VYSOKÁ ŠKOLA POLYTECHNICKÁ JIHLAVA

Aplikovaná informatika

VYKRESLOVÁNÍ 3D MODELŮ POMOCÍ HTML CANVAS V 2D KONTEXTU

Bakalářská práce

Autor práce: Jáchym Čech

Vedoucí práce: PaedDr. František Smrčka, Ph.D.

Jihlava 2026

Vysoká škola polytechnická Jihlava

Tolstého 16, 586 01 Jihlava

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Autor práce:	**Jáchym Čech**
Studijní program:	Aplikovaná informatika
Garant studijního programu:	doc. Ing. Lenka Kuklišová Pavelková, Ph.D.
Název práce:	**Vykreslování 3D modelů pomocí HTML Canvas v 2D kontextu**
Vedoucí práce:	PaedDr. František Smrčka, Ph.D.
Cíl práce:	Cílem práce je vytvořit nástroj pro vykreslování 3D modelů ve formátu OBJ v prostředí HTML Canvas, a to s využitím pouze 2D kontextu, bez použití 3D knihoven. Výsledkem bude zobrazení 3D objektů, které si uživatelé budou moci ve webovém prohlížeči jednoduše prohlížet a provádět na nich menší úpravy. Nástroj umožní importovat Wavefront OBJ soubory pro samotné 3D objekty, MTL soubory pro popis materiálů a textury ve formě obrázků. Součástí práce bude metodika pro vytvoření vlastního programu vykreslování 3D modelů pomocí 2D API.

Abstrakt

Bakalářská práce je zaměřena na vytvoření programu pro vykreslování 3D modelů bez použití specializovaných grafických knihoven. Cílem je ukázat postup převodu dat modelu na výsledný obraz a přiblížit základní principy grafických technik. Vytvořený program umožňuje načítat modely ve formátu obj, základní materiály a textury, provádět úpravy geometrie a barev a následně je exportovat. Vykreslovací proces je založen na rasterizaci prováděné pomocí Z-bufferu a na mapování obrázků pomocí afinního zobrazení. Při tvorbě řešení byla pozornost zaměřena na jasné oddělení jednotlivých kroků vykreslování, aby bylo možné snadno sledovat průběh zpracování dat. Výsledkem je nástroj, který umožňuje zobrazit trojrozměrné objekty, a může tak sloužit jako výuková pomůcka, protože ukazuje praktické využití základních principů 3D grafiky.

Klíčová slova

Počítačová grafika; javascript; canvas 2D context; obj formát; z-buffer

Abstract

The bachelor's thesis focuses on developing a program for rendering 3D models without the use of specialized graphics libraries. The aim is to demonstrate the process of converting model data into the final image and to illustrate the fundamental principles of graphics techniques. The created program allows loading models in the OBJ format, basic materials and textures, performing modifications of geometry and colors, and subsequently exporting them. The rendering process is based on rasterization using a Z-buffer and on image data mapping using affine transformations. During the development of the solution, attention was given to clearly separating the individual steps of rendering, making it easy to follow the flow of data processing. The result is a tool that enables the visualization of three-dimensional objects and can serve as a teaching aid, as it demonstrates the practical application of the fundamental principles of 3D graphics.

Keywords

Computer graphics; javascript; canvas 2D context; obj format; z-buffer

Prohlašuji, že předložená bakalářská práce je původní a zpracoval/a jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem v práci neporušil/a autorská práva (ve smyslu zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů, v platném znění, dále též „**AZ**“).

Byl/a jsem seznámen/a s tím, že na mou bakalářskou práci se plně vztahuje **AZ**, zejména § 60 (školní dílo).

Podle § 47b zákona o vysokých školách souhlasím se zveřejněním své práce podle Směrnice pro vedení, vypracování a zveřejňování závěrečných prací na VŠPJ, a to bez ohledu na výsledek obhajoby.

Beru na vědomí, že VŠPJ má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom/a toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem VŠPJ, která má právo ode mě požadovat přiměřený příspěvek na úhradu nákladů, vynaložených vysokou školou na vytvoření díla (až do jejich skutečné výše), z výdělku dosaženého v souvislosti s užitím díla či poskytnutím licence.

V Jihlavě dne 7. dubna 2026

…….

Podpis studenta

Poděkování

Rád bych poděkoval vedoucímu své bakalářské práce, PaedDr. Františku Smrčkovi, Ph.D., za jeho trpělivý přístup po celou dobu tvorby práce. Dále děkuji své rodině, přátelům a spolužákům za podporu během studia a všem učitelům za jejich vstřícnost a cenné rady.

Obsah

Seznam obrázků

Seznam zkratek

ADAM	Automated Drafting and Machining
AR	Augmented Reality (rozšířená realita)
CAD	Computer-Aided Design
FXAA	Fast Approximate Anti-Aliasing
MSAA	Multisample Anti-Aliasing
MTL	Material Template Library
SSAA	Supersample Anti-Aliasing
VR	Virtual Reality (virtuální realita)

Úvod

Počítačová grafika je obor, který se na první pohled může zdát složitý a těžko pochopitelný. Lidé obvykle vědí, že 3D scény vznikají z trojúhelníků, ale už nerozumí tomu, co se s nimi děje, než se objeví na obrazovce. Proto jsem chtěl krok za krokem pochopit, jak se z trojúhelníku popsaného několika souřadnicemi stane finální obraz složený z pixelů.

Grafické technologie se dnes využívají v mnoha různých oblastech. Nejčastěji si ji lidé spojují s filmy a počítačovými hrami, ale využívá se i v jiných oborech. Pomáhá architektům při navrhování budov, lékařům při zobrazování výsledků vyšetření a vědcům při práci s daty. Přestože se jednotlivé oblasti liší, princip zůstává podobný. Hlavní úkol spočívá v tom, že se matematický popis objektů musí změnit na obraz, který člověk dokáže vnímat.

V současnosti je k dispozici řada nástrojů, které práci s grafikou výrazně usnadňují. Mezi nejznámější patří například knihovny OpenGL, DirectX a WebGL. Díky nim lze vytvářet i složité scény bez nutnosti řešit každý detail vykreslovacího procesu. Zmíněné technologie šetří čas a umožňují zaměřit se především na obsah a vizuální stránku programu. Při využití uvedených knihoven však zůstávají základní kroky vykreslování skryté.

V rámci práce se tedy zaměřím na vytvoření jednoduchého programu pro vykreslování 3D objektů bez použití hotových knihoven. Budu pracovat pouze s HTML Canvas v 2D kontextu, protože JavaScript je pro mě nejbližší jazyk a zároveň jde o prostředí snadno dostupné každému, kdo se chce s touto problematikou seznámit. Výsledné řešení ale nebude omezeno na použitou technologii, protože stejné principy lze použít v jakémkoli jiném prostředí, kde je možné přímo pracovat s pixely na obrazovce.

Pro dané téma jsem se rozhodl, protože se o grafiku zajímám už delší dobu a chtěl jsem si prakticky vyzkoušet, co se skrývá za fungováním grafických programů. Práce má ukázat, že základním principům lze snadno porozumět a že i bez složitých knihoven lze vytvořit jednoduchý 3D program.

1 Teoretická část

V teoretické části je nejprve nastíněn historický vývoj programů pro trojrozměrnou grafiku. Následně jsou popsány základní principy potřebné při tvorbě programu pro práci s trojrozměrným prostředím.

1.1 Historie 3D vykreslování

3D vykreslování je proces tvorby digitálních obrazů a animací z trojrozměrných modelů pomocí specializovaného softwaru. Uplatňuje se v architektuře, filmech, videohrách či virtuální realitě a umožňuje prezentovat projekt v jeho finální vizuální podobě. (CAD Evangelist, 2023)

1.1.1 Rané počátky

První programy pro vykreslování 3D modelů začaly vznikat v 60. letech minulého století. Tehdejší počítače měly omezený výkon, zpracování dat trvalo dlouhou dobu a k dispozici pro daný účel bylo jen malé množství aplikací. První program pro 3D vykreslování vytvořil Ivan Sutherland v roce 1962 během svého studia na Massachusetts Institute of Technology. Program dostal název Sketchpad a stal se základem pro dnešní Computer-Aided Design (CAD) aplikace. Následující technologický vývoj se snažil o zvyšování výkonu a zdokonalování zobrazovacích metod s cílem rozšíření dostupnosti programů pro 3D vykreslování a zvýšit kvalitu jejich zobrazení. (CAD Evangelist, 2023)

1.1.2 Navazující vývoj

Na pokroky šedesátých let ve vykreslování modelů navázal v průběhu sedmdesátých let Edwin Catmull, který vyvinul postup k mapování pixelů textur na trojrozměrné objekty. V téže době vytvořil Martin Newell slavný model „Utah Teapot“ pomocí Bézierových křivek, který se stal jedním z nejznámějších objektů v počítačové grafice. Firmy si navíc začaly vyvíjet vlastní pokročilejší programy, jako byl například systém Automated Drafting and Machining, zkráceně ADAM. (CAD Evangelist, 2023)

V osmdesátých letech se technologie 3D vykreslování rozvíjela velikou rychlostí. CAD se díky osobním počítačům masově rozšířil do průmyslu, na trh přišel AutoCAD, Pixar představil svůj software RenderMan a ray tracing prošel významnými vylepšeními. S rostoucím využitím 3D vykreslování se nové technologie začaly prosazovat ve filmech, herním průmyslu a později i v domácnostech. (CAD Evangelist, 2023)

1.1.3 Současné technologie

V 90. letech se díky programům jako Maya či 3ds Max stalo 3D modelování a renderování mnohem dostupnější a CAD se prosadil jako průmyslový standard. Levnější technologie umožnily vstup do oboru většímu množství tvůrců, a současně se na trhu začala objevovat nová technologie 3D tisku. Během své krátké existence se 3D vykreslování posunulo od vizualizace jednoduchých tvarů až k dnešnímu využití v rozšířené realitě (AR) a virtuální realitě (VR) a stalo se nezbytnou technologií v mnoha odvětvích. (CAD Evangelist, 2023)

1.2 Struktura OBJ a MTL souborů

OBJ je formát pro popis tvaru 3D modelů, který může zároveň odkazovat na barvy a textury uložené v doplňkovém souboru MTL s popisem materiálů. Oba formáty, vyvinuté společností Wavefront Technologies, však neumožňují ukládání animací ani jiných složitějších dat, takže při jejich potřebě je nutné použít jiný formát. (Chakravorty, 2023)

1.2.1 Přehled OBJ souboru

Jde o otevřený a snadno čitelný formát pro ukládání 3D modelů, který uchovává informace prostřednictvím souřadnic vrcholů a dalších údajů popisujících jeho povrch. Díky své přehledné struktuře a malé datové velikosti je široce podporován a umožňuje spolehlivé zobrazení modelů napříč různými programy. (Chakravorty, 2023)

Každý řádek souboru začíná klíčovým slovem určujícím typ obsažené informace, za nímž ihned následují příslušné hodnoty oddělené mezerami. Jednotlivé řádky jsou od sebe odděleny znakem konce řádku. (Chakravorty, 2023)

Tabulka 1: Význam řádků v OBJ

Klíčové slovo	Hodnoty	Popis
#	text	Řádek je komentář, a je tedy programem ignorován.
v	x y z	Určuje vrchol pomocí souřadnic x, y, z.
vn	x y z	Určuje normálový vektor pomocí položek x, y, z.
vt	u v [w]	U a V jsou souřadnice na textuře od 0 do 1, W je volitelný doplňkový parametr.
f	v1[/vt1][/vn1] v2 v3 …	Vytváří plochu z definovaných vrcholů a volitelně k nim můžou být přiřazeny textury a normály oddělené lomítkem.
g	název	Seskupuje plochy do skupin dle názvu.
mtlib	název.mtl	Odkaz na soubor obsahující příslušné materiály.
usemtl	název	Určuje materiál, který se použije pro následující plochy.

Zdroj: Chakravorty, 2023

Klíčové slovo *f* při definici ploch odkazuje na příslušné prvky podle jejich pořadí v souboru s číslováním od 1 a výsledný počet vrcholů se určuje podle počtu uvedených souřadnic. Například řádek f 1 2 3 4 5 tvoří pětiúhelník definovaný prvními pěti vrcholy v souboru. (Chakravorty, 2023)

Klíčová slova *usemtl* a *g* se vztahují na všechny následující výskyty *f* a zůstávají v platnosti, dokud nejsou nahrazena novým výskytem stejného klíčového slova. (Chakravorty, 2023)

1.2.2 Přehled MTL souboru

Soubor MTL obsahuje informace o materiálech používaných v OBJ modelu, jako jsou barvy, textury a další vlastnosti povrchu. Stejně jako u OBJ souborů nese každý řádek MTL souboru jednu informaci, přičemž hodnoty jsou odděleny mezerami. Nový materiál se začíná definovat příkazem *newmtl* a trvá až do dalšího výskytu daného příkazu. Všechny řádky mezi těmito dvěma příkazy náleží k aktuálnímu materiálu. (Chakravorty, 2023)

Tabulka 2: Význam řádků v MTL

Klíčové slovo	Hodnoty	Popis
newmtl	název	Začátek nového materiálu s uvedeným názvem.
Ka	R G B	Nastavuje ambientní barvu materiálu.
Kd	R G B	Nastavuje difuzní barvu materiálu (hlavní barva ploch).
d	hodnota	Určuje průhlednost materiálu, kde 0 je plně průhledný a 1 je neprůhledný.
illum	hodnota (celé číslo)	Určuje způsob osvětlení materiálu dle předem určených předvoleb.
map_Ka / map_Kd	obrázek.jpeg /.png /…	Definuje texturu ze souboru s daným názvem.

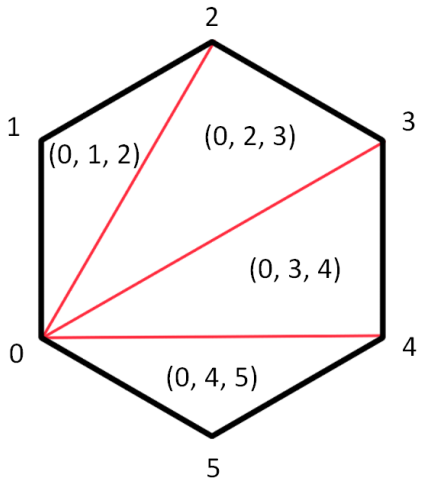
Zdroj: Chakravorty, 2023

Hodnoty R, G a B, stejně jako hodnota pro průhlednost, se v MTL souboru u klíčových slov *Ka* a *Kd* zapisují jako desetinná čísla v rozmezí 0 až 1. (Chakravorty, 2023)

1.3 Rozdělení ploch na trojúhelníky

Jak bylo uvedeno v předchozí části, plochy v OBJ souboru nemusí být tvořeny pouze trojúhelníky, ale i složitějšími tvary (Chakravorty, 2023). Složitější tvary lze rozložit pomocí triangulace, která je rozdělí na trojúhelníky tak, aby se nezměnil původní tvar objektu (Reid, 2017).

Nejjednodušším způsobem, jak rozdělit složitější tvar na trojúhelníky, je pomocí fan triangulation (v doslovném překladu „vějířová triangulace“). Jednoduchost je dána převodem tvarů s více než třemi vrcholy na trojúhelníky bez nutnosti vytvářet další vrcholy. Postup spočívá ve zvolení jednoho výchozího vrcholu, který se postupně propojuje s dvojicemi sousedních vrcholů. Tím vznikají trojúhelníky uspořádané do tvaru vějíře, jak ukazuje obrázek 1, podle něhož algoritmus získal svůj název. (Reid, 2017)



Obrázek 1: Ukázka fungování Fan triangulation
Zdroj: vlastní zpracování

1.4 Rotace ve 3D prostoru

Nejsnazší metodou rotace v 3D prostoru je pomocí rotačních matic, které umožňují otáčení kolem jednotlivých os X, Y a Z o zvolený úhel. K provedení rotace kolem vybrané osy je potřeba vždy spočítat odpovídající rotační matici. (Burzynski, 2023)

1.4.1 Rotace po ose X

Rotaci kolem osy X pro souřadnice x, y a z popisuje následující matice:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(úhel) & -\sin(úhel) \\ 0 & \sin(úhel) & \cos(úhel) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (1)$$

kde x, y a z jsou původní souřadnice bodu ve 3D prostoru, úhel určuje velikost rotace kolem osy X, a x‘, y‘ a z‘ jsou výsledné souřadnice otočeného bodu. (Burzynski, 2023)

1.4.2 Rotace po ose Y

Rotaci kolem osy Y pro souřadnice x, y a z popisuje následující matice:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(úhel) & 0 & \sin(úhel) \\ 0 & 1 & 0 \\ -\sin(úhel) & 0 & \cos(úhel) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (2)$$

kde x, y a z jsou původní souřadnice bodu ve 3D prostoru, úhel určuje velikost rotace kolem osy Y, a x‘, y‘ a z‘ jsou výsledné souřadnice otočeného bodu. (Burzynski, 2023)

1.4.3 Rotace po ose Z

Rotaci kolem osy Z pro souřadnice x, y a z popisuje následující matice:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos(\acute{u}hel) & -\sin(\acute{u}hel) & 0 \\ \sin(\acute{u}hel) & \cos(\acute{u}hel) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (3)$$

kde x, y a z jsou původní souřadnice bodu ve 3D prostoru, úhel určuje velikost rotace kolem osy Z, a x‘, y‘ a z‘ jsou výsledné souřadnice otočeného bodu. (Burzynski, 2023)

1.4.4 Pořadí rotací

Při provádění více rotací najednou ovlivňuje výslednou orientaci pořadí, v jakém se rotační matice počítají, protože každá další rotace navazuje na polohu os, která byla změněna předchozími rotacemi. Změna pořadí tedy vede k odlišnému výsledku, i když úhly rotace zůstanou stejné, například výsledná rotace kolem os v pořadí X, Y a Z je jiná než rotace kolem os v pořadí Y, Z a X. (Corke, 2018)

1.4.5 Gimbal lock

Gimbal lock nastává, když se dvě rotační osy srovnají do stejného směru, což znemožní otáčení kolem jedné z nich. Místo tří os otáčení tak zbývají pouze dvě. Problém lze odstranit například použitím kvaternionů, které netrpí gimbal lockem a spolehlivě zabraňují ztrátě stupně volnosti při rotacích, ale pro začátečníky mohou být méně srozumitelné. (Ena, 2022)

