Elektronická podpora pro vybrané oblasti počítačové grafiky

Pavel Mikulecký

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Pavel Mikulecký**
Osobní číslo:	**A20412**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Elektronická podpora pro vybrané oblasti počítačové grafiky**
Téma práce anglicky:	**Electronic Support for Selected Areas of Computer Graphics**

Zásady pro vypracování

1. Zpracujte literární rešerši na problematiku zabývající se oblastí teorie informace a možnostmi jejího využití.
2. Pro vybrané oblasti teorie počítačové grafiky připravte a vytvořte dostatečné množství podkladů, které budou obsahovat nezbytnou teorii a dostatečný počet názorných příkladů.
3. Vytvořte také podpůrné SW výstupy s využitím vybraných SW nástrojů, které by měly sloužit jako zpětná vazba k vytvořeným podkladům pro vybrané oblasti počítačové grafiky.
4. Navrhněte formu prezentace a zpracování získaných a vytvořených podkladů a proveďte její realizaci a vyhodnocení.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Tristan, Bunn. Learn Python Visually: Creative Coding with Processing.Py. 1. San Francisco: No Starch Press, 2021. ISBN 978-1-7185-0097-6.
2. Gambetta, Gabriel. Computer Graphics from Scratch: A Programmer's Introduction to 3D Rendering. 1. San Francisco: No Starch Press, 2021. ISBN 978-1-7185-0077-8.
3. Žára, Jiří. Moderní počítačová grafika. 2., přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.
4. Dobeš, Michal. Zpracování obrazu a algoritmy v C#. 1. Praha: BEN – technická literatura, 2008. ISBN 978-807-3002--336.
5. Lupiani, Isabel. Blender Scripting With Python: Write Scripts to Build Your Own 3D Models. 1. San Francisco: No Starch Press, 2018. ISBN 9781593278724.

Vedoucí bakalářské práce: **Ing. Pavel Navrátil, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 25. 5. 2023

Pavel Mikulecký v.r.
podpis studenta

ABSTRAKT

Bakalářská práce „Elektronická podpora pro vybrané oblasti počítačové grafiky“ se orientuje na problematiku počítačové grafiky. První teoretická část popisuje základní matematické operace a pojmy a sleduje jejich význam v oblasti počítačové grafiky. V následující části jsou představeny nástroje pro editaci a zpracování obrazu. Je zde kladen důraz na orientaci v uživatelském prostředí a na oblasti využití daných editačních nástrojů. Třetí část se zaměřuje na problematiku grafických objektů z pohledu jejich matematických definic, vlastností a využití v praktické tvorbě grafických objektů. Čtvrtá část pojednává o transformaci pomocí transformační matice. Krom definice transformační matice jsou zde zmíněny základní transformace, jako je posun, rotace nebo změna měřítka. Následující části se zabývají principem rasterizace a vektorizace a jejich významem v počítačové grafice. Dále jsou zde zmíněny základní barevné modely a grafické formáty s popisem jejich specifických vlastností a z toho vyplývajících vhodných oblastí využití. Praktická část se zaměřuje na tvorbu podpůrných SW výstupů s využitím zmíněných SW nástrojů. Tyto výstupy se opírají o sepsanou teorii v předešlé části práce a mají podobu interaktivních scriptů Processing.py, pluginů pro SW nástroje nebo jiných vizualizací a příkladů. Kompletní výstupy jsou uložené v příloze a roztříděné dle kapitol.

Klíčová slova: Blender, GIMP, Inkscape, konvoluce, počítačová grafika, Processing.py, Python, grafické objekty, grafické úpravy, teorie informace, transformace

ABSTRACT

The bachelor thesis "Electronic Support for Selected Areas of Computer Graphics" focuses on computer graphics. The first theoretical part describes basic mathematical operations and concepts and traces their importance in the field of computer graphics. In the following part, tools for image editing and processing are introduced. Emphasis is placed on orientation in the user environment and on the areas of application of the editing tools. The third part focuses on the issue of graphical objects in terms of their mathematical definitions, properties and use in the practical creation of graphical objects. The fourth part deals with the transformation of the post-power transformation matrix. In addition to the definition of the transformation matrix, basic transformations such as shift, rotation, and scaling are mentioned. The following sections deal with the principles of rasterization and vectorization and their

importance in computer graphics. Furthermore, basic colour models and graphic formats are mentioned, with a description of their specific properties and the resulting appropriate areas of application. The practical part focuses on the creation of sub-supporting SW outputs using the mentioned SW tools. These outputs are based on the theory presented in the previous part of the thesis and take the form of interactive Processing.py scripts, plugins for SW tools or other visualizations and diagrams. The complete outputs are stored in the appendix and categorized by chapter.

Keywords: Blender, GIMP, Inkscape, convolution, computer graphics, Processing.py, Python, graphical objects, graphical editing, information theory, transformations

Děkuji touto cestou Ing. Pavlu Navrátilovi, Ph.D. za pomoc, cenné rady a čas, který mi věnoval při tvorbě této bakalářské práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato bakalářská práce se zabývá studiem základů vybraných oblastí počítačové grafiky a tvorbou podkladů s využitím zvolených softwarových nástrojů.

Jako základní prvek lidské komunikace má grafika hluboké kořeny sahající do historie lidstva. Smyslem grafiky bylo a stále je komunikovat a prezentovat své myšlenky. Dnes, s příchodem digitální éry, se tato forma komunikace vyvíjí a adaptuje pomocí počítačové grafiky a její význam je nyní nepřehlédnutelný ve většině aspektech našeho života – od digitálních médií, přes vývoj her, až po vědeckou vizualizaci a simulace.

S masovou adopcí počítačové grafiky přišlo také velké množství softwarových nástrojů, které jsou dostupné pro širokou veřejnost a jejich uživatelská přívětivost s časem neustále stoupá. Ačkoli je tato dostupnost nesmírně přínosná pro naši společnost, základní porozumění principům a technickým aspektům počítačové grafiky zůstává nadále klíčové, pokud je naším cílem co největší efektivita a kvalita výsledné práce.

Při tvorbě digitálního obsahu, ať už je to web, hra, nebo jakákoliv vizuální prezentace, je důležité vědět, jak fungují různé grafické komponenty a technologie. Příkladem může být tvorba webové stránky, kdy je nezbytné optimalizovat grafický obsah tak, aby byl snadno přístupný a rychle se načítal pro uživatele. Bez pochopení, jak fungují formáty obrázků a komprese, může tento proces vést k neoptimálním výsledkům.

Bakalářská práce je členěna na část teoretickou a praktickou. Je rozdělena do třinácti kapitol. Prvních sedm kapitol je zaměřeno na teoretickou část a zbylých šest kapitol se věnuje praktickým výstupům práce.

V první kapitole jsou nejprve definovány základní matematické operace a pojmy, které s počítačovou grafikou souvisí. Smyslem této části je poskytnout teoretický podklad pro pochopení následujících kapitol. Další kapitola se zaměřuje na softwarové nástroje, které budou využívány v praktické části práce. Je zde kladen důraz na orientaci v uživatelském prostředí a na oblasti využití těchto nástrojů. Další části pojednávají o vybraných oblastech počítačové grafiky, jako jsou grafické objekty, transformace, rasterizace a vektorizace, typy barevných formátů a typy grafických formátů.

Praktická část práce se věnuje tvorbě podpůrných SW výstupů s využitím zmíněných SW nástrojů. Tyto výstupy se opírají o sepsanou teorii v předešlé části práce a mají podobu

interaktivních scriptů Processing.py, pluginů pro SW nástroje nebo jiných vizualizací a příkladů. Kompletní výstupy jsou uložené v příloze a roztříděné dle kapitol.

I. TEORETICKÁ ČÁST

1 MATEMATICKÉ OPERACE A POJMY

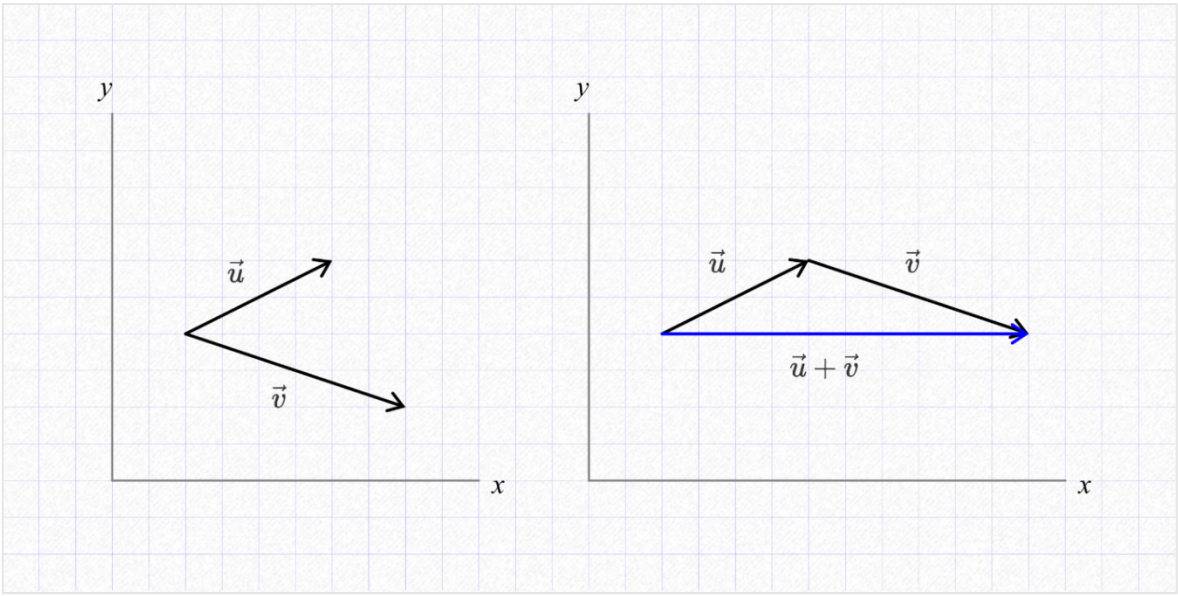
V oblasti počítačové grafiky a designu hrají matematické operace klíčovou roli. Znalost základních pojmů a principů matematiky může pomoci lépe pochopit a efektivněji vyřešit daný problém. Mezi takové oblasti matematiky patří zejména práce s vektory a maticemi, které se využívají nejen k manipulaci grafických objektů, ale také k tvorbě komplexních algoritmů pro zpracování obrazového obsahu.

1.1 Vektory

Vektor je typ úsečky, která kromě své velikosti má daný směr. Lze ho tedy označit jako směrovou úsečku umístěnou v Euklidovském prostoru, definovanou bodem A, který představuje začátek vektoru a bodem B, který vektor zakončuje. Matematicky lze vektor zapsat jedním písmenem $\vec{n}$ nebo dvojicí bodů $\overrightarrow{AB}$, kde pořadí písmen určuje směr vektoru. [1]

1.1.1 Sčítaní a odčítání

Při sčítání dvou a více vektorů je brán postupně každý jednotlivý prvek vektoru a sčítán s prvky ostatních vektorů stejného pořadí. Při daných vektorech $\vec{v} = (a_1, b_1)$ a $\vec{v} = (a_2, b_2)$ bude součet těchto vektorů vypadat takto: $\vec{u} + \vec{v} = (a_1 + a_2, b_1 + b_2)$. Graficky si operaci lze představit jako poskládání vektorů v návaznosti za sebe. První a poslední bod vzniklé křivky poté utvoří výsledný vektor. [2]

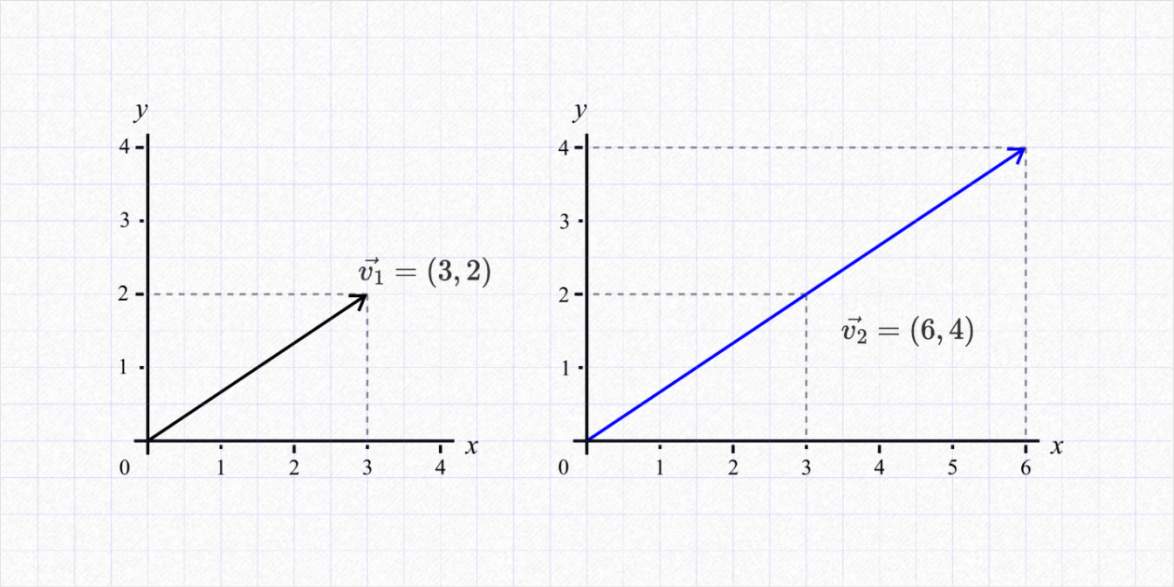


Obrázek 1: Ukázka grafického součtu vektorů

Odčítání vektorů představuje naprosto stejný proces, jako je sčítání vektorů. Dochází k němu tehdy, když je jeden vektor opačný. To znamená, že všechny jeho hodnoty jsou vynásobené -1.

1.1.2 Dělení a násobení

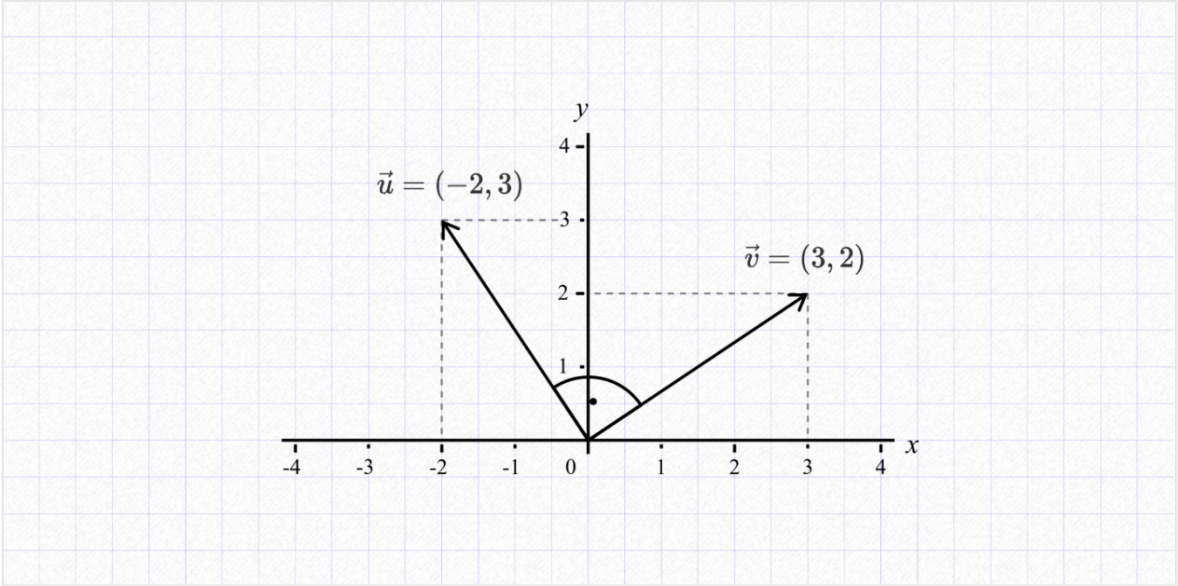
Násobením vektoru číslem měníme velikost vektoru a zároveň zachováváme jeho původní směr. Aby vznikl výsledný vektor, musí být všechny prvky vektoru $\vec{v} = (a, b, \dots, n)$ vynásobené konkrétním číslem k. Ve chvíli, kdy činitel nabude hodnoty $k < 1, k > 0$, dochází k dělení vektoru, tedy ke snižování jeho velikosti při zachování jeho směru. [3]



Obrázek 2: Vynásobení vektoru číslem

1.1.3 Skalární součin

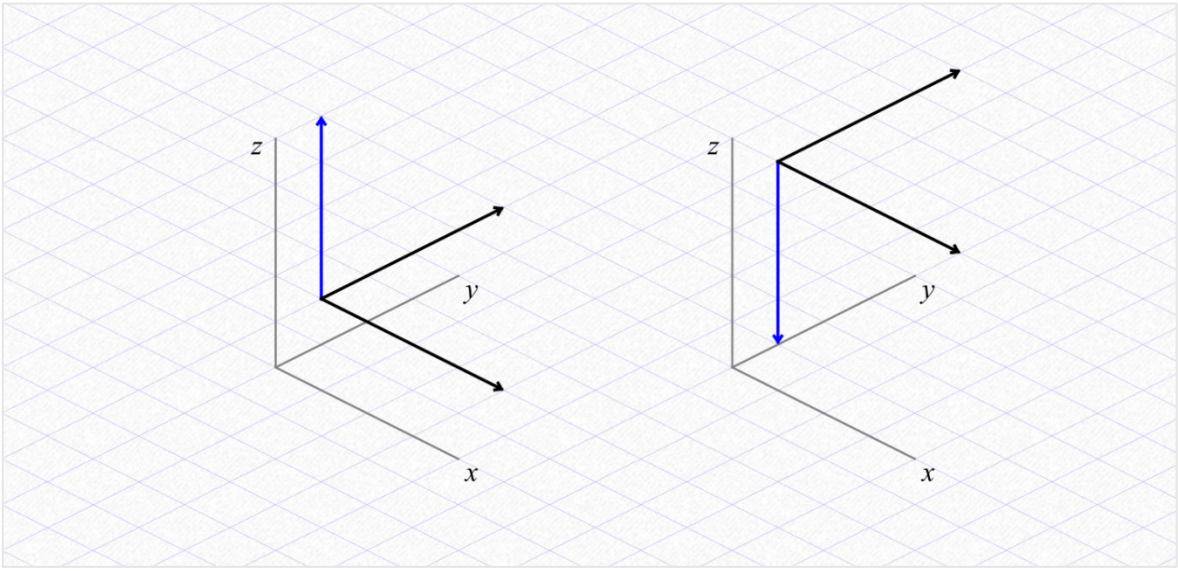
Při provádění skalárního součinu jsou vektory násobeny mezi sebou tak, aby se výslednou hodnotou stalo jedno číslo. Skalární součin dvou vektorů je dán vzorcem $\vec{u} \cdot \vec{v} = |\vec{u}| \cdot |\vec{v}| \cdot cos(\varphi)$, tedy součinem jejich velikostí vynásobených cosinem úhlu, který mezi sebou svírají. Velikost vektoru $\vec{u} = (a, b)$ lze získat skalárním vynásobením totožným vektorem pod odmocninou, tedy $\overrightarrow{|u|} = \sqrt{\vec{u} \cdot \vec{u}} = \sqrt{a^2 + b^2}$. Pokud vezmeme v potaz hodnotu $cos(90°) = 0$, poté ze vzorce vyplývá, že dva vektory jsou na sebe kolmé tehdy, kdy je jejich skalární součin roven nule. [4]



Obrázek 3: Vizualizace skalárního součinu

1.1.4 Vektorový součin

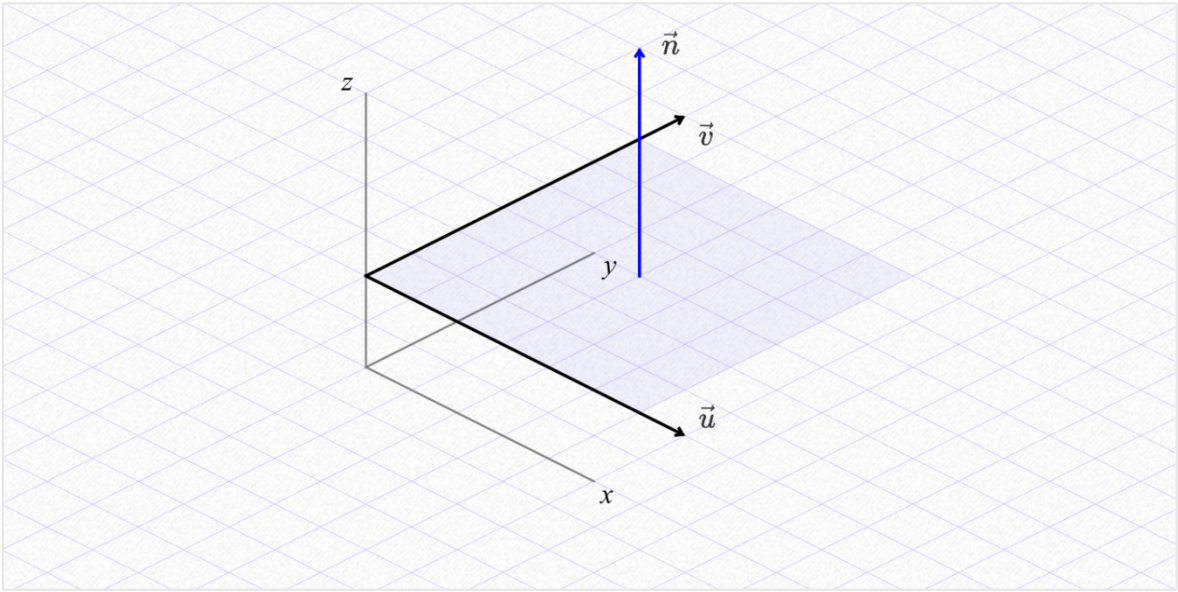
Vektorový součin se od skalárního součinu liší znaménkem × a zapisujeme ho jako $\vec{u} \times \vec{v} = \vec{w}$. Výsledkem vektorového součinu je vektor $\vec{w}$, který je zároveň kolmý na oba vektory $\vec{u}$ a $\vec{v}$. Ve dvourozměrném prostoru takovou podmínku splnit nelze, a proto je vektorový součin platný pouze pro vektory definované v trojrozměrném prostoru. Při násobení vektorovým součinem záleží na pořadí jednotlivých činitelů. Jejich pořadí určuje, zda bude směr vektoru $\vec{w}$ kladný, nebo záporný. [5]



Obrázek 4: Vektorový součin dvou vektorů vytváří kolmý vektor v závislosti na pořadí součinu

1.1.5 Normálový vektor – normála

Normálový vektor je vektor, který je kolmý k povrchu nebo objektu v daném bodě. Používá se k vyjádření orientace povrchu objektu a pro výpočty osvětlení a určení interakce světla s povrchem objektu. Normálový vektor lze jednoduše získat prohozením souřadnic původního vektoru a změnou jednoho znaménka. Z vektoru $\vec{v} = (3,2)$ získáme normálový vektor $\vec{u} = (-2,3)$. [6]



Obrázek 5: Vizualizace vektorového součinu dvou vektorů v rovině

1.2 Matice

Matice je z pohledu matematiky vícerozměrné pole čísel, symbolů nebo výrazů uspořádaných do řádků a sloupců ve čtvercovém nebo obdélníkovém tvaru. Velikost matice je obecně definována počtem řádků m a sloupců n a lze ji zapsat jako matici o rozměrech $m \times n$.

1.2.1 Sčítání a odčítání

Aby bylo možné sečíst dvě matice, musí mít obě matice stejné rozměry (tj. stejný počet řádků a sloupců). Každý prvek výsledné matice lze získat sečtením odpovídajících prvků původních matic. Tato operace je komutativní, tedy nezáleží na pořadí, ve kterém matice sčítáme. Pomocí součtu matic lze spojit několik lineárních transformací do jediné matice, což může usnadnit a zpřehlednit danou situaci.

$$\boldsymbol{A + B = C}$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} + \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} = \begin{bmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} & a_{1,3} + b_{1,3} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} & a_{2,3} + b_{2,3} \\ a_{3,1} + b_{3,1} & a_{3,2} + b_{3,2} & a_{3,3} + b_{3,3} \end{bmatrix}$$

1.2.2 Násobení

Aby bylo možné provést násobení matic, musí být počet sloupců první matice stejný jako počet řádků druhé matice. Výsledná matice má stejný počet řádků jako první matice a stejný počet sloupců jako druhá matice. Každý prvek ve výsledné matici se vypočítá jako bodový součin příslušného řádku první matice a příslušného sloupce druhé matice. Násobení matic není komutativní, což znamená, že pořadí matic ovlivňuje výslednou podobu matice. [7]

$$\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{C}$$

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \end{bmatrix} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix}$$

Prvky matice $\boldsymbol{C}$ se poté vypočítají následovně:

$$c_{1,1} = a_{1,1} \cdot b_{1,1} + a_{1,2} \cdot b_{2,1} + a_{1,3} \cdot b_{3,1}$$
$$c_{2,1} = a_{2,1} \cdot b_{1,1} + a_{2,2} \cdot b_{2,1} + a_{2,3} \cdot b_{3,1}$$
$$c_{1,2} = a_{1,1} \cdot b_{1,2} + a_{1,2} \cdot b_{2,2} + a_{1,3} \cdot b_{3,2}$$
$$c_{2,2} = a_{2,1} \cdot b_{1,2} + a_{2,2} \cdot b_{2,2} + a_{2,3} \cdot b_{3,2}$$

1.2.3 Transponace matice

Transponace matice je operace, při které se řádky původní matice převádějí na sloupce a sloupce na řádky. Výsledkem je nová matice, která se nazývá transponovaná matice. Matematicky lze transponaci zapsat jako $\boldsymbol{A}^T$, kde $\boldsymbol{A}$ je původní matice a $\boldsymbol{A}^T$ je její transponovaná matice. Pokud je $\boldsymbol{A}$ matice o rozměrech $m \times n$, (m řádků a n sloupců), pak její transponovaná matice $\boldsymbol{A}^T$ bude mít rozměry $n \times m$ (n řádků a m sloupců). Pro prvky transponované matice platí:

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{bmatrix} \rightarrow \boldsymbol{A}^T = \begin{bmatrix} a_{1,1} & a_{2,1} & a_{3,1} \\ a_{1,2} & a_{2,2} & a_{3,2} \end{bmatrix}$$

[20]

1.2.4 Inverze matice

Pokud má matice stejný počet sloupců a řádků, lze jí označit jako regulární a pro takovou matici existuje inverzní matice. Pokud se počet sloupců a řádků matice liší, nelze pro ni inverzi vytvořit. Taková matice se nazývá singulární.

Inverzní matice se v oblasti transformací používá ke zjištění zpětné (inverzní) transformace. Pokud je dána transformační matice $\boldsymbol{A}$ a chceme zjistit, jaká transformace by vrátila transformaci zpět k původnímu stavu, lze použít inverzní matici $\boldsymbol{A}^{-1}$. [9]

V praxi to znamená, že pokud je dána matice $\boldsymbol{X}$, která byla transformována pomocí matice $\boldsymbol{A}$ do podoby $\boldsymbol{Y}$, lze použít inverzní matici $\boldsymbol{A}^{-1}$ k získání původního stavu $\boldsymbol{X}$. Matematické vyjádření vypadá následovně:

$$\boldsymbol{Y} = \boldsymbol{A} \cdot \boldsymbol{X}$$

$$\boldsymbol{X} = \boldsymbol{A}^{-1} \cdot \boldsymbol{Y}$$

$$\begin{bmatrix} a_y \\ b_y \\ c_y \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 2 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_x \\ b_x \\ c_x \end{bmatrix}$$

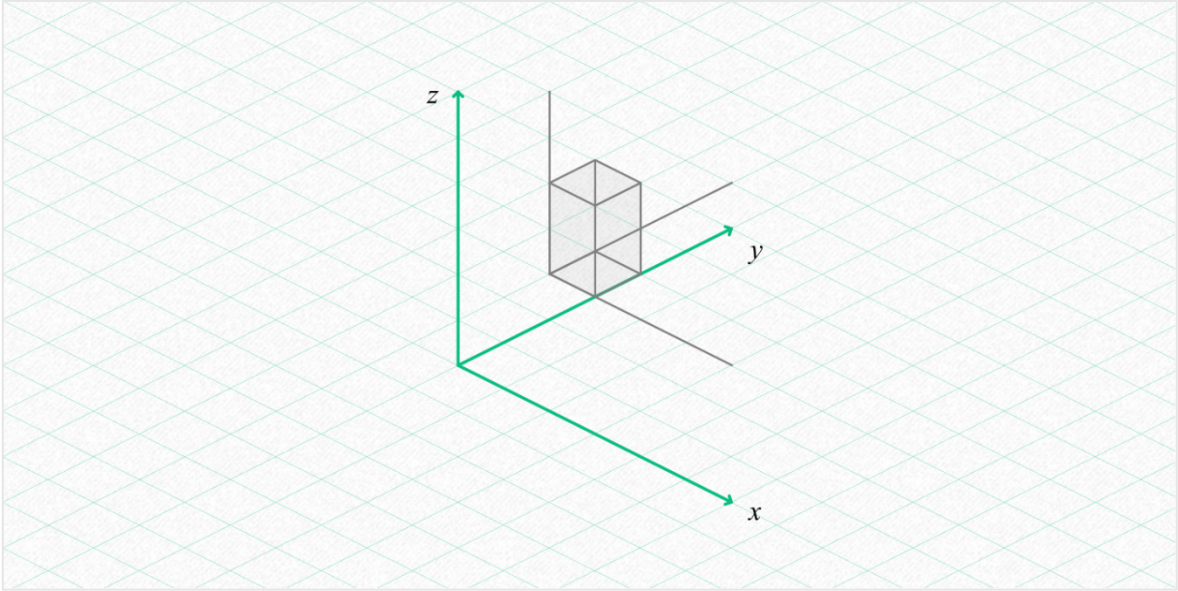
$$\begin{bmatrix} a_x \\ b_x \\ c_x \end{bmatrix} = \begin{bmatrix} 3 & -1 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} a_y \\ b_y \\ c_y \end{bmatrix}$$

1.3 Souřadnicové systémy

Souřadnicový systém je soubor pravidel pro určení polohy a orientace objektů ve virtuálním prostoru. Tyto souřadnice lze reprezentovat ve formě uspořádané trojice (x, y, z) pro 3D prostor a dvojice (x, y) pro 2D prostor.

1.3.1 Globální souřadnicový systém

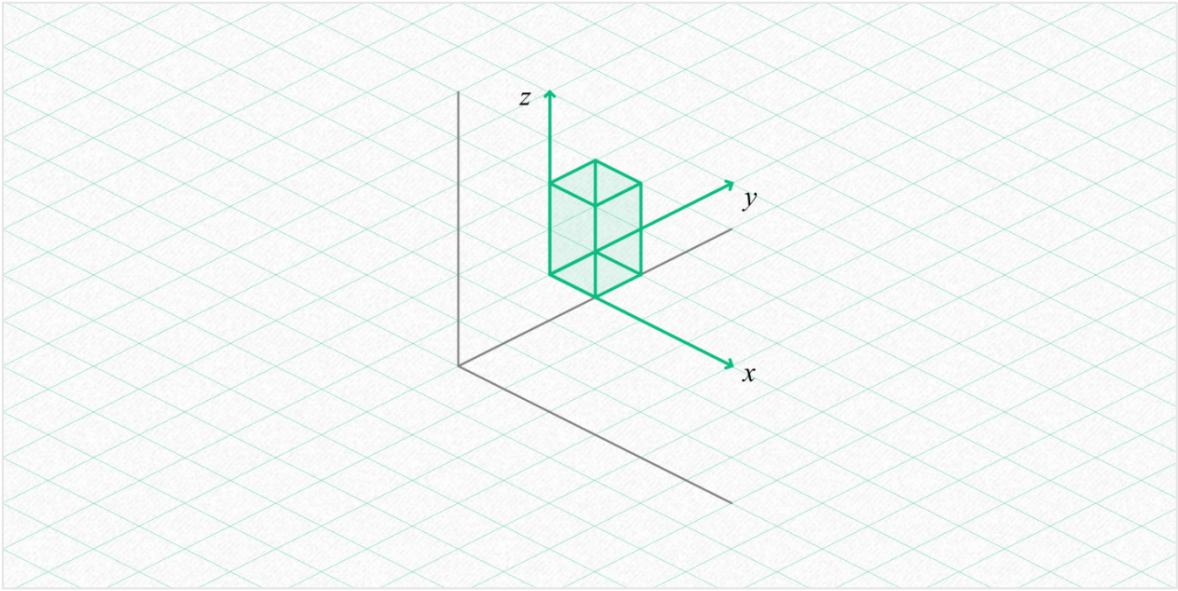
V globálním souřadnicovém systému, známém také jako světový souřadnicový systém, se k určení polohy bodů v prostoru používá pevná sada souřadnic. Tyto souřadnice zůstávají stejné bez ohledu na jakoukoli transformaci aplikovanou na objekt. Počátek globálního souřadnicového systému je často zvolen jako umístění pevného bodu ve scéně, například středu světa. Osy systému jsou často zvoleny tak, aby odpovídaly některým známým nebo vhodným směrům ve scéně, například osa x směřuje doprava, osa y směřuje nahoru a osa z směřuje ven z obrazovky.



Obrázek 6: Globální souřadnicový systém

1.3.2 Lokální souřadnicový systém

Lokální souřadnicový systém, známý také jako objektový souřadnicový systém, je souřadnicový systém, který je pevně spojen s jednotlivým objektem. Každý objekt má svůj vlastní lokální souřadnicový systém, který je pro daný objekt jedinečný. Používá se k popisu polohy a orientace objektu vzhledem k sobě samému.



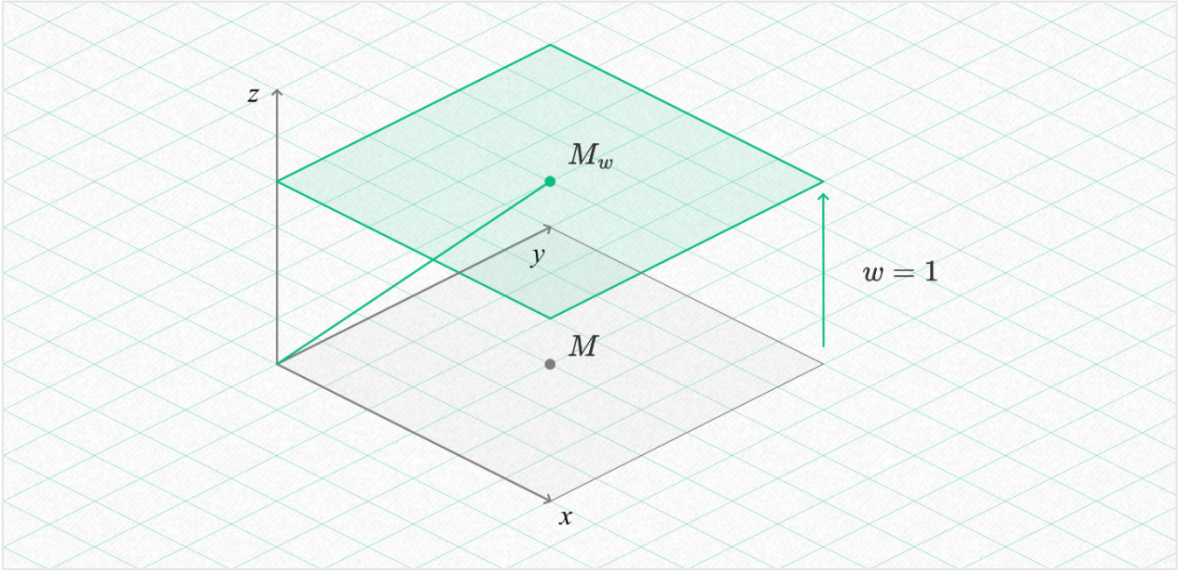
Obrázek 7: Lokální souřadnicový systém

1.3.3 Homogenní souřadnice

Homogenní souřadnice jsou rozšířením běžných kartézských souřadnic (souřadnicový systém se dvěma nebo více osami) a slouží k reprezentaci bodů v projektivní geometrii. Jsou užitečné zejména pro zjednodušení matematických výpočtů a při práci s maticemi v oblastech, jako je počítačová grafika.

