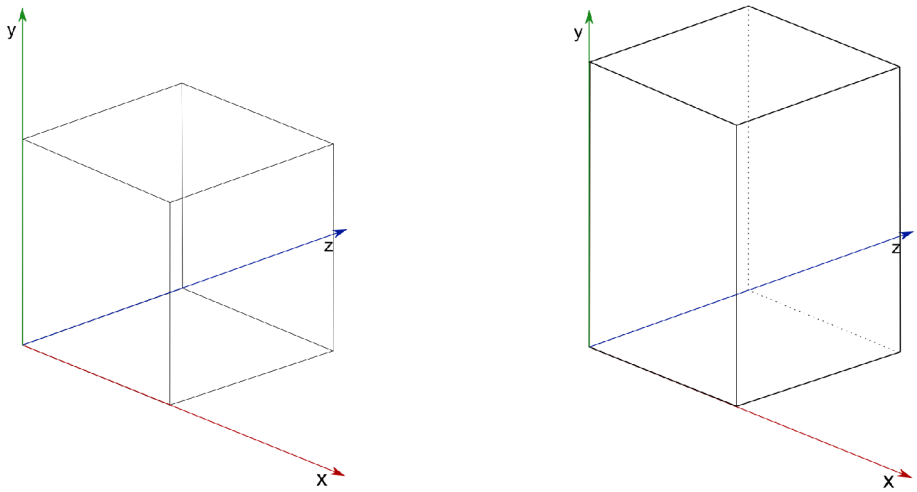
Obr. 1.5: Posunutí objektu podle osy z.

1.3.3 Změna měřítka

Změna měřítka, nebo také scaling, je určena změnou velikosti objektu ve směru jednotlivých os. Obecně můžeme transformační matici A, pro změnu měřítka, a její inverzní matici A^{-1} zapsat vztahem (1.29). [2][1][5][7][6]

$$A = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 & 0 \\ 0 & 0 & \frac{1}{s_z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.29}$$

Ukázku změny měřítka ve směru osy x, například pro hodnoty $s_x = 1,5$ a $s_y = s_z = 1$, můžeme vidět na obrázku 1.6b.



(a) Původní zobrazení objektu v počátku. (b) Výsledný transformovaný objekt.

Obr. 1.6: Změna měřítka objektu podle osy y.

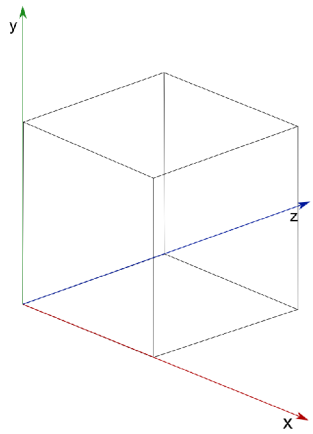
1.3.4 Rotace

Rotace ve 3D prostoru je určena jednou ze tří souřadných os, podle které objekt rotujeme, a úhlem α, který udává, do jaké míry objekt rotujeme. Pro každou souřadnou osu, respektive pro rotaci podle této osy, je definována transformační matice A a její inverzní matice A^{-1}. Tyto matice pro jednotlivé souřadné osy můžeme zapsat vzorcem (1.30). Ukázka rotace podle osy x je vidět na obrázku 1.7b. [2][1][5][7][6]

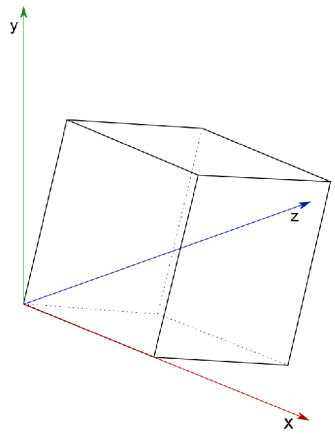
$$A_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_x^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.30}$$

$$A_y = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_y^{-1} = \begin{bmatrix} \cos(\beta) & 0 & -\sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.31}$$

$$A_z = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_z^{-1} = \begin{bmatrix} \cos(\gamma) & \sin(\gamma) & 0 & 0 \\ -\sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.32}$$



(a) Původní zobrazení objektu v počátku.



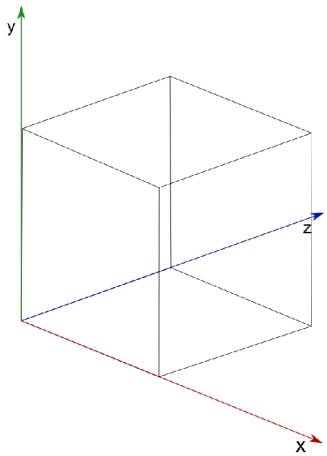
(b) Výsledný transformovaný objekt.

Obr. 1.7: Objekt rotovaný podle osy x.

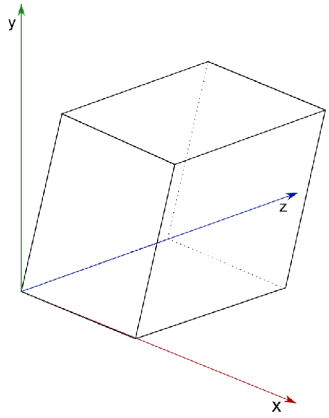
1.3.5 Zkosení

Zkosení, nebo také shearing, můžeme znovu provádět pro každou ze tří souřadných os jednotlivě. Obecný zápis pro zkosení na všech třech osách můžeme zapsat vztahem (1.33). Ukázka zkosení pro danou hodnotu sh_y^z je vidět na obrázku 1.8b. [2][1][5][7][6]

$$A = \begin{bmatrix} 1 & sh_x^y & sh_x^z & 0 \\ sh_y^x & 1 & sh_y^z & 0 \\ sh_z^x & sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 1 & -sh_x^y & -sh_x^z & 0 \\ -sh_y^x & 1 & -sh_y^z & 0 \\ -sh_z^x & -sh_z^y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.33}$$



(a) Původní zobrazení objektu v počátku.



(b) Výsledný transformovaný objekt.

Obr. 1.8: Zkosený objekt podle roviny yz.

1.3.6 Skládání transformací

Pro aplikování komplexnějších transformací můžeme jednotlivé transformace skládat. Skládání transformací je možné realizovat jednoduchým násobením matic. Násobení matic však není komutativní, například pokud objekt nejdřív posuneme a následně rotujeme, nejedná se o stejnou transformaci jako v případě rotování a následného posunutí. Pokud chceme postupně aplikovat transformační matice A_1 a poté A_2, tak výsledný vztah můžeme zapsat jako (1.34).

$$P' = A_2 A_1 P \tag{1.34}$$

Pokud afinní transformace zapíšeme obecně jako (1.35), tak můžeme výslednou transformační matici, vytvořenou násobením matic dílčích transformací, zapsat stejným způsobem.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.35}$$

Příkladem, kdy využíváme skládání transformací, může být rotace okolo obecné osy. V takovém případě skládáme rotace okolo jednotlivých os a dostaneme výslednou rotaci okolo obecné osy. [1][2]

1.4 Projekce

Pojem projekce, nebo také promítání, chápeme v počítačové grafice jako operaci, při které převádíme 3D objekty na jejich 2D reprezentace. Toho se využívá hlavně při zobrazování modelovaných 3D objektů, jelikož většina zobrazovacích zařízení, které v dnešní době využíváme, zobrazuje ve dvou rozměrech. V závislosti na využití existují různé druhy projekcí.

Jedním ze základních pojmů v projekci je průmětna. Průmětna je plocha ve 3D prostoru, na kterou dopadají promítací paprsky ze všech objektů ve scéně a tím vytváří obraz na této ploše neboli průmět. Průmětnu si můžeme představit jako obrazovku, na které se zobrazí modelovaná scéna. Zmiňované promítací paprsky jsou polopřímky vycházející z bodů, které jsou promítány. Jejich směr je závislý na zvolené promítací metodě. V případě projekce na rovinnou průmětnu existují dvě metody:

- paralelní projekce,
- perspektivní projekce.

1.4.1 Kamera

Pozice kamery je dána bodem, ze kterého pozorovatel sleduje scénu. Pro potřeby projekce využíváme dva souřadné systémy:

- World Coordinate System (dále jen WCS),
- Viewing Coordinate System (dále jen VCS).

WCS tvoří neměnné souřadnice, ve kterých definujeme zobrazovanou scénu. VCS je souřadný systém průmětny a mění se podle pozice a natočení kamery. VCS má čtyři homogenní souřadnice, nicméně při zobrazování na rovinnou průmětnu uvažujeme ve dvou rozměrech, tedy budou nás zajímat první dvě souřadnice. Pokud souřadnice pozice kamery v systému WCS zapíšeme jako $[x_{\text{cam}}, y_{\text{cam}}, z_{\text{cam}}, 1]$ a souřadnice bodu, na který kamera směřuje, jako $[x_{\text{obj}}, y_{\text{obj}}, z_{\text{obj}}, 1]$, tak můžeme směr pozorování kamery $\vec{L}$, který je rovnoběžný s hlavní optickou kamerou osy, určit z rozdílu těchto souřadnic a to zápisem (1.36). [1][2]

$$\vec{L} = \begin{bmatrix} L_x \\ L_y \\ L_z \\ 0 \end{bmatrix} = \begin{bmatrix} x_{\text{obj}} \\ y_{\text{obj}} \\ z_{\text{obj}} \\ 1 \end{bmatrix} - \begin{bmatrix} x_{\text{cam}} \\ y_{\text{cam}} \\ z_{\text{cam}} \\ 1 \end{bmatrix} \tag{1.36}$$

Následně můžeme z vektoru $\vec{L}$ určit jednotkový vektor $\vec{l}$ a budeme hledat matici projekce T_{cam}, která převede tento jednotkový vektor do tvaru $[0, 0, -1, 0]^T$, tedy bude pro VCS ve směru záporné poloosy z kolmý na průmětnu. Dále potřebujeme určit vektor $\vec{h}$, který určuje orientaci kamery. Znovu chceme pomocí matice T_{cam} převést vektor $\vec{h}$ na tvar $[0, 1, 0, 0]^T$, tedy bude kolmý na $\vec{l}$. Následně můžeme vektorovým součinem vektorů $\vec{l}$ a $\vec{h}$ vypočítat vektor $\vec{p}$, který je na tyto vektory kolmý, a po aplikování matice T_{cam} bude mít pro VCS tvar$[1, 0, 0, 0]^T$. Jednotlivé operace s vektory můžeme zapsat jako (1.37). [1][2]

$$T_{\text{cam}} \begin{bmatrix} l_x \\ l_y \\ l_z \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1 \\ 0 \end{bmatrix}; \qquad T_{\text{cam}} \begin{bmatrix} h_x \\ h_y \\ h_z \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}; \qquad T_{\text{cam}} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{1.37}$$

Pokud jednotlivé vektory sestavíme do matice a následně k této matici vytvoříme inverzní matici, získáme hledanou matici T_{cam} ve tvaru (1.38). Potřebnou matici pro následné promítání získáme, pokud do matice T_{cam} doplníme posun do bodu podle matice (1.28) a to konkrétně na souřadnice pozice kamery. [5][1][2]

$$T_{\text{cam}} = \begin{bmatrix} p_x & p_y & p_z & 0 \\ h_x & h_y & h_z & 0 \\ -l_x & -l_y & -l_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.38}$$