1.5 Perspektivní promítání

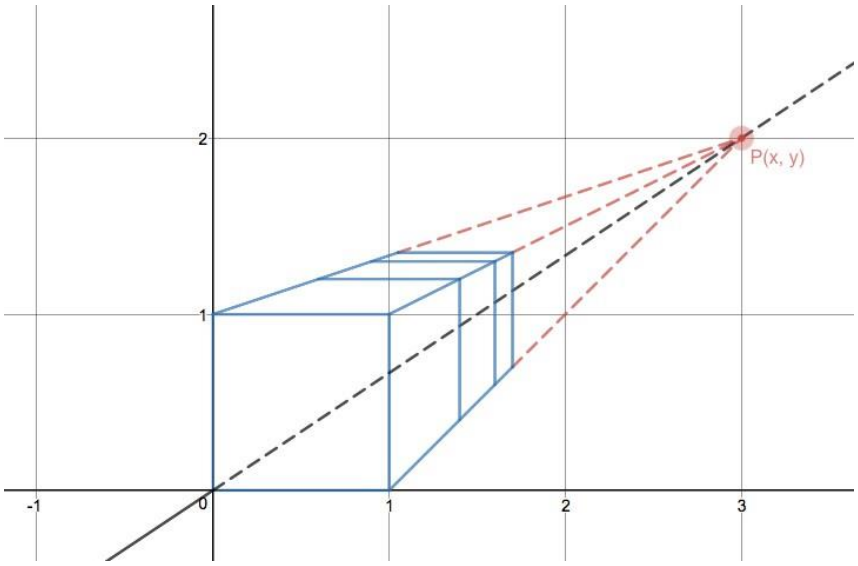
Perspektivní promítání je metoda převodu trojrozměrné scény na plochý obraz tak, aby působil přirozeně. Všechny pomyslné linie, podle nichž se body zobrazují, míří k jedinému bodu v dálce. Výsledkem je, že se vzdálené objekty zmenšují a blízké působí větší, čímž vzniká dojem hloubky a prostoru podobnou skutečnému pohledu na okolí, jak je ukázáno na obrázku 2. (Kawalpreet, 2025)

Perspektivní promítání pro bod ve 3D prostoru se vypočítá následujícím způsobem:

$$x' = x * \left(\frac{d}{z+d}\right) \quad (4)$$

$$y' = y * \left(\frac{d}{z+d}\right) \quad (5)$$

kde x, y a z jsou původní souřadnice bodu v 3D prostoru, d je vzdálenost od místa pozorování a x‘, y‘ jsou výsledné souřadnice bodu v 2D prostoru. (Nominal Animal, 2017)



Obrázek 2: Znázornění perspektivního promítání směřujícího do jednoho bodu
Zdroj: Auggie, 2017

1.6 Výpočet směru plochy ve 3D prostoru

K určení směru plochy ve 3D prostoru slouží normála, tedy vektor kolmý k dané ploše, který udává její orientaci v prostoru. V počítačové grafice se normály používají pro zpracování zobrazení osvětlení, stínů a jiných efektů. Směr normály je navíc určen tak, aby směřoval ven z objektu (Baker, nedatováno). To umožňuje rozlišovat viditelné a skryté strany ploch, což se označuje jako back face culling. (Frantz, 2006)

1.6.1 Výpočet normálového vektoru

Pro samotný výpočet normály stačí spočítat vektorový součin dvou hran trojúhelníku. Pro výpočet hran A, B slouží vzorce:

$$\mathrm{A} = \mathrm{b2} - \mathrm{b1} \tag{6}$$

$$\mathrm{B} = \mathrm{b3} - \mathrm{b1} \tag{7}$$

kde b1, b2 a b3 jsou body trojúhelníku a A, B jsou výsledné hrany trojúhelníku. (Im, 2017)

Pro výpočet normály je nutné spočítat rozdíl souřadnic na každé ose zvlášť. Například složka Ax se spočítá jako rozdíl souřadnice X bodu b2 a souřadnice X bodu b1:

$$Ax = \mathrm{b2}x - b1x \tag{8}$$

kde b1x a b2x jsou souřadnice X bodů b1 a b2 trojúhelníku a Ax je část hrany A na ose X. (Im, 2017)

Po určení všech složek obou hran se normálový vektor Vx, Vy a Vz vypočítá podle následujících vzorců:

$$\mathrm{Vx} = \mathrm{Ay} * \mathrm{Bz} - \mathrm{Az} * \mathrm{By} \tag{9}$$

$$\mathrm{Vy} = \mathrm{Az} * \mathrm{Bx} - \mathrm{Ax} * \mathrm{Bz} \tag{10}$$

$$\mathrm{Vz} = \mathrm{Ax} * \mathrm{By} - \mathrm{Ay} * \mathrm{Bx} \tag{11}$$

kde Ax, Ay a Az jsou části hrany A, Bx, By a Bz jsou části hrany B a Vx, Vy a Vz jsou výsledný normálový vektor. (Baker, nedatováno)

1.6.2 Normalizace vektoru

Pro další výpočty je však nutné výsledný normálový vektor ještě normalizovat (Baker, nedatováno). Normalizace znamená, že z vektoru, který uchovává jak směr, tak velikost, se odstraní informace o velikosti a ponechá se pouze informace o směru. Výsledkem je jednotkový vektor, využívaný například při výpočtu skalárního součinu. (Ellison, 2023)

Pro normalizaci vektoru je nejprve nutné znát jeho délku, která se vypočítá pomocí vzorce:

$$\text{délka} = \sqrt{Vx * Vx + Vy * Vy + Vz * Vz} \tag{12}$$

kde Vx, Vy a Vz jsou části normálového vektoru a délka je výsledná délka normálového vektoru. (Ellison, 2023)

Po zjištění délky vektoru se výsledný normalizovaný normálový vektor Vx‘, Vy‘ a Vz‘ získá vydělením každé složky jeho délkou:

$$\text{Vx}' = \frac{\text{Vx}}{\text{délka}} \tag{13}$$

$$\text{Vy}' = \frac{\text{Vy}}{\text{délka}} \tag{14}$$

$$\text{Vz}' = \frac{\text{Vz}}{\text{délka}} \tag{15}$$

kde Vx, Vy a Vz jsou původní části normálového vektoru, délka je jeho vypočítaná velikost a Vx‘, Vy‘ a Vz‘ tvoří výsledný jednotkový vektor. (Ellison, 2023)

1.6.3 Skalární součin

Skalární součin (anglicky dot product) určuje, jak moc jsou dva vektory nasměrovány stejným směrem. Výsledkem je číslo, které závisí na délkách obou vektorů a na úhlu mezi nimi. (Sheldon, 2022)

Hodnotu skalárního součinu lze získat podle pomocí vzorce

$$\text{A} \bullet \text{B} = (Ax * Bx) + (Ay * By) + (Az * Bz) \tag{16}$$

kde Ax, Ay a Az jsou hodnoty prvního vektoru a Bx, By a Bz jsou hodnoty druhého vektoru. (Sheldon, 2022)

1.7 Afinní zobrazení

Afinní zobrazení je zobrazení, které zachovává body ležící na jedné přímce i poměry vzdáleností mezi nimi. Používá k tomu různé operace, jako je například posunutí (translation), otočení (rotation), zvětšení (dilation), zkosení (shear), a jejich libovolné kombinace. Na rozdíl od jiných zobrazení nemusí afinní zobrazení zachovávat úhly ani délku, a proto lze libovolný trojúhelník převést na jakýchkoliv jiný. (Weisstein, 2004)

Pro výpočet afinní transformace slouží následující vzorce:

$$A = \frac{\big((X2 - X3) * (V1 - V2)\big) - \big((X1 - X2) * (V2 - V3)\big)}{\big((U2 - U3) * (V1 - V2)\big) - \big((U1 - U2) * (V2 - V3)\big)} \quad (17)$$

$$B = \frac{\big((Y2 - Y3) * (V1 - V2)\big) - \big((Y1 - Y2) * (V2 - V3)\big)}{\big((U2 - U3) * (V1 - V2)\big) - \big((U1 - U2) * (V2 - V3)\big)} \quad (18)$$

$$C = \frac{\big((X2 - X3) * (U1 - U2)\big) - \big((X1 - X2) * (U2 - U3)\big)}{\big((V2 - V3) * (U1 - U2)\big) - \big((V1 - V2) * (U2 - U3)\big)} \quad (19)$$

$$D = \frac{\big((Y2 - Y3) * (U1 - U2)\big) - \big((Y1 - Y2) * (U2 - U3)\big)}{\big((V2 - V3) * (U1 - U2)\big) - \big((V1 - V2) * (U2 - U3)\big)} \quad (20)$$

$$E = X1 - (U1 * A) - (V1 * C) \quad (21)$$

$$F = Y1 - (U1 * B) - (V1 * D) \quad (22)$$

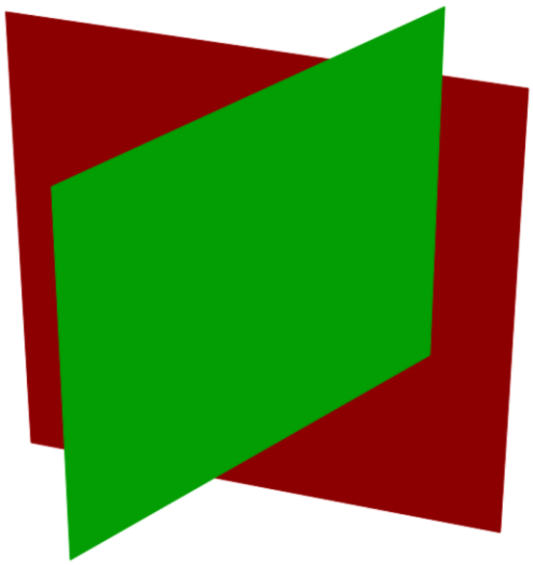
kde Ux a Vx představují původní souřadnice bodů ve 2D prostoru, Xx a Yx jsou cílové souřadnice, na které má být trojúhelník transformován, a A–F označují parametry výsledného afinního zobrazení. (Borgen, 2018)

1.8 Určování viditelných ploch

Při zobrazování trojrozměrné scény je nutné určit, které části zůstanou viditelné a které jsou zakryté jinými. Algoritmy pro odstranění skrytých ploch zajišťují, aby se na obrazovce zobrazily jen ty části objektů, které jsou z daného pohledu skutečně vidět. (TutorialsPoint, nedatováno)

1.8.1 Malířův algoritmus (depth sorting)

Malířův algoritmus vykresluje objekty od nejvzdálenějších k nejbližším, čímž řeší problém viditelnosti podobně jako malíř, který nejprve začne kreslit vzdálené objekty a postupně přidává vrstvy směrem k popředí. Plochy se seřadí podle hloubky a postupně překrývají, což je jednoduché na implementaci, ale může být neefektivní, protože často zpracovává i části, které nakonec nejsou vidět. Problémem jsou i případy složitého překrývání nebo protínání ploch, jak je vidět na obrázku 3, kdy algoritmus nemusí dát správný výsledek, a také vyšší nároky na výkon kvůli třídění a opakovanému vykreslování. (Griffin, 2022)



Obrázek 3: Ukázka chybného vykreslení protínajících se ploch v malířově algoritmu
Zdroj: vlastní zpracování

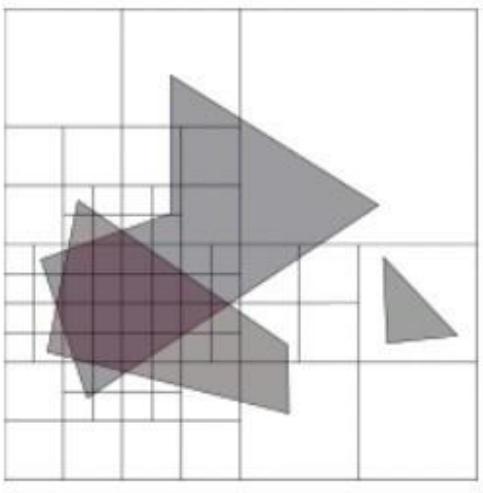
1.8.2 Z-buffer (depth buffer)

Z-buffer si pro každý pixel pamatuje, jak daleko je od kamery. Pokud je nový bod blíže, aktualizuje se barva i hloubka, jinak se ignoruje. Díky fungování Z-bufferu se zobrazí jen viditelné plochy bez nutnosti třídit objekty, což usnadňuje práci i u složitých scén a eliminuje problémy, se kterými si jiné metody hůře poradí. Algoritmus je rychlý a univerzální, ale může vykazovat problémy s přesností, například s Z-fightingem, a navíc vyžaduje dostatek paměti pro uložení dat o všech pixelech. (Yadav, 2025)

Z-buffer si ale neporadí s průhlednými objekty, protože ukládá jen jednu hloubku na pixel. Průhlednost se zde řeší pomocí rozšíření Z-bufferu zvaného A-buffer, který si pro každý pixel ukládá více informací, takže dokáže správně složit průhledné vrstvy. Nevýhodou je však vyšší paměťová náročnost a větší zátěž na výkon. (Bansal, 2021)

1.8.3 Area-subdivision (Warnockův algoritmus)

Warnockův algoritmus řeší viditelnost povrchů tak, že zobrazovanou oblast opakovaně rozděluje na menší části, dokud nejsou dost jednoduché na přímé vykreslení. V každém dílčím výřezu se vyberou plochy, které do něj spadají, vyloučí se ty ležící mimo a následně se provede kontrola viditelnosti. Pokud nelze rozhodnout, oblast se znovu rozdělí, jak ukazuje obrázek 4. Algoritmus zvládá i složité scény, ale jeho rychlost výrazně závisí na počtu pixelů a ploch. (BrainKart, 2016)



Obrázek 4: Ukázka fungování Warnockova algoritmu
Zdroj: BrainKart, 2016

1.8.4 Scan-line

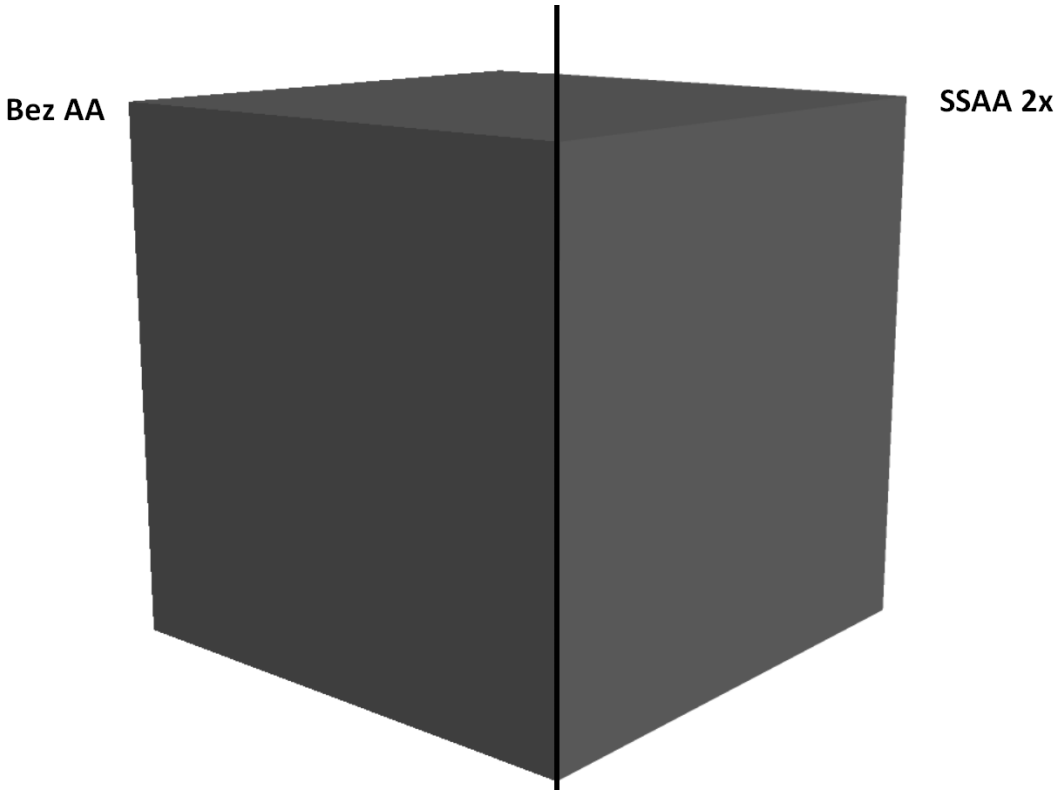
Metoda Scan-line určuje viditelné plochy po jednotlivých vodorovných řádcích obrazu. Pro každý řádek se vyberou plochy, které jej protínají, a pomocí tabulky hran a tabulky ploch se určí jejich pořadí a vlastnosti. Seznam hran seřazený podle souřadnice x sleduje, zda se aktuální pozice nachází uvnitř plochy, a hloubka se počítá jen tam, kde se překrývá více povrchů. (TutorialsPoint, nedatováno)

1.9 Antialiasing

Antialiasing je technika používaná k omezení zkreslení, které vzniká při nedostatečném vzorkování obrazu. Takový jev se nazývá aliasing, a projevuje se například zubatými hranami na šikmých liniích. Antialiasing pomáhá vyhladit uvedené artefakty, díky čemuž obraz působí čistěji. (Evanson, 2021)

1.9.1 Supersample Anti-Aliasing (SSAA)

SSAA je metoda, při níž se vykresluje scéna ve vyšším rozlišení, než je cílové, a poté se zmenší zpátky na požadovanou velikost. Tím účinně vyhlazuje hrany a odstraňuje aliasing, protože jemnější detaily se při zmenšení zprůměrují, jak je ukázáno na obrázku 5. Nevýhodou je ale vysoká výpočetní náročnost, která může výrazně snížit výkon. (Evanson, 2021)



Obrázek 5: Ukázka fungování SSAA
Zdroj: vlastní zpracování

1.9.2 Multisample Anti-Aliasing (MSAA)

MSAA je efektivnější metoda, která oproti SSAA provádí více vzorkování uvnitř jednoho pixelu pouze na okrajích plochy. Tím dosáhne nižšího dopadu na výkon než SSAA, ale na druhou stranu neovlivňuje textury uvnitř ploch, takže při pohybu plochy může být rozdíl mezi vyhlazenými hranami a zbytkem plochy viditelnější. (Evanson, 2021)

1.9.3 Fast Approximate Anti-Aliasing (FXAA)

FXAA funguje jako filtr, který analyzuje hotový obraz a rozmazává pixely na hranách s velkým kontrastem. Je velmi rychlý a nenáročný na výkon, proto je oblíbený například ve hrách. Nevýhodou je nižší přesnost než v předchozích dvou algoritmech, a také že může způsobit rozostření jemných detailů. (Evanson, 2021)

2 Praktická část

Praktická část práce navazuje na předchozí kapitoly a ukazuje využití popsaných principů při tvorbě programu pro vykreslování 3D objektů v HTML Canvas pomocí JavaScriptu.

2.1 Import a zpracování 3D modelu

Před zahájením práce s 3D objekty je nutné nahrát data modelu ze souboru. Následně je potřeba správně porozumět a zapsat data obsažená v souborech, aby mohla aplikace správně fungovat.