V homogenních souřadnicích je bod v prostoru reprezentován vektorem s počtem prvků o jeden vyšší, než je počet dimenzí. Například bod v 2D kartézském systému má souřadnice (x, y), zatímco v homogenních souřadnicích má souřadnice (x, y, w), kde w je váhový faktor. Analogicky, bod v 3D kartézském sytému má souřadnice (x, y, z) a v homogenních souřadnicích je definován jako (x, y, z, w).

Převod z homogenních souřadnic zpět na kartézské souřadnice se provádí dělením prvních $n - 1$ prvků vektoru (pro n-dimenzionální prostor) váhovým faktorem w. Například bod ve 2D homogenních souřadnicích (x, y, w) se převede na kartézské souřadnice $(x/w, y/w)$. Homogenní souřadnice umožňují vyjádřit transformace, jako jsou translace, rotace a škálování, pomocí jediné matice, což zjednodušuje matematické výpočty a implementaci algoritmů. [10]

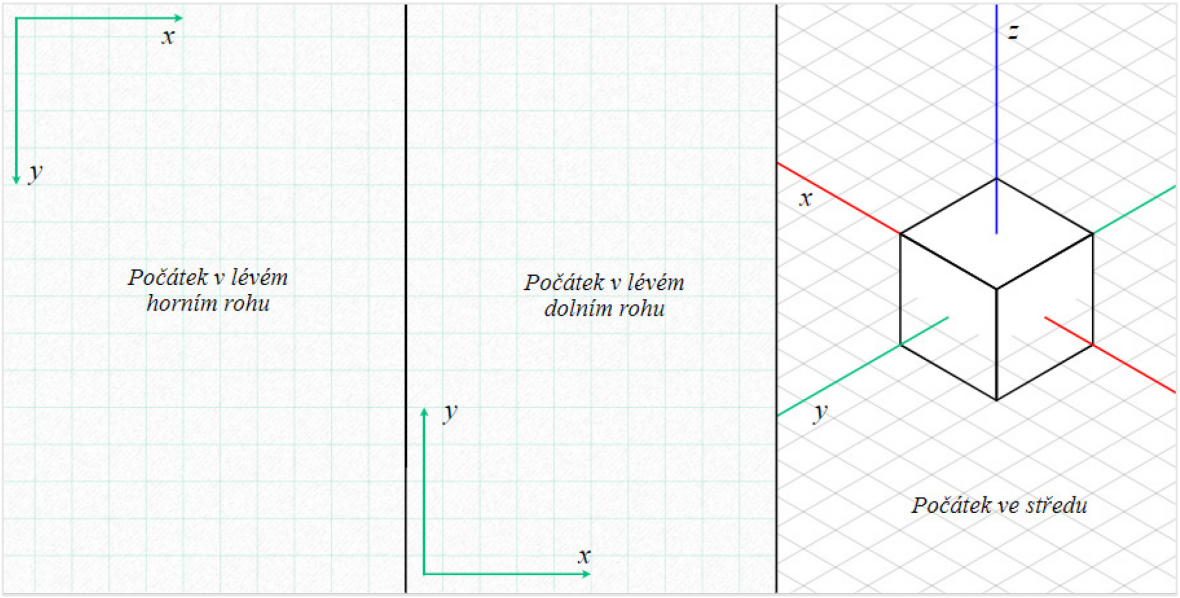


Obrázek 8: Homogenní souřadnice

1.1.1 Souřadnicové systémy v SW nástrojích

Souřadnicové systémy jsou základním aspektem grafického softwaru, přičemž každý nástroj používá jedinečný systém přizpůsobený svým primárním funkcím.

Processing.py i GIMP používají kartézské souřadnicové systémy s počátkem v levém horním rohu, které jsou vhodné pro 2D grafiku a úpravu obrázků. Naproti tomu Inkscape používá kartézský souřadnicový systém s počátkem v levém dolním rohu, který odpovídá tradičním matematickým konvencím a je vhodný pro úpravy vektorové grafiky. Blender, software pro 3D modelování a animaci, využívá trojrozměrný souřadnicový systém s počátkem ve středu 3D prostoru, který umožňuje komplexní úlohy modelování, animace a vykreslování. Pochopení těchto různých souřadnicových systémů je klíčové pro efektivní navigaci a tvorbu v těchto aplikacích.

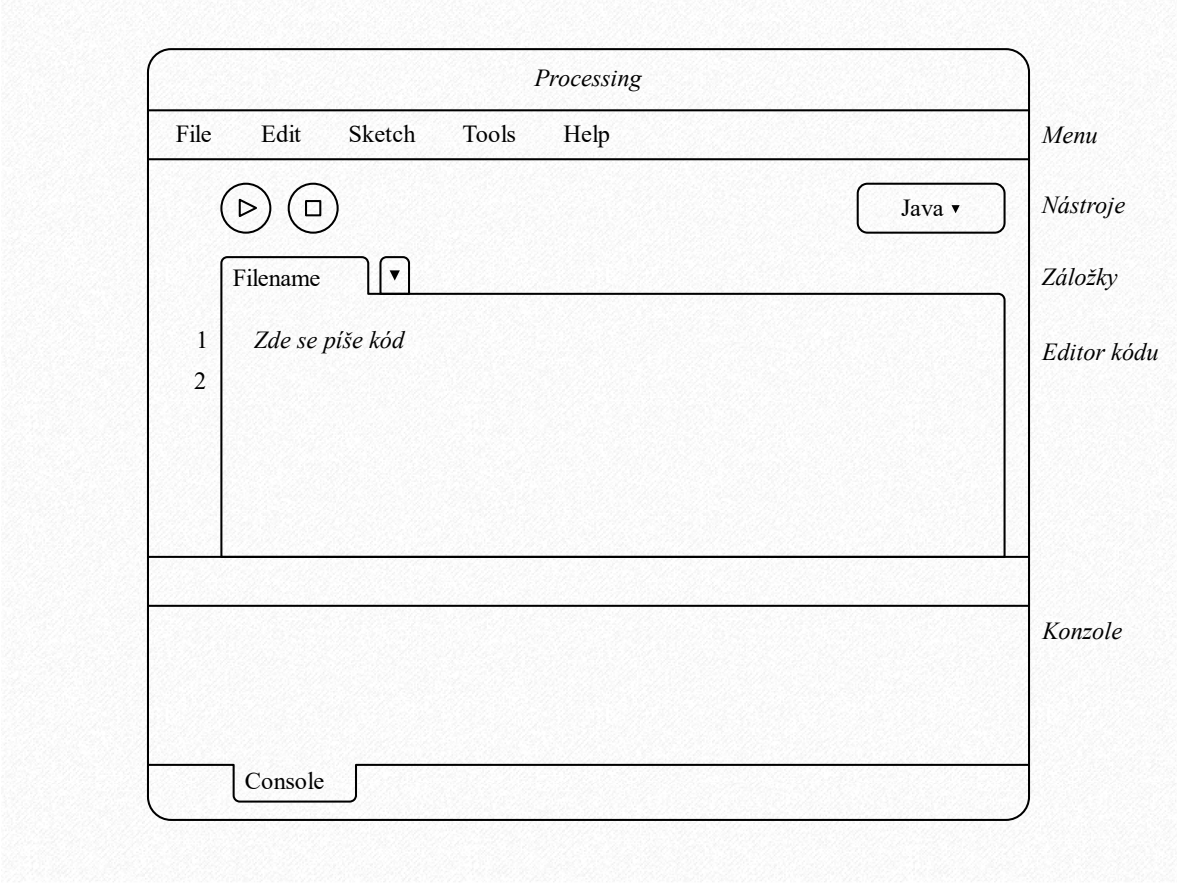


Obrázek 9: Ukázky tří typů souřadnicového systému

2 NÁSTROJE

2.1 Processing.py

Processing.py je software využívající jazyk Python k vykreslování grafiky a animací. Tato konkrétní verze s příponou *.py* vychází z původního Processing, který vznikl v roce 2001 a je postaven na programovacím jazyce Java. Processing je primárně určený pro umělce a designéry, kteří s jeho pomocí vytvářejí vše od jednoduchých náčrtů až po složitá interaktivní umělecká díla. Processing.py poskytuje v rámci jednoduchého editoru výkonnou sadu nástrojů pro vytváření a manipulaci s grafickými tvary, a proto je obzvláště užitečný pro demonstraci jak grafických objektů, tak i jiných obrazových úprav. Zároveň lze vzniklý Python kód využít jako podklad pro následnou tvorbu praktických skriptů a pluginů.



Obrázek 10: Schéma aplikace Processing

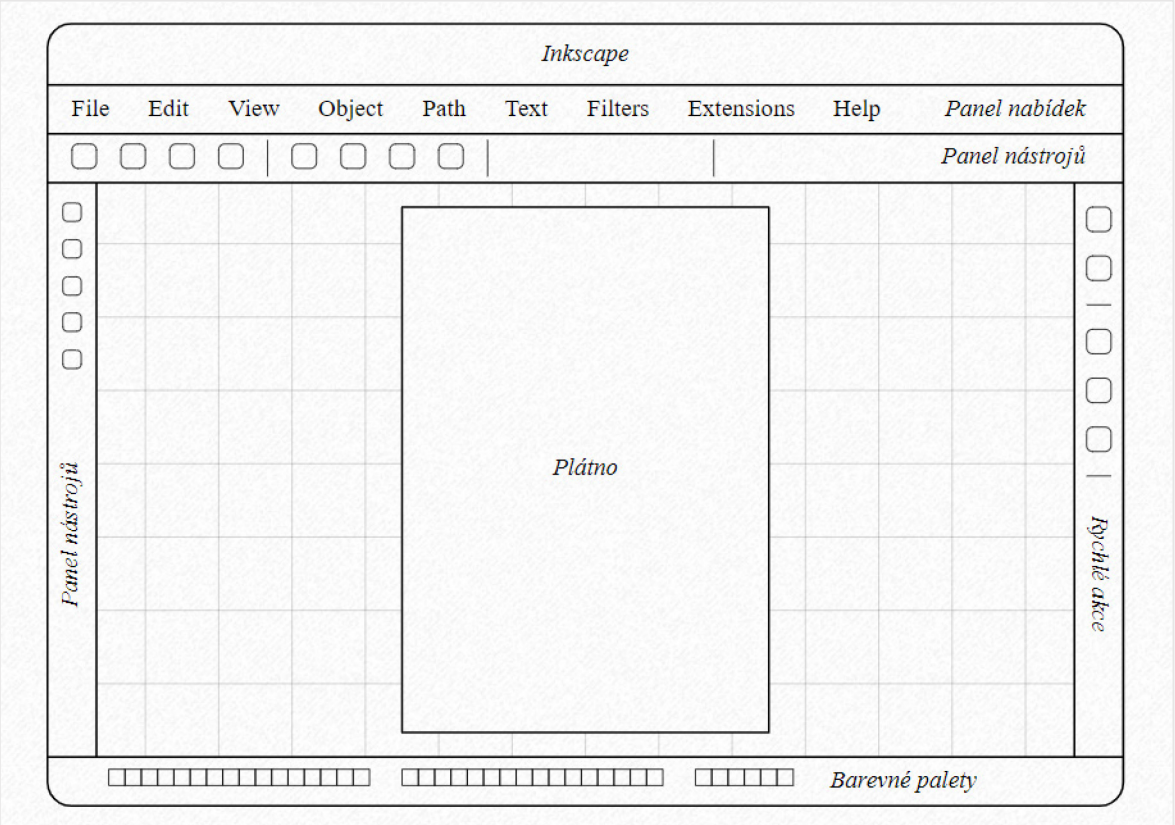
Uživatelské prostředí Processing klade důraz na co největší jednoduchost, a proto obsahuje pouze nejnutnější prvky ke zprovoznění kódu. Krom běžné horní lišty s menu se zde nachází panel nástrojů s dvěma tlačítky pro spuštění nebo pozastavení programu a třetím tlačítkem pro výběr módu. Defaultně je Processing nastavený na práci v Javě.

2.1.1 Souřadnicový systém Processing.py

Souřadnicový systém pro zobrazování na displeji začíná v levém horním rohu, nikoli v levém dolním rohu, jak je zvykem u většiny běžných souřadnicových systémů. V tomto případě se totiž souřadnicový systém řídí podle prvního pixelu displeje a pokračuje po řádcích směrem dolů. [11]

2.2 Inkscape

Inkscape je bezplatný vektorový open-source grafický editor, který uživatelům umožňuje vytvářet a upravovat škálovatelné vektorové grafické soubory (SVG). Nabízí širokou škálu nástrojů pro navrhování, kreslení a manipulaci s grafikou a je tak vhodnou alternativou ke komerčnímu softwaru, jako je například Adobe Illustrator. Inkscape je vhodný pro různé projekty, včetně návrhu loga, webové grafiky, ilustrací a tvorby ikon. Inkscape je k dispozici pro systémy Windows, MacOS a Linux.



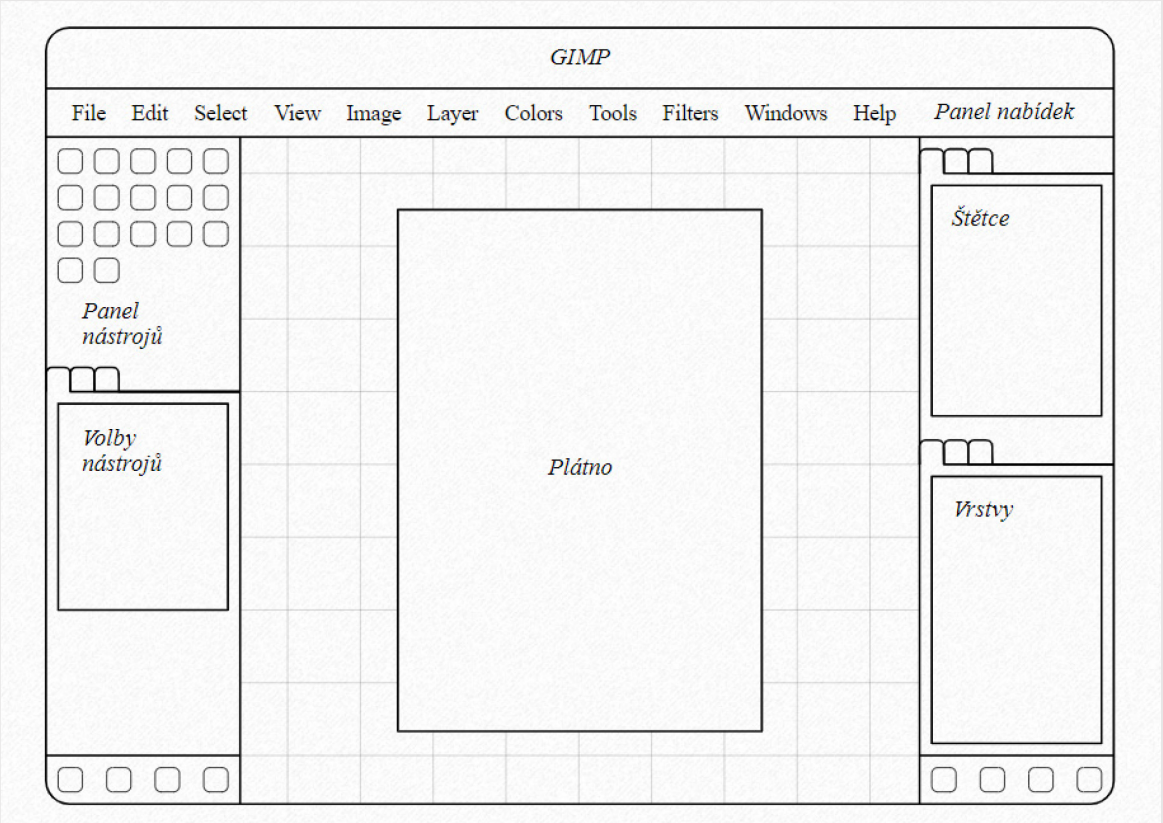
Obrázek 11: Schéma aplikace Inkscape

2.2.1 Souřadnicový systém v programu Inkscape

V aplikaci Inkscape je souřadnicový systém založen na standardu SVG (Scalable Vector Graphics), který definuje počátek (0,0) v levém horním rohu plátna. Při pohybu směrem doprava se souřadnice x zvětšuje a při pohybu směrem dolů se zvětšuje souřadnice y. Souřadnicový systém v aplikaci Inkscape tedy funguje stejně jako v Processing.

2.3 GIMP

GIMP (GNU Image Manipulation Program) je volně dostupný open-source editor rastrové grafiky, který je považován za dobrou alternativu k programu Adobe Photoshop. GIMP je k dispozici v operačních systémech Windows, MacOS a Linux a nabízí komplexní sadu nástrojů a funkcí pro úpravu, retušování a manipulaci s obrázky. GIMP podporuje širokou škálu formátů souborů a poskytuje různé funkce, jako jsou vrstvy, masky, filtry a přizpůsobitelné štětce. Je vhodný pro úkoly, jako jsou úpravy fotografií, grafický design a tvorba digitálního umění. Díky velké a aktivní komunitě těží GIMP z pravidelných aktualizací, výukových programů a pluginů, které rozšiřují jeho možnosti.



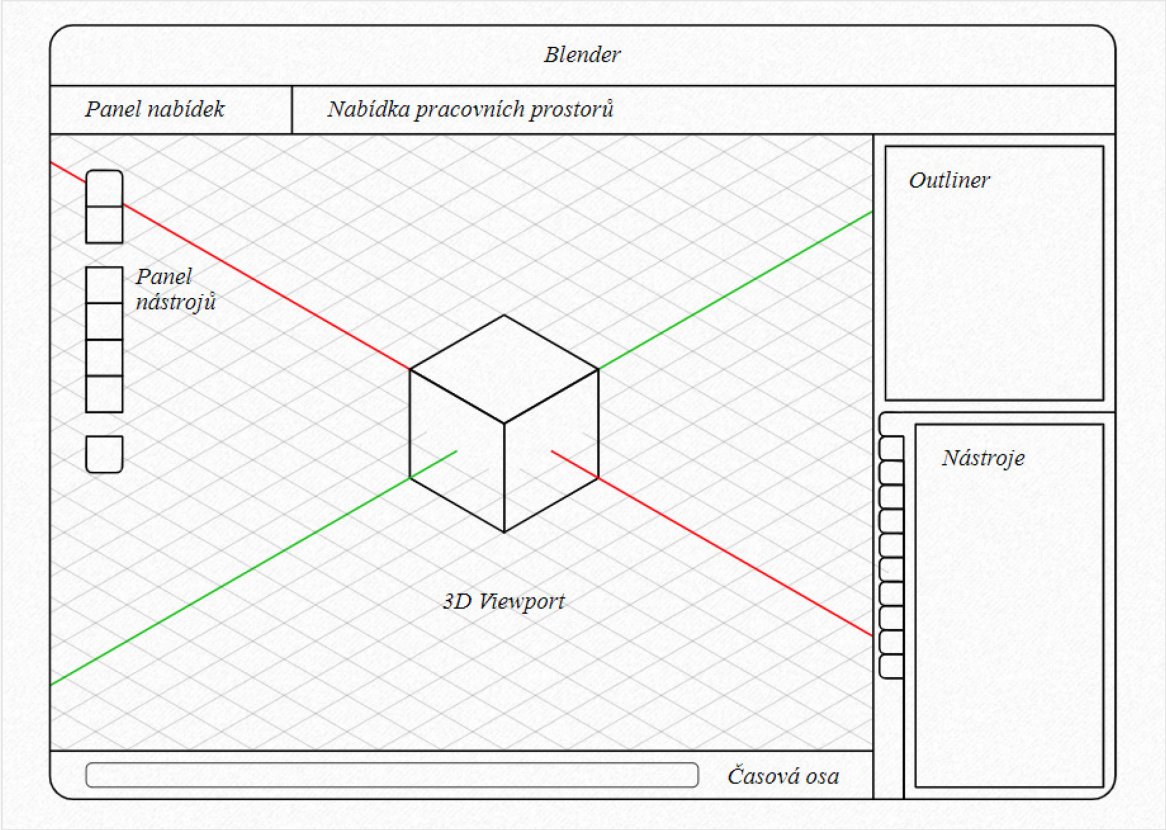
Obrázek 12: Schéma prostředí aplikace GIMP

2.3.1 Souřadnicový systém v programu GIMP

V programu GIMP má souřadnicový systém počátek (0,0) v levém horním rohu obrázku, přičemž vodorovná osa x se táhne směrem doprava a svislá osa y směrem dolů. Každý pixel v obrázku má jedinečnou souřadnici (x, y) založenou na jeho poloze vzhledem k počátku. GIMP zobrazuje aktuální souřadnice kurzoru v levém dolním rohu okna obrázku a umožňuje používat vodítka a pravítka pro přesné zarovnání. Souřadnicový systém pomáhá přesně umístit prvky při práci s nástroji, výběry nebo transformacemi. [13]

2.4 Blender

Blender je volně dostupný open-source nástroj pro tvorbu 3D grafiky, který je určen jak profesionálům, tak amatérům. Podporuje komplexní 3D pipeline, která zahrnuje modelování, rigging, animaci, simulaci, vykreslování, kompozici a sledování pohybu. Díky svým všestranným funkcím je Blender široce používán pro tvorbu 3D umění, vizuálních efektů, videoher a animovaných filmů. Tento software má silnou komunitu, která přispívá k jeho růstu, a nabízí rozsáhlé zdroje včetně výukových programů a doplňků, díky čemuž je oblíbenou volbou 3D umělců po celém světě.



Obrázek 13: Schéma prostředí aplikace Blender

2.4.1 Souřadnicový systém v programu Blender

Souřadnicový systém v Blenderu využívá 3D kartézský systém se třemi osami: x (červená), y (zelená) a z (modrá). Ve výchozím nastavení Blender používá pravotočivý souřadnicový systém s kladnou osou x směřující doprava, kladnou osou y směřující dopředu a kladnou osou z směřující nahoru. Každý objekt má lokální souřadnicový systém založený na svém počátku a pro specifické případy použití existují další souřadnicové systémy, jako jsou souřadnice pohledu a normální souřadnice. Uživatelé mohou přepínat mezi souřadnicovými systémy pro transformace pomocí rozbalovací nabídky orientace transformace v okně ***3D Viewport*** nebo klávesové zkratky ***Alt + Space***. [15]

3 GRAFICKÉ OBJEKTY

Grafický objekt lze chápat jako vizuální reprezentaci matematického konceptu nebo rovnice. Grafické objekty jsou užitečným nástrojem pro pochopení složitých matematických pojmů, protože poskytují vizuální a více intuitivní znázornění abstraktních myšlenek, které mohou být jinak obtížně uchopitelné. Může se jednat o přímku reprezentující rovnici, nebo graf znázorňující vztah mezi dvěma proměnnými.

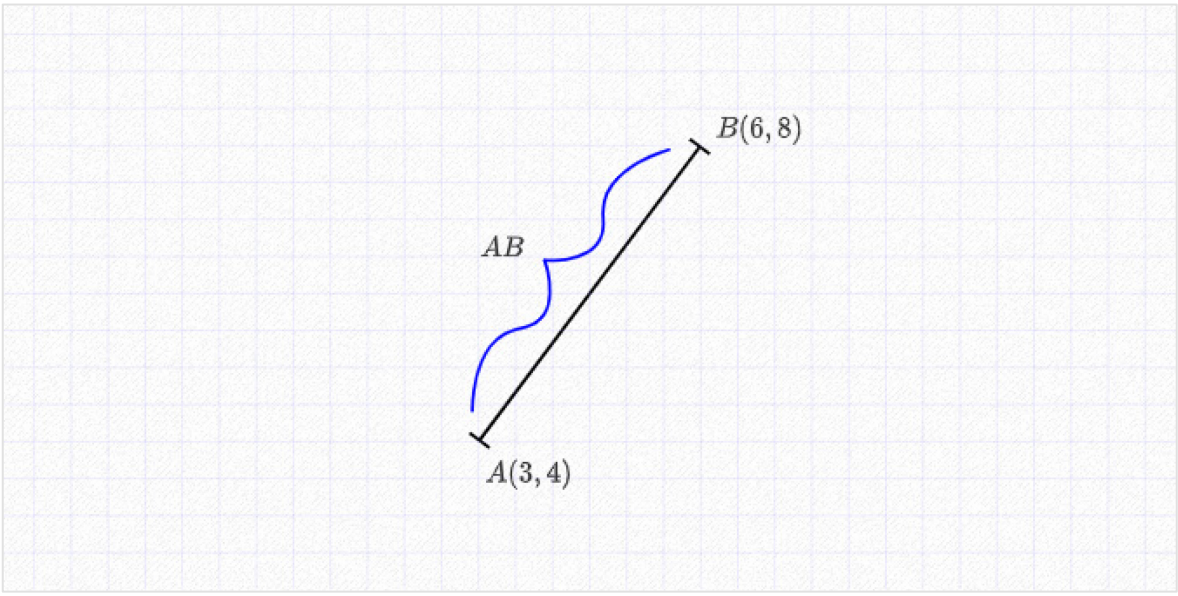
3.1 Úsečka

Úsečku si lze představit jako vyhrazenou část nekonečné přímky definovanou podle vzorce $y = kx + q$. Úsečka je následně definována dvěma body $[x_1, y_1]$ a $[x_2, y_2]$ ležícími na přímce, nebo jako jeden bod $[x_1, y_1]$ s vektorem určujícím rozdíl souřadnic $(\Delta x, \Delta y) = (x_1 y_1 - x_2 y_2)$. [10]

3.1.1 Početní operace s úsečkou

Při sčítání dvou a více vektorů je brán postupně každý jednotlivý prvek vektoru a sčítán s prvky ostatních vektorů stejného pořadí. Při daných vektorech $\vec{u} = (a_1, b_1)$ a $\vec{v} = (a_2, b_2)$ bude součet těchto vektorů vypadat takto: $\vec{u} + \vec{v} = (a_1 + a_2, b_1 + b_2)$. Graficky si operaci lze představit jako poskládání vektorů v návaznosti za sebe. První a poslední bod vzniklé křivky poté utvoří výsledný vektor.

Délka úsečky

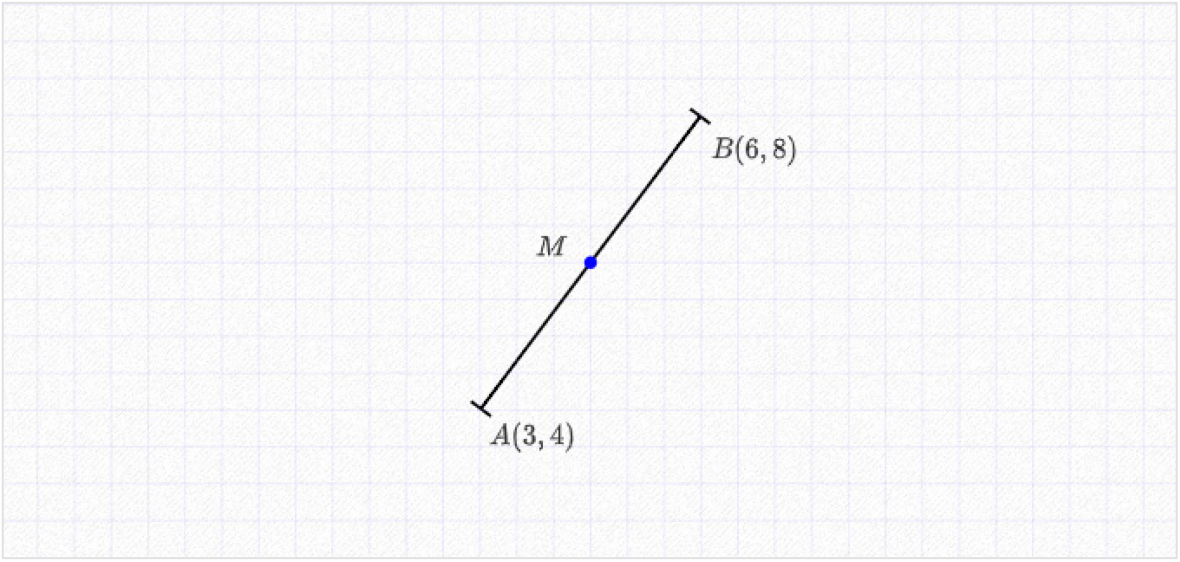


Obrázek 14: Ilustrace úsečky

Jsou-li dány koncové body úsečky $A(x_1, y_1)$ a $B(x_2, y_2)$, lze délku úsečky AB vypočítat pomocí vzorce délky:

$$AB = \sqrt{(x_1 - x_2)^2 + (y_1 + y_2)^2}$$

Střed úsečky

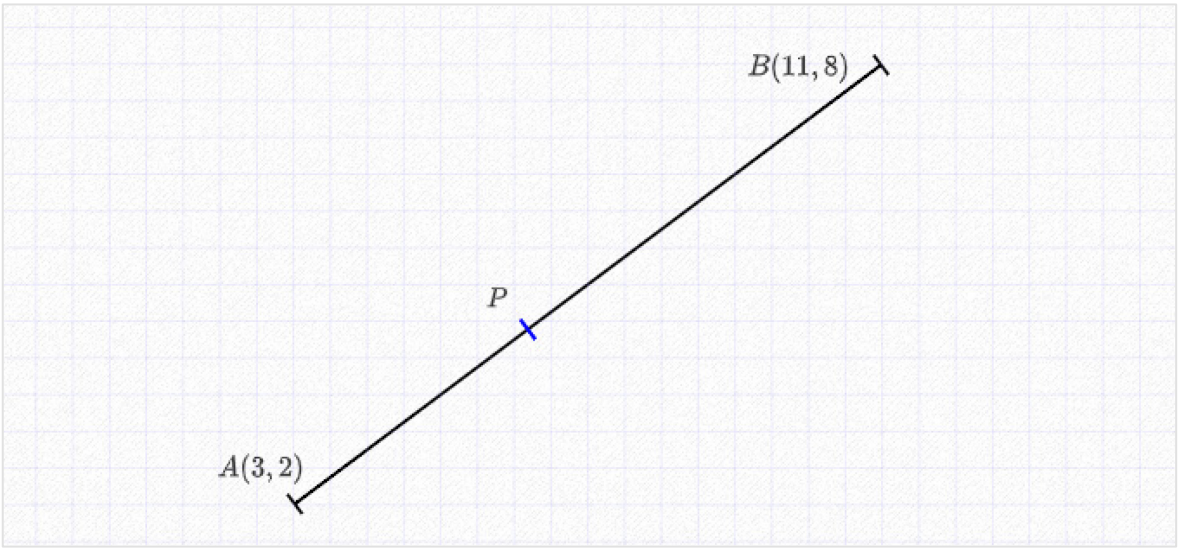


Obrázek 15: střed úsečky

Střed úsečky AB s koncovými body $A(x_1, y_1)$ a $B(x_2, y_2)$ má souřadnice [16]:

$$M\left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right)$$

Rozdělení úsečky v poměru

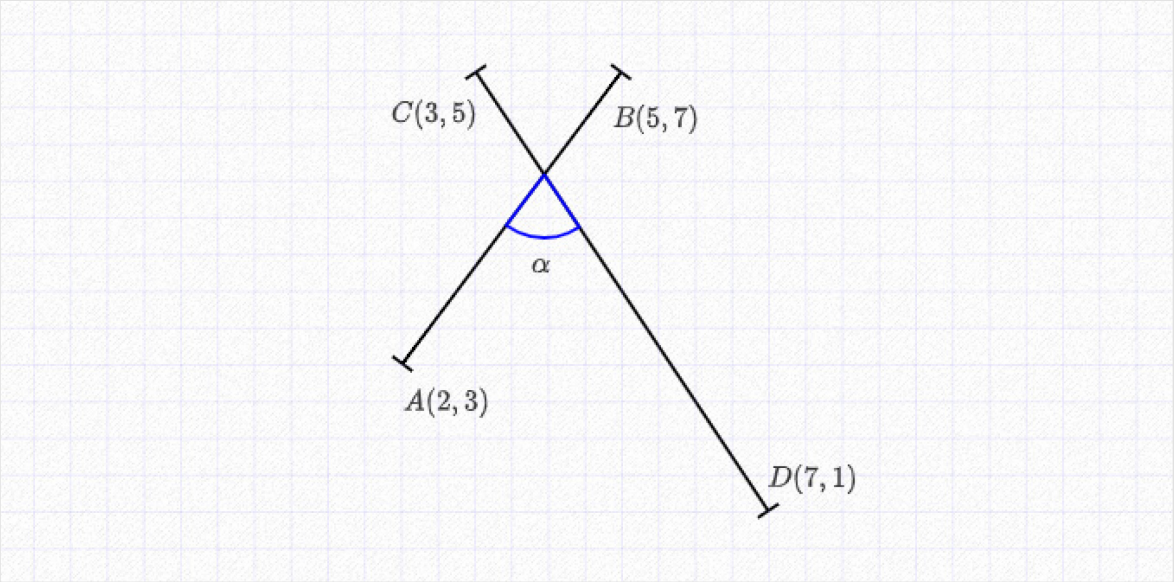


Obrázek 16: Úsečka rozdělená v daném poměru

Bod P rozděluje úsečku AB v poměru $m:n$, kde $A(x_1, y_1)$ a $B(x_2, y_2)$ jsou koncové body úsečky. Souřadnice bodu P jsou:

$$P\left(\frac{mx_2 + nx_1}{m+n}, \frac{my_2 + ny_1}{m+n}\right)$$

Zjištění pravého úhlu



Obrázek 17: Určení kolmosti úseček

Jsou-li dány úsečky AB a CD s koncovými body $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ a $D(x_4, y_4)$, lze určit, zda jsou úsečky kolmé, pokud jejich směrnice splňují podmínku:

$$m_1 \cdot m_2 = -1$$

kde m_1 a m_2 jsou směrnice přímek, na kterých leží úsečky AB a CD. [4]

$$\text{Směrnice } m_1 = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\text{Směrnice } m_2 = \frac{y_4 - y_3}{x_4 - x_3}$$

Rovnoběžnost úseček

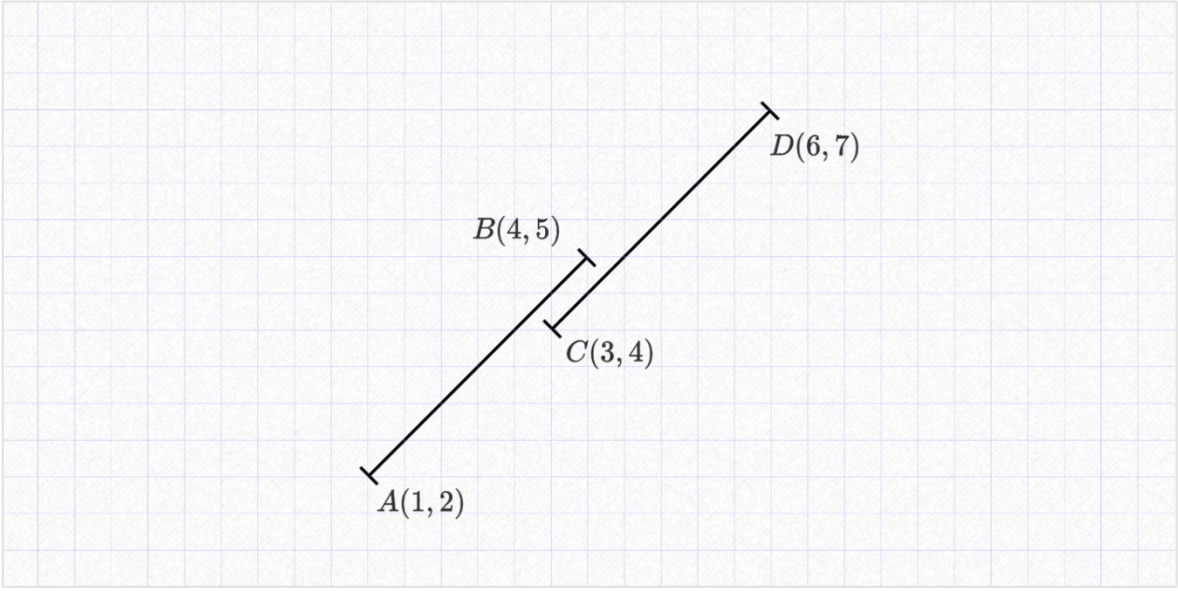
Jsou-li dány úsečky AB a CD s koncovými body $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ a $D(x_4, y_4)$, lze určit, zda jsou úsečky rovnoběžné, pokud mají stejné směrnice:

$$m_1 = m_2$$

kde m_1 a m_2 jsou směrnice přímek, na kterých leží úsečky AB a CD. [4]

$$\text{Směrnice } m_1 = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\text{Směrnice } m_2 = \frac{y_4 - y_3}{x_4 - x_3}$$



Obrázek 18: Rovnoběžné úsečky

3.1.2 Úsečka ve 3D prostoru

V rovině je vzorec přímky obvykle vyjádřen ve tvaru $y = kx + q$, kde k je směrnice a q je řez s osou y.