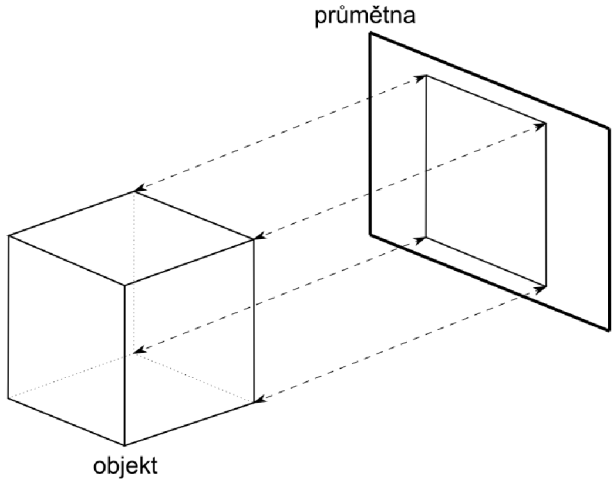
1.4.2 Paralelní projekce

První z metod promítání na rovinnou průmětnu je paralelní, nebo také rovnoběžná, projekce. Jak může být z názvu patrné, charakteristikou paralelní projekce je rovnoběžnost promítacích paprsků, která je zřejmá z obrázku 1.9. Jedná se poměrně jednoduchou a jednu z nejzákladnějších metod. Pokud budeme uvažovat za průmětnu některou z rovin:

- $x = x_0$, rovnoběžnou s rovinou yz,
- $y = y_0$, rovnoběžnou s rovinou xz,
- $z = z_0$, rovnoběžnou s rovinou xy,

tak můžeme jednotlivé matice paralelní projekce T_{par} zapsat jako (1.39). [5][7][1][2]

$$T_{\text{par}} = \begin{bmatrix} 0 & 0 & 0 & x_0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad T_{\text{par}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad T_{\text{par}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.39}$$



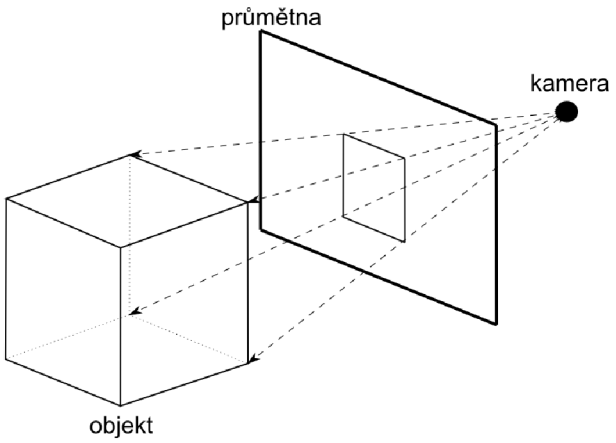
Obr. 1.9: Paralelní projekce krychle na průmětnu.

1.4.3 Perspektivní projekce

Druhou z metod promítání na rovinnou průmětnu je perspektivní, nebo také středová, projekce. Oproti předchozí metodě 1.4.2 nezachovává rovnoběžnost promítacích paprsků. Umožňuje nám však přirozenější zachycení prostorového vnímání na průmětnu, například díky proporcionálnímu měnění velikosti objektů při oddalování či přibližování kamery. Jedná se o častěji používanou metodu, jelikož se více přibližuje reálnému lidskému vidění. Ukázku perspektivní projekce můžeme vidět na obrázku 1.10. Pokud budeme uvažovat zobrazení na průmětnu, jenž je rovina

xy, tedy je kolmá na osu z, a souřadnice pozice kamery pro WCS budou $[0, 0, z_0]$, můžeme matici projekce T_{per}, zapsat vztahem (1.40). [5][7][1][2]

$$T_{\text{per}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{z_0} & 1 \end{bmatrix} \tag{1.40}$$



Obr. 1.10: Perspektivní projekce krychle na průmětnu.

2 Návrh a funkcionalita aplikací

Tato kapitola se věnuje samotnému návrhu a funkční stránce jednotlivých aplikací. Je požadováno, aby aplikace sloužily jako doplněk k výuce počítačové grafiky a ne pouze jako zobrazení dané problematiky. Například jak je popsáno v části 2.2.3, druhá aplikace „Beziérova plocha“ nezobrazuje pouze Beziérovu plochu, ale také plochy grafu pro jednotlivé osy a umožňuje zobrazení bodu na ploše. Návrhy aplikací byly vytvořeny ve spolupráci s vedoucím práce a postupně upravovány po dílčích konzultacích tak, aby správně vyhovovaly potřebám výuky.

Všechny aplikace obsahují krátký text, který vysvětluje, jak aplikaci používat, respektive jsou zde popsány všechny jednotlivé prvky ovládání. Samotná teorie dané problematiky, kterou se aplikace zabývá, není v aplikacích popsána, jelikož jak bylo zmíněno v předchozím odstavci, aplikace slouží jako doplněk k výuce. Tedy teorie ze které aplikace vycházejí je popsána v této diplomové práci, případně ve skriptech konkrétního vyučovaného předmětu. Následně aplikace obsahují samotné grafické řešení, tedy zobrazení dané problematiky pomocí frameworku THREE.js, které je napojeno na ovládací prvky.

2.1 1. aplikace – Transformace objektů

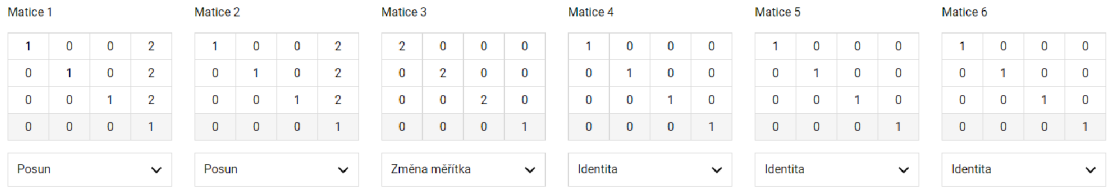
Aplikace je zaměřena na využití geometrických, respektive lineárních transformací objektů ve 3D scéně. Implementovány jsou čtyři základní lineární transformace – posunutí, rotace, zkosení, změna měřítka, kde pro každou transformaci je zvolena fixní výchozí hodnota. Uživatel také může do matice vyplnit lineární transformaci s vlastními hodnotami. Transformace se do matic zadávají jednotlivě a jsou následně vynásobeny do finální matice, která je aplikována na objekt ve scéně.

2.1.1 Zadávání transformací

Jak je možné vidět na obrázku 2.1, pro zadání jednotlivých transformací je připraveno šest matic. U každé matice je umístěno rozbalovací okno, ve kterém je možné zvolit jednu ze zmiňovaných základních transformací a tato transformace je následně s fixními hodnotami předvyplněna do dané matice. V rozbalovacím okně je možné také vybrat matici identity a to pokud uživatel nechce do matice vyplnit žádnou transformaci, tedy nechce aby tato matice měla vliv na finální matici.

Až na spodní řádky matic může uživatel jednotlivé hodnoty libovolně upravovat a tím definovat vlastní transformace. Spodní řádky matic jsou deaktivovány, jelikož pro účely aplikace stačí definovat pouze lineární transformace, bez nutnosti využití váhy homogenních souřadnic. Do matic je možné zadávat hodnoty jako desetinná

čísla. V případě rotace je nutné zadat hodnotu reprezentující výsledek goniometrické funkce.



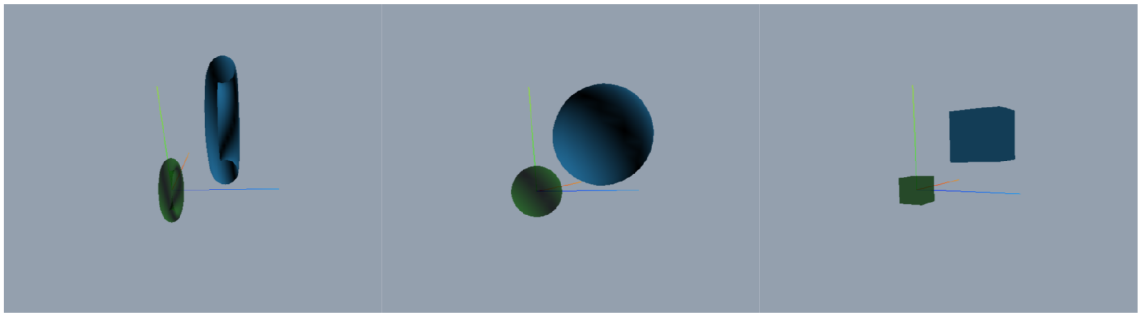
Obr. 2.1: Ukázka zadávání hodnot do matic.

2.1.2 Vykreslená scéna

Ve spodní části aplikace se nachází canvas, ve kterém je vykreslena vlastní scéna s objektem, na který jsou aplikovány vybrané transformace. V aplikaci jsou připraveny tři různé tvary objektu, konkrétně se jedná o krychli, kouli a torus. Tyto objekty mají fixní velikost a uživatel nemůže přidávat vlastní objekty.

Na obrázku 2.2 je možné vidět ukázku tří scén pro jednotlivé tvary objektů. Ve scéně jsou vždy zobrazeny dva objekty, kde modrý objekt reprezentuje stav po aplikaci transformace a zelený objekt reprezentuje objekt před provedením transformace. Na zmiňované ukázce byl objekt před provedením transformace v počátku souřadnic a v současném stavu jsou na něj aplikovány transformace z obrázku 2.1. Oba objekty jsou průhledné a v případě překryvu objektů jsou oba viditelné.

Ve scéně je zároveň možné pohybovat kamerou. Pravé tlačítko umožňuje posun bodu, na který kamera směřuje, a levé tlačítko umožňuje rotaci kolem tohoto bodu, kolečko myši ovládá přibližování kamery. Zároveň pro lepší orientaci v prostoru je vykreslena základní osa souřadnic a ve scéně je bodový zdroj světla. Scéna není vykreslována responzivně, a tedy pokud je otevřena konzole nebo je zmenšeno okno prohlížeče, je nutné stránku znovu načíst.



Obr. 2.2: Ukázka scény a jednotlivých tvarů objektů.

2.1.3 Finální matice a hlavní ovládání

Mezi hlavními prvky ovládání aplikace je umístěno rozbalovací okno pro volbu objektu zmiňované v části 2.1.2. Dále je zde umístěno zobrazení finální matice, která je vytvořena vynásobením jednotlivých matic transformací. Tuto matici nemůže uživatel přímo upravit, jde upravovat pouze přes dílčí matice z části 2.1.1.

Pod maticí identity je umístěno tlačítko „Restart scény“, které smaže současný stav scény, vytvoří nové objekty ve zvoleném tvaru a umístí je do počátku souřadnic. Tlačítko uživatel využije, pokud chce změnit tvar objektu nebo pokud chce restartovat současný objekt do počátečního nastavení. Při restartování scény se hodnoty v maticích transformací nezmění.

Dále následuje tlačítko „Transformuj“, které aplikuje vybrané transformace na objekt ve scéně. Dílčí matice z části 2.1.1 jsou vynásobeny zleva doprava, respektive na první matici je aplikována druhá matice a následně všechny další. Vynásobená matice je vyplněna do „finální matice“ a tato matice je aplikována na matici objektu ve scéně.

Obr. 2.3: Ukázka finální matice a výběru tvaru.