2.1.1 Získání souborů od uživatele

Pro nahrávání souborů v HTML slouží vstupní prvek typu file. Kvůli lepší možnosti úpravy vzhledu je vstupní prvek skrytý a nahrazen tlačítkem Import files. Po stisknutí tlačítka JavaScript aktivuje vstupní prvek a otevře dialogové okno pro výběr souborů. Vstupní prvek je zde nastaven pro výběr více souborů s příponami .jpg, .jpeg, .png, .obj a .mtl.

```
<p class="menuLine">
    <button id="importButton" style="background-color: green; color: white; padding: 10px; border: none;
        cursor: pointer; width: 100%;">Import files</button>
    <input type="file" id="import" accept=".jpg,.jpeg,.png,.obj,.mtl" multiple style="display: none;">
</p>


<script>
                document.getElementById('importButton').addEventListener('click', function () {
                    document.getElementById('import').click();
                });
</script>
```

Obrázek 6: získávání souborů od uživatele

Zdroj: vlastní zpracování

První rozdíl oproti běžnému programu spočívá v tom, že běží v prostředí webového prohlížeče. Z bezpečnostních důvodů tedy není možné přistupovat přímo k souborům uloženým na počítači (Peoples, 2019). Pokud tedy jakýkoliv soubor odkazuje na jiný prostřednictvím cesty v adresáři, lze pracovat pouze s daty, která jsou přímo nahrána uživatelem přes vstupní prvek.

Po výběru souborů se spustí funkce, která získá seznam nahraných položek a zjistí jejich počet. Následně se kontroluje, zda je přítomen právě jeden soubor s příponou obj, jinak se zpracování předběžně ukončí s chybovým hlášením. Pokud je podmínka splněna, vymažou se předchozí data a aplikace se připraví na nové zpracování. Poté se jednotlivé soubory projdou podle přípony a jsou předány příslušným funkcím. Vybrané obrázky jdou k funkci handleImage, soubor obj funkci handleOBJ a soubory mtl funkci handleMTL, zatímco ostatní soubory jsou ignorovány.

```
upload.addEventListener('change', function (event) {
    const files = event.target.files;
    filesLength = files.length;

    if (filesLength === 0) return;
    filesDone = 0;
    let objFileCount = 0;

    for (let i = 0; i < files.length; i++) {
        if (files[i].name.endsWith('.obj')) {
            objFileCount++;
        }
    }
    if (objFileCount === 0) {
        alert("please select a .obj file");
        return;
    }
    else if (objFileCount > 1) {
        alert("please select exactly one .obj file");
        return;
    }

    clearAll();
    containsImage = 0;
    isImportDone = 0;


    for (let i = 0; i < files.length; i++) {
        const file = files[i];
        const fileExtension = file.name.split('.').pop().toLowerCase();

        if (['jpg', 'jpeg', 'png'].includes(fileExtension)) handleImage(file);
        else if (fileExtension === 'obj') handleOBJ(file);
        else if (fileExtension === 'mtl') handleMTL(file);
        else newFileDone();
    }
});
```

Obrázek 7: Třídění souborů

Zdroj: vlastní zpracování

2.1.2 Pozastavení chodu programu při importu

Čtení souborů pomocí FileReader pracuje asynchronně, takže program sám o sobě nečeká na dokončení načítání souborů (Mozilla Foundation, 2025). Aby nedošlo k práci s nekompletními daty, je použita pomocná proměnná isImportDone, která je při zahájení importu nastavena na hodnotu 0. Jakmile se počet zpracovaných souborů stejný jako počet celkového počtu souborů ze vstupu, zmíněná proměnná se nastaví zpět na hodnotu 1, čímž se označí dokončení zpracování všech souborů.

```
function handleImage(file)
{
    newFileDone();
}

function handleOBJ(file)
{
    newFileDone();
}

function handleMTL(file)
{
    newFileDone();
}


function newFileDone()
{
    filesDone++;
    if (filesDone === filesLength)
    {
        isImportDone = 1;
    }
}
```

Obrázek 8: Pozastavení chodu programu při importu

Zdroj: vlastní zpracování

2.1.3 Čtení obrázků ze souboru

Pokud je zavolána funkce handleImage, vytvoří se objekt FileReader určený k načtení obsahu souboru. Metoda readAsDataURL načte obsah souboru a převede ho do formátu datového URL, takže jej lze přímo použít jako zdroj pro objekt Image (JavaScript Tutorial, 2021). Po dokončení načítání se spustí událost, v níž je vytvořen nový objekt Image. Jakmile je obrázek plně načten, je uložen do pole imagesArray spolu s názvem souboru. Na závěr se zavolá funkce newFileDone, která aktualizuje počet dokončených souborů.

```
function handleImage(file) {
    const reader = new FileReader();
    reader.addEventListener("load", function () {
        const image = new Image();
        image.src = reader.result;
        image.onload = function () {
            imagesArray.push({ name: file.name, image: image });
            newFileDone();
        };
    });

    reader.readAsDataURL(file);
}
```

Obrázek 9: Načítání obrázku

Zdroj: vlastní zpracování

2.1.4 Čtení obsahu OBJ souboru

Při zpracování souboru OBJ je opět využit objekt FileReader, který pomocí metody readAsText načte celý obsah souboru do textové podoby (Mozilla Foundation, 2025). Načtený text je následně rozdělen na jednotlivé řádky a ty jsou dále zpracovány podle klíčových slov uvedených v tabulce 1. Každý řádek je převeden na potřebné hodnoty a uložen do příslušných polí, přičemž se kontroluje jejich správnost a úplnost. U ploch je přidáno počítadlo, které slouží k určení, na které obličeje se vztahuje definice materiálu. Jelikož jsou složitější plochy později rozdělovány na trojúhelníky, je tomuto rozdělení přizpůsobeno i počítadlo, aby bylo možné správně přiřadit materiály.

```
function handleOBJ(file) {
    const reader = new FileReader();
    let lineCounter = 0;
    reader.addEventListener("load", function () {
        const lines = reader.result.split('\n');
        originalOBJfileName = file.name;
        originalOBJfile = reader.result;
        for (let line = 0; line < lines.length; line++) {
            const parts = lines[line].trim().split(/\s+/);
            const keyword = parts[0];
            lineCounter++;
            let arrToPush = [];

            switch (keyword) {
                case 'v':
                    if (parts.length !== 4) {
                        alert("Incomplete vertex information on line " + lineCounter);
                        clearAll(); isImportDone = 1; return;
                    }
                    for (let i = 1; i < parts.length; i++) {
                        if (isNaN(parts[i])) {
                            alert("Vertex doesn't use valid number on line " + lineCounter);
                            clearAll(); isImportDone = 1; return;
                        }
                        arrToPush.push(Number(parts[i]));
                    }
                    vertices.push(arrToPush);
                    break;
                case 'f':
                    if (parts.length < 4) {
                        alert("Incomplete face information on line " + lineCounter);
                        clearAll(); isImportDone = 1; return;
                    }
                    for (let i = 1; i < parts.length; i++) {
                        let subParts = parts[i].split("/");
                        let indices = [null, null, null];
                        for (let j = 0; j < subParts.length; j++) {
                            if (subParts[j] !== "") {
                                if (isNaN(subParts[j])) {
                                    alert("Face doesn't use valid number on line " + lineCounter);
                                    clearAll(); isImportDone = 1; return;
                                }
                                indices[j] = Number(subParts[j]);
                            }
                        }
                        arrToPush.push(indices);
                    }
                    faces.push(arrToPush);
                    faceCounter += 1 + arrToPush.length - 3;
                    break;
                //ostatní případy podobným způsobem podle tabulky 1
            }
        }
        if (!verifyFileInfoLength()) { clearAll(); isImportDone = 1; return; }
        newFileDone();
    });

    reader.readAsText(file);
}
```

Obrázek 10: Zpracování OBJ souboru

Zdroj: vlastní zpracování

Ostatní klíčová slova jsou podobně implementována podle struktury popsané v tabulce 1.

2.1.5 Čtení obsahu MTL souboru

Během zpracování souboru MTL je obsah převeden do textové podoby a rozdělen podle klíčového slova newmtl na bloky představující jednotlivé materiály. Každý blok je dále rozdělen na řádky a ty jsou vyhodnocovány podle klíčových slov uvedených v tabulce 2. Hodnoty se při zpracování kontrolují a v případné nesrovnalosti jsou nahrazeny výchozími hodnotami. Po dokončení je každý materiál uložen do pole, které uchovává všechny vlastnosti, díky čemuž může být nahráno i více souborů MTL současně.

```
function handleMTL(file) {
    const reader = new FileReader();
    reader.addEventListener("load", function () {
        const chunks = reader.result.split('newmtl');
        originalMTLfile.push({ name: file.name, result: reader.result });
        for (let chunk = 0; chunk < chunks.length; chunk++) {
            let material = {};
            const lines = chunks[chunk].split('\n');
            let matNameCheck = 0;

            for (let line = 0; line < lines.length; line++) {
                const parts = lines[line].trim().split(/\s+/);
                const keyword = parts[0];

                if (!matNameCheck) {
                    material.name = keyword;
                    matNameCheck = 1;
                }
                else {
                    let tempArr = [];
                    switch (keyword) {
                        case 'Kd':
                            if (parts.length !== 4) {
                                alert("Incomplete Kd information for material: " + material.name);
                                material.diffuse = [0.500000, 0.500000, 0.500000]; continue;
                            }
                            for (let i = 1; i < parts.length; i++) {
                                if (isNaN(parts[i])) {
                                    alert("Kd doesn't use valid number in material: " + material.name);
                                    tempArr = [0.500000, 0.500000, 0.500000]; break;
                                }
                                tempArr.push(Number(parts[i]));
                            }
                            material.diffuse = tempArr;
                            break;
                        case 'map_Kd':
                            material.imgName = parts.slice(1).join(' ').split(/[\\/]/).pop();
                            containsImage = 1;
                            break;
                        //ostatní případy podobným způsobem podle tabulky 2
                    }
                }
            }
            colorsFromFile.push(material);
        }
        newFileDone();
    });

    reader.readAsText(file);
}
```

Obrázek 11: Zpracování MTL souboru

Zdroj: vlastní zpracování

2.1.6 Úprava souřadnic modelu

Po načtení a zpracování všech souborů je nutné provést ještě další úpravy, aby výsledné objekty odpovídaly požadovanému zobrazení. Kód proto musí provést několik kroků, které zajistí správnou velikost i polohu modelu v prostoru.

Nejprve jsou všechny souřadnice převedeny na hodnoty v rozsahu od -1 do 1. K výpočtu je využita lineární interpolace, která postupně přepočítá každý bod do daného rozsahu podle aktuálně používané minimální a maximální hodnoty v souboru (Cuemath, 2021). Výsledkem je, že se všechny body dostanou do jednotného rozsahu, což usnadňuje další práci se zobrazováním modelu.

```
const flatArray = vertices.slice().flat(Infinity);
const minNum = Math.min(...flatArray);
const maxNum = Math.max(...flatArray);

for (let i = 0; i < vertices.length; i++) {
    for (let j = 0; j < vertices[i].length; j++) {
        vertices[i][j] = -1 + (((vertices[i][j] - minNum) * 2) / (maxNum - minNum));
    }
}
```

Obrázek 12: Převod hodnot do jednotného rozsahu

Zdroj: vlastní zpracování

Lineární interpolaci pro vstupní hodnotu popisuje následující rovnice:

$$x = nMin + \frac{(hodnota - sMin) * (nMax - nMin)}{(sMax - sMin)} \tag{23}$$

kde hodnota je původní vstupní číslo, sMin a sMax představují staré minimum a maximum, nMin a nMax určují nové minimum a maximum, a x je finální převedená hodnota. (Cuemath, 2021)

Následně se vypočítá průměrná velikost objektu ve všech třech osách. Některé modely mohou být úzké a vysoké, jiné naopak široké a nízké. Proto se provede přepočet tak, aby průměrná velikost všech modelů byla podobná. Všechny souřadnice se násobí stanoveným poměrem, čímž se model zvětší nebo zmenší na vhodnou velikost.

```
let minX = Infinity, minY = Infinity, minZ = Infinity;
let maxX = -Infinity, maxY = -Infinity, maxZ = -Infinity;

for (let i = 0; i < vertices.length; i++) {
    const [x, y, z] = vertices[i];

    if (x < minX) minX = x;
    if (x > maxX) maxX = x;

    if (y < minY) minY = y;
    if (y > maxY) maxY = y;

    if (z < minZ) minZ = z;
    if (z > maxZ) maxZ = z;
}

const xVal = maxX - minX;
const yVal = maxY - minY;
const zVal = maxZ - minZ;

let avgVal = (xVal + yVal + zVal) / 3;
if (avgVal === 0) avgVal = 0.01;

const multiplyVal = 2 / avgVal;

for (let i = 0; i < vertices.length; i++) {
    for (let j = 0; j < vertices[i].length; j++) {
        vertices[i][j] *= multiplyVal;
    }
}
```

Obrázek 13: Změna velikosti modelu

Zdroj: vlastní zpracování

Nakonec se provede posun modelu do středu. Nejprve se zjistí průměrná poloha všech bodů na osách X, Y a Z. Průměr ukazuje, kde se nachází aktuální střed objektu (von Gagern, 2014). Poté se od každého bodu odečte hodnota zjištěného středu, čímž se celý model posune tak, aby jeho nový střed ležel v bodě 0,0,0 (Wolfe, 2016).

```
let sumX = 0;
let sumY = 0;
let sumZ = 0;

for (let i = 0; i < vertices.length; i++) {
    sumX += vertices[i][0];
    sumY += vertices[i][1];
    sumZ += vertices[i][2];
}

const centerX = sumX / vertices.length;
const centerY = sumY / vertices.length;
const centerZ = sumZ / vertices.length;

for (let i = 0; i < vertices.length; i++) {
    vertices[i][0] -= centerX;
    vertices[i][1] -= centerY;
    vertices[i][2] -= centerZ;
}
```

Obrázek 14: Posun modelu na střed

Zdroj: vlastní zpracování

2.1.7 Rozdělení modelu na trojúhelníky

Následně je nutné provést převod všech ploch na trojúhelníky, aby byla práce s objektem jednodušší. Kód prochází všechny plochy a kontroluje jejich počet vrcholů, jak je popsáno v teoretické části 1.3. Pokud má plocha více než tři vrcholy, je postupně rozdělena na trojúhelníky tak, že se vždy vybere první vrchol a k němu dvojice dalších, čímž vznikne nová trojice. První trojúhelník je přidán samostatně, protože není zahrnut ve smyčce. Plochy, které již mají tři vrcholy, jsou ponechány beze změny. Výsledkem je seznam ploch, kde každá je reprezentována pouze trojúhelníky. (Reid, 2017)

```
function convertFacesToTriangles(faces) {
    let newFaces = [];
    for (let i = 0; i < faces.length; i++) {
        let face = faces[i];
        if (face.length > 3) {
            for (let j = 1; j < face.length - 2; j++) {
                newFaces.push([face[0], face[j + 1], face[j + 2]]);
            }
            newFaces.push([face[0], face[1], face[2]]);
        }
        else newFaces.push(face);
    }
    return newFaces;
}
```

Obrázek 15: Rozdělení modelu na trojúhelníky

Zdroj: vlastní zpracování dle Reid, 2017

2.1.8 Příprava obrázků na pozdější používání

Pro snazší práci s obrázky se připraví jejich výřezy, takže následné vykreslování bude rychlejší, jelikož se bude pracovat s menšími obrázky. Nejprve se projdou všechny materiály ze souboru obj a zjistí se, zda mají přiřazený obrázek. Pokud ano, vyhledají se plochy patřící k danému materiálu a z jejich souřadnic se určí hranice výřezu, přičemž se okraje mírně rozšíří, aby nedošlo k jejich nežádoucímu oříznutí. V prostředí canvas představuje souřadnice 0,0 levý horní roh, zatímco v UV souřadnicích znamená 0,0 levý spodní roh, proto je nutné provést převrácení souřadnic při práci s hodnotami (Slynch, 2018). Pro každý výřez se vytvoří samostatná plocha canvas, do níž se vykreslí odpovídající část obrázku. Výsledné plochy se uloží spolu s identifikátorem a souřadnicemi.

```
function cutOutShapesFromImage() {
    const cutouts = [];
    for (let index = 0; index < matName.length; index++) {
        const material = matName[index];
        const materialName = material.name;
        const materialId = material.id;
        let nextMaterialId;
        if (index + 1 < matName.length) nextMaterialId = matName[index + 1].id;
        else nextMaterialId = faces.length;
        let colorEntry = colorsFromFile.find(entry => entry.name === materialName);

        if (colorEntry) {
            let imgName = colorEntry.imgName || null;
            let imageEntry = imagesArray.find(entry => entry.name === imgName);
            if (imageEntry) {
                const image = imageEntry.image;
                let relevantFaces = faces.filter(
                    (face, currentId) => currentId >= materialId && currentId < nextMaterialId);

                for (let i = 0; i < relevantFaces.length; i++) {
                    const face = relevantFaces[i];
                    let minX = Infinity, minY = Infinity, maxX = -Infinity, maxY = -Infinity;
                    let tempArrU = [];
                    let tempArrV = [];

                    for (let i = 0; i < face.length; i++) {
                        const vertex = face[i];
                        const textureIndex = vertex[1];
                        if (textureIndex) {
                            const [u, v] = textureCoord[textureIndex - 1];
                            const x = Math.floor(u * image.width);
                            const y = Math.floor((1 - v) * image.height);
                            tempArrU.push(x);
                            tempArrV.push(y);
                            if (x < minX) minX = x;
                            if (y < minY) minY = y;
                            if (x > maxX) maxX = x;
                            if (y > maxY) maxY = y;
                        }
                    }

                    const cutoutCanvas = document.createElement('canvas');
                    const cutoutCtx = cutoutCanvas.getContext('2d', { willReadFrequently: true });
                    minX -= imageOffset;
                    minY -= imageOffset;
                    maxX += imageOffset;
                    maxY += imageOffset;
                    cutoutCanvas.width = maxX - minX;
                    cutoutCanvas.height = maxY - minY;

                    cutoutCtx.translate(-minX, -minY);
                    cutoutCtx.drawImage(image, 0, 0);

                    if (cutoutCanvas.width !== 0 && cutoutCanvas.height !== 0) {
                        cutouts.push({
                            canvas: cutoutCanvas,
                            faceId: faces.indexOf(face),
                            posU: tempArrU,
                            posV: tempArrV
                        });
                    }
                }
            }
        }
    }
```

Obrázek 16: Rozříznutí obrázku na menší části

Zdroj: vlastní zpracování

Původní záměr byl sloučit všechny výřezy do jedné velké plochy canvas, kde by byly obrázky naskládané za sebou, aby se vždy mohla rychle a jednoduše vzít jen potřebná část. Ukázalo se však, že prohlížeč má omezení velikosti canvas na 32 tisíc pixelů, což u rozsáhlejších modelů způsobovalo nefunkčnost (Thakur, 2019). Proto každý obrázek zůstane na své vlastní canvas. Nakonec se připravené výřezy seřadí podle identifikátoru plochy a uloží do pole, které obsahuje buď odpovídající obrázek, nebo prázdnou hodnotu, aby bylo snadné otestovat, jestli plocha má v sobě obrázek nebo ne.

```
let sortedCutouts = new Array(faces.length).fill(null);
for (let i = 0; i < cutouts.length; i++) {
    const position = cutouts[i];
    if (position.faceId <= sortedCutouts.length) {
        sortedCutouts[position.faceId] = position;
    }
}
if (sortedCutouts.length === 0) {
    imagePositions = [];
}
else {
    imagePositions = sortedCutouts;
}
```

Obrázek 17: Uložení rozřízlých obrázků

Zdroj: vlastní zpracování

2.2 Zpracování uživatelského vstupu

Aby aplikace mohla plnit svůj účel, je nutné reagovat na vstupy od uživatele. Kapitola se zaměřuje na kód, který sleduje pohyb kurzoru, stisknutí tlačítek a další akce uživatele potřebné pro práci s programem.