V 3D prostoru by se ale podle původní definice nejednalo o přímku v prostoru, ale o plochu. Proto se přímka obvykle definuje pomocí parametrických rovnic. Pro přímku, která prochází bodem (x_0, y_0, z_0) a má směrový vektor (a, b, c), jsou rovnice následující:

$$x = x_0 + at$$

$$y = y_0 + bt$$

$$z = z_0 + ct$$

kde t je parametr, který mění hodnoty x, y a z. Každá hodnota t odpovídá jednomu bodu na přímce.

Úsečka v 3D prostoru je podobně jako přímka často definována pomocí parametrických rovnic. Máme-li dva body, mezi kterými se úsečka nachází, $A(x1, y1, z1)$ a $B(x2, y2, z2)$, můžeme použít parametr t v intervalu [0,1] k definování úsečky AB následovně:

$$x = x_1 + t(x_2 - x_1)$$

$$y = y_1 + t(y_2 - y_1)$$

$$z = z_1 + t(z_2 - z_1)$$

Tento přístup spočívá v interpolaci mezi bodem A a bodem B. Pro $t = 0$ dostáváme bod A, pro $t = 1$ dostáváme bod B a pro $0 < t < 1$ dostáváme body na úsečce mezi A a B. [17]

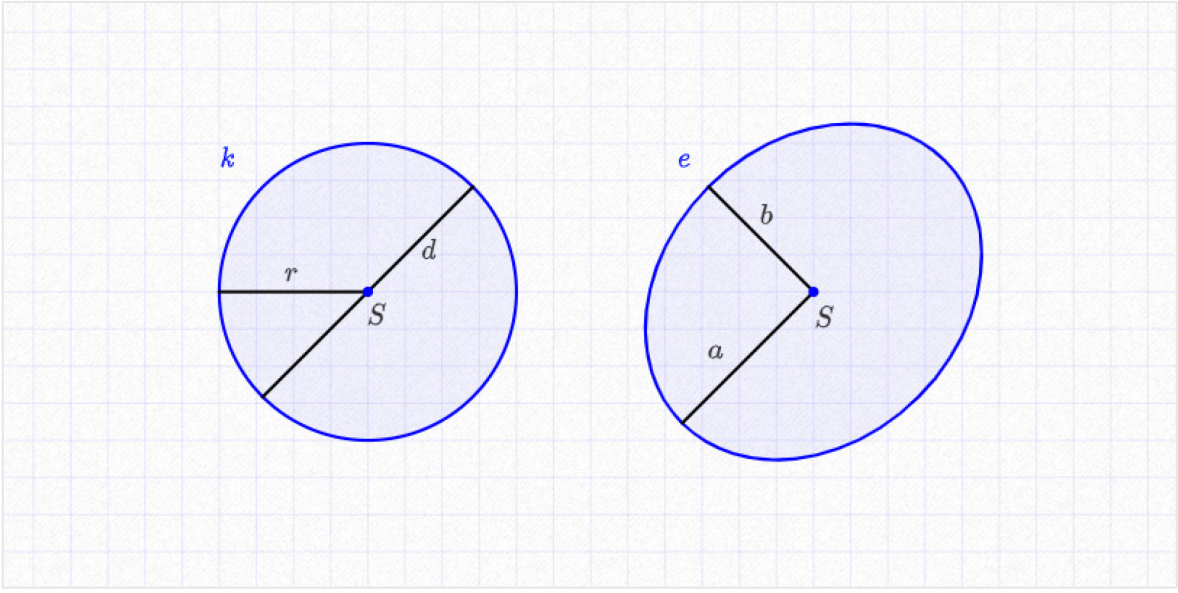
3.2 Kružnice a elipsa

Kružnice je dvourozměrný útvar, který je definován množinou bodů v rovině, které jsou stejně vzdálené od jediného bodu, nazývaného střed kružnice. Vzdálenost od středu k libovolnému bodu na kružnici se nazývá poloměr, který se označuje jako r. Rovnice kružnice se středem (a, b) a poloměrem r je dána rovnicí $(x - a)^2 + (y - b)^2 = r^2$. Elipsa je také dvourozměrný útvar definovaný množinou bodů v rovině. Na rozdíl od kružnice je vzdálenost mezi dvěma pevnými body, nazývanými ohniska, a libovolným bodem na elipse konstantní. Hlavní osa elipsy je nejdelší průměr elipsy, zatímco vedlejší osa je nejkratší průměr. Rovnice elipsy se středem (h, k), délkou hlavní osy a a délkou vedlejší osy b je dána rovnicí:

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$

3.2.1 Početní operace s kružnicí a elipsou

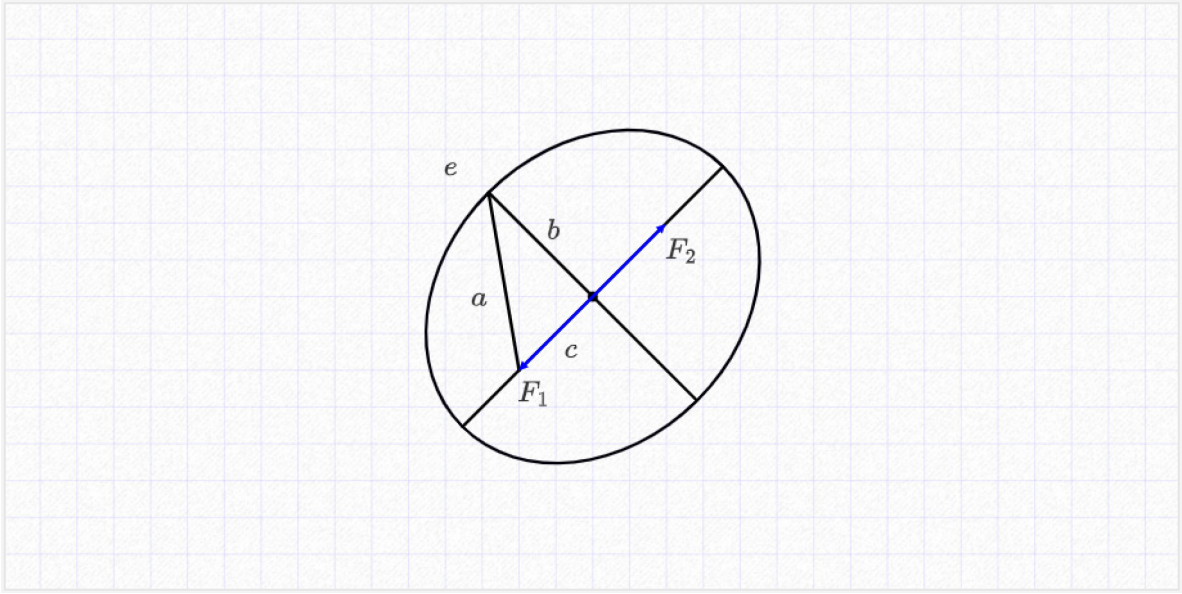
Výpočet obvodu a obsahu



Obrázek 19: Schéma kružnice a elipsy

U kružnice se obvod O vypočítá jako $O = 2\pi r$ a obsah A jako $A = \pi r^2$, kde r je poloměr. Obvod elipsy nelze vyjádřit jednoduchým vzorcem, ale lze jej aproximovat pomocí různých metod. Obsah A se vypočítá jako $A = \pi ab$, kde a je velká poloosa a b je malá poloosa.

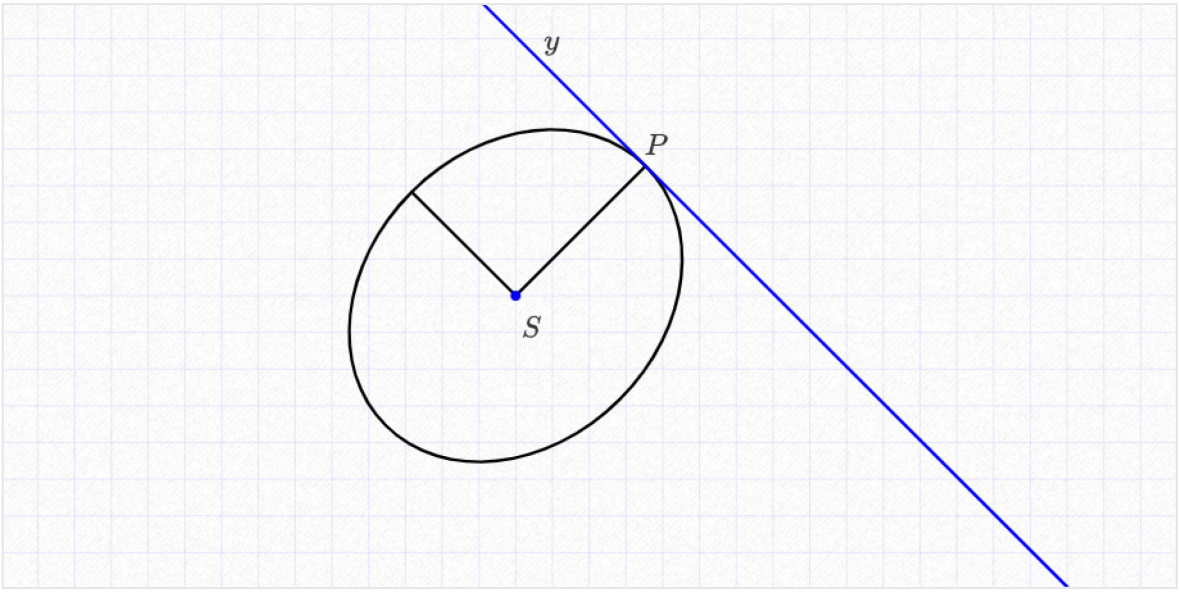
Výpočet vzdálenosti mezi středem a ohniskem elipsy



Obrázek 20: Ohniska elipsy

Vzdálenost mezi středem a ohniskem elipsy lze spočítat podle vzorce $c^2 = a^2 + b^2$, kde a je velká poloosa a b je malá poloosa. [18]

Tangenta a normála na kružnici na elipsu



Obrázek 21: Tečna na elipsu

Tangenta je přímka, která se dotýká kružnice nebo elipsy v jednom bodě a je kolmá na poloměr (v případě kružnice) nebo normálu (v případě elipsy) v dotykovém bodě. Normála je přímka, která je kolmá na tangentu v dotykovém bodě.

Obecný vzorec pro normálu kružnice v bodě $P(x_1, y_1)$ se středem $S(h, k)$ je:

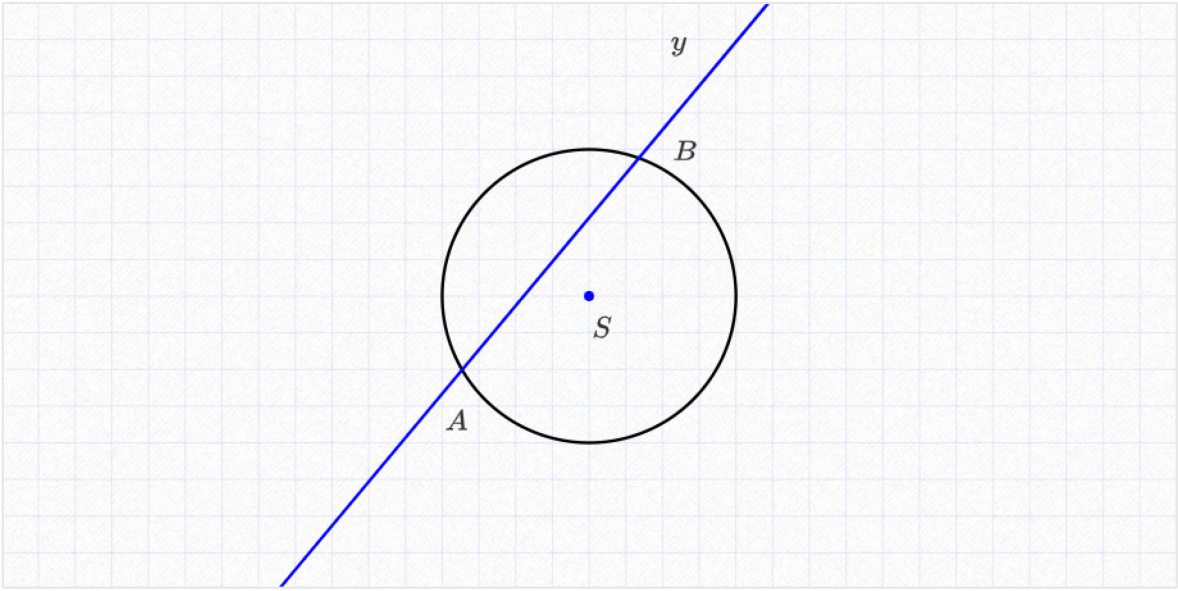
$$y - y_1 = \frac{(k - y_1)}{(h - x_1)} \cdot (x - x_1)$$

A obecný vzorec pro tangentu kružnice v bodě $P(x_1, y_1)$ je:

$$y - y_1 = -\frac{(h - x_1)}{(k - y_1)} \cdot (x - x_1)$$

Tyto vzorce platí pro bod P na kružnici a střed S.

Průnik kružnice s přímkou



Obrázek 22: Průnik kružnice s přímkou

Nalezení průnikových bodů kružnice se středem v počátku a poloměrem r a přímky $y = c$ spočívá ve soustavě rovnic $x^2 + y^2 = r^2$ a $y = c$. Průnikové body jsou poté výsledkem řešení kvadratické rovnice pro x pomocí determinantu.

3.3 Křivky

Křivky jsou základním nástrojem počítačové grafiky, který se používá k vytváření plynulých a přesných tvarů. Používají se k vytváření všech možných objektů, od jednoduchých

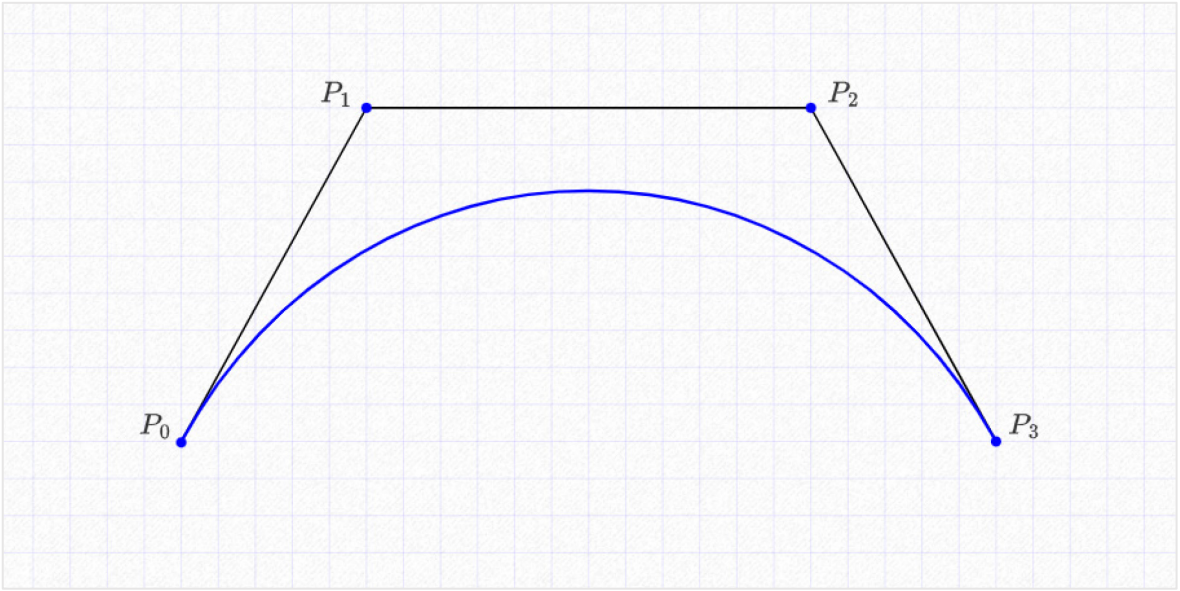
geometrických až po složité organické formy. Křivku lze popsat jako hladkou, spojitou, matematicky definovatelnou cestu, kterou lze reprezentovat sadou bodů.

V počítačové grafice se běžně používá několik typů křivek, např: Bézierovy křivky, B-spline křivky, křivky NURBS nebo Hermitovy křivky.

3.3.1 Bézierovy křivky

Bézierova křivka je definována sadou řídicích bodů, které určují tvar křivky. Samotná křivka těmito řídicími body nutně neprochází, ale její tvar vychází z polohy těchto bodů. Křivka je definována polynomickou rovnicí.

Stupeň Bézierovy křivky je určen počtem řídicích bodů použitých k její definici. Například lineární Bézierova křivka je definována dvěma řídicími body, zatímco kvadratická Bézierova křivka je definována třemi řídicími body. Kubická Bézierova křivka je definována čtyřmi řídicími body, což představuje ideální poměr mezi schopností tvořit dostatečně plynulé křivky a snadnou ovladatelností. Proto jsou kubické křivky zdaleka nejčastěji používaný stupeň Bézierovy křivky v počítačové grafice.



Obrázek 23: Kubická Bézierova křivka

Rovnice polynomu Bézierovy křivky stupně n, která je definována $n + 1$ řídicími body $P0, P1, P2, \ldots Pn$, je dána vztahem:

$$B(t) = (1-t)^n \cdot P_0 + n \cdot (1-t)^{(n-1)} \cdot t \cdot P_1 + \cdots + n \cdot (1-t) \cdot P_{n-1} + t^n \cdot P_n$$

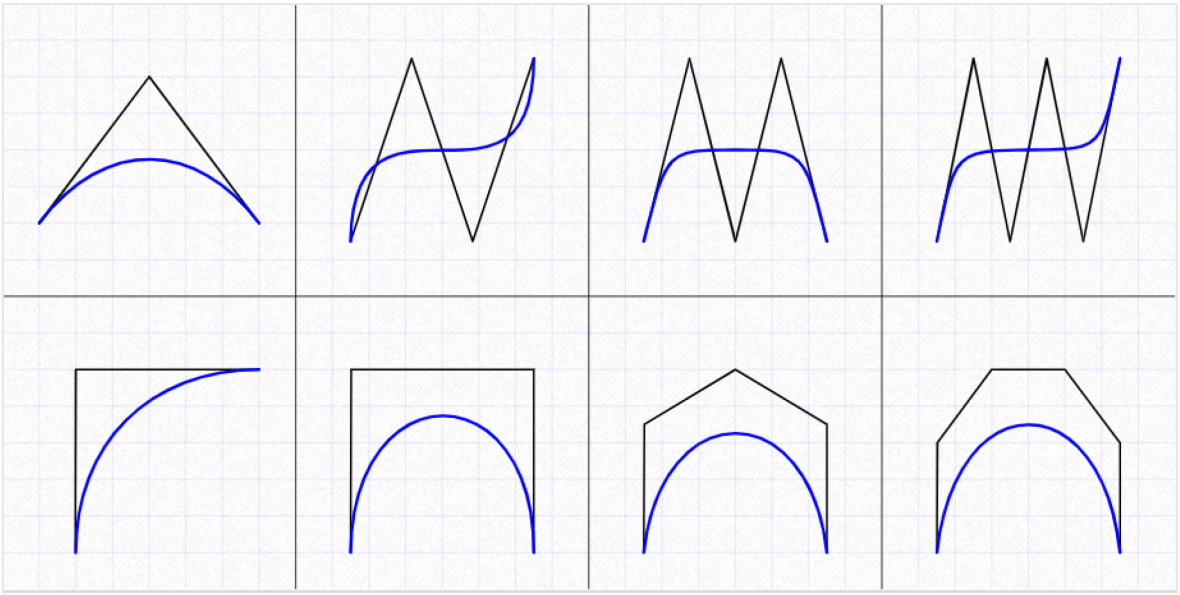
kde t je parametr v rozsahu 0 až 1 a $B(t)$ je bod na křivce odpovídající hodnotě t. Pokud se tedy hodnota bodu t bude postupně zvyšovat od nuly k jedné, bude se bod $B(t)$ úměrně posouvat po křivce od počátku $P0$ až do konce Pn.

V této rovnici představuje první člen $(1-t)^n \cdot P_0$ váhu prvního řídicího bodu P0 ke křivce a poslední člen $(t^n \cdot P_n)$ představuje váhu posledního řídicího bodu Pn k finální podobně křivky. Zbývající členy zahrnují kombinaci ostatních řídicích bodů $P_1, P_2, \ldots, P_{(n-1)}$ a parametru t.

Člen $(1-t)^n$ se nazývá váhový faktor a představuje vliv prvního řídicího bodu $P0$ na křivku. Podobně výraz t^n představuje vliv posledního řídicího bodu Pn na křivku. Zbývající členy představují vliv ostatních řídicích bodů na křivku, přičemž stupeň vlivu závisí na hodnotě t. [20]

V praxi se nejčastěji využívají kubické křivky (se 4 řídicími body) kvůli jejich dobré rovnováze mezi flexibilitou a výpočetní náročností. Stupně Bézierových křivek však nejsou omezeny na 4 řídicí body, vyšší stupně umožňují vytvářet složitější a hladší tvary a v praxi se s nimi lze také setkat.

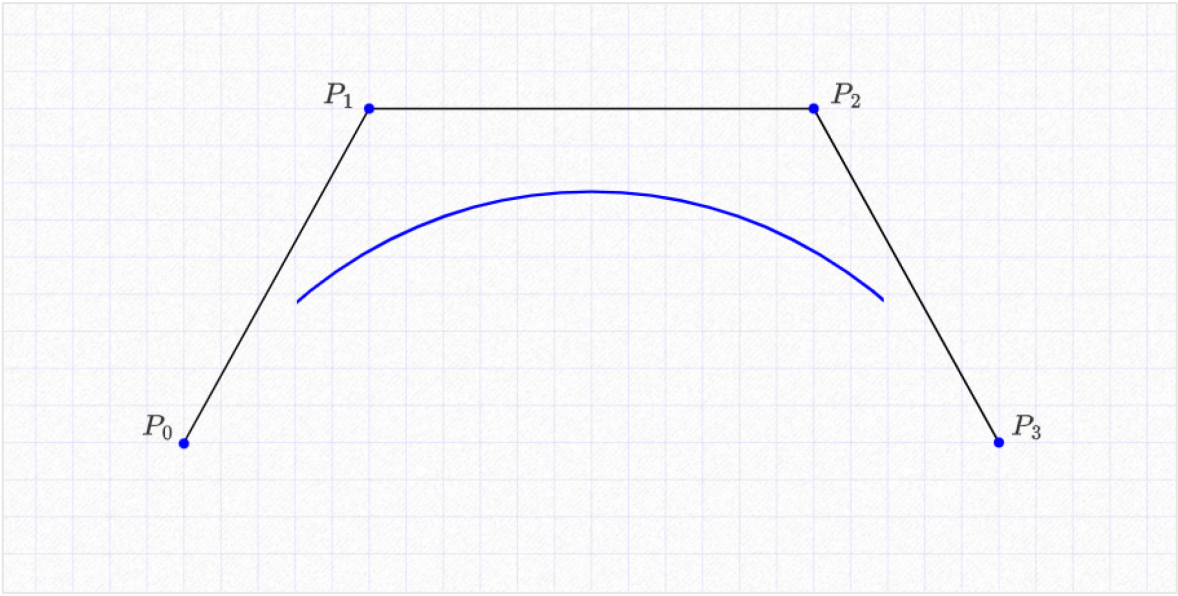
Nicméně, s vyšším počtem řídicích bodů se zvyšuje i složitost křivky, což může vést k nárůstu výpočetní náročnosti. Proto se často používají tzv. Bézierovy křivky nižšího stupně (např. kubické) a skládají se z více segmentů, aby se dosáhlo požadovaného tvaru s menší výpočetní zátěží.



Obrázek 24: Ukázka Bézierových křivek od kvadratické až po sedmistupňové

3.3.2 B-spline křivky

B-spline křivky jsou zobecněním Bézierových křivek, které poskytují lepší kontrolu a flexibilitu v tvarování křivek. Narozdíl od Bézierových křivek, které jsou ovlivňovány všemi kontrolními body, B-spline křivky umožňují lokální kontrolu nad tvarováním. To znamená, že změna jednoho kontrolního bodu ovlivní pouze část křivky v jeho blízkosti. Díky tomu je manipulace s B-spline křivkami snazší a přesnější.



Obrázek 25: Ukázka kubické B-spline křivky

Tvar B-spline křivky lze vypočítat pomocí následujícího vzorce:

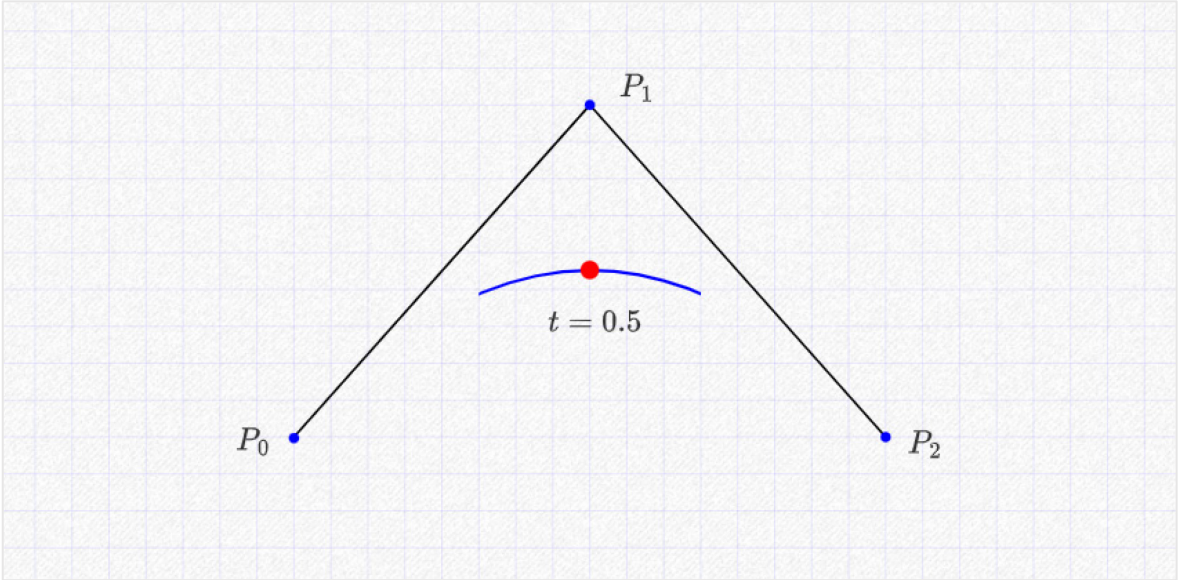
$$C(t) = \sum_{i}^{n} N_{i,p}(t) \cdot P_i$$

kde $C(t)$ je bod na křivce v parametrickém čase t, $N_{i,p}(t)$ je B-spline základní funkce řádu p, P_i jsou kontrolní body křivky, a $\sum$ značí součet přes všechny kontrolní body. B-spline základní funkce $N_{i,p}(t)$ se vypočítává pomocí de Boor-Coxova algoritmu, který je rekurzivní vztah:

$$N_{i,0}(t) = \begin{cases} 1 \text{ pokud } t_i \leq t < t_{i+1} \text{ a } t_i < t_{i+1} \\ 0 \text{ jinde} \end{cases}$$

$$N_{i,p}(t) = \frac{t - t_1}{t_{i+p} - t_i} \cdot N_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{t+p+1} - t_{i+t}} \cdot N_{i+1,p-1}(t)$$

kde t_i jsou uzlové body, které jsou součástí uzlového vektoru $T = \{t_0, t_1, \dots, t_{n+p+1}\}$. Uzlový vektor T určuje rozložení a váhu základních funkcí. Jeho volbou můžeme ovlivnit hladkost a průběh křivky. B-spline křivka potom interpoluje mezi kontrolními body, podle hodnot parametru t a základních funkcí $N_{i,p}(t)$. [20]



Obrázek 26: Ukázka kvadratické B-spline křivky s vyznačeným středem

3.3.3 NURBS křivky

Jedná se o rozšíření B-spline křivek, které umožňuje přesnější kontrolu nad tvarováním a modelováním složitých tvarů. NURBS křivky se definují pomocí řady kontrolních bodů, váhových hodnot a uzlového vektoru. Kontrolní body určují tvar a polohu křivky, váhové hodnoty ovlivňují její tvar a uzlový vektor definuje rozdělení parametrů křivky. B-spline křivky jsou zvláštním případem NURBS křivek. B-spline křivky nemají váhové hodnoty a jsou tedy vždy definovány pouze pomocí kontrolních bodů a uzlového vektoru. Díky tomu jsou B-spline křivky jednodušší a méně flexibilní než NURBS křivky, které umožňují vyšší úroveň kontroly nad tvarováním křivek pomocí váhových hodnot.

Hlavní výhodou NURBS křivky je umožnění přesné reprezentace řady geometrických tvarů, včetně kruhů a elips.

NURBS křivka se vypočítá pomocí kombinace B-spline bázových funkcí a váhových hodnot. Tvar NURBS křivky můžeme zapsat jako:

$$C(u) = \sum_{i=0}^{n} R_i(u) \cdot P_i$$

kde u je parametr křivky, P_i jsou kontrolní body, a $R_i(u)$ jsou racionální bázové funkce. Racionální bázové funkce $R_1(u)$ se vypočítají jako:

$$R_i(u) = \frac{N_{i,}p(u) \cdot w_i}{\sum_{j'=0}^{n} N_{j,}P(u) \cdot w_j}$$

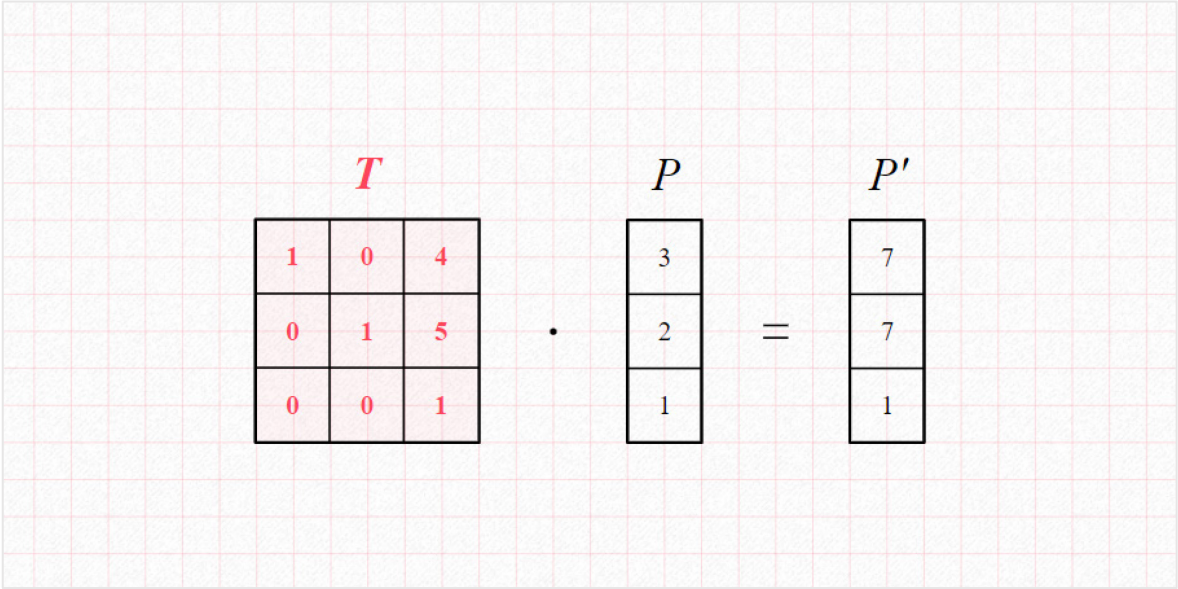
kde $N_{i,}p(u)$ jsou Bázové funkce stupně p, w_i jsou váhové hodnoty pro kontrolní body, a sumace probíhá přes všechny kontrolní body j.

Bázové funkce $N_{i,}p(u)$ se vypočítají pomocí de Boor-Coxovy rekurence. [21]

4 GEOMETRICKÉ OPERACE

4.1 Transformační matice

Transformační matice je matematická reprezentace geometrické transformace, která se používá pro změnu polohy, orientace, velikosti a tvaru objektů v prostoru. Transformační matice funguje na principu maticového násobení souřadnic bodů nebo objektů, které mají být transformovány.



Obrázek 27: Ukázka transformační matice

Transformační matice může být čtvercová matice o rozměrech 2x2, 3x3, 4x4 nebo vyšší, v závislosti na dimenzi prostoru (2D, 3D atd.) a typu transformace (posun, rotace, škálování atd.). Ve 2D prostoru se obvykle používají 2x2 nebo 3x3 matice, zatímco v 3D prostoru se používají 3x3 nebo 4x4 matice.

Obecně lze transformační matici $\boldsymbol{T}$ definovat jako čtvercovou matici o rozměrech $n \times n$, kde n je dimenze prostoru plus jeden pro posun (pokud se používají homogenní souřadnice). Matice $\boldsymbol{T}$ se pak aplikuje na vektor souřadnic bodu nebo objektu v prostoru (obvykle ve formě sloupcového vektoru) prostřednictvím maticového násobení.

Základní princip transformační matice spočívá v násobení matic, které aplikuje transformaci na souřadnice bodů nebo objektů. Například pokud je dán bod P se souřadnicemi $[x, y]$ a transformační matici $\boldsymbol{T}$ o rozměrech 2x2, výsledné souřadnice P' po transformaci budou dány následovně:

$$P' = \boldsymbol{T} \cdot P$$

kde · značí maticové násobení a P' jsou nové souřadnice bodu po transformaci. [20]

4.1.1 Sloupcový a řádkový tvar transformační matice

Pokud je bod $\boldsymbol{P}$ ve tvaru svislé matice, je to sloupcový vektor, například:

$$\boldsymbol{P}_s = \begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

kde x,y jsou souřadnice bodu ve 2D prostoru, w je váha, která se obvykle volí 1. Daný vektor je zapsán ve tvaru tzv. homogenních souřadnic, díky nimž je možno provádět transformace s využitím násobení transformačních matic.