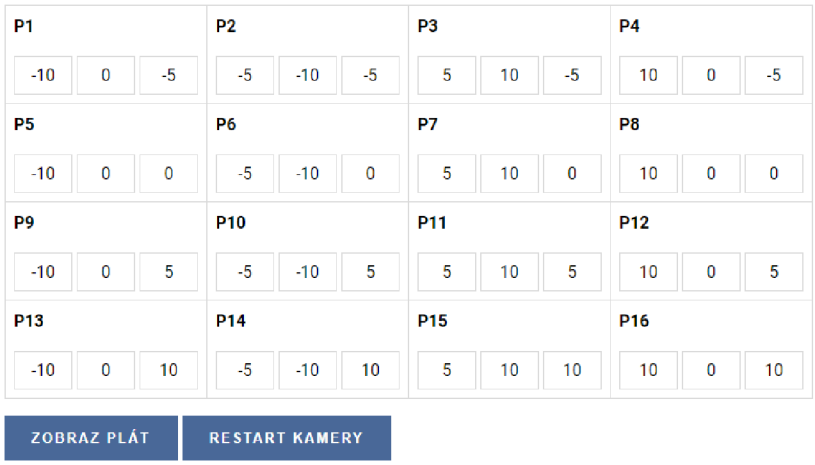
2.2 2. aplikace – Beziérův plát

Tato aplikace se věnuje zobrazení Beziérových plátů ve 3D prostoru. Beziérův plát je možné zadat pomocí šestnácti řídících bodů a plocha je následně vykreslena s pomocnou osou pro jednotlivé souřadnice. Zobrazen není pouze samotný plát, ale také

další tři pláty pro každou osu, které tvoří výsledný Beziérův plát. V aplikaci je možné vybrat bod na plátu podle zadání parametrů u a v. Aplikace obsahuje vzorové Beziérovy pláty a jeden z těchto plátů je automaticky zobrazen při načtení aplikace.

2.2.1 Zadávání plochy

Na obrázku 2.4 je možné vidět matici pro zadávání řídících bodů. Řídící body jsou označeny P1 až P16. Pro každý bod je nutné zadat všechny tři souřadnice, kde každá souřadnice má vlastní okno pro lepší přehlednost. Souřadnice jsou zadávány jako desetinná čísla. Pod maticí se nachází tlačítko „Zobraz plát“, které načte z matice všechny řídící body a pomocí těchto bodů vykreslí plát. Pokud některý z bodů není zadán, plát není vykreslen a aplikace vypíše varovnou hlášku s označením, které body nejsou vyplněny. Po znovu načtení plátu zůstane kamera v původní poloze, tedy uživatel může změnit plát a pohled zůstane stejný. Pro restartování kamery do výchozí pozice slouží tlačítko „Restart Kamery“.



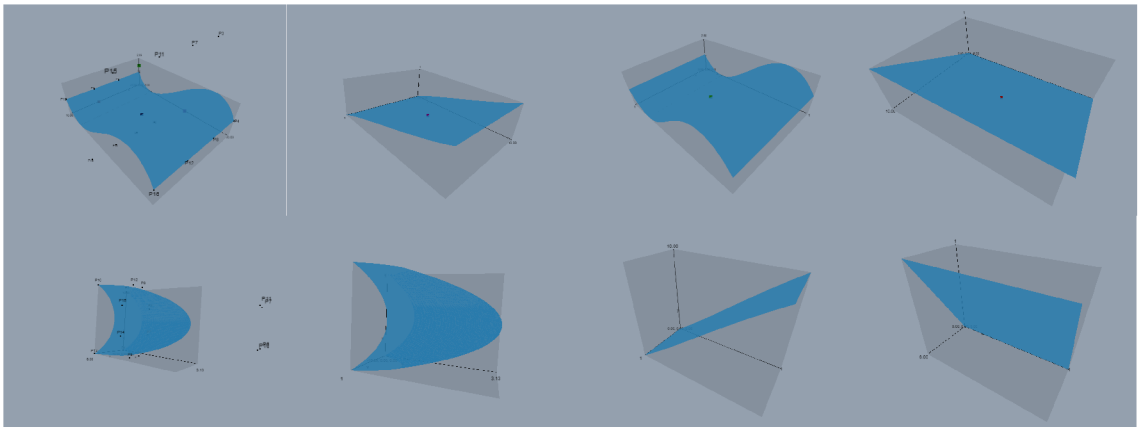
Obr. 2.4: Zadávání řídících bodů plátu.

2.2.2 Vykreslené scény

Jak bylo zmíněno, v aplikaci jsou vykresleny čtyři pláty. Každý z plátů je vykreslen do vlastního canvasu ve spodní části aplikace. V každé scéně je plát zobrazen do grafu, respektive u plátu jsou vykresleny všechny tři osy s popisem minimální a maximální hodnoty na ose.

V první scéně je zobrazen samotný definovaný plát, který je vypočten využitím zadaných řídících bodů podle vzorce 1.21. Jsou zde zároveň zobrazeny i jednotlivé řídící body a jejich označení. V následujících třech scénách je vždy zobrazena

funkce parametrů u a v pro danou osu. Hodnoty na zobrazované ose jsou zachovány a hodnoty na ostatních dvou osách jsou nahrazeny parametry u a v, které nabývají hodnoty 0 až 1. Následně je znovu využit vzorec 1.21 pro vypočtení bodů plátu. Na obrázku 2.5 je ukázka všech scén pro oba předdefinované pláty.



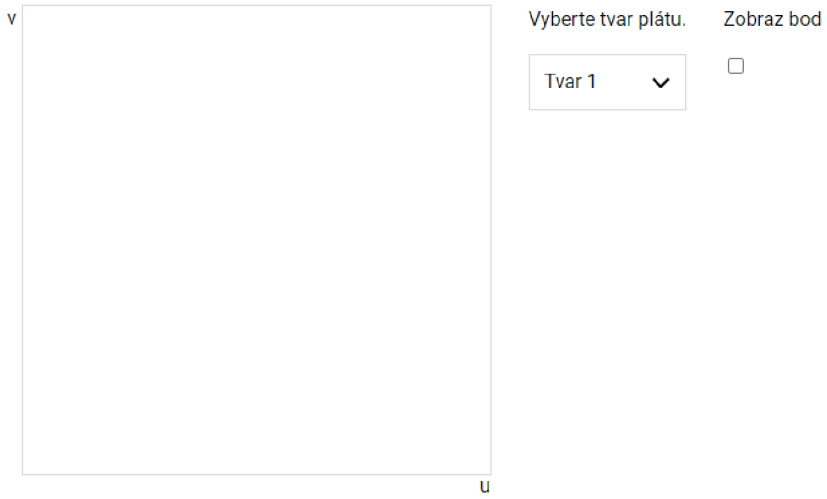
Obr. 2.5: Ukázka vykreslování scén.

Kamera ve scénách je zaměřena na střed plátu a její vzdálenost je přepočítávána podle velikosti plátu, nicméně v některých případech může mít plát natolik specifický tvar, že kamera může být významně oddálena nebo naopak přiblížena. Kamerou je zároveň možné pohybovat, pravé tlačítko umožňuje posun bodu, na který kamera směřuje, a levé tlačítko umožňuje rotaci kolem tohoto bodu, kolečko myši přibližuje a oddaluje kameru. Scény nejsou vykreslovány responzivně, a tedy pokud je otevřena konzole nebo je zmenšeno okno prohlížeče, je nutné stránku znovu načíst. V každé scéně je ambientní světlo, tedy nasvícení všech objektů je stejné a objekty nevrhají stíny.

2.2.3 Zobrazení bodu

Aplikace umožňuje ve scénách zobrazit bod plátu. Tento bod je definován parametry u a v, které nabývají hodnot 0 až 1. Hodnoty parametrů jsou zadávány přes čtvercovou plochu, kterou je možné vidět na obrázku 2.6. Po kliknutí do čtvercového pole jsou načteny hodnoty pro parametry a vypsány pod pole. Levý dolní roh reprezentuje hodnotu 0 a pravý horní roh hodnotu 1 pro oba parametry. Pro zobrazení bodu je nutné zaškrtnout pole „Zobraz bod“. Bod je zobrazen na všech plochách, zároveň na hlavní scéně jsou na osách přidány body, které reprezentují body z ostatních scén, a jsou vyznačeny stejnou barvou.

Vedle ovládání zobrazení bodu se nachází rozbalovací okno pro výběr tvaru plátu. Po výběru tvaru jsou nastaveny všechny řídící body daného plátu, ale samotný plát je zobrazen až po kliknutí na tlačítko „Zobraz plát“. Pokud chce uživatel vymazat



Obr. 2.6: Zobrazování bodu plátu.

hodnoty z matice řídících bodů, je možné ve výběru tvarů vybrat první možnost, která hodnoty vymaže.

2.3 3. aplikace – Kamera ve scéně

Cílem poslední aplikace je ukázka fungování kamery ve scéně při perspektivní projekci. Aplikace zobrazuje modelovanou scénu z pohledu třetí osoby a zároveň z pohledu reálné kamery v této scéně. Pro lepší přehlednost je zobrazena ve scéně i průmětna a zorné pole kamery včetně bodu, na který kamera směřuje. Z důvodu návaznosti na vyučovanou teorii jsou v aplikaci zobrazeny matice projekce.

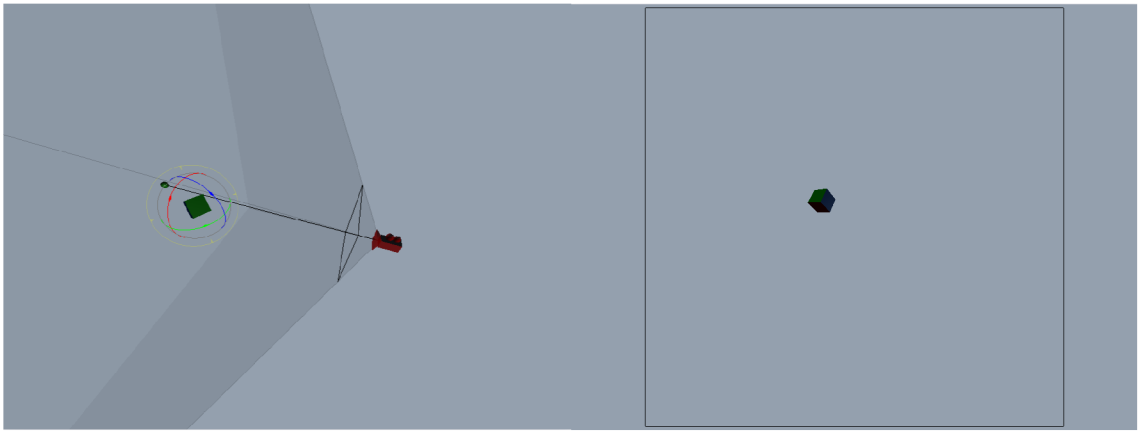
V následujícím textu je využíváno pojmu scéna a kamera v různých kontextech. Pojem scéna je využit ve třech různých případech. V prvních dvou případech je pojem scéna využíván pro vykreslované scény v aplikaci. První levá scéna je označována jako hlavní scéna, druhá pravá scéna je označována jako scéna kamery. Pojmem modelovaná scéna je myšlena uvažovaná scéna, která je v aplikaci reálně zobrazována pomocí vykreslovaných scén.

Pojem „kamera“ je používán ve dvou případech. Prvním případem je kamera ve scéně, je tím myšlena kamera určující pohled. Tuto kameru, respektive pohled kamery, je možné měnit pomocí myši. V druhém případě je používán pojem „objekt kamery“. Tím je myšlen model kamery, respektive červený objekt ve scéně, který reprezentuje pozici kamery v modelované scéně.

2.3.1 Zobrazení scén

Ve spodní části aplikace se nacházejí dva canvasy se scénami. Na obrázku 2.7 je ukázka těchto scén, přitom levá hlavní scéna reprezentuje zobrazení modelované scény z pohledu třetí osoby a pravá scéna ukazuje reálný pohled kamery z předchozí scény.

Hlavní scéna obsahuje červený objekt kamery, který reprezentuje pozici kamery v modelované scéně, a průmětnu umístěnou před touto kamerou. Objekt kamery je spojen optickou osou se zeleným kulovým objektem, což je cílový bod, na který kamera směřuje. Z objektu kamery skrze průmětnu vychází šedé průhledné plochy reprezentující zorné pole kamery, jinak známé jako FOV (field of view). Posledním objektem ve scéně je samotný objekt modelované scény, pro ukázku je jako tvar zvolena krychle s různými barvami stěn pro lepší přehlednost natočení objektu. Uživatel nemůže do scény přidat další objekty nebo měnit tvar zobrazovaného objektu.