2.2.1 Vstup z myši

Po stisku tlačítka myši se nejprve ověří, zda není otevřené menu. Pokud je aktivní, zpracování je okamžitě ukončeno. V opačném případě se uloží souřadnice kliknutí a přidá se posluchač pohybu, který sleduje změny polohy myši po celou dobu, kdy je tlačítko drženo.

```
window.addEventListener('mousedown', function (event) {
    if (isMenuOpen) return;

    mouseX = event.clientX;
    mouseY = event.clientY;

    mouseTouchPointX = event.clientX;
    mouseTouchPointY = event.clientY;

    window.addEventListener('mousemove', onMouseMove);
});
```

Obrázek 18: Zjištění stisknutí myši

Zdroj: vlastní zpracování

Při pohybu myši se nejprve určí změna souřadnic oproti předchozí pozici, čímž se zjistí, jak moc se myš pohnula od posledního zaznamenaného pohybu, a poté se zapíšou nové hodnoty (Grams, 2019). Pokud je stisknuta klávesa Shift, pohyb se zpomalí. Levé tlačítko slouží k rotaci objektu, pokud není nastaveno jeho použití pro výběr vrcholů. Úhel směru pohybu je vypočítán na základě posunu myši a je ukládán jako číselná hodnota určující natočení objektu. Pro další práci je nutné ji převést do stupňů. Následně se provede normalizace hodnoty a funkce vyhodnotí, zda je objekt převrácený.

```
function onMouseMove(event) {

    mouseChangeX = event.clientX - mouseX;
    mouseChangeY = event.clientY - mouseY;

    if (IsShiftHeld) {
        mouseChangeX *= 0.25;
        mouseChangeY *= 0.25;
    }

    mouseX = event.clientX;
    mouseY = event.clientY;

    if (event.buttons === 1 && ((enableEditMode && !selectWithLeftClick) || !enableEditMode)) {
        isMouseMoving = 1;

        const normalizedAngleX = ((180 / Math.PI * angleX) % 360 + 360) % 360
        const normalizedAngleZ = ((180 / Math.PI * angleZ) % 360 + 360) % 360

        checkIfUpsideDown(normalizedAngleX, normalizedAngleZ);
    }
}
```

Obrázek 19: Aktualizace pohybu myši

Zdroj: vlastní zpracování

Převod na stupně je proveden podle vztahu:

$$stupně = \frac{180}{\pi} * u \tag{24}$$

kde stupně představují úhel vyjádřený ve stupních a u představuje vstupní hodnotu používanou pro výpočet úhlu. (Małek a Żuławińska, 2015)

Nyní je však ještě nutné provést následnou úpravu pro převedení úhlu do požadovaného rozsahu. Úprava výsledku do intervalu od 0° do 360° je vyjádřena rovnicí:

$$x = (stupně\ mod\ 360 + 360) mod\ 360 \tag{25}$$

kde stupně jsou původní úhel ve stupních v libovolném rozmezí a x jsou výsledné stupně převedené do rozsahu od 0 do 360. (Overby, 2017)

Funkce pro kontrolu převrácení následně určuje, zda se úhel na ose X nebo Z nachází mezi 90 a 270 stupni. Podle výsledku se uloží informace, jestli je objekt převrácený vzhůru nohama.

```
function checkIfUpsideDown(normalizedAngleX, normalizedAngleZ) {
    const xUpside = (normalizedAngleX > 90 && normalizedAngleX < 270);
    const zUpside = (normalizedAngleZ > 90 && normalizedAngleZ < 270);

    if (xUpside ^ zUpside) isUpsideDown = 1;
    else isUpsideDown = 0;
}
```

Obrázek 20: Kontrola převrácení modelu

Zdroj: vlastní zpracování

Rotace normálně probíhá kolem os X a Y, nebo pokud je stisknuto tlačítko Ctrl, tak se otáčí kolem osy Z. Pravé tlačítko místo rotace posouvá objekt po obrazovce podle změny polohy kurzoru.

```
if (IsCtrlHeld) {
    if (mouseChangeX !== 0 && normalizedAngleX > 90 && normalizedAngleX < 270) angleZ -= mouseChangeX * 0.01;
    else if (mouseChangeX !== 0) angleZ += mouseChangeX * 0.01
}

else if (isUpsideDown) {
    if (mouseChangeX !== 0) angleY -= mouseChangeX * 0.01;
    if (mouseChangeY !== 0) angleX -= mouseChangeY * 0.01;
}
else {
    if (mouseChangeX !== 0) angleY += mouseChangeX * 0.01;
    if (mouseChangeY !== 0) angleX -= mouseChangeY * 0.01;
}


if (event.buttons === 2) {
        posX += mouseChangeX;
        posY += mouseChangeY;
}
```

Obrázek 21: Aktualizování úhlů a pozice modelu podle pohybu myši

Zdroj: vlastní zpracování

Při použití kolečka myši se mění velikost objektu. Posun nahoru měřítko zvětší, posun dolů ho zmenší, ale jen do určité minimální hranice. Nová hodnota se zároveň aktualizuje v menu, aby se hodnoty v něm zobrazovaly správně.

```
const slider = document.getElementById('scaling');
window.addEventListener('wheel', function (event) {
    if (event.deltaY < 0) {
        scaling += 10 + (scaling * 0.1);
        slider.value = scaling;
        isRedrawNeeded = 1;
    }
    else {
        if (scaling > 15) scaling -= 10 + (scaling * 0.1);
        slider.value = scaling;
        isRedrawNeeded = 1;
    }
});
```

Obrázek 22: Změna velikosti modelu

Zdroj: vlastní zpracování

2.2.2 Vstup z klávesnice a úprava chování

Při stisku klávesy Shift nebo Ctrl se uloží informace, že je daná klávesa držena. Při jejich puštění se proměnná vrátí zpět na nulu. U klávesy E se při uvolnění navíc přepne režim úprav, takže ho lze rychle zapínat nebo vypínat pomocí klávesnice.

```
window.addEventListener('keydown', function (event) {
    if (event.key === 'Shift') {
        IsShiftHeld = 1;
    }

    if (event.key === 'Control') {
        IsCtrlHeld = 1;
    }
});

window.addEventListener('keyup', function (event) {
    if (event.key === 'Shift') {
        IsShiftHeld = 0;
    }

    if (event.key === 'Control') {
        IsCtrlHeld = 0;
    }

    if (event.code === 'KeyE') {
        const checkBox = document.getElementById('editMode');
        if (checkBox) checkBox.click();
    }
});
```

Obrázek 23: Zjišťování stisknutých tlačítek na klávesnici

Zdroj: vlastní zpracování

Následně je vhodné upravit chování funkce pinch to zoom na noteboocích. Při běžném použití se totiž posouvá celý obsah stránky, což není žádoucí. Uvedený kód zabrání posunu stránky a zajistí, že se pinch to zoom bude chovat stejně, jako kdyby uživatel používal kolečko myši. (Samuel, 2020)

```
window.addEventListener('wheel', function (e) {
    if (e.ctrlKey) {
        e.preventDefault();
        return;
    }
}, { passive: false });
```

Obrázek 24: Vypnutí pinch to zoom

Zdroj: vlastní zpracování dle Samuel, 2020

2.3 Hlavní smyčka

Pro hlavní smyčku se používá requestAnimationFrame, protože je vhodnější pro animace než klasické časovače jako setTimeout. Synchronizuje se totiž s vykreslováním webového prohlížeče, takže animace jsou plynulejší a zároveň šetří výkon při přepnutí do jiného okna. (OpenReplay Team, 2025)

Podmínka if zajišťuje, že se scéna překresluje jen tehdy, když je to potřeba, například při pohybu myši nebo pokud je programem nastaveno, že je nutné překreslení. Tím se předchází zbytečnému vykreslování v okamžicích, kdy se nic nemění, nebo kdy by to bylo nežádoucí, jako při importu. Jakmile je překreslení dokončeno, proměnná isRedrawNeeded se nastaví zpět na nulu, takže se další vykreslování neprovádí, dokud není znovu vyžádáno.

```
function drawObj(time) {
    menuShouldOBJrotateAutomatically(time);

    if ((isMouseMoving || isRedrawNeeded) && isImportDone) {

        //nové vykreslení modelu při změně stavu

        isRedrawNeeded = 0;
    }
    requestAnimationFrame(drawObj);
}
```

Obrázek 25: Hlavní smyčka

Zdroj: vlastní zpracování

Funkce requestAnimationFrame navíc poskytuje časovou značku (timestamp), která udává čas volání funkce. Z časové značky je spočítán rozdíl mezi jednotlivými voláními. Vypočítaný rozdíl je následně využíván pro provedení rotace objektu, která může probíhat zároveň kolem osy X, osy Y i osy Z. To umožní plynulé a konzistentní otáčení bez ohledu na výkon zařízení. (OpenReplay Team, 2025)

```
let lastTime = null;
function menuShouldOBJrotateAutomatically(now) {
    if (!rotateAroundXaxis && !rotateAroundYaxis && !rotateAroundZaxis) return;

    if (lastTime === null) lastTime = now;
    const time = (now - lastTime) * 0.05;
    lastTime = now;
    const moveAmount = 0.01 * time;

    if (rotateAroundXaxis) {
        angleX += moveAmount;
        isRedrawNeeded = 1;
    }
    if (rotateAroundYaxis) {
        angleY += moveAmount;
        isRedrawNeeded = 1;
    }
    if (rotateAroundZaxis) {
        angleZ += moveAmount * 0.7;
        isRedrawNeeded = 1;
    }
}
```

Obrázek 26: Automatické otáčení modelu

Zdroj: vlastní zpracování dle OpenReplay Team, 2025

2.4 Rotace bodů a skládání do ploch

Celý proces probíhá v hlavní smyčce popsané v kapitole 2.3. Nejprve se připraví jednotlivé vrcholy a podle rotačních matic uvedených v kapitole 1.4 se vypočítají nové souřadnice. Rotace probíhá postupně kolem osy Y, Z a X, výsledné hodnoty se následně ukládají do pole hotových vrcholů. Z různých vyzkoušených možností se uvedená kombinace ukázala jako nejvhodnější, protože nejlépe odpovídala požadovanému chování a zároveň přesunula problém gimbal locku na osu Z, která je v rámci práce využívána pro pohyb objektu nejméně, jelikož manipulace v uvedeném směru vyžaduje stisk dalšího tlačítka (Ena, 2022).

```
rotatedVertices.length = 0;
for (let i = 0; i < vertices.length; i++) {
    const x = vertices[i][0];
    const y = vertices[i][1];
    const z = vertices[i][2];

    //Y
    const newX = x * Math.cos(angleY) + z * Math.sin(angleY);
    const newZ = -x * Math.sin(angleY) + z * Math.cos(angleY);
    //Z
    const anotherX = newX * Math.cos(angleZ) - y * Math.sin(angleZ);
    const newY = newX * Math.sin(angleZ) + y * Math.cos(angleZ);
    //X
    const anotherY = newY * Math.cos(angleX) - newZ * Math.sin(angleX);
    const anotherZ = newY * Math.sin(angleX) + newZ * Math.cos(angleX);

    rotatedVertices.push([anotherX, anotherY, anotherZ]);
}
```

Obrázek 27: Rotace vrcholů

Zdroj: vlastní zpracování

Následně se určí střed obrazu podle rozměrů canvas a k výsledku se přičte posun vzniklý manipulací s objektem pomocí pravého tlačítka myši. Vrcholy po rotaci jsou poté posunuty na správnou pozici, zvětšeny podle měřítka a následně se uplatní perspektivní promítání popsané v kapitole 1.5 (Nominal Animal, 2017).

```
const centerX = (canvas.width / 2) + posX;
const centerY = (canvas.height / 2) + posY;
for (let i = 0; i < rotatedVertices.length; i++) {
    const posX = (centerX - rotatedVertices[i][0] * (distance / (rotatedVertices[i][2] + distance)
        ) * scaling) * antiAliasingMultiplier;
    const posY = (centerY - rotatedVertices[i][1] * (distance / (rotatedVertices[i][2] + distance)
        ) * scaling) * antiAliasingMultiplier;
    const posZ = rotatedVertices[i][2];
    verticesOnAAcanvas.push([posX, posY, posZ]);
}
```

Obrázek 28: Perspektivní promítání a přesun modelu

Zdroj: vlastní zpracování

Nakonec se připraví plochy definované indexy vrcholů podle struktury OBJ souborů popsané v kapitole 1.2 (Chakravorty, 2023). Souřadnice jednotlivých ploch se načtou a uloží do polí využívaných při vykreslování. Pokud je zapnuto zobrazení drátového modelu, tak se na základě uvedených hodnot sestaví cesta, která slouží k jeho zobrazení.

```
for (let i = 0; i < faces.length; i++) {
    const innerArray = faces[i];
    facesToDraw[i].arrX.length = 0;
    facesToDraw[i].arrY.length = 0;
    facesToDraw[i].arrZ.length = 0;

    const path = new Path2D();

    for (let j = 0; j < innerArray.length; j++) {
        const getItem = (innerArray[j][0] - 1);
        const positionX = verticesOnAAcanvas[getItem][0];
        const positionY = verticesOnAAcanvas[getItem][1];
        const positionZ = verticesOnAAcanvas[getItem][2];

        if (showWireframe) {
            if (j == 0) path.moveTo(positionX, positionY);
            else path.lineTo(positionX, positionY);
        }

        facesToDraw[i].arrX.push(positionX);
        facesToDraw[i].arrY.push(positionY);
        facesToDraw[i].arrZ.push(positionZ);
    }

    if (showWireframe) {
        path.closePath();
        facesToDraw[i].path = path;
    }
}
```

Obrázek 29: Skládání vrcholů do trojúhelníků

Zdroj: vlastní zpracování

2.5 Výpočet směru plochy a jeho využití

Kapitola se věnuje praktickému výpočtu normálového vektoru plochy, jeho využití pro výpočet stínů a back-face culling, přičemž teoretický základ byl uveden v části 1.6.

2.5.1 Výpočet normálového vektoru

Vytvoří se smyčka, která projde všechny plochy v poli faces, takže každá plocha má svůj normálový vektor uložený na stejné pozici v poli faceNormals. Na začátku každého průchodu se vytvoří pomocné proměnné a začne se s výpočtem normálu na základě vrcholů dané plochy. Poté se vypočítají dvě hrany trojúhelníku odečtením souřadnic mezi vrcholy. Ze získaných hran se určí normálový vektor pomocí vektorového součinu (Baker, nedatováno). Normálový vektor se normalizuje vydělením jeho složek délkou, čímž vznikne jednotkový vektor (Ellison, 2023). Normalizovaný vektor se uloží do pole faceNormals pro další použití jinými funkcemi.

```
function calculateFaceNormals(arr) {
    faceNormals.length = 0;

    for (let i = 0; i < faces.length; i++) {
        const innerArray = faces[i];

        let vertexArr = [];
        let normalsArr = [];
        let edge1 = [];
        let edge2 = [];

        for (let j = 0; j < 3; j++) {
            const getItem = (innerArray[j][0] - 1);
            vertexArr.push([arr[getItem][0], arr[getItem][1], arr[getItem][2]]);
        }
        edge1.push(vertexArr[1][0] - vertexArr[0][0], vertexArr[1][1] - vertexArr[0][1], vertexArr[1][2] - vertexArr[0][2]);
        edge2.push(vertexArr[2][0] - vertexArr[0][0], vertexArr[2][1] - vertexArr[0][1], vertexArr[2][2] - vertexArr[0][2]);

        normalsArr.push((edge1[1] * edge2[2]) - (edge1[2] * edge2[1]), (edge1[2] * edge2[0]) - (edge1[0] * edge2[2]),
            (edge1[0] * edge2[1]) - (edge1[1] * edge2[0]));

        const vectorLength = Math.sqrt(normalsArr[0] * normalsArr[0] + normalsArr[1] * normalsArr[1] + normalsArr[2] * normalsArr[2]);

        normalsArr[0] = normalsArr[0] / vectorLength;
        normalsArr[1] = normalsArr[1] / vectorLength;
        normalsArr[2] = normalsArr[2] / vectorLength;

        faceNormals.push(normalsArr);
    }
}
```

Obrázek 30: Výpočet normálových vektorů

Zdroj: vlastní zpracování

2.5.2 Výpočet barvy pomocí normálového vektoru

Protože stínování je v práci počítáno na procesoru, byla použita technika, při níž je určena pouze jedna barva na celou plochu, označovaná jako flat shading (Gambetta, 2021). V souboru obj navíc nejsou informace o světelném zdroji, proto je směr zdroje světla definován přímo v kódu. Smyčka projde všechny normály v poli faceNormals, pro každý se vypočítá skalární součin se směrem světla, čímž se získá hodnota toho, jak moc je plocha nasměrována ke zdroji světla, jak je popsáno v 1.6.3 (Sheldon, 2022). Hodnota se následně převede do rozsahu 0 až 1 pomocí lineární interpolace uvedené v části 2.1.6 (Cuemath, 2021). Podle získané hodnoty se upraví složky r, g, b převzaté z pole colorByIdWithNumValues, které obsahuje barvy načtené z MTL souboru, čímž se získá výsledná odstínovaná barva plochy (Gambetta, 2021). Výsledné složky se uloží do pole colorById ve stejném pořadí, jako plochy, ke kterým patří.

```
function dotProduct(vec1, vec2) {
    return vec1[0] * vec2[0] + vec1[1] * vec2[1] + vec1[2] * vec2[2];
}

function calculateFaceColor() {
    const lightDir = [-0.3, 0.2, -0.3];

    for (let i = 0; i < faceNormals.length; i++) {
        const dot = dotProduct(faceNormals[i], lightDir);
        const color = (dot + 1) * 0.5;
        facesToDraw[i].colorDotProduct = color;

        let r = Math.floor(colorByIdWithNumValues[i][0] * color);
        let g = Math.floor(colorByIdWithNumValues[i][1] * color);
        let b = Math.floor(colorByIdWithNumValues[i][2] * color);

        colorById[i] = [r, g, b];
    }
}
```

Obrázek 31: Výpočet barvy plochy

Zdroj: vlastní zpracování

2.6 Vykreslení ploch

Po provedení všech potřebných výpočtů je nutné převést získané údaje na obraz zobrazený na obrazovce. Obraz se skládá z ploch, které jsou vykresleny do podoby viditelného objektu.