Pokud je bod P ve tvaru vodorovné matice, je to řádkový vektor, například:

$$\boldsymbol{P}_{\check{\text{r}}} = [x \quad y \quad w] = [x \quad y \quad 1]$$

Rozdíl mezi svislou a vodorovnou maticí spočívá v tom, jak se matice násobí transformačními maticemi. Pro svislé matice (sloupcové vektory) násobíme transformační matici $\boldsymbol{T}$ zleva:

$$\boldsymbol{P}_s{}' = \boldsymbol{T}_s \cdot P_s$$

$$\boldsymbol{P}_s' = \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$$

Pro vodorovné matice (řádkové vektory) násobíme transformační matici $\boldsymbol{T}$ zprava:

$$\boldsymbol{P}_{\check{\text{r}}}' = \boldsymbol{P}_{\check{\text{r}}} \cdot \boldsymbol{T}_{\check{\text{r}}} = \boldsymbol{P}_{\check{\text{r}}} \cdot \boldsymbol{T}_s{}^T$$

$$\boldsymbol{P}_{\check{\text{r}}}' = [3 \quad 2 \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 5 & 1 \end{bmatrix} = [7 \quad 7 \quad 1] = [x' \quad y' \quad w']$$

Výsledný (transformovaný) bod P' bude také ve tvaru svislé nebo vodorovné matice, v závislosti na formě původního bodu P.

Výsledky násobení matic budou stejné, pokud je použit správný způsob násobení, ale je důležité být konzistentní při práci s transformacemi v homogenních souřadnicích. Platí přitom:

$$\boldsymbol{P}_{\check{\text{r}}}{}^T = \boldsymbol{P}_s$$

$$\begin{bmatrix} 7 \\ 7 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \\ 1 \end{bmatrix}$$

V další části uvedené transformační matice jsou uvažovány, pokud není uvedeno většinou jinak, pro tvar odpovídající zápisu řádkového vektoru pro dané body $\boldsymbol{P}$, tj. $\boldsymbol{P}_{\check{\mathrm{r}}}$

4.1.2 Kombinace transformačních matic

Velkou výhodou transformačních matic je možnost kombinovat více transformací do jedné matice prostřednictvím maticového násobení. Například pokud jsou dány transformace v pořadí $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ pak výsledná transformační matice může být v dále uvedených tvarech, přičemž je také důležité, v jaké tvaru bude uvažován samotný zápis daných bodů $\boldsymbol{P}$, tj. zda se bude jednat o sloupcový zápis nebo řádkový zápis. Tedy

$$\boldsymbol{P}'_S = \boldsymbol{T}_S \cdot \boldsymbol{P}_S = (\boldsymbol{T}_{S,C} \cdot \boldsymbol{T}_{S,B} \cdot \boldsymbol{T}_{S,A}) \cdot \boldsymbol{P}_S$$

kde

$$\boldsymbol{T}_S = \boldsymbol{T}_{S,C} \cdot \boldsymbol{T}_{S,B} \cdot \boldsymbol{T}_{S,A}$$

nebo

$$\boldsymbol{P}'_{\check{\mathrm{r}}} = \boldsymbol{P}_{\check{\mathrm{r}}} \cdot \boldsymbol{T}_{\check{\mathrm{r}}} = \boldsymbol{P}_{\check{\mathrm{r}}} \cdot (\boldsymbol{T}_{\check{\mathrm{r}},A} \cdot \boldsymbol{T}_{\check{\mathrm{r}},B} \cdot \boldsymbol{T}_{\check{\mathrm{r}},C})$$

kde

$$\boldsymbol{T}_{\check{\mathrm{r}}} = \boldsymbol{T}_{\check{\mathrm{r}},A} \cdot \boldsymbol{T}_{\check{\mathrm{r}},B} \cdot \boldsymbol{T}_{\check{\mathrm{r}},C}.$$

Dále také platí

$$\boldsymbol{T}_{\check{\mathrm{r}},A} = \boldsymbol{T}_{S,A}{}^T,\ \boldsymbol{T}_{\check{\mathrm{r}},B} = \boldsymbol{T}_{S,B}{}^T,\ \boldsymbol{T}_{\check{\mathrm{r}},C} = \boldsymbol{T}_{S,C}{}^T \text{ a } \boldsymbol{P}_{\check{\mathrm{r}}} = \boldsymbol{P}_S{}^T$$

tj. lze tedy zapsat výsledný vztah i ve tvaru

$$\begin{aligned}\boldsymbol{P}'_{\check{\mathrm{r}}} &= \boldsymbol{P}_{\check{\mathrm{r}}} \cdot \boldsymbol{T}_{\check{\mathrm{r}}} = (\boldsymbol{T}_S \cdot \boldsymbol{P}_S)^T = \boldsymbol{P}_S{}^T \cdot \boldsymbol{T}_S{}^T = \boldsymbol{P}_S{}^T \cdot (\boldsymbol{T}_{S,C} \cdot \boldsymbol{T}_{S,B} \cdot \boldsymbol{T}_{S,A})^T \\ &= \boldsymbol{P}_S{}^T \cdot \boldsymbol{T}_{S,A}{}^T \cdot \boldsymbol{T}_{S,B}{}^T \cdot \boldsymbol{T}_{S,C}{}^T = \boldsymbol{P}_{\check{\mathrm{r}}} \cdot \boldsymbol{T}_{\check{\mathrm{r}},A} \cdot \boldsymbol{T}_{\check{\mathrm{r}},B} \cdot \boldsymbol{T}_{\check{\mathrm{r}},C}\end{aligned}$$

Tedy výše určený vztah zobrazuje souvislost mezi řádkovým a sloupcovým zápisem nového hledaného bodu $\boldsymbol{P}'$ (nových souřadnic) při využití více transformačních matic při výpočtu nové pozice daného transformovaného bodu.

4.1.3 Inverze

Pro každou transformační matici $\boldsymbol{T}$ existuje inverzní matice $\boldsymbol{T}^{-1}$, která je reverzní transformační matice $\boldsymbol{T}$. Pokud je tedy bod P vynásoben maticemi $\boldsymbol{T}$ a poté $\boldsymbol{T}^{-1}$, dostaneme zpět původní souřadnice P.

Tato vlastnost inverzní matice je důležitá pro mnoho aplikací v počítačové grafice a vizualizaci, například pro korekci perspektivy nebo transformace objektů mezi různými souřadnicovými systémy. Inverzní matice se vypočítá jako $\boldsymbol{T}^{-1} = \frac{adj(T)}{\det(\boldsymbol{T})}$, kde $adj(T)$ je adjungovaná matice, což je transponovaná matice algebraických doplňků a $det(\boldsymbol{T})$ je determinant matice $\boldsymbol{T}$.

4.1.4 Jednotková matice

Existuje speciální transformační matice nazývaná jednotková matice, která nemá žádný efekt na souřadnice bodu. Jednotková matice je čtvercová matice, která má na diagonále jedničky a všechny ostatní prvky nulové. Pro 2D transformace je jednotková matice (identita) 2x2:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Pokud jsou uvažovány homogenní souřadnice, je u 2D transformace jednotková matice rozměru 3x3.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Pro 3D transformace při použití homogenních souřadnic má pak jednotková matice rozměr 4x4.

Když vynásobíme bod (ve vektorovém tvaru) jednotkovou maticí, získáme původní bod, protože jednotková matice nemá žádný efekt na souřadnice bodu. To znamená, že se nezmění posun, rotace ani škálování bodu.

4.2 Posun pomocí transformační matice

Posun v rovině lze zapsat pomocí transformační matice 3x3. Pro posun o vektor (x, y) lze použít následující transformační matici:

$$\boldsymbol{T}_{posun} = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

Když je tato matice aplikována na bod ve 2D prostoru, který je reprezentován homogenními souřadnicemi $(x, y, 1)$, získáme:

$$P_h{}' = \boldsymbol{T}_{posun} \cdot P_h$$

4.3 Rotace pomocí transformační matice

Rotace ve 2D prostoru může být popsána pomocí 3x3 transformační matice. Podoba matice rotace závisí na úhlu rotace. Maticové vyjádření transformace otáčení má tvar:

$$\boldsymbol{T}_{rot} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.4 Změna měřítka pomocí transformační matice

Změna měřítka v 2D prostoru může být popsána pomocí 2x2 transformační matice. Matice změny měřítka závisí na hodnotách změny měřítka podél jednotlivých os (x, y). Obecná matice změny měřítka má následující tvar:

$$\boldsymbol{T}_{scale} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.5 Zrcadlení pomocí transformační matice

Zrcadlení (nebo zobrazení s inverzní orientací) v 2D prostoru může být popsáno pomocí 2x2 transformační matice. Matice zrcadlení závisí na ose, kolem které chceme provést zrcadlení.

Zrcadlení kolem osy x:

$$\boldsymbol{T}_{\text{reflect}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Zrcadlení kolem osy y:

$$\boldsymbol{T}_{\text{reflect}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

[20]

5 RASTERIZACE A VEKTORIZACE

5.1 Rasterizace

Rasterizace vektorového objektu je proces převodu matematicky definovaných tvarů a objektů, které vytvářejí vektorovou grafiku, na rastrový obrázek (bitmapu) tvořený pixely. Vektorové objekty jsou založeny na matematických rovnicích, které definují jejich tvar a umožňují snadnou škálovatelnost a manipulaci. Naopak, rastrové obrázky jsou složeny z mřížky pixelů, které mají pevnou velikost a rozlišení.

5.2 Antialiasing

Antialiasing je technika používaná v počítačové grafice ke zlepšení vizuální kvality obrazu tím, že se redukují viditelné artefakty, jako jsou například schodovité (zubaté) okraje. Tyto artefakty vznikají v důsledku omezeného rozlišení obrazovky a diskrétního způsobu, jakým jsou pixely reprezentovány.

Antialiasing funguje na principu zkreslování digitálního signálu tak, aby odpovídal analogovému, což zahrnuje použití filtrů a průměrování pixelů. [10] Existuje několik různých technik antialiasingu.

5.2.1 Multisampling Antialiasing (MSAA):

MSAA je metoda, která zvyšuje vzorkování pouze na okrajích objektů, kde je nejpravděpodobnější výskyt zubatých hran. Tímto způsobem se snižuje zatížení grafického procesoru (GPU) a zlepšuje se výkon ve srovnání s technikami, které používají plnou scénu.

5.2.2 Supersampling Antialiasing (SSAA):

SSAA je technika, která vykreslí scénu v mnohem vyšším rozlišení, než je zobrazováno na obrazovce, a poté sníží rozlišení zpět na původní velikost. Tímto způsobem se průměrují barvy sousedních pixelů, což vede k hladším okrajům. SSAA je výpočetně náročnější než MSAA, ale poskytuje lepší kvalitu obrazu.

5.2.3 Fast Approximate Antialiasing (FXAA):

FXAA je jednoduchá a rychlá metoda, která se zaměřuje na detekci hran ve scéně a jejich vyhlazování. Tato technika se provádí pouze na úrovni pixelů a nevyžaduje dodatečné

vzorkování. FXAA obvykle nemá tak vysokou kvalitu obrazu jako MSAA nebo SSAA, ale je rychlejší a méně náročná na výkon. [22]

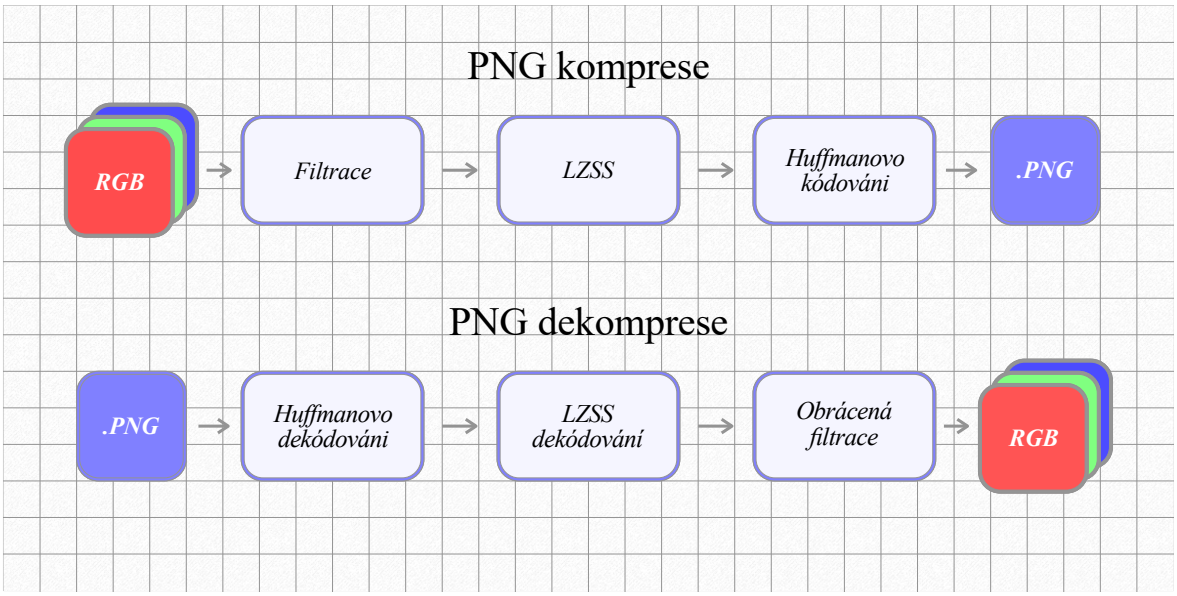
5.3 Komprese

Komprese v počítačové grafice je proces snižování velikosti datových souborů grafických obrazů s cílem snížit nároky na úložiště a zrychlit přenos dat. Obecně existují dva typy komprese: bezztrátová a ztrátová. [10]

5.3.1 Bezztrátová komprese

Tento typ komprese umožňuje snížit velikost souboru bez jakékoliv ztráty kvality obrazu. Po dekomprimaci získáme přesně stejný obraz jako před kompresí. Příklady bezeztrátových kompresních algoritmů zahrnují PNG a GIF.

Principy PNG komprese



Obrázek 28: Schéma PNG komprese a dekomprese

PNG (Portable Network Graphics) je formát bezztrátové komprese obrázků, který byl vytvořen jako náhrada za starší formát GIF. Komprese PNG funguje na základě metody nazývané deflate, což je kombinace algoritmu LZSS a Huffmanova kódování.

Filtrace v PNG je proces používaný před samotnou kompresí dat, který může zlepšit kompresní poměr. Filtrace je použita na jednotlivé řádky obrazových dat a využívá se pro předpovězení hodnot pixelů na základě hodnot jiných pixelů.

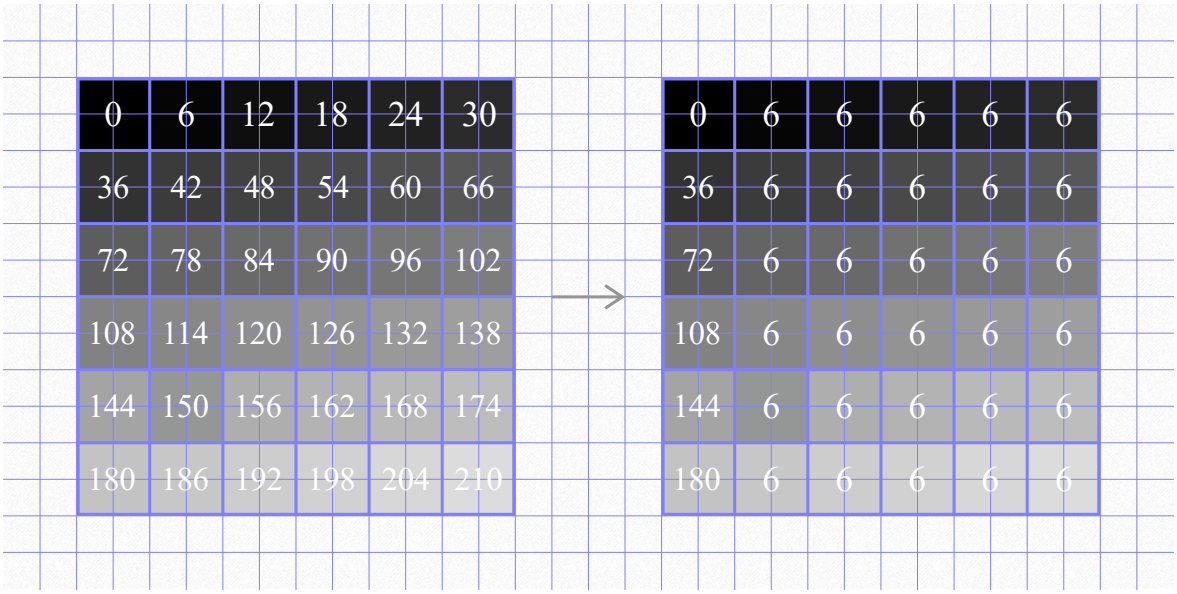
- **Nulový filtr**: Neaplikuje žádnou filtraci.

- **Sub filtr**: Odečte hodnotu levého pixelu od aktuálního pixelu.
- **Up filtr**: Odečte hodnotu pixelu výše od aktuálního pixelu.
- **Průměrný filtr**: Odečte průměrnou hodnotu levého pixelu a pixelu výše od aktuálního pixelu.
- **Paeth filtr**: Využívá předchozích tří pixelů (pixel vlevo, pixel výše, a pixel vlevo-výše) pro předpovězení hodnoty aktuálního pixelu.

Pro každý řádek obrazu je vybrán jeden z těchto filtrů, který minimalizuje celkovou hodnotu datového řádku, což zase vede k efektivnější kompresi v následujícím kroku deflate. Filtraci lze chápat jako formu predikce, která usnadňuje úlohu kompresního algoritmu. Princip Sub filtru lze matematicky zapsat následovně:

$$\begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \rightarrow \begin{bmatrix} A & B-A & C-B \\ D-C & E-D & F-E \\ G-F & H-G & I-H \end{bmatrix}$$

Písmena představují hodnoty jednotlivých pixelů. První pixel je vynechaný a hodnota každého následující pixelu je snížena o hodnotu předešlého pixelu. V případě, že by mělo dojít k záporným hodnotám se použije funkce modulo 256.

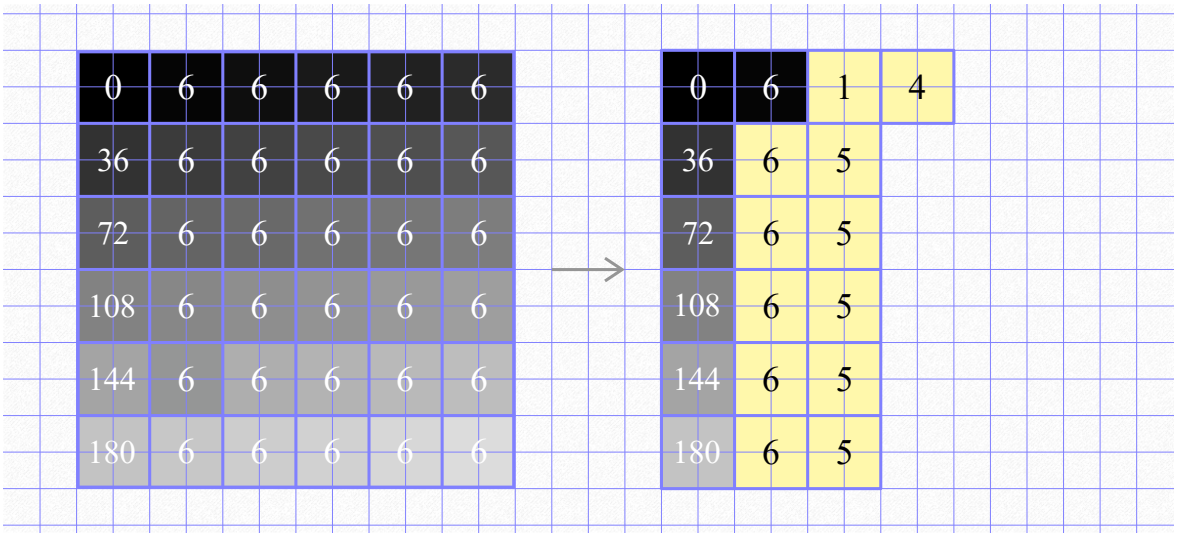


Obrázek 29: Ukázka Sub filtru v PNG kompresi

LZSS, neboli Lempel-Ziv-Storer-Szymanski, je algoritmus bezztrátové komprese dat, který pracuje na principu nahrazování opakujících se řetězců odkazy na jejich předchozí výskyty.

Klíčem k fungování LZSS je tzv. klouzavé okno, které sleduje nedávnou historii vstupních dat. Když algoritmus narazí na řetězec, který se již v okně vyskytuje, nahradí tento řetězec

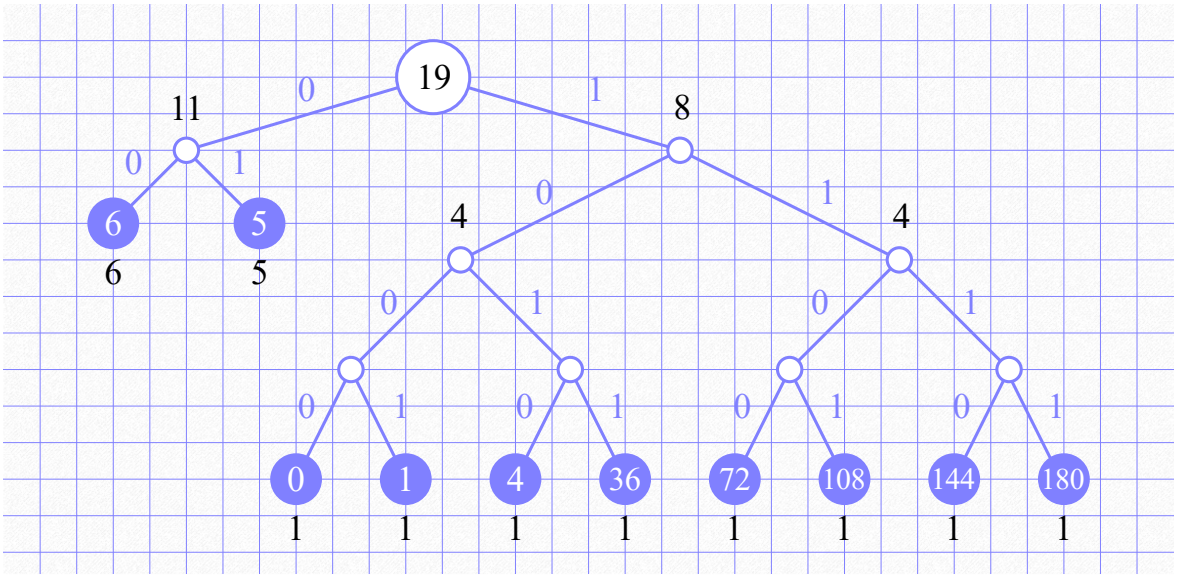
odkazem na jeho pozici a délku v okně. Toto odkazování je efektivní zejména v případech, kdy se v datech často opakují delší řetězce.



Obrázek 30: Ukázka LZSS algoritmu pro filtrovaný obraz

Huffmanovo kódování je populární metoda bezztrátové komprese dat, která využívá principu variabilní délky kódů pro reprezentaci symbolů. Byla vyvinuta Davidem Huffmanem v roce 1952.

Algoritmus Huffmanova kódování začíná vytvořením tabulky frekvencí pro každý symbol vstupních dat. Poté se vytvoří tzv. Huffmanův strom, což je binární strom, kde každý list reprezentuje jeden symbol. Častější symboly jsou umístěny blíže ke kořeni stromu, zatímco méně časté symboly jsou umístěny dále.



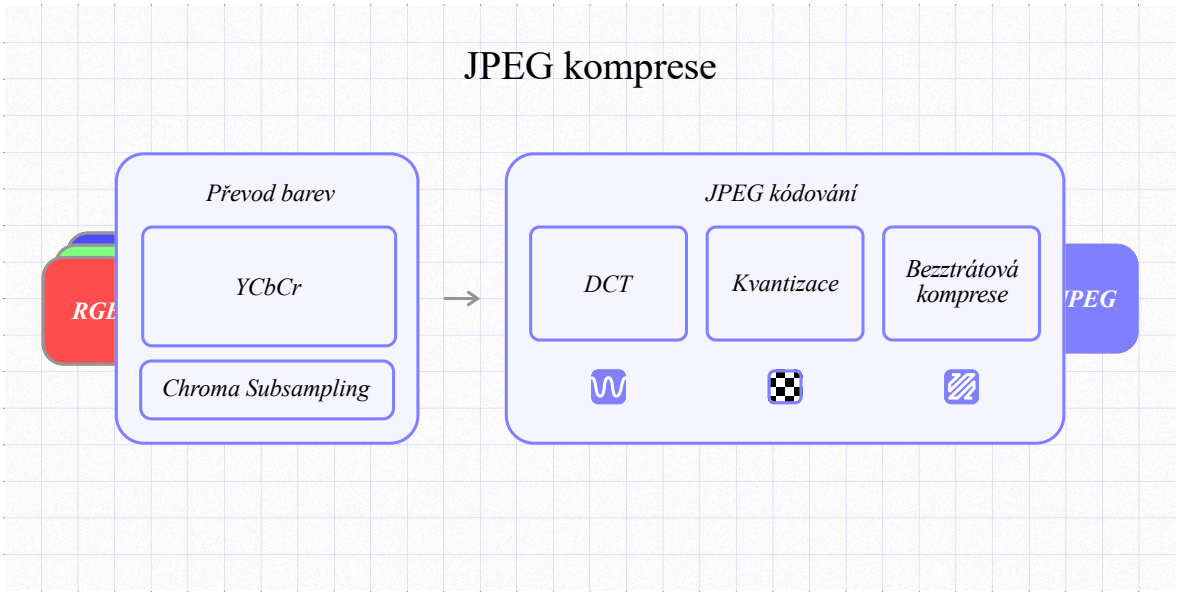
Obrázek 31: Ukázka Huffmanova kódování

PNG také využívá CRC-32 (Cyclic Redundancy Check) pro kontrolu integrity dat. Tento kontrolní součet zajišťuje, že při přenosu nebo ukládání souboru nedošlo k poškození dat. [23]

5.3.2 Ztrátová komprese

Tento typ komprese snižuje velikost souboru za cenu určité ztráty kvality obrazu. Ztrátová komprese je často používána pro fotografie nebo jiné obrazy s vysokým rozlišením, kde je malá ztráta kvality nepostřehnutelná nebo přijatelná. Příklad ztrátové kompresní techniky je JPEG.

Principy JPEG komprese



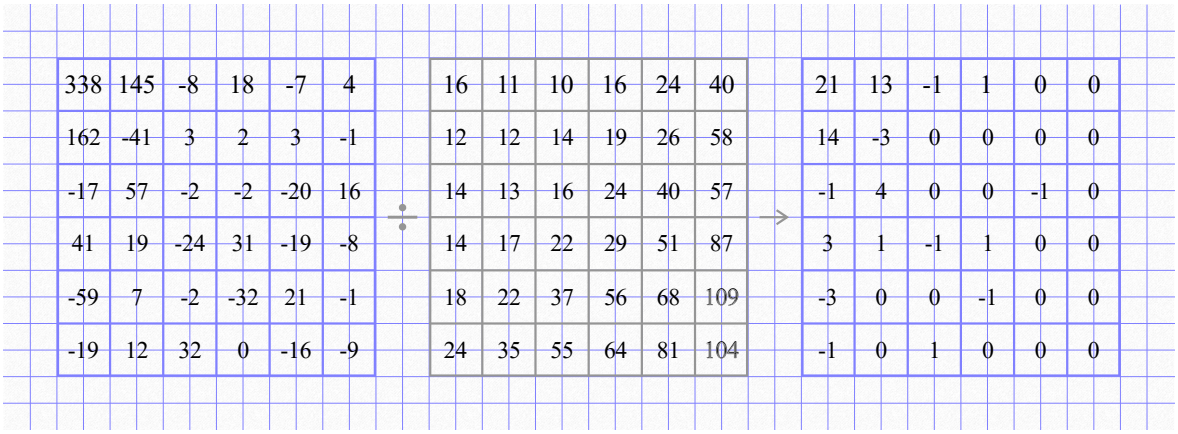
Obrázek 32: Schéma JPEG komprese

Obraz je převeden z RGB (červená, zelená, modrá) do YCbCr barevného prostoru. Y reprezentuje jas, zatímco Cb a Cr reprezentují chromatické složky (barevné informace). Tento krok zohledňuje skutečnost, že lidské oko je citlivější na jas než na barvu.

Chromatické složky Cb a Cr jsou sníženy, aby se dosáhlo menší velikosti souboru. Tento krok využívá skutečnost, že lidské oko je méně citlivé na barevné detaily než na detaily jasové.

Obraz je rozdělen do 8x8 pixelových bloků. Každý 8x8 blok prochází diskrétní kosinovou transformací, která převádí obrazová data do frekvenčního spektra. Tento krok umožňuje identifikovat a oddělit důležité obrazové informace od méně důležitých.

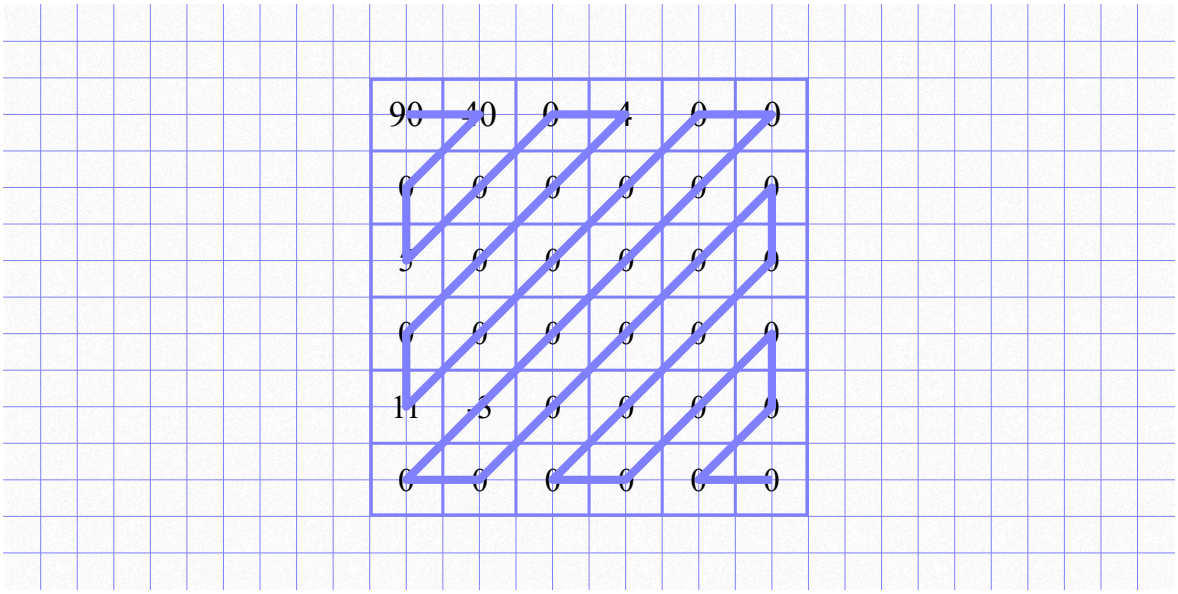
Frekvenční koeficienty výsledného spektra jsou kvantizovány (zaokrouhleny) na celá čísla. Kvantizační tabulka určuje, jakým způsobem se provádí zaokrouhlení a jaký stupeň komprese bude dosažen. Vyšší úroveň komprese způsobí ztrátu více informací a nižší kvalitu obrazu.



Obrázek 33: Ukázka kvantizace obrazu

Run-Length Encoding, je jednoduchá metoda bezztrátové komprese dat. Je velmi efektivní pro data, která obsahují velké sekvence opakujících se hodnot.

Základní princip RLE je nahradit opakující se sekvence stejných hodnot jediným párem hodnot: hodnotou symbolu a počtem opakování. Například, sekvence "AAAAA" by byla v RLE kódování reprezentována jako $(A, 5)$, což znamená, že písmeno A se opakuje pětkrát.



Obrázek 34: Ukázka RLE

V JPEG kompresi se obrazová data pro RLE nečtou po řádcích, ale tzv. cikcak metodou, která zvyšuje pravděpodobnost seřazení více stejných čísel za sebou. Po provedení RLE je dále možné aplikovat Huffmanovo kódování pro ještě větší kompresi. [24]

5.3.3 Vektorizace

V počítačové grafice se vektorizace rastru vztahuje k procesu převodu rastrového obrazu, jako je bitmapa nebo pixelová reprezentace, na vektorový formát. Rastrový obraz je tvořený maticí pixelů, zatímco vektorový obraz je reprezentován pomocí matematických objektů jako jsou body, čáry, křivky a mnohoúhelníky. Vektorizace rastru zahrnuje rozpoznání geometrických tvarů v rastrovém obrazu a jejich převedení na vektorové prvky.

6 BAREVNÉ FORMÁTY

Barevné formáty jsou systémy používané k reprezentaci a popisu barev pomocí matematických modelů a souborů číselných hodnot.

6.1 RGB

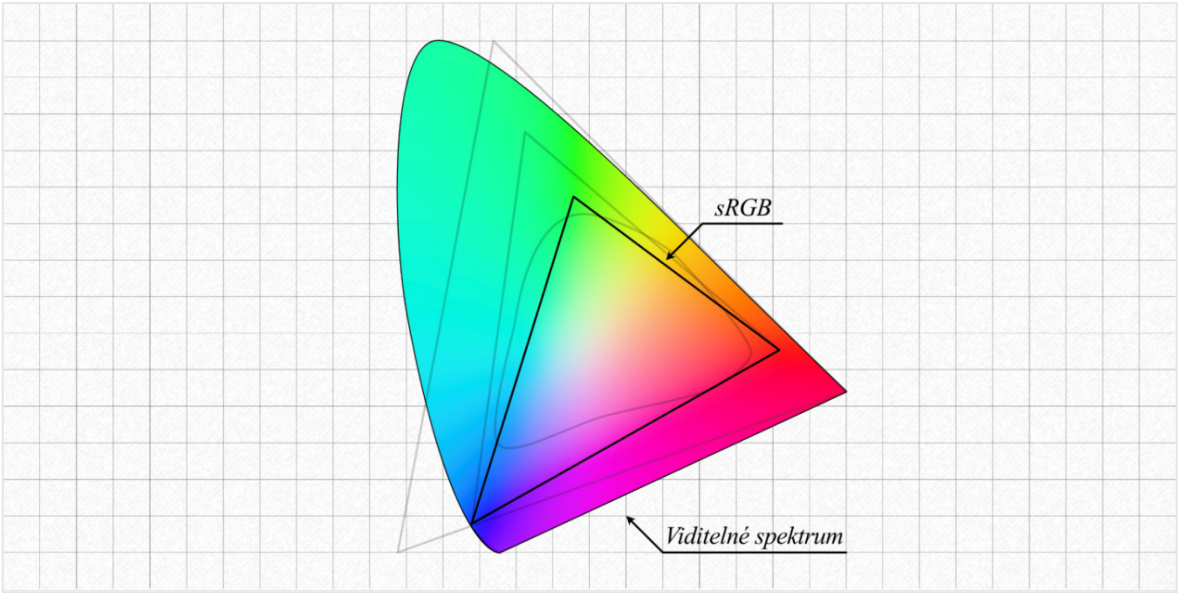
RGB je zkratka pro tři základní barvy: červenou (Red), zelenou (Green) a modrou (Blue). Každá barva v RGB modelu je kombinací těchto tří základních barev v různých intenzitách.

RGB funguje na principu aditivního míchání barev. Při míchání červené, zelené a modré ve stejných intenzitách můžeme dosáhnout celé škály barev. Když se všechny tři základní barvy přidají dohromady ve svém maximálním intenzitě, vytvoří bílou barvu. Naopak, když nejsou žádné z těchto barev přítomny, vytvoří černou barvu.

V digitálním obraze je každý pixel reprezentován hodnotami červené, zelené a modré. Tyto hodnoty jsou obvykle ukládány jako 8bitová čísla, což znamená, že každá barva může nabývat hodnot od 0 do 255. Tímto způsobem je možné vytvořit až 16,7 milionu odstínů barev (256 × 256 × 256).