Obr. 2.7: Ukázka obou scén.

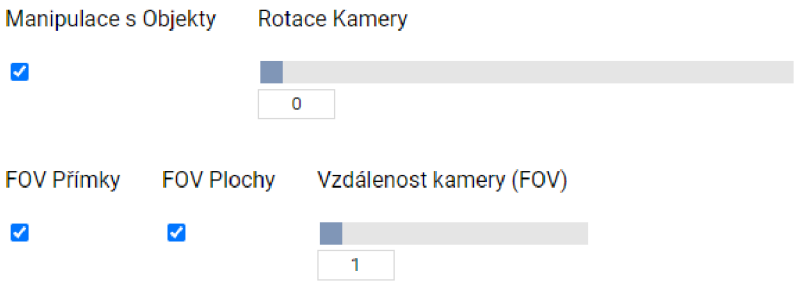
Scéna kamery slouží pouze pro zobrazení samotného pohledu kamery, tedy obsahuje průmětnu skrze kterou kamera směřuje, a zobrazovaný objekt v modelované scéně. Ovládání objektů ve scénách je napojeno na hlavní scénu a prováděné změny jsou automaticky promítnuty do scény kamery. Hlavní scéna obsahuje základní ovládání kamery, tedy kamera se dá otáčet levým tlačítkem myši, pravým tlačítkem je možné posunout bod, na který kamera směřuje, kolečko myši přibližuje a oddaluje kameru. Po aktivování funkcionality pro manipulaci s objekty popsané v části 2.3.2 je ovládání kamery popsané v předchozí větě deaktivováno a myš slouží pro posun objektů. Objekt kamery a cílový objekt kamery je možné pomocí levého tlačítka myši přetáhnout na jiné místo v prostoru.

Zobrazovaný objekt krychle můžeme kromě posunu také rotovat nebo měnit jeho velikost. Aktivování jednotlivých transformací nad objektem je ovládáno tlačítky, konkrétně tlačítkem „T“ pro posun, „R“ pro rotaci a „S“ pro změnu měřítka.

Při aktivování jedné z transformací se okolo objektu zobrazí barevné křivky pro jednotlivé osy, které lze ovládat levým tlačítkem myši, a tedy objekt je možné transformovat pouze podle jedné z os. Zároveň pokud uživatel drží levé tlačítko přímo na samotném objektu, může ho transformovat po všech osách. Ukázka aktivované transformace rotace je vidět na obrázku 2.7.

2.3.2 Ovládání scén

Ovládání samotné aktivace je umístěno v pravém horním rohu aplikace a je vidět na obrázku 2.8. Hlavním prvkem je zaškrtávací pole „Manipulace s objekty“, které je možné ovládat také klávesou „Q“. Při jeho zaškrtnutí je deaktivováno ovládání kamery v hlavní scéně a aktivuje se ovládání objektů popsané v části 2.3.1. Pod tímto polem se nachází další dvě zaškrtávací pole „FOV Přímky“ a „FOV Plochy“, které udávají, jestli se mají v hlavní scéně zobrazit pomocné přímky a plochy ukazující zorné pole kamery.



Obr. 2.8: Ovládání objektů ve scéně.

Na zmiňované pole navazují dva posuvníky „Rotace kamery“ a „Vzdálenost kamery“. První posuvník slouží pro rotaci kamery a průmětny kolem optické osy, kde hodnota je reprezentována ve stupních. Druhý posuvník slouží k nastavení vzdálenosti kamery od průmětny a tím nepřímo nastavuje zorné pole kamery. Vzdálenost je udávána v celých číslech v rozsahu 1 až 10, kde při rostoucích hodnotách se kamera oddaluje od průmětny. U obou posuvníků je umístěno pole, které je s posuvníkem provázáno, tedy reflektuje hodnotu na posuvníku a zároveň je možné skrze toto pole zadat na posuvník přesnou hodnotu.

2.3.3 Zobrazení matic

V levé horní části aplikace se nacházejí tři matice, které uživatel nemůže upravovat. Matice kamery vychází ze vztahu 1.38 a realizuje posunutí kamery do bodu $[0, 0, 0, 1]$. Matice projekce je definována vztahem 1.40 a umožňuje projekci bodu

na průmětnu. Pokud matici projekce vynásobíme maticí kamery, dostaneme finální matici. Tato matice nám umožňuje převést bod ze souřadnic WCS do souřadnic průmětny VCS.

Matice Kamery

0	0	1	0
0	1	0	0
1	0	0	-20
0	0	0	1

Matice Projekce

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0.3225	0

Finální Matice

0	0	1	0
0	1	0	0
1	0	0	-20
0.3225	0	0	-6.4516

Obr. 2.9: Výsledné matice pro převod na průmětnu.

3 Technologie a implementace aplikací

Tato kapitola se věnuje konkrétní implementaci jednotlivých aplikací. V krátkosti popisuje různé obecně využívané technologie pro grafické aplikace v jazyce JavaScript. Dále se věnuje ostatním technologiím, které jsou využity při implementaci zmiňovaných aplikací.

JavaScript je široce rozšířený, objektově orientovaný, multiplatformní, skriptovací jazyk. Stejně jako při implementaci této práce je nejčastěji využíván jako součást HTML kódu webové stránky, tedy je spuštěn na straně klienta. Pomocí HTML je definována samotná podoba stránek a jejich prvky [8] a jazyk CSS slouží ke stylování těchto stránek [9].

Samotná funkcionalita aplikace je napsána ve zmiňovaném jazyce JavaScript. Tento jazyk je v současné době standardem pro vytváření webových aplikací, které běží na straně klienta. Zároveň obsahuje pokročilé knihovny a rozšíření pro tvorbu 3D aplikací, proto byl zvolen i pro implementaci aplikací této práce.

3.1 Grafické aplikace v jazyce JavaScript

Pro tvorbu grafických aplikací v jazyce JavaScript existuje velká řada různých knihoven a API. V následující části jsou stručně popsány tři porovnávané možnosti pro tvorbu aplikací v této práci.

3.1.1 WebGL

WebGL je základní webové API pro tvorbu 2D a 3D grafiky bez nutnosti využití dalších zásuvných modulů [10]. Skládá se z řídícího kódu, který je napsaný v jazyce JavaScript s využitím canvas elementů jazyka HTML, a kódu shaderu založeném na OpenGL ES [11], který pro výpočet využívá grafickou kartu. WebGL je vytvořeno a udržováno společností Khronos Group. První verze byla vydána v roce 2011 a od počátku je WebGL podporováno všemi běžně využívanými prohlížeči.

3.1.2 p5.js

Jedná se o open-source grafickou knihovnu jazyka JavaScript. Tato knihovna vznikla s cílem zjednodušení vytváření grafických aplikací v jazyce JavaScript [12]. Jedná se o jednu z nejpopulárnějších knihoven tohoto jazyka pro vytváření 2D grafiky, nicméně umožňuje i práci s 3D grafickými návrhy pomocí WebGL API. Velkou výhodou této knihovny je její jednoduchost a zároveň podrobná dokumentace se silnou aktivní komunitou vývojářů.

3.1.3 Three.js

Three.js je volně dostupný framework jazyka JavaScript, který je založený na WebGL API [13]. Jde o jeden z nejpopulárnějších frameworků pro tvorbu 3D webové grafiky a také byl zvolen pro tvorbu aplikací, které jsou součástí této práce. Three.js totiž umožňuje využití potenciálu WebGL, zároveň však výrazně usnadňuje práci s tímto API. Oproti p5.js je Three.js primárně zaměřené na tvorbu 3D grafiky. Je tedy lépe optimalizované pro účely 3D grafiky a zároveň jeho knihovny obsahují objekty a metody definované přímo pro práci ve 3D, což bylo hlavním důvodem, proč byl pro implementaci vybrán tento framework.

3.2 Ostatní využité technologie

Mimo popisovaný jazyk JavaScript a framework THREE.js bylo pro tvorbu aplikací využito několik dalších technologií. Zdrojový kód aplikací byl vytvořen v programovacím prostředí Webstorm ulehčující tvorbu kódu. Pro správu a lepší přenositelnost aplikací je použit správce JavaScript balíčků NPM. Jednotlivé aplikace vycházejí ze stejné šablony, která je definována pomocí jazyka Nunjucks.

3.2.1 NPM

NPM (Node Package Manager) je správce balíčků původně vyvinutý pro Node.js aplikace [14]. V současnosti je využíván i jako správce JavaScript balíčků. Pomocí klienta v příkazové řádce umožňuje stahování a instalaci balíčků, jako například THREE.js, z veřejné databáze. NPM je napsáno v jazyce JavaScript a je instalováno společně s Node.js. S rozvíjejícími se technologiemi je namísto NPM často využíván novější správce balíčků Yarn. V rámci této práce je využito NPM pro instalaci THREE.js a dalších knihoven do projektu aplikace. Následně je projekt pomocí NPM zkompilován do finálních souborů, které lze nahrát přímo na server.

3.2.2 Nunjucks

Nunjucks je jazyk využívaný pro tvorbu HTML šablon, respektive se jedná o preprocesor HTML [15]. Šablony jsou psány pomocí syntaxe Nunjucks a HTML, následně je šablona při kompilaci převedena do čistého HTML. Hlavní výhodou jazyka je možnost využití pokročilejší syntaxe, jelikož v HTML není možné využívat například podmínky nebo cykly. V těchto aplikacích je toho využito například pro snadnější definování matic v aplikaci. Matice je definována pomocí cyklu, který v každém kroku vytvoří HTML pole, a není nutné explicitně definovat všech šestnáct polí matice.

3.2.3 Webstorm

Jedná se o jedno z nejlépe hodnocených programovacích prostředí využívané pro tvorbu webových aplikací [16]. Webstorm nabízí základní nástroje jako je našeptávání kódu, přímé zobrazování v prohlížeči nebo zvýraznění syntaxe. Také však nabízí spoustu pokročilejších nástrojů jako je například refactoring kódu, tvorba unit testů nebo možnost integrace dalších nástrojů z obsáhlé veřejné databáze. Zároveň obsahuje příkazovou řádku využívanou například pro spravování projektu pomocí NPM nebo debuggování.

3.3 Implementace

Všechny aplikace jsou naprogramovány pro kompatibilitu s běžně používanými prohlížeči – Google Chrome, Mozilla Firefox, Opera a Microsoft Edge. Kompatibilita s prohlížečem Internet Explorer není zajištěna, jelikož u tohoto prohlížeče je ukončena podpora, a dochází k problémům v kompatibilitě s novějšími technologiemi. Aplikace jsou primárně implementovány pro počítače, jelikož na malých obrazovkách mobilních telefonů je vykreslování a ovládání 3D scén problematické.

Aplikace jsou psané imperativním přístupem, tedy zdrojový kód je rozdělen do funkcí, které jsou vzájemně volány v průběhu programu. Objektově orientovaný přístup nebyl zvolen z důvodu nižšího rozsahu aplikací a komplexnosti přenesení řešené problematiky do kontextu objektů. Některé z méně komplexních funkcí jsou psané formou pure funkce, tedy výstup funkce odpovídá vstupu a nemá žádné vedlejší efekty.