2.6.1 Mapování obrázků

Pro práci s obrázky je potřeba spočítat afinní zobrazení, které je namapuje na trojúhelníky, jak bylo vysvětleno v části 1.7. Nejprve se vyprázdní pole affineTransformCalculations a ověří se, jestli byl uživatelem nahrán jakýkoliv obrázek. Poté se postupně procházejí plochy určené k výpočtu. Pokud plocha nemá definovány souřadnice, uloží se hodnota null. U ostatních se načtou souřadnice posU a posV jako hodnoty, ze kterých bude trojúhelník pocházet a aktuální otočené vrcholy jsou uloženy do arrX a arrY jako cílové souřadnice trojúhelníku. Hodnoty v arrX a arrY jsou mírně zvětšeny, aby obrazy navazovaly plynuleji a působily přirozeněji. Z načtených souřadnic se podle vzorce uvedeného v části 1.7 vypočítají koeficienty A až F, které určují transformaci obrazu. (Borgen, 2018)

```
function calculateAffineTransform(facesToProcess) {
    affineTransformCalculations.length = 0;

    if (!containsImage) return;
    for (let i = 0; i < facesToProcess.length; i++) {
        if (facesToProcess[i].mapping.length === 0) {
            affineTransformCalculations.push(null);
            continue;
        }
        const posU = facesToProcess[i].mapping.posU;
        const posV = facesToProcess[i].mapping.posV;
        const arrX = expandPoints(facesToProcess[i].arrX);
        const arrY = expandPoints(facesToProcess[i].arrY);

        const A = (((arrX[1] - arrX[2]) * (posV[0] - posV[1])) - ((arrX[0] - arrX[1]) * (posV[1] - posV[2]))
                ) / (
                ((posU[1] - posU[2]) * (posV[0] - posV[1])) - ((posU[0] - posU[1]) * (posV[1] - posV[2])));

        const B = (((arrY[1] - arrY[2]) * (posV[0] - posV[1])) - ((arrY[0] - arrY[1]) * (posV[1] - posV[2]))
                ) / (
                ((posU[1] - posU[2]) * (posV[0] - posV[1])) - ((posU[0] - posU[1]) * (posV[1] - posV[2])));

        const C = (((arrX[1] - arrX[2]) * (posU[0] - posU[1])) - ((arrX[0] - arrX[1]) * (posU[1] - posU[2]))
                ) / (
                ((posV[1] - posV[2]) * (posU[0] - posU[1])) - ((posV[0] - posV[1]) * (posU[1] - posU[2])));

        const D = (((arrY[1] - arrY[2]) * (posU[0] - posU[1])) - ((arrY[0] - arrY[1]) * (posU[1] - posU[2]))
                ) / (
                ((posV[1] - posV[2]) * (posU[0] - posU[1])) - ((posV[0] - posV[1]) * (posU[1] - posU[2])));

        const E = arrX[0] - (posU[0] * A) - (posV[0] * C);
        const F = arrY[0] - (posU[0] * B) - (posV[0] * D);

        affineTransformCalculations.push([A, B, C, D, E, F]);
    }
}
```

Obrázek 32: Výpočet afinního zobrazení

Zdroj: vlastní zpracování dle Borgen, 2018

2.6.2 Malířův algoritmus

Malířův algoritmus byl použit jako první řešení pro vykreslování, vytvořený podle popisu uvedeného v části 1.8.1. Nejprve byla pro každou plochu vypočítána průměrná vzdálenost všech jejích vrcholů. Podle vypočtených hodnot byly plochy seřazeny od nejvzdálenější po nejbližší (Griffin, 2022). Následně byla spuštěna smyčka, která postupně prochází všechny plochy. Pro každou plochu byly načteny barvy z pole colorById, odstíny z pole shadeById a barvy hran z pole edgeShadeById, protože v původní verzi byly barvy a stínování uloženy a vykresleny odděleně. Pomocí vlastností fillStyle a strokeStyle byly nastaveny barvy. Následně funkce stroke a fill zajistily vykreslení, což jsou funkce dostupné v JavaScriptu pro práci s HTML canvas. Poté byly aplikovány odstíny, které dodaly ploše hloubku.

```
const canvas = document.getElementById('3D');
const context = canvas.getContext('2d', { willReadFrequently: true });

function calculateAvgZ(arr) { return (arr.reduce((sum, cur) => sum + cur, 0)) / arr.length; }
for (let i = 0; i < facesToDraw.length; i++) {
    facesToDraw[i].avgZ = calculateAvgZ(facesToDraw[i].arrZ);
}
let sortedFaces = facesToDraw.slice();
sortedFaces.sort(function (a, b) { return b.avgZ - a.avgZ });

for (let draw = 0; draw < sortedFaces.length; draw++) {
    const face = sortedFaces[draw];
    const color = colorById[face.id];
    const shade = shadeById[face.id];
    const edgeShade = edgeShadeById[face.id];

    context.fillStyle = color;
    context.strokeStyle = color;
    context.stroke(face.path);
    context.fill(face.path);

    context.fillStyle = shade;
    context.strokeStyle = edgeShade;
    context.stroke(face.path);
    context.fill(face.path);
}
```

Obrázek 33: Vykreslení objektu pomocí malířova algoritmu

Zdroj: vlastní zpracování

2.6.3 Z-buffer

Kvůli obtížím při vykreslování složitějších modelů byl malířův algoritmus v práci nahrazen metodou Z-bufferu. Při použití Z-bufferu je vytvořeno pole, které uchovává informace o každém pixelu, aby bylo dosaženo správného vykreslení scény (Yadav, 2025). Při inicializaci se hodnoty v polích zBuffer a pixelBuffer nastaví na nekonečno a −1, čímž se připraví prostor pro ukládání výsledků výpočtu. Z-buffer s hodnotami nekonečno slouží k uchování dat o hloubce jednotlivých pixelů, zatímco pixelBuffer ukládá informaci o tom, které ploše daný pixel náleží.

Pro zvýšení kvality obrazu byla použita metoda SSAA, která funguje tak, že se obraz vykreslí ve vyšším rozlišení a poté se zmenší, čímž dojde k uhlazení hran (Evanson, 2021). Výpočet probíhá nad zvětšeným canvas podle nastavené úrovně SSAA, kde se pro každý trojúhelník určí hranice na osách X a Y. Uvnitř určených hranic se pro každý pixel vypočítají barycentrické souřadnice pomocí ploch trojúhelníků, které rozhodnou, zda pixel leží uvnitř trojúhelníku (Sokolov, 2025). Pokud pixel leží uvnitř trojúhelníku, určí se jeho hloubka a porovná se s aktuální zapsanou hodnotou v bufferu. Pokud je nový pixel blíže, přepíšou se hodnoty v Z-bufferu spolu s identifikací v pixelBufferu, čímž se zajistí správné překrývání ploch (Yadav, 2025).

```
const AAcanvasWidth = offScreenAntiAliasingCanvas.width;
const AAcanvasHeight = offScreenAntiAliasingCanvas.height;
antiAliasingCanvasContext.clearRect(0, 0, AAcanvasWidth, AAcanvasHeight);

const zBuffer = new Float32Array(AAcanvasWidth * AAcanvasHeight).fill(Infinity);
const pixelBuffer = new Int32Array(AAcanvasWidth * AAcanvasHeight).fill(-1);

let startingPos = 0;
if (showWireframe) startingPos = facesLength;
for (let i = startingPos; i < sortedFaces.length; i++) {
    const id = sortedFaces[i].id;
    const triangleArrX = sortedFaces[i].arrX;
    const triangleArrY = sortedFaces[i].arrY;
    const triangleArrZ = sortedFaces[i].arrZ;

    const minX = Math.max(0, Math.floor(Math.min(triangleArrX[0], triangleArrX[1], triangleArrX[2])));
    const maxX = Math.min(AAcanvasWidth - 1, Math.ceil(Math.max(triangleArrX[0], triangleArrX[1], triangleArrX[2])));
    const minY = Math.max(0, Math.floor(Math.min(triangleArrY[0], triangleArrY[1], triangleArrY[2])));
    const maxY = Math.min(AAcanvasHeight - 1, Math.ceil(Math.max(triangleArrY[0], triangleArrY[1], triangleArrY[2])));
    const area = calculate_triangle_area(triangleArrX[0], triangleArrY[0], triangleArrX[1], triangleArrY[1], triangleArrX[2], triangleArrY[2]);

    for (let x = minX; x <= maxX; x++) {
        for (let y = minY; y <= maxY; y++) {
            const pixelIndex = y * AAcanvasWidth + x;

            const a = calculate_triangle_area(x, y, triangleArrX[1], triangleArrY[1], triangleArrX[2], triangleArrY[2]) / area;
            const b = calculate_triangle_area(x, y, triangleArrX[2], triangleArrY[2], triangleArrX[0], triangleArrY[0]) / area;
            const c = calculate_triangle_area(x, y, triangleArrX[0], triangleArrY[0], triangleArrX[1], triangleArrY[1]) / area;

            if (a < 0 || b < 0 || c < 0) continue;

            const z = a * triangleArrZ[0] + b * triangleArrZ[1] + c * triangleArrZ[2];
            if (z < zBuffer[pixelIndex]) {
                zBuffer[pixelIndex] = z;
                pixelBuffer[pixelIndex] = id;
            }

        }
    }
}
```

Obrázek 34: Vyplnění Z-bufferu

Zdroj: vlastní zpracování dle Sokolov, 2025

Plocha trojúhelníku tvořená třemi vrcholy A, B, C se vypočítá podle vztahu:

$$sABC = \frac{1}{2} * \big((By - Ay) * (Bx + Ax) + (Cy - By) * (Cx + Bx) + (Ay - Cy) * (Ax + Cx)\big) \quad (26)$$

kde Ax, Ay jsou souřadnice vrcholu A, Bx, By jsou souřadnice vrcholu B, Cx, Cy jsou souřadnice vrcholu C a sABC je výsledná plocha trojúhelníku ABC. (Sokolov, 2025)

Pro výpočet barycentrických souřadnic bodu Px, Py se stejným způsobem spočítají plochy podtrojúhelníků sPBC, sPCA, sPAB kde se ke dvojici vrcholů původního trojúhelníku doplní souřadnice pixelu P. Podílem jednotlivých ploch s celkovou plochou sABC se získají barycentrické souřadnice (Sokolov, 2025). Následným vynásobením barycentrických souřadnic příslušnými

hodnotami souřadnice Z jednotlivých vrcholů se určí výsledná souřadnice Z v bodě P (Yadav, 2025).

Při vykreslování bez obrázků se barvy ploch zapisují přímo z pixelBufferu. Nejprve se vytvoří nové pole imageData o velikosti zvětšené canvas. Poté se projdou všechny pixely a tam, kde je přiřazena plocha, se vypočítá index a získá se barva z colorById. Alfa kanál se zde nastaví na 255, protože z-buffer nepodporuje použití průhlednosti (Bansal, 2021). Nakonec se celé pole imageData vloží zpět na canvas pomocí putImageData.

```
const imageData = antiAliasingCanvasContext.createImageData(AAcanvasWidth, AAcanvasHeight);
const data = imageData.data;

for (let i = 0; i < pixelBuffer.length; i++) {
    if (pixelBuffer[i] !== -1) {
        const index = i * 4;
        const color = colorById[pixelBuffer[i]];

        data[index] = color[0];
        data[index + 1] = color[1];
        data[index + 2] = color[2];
        data[index + 3] = 255;
    }
}
antiAliasingCanvasContext.putImageData(imageData, 0, 0);
```

Obrázek 35: Vykreslení Z-bufferu

Zdroj: vlastní zpracování

2.6.4 Přidání obrázků do Z-bufferu

Když plocha používá obrázek, vytvoří se pole masek, do kterého se ukládají indexy pixelů patřících jednotlivým trojúhelníkům. Pro plochy s definovaným afinním zobrazením se obrázek vykreslí na dočasnou canvas, kde se na něj aplikuje transformace z UV souřadnic. Tím se obraz správně přemístí a natočí do cílového trojúhelníku (Borgen, 2018).

```
if (faceUsesAnImage > 0 && !showWireframe) {
    const imgCount = sortedFaces.length;
    const masks = new Array(facesToDraw.length + 2);
    for (let i = 0; i < masks.length; i++) {
        masks[i] = [];
    }
    for (let idx = 0; idx < pixelBuffer.length; idx++) {
        const imgIdx = pixelBuffer[idx];
        if (imgIdx >= 0) masks[imgIdx].push(idx);
    }
    const outImageData = antiAliasingCanvasContext.createImageData(AAcanvasWidth, AAcanvasHeight);
    const outData = outImageData.data;
    for (let i = 0; i < imgCount; i++) {
        if (affineTransformCalculations[i] !== null && affineTransformCalculations[i] !== undefined) {
            const mask = masks[sortedFaces[i].id];
            if (!mask.length) continue;
            const imageValues = sortedFaces[i].mapping;
            const img = imageValues.canvas;

            const arrX = sortedFaces[i].arrX;
            const arrY = sortedFaces[i].arrY;

            const minX = Math.floor(Math.min(...arrX));
            const minY = Math.floor(Math.min(...arrY));
            const maxX = Math.ceil(Math.max(...arrX));
            const maxY = Math.ceil(Math.max(...arrY));

            const width = maxX - minX;
            const height = maxY - minY;
            tempImgContext.setTransform(1, 0, 0, 1, 0, 0);
            tempImgContext.clearRect(0, 0, AAcanvasWidth, AAcanvasHeight);
            tempImgContext.setTransform(...affineTransformCalculations[i]);

            const destX = Math.min(...imageValues.posU);
            const destY = Math.min(...imageValues.posV);
            const imgWidth = img.width;
            const imgHeight = img.height;

            tempImgContext.drawImage(
                img,
                imageOffset, imageOffset,
                imgWidth, imgHeight,
                destX, destY,
                imgWidth, imgHeight
            );
```

Obrázek 36: Vytvoření masky plochy a načtení obrázku

Zdroj: vlastní zpracování

Z dočasné canvas se vybere oblast odpovídající hranicím trojúhelníku a pro každý pixel z masky se zkopírují barevné hodnoty. Průhledné pixely se přeskočí, protože je z-buffer nepodporuje, a následně se barvy násobí koeficientem pro stínování, čímž se ploše dodá hloubka.

```
const color = sortedFaces[i].colorDotProduct;
const colorMultiply = 0.5 + color * 0.5;
const tempImg = tempImgContext.getImageData(minX, minY, width, height);
const tempData = tempImg.data;

for (let k = 0; k < mask.length; k++) {
    const maskIndex = mask[k];

    const MainCanvasPositionX = maskIndex % AAcanvasWidth;
    const MainCanvasPositionY = Math.floor(maskIndex / AAcanvasWidth);

    if (MainCanvasPositionX < minX || MainCanvasPositionX >= maxX || MainCanvasPositionY < minY || MainCanvasPositionY >= maxY) continue;

    const imageCopyPositionX = MainCanvasPositionX - minX;
    const imageCopyPositionY = MainCanvasPositionY - minY;

    const finalPosition = (imageCopyPositionY * width + imageCopyPositionX) * 4;
    const outIdx = maskIndex * 4;

    if (tempData[finalPosition + 3] === 0) continue;

    outData[outIdx] = tempData[finalPosition] * colorMultiply;
    outData[outIdx + 1] = tempData[finalPosition + 1] * colorMultiply;
    outData[outIdx + 2] = tempData[finalPosition + 2] * colorMultiply;
    outData[outIdx + 3] = 255;
}
```

Obrázek 37: Vybrání pixelů z obrázku pomocí masky

Zdroj: vlastní zpracování

Pokud plocha obrázek nepoužívá, vyplní se maska pevnou barvou podle identifikátoru plochy podobně, jako v části 2.6.3. Nakonec se všechny hodnoty uloží do hlavní canvas zase pomocí putImageData.

```
    else {
        const mask = masks[sortedFaces[i].id];
        if (!mask.length) continue;

        const color = colorById[sortedFaces[i].id];

        for (let k = 0; k < mask.length; k++) {
            const index = mask[k] * 4;
            outData[index] = color[0];
            outData[index + 1] = color[1];
            outData[index + 2] = color[2];
            outData[index + 3] = 255;
        }
    }
}

antiAliasingCanvasContext.putImageData(outImageData, 0, 0);
```

Obrázek 38: Dodatečné vykreslení ploch bez obrázku

Zdroj: vlastní zpracování

2.6.5 Drátový model

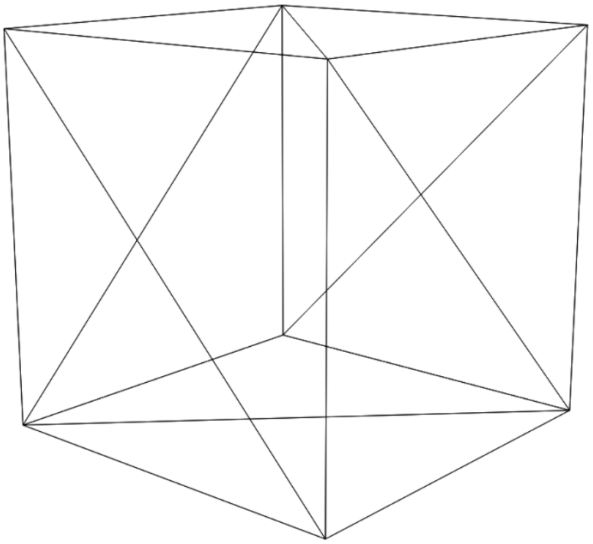
Při zapnutém režimu drátového modelu se z modelu vykreslují pouze hrany ploch. Nejprve se nastaví režim kreslení tak, aby se čáry zobrazily pod aktuálním obsahem canvas, což ponechá body na chytání vrcholů viditelné nad modelem. Poté se projdou všechny plochy s definovanou cestou path, nastaví se barva podle identifikátoru a provede se vykreslení hran pomocí stroke. Nakonec se režim kreslení vrátí na výchozí hodnotu.

```
if (showWireframe) {
    antiAliasingCanvasContext.globalCompositeOperation = 'destination-over';
    for (let i = 0; i < sortedFaces.length; i++) {
        if (sortedFaces[i].path !== undefined) {
            const face = sortedFaces[i];
            const color = colorById[face.id];
            const path = face.path;

            antiAliasingCanvasContext.strokeStyle = `rgb(${color[0]}, ${color[1]}, ${color[2]})`;
            antiAliasingCanvasContext.stroke(path);
        }
    }
    antiAliasingCanvasContext.globalCompositeOperation = 'source-over';
}
```

Obrázek 39: Vykreslení drátového modelu

Zdroj: vlastní zpracování

Díky tomu je vytvořeno zobrazení drátového modelu, jak je znázorněno na obrázku 40, kde jsou viditelné pouze hrany ploch daného modelu.



Obrázek 40: Ukázka drátového modelu

Zdroj: vlastní zpracování

2.7 Úprava 3D modelu uživatelem

Na začátku sekce se popisuje interaktivní práce s modelem, která zahrnuje přidávání chytacích bodů na vrcholech, jejich výběr, pohyb a rotaci. Součástí je také změna barvy modelu, která slouží k úpravě jeho zobrazení při editaci.