RGB formát je široce používán ve fotografii, počítačové grafice, televizi a displejích. Je ideální pro zařízení, která vytvářejí barvy pomocí světla, jako jsou monitory, televize nebo projektory. [25]

6.1.1 sRGB



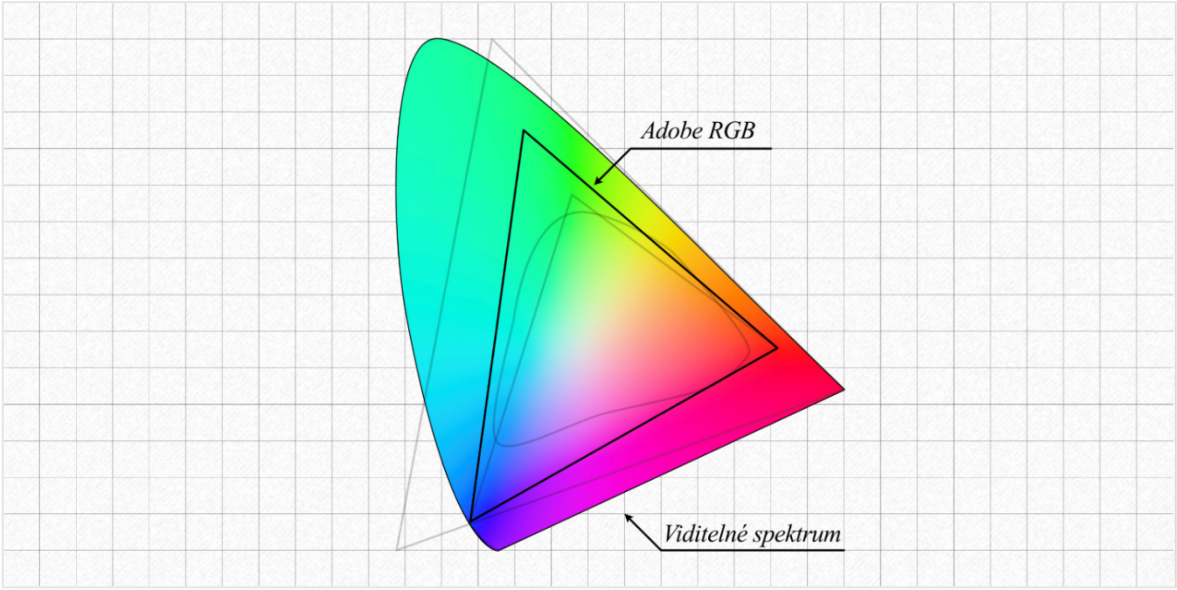
Obrázek 35: Barevné spektrum sRGB v porovnání s viditelným spektrem

sRGB je zkratka pro "standard Red Green Blue" a je to standardní barevný prostor (nebo barevný profil) pro internet a mnoho zařízení, jako jsou digitální fotoaparáty, monitory, tiskárny a skenery. Byl navržen v roce 1996 společnostmi Microsoft a Hewlett-Packard jako univerzální standard pro správu barev a dosud je to nejběžnější barevný profil používaný v digitálních médiích.

sRGB definuje konkrétní sadu barev a jejich vztahů, které mají být zobrazeny na displejích nebo reprodukovány tiskem. Jeho cílem je zajistit, aby barvy na jednom zařízení (například na monitoru) byly co nejvíce podobné barvám zobrazeným na jiném zařízení (například na tiskárně).

Jednou z hlavních výhod sRGB je jeho široká podpora a kompatibilita s různými zařízeními a softwarovými aplikacemi. Díky tomu se stalo de facto standardem pro internet a mnoho dalších oblastí. Nicméně sRGB má omezený dynamický rozsah a barevné podání ve srovnání s jinými, modernějšími a pokročilejšími barevnými profily, jako jsou Adobe RGB nebo ProPhoto RGB.

6.1.2 Adobe RGB



Obrázek 36: Barevné spektrum Adobe RGB v porovnání s viditelným spektrem

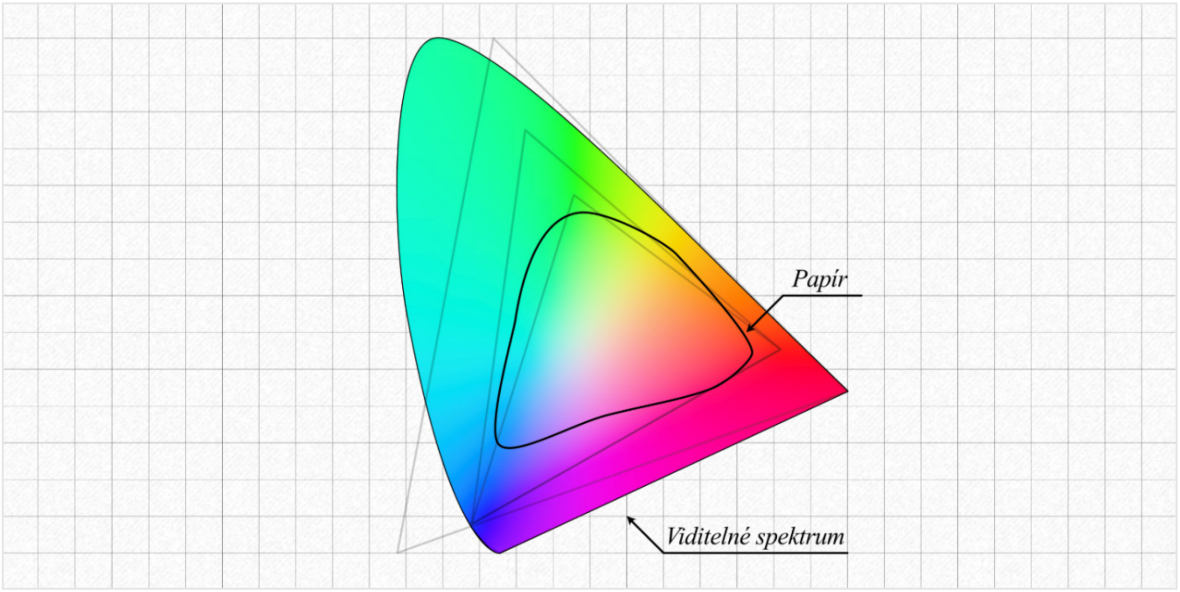
Adobe RGB je pokročilý barevný profil navržený společností Adobe Systems v roce 1998. Je široce používán profesionálními fotografy, grafickými designery a lidmi pracujícími v oblasti tisku a publikování. Adobe RGB nabízí širší barevný rozsah než sRGB, což umožňuje reprodukovat více barev a odstínů, zejména v oblastech zelené a azurové barvy.

Adobe RGB byl navržen tak, aby poskytoval lepší reprodukci barev pro tisk a další profesionální účely, kde je důležitá přesnost barev. Jeho širší barevný gamut zahrnuje více barev, které jsou schopny být reprodukovány pomocí tiskových technologií, jako je CMYK (Cyan, Magenta, Yellow, Key - černá), což je běžně používaný proces pro tisk na papír.

Nicméně, Adobe RGB není ideální pro běžné účely nebo pro zobrazení na webu, protože většina zařízení a webových prohlížečů je navržena pro sRGB a nemusí správně zobrazovat barvy obsažené v Adobe RGB. Při použití Adobe RGB je důležité správně spravovat barevné profily a konvertovat je na sRGB, pokud je obrázek určen pro web nebo pro zobrazení na zařízeních, které nepodporují Adobe RGB. [26]

6.2 CMYK

CMYK je zkratka pro Cyan, Magenta, Yellow, and Key (Black), což jsou čtyři základní barvy používané v procesu tisku a barevném formátu. CMYK je založen na substraktivním modelu míchání barev, což znamená, že barvy se míchají dohromady, aby se odstranilo světlo. To je opačný princip v porovnání s aditivním systémem RGB (Red, Green, Blue), který se používá pro digitální zobrazení, kde se barvy míchají dohromady, aby vytvořily světlejší odstíny. [26]



Obrázek 37: Barevné spektrum tištěných barev na papír v porovnání s viditelným spektrem

6.2.1 Princip fungování CMYK

Vstupní digitální obraz (např. ve formátu RGB) je převeden na CMYK barevný prostor pomocí barevného profilu a konverzního algoritmu. Během tohoto procesu jsou barvy rozloženy na čtyři základní složky: azurová (C), purpurová (M), žlutá (Y) a černá (K).

Pro tisk jsou použity čtyři základní barvy (CMYK) ve formě inkoustu nebo toneru. Tyto barvy se nanášejí na tiskový materiál (papír, plátno atd.) ve formě malých teček. Tyto tečky se kombinují a překrývají (často za pomoci tepla), aby vytvořily širokou škálu odstínů a barev.

Když se tyto tečky překrývají, dochází k substraktivnímu míchání, které odstraňuje světlo a vytváří tmavší odstíny. Například kombinace azurové a purpurové vytvoří modrou, azurové a žluté zelenou a purpurové a žluté červenou.

Kombinace všech tří (C, M, Y) vytvoří teoreticky černou, ale v praxi často vytváří spíše tmavě šedou. Proto je přidána černá (K) jako čtvrtá složka, aby se dosáhlo syté černé a širší škály odstínů. [10]

7 GRAFICKÉ FORMÁTY

7.1 Rastrové

7.1.1 BMP

BMP (Bitmap) je grafický formát souboru používaný pro ukládání rastrových (bitmapových) digitálních obrázků. BMP byl vyvinut společností Microsoft a je součástí operačního systému Windows od jeho prvních verzí. BMP je také známý jako DIB (Device Independent Bitmap) formát, protože byl navržen pro nezávislost na konkrétním zařízení. [27]

BMP soubor se skládá ze tří hlavních částí: hlavičky souboru, informační hlavičky a datové části. BMP formát nevyužívá žádnou kompresi nebo používá pouze jednoduchou bezztrátovou kompresi (RLE – Run Length Encoding), což znamená, že soubory BMP mají tendenci být velké. To je důvod, proč tento formát není ideální pro přenos nebo ukládání obrázků na webu, kde se často používají formáty s efektivnější kompresí, jako je JPEG nebo PNG. Nicméně BMP je stále vhodný pro jednoduché obrázky nebo pro použití v určitých aplikacích, kde je důležitá rychlost načítání nebo bezztrátová kvalita obrázku. [20]

7.1.2 JPEG

JPEG (Joint Photographic Experts Group) je běžný grafický formát používaný pro ukládání a přenos fotografií a obrazových dat. Tento formát byl vytvořen a standardizován skupinou Joint Photographic Experts Group, odtud název JPEG. Jedná se o metodu komprese obrazu, která umožňuje snížit velikost souboru bez výrazné ztráty kvality. JPEG využívá ztrátovou kompresi, což znamená, že některé obrazové informace jsou odstraněny při kompresi a nemohou být plně obnoveny. [26]

7.1.3 PNG

PNG (Portable Network Graphics) je grafický formát, který byl vytvořen jako alternativa k formátu GIF (Graphics Interchange Format) a je navržen pro bezestrátovou kompresi obrázků. PNG je široce používán na webu, zejména pro obrázky s průhledností a vyšší kvalitou barev.

Výhody PNG zahrnují bezztrátovou kompresi, podporu průhlednosti a širokou paletu barev. Nevýhodou může být větší velikost souboru ve srovnání s jinými formáty, jako je JPEG,

který používá ztrátovou kompresi a je vhodnější pro fotografie nebo obrázky s mnoha barevnými přechody a detaily.

PNG je vhodnější pro situace, kdy je důležitá bezztrátová komprese a zachování kvality obrazu, například pro loga, ikony, textové grafiky, screenshoty nebo ilustrace. Na druhou stranu, JPEG může být lepší volbou pro fotografie nebo obrázky s komplexním obsahem a mnoha barevnými přechody, kde malé ztráty kvality nejsou tak kritické. [20]

7.1.4 WebP

Jak již z názvu vyplívá, formát WebP je určen primárně pro užití na webových stránkách. Tento formát je zvláštní tím, že nabízí bezstratovou i se ztrátou komprimaci dat. WebP soubory bývají obecně menší než soubory ve formátech JPEG, PNG a GIF, při zachování stejné kvality obrazu. Pokud je tedy cílem nejlépe optimalizovat rychlost webových stránek, je WebP vhodnou volbou.

7.2 2D Vektor

7.2.1 SVG

SVG (Scalable Vector Graphics) je grafický formát založený na XML, který je určen pro popis dvourozměrné vektorové grafiky. Jedná se o otevřený standard vyvinutý organizací W3C (World Wide Web Consortium). Vektorová grafika se liší od rastrové (bitmapové) grafiky, jako jsou formáty JPEG nebo PNG, tím, že je založena na matematických křivkách a tvarech, které jsou definovány pomocí bodů, čar a křivek.

SVG funguje tak, že vytváří hierarchii elementů XML, které definují jednotlivé části vektorového obrázku, jako jsou tvary, barvy, výplně, čáry, texty, transformace a další. Prohlížeče pak zpracovávají a zobrazují tyto elementy podle definovaných atributů a vztahů.

7.2.2 DXF

DXF je souborový formát vytvořený společností Autodesk v roce 1982, který umožňuje výměnu dat mezi různými CAD (Computer-Aided Design) aplikacemi. Je navržen tak, aby byl kompatibilní s různými CAD softwarem bez ohledu na výrobce.

DXF soubory mají strukturovaný formát sestávající z hlaviček, tabulek, bloků a entit. Hlavičky obsahují informace o souboru, tabulky definují vrstvy, barvy a další atributy, bloky

obsahují definice opakovaně používaných objektů a entity představují grafické prvky, jako jsou čáry, kruhy a oblouky. [28]

7.3 3D vektor

7.3.1 STL

STL (STereoLithography) je grafický formát souboru, který se běžně používá v oblasti 3D tisku a počítačového modelování. Tento formát umožňuje ukládání informací o geometrii a tvaru trojrozměrných objektů. STL byl původně vyvinut společností 3D Systems v roce 1987, ale od té doby se stal široce používaným standardem v 3D tisku a CAD (Computer-Aided Design) softwarech.

STL formát pracuje s trojúhelníkovou reprezentací 3D modelu. Objekty jsou rozděleny na malé trojúhelníkové plošky, které dohromady tvoří povrch 3D modelu. Každý trojúhelník je definován třemi vrcholy a normálovým vektorem, který určuje orientaci trojúhelníku ve 3D prostoru. STL formát neobsahuje žádné informace o barvách, texturách nebo materiálech.

STL soubory mohou být uloženy ve dvou formátech: ASCII nebo binární. ASCII STL soubor obsahuje čitelný text, který popisuje trojúhelníky modelu, zatímco binární STL soubor ukládá stejné informace ve formě binárních dat, což vede k menším velikostem souborů. [28]

7.3.2 OBJ

OBJ (Wavefront) je grafický formát souboru, který byl původně vyvinut společností Wavefront Technologies pro jejich grafický software Advanced Visualizer. Později byl tento formát převzat společností Autodesk a stal se jedním z nejběžnějších formátů pro 3D modelování a výměnu dat mezi různými 3D modelovacími programy. OBJ je textový formát, což znamená, že data jsou uložena v čitelné formě pro člověka.

OBJ formát funguje tak, že ukládá informace o geometrii 3D modelu, jako jsou vrcholy, texturové souřadnice a normály, a také informace o tom, jak jsou tyto prvky spojeny do trojúhelníkových plošek (polygonů). [29]

7.3.3 X3D

X3D je standardizovaný formát pro 3D počítačovou grafiku. X3D nabízí širokou škálu základních geometrických tvarů (např. koule, válce, kužely), které mohou být dále upravovány

pomocí různých modifikátorů a transformací. X3D také podporuje různé typy světel a stínování, což umožňuje vytváření realistických stínů a odlesků, zároveň umožňuje vytváření animací a interakcí mezi objekty nebo s uživatelem pomocí časovačů, událostí a skriptování. Formát X3D je často používán k zobrazování 3D objektů na webu. [30]

II. PRAKTICKÁ ČÁST

8 CÍL PRAKTICKÉ ČÁSTI

Cílem praktické části bylo vytvořit dostatečné množství podkladků týkajících se tvorby grafických objektů, transformace grafických objektů pomocí transformační matice, aplikace nejpoužívanějších filtrů s využitím konvoluční matice, barevných modelů a grafických formátů.

Smyslem těchto podkladů je nastínit principy a techniky, které za těmito známými operacemi a pojmy stojí. Z toho důvodu byl pro většinu prací použit program Processing.py, ve kterém je možné zrekonstruovat širokou škálu transformací a efektů pomocí jazyka Python s možností interaktivního náhledu. Teprve s hotovým kódem Processing.py a s nabytými znalostmi k danému tématu bylo přistoupeno k tvorbě scriptů a pluginů, které vycházely ze získaných zkušeností.

9 GRAFICKÉ OBJEKTY

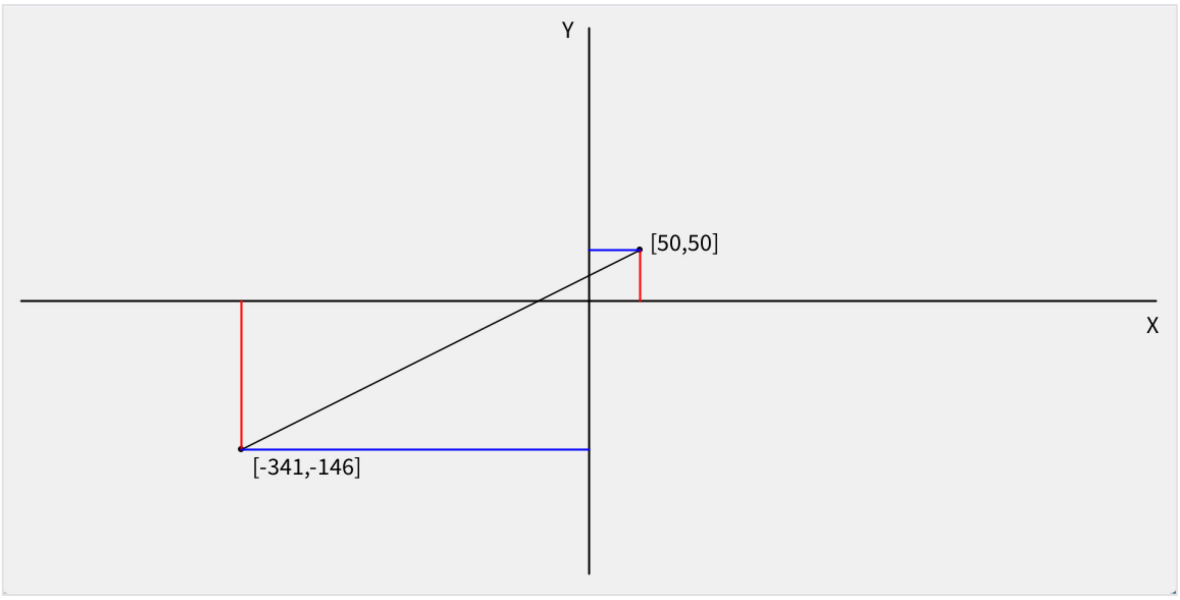
Jednou ze stěžejních oblastí, kterou se praktická část zabývá je tvorba základních objektů. Tato kapitola se zaměřuje na tři typy objektů, pomocí kterých je možné vytvořit prakticky jakýkoli vektorový obrazec. Jedná se o úsečku, kruh nebo elipsu a oblou křivku. Nejprve bylo třeba zrekonstruovat objekty v Processing.py. Tento proces poskytl dobrý přehled o vlastnostech jednotlivých objektů. Vytvořené modely jsou interaktivní, lze tak manipulovat s jejich velikostí nebo zakřivením kurzorem myši. Následně byly vytvořeny tutoriály pro nástroje v programu Inkscape

9.1 Úsečka

Úsečka je ve vektorové grafice jeden z nejjednodušších objektů. Při první tvorbě objektu v Processing.py je vhodné začít právě s ní, protože se můžeme více zaměřit na syntaxi Processing.py a méně na logiku objektu.

9.1.1 Tvorba úsečky v Processing.py

Pro vytvoření úsečky v Processing lze využít vestavěnou funkci ***line*(*x*1, *y*1, *x*2, *y*2)**, do které lze zadat souřadnice bodů úsečky. V tomto případě bude funkce line vypadat takto: ***line*(50, 50, *mouseX-width*/2, *mouseY-width*/2)**. První dva parametry funkce definují statický bod, a ve zbylých dvou parametrech získáváme souřadnice myši pomocí vestavěné proměnné ***mouseX/Y-width***.



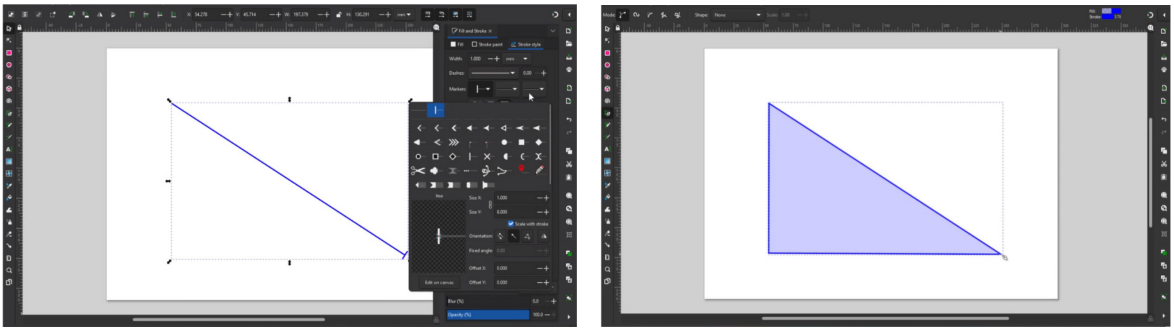
Obrázek 38: Výstup kódu Processing.py pro zobrazení úsečky

Kód - Processing.py

```
def drawTracers():
    ...
    pushMatrix()
    translate(width/2, height/2)
    stroke(0)
    strokeWeight(1.5)
    line(50,-50,mouseX-width/2,mouseY-height/2)
    ...
def drawBase():
    ...
```

9.1.2 Práce s úsečkou v Inkscape

Po vytvoření interaktivní úsečky v Processing.py byl sestaven tutoriál pro práci s rovnou linií v Inkscape. Tutoriál obsahuje základní definování linie, práci s atributy linie, jako je tloušťka čáry, barva nebo čerchování. Nakonec je ukázána možnost linii rozšířit o další body.

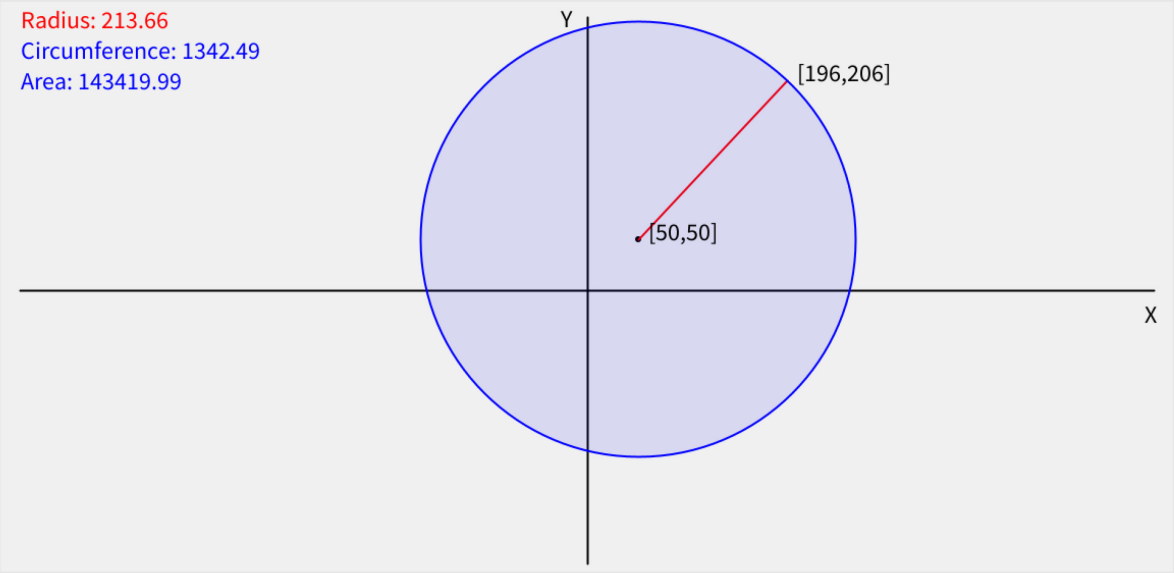


Obrázek 39: Náhled tutoriálu pro práci s úsečkou v Inkscape

9.2 Kružnice a elipsa

9.2.1 Tvorba kruhu v Processing.py

Následující kód je vizuální reprezentací kruhu pomocí Processing.py Začíná nastavením plátna a definováním počátečních konfigurací, jako jsou rozměry a velikost textu. Poté je na plátno nakreslen souřadnicový systém, který poskytuje kontext pro polohu kruhu a jeho vlastnosti, přičemž jsou použita základní kreslicí primitiva, jako jsou čáry a textové popisky pro osy.



Obrázek 40: Výstup kódu Processing.py pro tvorbu kruhu

Kód – Processing.py

```
def drawCircle():
    …
    circle_x = 50
    circle_y = -50
    circle_radius = dist(circle_x, circle_y, mouseX-width/2, mouseY-
        height/2)
    circle_circumference = 2 * PI * circle_radius
    circle_area = PI * circle_radius * circle_radius

    fill(0)
    noStroke()
    ellipse(circle_x, circle_y, 6, 6)

    stroke(255, 0, 0)
    line(circle_x, circle_y, mouseX-width/2, mouseY-height/2)

    stroke(0, 0, 255)
    fill(0, 0, 255, 25)
    ellipse(circle_x, circle_y, circle_radius * 2, circle_radius * 2)

    fill(0, 0, 0)
    text("["+str(mouseX-width/2)+","+str(-(mouseY-height/2))+"]",
        mouseX-width/2+10, mouseY-height/2)
    text("[50,50]", 60, -50)

    fill(255, 0, 0)
```

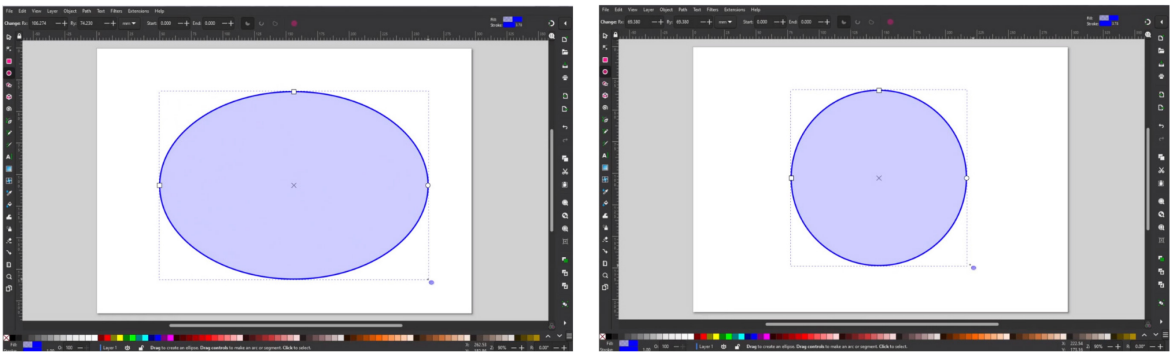
```
    text("Radius: {:.2f}".format(circle_radius), -width/2 + 20, -
        height/2 + 30)

    fill(0, 0, 255)
    text("Circumference: {:.2f}".format(circle_circumference), -
width/2
        + 20, -height/2 + 60)

    fill(0, 0, 255)
    text("Area: {:.2f}".format(circle_area), -width/2 + 20, -height/2 +
90)
    ...
```

Základní funkce zahrnuje výpočet vlastností kruhu na základě polohy kurzoru myši. Střed kruhu je pevně nastaven na (50, -50) a vzdálenost mezi středem a kurzorem myši určuje poloměr. Na základě poloměru se vypočítá obvod a plocha pomocí standardních geometrických vzorců.

Aby uživatel lépe pochopil vlastnosti kruhu, zobrazuje kód na plátně číselné hodnoty poloměru, obvodu a plochy. Barva textu těchto hodnot je sladěna s odpovídající vizuální reprezentací, aby mezi nimi vznikla jasná asociace. Kód navíc poskytuje aktualizace těchto vlastností v reálném čase při pohybu kurzoru myši, což uživatelům umožňuje dynamicky zkoumat a chápat geometrické vztahy, které se týkají kruhu.

9.2.2 Práce s nástrojem kruh a elipsa v Inkscape

Nástroj Kruh a elipsa v aplikaci Inkscape nabízí jednoduchý a efektivní způsob vytváření a úpravy kruhových a eliptických tvarů ve vektorových kresbách. Tutoriál se zaměřuje na tvorbu elipsy pomocí táhnutí myši a tvorbu kruhu při stisknuté klávese ***Ctrl*** nebo ***Shift***



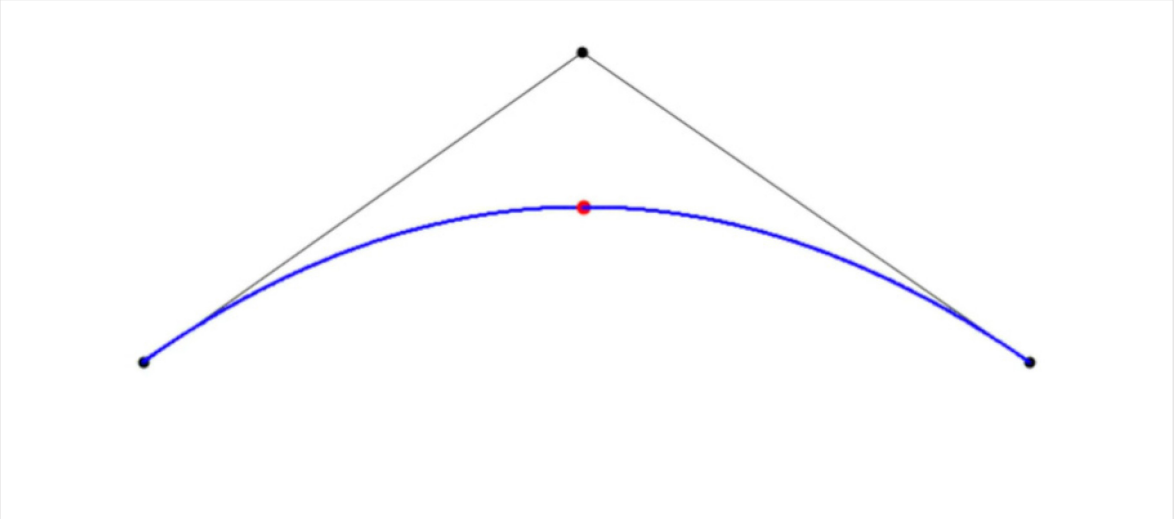
Obrázek 41: Náhled tutoriálu pro tvorbu kruhu a elipsy v Inkscape

9.3 Křivky

Křivky jsou nejuniverzálnějším objektem a s jejich znalostí lze vytvořit téměř jakýkoli tvar. Matematika, která stojí za vykreslováním křivek je o poznání složitější, než tomu tak bylo u úsečky nebo kruhu. Zároveň existuje více druhů křivek s odlišnými vlastnostmi, které se svou matematickou definicí liší. Tato práce se zaměřuje zejména na Bézierovy křivky a B-spline křivky, které patří mezi ty nejpoužívanější.

9.3.1 Tvorba Bézierovy křivky v Processing.py

V Processing.py byl vytvořen názorný příklad Bézierovy křivky s třemi řídícími body, přičemž prostřední řídící bod je vázaný na polohu myši. Křivku tak lze posunem myši ohýbat a sledovat její vlastnosti. Následně byly vytvořený příklady pro Bézierovu křivku o čtyřek a pěti řídících bodech.

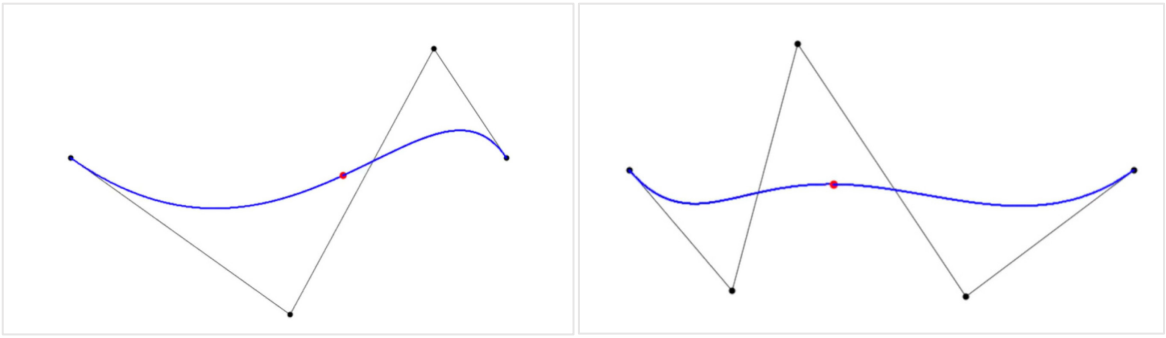


Obrázek 42: Výstup kódu Processing,py pro tvorbu kvadratické Bézierovy křivky

Kód – Processing.py

```
def setup():
    size(800, 500)
    noFill()
    strokeWeight(2)

    global P0, P2
    P0 = PVector(100, 250)
    P2 = PVector(700, 250)
```

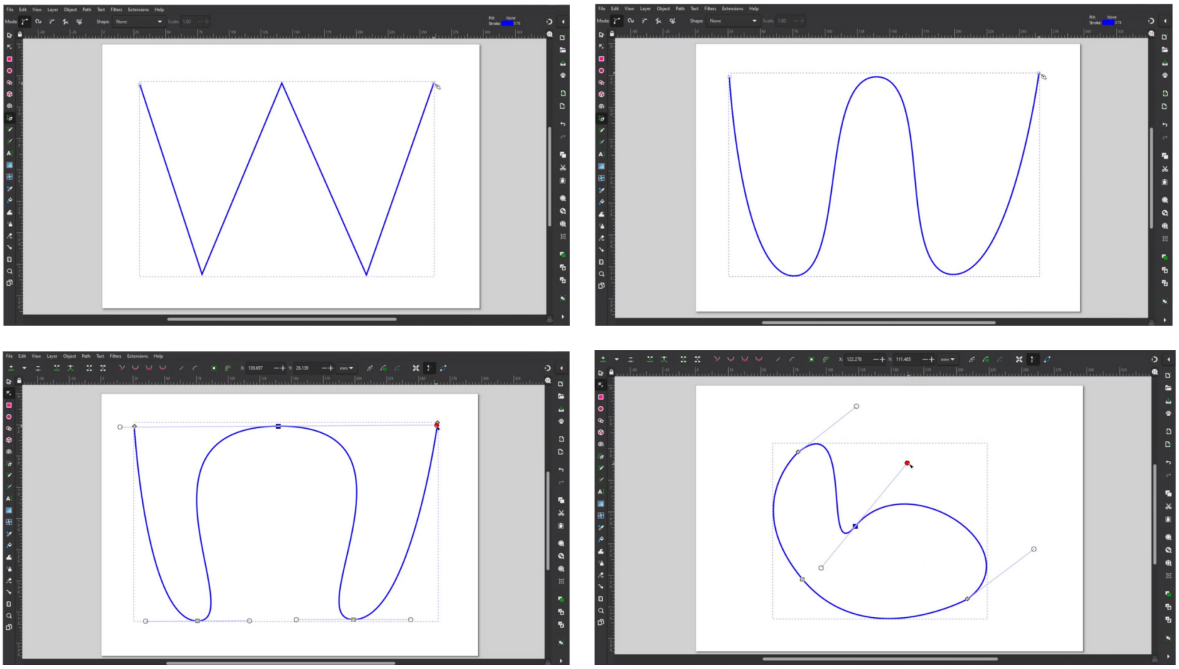
```
def draw():
    background(255)
    P1 = PVector(mouseX, mouseY)
    drawBezier(P1)

def drawBezier(P1):
    stroke(0)
    strokeWeight(8)
    point(P0.x, P0.y)
    point(P1.x, P1.y)
    point(P2.x, P2.y)

    stroke(0)
    strokeWeight(1)
    line(P0.x, P0.y, P1.x, P1.y)
    line(P1.x, P1.y, P2.x, P2.y)

    stroke(0, 0, 255)
    strokeWeight(2)
    for t in range(0, 1001):
        t = t / 1000.0
        x = ((1-t)**2) * P0.x + 2*t*(1-t) * P1.x + (t**2) * P2.x
        y = ((1-t)**2) * P0.y + 2*t*(1-t) * P1.y + (t**2) * P2.y
        if t == 0.5:
            stroke(255, 0, 0)
            fill(255, 0, 0)
            ellipse(x, y, 8, 8)
        if t == 0:
            x1, y1 = x, y
        else:
            stroke(0, 0, 255)
            line(x1, y1, x, y)
            x1, y1 = x, y
```



Obrázek 43: Výstup kódu Processing.py pro tvoru Bézierovy křivky se čtyřmi a pěti řídícím body

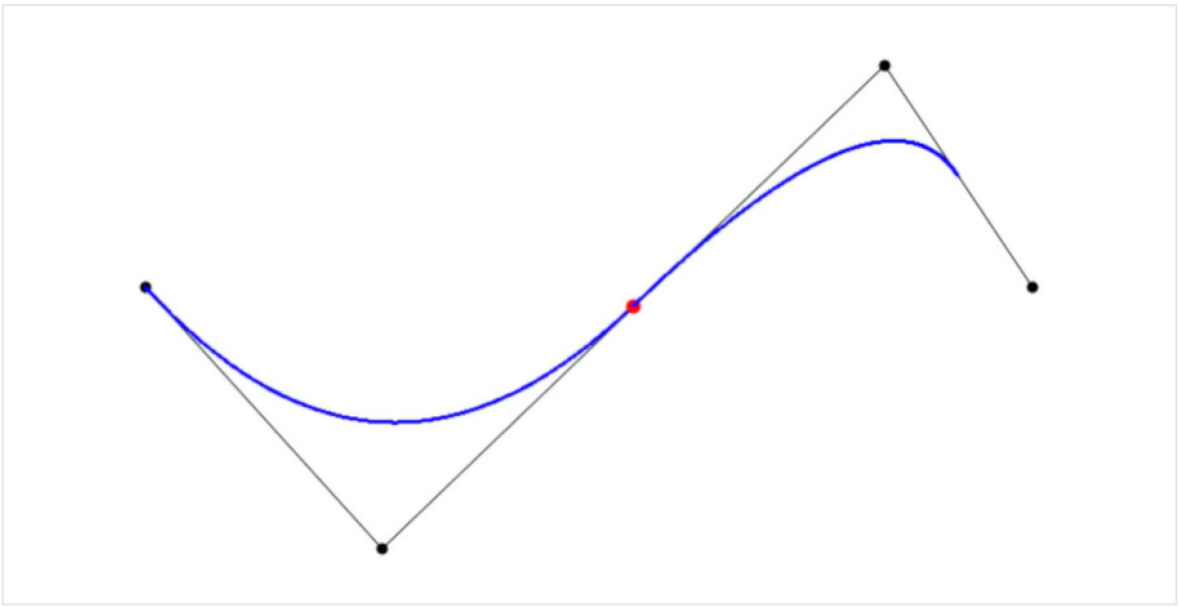
9.3.2 Práce s nástrojem Bézier v Inkscape

Nástroj Bézier v Inkscape umožňuje širokou škálu nastavení pro docílení vhodné křivky. Sestavený tutoriál se zaměřuje na její nejpodstatnější atributy, jako jsou kotevní body a jejich nastavení.