Každá aplikace obsahuje dva HTML soubory, jeden s obecnou kostrou a popisem aplikace, druhý soubor obsahuje samotné prvky aplikace, na které jsou navázány JavaScript funkce. Zdrojový JavaScript kód je také rozdělen do dvou souborů, jeden soubor obsahuje přípravu samotné scény a je téměř identický pro všechny aplikace. Druhý soubor obsahuje hlavní funkcionalitu dané aplikace. U druhé aplikace je definováno klikací okno pro zobrazení bodu jako samostatná třída. Jedná se o jediný případ implementace, kdy je využito objektově orientovaného programování, protože se jednalo o samostatnou komponentu, která není nijak navázána na samotný kód programu a jeho funkce.

V textu je využíváno pojmu funkce pro části kódu, které jsou mnou implementované. Pokud je zmiňována nějaká metoda, je tím myšlena metoda THREE.js objektu. Důvodem je to, že v aplikacích jsem implementoval pouze jednu vlastní třídu a její metody nejsou zmiňovány. Pojem objekt může reprezentovat THREE.js objekt, nebo je tím myšlen zobrazovaný objekt ve scéně, podle probíraného kontextu.

Některé prvky funkcionality jsou stejné nebo velice podobné pro všechny aplikace. Jedná se například o ovládání kamery nebo vytváření scény. Tyto prvky jsou popsány společně pro všechny aplikace.

3.3.1 Ovládání kamery

Možnost manipulace s pohledem kamery je důležitým prvkem každé 3D aplikace, a proto je implementováno i ve zmiňovaných aplikacích. Ovládání kamery je v první a druhé aplikace identické. Ve třetí aplikaci je základní ovládání kamery provázáno s ovládáním objektů, které jsou popsány v části 3.3.5.

Zdrojové kódy THREE.js neobsahují přímo třídu pro manipulaci s kamerou, nicméně byla vytvořena třída *OrbitControls*, která je součástí příkladů zdrojových kódů v knihovně THREE.js. Tuto třídu je nutné explicitně importovat do zdrojového kódu a je běžně využívána různými grafickými aplikacemi. Instance této třídy je navázána na objekt kamery a DOM element, který je využíván pro renderování. DOM element reprezentuje základní prvky stránky v paměti a odkazuje na samotné HTML prvky.

Někteří vývojáři preferují vlastní implementaci ovládání kamery, a to hlavně v případech, kdy je požadována specifická operace s kamerou. Pro potřeby aplikací této práce je *OrbitControls* naprosto dostačující, a je tedy využit místo vlastní implementace.

3.3.2 Vytvoření scény

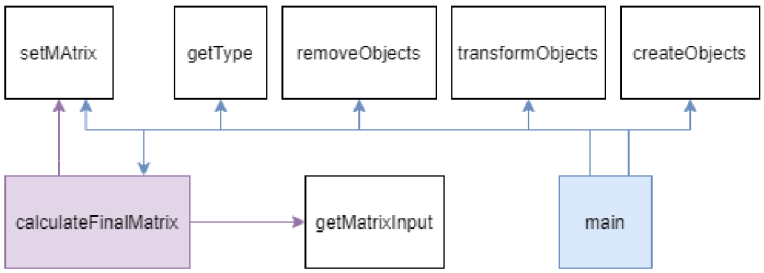
Základní prvkem pro vykreslování grafické scény ve THREE.js je objekt *scene*. Na tento objekt jsou následně navázány všechny další vytvářené objekty. Inicializace scény je pro všechny aplikace převážně stejná.

Inicializace scény je provedena voláním funkce *initiateScene* a jako parametr je předán canvas, do kterého má být scéna vykreslena. Volaná funkce nejdříve vytvoří instanci třídy *scene*, následně zavolá funkce pro vytvoření objektů kamery a rendereru. Pro objekt *camera* je nastavena pozice, zorné pole a oddálení. V objektu *renderer* je nastavena velikost a barva pozadí scény.

Po inicializaci samotné scény a dílčích objektů je přidáno osvětlení scény. Osvětlení se mezi jednotlivými aplikacemi mírně liší, v druhé aplikace je využito ambientní osvětlení, v ostatních aplikacích jsou bodové zdroje světla. V objektu osvětlení je nastavena pozice a barva. Poslední prvkem inicializace scény je přidání ovládání kamery popsané v části 3.3.1.

3.3.3 1. aplikace – Transformace objektů

Aplikace slouží pro ukázku transformací objektů ve 3D scéně. Při načtení aplikace je volána hlavní funkce. Tato funkce nejdříve načte DOM elementy jednotlivých prvků ovládání do proměnných, jedná se o všechna rozbalovací okna a tlačítka v aplikaci. Následně jsou na tyto elementy napojeny listenery. Listenery sledují, jestli došlo na požadovaném elementu ke změně, a případně volají požadované funkce. Na obrázku 3.1 je možné vidět diagram volání funkcí této aplikace.



Obr. 3.1: Diagram aplikace pro transformaci objektů.

Dalším úkolem této funkce je inicializovat scénu. Do scény je následně přidán objekt *axesHelper*, který zobrazuje pomocnou osu souřadnic. Posledním krokem je volání funkce *createObjects*. Tato funkce vytvoří oba zobrazované objekty v požadovaném tvaru a naváže je na objekt scény. Předvoleným tvarem objektu je krychle.

Volba transformace

Listener navázaný na jednotlivá rozbalovací okna, umístěná pod maticemi transformací, je aktivován při zvolení transformace. Při aktivaci je zavolána funkce *getType* a jako parametr je předáno číslo vybrané transformace. Funkce vytvoří THREE.js objekt *matrix4* a zavolá metodu, která podle zadaných hodnot nastaví objekt jako matici příslušné transformace. Například pro matici posunu je volána metoda *makeTranslation(2,2,2)*, jenž nastaví objekt jako matici translace po všech osách o hodnotu dva.

Po zavolání funkce pro získání matice transformace je volána funkce *setMatrix*. Jako parametr je předán vytvořený objekt matice transformace a číslo reprezentující DOM element nastavované matice. Zmiňovaný objekt matice je ve THREE.js vždy uložen v transponovaném stavu, proto funkce nejdříve matici transponuje a následně její hodnoty nastaví do matice v aplikaci.

Transformace objektů

Jako první krok transformace objektů je volání funkce *calculateFinalMatrix*. Funkce inicializuje nový objekt *matrix4* reprezentující finální matici transformace. V po-

stupných iteracích je vždy volána funkce *getMatrixInput*, která načte hodnoty matice v aplikaci do objektu *matrix4*. Následně je získanou maticí vynásoben objekt finální matice transformace. Po vynásobení všemi dílčími maticemi je objekt finální matice vrácen. Získáno je vždy všech šest matic, tedy aplikace nerozlišuje, jestli je v matici nastavena transformace nebo identita.

Po získání finální matice je volána funkce *transformObjects*. Tato funkce vynásobí matici objektu, který reprezentuje stav po transformaci, získanou finální maticí. Matice objektu, který reprezentuje stav před transformací, je vynásobena uloženou finální maticí z předchozího kroku. Po transformaci objektů je současná finální matice uložena do finální matice z předchozího kroku. V prvním kroku je finální matice z předchozího kroku nastavena na matici identity, tedy druhý objekt se nijak nezmění.

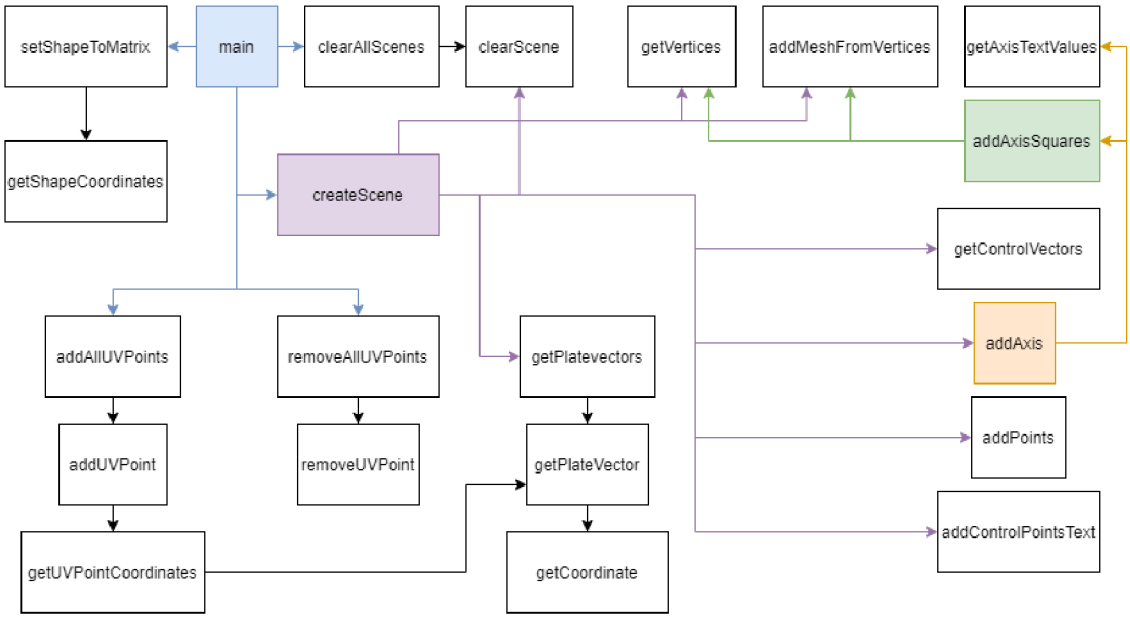
Restart Scény

Při restartování scény je jako první krok volána funkce *removeObjects*. Tato funkce odebere ze scény oba zobrazované objekty, tedy objekt po transformaci a před transformací. Ostatní objekty ve scéně zůstávají nezměněny, tedy například objekt pomocné osy souřadnic je zachován. Následně je zavolána funkce *createObjects* pro přidání objektů. Na obrázku 3.2 je ukázán diagram aplikace bez méně významných funkcí.

3.3.4 2. aplikace – Beziérův plát

V hlavní funkci jsou znovu načteny všechny potřebné DOM elementy do proměnných a následně přidány listenery volající potřebné operace. Pro všechny čtyři canvasy je zavolána funkce *initiateScene*, která inicializuje scény. Také je vytvořena instance třídy *ClickerBox*, která reprezentuje okno pro klikání souřadnic zobrazovaného bodu plátu.

Hlavní funkce také zajišťuje, aby při načtení aplikace byl již zobrazen nějaký plát. Tedy v hlavní funkci je následně zavolána funkce *setShapeToMatrix*. Tato funkce načte hodnotu z rozbalovacího okna pro výběr tvaru plátu a předá tuto hodnotu jako parametr do funkce *getShapeCoordinates*, která podle předané hodnoty vrátí pole řídících bodů vybraného tvaru. Tyto řídící body jsou následně nastaveny do matice řídících bodů v aplikaci. Stejná funkce je také volána, pokud je změněna hodnota v rozbalovacím okně tvaru plátu. Posledním krokem je zavolání funkce *createScene* pro každý z inicializovaných plátů.



Obr. 3.2: Diagram funkcí druhé aplikace bez dílčích funkcí.

Zobrazení plátu

Hlavním prvkem zobrazení plátu je funkce *createScene*. Tato funkce musí nejdříve zajistit, že jsou smazány všechny objekty ve scéně, které byly vytvořeny při vykreslení předchozího plátu. To je provedeno voláním funkce *clearScene*. Po smazání objektů funkce přejde k vytvoření nového plátu. Pro lepší práci s body na ploše je využito THREE.js objektu *Vector3*, který reprezentuje vektor v prostoru. Nejdříve je zavolána funkce *getControlVectors*, která převede souřadnice řídících bodů do pole objektů *Vector3*.