2.7.1 Přidání chytacích bodů na vrcholech

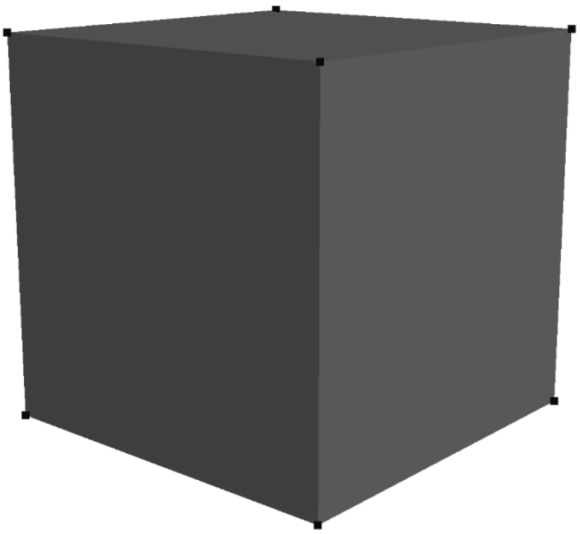
Při zapnutém editačním režimu se na všech vrcholech vykreslí čtvercové chytací body. Velikost bodů se určí podle antiAliasingMultiplier, protože je použito SSAA. Obraz se totiž nejprve vykreslí na větší canvas a poté se zmenšený přenese na aktuální canvas, takže čím vyšší hodnota antiAliasingMultiplier, tím větší musí být chytací bod, aby se jeho velikost při zobrazení zachovala (Evanson, 2021). Pro každý vrchol se vypočítají souřadnice rohů čtverce a podle toho, zda je vrchol označen, se nastaví jeho pozice a barva, a následně se vloží jako plocha do pole sortedFaces, aby byl vykreslen se zbytkem modelu.

```
const squareSize = 5 * antiAliasingMultiplier;
const startingId = facesToDraw.length;
for (let i = 0; i < verticesOnAAcanvas.length; i++) {
    const posX = verticesOnAAcanvas[i][0]
    const posY = verticesOnAAcanvas[i][1]

    const X1 = posX - squareSize;
    const Y1 = posY - squareSize;
    const X2 = posX + squareSize;
    const Y2 = posY - squareSize;
    const X3 = posX - squareSize;
    const Y3 = posY + squareSize;
    const X4 = posX + squareSize;
    const Y4 = posY + squareSize;

    if (highlightArray.includes(i)) {
        let posZ;
        if (drawSelectedPointsOnTop) posZ = -9999;
        else posZ = verticesOnAAcanvas[i][2] - 0.1;

        sortedFaces.push({
            id: startingId + 1,
            arrX: [X1, X2, X4],
            arrY: [Y1, Y2, Y4],
            arrZ: [posZ, posZ, posZ],
        });
        sortedFaces.push({
            id: startingId + 1,
            arrX: [X1, X3, X4],
            arrY: [Y1, Y3, Y4],
            arrZ: [posZ, posZ, posZ],
        });
    }

    else {
        const posZ = verticesOnAAcanvas[i][2] - 0.1;
        sortedFaces.push({
            id: startingId,
            arrX: [X1, X2, X4],
            arrY: [Y1, Y2, Y4],
            arrZ: [posZ, posZ, posZ],
        });
        sortedFaces.push({
            id: startingId,
            arrX: [X1, X3, X4],
            arrY: [Y1, Y3, Y4],
            arrZ: [posZ, posZ, posZ],
        });
    }
}
```

Obrázek 41: Přidání chytacích bodů na vrcholy modelu

Zdroj: vlastní zpracování

Provedení uvedeného kódu je znázorněno na obrázku 42, kde jsou na všech vrcholech viditelné čtvercové chytací body.



Obrázek 42: Ukázka vykreslení chytacích bodů

Zdroj: vlastní zpracování

2.7.2 Zjištění kliknutých bodů

Při stisknutí tlačítka myši se souřadnice kliknutí přepočítají podle antiAliasingMultiplier a následně zaokrouhlí, aby vznikla platná celočíselná souřadnice. Z pixelBufferCopy se načte hodnota identifikátoru bodu a podle ní se zavolá funkce findClickedSquare.

```
if (enableEditMode) {
    const scaledMouseX = mouseX * antiAliasingMultiplier;
    const scaledMouseY = mouseY * antiAliasingMultiplier;

    const pixelPosition = Math.floor(scaledMouseY) * offScreenAntiAliasingCanvas.width + Math.floor(scaledMouseX);
    const pixelInfo = pixelBufferCopy[pixelPosition];
    const selectPointId = facesToDraw.length;

    if (event.buttons === 1 && pixelInfo === selectPointId) findClickedSquare(0);
    if (event.buttons === 1 && pixelInfo === (selectPointId + 1)) findClickedSquare(1);
}
```

Obrázek 43: Zjištění kliknutí na chytací vrchol

Zdroj: vlastní zpracování

Ve funkci se určí nejbližší čtverec k pozici myši a podle stavu klávesy Shift se buď přidá nový vrchol do pole highlightArray, nebo se pole vymaže a vloží pouze aktuální. Kliknutí na oranžový bod způsobí odstranění vrcholu.

```
function findClickedSquare(orange) {
    const squareSize = 5 * antiAliasingMultiplier;
    let closestSquareIndex = null;
    let closestDistance = Infinity;

    for (let i = 0; i < verticesOnScreen.length; i++) {
        const vertexX = verticesOnScreen[i][0];
        const vertexY = verticesOnScreen[i][1];
        const vertexZ = verticesOnScreen[i][2];
        const xMin = vertexX - squareSize;
        const xMax = vertexX + squareSize;
        const yMin = vertexY - squareSize;
        const yMax = vertexY + squareSize;

        if (mouseX >= xMin && mouseX <= xMax && mouseY >= yMin && mouseY <= yMax) {
            if (vertexZ < closestDistance) {
                closestDistance = vertexZ;
                closestSquareIndex = i;
            }
        }
    }

    if (closestSquareIndex !== null && !orange && IsShiftHeld) {
        if (!highlightArray.includes(closestSquareIndex)) {
            highlightArray.push(closestSquareIndex);
            isRedrawNeeded = 1;
        }
    }

    else if (closestSquareIndex !== null && !orange && !IsShiftHeld) {
        highlightArray.length = 0;
        highlightArray.push(closestSquareIndex);
        isRedrawNeeded = 1;
    }

    if (closestSquareIndex !== null && orange) {

        const index = highlightArray.indexOf(closestSquareIndex);
        if (index !== -1) {
            highlightArray.splice(index, 1);
            isRedrawNeeded = 1;
        }
    }
}
```

Obrázek 44: Přidání nebo odebrání označení jednoho bodu

Zdroj: vlastní zpracování

Podobný postup se používá i při výběru více bodů, kdy se stiskne prostřední tlačítko myši a tažením se určí hranice, přičemž do pole highlightArray se přidají všechny vrcholy ležící ve vybraném prostoru. Výběr se může provádět buď stisknutím kolečka myši nebo, pokud je v menu zvolena možnost select with left click, i pomocí levého tlačítka. Duplicity se při novém výběru odstraní a nastaví se nastaví isRedrawNeeded na 1, aby se aktualizovali aktuálně chytnuté body i na obrazovce.

```
function findPointsInSquare(startX, startY, endX, endY) {
    let minX = Math.min(startX, endX);
    let maxX = Math.max(startX, endX);
    let minY = Math.min(startY, endY);
    let maxY = Math.max(startY, endY);

    if (!IsShiftHeld) highlightArray.length = 0;

    for (let i = 0; i < verticesOnScreen.length; i++) {
        const x = verticesOnScreen[i][0];
        const y = verticesOnScreen[i][1];

        if (x >= minX && x <= maxX && y >= minY && y <= maxY) {
            highlightArray.push(i);
        }
    }

    highlightArray = [...new Set(highlightArray)]

    isRedrawNeeded = 1;
}
```

Obrázek 45: Označení více bodů

Zdroj: vlastní zpracování

2.7.3 Pohyb vybraných bodů

Ve funkci moveVertices se upraví souřadnice vybraných vrcholů podle změny polohy myši. Rotace se následně provede kolem os X, Z a Y, tedy v opačném pořadí i s opačnými úhly, než je v práci použito při běžném otáčení podle os Y, Z a X. Daným posunem se vrací souřadnice do stavu, v jakém byly před provedením rotace. Díky tomu je lze následně znovu rotovat, nyní již jako posunuté, protože po zpětném otočení odpovídají původním souřadnicím s přidaným posunem (Beaker, 2014). Výsledné hodnoty se uloží zpět do pole vertices a nastaví se isRedrawNeeded na 1 pro aktualizaci pohybu.

```
function moveVertices() {
    for (let i = 0; i < highlightArray.length; i++) {
        const verticeToMove = rotatedVertices[highlightArray[i]];
        verticeToMove[0] -= mouseChangeX / scaling;
        verticeToMove[1] -= mouseChangeY / scaling;
        const x = verticeToMove[0];
        const y = verticeToMove[1];
        const z = verticeToMove[2];

        const newY = y * Math.cos(-angleX) - z * Math.sin(-angleX);
        const newZ = y * Math.sin(-angleX) + z * Math.cos(-angleX);
        const newX = x * Math.cos(-angleZ) - newY * Math.sin(-angleZ);
        const anotherY = x * Math.sin(-angleZ) + newY * Math.cos(-angleZ);
        const anotherX = newX * Math.cos(-angleY) + newZ * Math.sin(-angleY);
        const anotherZ = -newX * Math.sin(-angleY) + newZ * Math.cos(-angleY);

        const newPositionVertice = [anotherX, anotherY, anotherZ];
        vertices[highlightArray[i]] = newPositionVertice;
        isRedrawNeeded = 1;
    }
}
```

Obrázek 46: Posun vrcholů podle pohybu myši a zapsání nové informace

Zdroj: vlastní zpracování

2.7.4 Změna barvy

Uživatel může měnit barvu modelu prostřednictvím nabídky materiálů. Při generování nabídky se kontroluje, zda je v poli materiálů obsažen výchozí materiál. Pokud mají všechny plochy přiřazený vlastní materiál, výchozí se odstraní a do menu se vloží pouze aktuálně používané. Poté se nastaví první materiál jako aktivní a jeho barva se zobrazí v ovládacím prvku pro výběr barvy.

```
function updateMatSelectionContent() {
    const selectMat = document.getElementById('matSelecter');

    let containsZero = 0;
    for (let i = 0; i < matName.length; i++) {
        if (matName[i].id === 0) {
            containsZero = 1;
            break;
        }
    }

    if (containsZero) {
        matName.shift();
        selectMat.innerHTML = '';
    }
    else {
        selectMat.innerHTML = '';
    }

    for (let i = 0; i < matName.length; i++) {
        const option = document.createElement('option');
        option.value = matName[i].name;
        option.textContent = matName[i].name;
        selectMat.appendChild(option);
    }
    selectMat.value = matName[0].name;
    matSelectorFirstUpdateOnLoad(selectMat.value);
}
```

Obrázek 47: Přidání nabídky na změnu barvy

Zdroj: vlastní zpracování

Při změně volby se vyhledá odpovídající materiál a jeho barva se převede do hexadecimálního formátu, protože v HTML lze do vstupního prvku pro barvu zapisovat a získávat hodnoty pouze v hexadecimálním formátu (Chhabra, 2024). Pokud uživatel barvu upraví, nová hodnota se převede zpět na RGB složky, uloží se do pole materiálů a aplikuje na model (Down, 2022). Následně se nastaví isRedrawNeeded na 1, aby se model překreslil s novými barvami.

```
function rgbToHex(r, g, b) {
    r = Math.floor(r);
    g = Math.floor(g);
    b = Math.floor(b);
    return "#" + (1 << 24 | r << 16 | g << 8 | b).toString(16).slice(1);
}
function matSelectorChange(selectedOption) {
    const foundMaterial = colorArr.find(material => material.name === selectedOption);
    if (foundMaterial) {
        const colorInput = document.getElementById('curColor');
        colorInput.value = rgbToHex(foundMaterial.color[0], foundMaterial.color[1], foundMaterial.color[2]);
    }
}


function matSelectorUpdateModelColor(selectedValue) {
    let bigint = parseInt(selectedValue.slice(1), 16);
    let r = (bigint >> 16) & 255;
    let g = (bigint >> 8) & 255;
    let b = bigint & 255;

    const selectMat = document.getElementById('matSelecter');
    const currentMat = selectMat.value;

    const foundMaterial = colorArr.find(material => material.name === currentMat);
    if (foundMaterial) {
        foundMaterial.color = [r, g, b];
        applyColors(colorArr);
        isRedrawNeeded = 1;
    }
}
```

Obrázek 48: Aktualizace barvy modelu a obsahu HTML vstupu

Zdroj: vlastní zpracování

2.8 Export 3D modelu

Export modelu probíhá podobně jako import. Místo načítání dat se obsah souborů upravuje a připravuje k uložení. Výsledkem jsou soubory OBJ, MTL a obrázky, které dohromady tvoří kompletní model.

2.8.1 Export OBJ

Při exportu do formátu OBJ se kontroluje dostupnost původního souboru. Pokud je uživatelem zvolena normalizace, souřadnice vrcholů se přepočítají do rozsahu od -1 do 1 a zaokrouhlují na šest desetinných míst, jinak se zapíšou aktuální vrcholy bez převodu.

```
if (exportObj) {
    let objFile = originalOBJfile;
    if (!objFile) {
        alert("Missing original OBJ file");
        return;
    }
    let lines = objFile.split('\n');
    let currentVertex = 0;

    if (normalizeModel) {
        let tempArray = vertices.slice();
        const flatArray = vertices.slice().flat(Infinity);
        const minNum = Math.min(...flatArray);
        const maxNum = Math.max(...flatArray);

        for (let i = 0; i < tempArray.length; i++) {
            for (let j = 0; j < tempArray[i].length; j++) {
                tempArray[i][j] = (-1 + (((tempArray[i][j] - minNum) * 2) / (maxNum - minNum))).toFixed(6);
            }
        }
        for (let i = 0; i < lines.length; i++) {
            let line = lines[i];
            let parts = line.trim().split(/\s+/);

            if (parts[0] === "v") {
                parts[1] = tempArray[currentVertex][0];
                parts[2] = tempArray[currentVertex][1];
                parts[3] = tempArray[currentVertex][2];
                currentVertex++;
            }
            lines[i] = parts.join(' ');
        }
        objFile = lines.join('\n');
    }

    else {
        for (let i = 0; i < lines.length; i++) {
            let line = lines[i];
            let parts = line.trim().split(/\s+/);

            if (parts[0] === "v") {
                parts[1] = vertices[currentVertex][0].toFixed(6);
                parts[2] = vertices[currentVertex][1].toFixed(6);
                parts[3] = vertices[currentVertex][2].toFixed(6);
                currentVertex++;
            }
            lines[i] = parts.join(' ');
        }
        objFile = lines.join("\n");
    }
```

Obrázek 49: Zapsání nových pozic vrcholů modelu

Zdroj: vlastní zpracování

Následně se ověřuje, zda je v souboru uveden odkaz na materiálovou knihovnu. Pokud chybí, vloží se na začátek souboru. Pokud je zapnuta volba zachování výchozího materiálu, přidá se příkaz usemtl defaultMat před první definici plochy, jak je popsáno ve struktuře OBJ souboru (Chakravorty, 2023). Upravený soubor se následně uloží a stáhne jako textový soubor pomocí javascriptu (Khuntwal, 2024).

```
    let lines2 = objFile.split('\n');

    for (let i = 0; i < lines2.length; i++) {
        let line = lines2[i];
        let parts = line.trim().split(/\s+/);

        if (parts[0] === "mtllib") {
            includedMtl = parts.slice(1).join(' ');
            break;
        }
    }
    if (!includedMtl) {
        if (originalMTLfile[0] && originalMTLfile[0].name) {
            lines2.unshift("mtllib " + originalMTLfile[0].name);
            includedMtl = originalMTLfile[0].name;
        }
        else {
            includedMtl = originalOBJfileName.replace(/\.obj$/, '.mtl');
            lines2.unshift("mtllib " + includedMtl);
        }
    }

    if (keepDefaultMat && matName[0].id === -1) {
        for (let i = 0; i < lines2.length; i++) {
            let line = lines2[i];
            let parts = line.trim().split(/\s+/);

            if (parts[0] === "f") {
                lines2.splice(i, 0, "usemtl defaultMat");
                break;
            }
        }

    }
    objFile = lines2.join('\n');


    const fileContent = objFile;
    const blob = new Blob([fileContent], { type: "text/plain" });
    const url = URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.style.display = "none";
    a.href = url;
    a.download = originalOBJfileName;

    document.body.appendChild(a);
    a.click();
    document.body.removeChild(a);
    URL.revokeObjectURL(url);
}
```

Obrázek 50: Další úprava dat modelu a export OBJ souboru

Zdroj: vlastní zpracování

2.8.2 Export MTL

Export materiálů začíná zjištěním všech definovaných materiálů v původních MTL souborech a všech materiálů použitých v OBJ souboru. Pokud některé chybí, doplní se.

```
let definedMats = [];
for (let i = 0; i < originalMTLfile.length; i++) {
    let testFile = originalMTLfile[i].result;
    let testLines = testFile.split('\n');
    for (let i = 0; i < testLines.length; i++) {
        let line = testLines[i];
        let parts = line.trim().split(/\s+/);
        if (parts[0] === "newmtl") {
            definedMats.push(parts.slice(1).join(' '));
        }
    }
}
let matsUsedInObjFile = [];
let testFile = originalOBJfile;
let lines = testFile.split('\n');
for (let i = 0; i < lines.length; i++) {
    let line = lines[i];
    let parts = line.trim().split(/\s+/);
    if (parts[0] === "usemtl") {
        matsUsedInObjFile.push(parts.slice(1).join(' '));
    }
}
if (keepDefaultMat && matName[0].id === -1) matsUsedInObjFile.unshift(matName[0].name);
const missingMats = matsUsedInObjFile.filter(mat => !definedMats.includes(mat));
```

Obrázek 51: Zjištění chybějících materiálů

Zdroj: vlastní zpracování

Pro každý materiál se aktualizuje barva podle hodnot uložených v poli colorArr. RGB složky se převádějí na rozsah od 0 do 1. Pokud je aktivní volba odstranění absolutní cesty k obrázkům, u příkazu map_Kd se ponechá pouze název souboru, jak je popsáno ve struktuře MTL souboru (Chakravorty, 2023).

```
for (let i = 0; i < originalMTLfile.length; i++) {
    if (i === 0) {
        let fileContent = originalMTLfile[i].result;
        let newFileContent = "";
        const chunks = fileContent.split('newmtl');
        for (let chunkcounter = 0; chunkcounter < chunks.length; chunkcounter++) {
            const chunk = chunks[chunkcounter];
            const lines = chunk.split('\n');
            let matName = null;

            for (let linesCounter = 0; linesCounter < lines.length; linesCounter++) {
                const line = lines[linesCounter];
                const parts = line.trim().split(/\s+/);
                const keyword = parts[0];

                if (!linesCounter) matName = keyword;
                else {
                    if (keyword === "Kd") {
                        const foundMaterial = colorArr.find(material => material.name === matName);
                        if (foundMaterial) {
                            lines[linesCounter] = "Kd " + (foundMaterial.color[0] / 255).toFixed(6)
                                + " " + (foundMaterial.color[1] / 255).toFixed(6)
                                + " " + (foundMaterial.color[2] / 255).toFixed(6);
                        }
                    }
                    else if (keyword === "map_Kd" && removeAbsolutePathForImg) {
                        lines[linesCounter] = "map_Kd " + parts.slice(1).join(' ').split(/[\\/]/).pop();
                    }
                }
            }
            if (chunkcounter !== 0) newFileContent += "newmtl " + lines.join("\n")
            else newFileContent += lines.join("\n");
        }
        newFileContent += "\n";

        for (let i = 0; i < missingMats.length; i++) {
            const foundMaterial = colorArr.find(material => material.name === missingMats[i]);
            if (foundMaterial) {
                newFileContent += "newmtl " + missingMats[i] + "\n";
                newFileContent += firstPartMtl;
                newFileContent += "Kd " + (foundMaterial.color[0] / 255).toFixed(6)
                    + " " + (foundMaterial.color[1] / 255).toFixed(6)
                    + " " + (foundMaterial.color[2] / 255).toFixed(6) + "\n";
                newFileContent += secondPartMtl + "\n";
            }
        }
        newFileContent = newFileContent.slice(0, -1);
```

Obrázek 52: Úprava obsahu MTL souboru

Zdroj: vlastní zpracování

Další úpravy na MTL souboru jsou prováděny podobným způsobem, jak je popsáno výše, a následně jsou exportovány stejným postupem jako v části 2.8.1 do textové podoby.