Obrázek 44: Náhled tutoriálu pro práci s nástrojem Bézier v Inkscpae

9.3.3 Tvorba B-Spline křivky v Processing.py

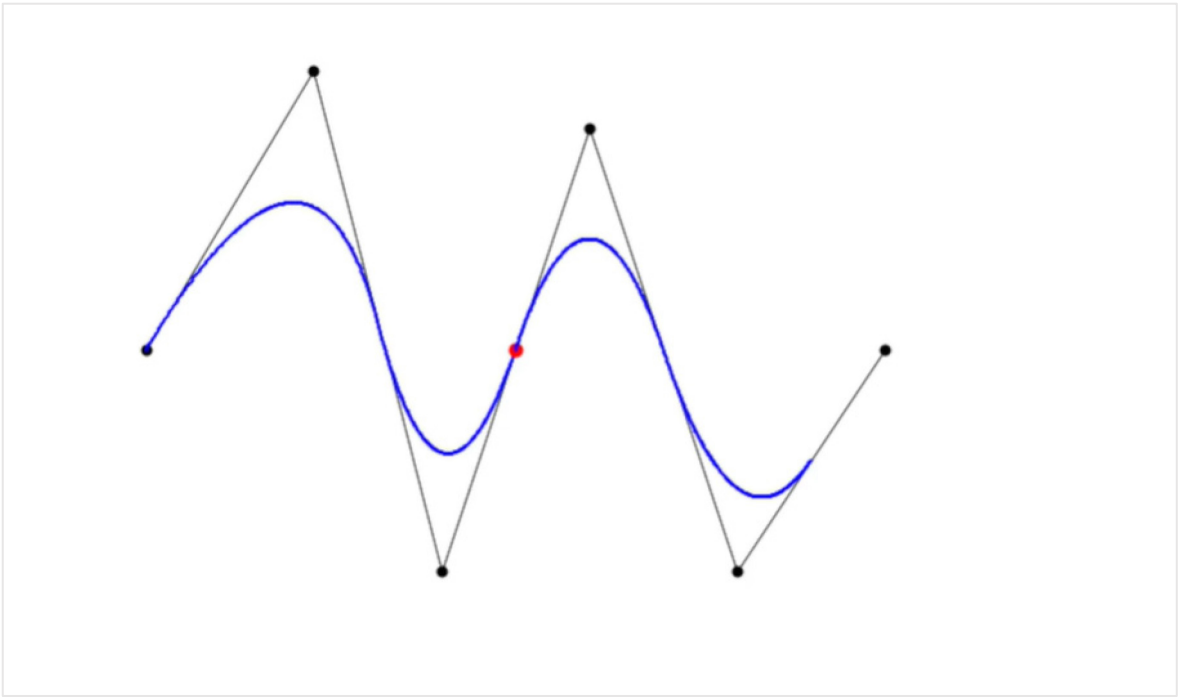


Obrázek 45: Kubická B-Spline křivka

Kód – Processing.py

```
def setup():
    size(800, 500)  # Nastavení velikosti okna na 800x500 pixelů
    noFill()  # Bez vyplnění tvarů
    strokeWeight(2)  # Tloušťka čáry

    global P0, P2, P3, P4
    # Definice kontrolních bodů
    P0 = PVector(100, 250)
    P2 = PVector(600, 100)
    P3 = PVector(700, 250)


def draw():
    background(255)  # Nastavení bílého pozadí
    P1 = PVector(mouseX, mouseY)  # Aktualizace polohy kontrolního
bodu P1 podle pozice myši
    drawBSpline(P1)  # Vykreslení B-spline křivky

# de Boor-Coxův algoritmus pro výpočet B-spline základní funkce
def cox_de_boor(t, k, degree, knots):
    if degree == 0:
        return 1 if knots[k] <= t < knots[k + 1] else 0

    alpha1 = (t - knots[k]) / (knots[k + degree] - knots[k]) if knots[k
+ degree] != knots[k] else 0
    alpha2 = (knots[k + degree + 1] - t) / (knots[k + degree + 1] -
knots[k + 1]) if knots[k + degree + 1] != knots[k + 1] else 0

    return cox_de_boor(t, k, degree - 1, knots) * alpha1 +
cox_de_boor(t, k + 1, degree - 1, knots) * alpha2


def drawBSpline(P1):
    # Vykreslení kontrolních bodů
    stroke(0)
    strokeWeight(8)
    point(P0.x, P0.y)
    point(P1.x, P1.y)
    point(P2.x, P2.y)
    point(P3.x, P3.y)

    # Vykreslení tenkých čar spojujících kontrolní body
    stroke(0)
    strokeWeight(1)
    line(P0.x, P0.y, P1.x, P1.y)
    line(P1.x, P1.y, P2.x, P2.y)
```

```
    line(P2.x, P2.y, P3.x, P3.y)

    # Vykreslení kvadratické B-Spline křivky
    degree = 2
    knots = [0, 0, 0, 1, 2, 3, 3,]
    control_points = [P0, P1, P2, P3]

    x1, y1 = P0.x, P0.y

    # Vykreslení kvadratické B-Spline křivky
    stroke(0, 0, 255)
    strokeWeight(2)
    for t in range(1, 1001):
        t = t / 1000.0 * (knots[-2]-1)
        S = PVector(0, 0)
        for i in range(len(control_points)):
            S += control_points[i] * cox_de_boor(t, i, degree, knots)
        stroke(0, 0, 255)
        line(x1, y1, S.x, S.y)
        x1, y1 = S.x, S.y
        if t == (0.5 * (knots[-2]-1)):
            # Výpočet a vykreslení bodu na křivce pro t = 0.5
            stroke(255, 0, 0)
            fill(255, 0, 0)
            ellipse(S.x, S.y, 8, 8)
```



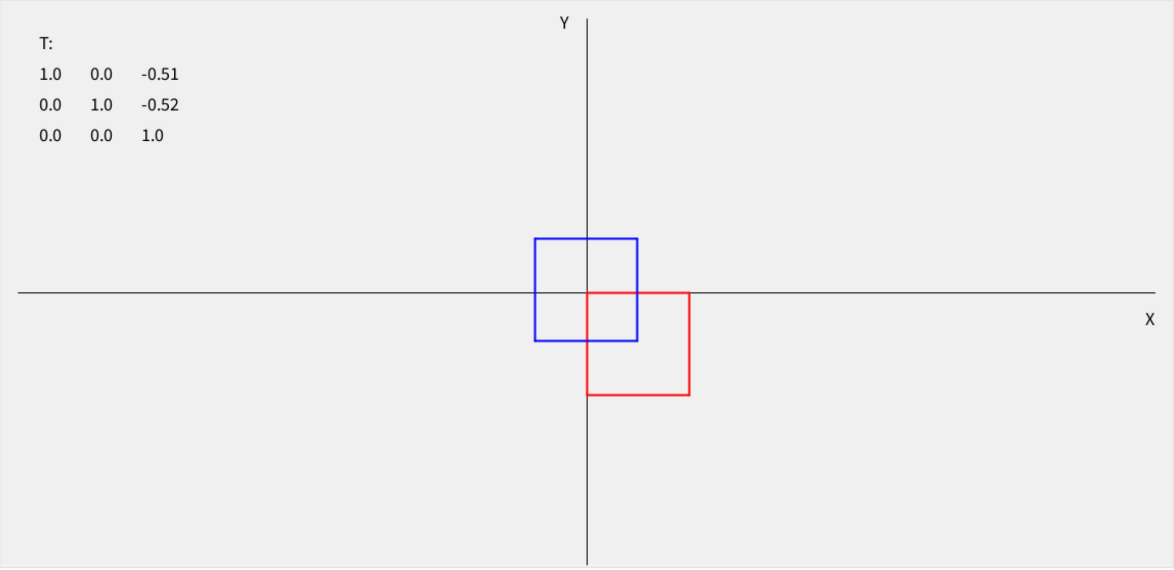
Obrázek 46: Výstup kódu Processing.py pro vykreslení B-Spline křivky s šesti řídicími body

10 GEOMETRICKÉ TRANSFORMACE

Pro vytvoření transformací byl nejprve použit Processing.py, ve kterém byly vytvořeny interaktivní skripty pro jednotlivé typy transformací. Jejich interaktivita spočívá v závislosti pozice kurzoru myši na hodnotách transformační matice. Tímto způsobem lze s objektem manipulovat. Na základě vytvořených scriptů v Processing.py byly vytvořeny pluginy pro Inkscape a Blender, které aplikují stejné transformační matice na reálné objekty.

10.1 Posun pomocí transformační matice

10.1.1 Posun v Processing.py



Obrázek 47: Posun pomocí transformační matice v Processing.py

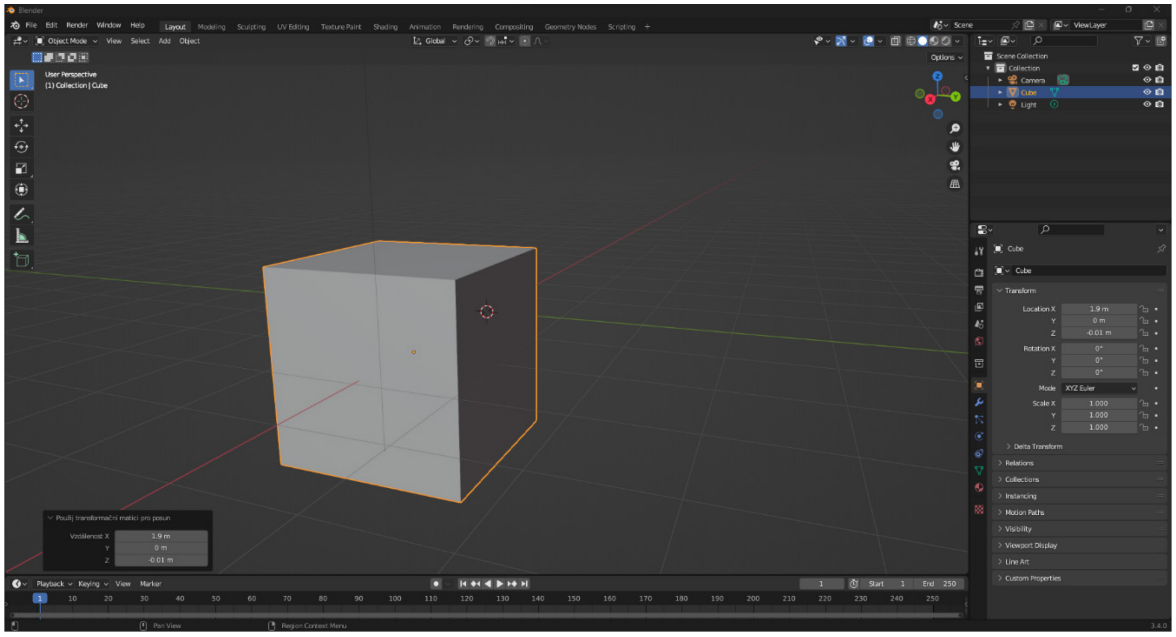
Kód – Processing.py

```
def setup():
    size(1153, 576)
    textSize(24)

def draw():
    drawBase()

    # Definice matice M
    M = [[1, 2],
         [3, 4]]

    # Definice hodnot posunu na základě polohy myši
    shift_x = map(mouseX, 0, width, -1, 1)
    shift_y = map(mouseY, 0, height, -1, 1)
```

```
    # Definice matice translace (posunu)
    T = [[1, 0, shift_x],
         [0, 1, shift_y],
         [0, 0, 1]]

    # Definice bodů, které představují původní matici
    points = [[0, 0], [0, 1], [1, 1], [1, 0]]

    # Použití transformační matice k posunu bodů
    points_shifted = []
    for point in points:
        x = point[0]
        y = point[1]
        x_shifted = T[0][0] * x + T[0][1] * y + T[0][2]
        y_shifted = T[1][0] * x + T[1][1] * y + T[1][2]
        points_shifted.append([x_shifted, y_shifted])

    # Nakreslení původní a posunuté matice jako souboru čar
    strokeWeight(2)
    stroke(255, 0, 0) # červená pro původní matici
    draw_matrix(points, height / 2)
    stroke(0, 0, 255) # modrá pro posunutou matici
    draw_matrix(points_shifted, height / 2)

    # Vypsání původní a posunuté matice
    text("T:",40, 50)
    display_rotation_matrix(T, 40, 50)

# Nakreslení matice jako množiny čar pomocí seznamu bodů
def draw_matrix(points, y_offset=0):
    n = len(points)
    for i in range(n):
        x1 = points[i][0] * 100 + width / 2
        y1 = points[i][1] * 100 + y_offset
        x2 = points[(i + 1) % n][0] * 100 + width / 2
        y2 = points[(i + 1) % n][1] * 100 + y_offset
        line(x1, y1, x2, y2)

def display_rotation_matrix(matrix, x_offset, y_offset):
    textSize(18)
    fill(0)
    for i in range(3):
        for j in range(3):
            matrix_value = round(matrix[i][j], 2)  # Zaokrouhlení
hodnoty na 2 desetinná místa
            text(str(matrix_value), x_offset + j * 50, y_offset + (i +
1) * 30)
```

```
def drawBase():
    strokeWeight(1)
    background(240)                 #nastavení barvy pozadí na šedou
    stroke(0)                       #nastavení barvy čar na černou
    pushMatrix()                    #nový souřadnicový systém
    translate(width/2, height/2)    #posunout souř. systém na střed
    line(0,0,(width/2-20),0)
    line(0,0,0,(-height/2+20))
    line(0,0,(-width/2+20),0)
    line(0,0,0,(height/2-20))
    popMatrix()                     #návrat k defaultnímu souř.
systému
    fill(0)                         #nastavení barvy textu na černou
    text("X", 1123, 320)
    text("Y", 550, 30)
```

10.1.2 Plugin posunu pro Blender



Obrázek 48: Použití vlastního pluginu pro posun objektu v Blenderu

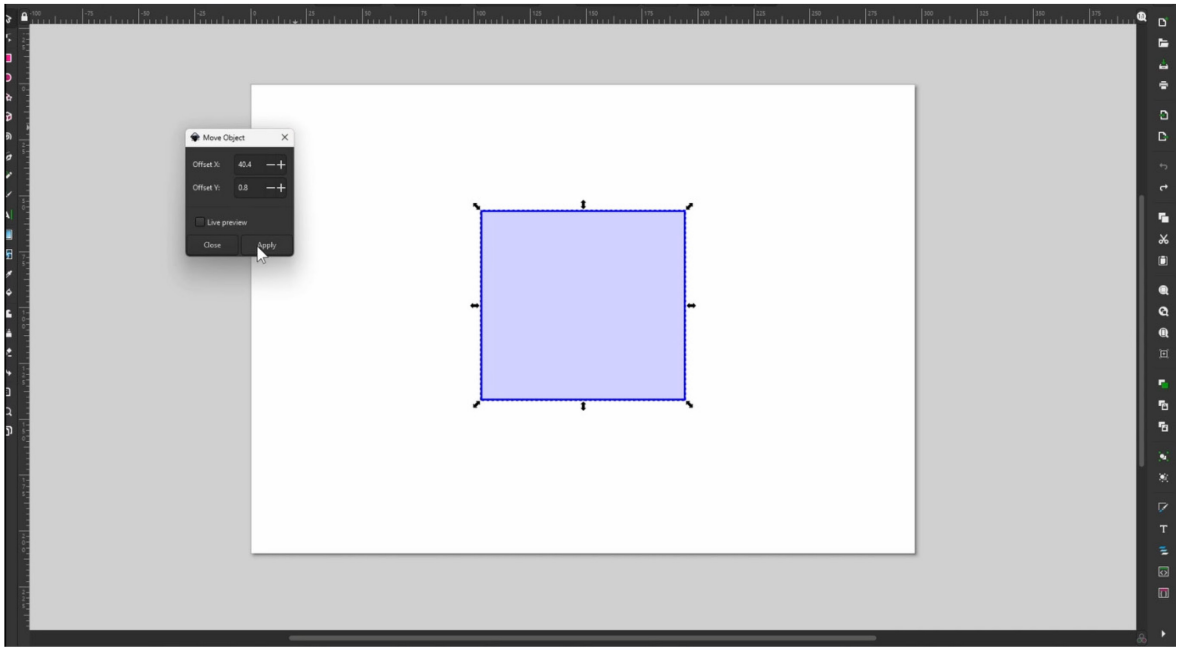
Kód – Plugin pro Blender

```
# Informace o doplňku
bl_info = {
    "name": "Použij transformační matici pro posun",
    "author": "Your Name",
    "version": (1, 0),
    "blender": (2, 93, 0),
    "location": "View3D > Object",
    "description": "Posun objektu o zadanou vzdálenost pomocí
transformační matice",
```

```
    "warning": "",
    "doc_url": "",
    "category": "Object",
}

import bpy
import mathutils
from bpy.props import FloatVectorProperty

# Definice operátoru pro posun pomocí transformační matice
class OBJECT_OT_transform_matrix_move(bpy.types.Operator):
    bl_idname = "object.transform_matrix_move"
    bl_label = "Použij transformační matici pro posun"
    bl_options = {'REGISTER', 'UNDO'}

    # Vlastnosti operátoru - vzdálenost posunu
    distance: FloatVectorProperty(
        name="Vzdálenost",
        default=(0.0, 0.0, 0.0),
        subtype='TRANSLATION',
        unit='LENGTH',
    )

    # Hlavní funkce operátoru
    def execute(self, context):
        obj = context.active_object
        # Vytvoření transformační matice pro posun
        matrix_translation =
mathutils.Matrix.Translation(self.distance)
        # Posun objektu pomocí transformační matice
        obj.matrix_world = matrix_translation @ obj.matrix_world
        return {'FINISHED'}

# Definice panelu pro ovládání posunu
class VIEW3D_PT_transform_matrix_move(bpy.types.Panel):
    bl_label = "Použij transformační matici pro posun"
    bl_idname = "OBJECT_PT_transform_matrix_move"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = "Object"

    # Funkce pro vykreslení panelu
    def draw(self, context):
        layout = self.layout
        layout.operator(OBJECT_OT_transform_matrix_move.bl_idname)

# Funkce pro přidání operátoru do menu
def menu_func(self, context):
    self.layout.operator(OBJECT_OT_transform_matrix_move.bl_idname
)
```

```
# Funkce pro registraci tříd a přidání položky do menu
def register():
    bpy.utils.register_class(OBJECT_OT_transform_matrix_move)
    bpy.utils.register_class(VIEW3D_PT_transform_matrix_move)
    bpy.types.VIEW3D_MT_object.append(menu_func)

# Funkce pro odregistraci tříd a odebrání položky z menu
def unregister():
    bpy.utils.unregister_class(OBJECT_OT_transform_matrix_move)
    bpy.utils.unregister_class(VIEW3D_PT_transform_matrix_move)
    bpy.types.VIEW3D_MT_object.remove(menu_func)

# Registrace doplňku při spuštění jako samostatného skriptu
if __name__ == "__main__":
    register()
```

10.1.3 Plugin posunu pro Inkscape



Obrázek 49: Použití vlastního pluginu pro posun objektu v Inkscape

Kód – Plugin pro Inkscape .inx

```
<?xml version="1.0" encoding="UTF-8"?>
<inkscape-extension
xmlns="http://www.inkscape.org/namespace/inkscape/extension">
    <name>Move Object</name>
    <id>org.inkscape.effect.move_object</id>
```

```
    <param name="offset_x" type="float" min="-10000" max="10000"
gui-text="Offset X:" gui-description="Move object horizontally (X-
axis)">0</param>
    <param name="offset_y" type="float" min="-10000" max="10000"
gui-text="Offset Y:" gui-description="Move object vertically (Y-
axis)">0</param>

    <effect>
        <object-type>all</object-type>
        <effects-menu>
            <submenu name="@Test"/>
        </effects-menu>
    </effect>

    <script>
        <command location="inx"
interpreter="python">move_object.py</command>
    </script>
</inkscape-extension>
```

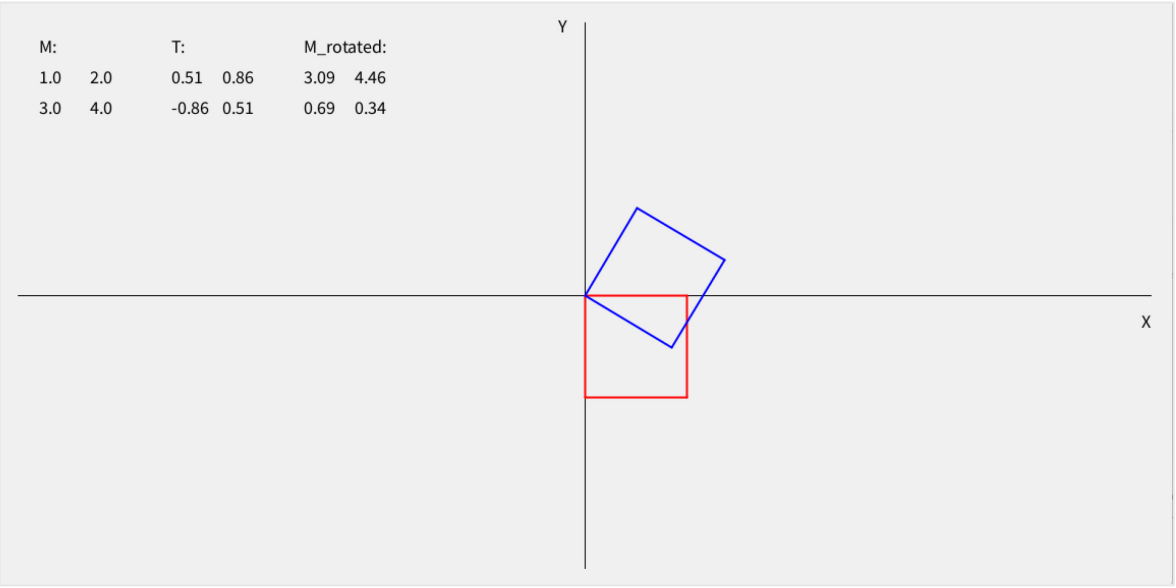
Kód – Plugin pro Inkscape .py

```
#!/usr/bin/env python
import inkex
from lxml import etree

class MoveObject(inkex.EffectExtension):

    def add_arguments(self, pars):
        pars.add_argument("--offset_x", type=float, default=0.0,
help="Offset X")
        pars.add_argument("--offset_y", type=float, default=0.0,
help="Offset Y")
    def effect(self):
        offset_x = self.svg.unittouu(str(self.options.offset_x) +
self.svg.unit)
        offset_y = self.svg.unittouu(str(self.options.offset_y) +
self.svg.unit)
```

```
        for node in self.svg.selected.values():
            transform = node.get('transform')
            translate_transform =
'translate({},{})'.format(offset_x, offset_y)
            if transform:
                node.set('transform', transform + ' ' +
translate_transform)
            else:
                node.set('transform', translate_transform)

if __name__ == '__main__':
    MoveObject().run()
```

1.2 Rotace pomocí transformační matice

10.1.4 Rotace v Processing.py



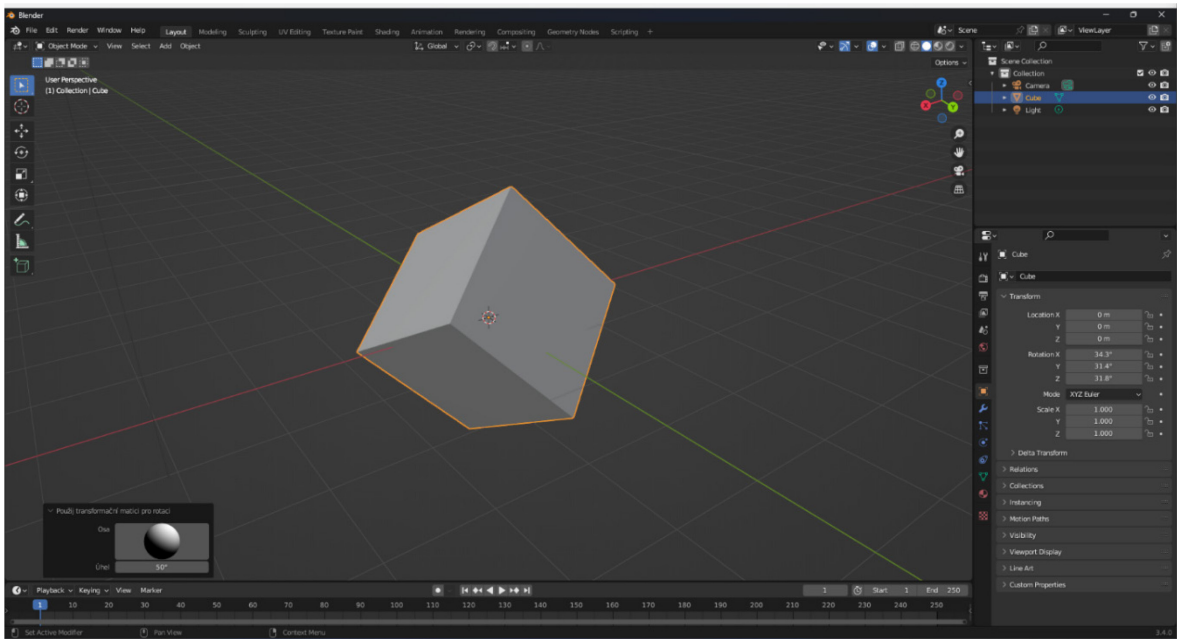
Obrázek 50: Rotace pomocí transformační matice v Processing.py

Kód – Processing.py

```
    # Definice hodnot rotace na základě polohy myši
    c = math.cos(theta_rad)
    s = math.sin(theta_rad)
    R = [[c, -s],
         [s, c]]

    # Definice matice translace (rotace)
```

```
    T = [[c, -s, 0],
         [s, c, 0],
         [0, 0, 1]]
```

10.1.5 Plugin rotace pro Blender



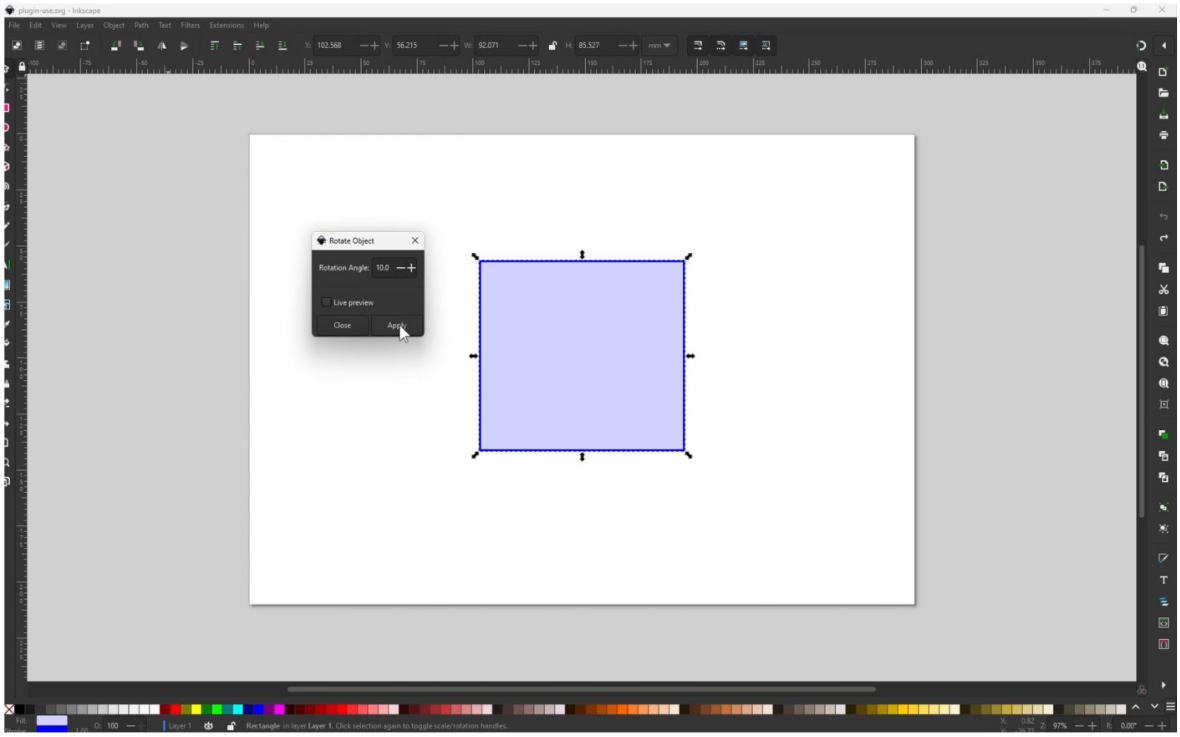
Obrázek 51: Použití vlastního pluginu pro rotaci objektu v Blenderu

Kód – Plugin pro Blender

```
# Definice operátoru pro rotaci pomocí transformační matice
class OBJECT_OT_transform_matrix_rotate(bpy.types.Operator):
    bl_idname = "object.transform_matrix_rotate"
    bl_label = "Použij transformační matici pro rotaci"
    bl_options = {'REGISTER', 'UNDO'}

    # Vlastnosti operátoru - osa rotace a úhel
    axis: FloatVectorProperty(
        name="Osa",
        default=(0.0, 0.0, 1.0),
        subtype='DIRECTION',
    )

    angle: FloatProperty(
        name="Úhel",
        default=math.radians(45),
        subtype='ANGLE',
        unit='ROTATION',
    )

    # Hlavní funkce operátoru
```

```
    def execute(self, context):
        obj = context.active_object
        # Vytvoření transformační matice pro rotaci
        matrix_rotation = mathutils.Matrix.Rotation(self.angle, 4,
self.axis)
        # Rotace objektu pomocí transformační matice
        obj.matrix_world = matrix_rotation @ obj.matrix_world
        return {'FINISHED'}
```

10.1.6 Plugin rotace pro Inkscape



Obrázek 52: Použití vlastního pluginu pro rotaci objektu v Inkscape

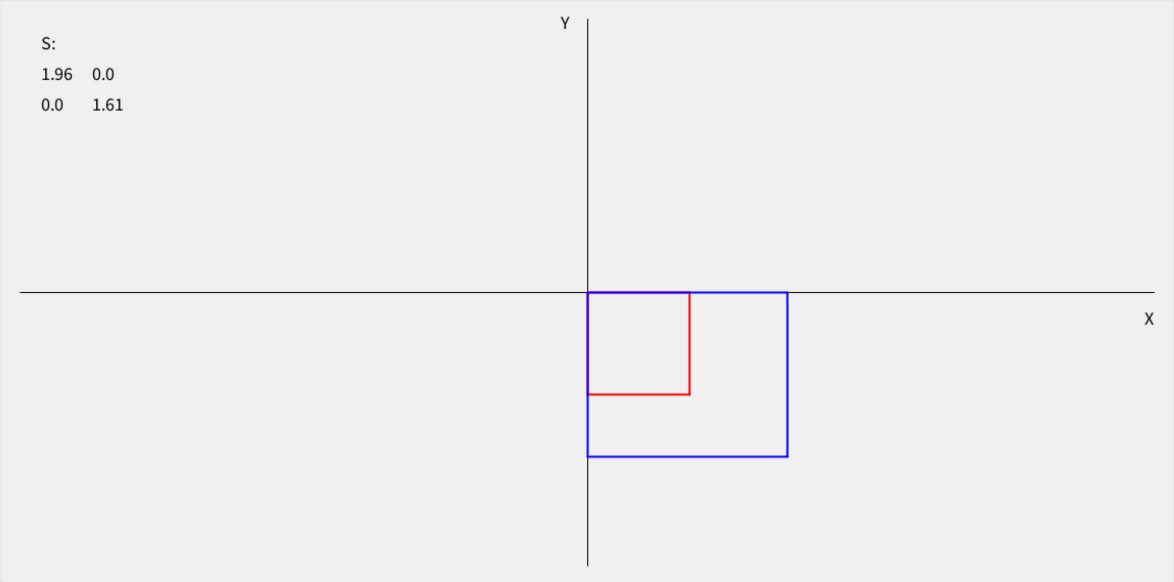
Kód – Plugin pro Inkscape .py

```
class RotateObject(inkex.EffectExtension):

    def add_arguments(self, pars):
        pars.add_argument("--rotation_angle", type=float,
default=0.0, help="Rotation Angle")

    def effect(self):
        angle_degrees = self.options.rotation_angle
        angle_radians = math.radians(angle_degrees)

        cos_theta = math.cos(angle_radians)
        sin_theta = math.sin(angle_radians)
```

```
        for node in self.svg.selected.values():
            transform = node.get('transform')
            matrix_transform = 'matrix({:.6f}, {:.6f}, {:.6f},
{:.6f}, 0, 0)'.format(cos_theta, -sin_theta, sin_theta, cos_theta)
            if transform:
                node.set('transform', transform + ' ' +
matrix_transform)
            else:
                node.set('transform', matrix_transform)
```

