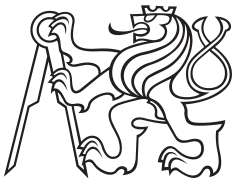
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Vektorová grafika s podporou implicitních křivek

Vojtěch Nesvačil

Školitel: Ing. Ladislav Čmolík, Ph.D.
Květen 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Nesvačil** Jméno: **Vojtěch** Osobní číslo: **507341**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Vektorová grafika s podporou implicitních křivek

Název bakalářské práce anglicky:

Vector Graphics with Suport for Implicit Curves

Pokyny pro vypracování:

1. Seznamte se s reprezentací základních 2D objektů (kruh, elipsa, obdélník) pomocí implicitních rovnic a s implicitní reprezentací 2D Bézierových křivek.
2. Na základě analýzy navrhněte a implementujte aplikaci umožňující načtení vektorové grafiky ve formátu SVG (Scalable Vector Graphics), vyjádření načtených 2D objektů v implicitní reprezentaci a jejich vykreslení. Umožněte vykreslování obrysů 2D objektů a jejich vyplňování konstantní barvou. Vykreslování 2D objektů v implicitní reprezentaci bude probíhat na grafické kartě v prostoru obrazu (fragment shader).
3. Na příkladech 2D vektorové grafiky s různou složitostí změřte rychlost vykreslování 2D objektů v implicitní reprezentaci pomocí realizované aplikace a porovnejte s tradičním způsobem vykreslování 2D vektorové grafiky na grafické kartě (konverze na polygonální reprezentaci a její vykreslení). Při měření rychlosti vykreslování uvažujte i základní operace s vektorovou grafikou jako posun, rotace a změna měřítka.

Seznam doporučené literatury:

[1] Marschner S. and P. Shirley, Fundamentals of Computer Graphics, 4th edition, CRC Press, 2015.
[2] Loop Ch. and J. Blinn, Resolution Independent Curve Rendering using Programable Graphics Hardware, In ACM SIGGRAPH 2005 Papers, pp. 1000-1009, 2005.
[3] Santina R., Resolution Independent NURBS Curves Rendering using Programmable Graphics Pipeline, In GraphiCon'2011 Conference Proceedings, pp. 70-73, 2011.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ladislav Čmolík, Ph.D. Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **08.01.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. Ladislav Čmolík, Ph.D.	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Ladislav Čmolík, Ph.D. za zájem, trpělivost, připomínky a čas, který mi a mé práci věnoval.

Dále bych chtěl poděkovat své rodině za zázemí, která mi byla poskytnuta tři studiu a tvorbě mé práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 24.5.2024

. . . . . .
podpis autora práce

Abstrakt

Tato bakalářská práce se zabývá vykreslováním vektorové grafiky s podporou implicitních křivek. Vykreslení probíhá pomocí implicitních rovnic, které jsou počítány v prostoru fragment shaderu. V rámci práce bylo implementováno vykreslení tvarů typu kruh, obdélník, Bézierovy křivky či složitějšího obrazce. Vykreslované obrázky jsou načítány ze souborů formátu SVG. Práce se zabývá rychlostí a správností vykreslení různých obrazců.

Klíčová slova: vektorová grafika, implicitní křivka, implicitní rovnice, aproximace křivky, Bézierova křivka, rasterizace na GPU, fragment shader

Školitel: Ing. Ladislav Čmolík, Ph.D.

Abstract

This bachelor's thesis focuses on rendering vector graphics with support for implicit curves. The rendering is done using implicit equations, which are computed in the fragment shader space. The rendering of shapes such as circles, rectangles, Bézier curves, and more complex shapes was implemented. The rendered images are loaded from SVG format files. The thesis examines the speed and accuracy of rendering various shapes.

Keywords: vector graphics, implicit curve, implicit equation, curve approximation, Bézier curve, GPU rasterization, fragment shader

Title translation: Vector Graphics with Support for Implicit Curves

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Na střední škole jsem se zabýval 2D (2 dimensions) a 3D (3 dimensions) vykreslováním. Nabyté zkušenosti jsem nakonec uplatnil při mé maturitní práci. Po seznámení se základními principy počítačové grafiky, dále prohlubuji své znalosti a mé další dovednosti a zájem směřuje k problematice vykreslení křivek. Jako téma této práce jsem si zvolil porovnání dvou způsobů vykreslení křivek na GPU (Graphics Processing Unit).