2.8.3 Export obrázků

Pokud je aktivní volba stahování obrázků, projde se pole imagesArray a pro každý obrázek se vytvoří odkaz, který spustí jeho stažení pod původním názvem (Khuntwal, 2024). Tím se zajistí, že všechny obrázky spojené s modelem budou uloženy spolu s OBJ a MTL soubory.

```
if (downloadImgFiles) {
    for (let i = 0; i < imagesArray.length; i++) {
        const link = document.createElement("a");
        link.href = imagesArray[i].image.src;
        link.download = imagesArray[i].name;
        document.body.appendChild(link);
        link.click();
        document.body.removeChild(link);
    }
}
```

Obrázek 53: Export obrázků

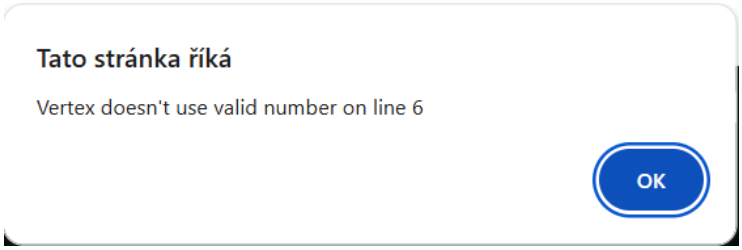
Zdroj: vlastní zpracování

2.9 Testování

V rámci práce bylo provedeno základní ověření správnosti vytvořeného programu. Testování se zaměřilo na správné načítání vstupních dat a vykreslení 3D modelu. Správnost fungování byla posuzována kontrolou výsledného vizuálního výstupu.

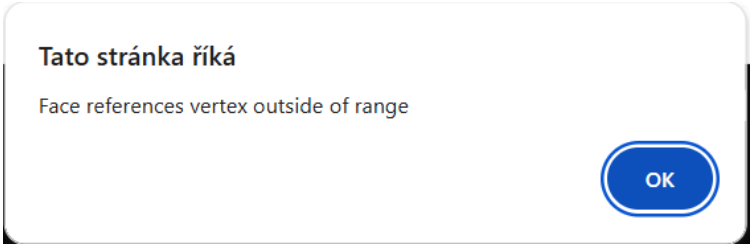
2.9.1 Pokus o načtení souboru s neplatným obsahem

Při testování bylo prověřeno chování programu v situaci, kdy je načten soubor s neplatným obsahem. Byly použity upravené soubory ve formátu OBJ, ve kterých došlo k porušení struktury dat. V jednom případě byl poškozen obsah řádku definujícího vrchol, ve druhém případě byl celý řádek odstraněn. Bylo sledováno, zda program na chybný vstup správně reaguje.



Obrázek 54: Chybová hláška při poškozeném řádku vrcholu
Zdroj: vlastní zpracování

Obrázek 54 ukazuje výsledek testu s upraveným řádkem vrcholu. Program chybný řádek detekoval správně.

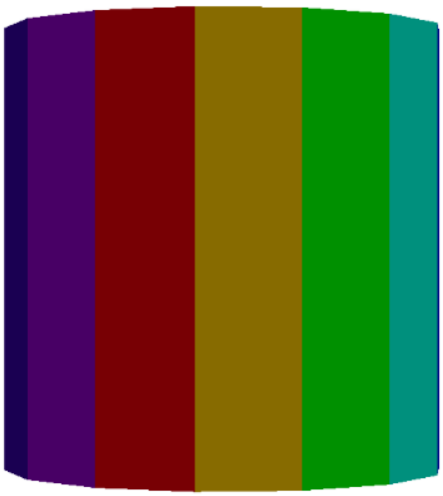


Obrázek 55: Chybová hláška při chybějícím řádku vrcholu
Zdroj: vlastní zpracování

Výsledek testu s odstraněným řádkem vrcholu je zobrazen na obrázku 55, který ukazuje, že program tuto neplatnou situaci rovněž správně rozpoznal.

2.9.2 Načtení modelu s materiály

Dále byla otestována správnost načtení modelu a přiřazení materiálů ze souboru MTL. Každá plocha modelu měla odlišnou barvu. Cílem bylo zjistit, zda jsou jednotlivým plochám správně přiřazeny odpovídající materiály.



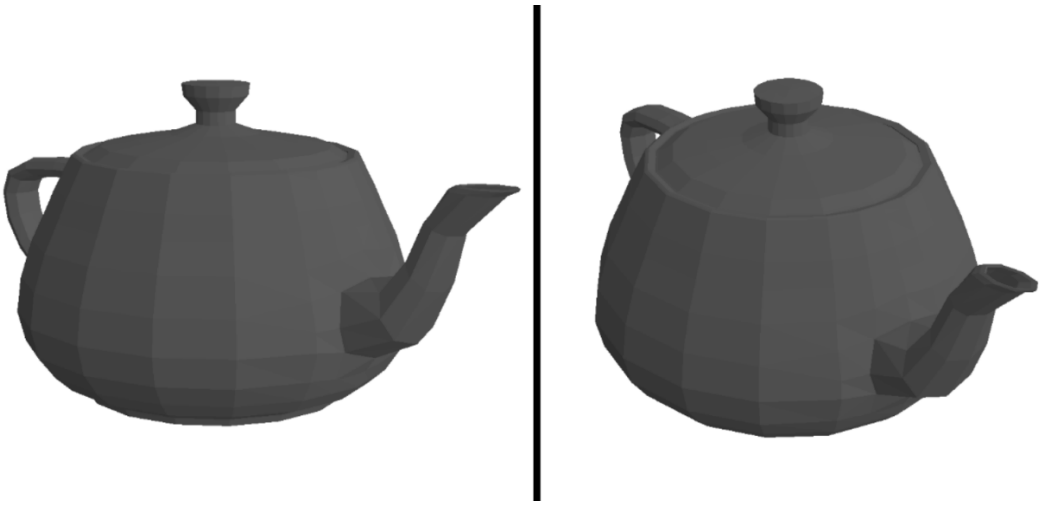
Obrázek 56: Vykreslený 3D model s rozdílnými plochami

Zdroj: vlastní zpracování

Na obrázku 56 je zobrazen načtený model s barevně odlišenými plochami. Je patrné, že model i materiály byly načteny a aplikovány správně.

2.9.3 Rotace a zobrazení modelu

Součástí ověřování byla také kontrola správného chování při otáčení modelu. Po načtení modelu byla provedena jeho rotace o několik stupňů. Účelem bylo zjistit, zda dochází ke správnému přepočtu zobrazení včetně perspektivy a stínování.



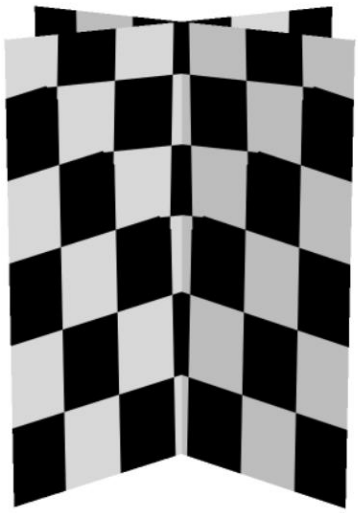
Obrázek 57: Model před a po otočení

Zdroj: vlastní zpracování

Průběh zobrazení modelu před a po otočení můžeme vidět na obrázku 57. Je zřejmé, že vykreslení proběhlo správně.

2.9.4 Vykreslení překrývajících se ploch a textur

Dále bylo posouzeno správné vykreslení překrývajících se ploch a aplikace textur za použití modelu obsahujícího protínající se plochy a texturu načtenou z obrázku. Při tom se kontrolovalo, zda jsou plochy správně vykresleny přes sebe a zda je textura správně zobrazena.



Obrázek 58: Vykreslený model s protínajícími se plochami a texturou

Zdroj: vlastní zpracování

Správnost vykreslení protínajících se ploch i aplikace textury je potvrzena výsledným vizuálním výstupem, který je uveden na obrázku 58.

2.9.5 Export modelu

Nakonec byla zkontrolována funkce exportu modelu. Program umožňuje stáhnout upravený soubor OBJ, soubor MTL a související obrázek s texturou. Při testu bylo sledováno, zda jsou všechny soubory správně vygenerovány a zda obsahují aktuální stav modelu po případných úpravách. Funkce exportu byla ověřena načtením souborů do programu podporujícího OBJ formát, čímž bylo potvrzeno, že soubory byly správně vytvořeny.

Obrázek 59: Exportované soubory OBJ, MTL a textura

Zdroj: vlastní zpracování

Obrázek 59 ukazuje soubory, jež byly vygenerovány při exportu. Každý soubor lze dále využít v softwaru zaměřeném na práci s 3D objekty.

Závěr

V práci byl vytvořen program v prostředí HTML a JavaScriptu využívající HTML Canvas v 2D kontextu, který umožňuje vykreslování 3D objektů ve formátu OBJ. Byly implementovány rotační matice pro manipulaci s objektem, perspektivní projekce, stínování metodou flat shading, mapování textur pomocí afinního zobrazení a zpracování obrazu prostřednictvím z-bufferu a SSAA. Program umožňuje pohyb s vrcholy modelu, změnu barvy objektu a export výsledného modelu. Součástí řešení jsou testovací modely, které umožňují ověření funkčnosti i bez nahraného souboru.

Podařilo se implementovat základní principy vykreslování 3D grafiky bez použití 3D knihoven a jednotlivé kroky byly realizovány postupně tak, aby bylo možné sledovat jejich fungování, a to i v prostředí webového prohlížeče.

Popsané principy však lze přenést do jiných prostředí a jazyků, protože práce ukazuje obecné vzorce a postupy, které nejsou závislé na konkrétní technologii. Výsledné řešení může sloužit jako výukový nástroj pro studenty informatiky a zájemce o počítačovou grafiku, protože názorně ukazuje postupy, které stojí za vykreslováním 3D scén.

Omezení práce vyplývá z použití flat shadingu, který neumožňuje využití všech informací z MTL souboru. Rozšíření směrem k výpočtu barvy na úrovni pixelů by umožnilo práci se všemi materiálovými parametry, avšak vyžadovalo by výraznou optimalizaci. Další rozšíření může spočívat v rozšíření podpory pro pokročilejší úpravy modelů a práci s jejich strukturou, což by umožnilo lepší práci s modelem.

Použití HTML Canvas a JavaScriptu se osvědčilo jako vhodná volba pro demonstraci principů. Při řešení se však projevily limity výkonu způsobené tím, že jazyk je interpretovaný a grafické výpočty probíhají na procesoru, což se projevilo zejména při mapování obrázků na model. Přesto bylo dosaženo funkčního řešení, které splňuje stanovené cíle.

Seznam použité literatury

AUGGIE. *One-point perspective formula*. Online. Mathematics Stack Exchange. 2017. Dostupné z: https://math.stackexchange.com/questions/2337183/one-point-perspective-formula/. [cit. 2025-08-26].

BAKER, Martin. *Maths - Normals*. Online. EuclideanSpace. Nedatováno. Dostupné z: https://www.euclideanspace.com/maths/algebra/vectors/applications/normals/index.htm. [cit. 2025-08-28].

BANSAL, Sahil. *A-Buffer Method*. Online. GeeksforGeeks. 2018, 2021. Dostupné z: https://www.geeksforgeeks.org/computer-graphics/a-buffer-method/. [cit. 2025-09-28].

BEAKER. *Inverting 3d rotation sequence*. Online. Stack Overflow. 2014. Dostupné z: https://stackoverflow.com/questions/22964800/inverting-3d-rotation-sequence. [cit. 2025-12-05].

BORGEN, Andreas. *Warp image on canvas*. Online. GitHub Gist. 2018. Dostupné z: https://gist.github.com/Sphinxxxx/b105be479d8745c6145d69d16557401e. [cit. 2025-09-06].

BRAINKART. *Warnock algorithm*. Online. BrainKart. 2016. Dostupné z: https://www.brainkart.com/article/Warnock-algorithm_5717/. [cit. 2025-09-27].

BURZYNSKI, Denny. *Rotation Matrices in 3-Dimensions*. Online. Mathematics LibreTexts. 2023. Dostupné z: https://math.libretexts.org/Bookshelves/Applied_Mathematics/Mathematics_for_Game_Developers_%28Burzynski%29/04%3A_Matrices/4.06%3A_Rotation_Matrices_in_3-Dimensions/. [cit. 2025-08-24].

CAD EVANGELIST. *The History of 3D Rendering: From Teapots to Toy Stories*. Online. BluEnt. 2023. Dostupné z: https://www.bluentcad.com/blog/history-of-3d-rendering/. [cit. 2025-08-19].

CORKE, Peter. *Roll-pitch-yaw angles*. Online. Peter Corke. 2017, 2018. Dostupné z: https://petercorke.com/robotics/roll-pitch-yaw-angles/. [cit. 2025-08-24].

CUEMATH. *Linear Interpolation Formula*. Online. Cuemath. 2021. Dostupné z: https://www.cuemath.com/linear-interpolation-formula/. [cit. 2025-10-18].

DOWN, Tim. *RGB to hex and hex to RGB*. Online. Stack Overflow. 2011, 2022. Dostupné z: https://stackoverflow.com/questions/5623838/rgb-to-hex-and-hex-to-rgb. [cit. 2025-12-06].

ELLISON, Gemma. *Unity: How to Normalize a Vector*. Online. Wayline. 2023. Dostupné z: https://www.wayline.io/blog/unity-how-to-normalize-a-vector/. [cit. 2025-08-29].

ENA, Alejandro. *Euler angles, rotations and gimbal lock*. Online. Medium. 2022. Dostupné z: https://medium.com/@lalesena/euler-angles-rotations-and-gimbal-lock-brief-explanation-de1d4764170/. [cit. 2025-08-24].

EVANSON, Nick. *How 3D Game Rendering Works: Anti-Aliasing*. Online. TechSpot. 2021. Dostupné z: https://www.techspot.com/article/2219-how-to-3d-rendering-anti-aliasing/. [cit. 2025-10-01].

FRANTZ, Andrew. *Backface Culling*. Online. EECS at Michigan. 2006. Dostupné z: https://www.eecs.umich.edu/courses/eecs487/f06/sa/ajfrantz.pdf. [cit. 2025-10-04].

GAMBETTA, Gabriel. Shading. In: *Computer graphics from scratch: a programmer's introduction to 3D rendering*. San Francisco: No Starch Press, 2021, s. 164–165. ISBN 978-1-7185-0077-8.

GRAMS, Ross. *Mouse movement tracking*. Online. Defold game engine forum. 2019. Dostupné z: https://forum.defold.com/t/mouse-movement-tracking-solved/58041. [cit. 2025-10-26].

GRIFFIN, Jason. *The Painters Algorithm*. Online. Albas Music. 2022. Dostupné z: https://www.albasmusic.com/the-painters-algorithm/. [cit. 2025-09-27].

CHAKRAVORTY, Dibya. *The OBJ File Format – Simply Explained*. Online. All3DP. 2023. Dostupné z: https://all3dp.com/2/obj-file-format-simply-explained/. [cit. 2025-08-31].

CHHABRA, Manas. *HTML input type="color"*. Online. GeeksforGeeks. 2019, 2024. Dostupné z: https://www.geeksforgeeks.org/html/html-input-typecolor/. [cit. 2025-12-06].

IM, Mee Seong. *Find a normal vector using cross product*. Online. Mathematics Stack Exchange. 2017. Dostupné z: https://math.stackexchange.com/questions/2461485/find-a-normal-vector-using-cross-product/. [cit. 2025-08-29].

JAVASCRIPT TUTORIAL. *JavaScript FileReader*. Online. JavaScript Tutorial. 2021. Dostupné z: https://www.javascripttutorial.net/web-apis/javascript-filereader/. [cit. 2025-10-12].

KAWALPREET. *Perspective Projection and its Types*. Online. GeeksforGeeks. 2020, 2025. Dostupné z: https://www.geeksforgeeks.org/computer-graphics/perspective-projection-and-its-types/. [cit. 2025-08-26].

KHUNTWAL, Siddharth. *How to Download a File Using JavaScript*. Online. Flexiple. 2024. Dostupné z: https://flexiple.com/javascript/download-flle-using-javascript. [cit. 2025-12-12].

MAŁEK, Piotr a ŻUŁAWIŃSKA, Julia. *Angle Conversion Calculator*. Online. Omni Calculator. 2015, 2025. Dostupné z: https://www.omnicalculator.com/conversion/angle-converter#what-is-a-radian-and-how-to-convert-degrees-to-radians. [cit. 2025-11-01].

MOZILLA FOUNDATION. *FileReader*. Online. MDN Web Docs. 2025. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/FileReader. [cit. 2025-10-11].

NOMINAL ANIMAL. *One-point perspective formula*. Online. Mathematics Stack Exchange. 2017. Dostupné z: https://math.stackexchange.com/questions/2337183/one-point-perspective-formula/. [cit. 2025-08-26].

OPENREPLAY TEAM. *RequestAnimationFrame vs setTimeout: When to Use Each*. Online. OpenReplay. 2025. Dostupné z: https://blog.openreplay.com/requestanimationframe-settimeout-use/. [cit. 2025-11-08].