10.2 Změna měřítka pomocí transformační matice

10.2.1 Změna měřítka v Processing.py

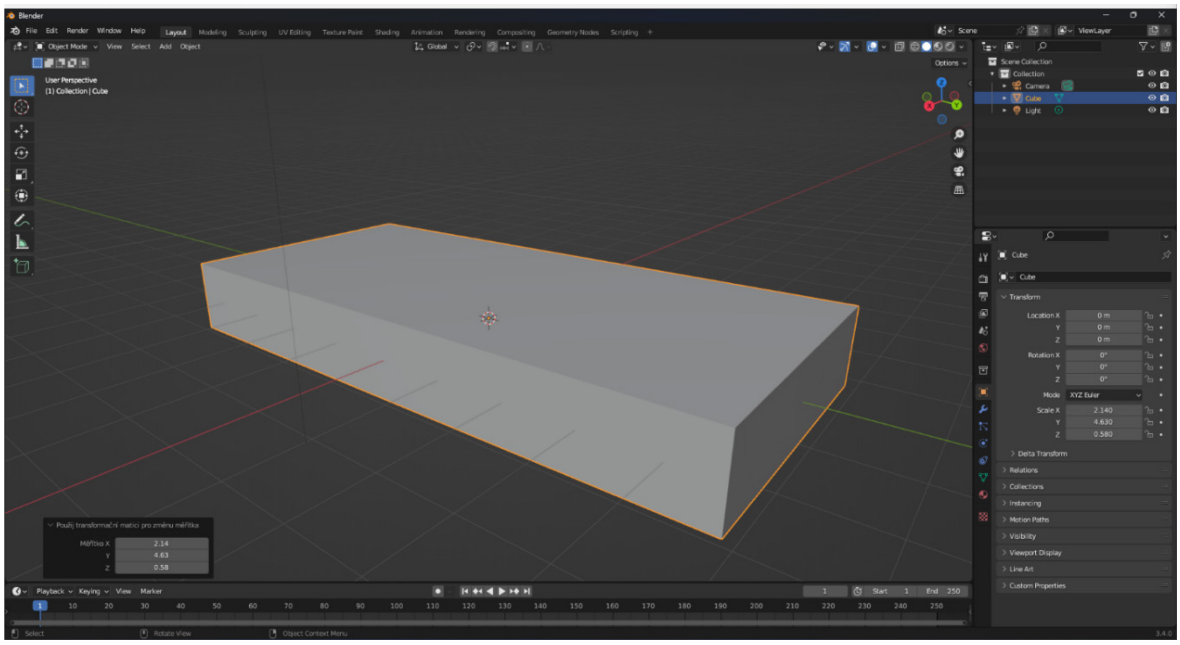


Obrázek 53: Změna měřítka pomocí transformační matice v Processing.py

Kód – Processing.py

```
    # Definice hodnot změny měřítka na základě polohy myši
    scale_x = map(mouseX, 0, width, 0.5, 2)
    scale_y = map(mouseY, 0, height, 0.5, 2)

    # Definice matice translace (změna měřítka)
    S = [[scale_x, 0, 0],
         [0, scale_y, 0],
         [0, 0, 1]]
```

10.2.2 Plugin změny měřítka pro Blender

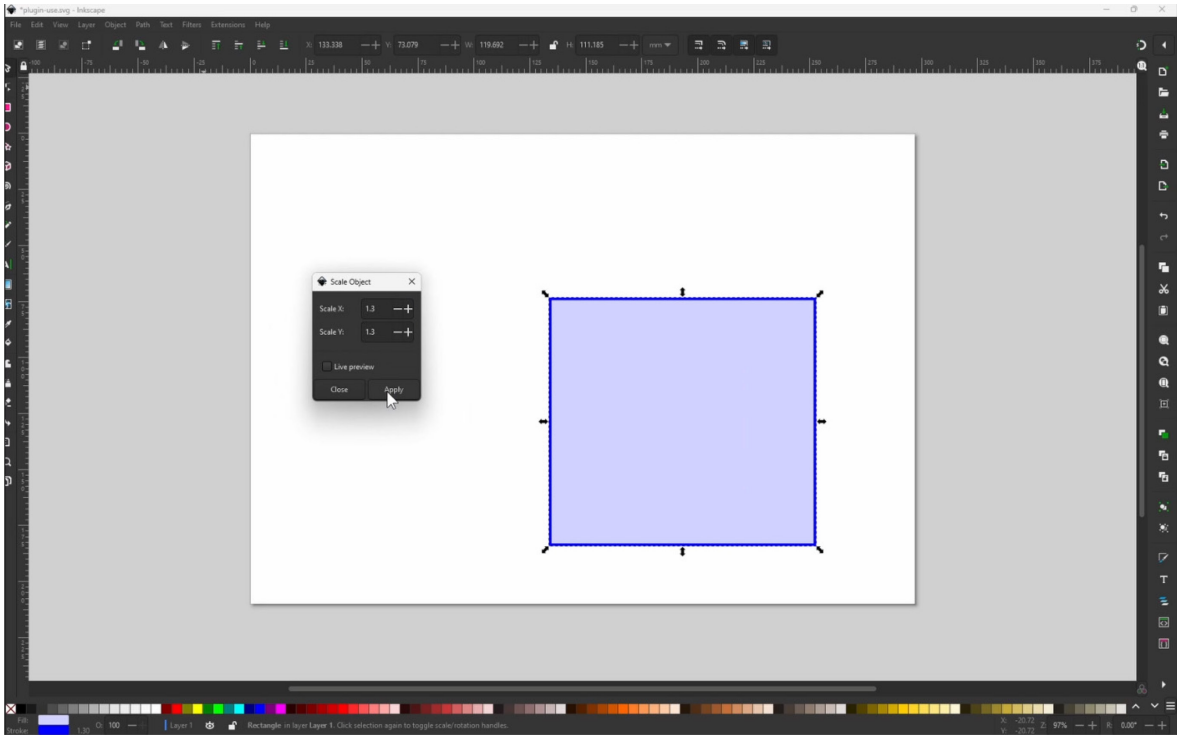


Obrázek 54: Použití vlastního pluginu pro změnu měřítka objektu v Blenderu

Kód – Plugin pro Blender

```
# Definice operátoru pro změnu měřítka pomocí transformační matice
class OBJECT_OT_transform_matrix_scale(bpy.types.Operator):
    bl_idname = "object.transform_matrix_scale"
    bl_label = "Použij transformační matici pro změnu měřítka"
    bl_options = {'REGISTER', 'UNDO'}

    # Vlastnosti operátoru - měřítko
    scale: FloatVectorProperty(
        name="Měřítko",
        default=(1.0, 1.0, 1.0),
        subtype='XYZ',
    )

    # Hlavní funkce operátoru
    def execute(self, context):
        obj = context.active_object
        # Vytvoření transformační matice pro změnu měřítka
        matrix_scale = mathutils.Matrix.Diagonal((*self.scale,
1.0)).to_4x4()
        # Změna měřítka objektu pomocí transformační matice
        obj.matrix_world = matrix_scale @ obj.matrix_world
        return {'FINISHED'}
```

10.2.3 Plugin změny měřítka pro Inkscape



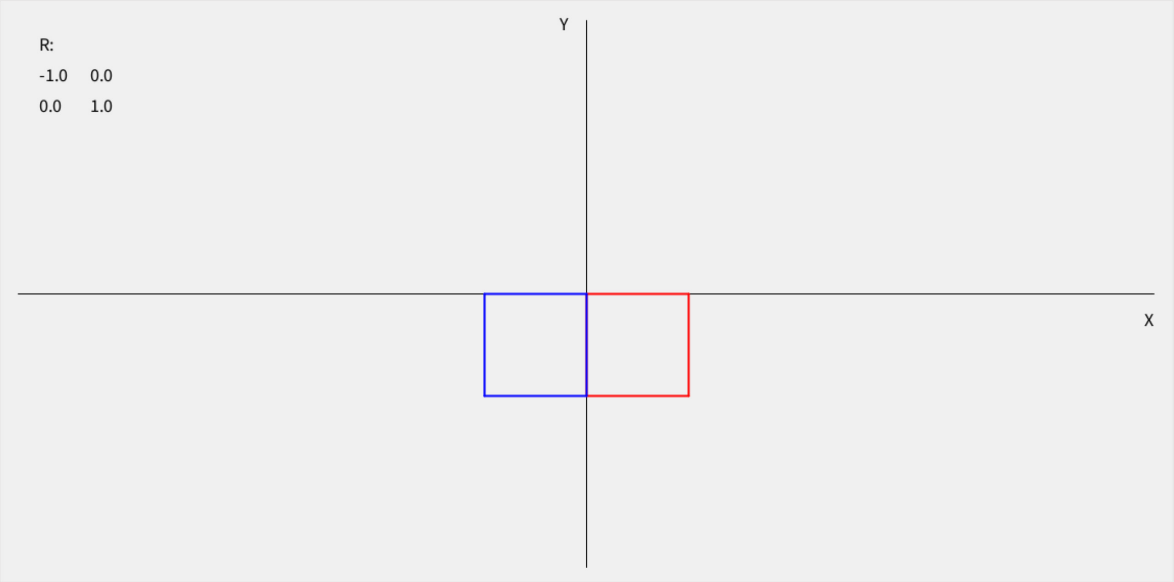
Obrázek 55: Použití vlastního pluginu pro změnu měřítka objektu v Blenderu

Kód – Plugin pro Inkscape .py

```
class ScaleObject(inkex.EffectExtension):

    def add_arguments(self, pars):
        pars.add_argument("--scale_x", type=float, default=1.0,
help="Scale X")
        pars.add_argument("--scale_y", type=float, default=1.0,
help="Scale Y")

    def effect(self):
        scale_x = self.options.scale_x
        scale_y = self.options.scale_y

        for node in self.svg.selected.values():
            transform = node.get('transform')
            matrix_transform = 'matrix({:.6f}, 0, 0, {:.6f}, 0,
0)'.format(scale_x, scale_y)
            if transform:
                node.set('transform', transform + ' ' +
matrix_transform)
            else:
                node.set('transform', matrix_transform)
```

10.3 Zrcadlení pomocí transformační matice

10.3.1 Zrcadlení v Processing.py



Obrázek 56: Zrcadlení pomocí transformační matice v Processing.py

Kód – Processing.py

```
 # Definice hodnot změny měřítka na základě polohy myši
reflection_x = 1 if mouseX < width / 2 else -1
reflection_y = 1 if mouseY < height / 2 else -1

 # Definice matice translace (změna měřítka)
R = [[reflection_x, 0, 0],
     [0, reflection_y, 0],
     [0, 0, 1]]
```

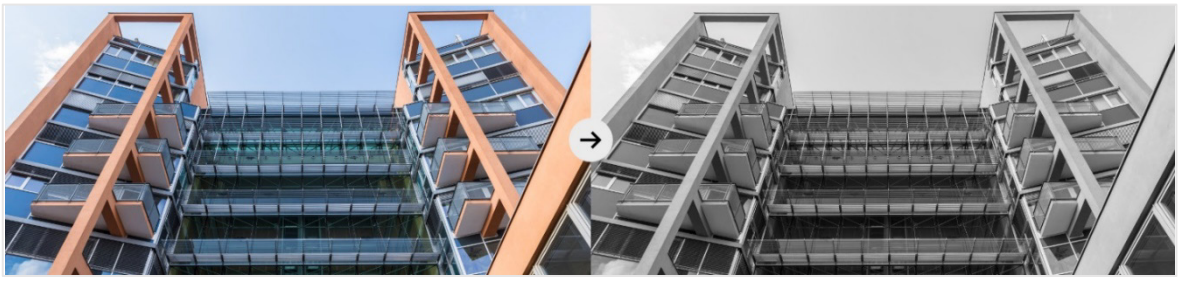
11 GRAFICKÉ ÚPRAVY

Různé typy grafických filtrů byly zrekonstruovány pomocí Processing.py postupným procházením nad každým pixelem a změnou jeho hodnoty. Kapitola je rozdělena na základní grafické filtry a konvoluční filtry. Základní filtry fungují na principu přepočítání pixelů podle vzorce, zatímco konvoluční filtry využívají konvoluční matice, které počítá vážený průměr dle vah určené maticí.

11.1 Základní grafické filtry

11.1.1 Odstíny šedé

Konverze do odstínů šedi je fundamentální proces v oblasti digitálního zpracování obrazu. Aby bylo možné docílit černobílého výstupu, je třeba eliminovat vstupní informace o odstínech a sytosti barev při zachování hodnot jasu.



Obrázek 57: Konverze do odstínů šedi pomocí Processing.py

V této funkci je použita iterace nad každým pixelem daného vstupu pomocí vnořených smyček for. Funkce ***get*()** nejprve zjistí barvu pixelu na pozici (x, y) a uloží ji do proměnné ***c***. Poté zprůměrováním červené, zelené a modré složky barvy lze získat hodnotu ve stupních šedi. Pomocí funkce ***set*()** dojde k přepsání původní hodnoty pixelu na odstín šedé. Po provedení všech iterací dojde k vykreslení černobílého výstupu pomocí funkce ***image*()**.

Kód – Processing.py

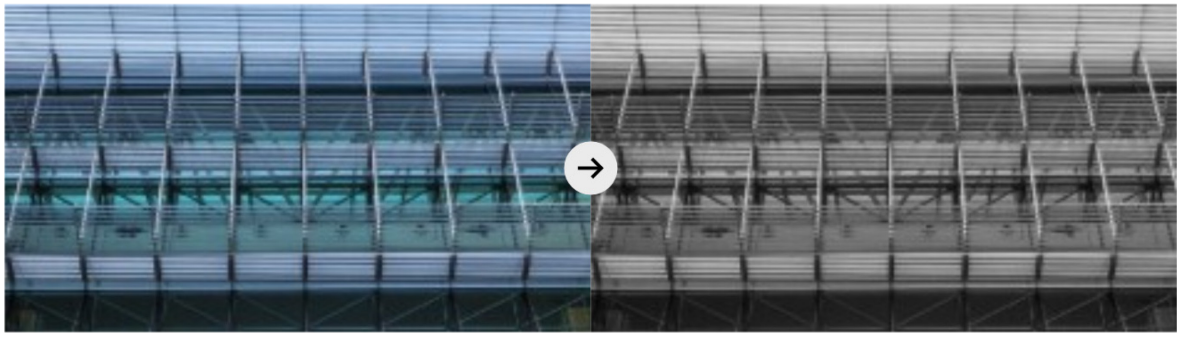
```
def setup():
  global img
  size(800,368)
  img = loadImage("photo.jpg")
  image(img, 0, 0)

def draw():
    for x in range(img.width):
        for y in range(img.height):
```

```
            c = img.get(x, y)
            gray = (red(c) + green(c) + blue(c)) / 3
            img.set(x, y, color(gray))
    image(img, 0, 0)
    save("export.png")
```

Výsledná funkce je již vhodná pro praktické použití, nicméně je dobré při průměrování barevných složek pixelu brát v potaz vlastnost lidského oka a každou složku vynásobit koeficientem odrážejícím množství rozpoznatelných odstínů konkrétní barvy.

```
gray = (0.299*red(c) + 0.587*green(c) + 0.114*blue(c))
```

Plugin efektu šedi pro GIMP



Obrázek 58: Výstup vlastního pluginu pro konverzi obrazu do odstínů šedi v aplikaci GIMP

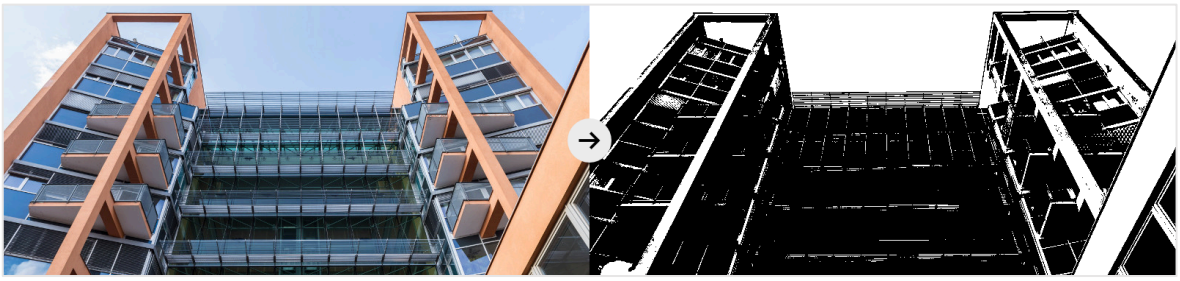
Kód – Plugin pro GIMP

```
from gimpfu import *

def python_grayscale_with_progress(image, drawable):
    width, height = drawable.width, drawable.height
    total_pixels = float(width * height)
    progress_step = 100.0 / total_pixels
    processed_pixels = 0

    pdb.gimp_image_undo_group_start(image)

    try:
        for y in range(height):
            for x in range(width):
                pixel = pdb.gimp_drawable_get_pixel(drawable, x,
y)[1]
                r, g, b = pixel[0], pixel[1], pixel[2]
                gray = int((r + g + b) / 3)
```

```
                pdb.gimp_drawable_set_pixel(drawable, x, y, 3,
(gray, gray, gray))

                processed_pixels += 1
                if processed_pixels % 100 == 0:
                    gimp.progress_update(processed_pixels /
total_pixels)
            gimp.displays_flush()

    finally:
        pdb.gimp_image_undo_group_end(image)
        pdb.gimp_drawable_update(drawable, 0, 0, width, height)
        gimp.displays_flush()

register(
    "python_fu_grayscale_with_progress",
    "Grayscale filter",
    "Converts image to grayscale and displays a progress bar",
    "Your Name",
    "Your Name",
    "2023",
    "/Filters/Custom/Grayscale filter",
    "RGB*, GRAY*",
    [],
    [],
    python_grayscale_with_progress)

main()
```

11.1.2 Prahování

Nejjednodušší příklad prahování porovnává pouze jednotlivé pixely s předem určenou hodnotou barevné intenzity. Následující kód je ideální ukázkou toho, jak lze pomocí jazyka Python a knihovny Processing takové prahování realizovat.



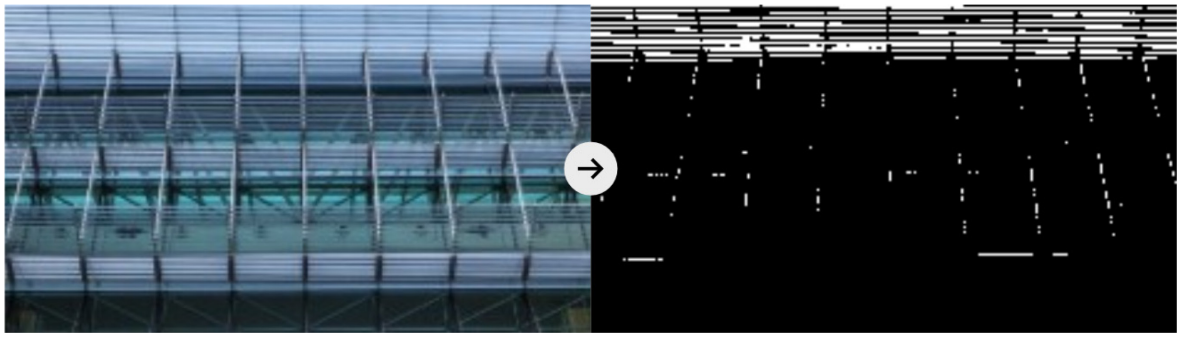
Obrázek 59: Binární prahování pomocí Processing.py

Kód nejprve převede obraz na stupně šedi a následně klasifikuje každý pixel jako černý nebo bílý na základě jeho hodnoty jasu. Kód používá funkce ***loadImage***() a ***createImage***() z

knihovny Processing k načtení obrázku a vytvoření nového výstupního obrázku. Poté projde ve smyčce všechny pixely vstupního obrázku a použije operaci prahování tak, že vypočítá hodnotu jasu každého pixelu a porovná ji s prahovou hodnotou. Výstupní obrázek se pak zobrazí na obrazovce pomocí funkce ***image*()** a uloží jako soubor PNG pomocí funkce ***save*()**.[34]

Kód – Processing.py

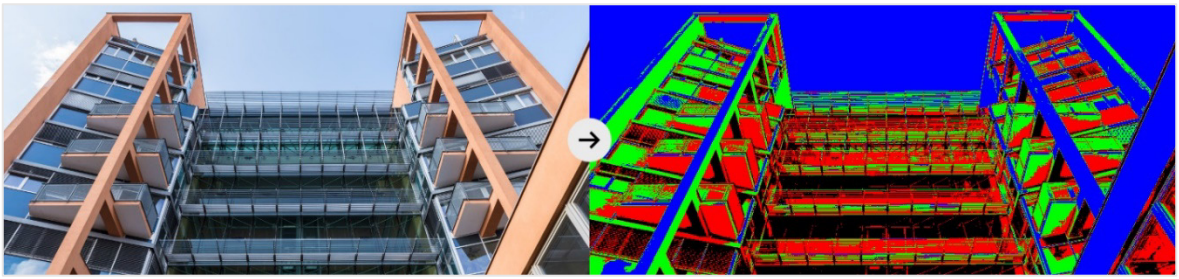
```
def draw():
    threshold_value = 180
    threshold_image = createImage(img.width, img.height, RGB)
    threshold_image.loadPixels()
    img.loadPixels()
    for x in range(img.width):
        for y in range(img.height):
            loc = x + y * img.width
            r = red(img.pixels[loc])
            g = green(img.pixels[loc])
            b = blue(img.pixels[loc])
            brightness_val = (r + g + b) / 3
            if brightness_val > threshold_value:
                threshold_image.pixels[loc] = color(255)
            else:
                threshold_image.pixels[loc] = color(0)
    threshold_image.updatePixels()
    image(threshold_image, 0, 0)
    save("export.png")
```



Obrázek 60: Výstup vlastního pluginu prahování pro GIMP

Může se stát, že rozdělení obrazu na černou a bílou nemusí plně dostačovat. Pomocí ne příliš zásadní modifikace původního kódu lze pixely rozdělit do více kategorií. Do kódu je nejprve přidána informace o dalších prahových hodnotách a ve for cyklu je s nimi následně

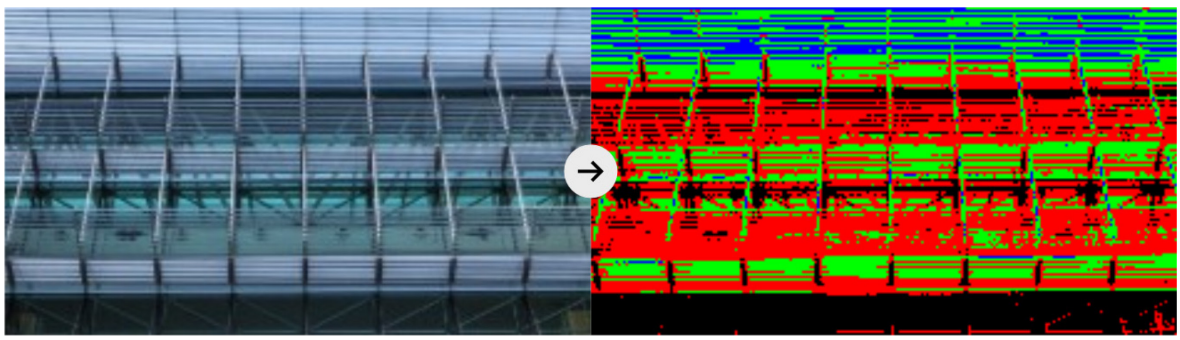
pracováno pomocí logiky if. Výsledný for cyklus, který rozdělí pixely na černou, červenou, zelenou a modrou by poté mohl vypadat následovně:



Obrázek 61: Vícebarevné prahování pomocí Processing.py

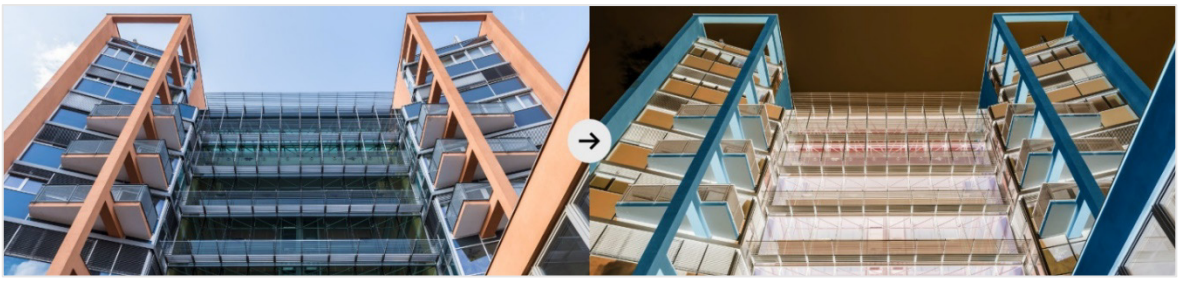
Kód – Processing.py

```
thresholds = [60, 120, 180]
for x in range(img.width):
    for y in range(img.height):
        loc = x + y * img.width
        if brightness(gray_img.pixels[loc]) < thresholds[0]:
            threshold_img.pixels[loc] = color(0)
        elif brightness(gray_img.pixels[loc]) < thresholds[1]:
            threshold_img.pixels[loc] = color(255, 0, 0)
        elif brightness(gray_img.pixels[loc]) < thresholds[2]:
            threshold_img.pixels[loc] = color(0, 255, 0)
        else:
            threshold_img.pixels[loc] = color(0, 0, 255)
threshold_img.updatePixels()
image(threshold_img, 0, 0)
save("export.png")
```



Obrázek 62: Výstup vlastního pluginu vícebarevného prahování pro GIMP

11.1.3 Negativ

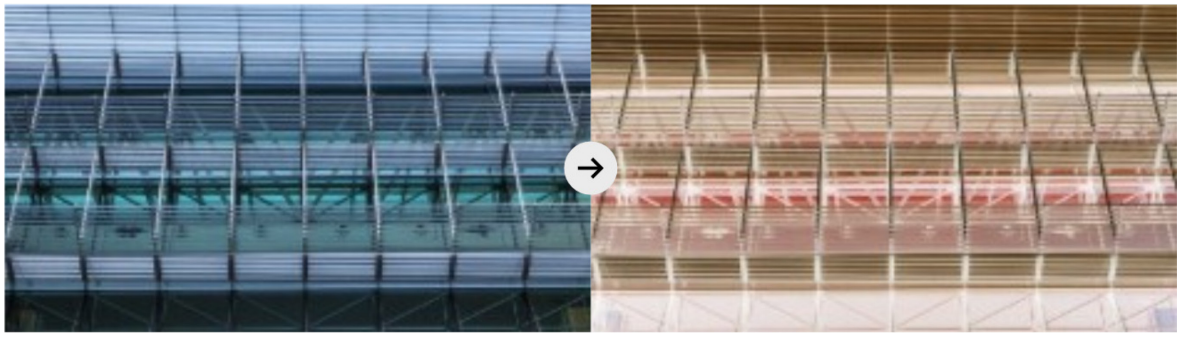


Obrázek 63: Negativ pomocí Processing.py

Proces úpravy obrazu v jeho negativní formu zahrnuje inverzi barev a hodnot jasu vstupních dat. Nejprve je třeba vytvořit prázdný obrázek se stejnými rozměry jako mají vstupní data a následně opět aplikovat princip s vnořenou smyčkou, díky které lze pracovat s jednotlivými pixely obrazu. V každé iteraci jsou brány v potaz všechny tři barevné složky pixelu, kterým je převrácena jejich hodnota odečtením od maxima. Tímto vznikne pixel s negativní barvou i světlostí. [34]

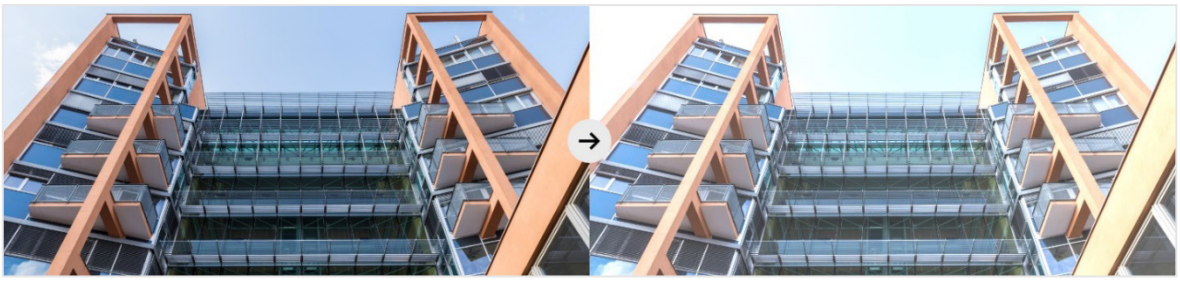
Kód – Processing.py

```
def draw():
    negative_img = createImage(img.width, img.height, RGB)
    negative_img.loadPixels()
    for x in range(img.width):
        for y in range(img.height):
            pixel_color = img.get(x, y)
            inverted_color = color(255 - red(pixel_color), 255 -
                green(pixel_color), 255 - blue(pixel_color))
            negative_img.set(x, y, inverted_color)
    negative_img.updatePixels()
    image(negative_img, 0, 0)
    save("export.png")
```



Obrázek 64: Výstup vlastního pluginu pro efekt negativu pro GIMP

11.1.4 Jas



Obrázek 65: Filtr jasu pomocí Processing.py

V tomto příkladě je názorně ukázáno, že funkci ***draw()*** není nutné vždy používat. Místo ní lze použít vlastní funkci ***brighten()***, ve které je pomocí for cyklu přičtena ke každé barevné složce pixelu určitá hodnota, o kterou je třeba obraz zesvětlit. [34]

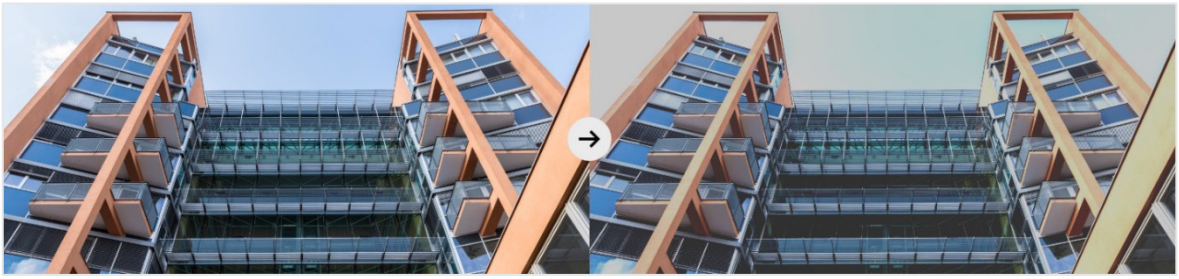
Kód – Processing.py

```
def brighten(img):
    img.loadPixels()
    for i in range(len(img.pixels)):
        c = color(img.pixels[i])
        r = red(c)
        g = green(c)
        b = blue(c)
        r = min(r + 50, 255)
        g = min(g + 50, 255)
        b = min(b + 50, 255)
        img.pixels[i] = color(r, g, b)
    img.updatePixels()
    image(img, 0, 0)
    save("export.png")
```

Obrázek 66: Výstup vlastního pluginu pro aditaci jasu pro GIMP

11.1.5 Ořez jasu (clipping)

V digitálním zpracovávání obrazu je clipping základní technikou, která umožňuje kontrolovat rozsah hodnot pixelů a tím zajišťuje, aby se pohybovaly jen ve stanovených mezích.

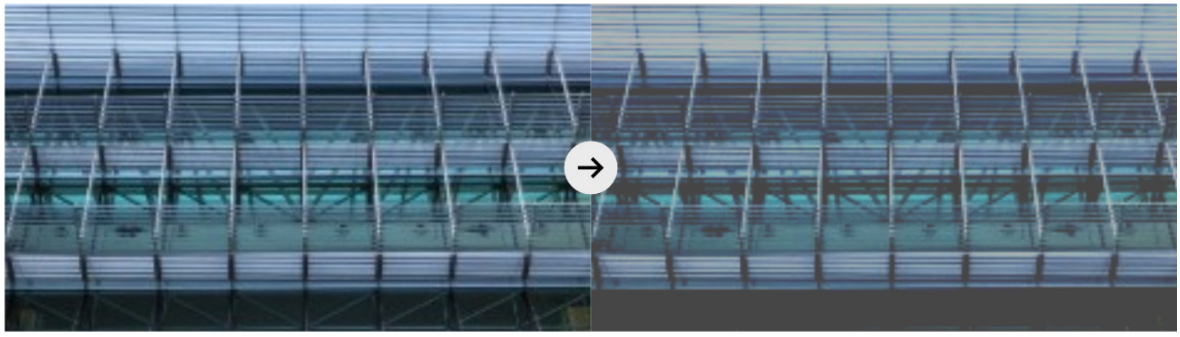


Obrázek 67: Filtr ořezu jasu pomocí Processing.py

V následujícím kódu je vytvořená funkce ***clip()***, která omezí hodnoty barevných složek každého pixelu na zadaný rozsah definovaný parametry ***low*** a ***high***. Pokud pixel překročí stanovenou maximální nebo minimální hodnotu, přepíše se.