Vektorová grafika a rastrová grafika jsou dva základní způsoby reprezentace obrazu v počítačové grafice. V rastrové grafice je obraz reprezentován pomocí mřížky bodů. Mřížka má za následek ztrátu kvality při transformacích, například při přiblížení obrázku. Nejčastěji jsou rastrovou grafikou reprezentovány fotografie a také obraz na moderních displejích.

Ve vektorové grafice je obraz reprezentován úsečkami, křivkami a dalšími geometrickými tvary. Při takové reprezentaci nedochází při transformacích ke ztrátě kvality. Používá se proto u písem a ikon, kde je třeba používat více různých velikostí.

Při zobrazení vektorové grafiky na moderním displeji je třeba provést rasterizaci, protože obraz na moderním displeji je reprezentován rastrovou grafikou. Proces rasterizace je náročný výpočet a může být efektivně proveden na grafické kartě.

1.1 Cíl práce

Tato práce se zabývá reprezentací křivek pomocí implicitních rovnic a následně jejich vykreslením za využití moderních grafických karet. Cílem této práce je změřit a porovnat rychlost vykreslování vektorové grafiky na grafické kartě s podporou implicitních křivek a tradičním způsobem triangulací křivek. Vektorová grafika s využitím implicitních rovnic není žádnou novinkou, avšak rychlosti jednotlivých způsobů vykreslování se mohou lišit. Důvodem je neustále vyvíjející se a měnící se architektury GPU na grafických kartách.

1.2 Struktura práce

Tato práce je rozdělena do několika kapitol. V druhé kapitole si projdeme reprezentaci geometrických tvarů s možnými řešeními vykreslování. V následující třetí kapitole provedeme implementaci programu. Ve čtvrté kapitole jsou demonstrovány funkčnost programu a výsledky měření. V poslední kapitole shrneme celou práci.

Kapitola 2
Analýza a Návrh

V této kapitole je popsaný rozdíl mezi vektorovou a rastrovou grafikou. Věnuje se problematice důležité části rasterizace na grafické kartě a popisujeme metody vykreslení křivek včetně transformace 2D prostoru. Zaměřujeme se také na metodu vykreslení křivek pomocí implicitních rovnic na grafické kartě. Metoda vykreslení křivek pomocí aproximace je zmíněna za účelem vyhodnocení práce.

2.1 Vektorová a rastrová grafika

Vektorová grafika a rastrová grafika jsou dva základní způsoby reprezentace obrazu v počítačové grafice. Rastrový způsob reprezentace obrazu je pomocí pixelů (barevných bodů) uspořádaných do mřížky. Při transformacích rastrového obrázku může dojít ke ztrátě kvality. Například na obrázku 2.1 došlo k přiblížení a zviditelnění jednotlivých bodů obrazu. Rastrovou grafikou se reprezentují fotografie a obrazy na moderních displejích.

Ve vektorové grafice je obraz vytvářen úsečkami, křivkami a dalšími geometrickými tvary. Vektorový způsob eliminuje ztrátu kvality, při provádění transformací. Vektorový způsob je vhodný pro písma, loga, ikony nebo ilustrace, které je třeba zobrazit v několika různých velikostech.

Při zobrazení vektorové grafiky na moderním displeji je třeba provést rasterizaci. Rasterizace je převod vektorové grafiky do rastrové grafiky bodů, protože obraz na moderním displeji je reprezentován v bodech. Proces rasterizace lze provádět na dedikovaném procesoru (GPU), který je k tomu hardwarově přizpůsobený.

2.2 Rasterizace na GPU

Rasterizace je na počítačích prováděna často, a proto byl vytvořen dedikovaný hardware zvaný GPU [1]. Přesunutí rasterizace na specializovanou část hardwaru GPU zvyšuje výkon stroje, jelikož jeho kapacitu uvolňuje pro jiné potřebné úkony. Samotná jednotka GPU provádí výpočty paralelně, a tak PC (Personal Computer) dosahuje větších rychlostí. Z prvopočátku byl proces rasterizace pevně daný, dnes jsou již některé části programovatelné. V prvních

Obrázek 2.1: Vektorová grafika vlevo, při přiblížení je kvalita obrazu zachována. Rastrová grafika vpravo, při přiblížení se kvalita obrazu zhoršuje.

verzích vykreslování byl pevně dán proces zpracování jednotlivých bodů s minimem nastavení. V současné době lze předat do grafiky vlastní instrukce postupu vykreslování.

Na moderní grafickém procesoru je na vstupu procesu rasterizace geometrie, která je sestavena z primitivních geometrických tvarů. Jedním z používaných tvarů je trojúhelník. Postup rasterizace trojúhelníka je podobný i úsečce či bodu. Každý trojúhelník geometrického tvaru je popsán vrcholy se souřadnicemi.