Získané pole řídících bodů je předáno jako parametr do funkce *getPlateVectors*. Cílem této funkce je jednak výpočet bodů plátu pro vytvoření meshe plátu, a jednak výpočet minimálních a maximálních hodnot tohoto plátu na všech osách. Pojmem mesh je myšlen povrch daného objektu, respektive zobrazovaného plátu. Jednotlivé body na ploše jsou získávány podle definovaného kroku, tedy tento krok definuje hustotu bodů plátu. Krok je přednastaven na hodnotu 0.01 v rozsahu 0 až 1 pro parametry u a v.

V každém kroku je zavolána funkce *getPlateVector*, která určí, pro jakou scénu je plát vytvářen, a se správnými hodnotami zavolá funkci *getCoordinate*. Tedy pokud je plát vytvořen pro první scénu, předány jsou všechny souřadnice vektoru. Pokud je plát vytvářen pro ostatní scény, předána je pouze hodnota osy, která je zobrazována, a hodnoty na ostatních osách jsou nahrazeny parametry u a v. Zmiňovaná funkce *getCoordinate* implementuje vzorec 1.21 s využitím THREE.js objektů *Matrix4* a *Vector4*. Jelikož je zobrazován pouze jeden plát ve scéně, je zmiňovaný vzorec

pro výpočet bodu plátu dostačující a nebylo nutné implementovat pokročilejší, více optimalizované postupy výpočtu.

Současně s výpočtem bodu plátu je v každém kroku porovnán vypočítaný bod s minimální a maximální uloženou hodnotou pro jednotlivé osy. Minimální a maximální hodnota je později využita pro zobrazení pomocných os grafu. Tato operace není vyčleněna do vlastní funkce z důvodu optimalizace, tedy aby nebylo nutné znovu iterovat přes všechny body plátu.

Po získání bodů plátu je zavolána funkce *getVertices*, které je jako parametr předáno pole získaných bodů plátu. Funkce z daných bodů vypočítá jednotlivé vertexy a uloží je do dvou polí. Dvě pole vertexů jsou vytvořeny z důvodu následného zbarvení plátu, tedy oba meshe mohou mít mírně odlišnou barvu a tím zlepšit prostorové vnímání plátu. Této možnosti nebylo ve finální verzi využito, nicméně implementace daných funkcí byla zachována, a je možné ji znovu využít, pokud by byla tato funkcionalita požadována.

Získaná pole vertexů jsou předána jako parametr do funkce *addMeshFromVertices*. Funkce nejdříve vytvoří dvě instance THREE.js třídy *BufferGeometry* pro oba meshe. Do vytvořeného objektu jsou pomocí metody *setAttribute* nastaveny vertexy. Následně je vytvořena instance třídy *MeshBasicMaterial*, které je nastavena barva a průhlednost. Pomocí objektů geometrie a materiálu jsou vytvořeny finální THREE.js objekty *Mesh* a přidány do scény.

Funkce také kontroluje, pro jakou scénu je plát vytvářen, a pokud se jedná o hlavní scénu, tak jsou do scény přidány řídící body. Řídící body jsou přidány pomocí funkce *addPoints*, která na základě předaných souřadnic do scény přidá body. K jednotlivým bodům jsou také přidány popisky, které označují, o jaký bod se jedná. Za tímto účelem je volána funkce *addControlPointsText*, která vytvoří popisek bodu, a následně ho přidá do scény funkcí *addText*.

Funkce *addText* přidá do scény na definovaných souřadnicích text. Text je vložen do nově vytvořeného canvasu, který je následně využit pro vytvoření THREE.js objektu *Sprite*. Tento objekt je 2D a vždy směřuje ke kameře, tedy text je čitelný i při natočení kamery.

Vytvoření grafu

Jak již bylo popsáno v části funkcionality, pláty jsou vykresleny do grafů. Všechny prvky těchto grafů bylo nutné vytvořit jako další objekty ve scéně, respektive jako přímky a plochy s popisem. Vytvoření těchto grafů je zajištěno funkcí *addAxis*, která je volána jako poslední operace ve funkci *createScene*, popsané v předchozí části.

Ve funkci je nejdříve vytvořen materiál pro přímky a je získán počátek souřadnic grafu, který je uložen jako objekt *Vector3*. Pro každou osu je vytvořeno pole obsahu-

jící dva vektory. Prvním vektorem je zmiňovaný počátek souřadnic a druhým je bod s maximální hodnotou na dané ose. Tato pole jsou následně využita pro vytvoření objektů *BufferGeometry*. Pomocí vytvořené geometrie a materiálu jsou vytvořeny THREE.js objekty *Line*, které reprezentují přímky os.

K přímkám je nutné přidat jednotlivé popisy hodnot na osách. Pro získání textu je zavolána funkce *getAxisTextValues*, která podle scény, do které má být plocha zobrazena, vrátí popis os. Následně jsou vypočítány souřadnice, na které má být text zobrazen, a pro každý popisek je zavolána funkce *addText*.

Zobrazení bodu

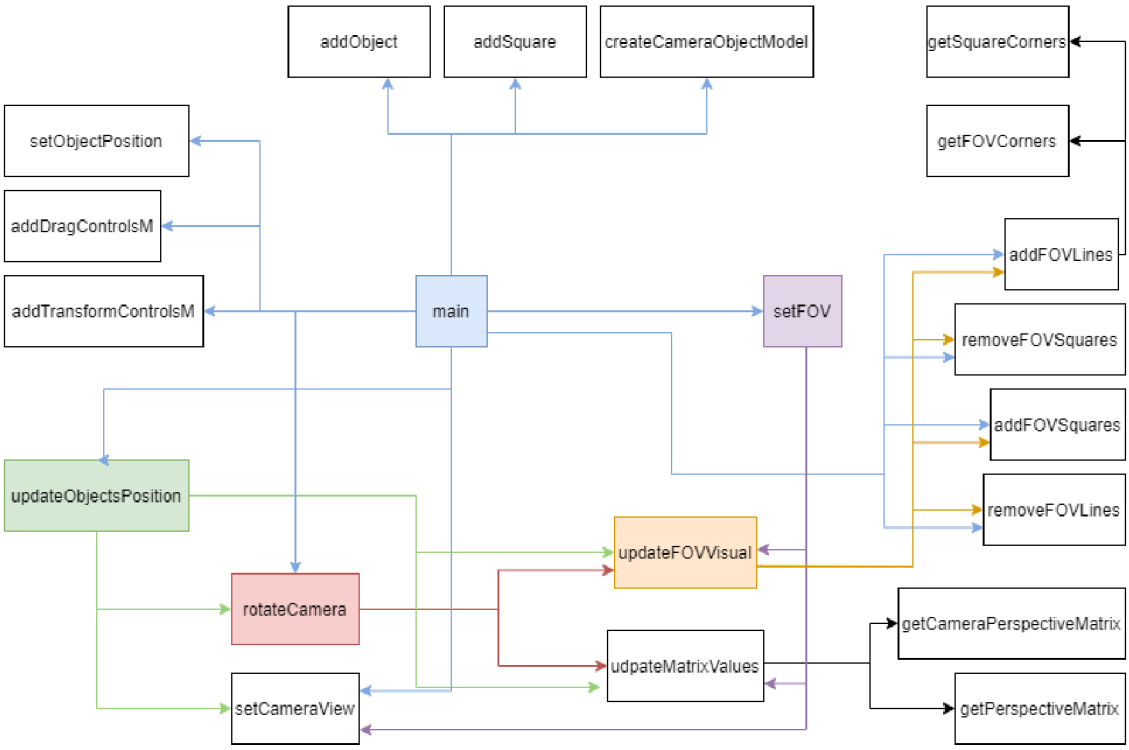
Zobrazení bodu je řízeno dvěma listenery, které jsou navázány na zaškrtávací pole a klikací plochu pro určení hodnot parametrů u a v. Při zaškrtnutí pole pro zobrazení bodu je zavolána funkce *addAllUVPoints*, při odškrtnutí pole je zavolána funkce *removeAllUVPoints*, která zobrazené body smaže. Při kliknutí do plochy parametrů je nejdříve zavolána funkce *removeAllUVPoints* a poté funkce *addAllUVPoints*. Tím je zajištěno, že pokud uživatel klikne do plochy parametrů a bod je ve scéně zobrazen, tak je automaticky překreslen na nově definovaný bod.

Funkce *addAllUVPoints* zavolá funkci *addUVpoint* pro všechny čtyři scény. Ve funkci *addUVPoint* je vypočítán bod na ploše pomocí funkce *getUVCoordinates*. Zároveň zkontroluje, jestli je bod zobrazován pro hlavní scénu a pokud ano, tak přidá do scény další tři body na jednotlivých osách.

3.3.5 3. aplikace – Kamera ve scéně

Při načtení aplikace je zavolána hlavní funkce, která zajišťuje přípravu scén do výchozího nastavení, zároveň jsou znovu propojeny jednotlivé prvky ovládání. V této funkci jsou definovány souřadnice jednotlivých objektů, které jsou následně do scény přidány. Dále se volá funkce pro přidání jednotlivých typů ovládání do první scény.

Funkce do načtených canvasů inicializuje jednotlivé scény. V této aplikaci se funkce pro inicializování scény mírně liší a to z důvodu ovládání kamery. V první scéně je oproti ostatním aplikacím následně přidáno další ovládání. V druhé aplikaci žádné ovládání kamery přidáno není, jelikož kamera má pořád fixní pozici, která je určena na základě manipulace s objektem kamery v první scéně. Tedy druhá scéna je inicializována bez přidání objektu *OrbitControls*. Diagram volání funkcí aplikace je možné vidět na obrázku 3.3. Pro lepší přehlednost diagramu jsou vynechány některé méně důležité funkce.



Obr. 3.3: Diagram volání funkcí třetí aplikace bez dílčích funkcí.

Zobrazení scén

Pro přidání základních objektů do scén je volána funkce *addObject*. Tato funkce na základě zvolených parametrů vytvoří objekt určitého tvaru a velikosti a přidá ho do scény. Objekt kamery a objekt průmětny jsou vytvářeny specifickými funkcemi, tedy není využito funkce *addObject*. Funkce *createCameraObjectModel* vytvoří pomocí základních geometrických tvarů model kamery, který je následně přidán do scény. Objekt průmětny je do scény přidán funkcí *addSquare*, která pomocí THREE.js objektu *LineSegments* vytvoří čtverec, a přidá ho do scény.

Následně je zavolána funkce *addCameraTargetConnect*, která do první scény přidá optickou osu kamery. Funkce *updateFOVVIsual* je zavolána pro přidání ploch zorného pole kamery. Hlavní funkce také obsahuje kontrolu vstupu pro oba posuvníky. Pokud je zadána nevyhovující hodnota, je využito základní JavaScript funkce *alert* pro zobrazení chybové hlášky.

Po přidání objektů do scény je nutné aktualizovat jejich pozice podle počátečního nastavení. K tomuto účelu slouží funkce *setObjectsPosition*. Funkce *updateObjectsPosition* je volána při manipulaci s některým z objektů. Nejdříve jsou získány pozice objektu kamery a objektu cíle kamery, které jsou využity pro výpočet nové pozice průmětny. Po aktualizaci pozice průmětny v obou scénách je pomocí metody *lookAt* nastaven směr pohledu objektu kamery v první scéně. Pro nastavení pohledu

kamery v druhé scéně je volána funkce *setCameraView*, která také využívá metody *lookAt*. Pro zachování rotace kamery je funkce *rotateCamera*.

Ovládání scény

Kromě samotného ovládání kamery pomocí objektu *OrbitControls* jsou do scény přidány další dva typy ovládání. Jedná se *DragControls* a *TansformControls*. Stejně jako *OrbitControls* se i tato ovládání musí explicitně importovat ze složky příkladů knihovny THREE.js.