Kód – Processing.py

```
def clip(img, low, high):
    img.loadPixels()
    for i in range(len(img.pixels)):
        c = color(img.pixels[i])
        r = red(c)
        g = green(c)
        b = blue(c)
        r = max(min(r, high), low)
        g = max(min(g, high), low)
        b = max(min(b, high), low)
        img.pixels[i] = color(r, g, b)
    img.updatePixels()
    image(img, 0, 0)
```



Obrázek 68: Výstup vlastního pluginu pro aplikaci efektu ořezu jasu pro GIMP

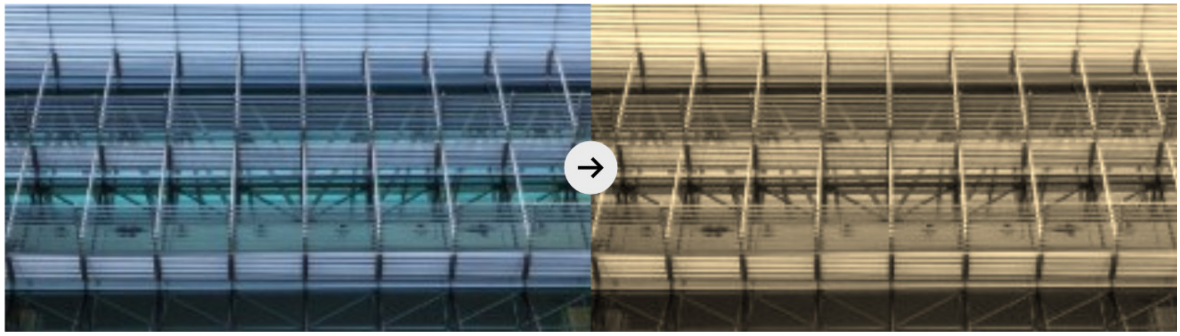
11.1.6 Sépie

Obrázek 69: Efekt sépie pomocí Processing.py

Pomocí iterace je třeba vynásobit každou barevnou složku pixelu vzorcem Sépie. Tento vzorec využívá vážený součet původních barevných kanálů. Váhy jsou zvoleny tak, aby výsledné zbarvení dosáhlo charakteristických teplých tónů sépiového efektu. [34]

Kód – Processing.py

```
def sepia(img):
    w = img.width
    h = img.height
    new_img = createImage(w, h, RGB)
    for x in range(w):
        for y in range(h):
            c = img.get(x, y)
            r = int((c >> 16) & 0xFF)
            g = int((c >> 8) & 0xFF)
            b = int(c & 0xFF)
            tr = int(0.393 * r + 0.769 * g + 0.189 * b)
            tg = int(0.349 * r + 0.686 * g + 0.168 * b)
            tb = int(0.272 * r + 0.534 * g + 0.131 * b)
            sepia_color = color(min(tr, 255), min(tg, 255),
min(tb, 255))
            new_img.set(x, y, sepia_color)
    return new_img
```



Obrázek 70: Výstup vlastního pluginu pro efekt sépie pro GIMP

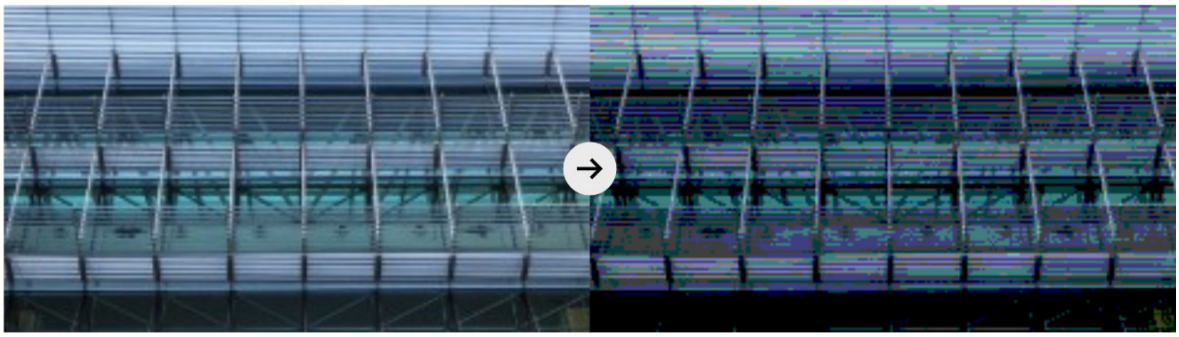
11.1.7 Posterizace

Obrázek 71: Efekt posterizace pomocí Processing.py

Aby bylo možné provést efekt posterizace, je třeba nejprve definovat počet barev, na které budou vstupní data omezeny. V následujícím kódu je počet barev omezen pomocí proměnné ***numLevels***. Tato proměnná je následně použita ve funkci ***map()***, kde je díky ní stanoveno celkové množství nastavitelných hodnot jednotlivé barevné složky pixelu. Pokud je tedy hodnota ***numLeves = 4***, možné hodnoty pro každou složku budou 0, 64, 128, 192, 255 a protože jsou v RGB modelu tří barevné složky, výsledný počet barev lze získat výpočtem $5^3 = 125$. [36]

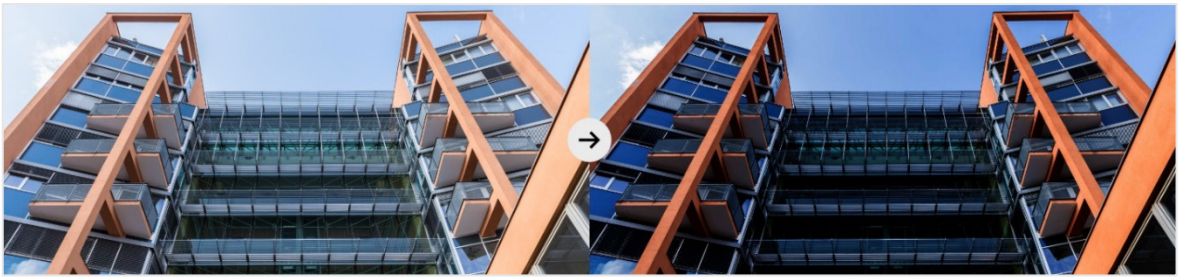
Kód – Processing.py

```
def setup():
    size(800,368)
    img = loadImage("photo.jpg")
    image(img, 0, 0)
    img.loadPixels()
    numLevels = 4
    for x in range(img.width):
        for y in range(img.height):
            loc = x + y * img.width
            c = img.pixels[loc]
            r = red(c)
            g = green(c)
            b = blue(c)
            rLevel = int(map(r, 0, 255, 0, numLevels))
            gLevel = int(map(g, 0, 255, 0, numLevels))
            bLevel = int(map(b, 0, 255, 0, numLevels))
            r = int(map(rLevel, 0, numLevels, 0, 255))
            g = int(map(gLevel, 0, numLevels, 0, 255))
            b = int(map(bLevel, 0, numLevels, 0, 255))
            img.pixels[loc] = color(r, g, b)
    img.updatePixels()
    image(img, 0, 0)
    save("export.png")
```



Obrázek 72: Výstup vlastního pluginu pro efekt posterizace pro GIMP

11.1.8 Gama



Obrázek 73: Gama korekce pomocí Processing.py

Část ***float*(r) / 255** dělí červenou složku pixelu r maximální možnou hodnotou pro 8bitový barevný kanál (255). Tato operace normalizuje hodnotu na rozsah mezi 0 a 1, což představuje relativní intenzitu barvy. Funkce ***float*()** se používá k zajištění toho, aby výsledkem dělení bylo číslo s pohyblivou řádovou čárkou, nikoli celé číslo. Funkce ***pow*(*float*(r) / 255, *gamma*)** zvyšuje normalizovanou hodnotu červené barvy na mocninu gama. Jedná se o základní operaci gama korekce. Protože hodnota gama je obvykle větší než 1 (často se používá 2,2), způsobí tato operace, že normalizovaná hodnota intenzity bude méně lineární, což kompenzuje reakci lidského oka na světlo a reakci monitoru na hodnoty pixelů. Po použití korekce gama je výsledná hodnota stále v rozmezí 0 až 1. Pro převod zpět na 8bitovou hodnotu barvy se korigovaná intenzita vynásobí maximální hodnotou 255.

Kód – Processing.py

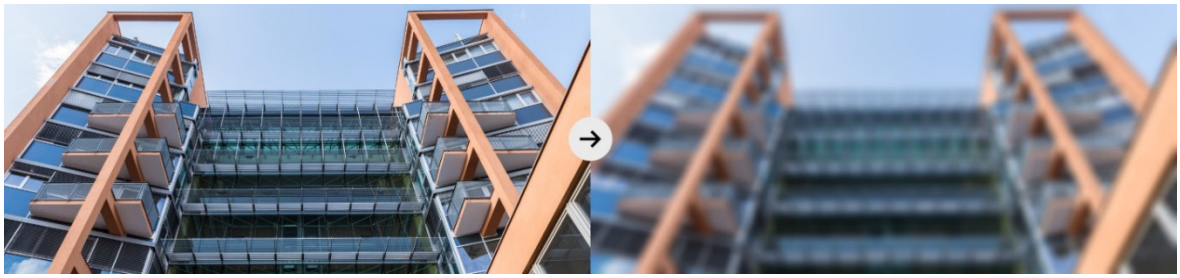
```
def gamma_correct(img, gamma):
    img.loadPixels()
    for i in range(len(img.pixels)):
        c = color(img.pixels[i])
        r = red(c)
        g = green(c)
        b = blue(c)
        r = int(255 * pow(float(r) / 255, gamma))
```

```
            g = int(255 * pow(float(g) / 255, gamma))
            b = int(255 * pow(float(b) / 255, gamma))
            img.pixels[i] = color(r, g, b)
    img.updatePixels()
    image(img, 0, 0)
    save("export.png")
```

11.2 Konvoluční filtry

11.2.1 Gaussovo rozostření

Gaussovo rozostření je technika zpracování obrazu, která využívá Gaussovský filtr (neboli Gaussovu jádro) k rozmazání nebo vyhlazení obrazu. Tento proces je aplikován na snížení šumu nebo detailů v digitálním obrazu, což může být užitečné v mnoha aplikacích, jako jsou úpravy fotografie, počítačové vidění nebo zpracování signálů.



Obrázek 74: Gaussovo rozmazání pomocí Processing.py

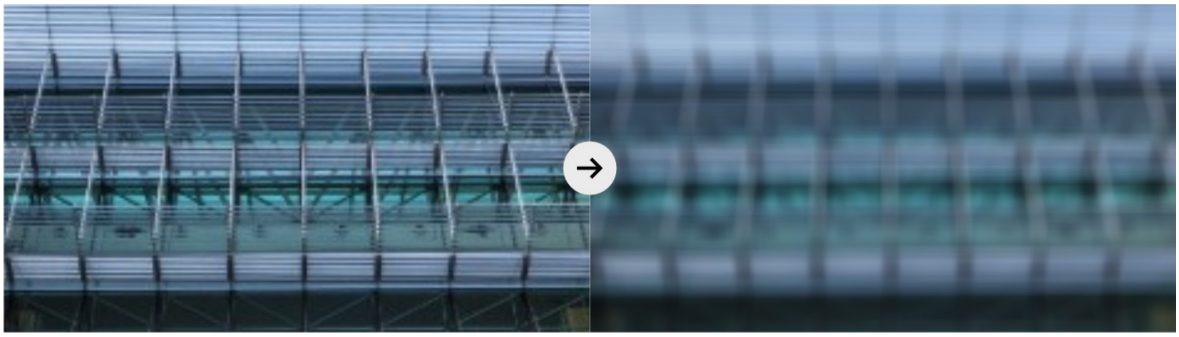
Gaussovský filtr je matice s hodnotami, které představují Gaussovu funkci (také známou jako normální rozdělení). Centrální hodnota matice má nejvyšší váhu a hodnoty klesají směrem ven od středu, což způsobuje rozmazání nebo vyhlazení obrazu. [34]

Kód – Processing.py

```
def setup():
    size(800,368)
    img = loadImage("photo.jpg")
    image(img, 0, 0)
    blurred_img = gaussian_blur(img, 10, 5)
    image(blurred_img, 0, 0)
    save("export.png")

def gaussian(x, y, sigma):
    return (1 / (2 * math.pi * sigma**2)) * math.exp(-(x**2 +
y**2) / (2 * sigma**2))
```

```
def generate_kernel(radius, sigma):
    kernel_size = radius * 2 + 1
    kernel = [[0 for x in range(kernel_size)] for y in
range(kernel_size)]
    for x in range(kernel_size):
        for y in range(kernel_size):
            kernel[x][y] = gaussian(x - radius, y - radius, sigma)
    return kernel

def gaussian_blur(img, radius, sigma):
    blurred_img = createImage(img.width, img.height, RGB)
    kernel = generate_kernel(radius, sigma)
    kernel_size = radius * 2 + 1

    for x in range(img.width):
        for y in range(img.height):
            r_total = 0
            g_total = 0
            b_total = 0
            weight_total = 0
            for i in range(kernel_size):
                for j in range(kernel_size):
                    x_k = x + i - radius
                    y_k = y + j - radius
                    print("x:", x, "y:", y)
                    image(blurred_img, 0, 0)
                    if x_k >= 0 and x_k < img.width and y_k >= 0
and y_k < img.height:
                        weight = kernel[i][j]
                        r_total += red(img.get(x_k, y_k)) * weight
                        g_total += green(img.get(x_k, y_k)) *
weight
                        b_total += blue(img.get(x_k, y_k)) *
weight
                        weight_total += weight
            blurred_img.set(x, y, color(r_total/weight_total,
g_total/weight_total, b_total/weight_total))
    return blurred_img
```

Plugin Gaussova rozostření pro GIMP



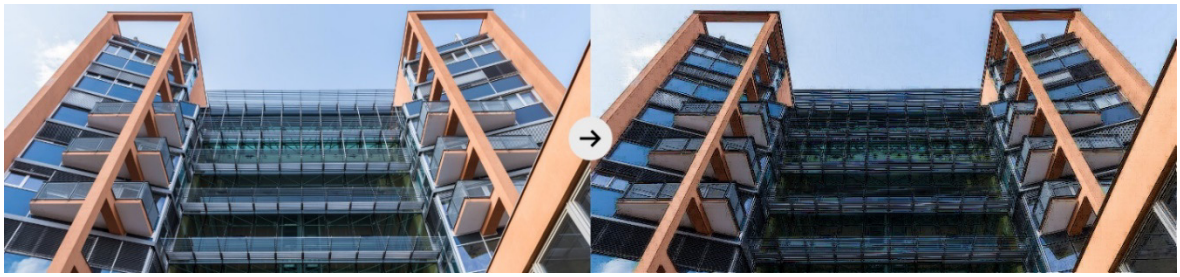
Obrázek 75: Výstup vlastního pluginu pro Gaussův filtr pro GIMP

Kód – Plugin pro GIMP

```
from gimpfu import *
import math

def gaussian(x, y, sigma):
    return (1 / (2 * math.pi * sigma**2)) * math.exp(-(x**2 +
y**2) / (2 * sigma**2))

def generate_kernel(radius, sigma):
    kernel_size = radius * 2 + 1
    kernel = [[0 for x in range(kernel_size)] for y in
range(kernel_size)]
    for x in range(kernel_size):
        for y in range(kernel_size):
            kernel[x][y] = gaussian(x - radius, y - radius, sigma)
    return kernel

def gaussian_blur(image, layer, radius, sigma):
    pdb.gimp_image_undo_group_start(image)
    blurred_layer = layer.copy()
    image.add_layer(blurred_layer, 0)
    kernel = generate_kernel(radius, sigma)
    kernel_size = radius * 2 + 1

    for x in range(blurred_layer.width):
        for y in range(blurred_layer.height):
            r_total = 0
            g_total = 0
            b_total = 0
            weight_total = 0
            for i in range(kernel_size):
```

```
                for j in range(kernel_size):
                    x_k = x + i - radius
                    y_k = y + j - radius
                    if x_k >= 0 and x_k < layer.width and y_k >= 0
and y_k < layer.height:
                        weight = kernel[i][j]
                        pixel = layer.get_pixel(x_k, y_k)
                        r_total += pixel[0] * weight
                        g_total += pixel[1] * weight
                        b_total += pixel[2] * weight
                        weight_total += weight
            blurred_layer.set_pixel(x, y,
(int(r_total/weight_total), int(g_total/weight_total),
int(b_total/weight_total)))
        progress = float(x) / blurred_layer.width
        gimp.progress_update(progress)

    pdb.gimp_image_undo_group_end(image)
    gimp.displays_flush()

def plugin_main(image, layer, radius, sigma):
    gaussian_blur(image, layer, radius, sigma)

register(
    "python_fu_gaussian_blur_progress",
    "Gaussian Blur",
    "Applies a Gaussian Blur with a progress bar.",
    "Your Name",
    "Your Name",
    "2023",
    "<Image>/Filters/Custom/Gaussian Blur",
    "RGB*, GRAY*",
    [
        (PF_INT, "radius", "Radius", 10),
        (PF_FLOAT, "sigma", "Sigma", 5.0),
    ],
    [],
    plugin_main)

main()
```

11.2.2 Zvýraznění hran



Obrázek 76: Konvoluční filtr pro zaostření pomocí Processing.py

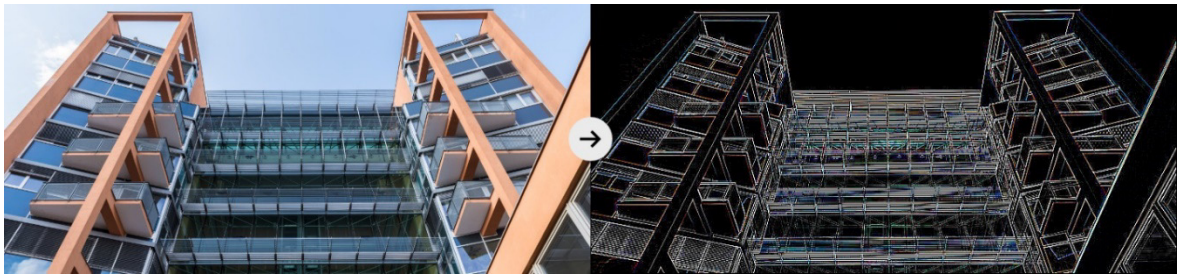
Matice má středovou hodnotu kladnou a okolní hodnoty záporné. Tento filtr potlačuje vysokofrekvenčních složek obrazu, jako jsou ostré hrany a detaily, zatímco umožňuje průchod nízkofrekvenčních složek jako je hladké pozadí a šum.

Kód – Processing.py

```
def filter(img):
    kernel = [
        [-1, -1, -1],
        [-1,  8, -1],
        [-1, -1, -1]
    ]
```

11.2.3 Detekce hran (Laplaceův filtr)

Laplacův filtr je druh lineárního filtru používaný ve zpracování obrazu a počítačovém vidění k detekci hran, ostrých změn intenzity nebo vylepšení detailů v obrazu. Laplacův filtr je založen na druhé derivaci obrazové funkce, což znamená, že je citlivý na změny intenzity v různých směrech. [36]



Obrázek 77: Konvoluční filtr pro detekci hran pomocí Processing.py

Kernel může být například ve 2D prostoru ve tvaru 3x3 nebo 5x5, kde středová hodnota má negativní váhu a okolní hodnoty mají kladnou váhu. Výsledkem je, že oblasti s konstantní

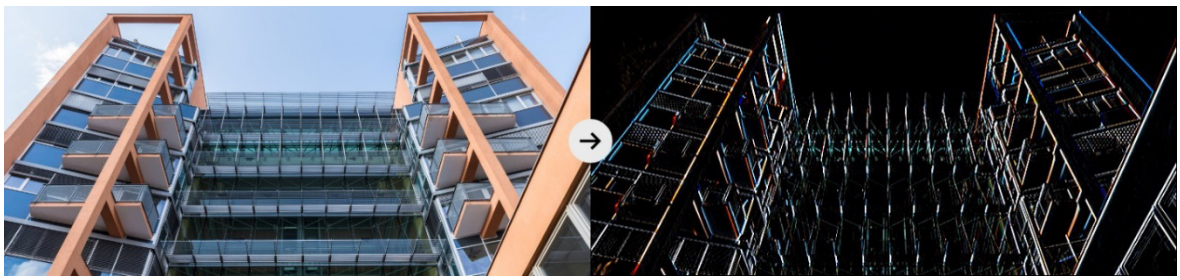
intenzitou mají nulový výstup, zatímco oblasti s vysokou změnou intenzity mají vysoký výstup.

Kód – Processing.py

```
def sobel_filter(img):
    kernel = [
        [1, 1, 1],
        [1, -8, 1],
        [1, 1, 1]
    ]
```

11.2.4 Detekce hran (Sobelův filtr)

Sobelův filtr je jedním z konvolučních filtrů používaných v zpracování obrazu a počítačovém vidění k detekci hran a vylepšení detailů v obrazu. Sobelův filtr se skládá ze dvou matic (kernels), jedna pro detekci vodorovných hran a druhá pro detekci svislých hran. Tyto matice jsou založeny na aproximaci prvního derivátu obrazové funkce a reagují na změny intenzity v různých směrech. [34]



Obrázek 78: Konvoluční Sobelův filtr pro detekci hran pomocí Processing.py

Kód – Processing.py

```
def sobel_filter(img):
    kernel = [
        [-1, -2, -1],
        [ 0,  0,  0],
        [ 1,  2,  1]
    ]
```

12 RASTERIZACE

Pro názorné ukázaní principu rasterizace byl vytvořen interaktivní script v Processing.py, který na základě pozice kurzoru rasterizuje původní obraz. Posunem myši tak lze zobrazit originální obraz a postupně jeho rozlišení snižovat až na minimum pixelů. [35]

Obrázek 79: Rasterizace pomocí Processing.py

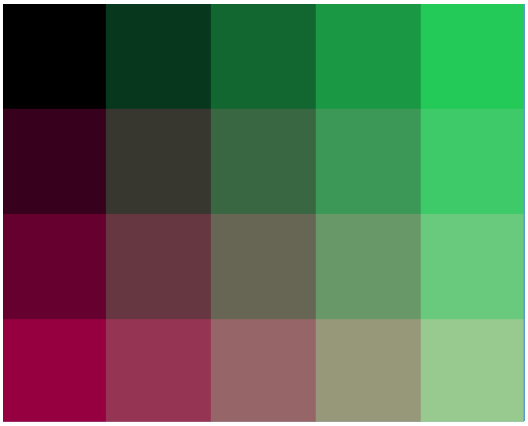
Kód – Processing.py

```
def draw():
    background(255)
    fill(0)
    noStroke()
    tiles = mouseX
    tileSize = width / tiles
    translate(tileSize / 2, tileSize / 2)

    for x in range(int(tiles)):
        for y in range(int(tiles)):
            c = img.get(int(x * tileSize), int(y * tileSize))
            size = map(brightness(c), 0, 255, tileSize, 0)

            if size > tileSize / 2:
                size = tileSize
                rect(x * tileSize, y * tileSize, size, size)
```

13 GRAFICKÉ FORMÁTY

13.1 Rastrové formáty

Pro rozbor grafických formátů byl vytvořen script v Pythonu, který vygeneroval barevný obrázek o rozměrech 4x5 pixelů. Tento obrázek byl následně uložen jako soubor BMP, JPEG a PNG.



Obrázek 80: Výstup python kódu pro tvorbu obrázku 4x5 pixelů

13.1.1 BMP

Pro přečtení souboru BMP byl vytvořen Python script, který pomocí známých identifikátorů vytvořil přehled obsahu BMP. Z výstupu kódu je možné zjistit, že se BMP soubor skládá z tří hlavních částí: hlavičky souboru, informační hlavičky a datové části. Kompletní rozbor BMP viz. příloha. [31]

Náhled souboru BMP

```
42 4d 76 00 00 00 00 00 00 00 36 00 00 00 28 00 00 00 05 00 00 00 04 00 00 00
01 00 18 00 00 00 00 00 40 00 00 00 c4 0e 00 00 c4 0e 00 00 00 00 00 00 00 00
00 00 3c 00 96 50 32 96 64 64 96 78 96 96 8c c8 96 00 28 00 64 3c 32 64 50 64
64 64 96 64 78 c8 64 00 14 00 32 28 32 32 3c 64 32 50 96 32 64 c8 32 00 00 00
00 14 32 00 28 64 00 3c 96 00 50 c8 00 00
```

13.1.2 JPEG

Pro přečtení souboru JPEG byl vytvořen Python script, který pomocí známých identifikátorů vyznačující jednotlivé segmenty souboru vytvořil přehled obsahu JPEG. Je v něm možné pozorovat segmenty pro kvantizační tabulku, DCT, nebo Huffmanovy tabulky. [32] Kompletní rozbor JPEG obrázku viz. příloha.

Náhled souboru JPEG

```
ff d8 ff e0 00 10 4a 46 49 46 00 01 01 00 00 01 00 01 00 00 ff db 00 43 00 08
06 06 07 06 05 08 07 07 07 09 09 08 0a 0c 14 0d 0c 0b 0b 0c 19 12 13 0f 14 1d
1a 1f 1e 1d 1a 1c 1c 20 24 2e 27 20 22 2c 23 1c 1c 28 37 29 2c 30 31 34 34 34
1f 27 39 3d 38 32 3c 2e 33 34 32 ff db 00 43 01 09 09 09 0c 0b 0c 18 0d 0d 18
32 21 1c 21 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32
32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32 32
32 32 ff c0 00 11 08 00 04 00 05 03 01 22 00 02 11 01 03 11 01 ff c4 00 1f 00
00 01 05 01 01 01 01 01 01 00 00 00 00 00 00 00 00 01 02 03 04 05 06 07 08 09
0a 0b ff c4 00 b5 …
```

13.1.3 PNG

Pro přečtení souboru PNG byl vytvořen Python script, který pomocí známých identifikátorů tzv. chunků vytvořil přehled PNG. Z přehledu je možné pozorovat jednotlivé typy chunků, jako je chunk o základních informacích, chunk s obrazovými daty nebo chunk, který ukončuje soubor. [33] Kompletní rozbor PNG viz. příloha.

Náhled souboru PNG

```
89 50 4e 47 0d 0a 1a 0a 00 00 00 0d 49 48 44 52 00 00 00 05 00 00 00 04 08 02
00 00 00 c9 51 62 17 00 00 00 19 49 44 41 54 78 9c 63 64 60 60 60 30 12 81 23
16 23 06 11 06 24 44 88 0f 00 6a 6a 02 e8 9e 1b 08 bd 00 00 00 00 49 45 4e 44
ae 42 60 82
```

13.2 Vektorové formáty

Čtení vektorových formátů nelze jednoduše automatizovat pomocí scriptů, proto byl vytvořen ukázkový vektorový obraz v Inkscape pro 2D formáty a v Blenderu pro 3D formáty a uložen jako SVG, DXF, STL a OBJ. Podrobný rozbor těchto formátů viz. příloha.

ZÁVĚR

Tato bakalářská práce se zabývala studiem základů vybraných oblastí počítačové grafiky a tvorbou podkladů s využitím zvolených softwarových nástrojů. Práce zahrnuje popis vybraných základních principů počítačové grafiky, stejně jako práci s technologiemi a nástroji, které se v tomto oboru používají. Díky praktickým výstupům bylo ověřeno, jak se tyto teoretické koncepty realizují v praxi.

V průběhu tvorby práce se softwarový nástroj Processing.py ukázal jako velmi užitečný při snaze zrekonstruovat známe grafické objekty (zejména Bézierovy křivky a B-spline křivky), geometrické transformace pomocí transformační matice nebo nejpoužívanější grafické filtry. Interaktivní výstup těchto scriptů slouží jako názorná ukázka matematických principů, které se za těmito objekty a operacemi skrývají. Tyto scripty dále sloužily jako dobrý podklad pro tvorbu praktických výstupů v podobě pluginů pro programy GIMP, Inkscape nebo Blender, které vycházejí z ověřených principů v Processing.py. Tímto postupem bylo prokázáno, že znalosti uvedené v teoretické části bakalářské práce je možné svépomocí uplatnit pro praktické využití v oblasti počítačové grafiky.

Dále byly popsány nejznámější barevné modely a grafické formáty. V tomto případě již Processing.py nenašel své využití, a proto byly pro lepší pochopení principů barevných modelů vytvořeny python scripty pro výpočet převodu mezi jednotlivými modely. Pro pochopení vlastností vybraných obrazových formátů byly vytvořeny python skripty, které přehledně vypisují obsah souborů obrazových formátů. Tento výpis byl podrobně popsán v příloze bakalářské práce.

Současným trendem v oblasti počítačové grafiky je snaha izolovat tvůrce od úkonů vyžadující určitou technickou znalost. Tento trend můžeme sledovat na vzniku mnoha no-code služeb a jiných minimalistických aplikací v posledních několika letech. I přesto, že tento trend s sebou přináší výhody v podobě větší dostupnosti a lepší efektivity při práci, nese s sebou i určité riziko závislosti na těchto službách. Jedním z cílů práce bylo ukázat, že se znalostí základních principů počítačové grafiky lze svépomocí docílit užitečných grafických výstupů.

I přes zúžení tématu počítačové grafiky do několika stěžejních oblastí se i nadále jedná o velmi široké téma, které je vhodné dále zkoumat do větší hloubky.

SEZNAM POUŽITÉ LITERATURY

[1] Vektory. *Matematika polopatě* [online]. 1. 12. 2015 [cit. 2023-02-19]. Dostupné z: https://www.matweb.cz/vektory/Druhý zdroj

[2] Operace s vektory. *Matematika polopatě* [online]. 1. 12. 2015 [cit. 2023-02-19]. Dostupné z: https://www.matweb.cz/vektory-operace/

[3] VALÁŠEK, Marek. Analytická geometrie 10: Vektory – násobení vektoru číslem. In: *YouTube.com* [online]. 27. 4. 2017 [cit. 2023-02-19]. Dostupné z: https://www.youtube.com/watch?v=sXTq_9Ec8p8

[4] Skalární součin. *Matematika polopatě* [online]. 1. 12. 2015 [cit. 2023-02-22]. Dostupné z: https://www.matweb.cz/skalarni-soucin/

[5] Vektorový součin. *Matematika polopatě* [online]. 1. 12. 2015 [cit. 2023-02-24]. Dostupné z: https://www.matweb.cz/vektorovy-soucin/

[6] Normálový vektor přímky. *Matematika polopatě* [online]. 1. 12. 2015 [cit. 2023-02-28]. Dostupné z: https://www.matweb.cz/normalovy-vektor-primky/

[7] Matice. *Matematika polopatě* [online]. [cit. 2023-03-01]. Dostupné z: https://www.matweb.cz/matice/

[8] ŠETEK, david. Matematika: *Matice 9 - Transpozice matice* (část 1). In: *YouTube.com* [online]. 22. 8. 2015 [cit. 2023-03-01]. Dostupné z: https://www.youtube.com/watch?v=m62433LeHtM

[9] ŠETEK, david. Matematika: *Matice 27 - Inverzní matice* matice 2x2. In: YouTube.com [online]. 22. 8. 2015 [cit. 2023-03-01]. Dostupné z: https://www.youtube.com/watch?v=pJ4voTUzYPo

[10] ŽÁRA, Jiří, Bedřich BENEŠ, Jiří SOCHOR a Petr FELKEL. *Moderní počítačová grafika. 2*. vyd. Praha: Computer Press, 2005. 609 s. I 1. ISBN 80-251-0454-0.

[11] BUNN, Tristan. Learn Python visually: *creative coding with processing.py*. San Francisco: No Starch Press, [2021]. ISBN 978-1-7185-0096-9.

[12] Inkscape.org: *Draw freely* [online]. [cit. 2023-03-21]. Dostupné z: https://inkscape.org/

[13] GIMP: *GNU Image Manipulation Program* [online]. [cit. 2023-03-21]. Dostupné z: https://www.gimp.org/

[14] Blender [online]. [cit. 2023-03-21]. Dostupné z: https://www.blender.org/

[15] VAN GUMSTER, Jason. Coordinate Systems in Blender. *Dummies* [online]. 3. 26. 2016 [cit. 2023-03-22]. Dostupné z: https://www.dummies.com/article/technology/software/animation-software/blender/coordinate-systems-in-blender-142885/

[16] CHABADA, Tomáš. Analytická geometrie: *délka úsečky, střed úsečky* [online]. In: . 8. 10. 2018 [cit. 2023-05-20]. Dostupné z: https://www.youtube.com/watch?v=bmIG2pbDLiM

[17] SANJU. Equation of a Line in 3D. *Geeksforgeeks.org* [online]. 11. 2. 2021 [cit. 2023-03-23]. Dostupné z: https://www.geeksforgeeks.org/equation-of-a-line-in-3d/

[18] Perimeter of Ellipse. *Cuemath* [online]. [cit. 2023-05-20]. Dostupné z: https://www.cuemath.com/measurement/perimeter-of-ellipse/

[19] CHLÁDEK, Dominik. MAT - Analytická geometrie: *Kružnice a přímka* [online]. In: . 31. 8. 2018 [cit. 2023-03-23]. Dostupné z: https://www.youtube.com/watch?v=QJB1VPph6pE

[20] POKORNÝ, Pavel. *Základy počítačové grafiky*. Zlín, 2004. Skripta. Univerzita Tomáše Baťi.

[21] PIEGL, *Les a Wayne TILLER. The NU'RBS Book*: *Monographs in Visual Communications*. 2. Tyler: Springer, 1997. ISBN 978-3-540-61545-3.

[22] GAPO, BRANKO. What Is Anti-Aliasing And Which Type Should You Use?. *GPU Mag* [online]. 1. 10. 2022 [cit. 2023-03-24]. Dostupné z: https://www.gpumag.com/anti-aliasing/

[23] How PNG Works: *Compromising Speed for Quality*. In: *YouTube.com* [online]. 29. 3. 2022 [cit. 2023-03-24]. Dostupné z: https://www.youtube.com/watch?v=EFUY-NoFRHQI&t=936s

[24] The Unreasonable Effectiveness of JPEG: *A Signal Processing Approach*. In: *YouTube.com* [online]. 19. 1. 2022 [cit. 2023-03-24]. Dostupné z: https://www.youtube.com/watch?v=0me3guauqOU&t=1725s

[25] DOBEŠ, Michal. *Zpracování obrazu a algoritmy v C#.* Praha: BEN - technická literatura, 2008, 143 s. ISBN 9788073002336.

[26] GRANGER, Matt. SRGB vs Adobe RGB: *which colour space should you use?*. *YouTube.com* [online]. 30. 3. 2015 [cit. 2023-04-01]. Dostupné z: https://www.youtube.com/watch?v=N2T62ZHIu6s

[27] PETRAS, Rastislav. *Algoritmy transformace 2D obrazu*. Zlín, 2011. Diplomová práce. Univerzita Tomáše Baťi. Vedoucí práce Ing. Pavel Pokorný, PhD.

[28] Vector files. In: *Adobe.com* [online]. [cit. 2023-04-01]. Dostupné z: https://www.adobe.com/creativecloud/file-types/image/vector.html

[29] CHAKRAVORTY, Dibya. The OBJ File Format: *Simply Explained*. *All3DP* [online]. 30. 3. 2023 [cit. 2023-04-03]. Dostupné z: https://all3dp.com/1/obj-file-format-3d-printing-cad/

[30] X3D *Examples* [online]. In: . 19. 8. 2020 [cit. 2023-05-02]. Dostupné z: https://www.youtube.com/watch?v=N8XUCxOtpH4

[31] OSUSKÝ, Andrej a Petr PŘINDIŠ. *Zpracování rastrových obrázků formátu BMP a PCX*. Zlín, 2016. Seminární práce. Univerzita Tomáše Baťi. Vedoucí práce Pavel Navrátil.

[32] TIŠNOVSKÝ, Pavel. *JPEG: král rastrových grafických formátů?* [online]. 7. 12. 2006 [cit. 2023-05-04]. Dostupné z: https://www.root.cz/clanky/jpeg-kral-rastrovych-grafickych-formatu/

[33] PNG (Portable Network Graphics) Specification: Version 1.2. *W3.org* [online]. [cit. 2023-05-24]. Dostupné z: http://www.libpng.org/pub/png/spec/1.2/PNG-Structure.html

[34] ČECH, Pavel. *Tvorba audio a video filtrů pro program VirtualDub*. Zlín, 2012. Bakalářská práce. Univerzita Tomáše Baťi. Vedoucí práce Ing. Pavel Navrátil, PhD.

[35] RODENBRÖKER, Tim. Rasterize 3D: *Processing Tutorial* [online]. In: . 20. 3. 2020 [cit. 2023-05-24]. Dostupné z: https://www.youtube.com/watch?v=WEBOTR-boXBE&t=612s

[36] NAVRÁTIL, Pavel. *Sbírka podkladů k počítačové grafice a teorie informace*

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SW	Software
×	Vektorový součin
2D	Dvoudimenzionální prostor
3D	Trojdimenzionální prostor
.py	Koncovka označující Python kód
SVG	Scalable Vector Graphics
GIMP	GNU Image Program
NURBS	Non Uniform Racional B-spline
PNG	Portable Network Graphics
GIF	Graphics Interchange Format
LZSS	Lempel-Ziv-Storer_Szymski
CRC-32	Cyclic Redundancy Check – Kontrolní součet o 32 bitech
JPEG	Joint Photographic Experts Group
RLE	Run Length Encoding
RGB	Red-Green-Blue Barevný model
sRGB	Barevný model spadající pod RGB
AdobeRGB	Barevný model spadající pod RGB
CMYK	Barevný model
BMP	Mitmap – grafický formát
DIB	Device Independent Bitmap
WebP	Grafický formát
XML	Programovací jazyk
W3C	World Wide Web Consorcium
CAD	Computer-Aided Design

DXF	Grafický formát
STL	Stereolithography – Grafický formát
X3D	Grafický formát
MSAA	Multisampling Antialiasing
SSAA	Super Sampling Antialiasing
FXAA	Fast Aproximate Antialiasing
.inx	Koncovka označující identifikátor pluginu pro Inkscape

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P1: CD

Přiložené CD obsahuje:

- Bakalářskou práci ve formátu .pdf BP_MikuleckyPavel_2023.pdf
- Přílohu ve formátu .pdf BP_Priloha.pdf
- Scripty a výstupy ve formátu .zip: prilohy.zip