OVERBY, Ronnie. *Normalise orientation between 0 and 360*. Online. Stack Overflow. 2017. Dostupné z: https://stackoverflow.com/questions/1628386/normalise-orientation-between-0-and-360. [cit. 2025-11-01].

PEOPLES, Connor. *Why do browsers disallow accessing files from local file system even if the HTML document is also on the local file system?* Online. Information Security Stack Exchange.

2019. Dostupné z: https://security.stackexchange.com/questions/201208/why-do-browsers-disallow-accessing-files-from-local-file-system-even-if-the-html. [cit. 2025-10-11].

REID, Linden. *Simple Mesh Tessellation & Triangulation Tutorial*. Online. Linden Reid. 2017. Dostupné z: https://lindenreidblog.com/2017/12/03/simple-mesh-tessellation-triangulation-tutorial/. [cit. 2025-08-21].

SAMUEL. *How to prevent native browser default pinch to zoom behavior?* Online. Stack Overflow. 2020. Dostupné z: https://stackoverflow.com/questions/61114830/how-to-prevent-native-browser-default-pinch-to-zoom-behavior. [cit. 2025-11-07].

SHELDON, Robert. *Dot product (scalar product)*. Online. TechTarget. 2022. Dostupné z: https://www.techtarget.com/whatis/definition/dot-product-scalar-product. [cit. 2025-08-30].

SLYNCH. *Texture Coordinates – D3D vs. OpenGL*. Online. PureDev Software. 2018. Dostupné z: https://www.puredevsoftware.com/blog/author/slynchpuredevsoftware-com/. [cit. 2025-10-25].

SOKOLOV, Dmitry V. *Triangle rasterization*. Online. Playing with code. 2025. Dostupné z: https://haqr.eu/tinyrenderer/rasterization/. [cit. 2025-11-15].

THAKUR, Arjun. *Maximum size of a <canvas> element in HTML*. Online. Tutorialspoint. 2019. Dostupné z: https://www.tutorialspoint.com/Maximum-size-of-a-canvas-element-in-HTML. [cit. 2025-10-25].

TUTORIALSPOINT. *Computer Graphics - Visible Surface Detection*. Online. TutorialsPoint. Nedatováno. Dostupné z: https://www.tutorialspoint.com/computer_graphics/visible_surface_detection.htm. [cit. 2025-09-27].

VON GAGERN, Martin. *Center point calculation for 3D polygon*. Online. Stack Overflow. 2014. Dostupné z: https://stackoverflow.com/questions/22465832/center-point-calculation-for-3d-polygon. [cit. 2025-10-19].

WEISSTEIN, Eric. *Affine Transformation*. Online. Wolfram MathWorld. 2004. Dostupné z: https://mathworld.wolfram.com/AffineTransformation.html. [cit. 2025-09-06].

WOLFE, Alan. *How to translate object to origin?* Online. Computer Graphics Stack Exchange. 2016. Dostupné z: https://computergraphics.stackexchange.com/questions/4310/how-to-translate-object-to-origin. [cit. 2025-10-19].

YADAV, Amit. *Z-Buffer Algorithm Explained*. Online. Medium. 2025. Dostupné z: https://medium.com/@amit25173/z-buffer-algorithmexplained-b6cbb01e6b3f/. [cit. 2025-09-28].

Přílohy

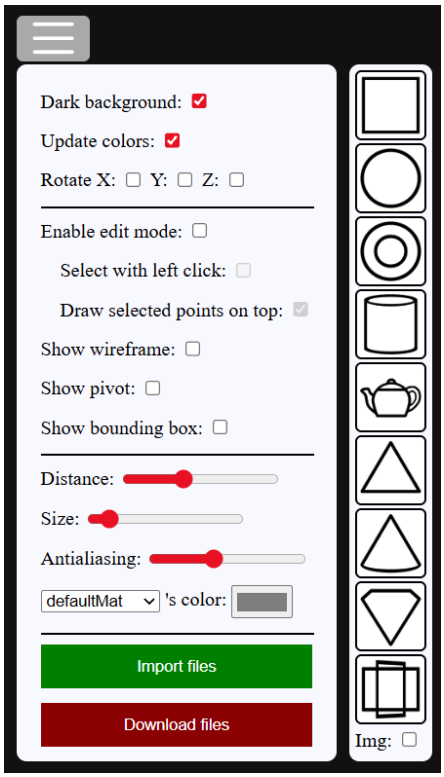
Přílohy A Manuál

Příloha A.1 Základní ovládání

Rotace modelu je prováděna pomocí levého tlačítka myši. Při jeho držení je zaznamenáván pohyb myši a na jeho základě je model otáčen kolem osy X a Y. Při současném stisku klávesy Ctrl je otáčení převedeno na pohyb kolem osy Z. Při stisku klávesy Shift je rychlost pohybu snížena.

Posun modelu po obrazovce je aktivován pravým tlačítkem myši. Při jeho držení je pohyb myši převáděn na posun daného modelu po obrazovce.

V menu jsou umístěna veškerá potřebná nastavení pro ovládání programu. Menu je umístěno v levém horním rohu obrazovky a je otevřeno po stisku levého tlačítka myši.



Obrázek 60: Ukázka menu
Zdroj: Vlastní zpracování

Příloha A.2 Vizuální nastavení a automatická rotace v menu

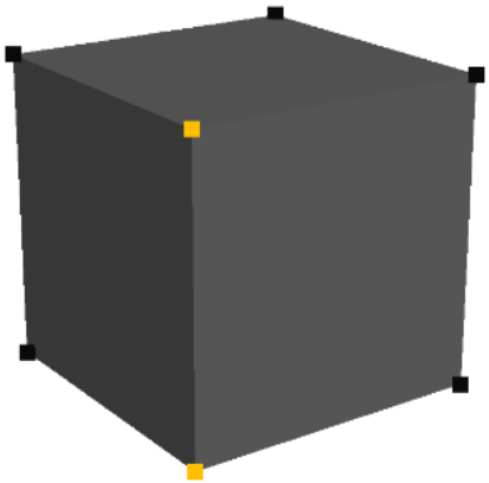
Volba Dark background slouží k přepínání barvy pozadí mezi černou a bílou. Před přepnutím do světlého režimu je uživatel dotázán, zda má být změna skutečně provedena.

Volba Update colors určuje, zda má být aktualizováno stínování modelu. Při deaktivaci je stínování ponecháno ve stavu odpovídajícím poslední provedené aktualizaci bez ohledu na další změny orientace modelu.

Volby Rotate X, Rotate Y a Rotate Z umožňují automatickou rotaci modelu kolem příslušné osy. Po aktivaci je model otáčen bez jakéhokoli vstupu od uživatele, ale zásah do rotace je stále možný. Lze spustit rotaci kolem více os současně.

Příloha A.3 Úprava modelu a zobrazovací pomůcky v menu

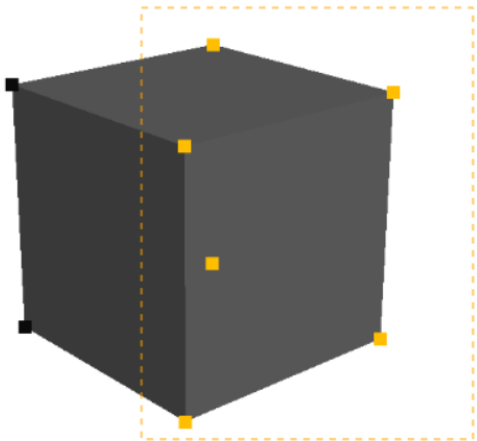
Volba Enable edit mode aktivuje režim úprav. Po aktivaci jsou na všech vrcholech modelu zobrazeny černé čtverce a jsou zpřístupněny volby Select with left click a Draw selected points on top. Výběr vrcholu je proveden stiskem levého tlačítka myši nad příslušným vrcholem. Po výběru lze s vrcholem pohybovat pomocí pohybu myši při stisknutém pravém tlačítku myši. Při výběru jiného vrcholu je původní výběr nahrazen. Při držení klávesy Shift lze vybrat více vrcholů současně bez zrušení předchozího výběru.



Obrázek 61: Ukázka vybraných a nevybraných bodů

Zdroj: vlastní zpracování

Výběr více vrcholů lze provést také pomocí stisknutého kolečka myši. Při jeho držení je pohybem myši vytvořen výběrový obdélník a po uvolnění kolečka jsou vybrány všechny vrcholy nacházející se uvnitř obdélníku. Pokud není k dispozici kolečko myši, může jeho funkci převzít levé tlačítko. Po aktivaci volby Select with left click začne levé tlačítko v editačním režimu sloužit k výběru místo k rotaci.

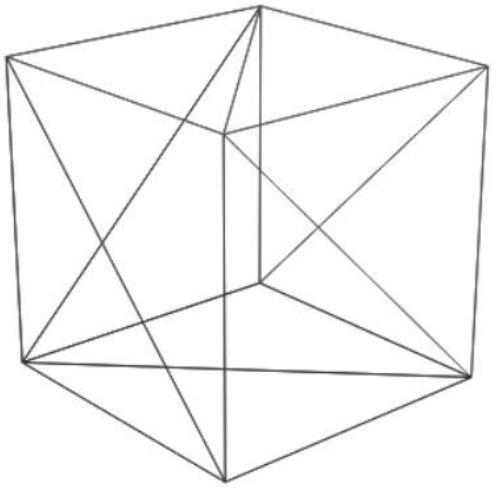


Obrázek 62: Ukázka výběru více vrcholů a možnosti zobrazení vybraných vrcholů na vrchu modelu

Zdroj: vlastní zpracování

Volba Draw selected points on top zajišťuje, že vybrané vrcholy jsou zobrazeny nad ostatními částmi modelu. Při deaktivaci mohou být vybrané vrcholy zakryty geometrií modelu, pokud se nachází za ní.

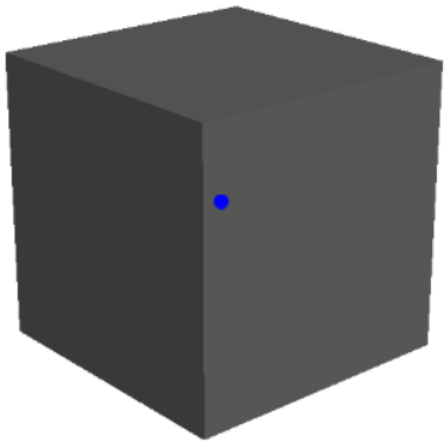
Volba Show wireframe zobrazí pouze hrany modelu bez vyplněných ploch. Zobrazení je provedeno pomocí barevných hran bez použití textur.



Obrázek 63: Ukázka zobrazení drátového modelu

Zdroj: vlastní zpracování

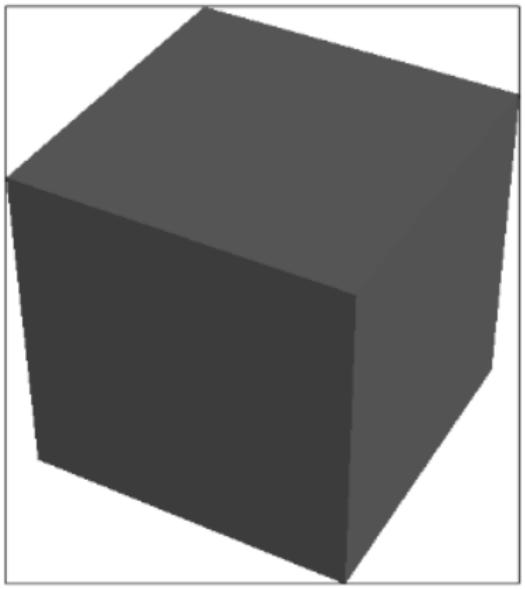
Volba Show pivot zobrazí střed modelu umístěný v souřadnicích 0, 0, 0. Zobrazený bod ukazuje, kolem jakého místa se model při rotaci otáčí.



Obrázek 64: Ukázka zobrazení středu rotace modelu

Zdroj: vlastní zpracování

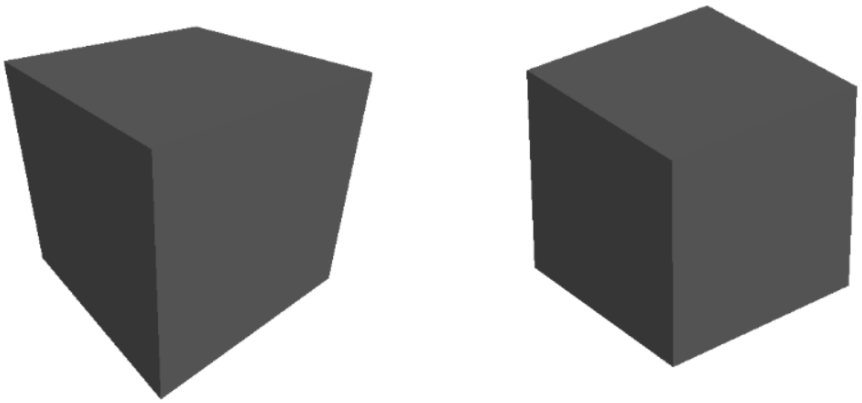
Volba Show bounding box zobrazí obdélník odpovídající nejvyšším a nejnižším souřadnicím modelu. Obdélník ukazuje prostor, ve kterém se model nachází.



Obrázek 65: Ukázka zobrazení obdélníku ukazující minimální a maximální rozměry modelu
Zdroj: vlastní zpracování

Příloha A.4 Nastavení projekce, měřítka a barev

Volba Distance určuje vzdálenost perspektivní projekce. Při nižší hodnotě je model zobrazen blíže k pozorovateli a dochází k výraznějšímu perspektivnímu zkreslení. Při vyšší hodnotě je zkreslení menší a model působí vzdáleněji.

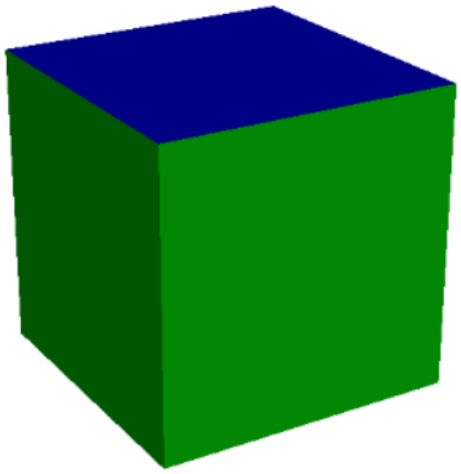


Obrázek 66: Ukázka zvětšení a zmenšení perspektivní projekce
Zdroj: vlastní zpracování

Volba Size mění měřítko modelu. Změnou hodnoty je model zvětšen nebo zmenšen bez ovlivnění jeho tvaru či orientace. Stejného výsledku může být dosaženo otáčením kolečka myši.

Volba Antialiasing určuje míru vyhlazování hran. Vyšší hodnota zajišťuje jemnější a hladší obraz, avšak s vyšší výpočetní náročností, což může zpomalit běh programu.

Nabídka [název materiálu]'s color umožňuje měnit barvu jednotlivých částí modelu. Části jsou rozděleny podle hodnot usemtl uvedených v souboru OBJ. Po výběru konkrétní části je barva upravena pomocí vstupu pro výběr barvy.



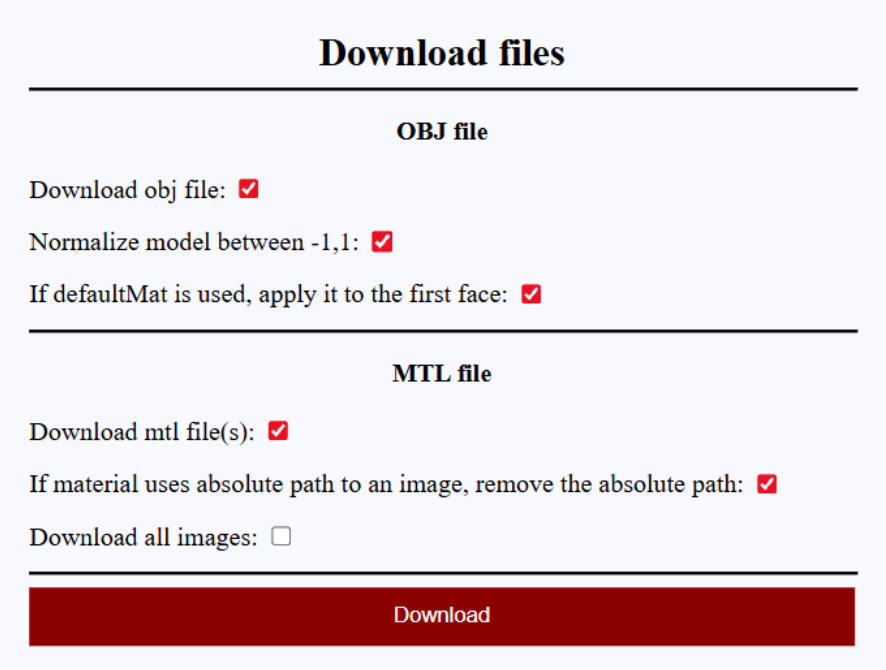
Obrázek 67: Ukázka změny barvy modelu
Zdroj: vlastní zpracování

Příloha A.5 Import a export souborů

Tlačítko Import files otevře dialog pro výběr souborů. Lze vybrat více souborů s příponami .jpg, .jpeg, .png, .obj a .mtl. Podmínkou je přítomnost právě jednoho souboru .obj, který musí být součástí výběru. Po potvrzení jsou vybrané soubory zpracovány a načteny do programu.

Tlačítko Download files otevře nabídku pro export modelu a souvisejících dat. Volba Download obj file zajistí vytvoření a stažení souboru .obj. Volba Normalize model between -1,1 upraví souřadnice modelu tak, aby se všechny hodnoty nacházely v rozsahu od −1 do 1. Volba If defaultMat is used, apply it to the first face přidá do exportovaného souboru .obj výchozí materiál v případě, že model neobsahuje vlastní definici materiálu pro všechny plochy.

V části MTL file je možné stáhnout jeden nebo více souborů .mtl. Volba If material uses absolute path to an image, remove the absolute path odstraní z materiálového souboru absolutní cestu k obrázku a ponechá pouze název souboru. Volba Download all images stáhne všechny obrázky, které byly načteny společně s modelem.



Obrázek 68: Ukázka menu pro export 3D modelu

Zdroj: Vlastní zpracování

Příloha A.6 Výběr předpřipravených modelů

V pravé části rozhraní je umístěno menu s předpřipravenými modely. Každý model je reprezentován svým náhledovým obrázkem. Po výběru je načten odpovídající model s výchozím materiálem bez použití textur.

Volba Img ve spodní části menu umožňuje načíst model společně s jeho texturou. Při aktivaci této volby je při výběru modelu načten také přidružený obrázek textury.

Přílohy B Webová adresa

Práce je dostupná na následující webové adrese: https://alpha.kts.vspj.cz/~cech24/projekt/