Pro posun objektů pomocí myši slouží *DragControls*. Kromě toho umožňuje toto ovládání odchycení různých událostí při posunu objektů. Je toho využito například pro změnu průhlednosti objektu, pokud je přesouván. Při posunu kurzoru nad objekt ve scéně je deaktivováno *OrbitControls*, aby uživatel mohl pomocí levého tlačítka přesouvat objekt místo rotace kamery. Přidání tohoto ovládání do scény je provedeno funkcí *addDragControls*.

Funkce *addTransformControls* je volána pro přidání ovládání *TansformControls*. Toto ovládání umožňuje uživateli transformace zobrazovaného objektu. Tedy uživatel může pomocí klávesnice vybrat typ transformace, a potom levým tlačítkem myši přímo transformaci provádět. Při manipulaci s objektem je *OrbitControls* a *DragControls* deaktivováno, aby uživatel mohl využívat myš pro toto ovládání.

Rotace kamery

Pro nastavení rotace kamery slouží funkce *rotateCamera*. Funkce nejdříve získá vektory, které reprezentují pozici objektu kamery a pozici cíle objektu kamery. Následně je z těchto bodů spočítán směrový vektor optické osy. Před zavoláním této funkce jsou uloženy matice objektu kamery a objektu průmětny pro následné využití při samotném rotování kamery.

Matice těchto objektů nejdou přímo nastavit, proto jsou objekty přesunuty do počátku souřadnic a jejich rotace je vynulována. Následně jsou objekty rotovány pomocí metody *rotateOnWorldAxis* o požadovaný úhel, a na jejich rotovanou matici je aplikována předem uložená matice. Tímto je objekt vrácen do původní pozice a je rotovaný o požadovaný úhel.

Popsaný postup sice rotuje objekty v první scéně, nicméně je nutné také rotovat zobrazovaný objekt v druhé scéně. To je zajištěno také metodou *rotateOnWorldAxis*, ale objekt je rotován v opačném směru, jelikož samotnou kamerou nelze rotovat.

Nastavení FOV

Zorné pole kamery je navázáno na zaškrtávací pole, která určují, jestli mají být plochy zorného pole zobrazeny, a na posuvník určující aktuální nastavenou hodnotu.

Pro aktualizaci nastavení FOV je volána funkce *updateFOVVisual*, která zkontroluje, jestli mají být přímky nebo plochy zobrazeny, a zavolá požadované funkce.

Při odškrtnutí pole pro zobrazení FOV přímek je pro smazání těchto přímek volána funkce *removeFOVLines*. Po zaškrtnutí pole je volána funkce *addFOVLines*. Pro získání rohů průmětny je volána funkce *getSquareCorners* a pro získání rohů samotného zorného pole je volána funkce *getFOVCorners*. Následně jsou vytvořeny čtyři objekty *BufferGeometry*, které mají nastaveny dva body. Jedním bodem je vždy pozice kamery a druhým jeden z rohů zorného pole. Jako další je vytvořen objekt základního materiálu *MeshLambertMaterial*, který je průhledný. Následně jsou pomocí těchto objektů vytvořeny objekty přímek *Line*.

Pokud je pole pro zobrazení FOV ploch odškrtnuto, tak je volána funkce *removeFOVSquares*, která odstraní zobrazené plochy. Pro zobrazení ploch je volána funkce *addFOVSquares*. Tato funkce využije souřadnice rohů zorného pole, které byly vypočítány při zobrazování přímek. Pomocí těchto souřadnic jsou znovu vytvořeny objekty *BufferGeometry*. Objekt materiálu pro zobrazení ploch má v tomto případě nastaven vysokou průhlednost a je znovu společně s objektem geometrie využit pro vytvoření objektů ploch *Mesh*.

Při změně hodnoty na posuvníku, který určuje vzdálenost kamery od průmětny, je volána funkce *setFOV*. Tato funkce nejdříve spočítá nový bod, na který má být kamera přesunuta, a aktualizuje pozici objektu kamery v obou scénách. Následně je přenastaven objekt geometrie optické osy tak, aby optická osa vycházela z nové pozice kamery. V druhé scéně je pomocí metody *setFocalLength* nastaveno FOV kamery podle nové vzdálenosti. Posledním krokem je zavolání funkce *setCameraView*, která aktualizuje pozici a směr pohledu kamery ve druhé scéně.

Výpočet matic

Aktualizace hodnot v maticích je prováděna funkcí *updateMatrixValues*. V této funkci jsou zavolány dílčí funkce *getCameraPerspectiveMatrix* a *getPerspectiveMatrix*, které vypočítají a vrátí první dvě matice. Následně jsou tyto matice transponovány a vynásobeny. Tím je získána finální matice. Všechny hodnoty matic jsou následně nastaveny pomocí funkce *setMatrix*. Matice jsou získány podle vzorců 1.38 a 1.40.

4 Možnosti rozšíření

Ve všech aplikacích je v současném stavu implementována základní požadovaná funkcionalita a zároveň je přidáno několik rozšiřujících prvků. Nicméně aplikace je možné rozšířit o další komplexnější funkcionality. Některé návrhy rozšíření jsou popsány v této kapitole.

Kromě rozšíření funkcionality je také možné aplikace vylepšit v oblasti implementace a to hlavně ve dvou oblastech. První oblastí je lepší optimalizace kódu. V současném stavu při otevření a používání aplikací nebyla zaznamenána žádná významná prodleva v načítání, nicméně aplikace jako celek i dílčí operace by bylo možné optimalizovat z hlediska výkonu. Druhou možností by bylo uhlazení vytvořeného kódu pro lepší čitelnost a tím tedy i jednodušší rozšiřování funkcionality.

4.1 Společná rozšíření

Při popisování implementace scény bylo zmíněno, že jednotlivé scény nejsou responzivní, tedy vykreslená scéna nemění svou velikost podle změny canvasu. Tento nedostatek není zásadní pro fungování aplikace, protože uživatel by neměl často měnit velikost okna a pokud ji změní, tak stačí aplikaci znovu načíst, nicméně toto rozšíření je jedno z prvních, které by bylo pro aplikace vhodné. Pro přidání responzivity vykreslování by bylo nutné při změně velikosti canvasu uložit nastavení současné scény do objektu, následně scénu vymazat a znovu vytvořit podle předešlého uloženého nastavení. Samotné canvasy jsou responzivní a mění svou velikost při změně velikosti okna prohlížeče.

Jak již bylo zmíněno, pro implementaci aplikací bylo využito imperativního přístupu. Před začátkem implementace jsem předpokládal, že aplikace nebudou výrazně komplexní a dlouhé, proto jsem považoval tento přístup za dostačující. Po implementaci mi přijde rozdělení do funkcí stále dostačující, nicméně objektový přístup považuji za vhodnější. Například u druhé aplikace by se scény a pláty daly definovat jako objekty, což by mohlo pomoci jednak v přehlednosti implementace a také pro případné rozšíření aplikace a přidání dalších komponent.

Dalším rozšířením implementace by byla optimalizace aplikací pro mobily. Jednotlivé prvky jsou převážně responzivní a neměl by nastat zásadní problém při načtení aplikací na mobilu z hlediska prvků stránky. Problém však nastává v samotném používání aplikací, které obsahují 3D scény, na mobilech. Mobily v porovnání s počítači nedisponují stejnými možnostmi ovládání, tedy bylo by nutné vyřešit, jak uživatel může ovládat scénu pomocí dotykového displeje. V první a druhé aplikaci, kde uživatel posouvá pouze pohled kamery, problém s ovládáním nenastává,

ale v případě třetí aplikace, kde uživatel může manipulovat s objekty ve scéně, dochází k problémům s ovládáním. Zároveň vykreslené scény ve všech aplikacích jsou na mobilních telefonech malé.

Posledním návrhem rozšíření všech aplikací je přidání možnosti nastavení světla. Světlo a stíny jsou ve 3D scénách důležitými prvky přehlednosti scény. V každé aplikaci bylo světlo nastaveno tak, aby scéna byla dobře přehledná, avšak každý uživatel má jiné preference a mít možnost změny barvy, pozice nebo intenzity světla by bylo vhodné.

4.2 1. aplikace – Transformace objektů

Jedním z jednoduchých rozšíření první aplikace by bylo přidání dalších tvarů objektů. První možností by bylo do výběru tvaru přidat další definované možnosti. Komplexnější možností by bylo přidat funkcionalitu, přes kterou by uživatel mohl přidat vlastní definovaný vektorový objekt, který by byl do aplikace importován.

Kromě přidání dalších objektů by mohla být přidána funkcionalita, která umožňuje změnit barvu objektů a případně jejich velikost při vytvoření. Znovu se nejedná o zásadní rozšíření aplikace, ale mohlo by uživateli zpříjemnit využívání aplikace. Změna barvy může zlepšit přehlednost scény a změna velikosti umožní uživateli definovat větší objekty než definovaná velikost.

Poslední návrh rozšíření je zaměřen na matice transformací. V tuto chvíli je uživatel limitován při zadávání transformací na šest matic. Do aplikace by byla přidána funkcionalita, která umožňuje přidání nebo odebrání matic transformací. Uživatel by tedy mohl vytvořit opravdu komplexní transformace nebo naopak matice odebrat, pokud je nevyužívá. Zároveň by bylo možné aplikaci rozšířit i pro zadávání homogenních souřadnic.

4.3 2. aplikace – Beziérův plát

Stejně jako v první aplikaci by bylo i zde možné přidat další tvary. V tomto případě by se jednalo o tvary plátu, který je zobrazován. Zároveň by uživatel mohl měnit barvu plátu, například přepínat mezi dvojbarevným zobrazením plátu, které by mohlo pomoci v přehlednosti u některých plátů. Kromě barvy by mohlo zlepšení přehlednosti sloužit nastavení průhlednosti zobrazovaného plátu.

V současné chvíli může uživatel definovat řídící body jakkoliv. Tento fakt může způsobit komplikace s kamerou při zobrazování atypických plátů. Například pokud by uživatel zadal extrémně velký plát, pozice kamery by mohla být příliš blízko plátu a nestačilo by ani maximální oddálení kamery. Tato problematika se dá řešit

dvěma způsoby. Prvním je kontrola zadaných řídících bodů uživatelem a případné zobrazení varovné hlášky uživateli. Druhou možností je komplexnější implementace přepočítání pozice kamery při zobrazování plátu.

Dalším návrhem je zlepšení ovládání klikacího pole pro zadání hodnot parametrů. U tohoto prvku mohou nastat problémy při klikání do krajů pole, například zvolit hodnotu 0 pro oba parametry může být náročnější. Pomoc se zadáváním hodnot by ulehčilo i možnost upravovat parametry přímo zadáním hodnoty. Pod klikacím polem jsou zobrazovány aktuální zvolené hodnoty pro parametry. Tyto hodnoty by bylo možné upravovat a po zadání by se překreslil zvolený bod v klikacím poli.

V aplikaci by bylo možné vylepšit zobrazování textu ve scéně, tedy jednotlivé popisky os a bodů. V současné chvíli je text mírně rozmazaný a bylo by nutné přepracovat nastavení objektu *Sprite*, který je pro zobrazení textu využit.

Posledním rozšířením by bylo optimalizování algoritmu pro výpočet plochy. Současná implementace pláty vykresluje okamžitě, nicméně optimalizování výpočtu by bylo vhodné, pokud bychom chtěli v aplikaci zobrazovat extrémně velké pláty.

4.4 3. aplikace – Kamera ve scéně

Jako hlavní možnost rozšíření této aplikace vidím optimalizaci implementace. Návrh aplikace byl několikrát mírně upravován v průběhu implementace a zároveň logika aplikace je výrazně složitější než u předchozích aplikací. Z těchto důvodů výsledná implementace není ideální a bylo by vhodné kód refaktorovat.

V současném stavu je nastavení FOV nepřímé, tedy je nastavována vzdálenost kamery od průmětny, což ovlivňuje zorné pole kamery. Nastavení by se dalo přepracovat tak, aby uživatel mohl zvolit úhel zorného pole a tím ho přímo nastavit.

Dále by bylo vhodné implementovat možnost přidávání dalších zobrazovaných objektů do scén. V tuto chvíli je zobrazena pouze jedna krychle. Uživatel by mohl přidat další objekty a případně měnit jejich tvary, což by umožnilo sledovat interakci a zobrazení více objektů ve scéně. Zároveň by uživatel mohl měnit světlo ve scéně a barvy objektů.

Závěr

Práce se věnovala vytvoření tří aplikací pro výuku počítačové grafiky. Každá z aplikací je zaměřena na jednu z konkrétních oblastí počítačové grafiky. V průběhu přípravy aplikací byla nastudována základní potřebná teorie, která byla následně sepsána do této práce. Kapitola s teoretickou částí obsahuje základní klíčové informace pro implementované aplikace. Aplikace budou sloužit jako doplňující prvek při výuce počítačové grafiky, tedy jejich implementace je v souladu s popsanou teorií ve skriptech předmětu MGMP.

Pro všechny aplikace byla implementována základní požadovaná funkcionalita a také několik rozšiřujících funkcionalit. Aplikace jsou spustitelné na běžně využívaných prohlížečích. Scény zobrazované aplikacích jsou poměrně velké a je tedy doporučeno a předpokládáno, že uživatel by měl aplikace používat na počítači.

Při implementaci bylo využito několika pokročilých technologií, které jsou v práci krátce popsány. Před zahájením implementace aplikací bylo nutné nastudovat fungování frameworku THREE.js, který byl pro implementaci využit. Práce s frameworkem byla v některých částech obtížná, jelikož dokumentace sice obsahuje popis všech komponent, nicméně často chybí detailní příklady využití jednotlivých komponent.

Při návrhu a implementaci aplikací bylo uvažováno několik rozšiřujících funkcionalit, které nakonec nejsou implementovány, jelikož nejsou pro používání aplikací zásadní a ani nebyly v rámci této práce požadovány. Návrhy těchto funkcionalit jsou popsány v poslední kapitole jako případná rozšíření těchto aplikací v budoucnu. Kapitola zároveň uvádí známé nedostatky jednotlivých aplikací, nicméně pro účely výuky považuji aplikace za dokončené a podle zpětné vazby vedoucího budou využity při výuce.

Literatura

[1] P. Rajmic and J. Schimmel. *Moderní počítačová grafika.* Vysoké učení technické v Brně Fakulta elektrotechniky a komunikačních technologií Ústav telekomunikací, Brno, 1 edition, 2013.

[2] J. Žára, B. Beneš, J. Sochor, and P. Felkel. *Moderní počítačová grafika.* Computer Press, Brno, 2 edition, 2004.

[3] V. M. Shekhat. Basics of Computer Graphics. [online], Naposledy navštíveno 16. 5. 2021. URL: `http://www.darshan.ac.in/Upload/DIET/Documents/CE/2160703_CG_GTU_Study_Material_2017_11042017_033102AM.pdf`.

[4] University of Cambridge. Computer Graphics and Image Processing. [online], Naposledy navštíveno 16. 5. 2021. URL: `https://www.cl.cam.ac.uk/teaching/1998/CGraphIP/cgip.pdf`.

[5] J. Collomosse. Fundamentals of computer graphics. Technical report, University of Bath, September 2008. URL: `http://personal.ee.surrey.ac.uk/Personal/J.Collomosse/pubs/cm20219.pdf`.

[6] Y. G. Shin. Computer graphics by seoul national university. Technical report, Seoul National University, November 2008. URL: `http://ocw.snu.ac.kr/node/2195`.

[7] M. Langer. Fundamentals of computer graphics. Technical report, School of Computer Science McGill University, December 2015. URL: `http://www.cim.mcgill.ca/~langer/557/lecturenotesCOMP557.pdf`.

[8] WHATWG. HTML Standard. [online], Naposledy navštíveno 16. 5. 2021. URL: `https://html.spec.whatwg.org/`.

[9] Bert Bos. All CSS specification. [online], Naposledy navštíveno 16. 5. 2021. URL: `https://www.w3.org/Style/CSS/specs.en.html`.

[10] Khronos Group. WebGL Overview - The Khronos Group Inc. [online], Naposledy navštíveno 16. 5. 2021. URL: `https://www.khronos.org/webgl/`.

[11] Khronos Group. OpenGL ES Overview - The Khronos Group Inc. [online], Naposledy navštíveno 16. 5. 2021. URL: `https://www.khronos.org/opengles/`.

[12] Processing Foundation. home | p5.js. [online], Naposledy navštíveno 16. 5. 2021. URL: `https://p5js.org/`.

[13] three.js – JavaScript 3D library. [online], Naposledy navštíveno 16. 5. 2021. URL: https://threejs.org/.

[14] Inc. npm. npm. [online], Naposledy navštíveno 16. 5. 2021. URL: https://www.npmjs.com/.

[15] Nunjucks. [online], Naposledy navštíveno 16. 5. 2021. URL: https://mozilla.github.io/nunjucks/.

[16] JetBrains s.r.o. Webstorm: The Smartest Javascript IDE by JetBrains. [online], Naposledy navštíveno 16. 5. 2021. URL: https://www.jetbrains.com/webstorm/.

Seznam symbolů, veličin a zkratek

2D	označuje dvoudimenzionální prostor
3D	označuje trojdimenzionální prostor
API	zkratka pro *Application Programming Interface*
WCS	zkratka pro *World Coordinate System*
VCS	zkratka pro *Viewing Coordinate System*
NPM	zkratka pro *Node Package Manager*
HTML	zkratka pro *Hypertext Markup Language*
CSS	zkratka pro *Cascading Style Sheets*
DOM	zkratka pro *Document Object Model*

Seznam příloh

A Přiložené soubory a kompilace

Aplikace jsou umístěny na soukromé doméně a pro jejich přístup je nutné použít následující konkrétní odkazy:

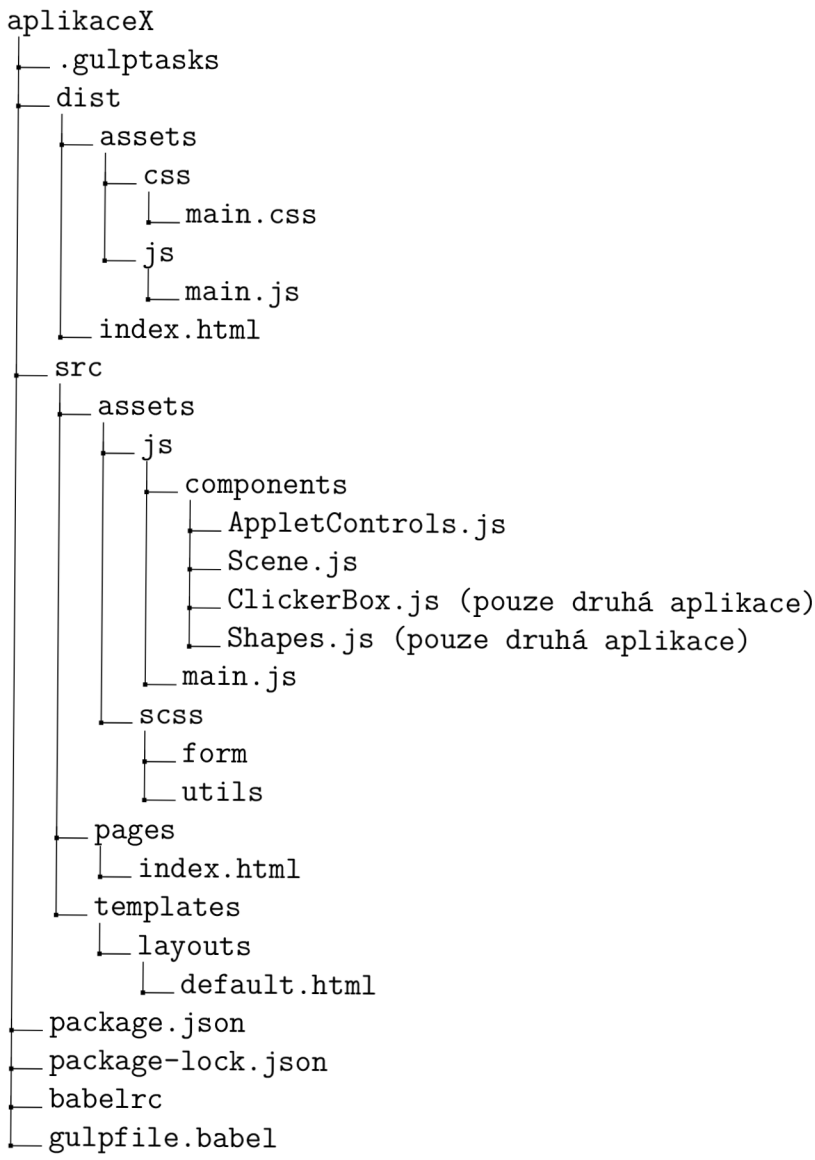
- aplikace 1 – `https://applet1.mrlcs.cz`,
- aplikace 2 – `https://applet2.mrlcs.cz`,
- aplikace 3 – `https://applet3.mrlcs.cz`.

V přiložených souborech jsou umístěny tři složky *aplikace1*, *aplikace2* a *aplikace3*. Každá složka reprezentuje zdrojové soubory pro danou aplikaci. Pod tímto textem je přiložena stromová struktura zmiňovaných složek. Tato struktura je pro každou složku stejná, pouze v druhé aplikaci jsou přidány dva JavaScript soubory.

Složka *dist* obsahuje zkompilované projekty projekty, tedy pokud klikneme na soubor *index.html* v této složce, tak se spustí samotná aplikace. Zároveň pro nahrání na server stačí zkopírovat celý obsah této složky. Tyto soubory nejsou po kompilaci normálně čitelné, tedy pro nahlédnutí do zdrojového kódu je potřeba otevřít soubory před kompilací popsané v následujícím odstavci.

Ve složce *src/assets/js* jsou jednotlivé zdrojové JavaScript soubory před kompilací. Ve složkách *src/pages* a *src/templates* se nachází jednotlivé HTML soubory. Složka *src/assets/scss* obsahuje jednotlivé CSS komponenty. Z důvodu velkého počtu těchto CSS souborů, jsou v přiložené stromové struktuře vynechány. Ostatní složky a soubory obsahují pomocné nastavení jednotlivých technologií a zajišťují správný běh kompilace.

Pokud by uživatel chtěl projekt zkompilovat, například po změně některé části zdrojového kódu, je nutné mít nainstalované NPM. Instalace NPM se provede automaticky při instalaci Node.js, který je možné stáhnout z odkazu `https://nodejs.org/en/`. Po nainstalování NPM, respektive Node.js, musí uživatel otevřít konzoli a v konzoli přejít do složky s projektem. Následně je nutné zadat příkaz *npm install*, který nainstaluje potřebné knihovny a technologie. Pro samotnou kompilaci je nutné zadat příkaz *npm run build*, který projekt zkompiluje a uloží do zmiňované složky *dist*. Může nastat případ, kdy se v konzoli spuštěná operace *npm run build* sama neukončí, stačí počkat několik sekund na vypsání řádku „Finished 'build'“ a následně pomocí kláves „Ctrl + c“ operaci vypnout.


aplikaceX
.gulptasks
dist
assets
css
main.css
js
main.js
index.html
src
assets
js
components
AppletControls.js
Scene.js
ClickerBox.js (pouze druhá aplikace)
Shapes.js (pouze druhá aplikace)
main.js
scss
form
utils
pages
index.html
templates
layouts
default.html
package.json
package-lock.json
babelrc
gulpfile.babel