Jednotlivé kroky rasterizace jsou vidět na obrázku 2.2.

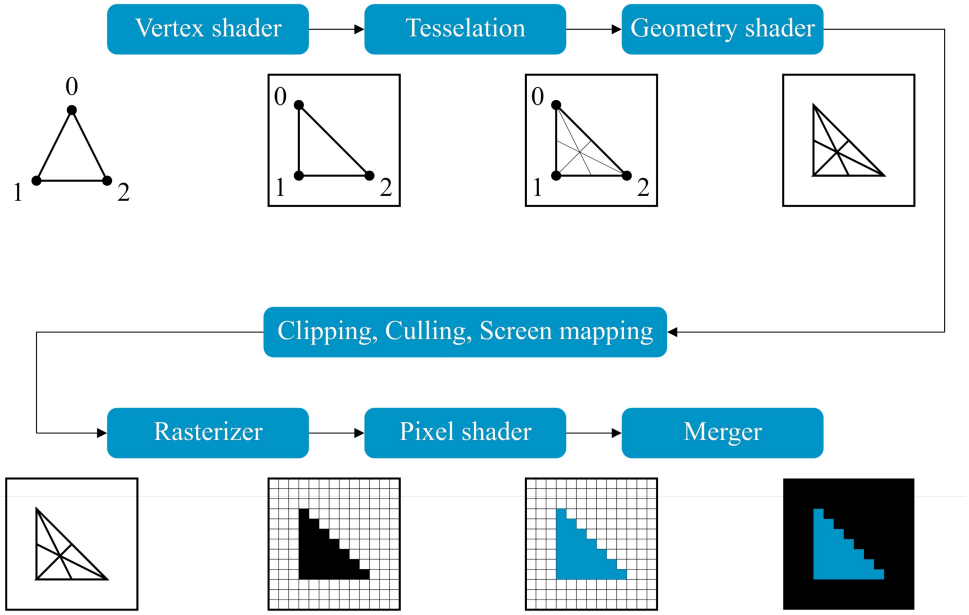
Nejprve jsou všechny vrcholy transformovány do prostoru obrazovky. Následně jsou vrcholy geometrie seskupeny do trojúhelníků. Trojúhelníky, které se nachází mimo hranice obrazovky jsou odebrány. Dále je každý trojúhelník rasterizován. Rasterizací rozumíme převedení trojúhelníku do bodů. Každý bod má své souřadnice a vytváří mřížku. V dalším kroku se v pixel shaderu jednotlivým bodům nastaví barva. Vykreslované body jsou podrobeny různým testům. Pokud bod testy projde, barva bodu je následně smíchána s předchozí barvou výsledného obrazu.

2.2.1 Shader

Při procesu rasterizace jsou programované části nazývány shadery. Původně se jako shader označoval proces počítání barev bodů. Dnes je pojmem shader označen jakýkoliv programový kód na grafické kartě [2].

Nejčastější shadery jsou:

- Vertex shader transformuje vrcholy geometrie.
- Tesselation shadery (hull shader a domain shader) umožňují rozdělit geometrické primitiva (například trojúhelník na více trojúhelníků).



Obrázek 2.2: Rasterizace na GPU

- Geometry shader umožňuje generovat nové geometrické primitiva a tím zpřesnit aproximaci reprezentovaného tvaru.
- Pixel shader řeší barvy pixelu či vzorku pixelu.

2.2.2 Vertex shader

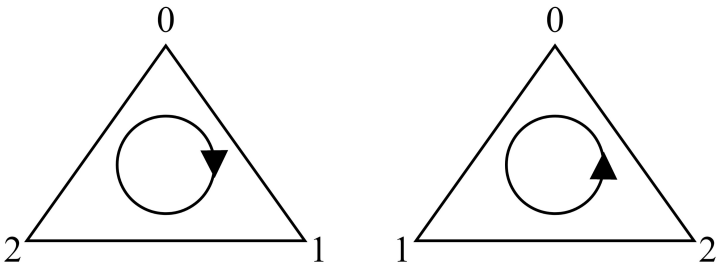
Prvním krokem je sestavení seznam vrcholů geometrie. Vrchol geometrie vedle souřadnic os x, y a z obsahovat další data, například texturovací souřadnice, pro mapování textur na geometrii objektu, nebo směr kolmice k ploše pro počítání osvětlení [1, 3].

Po sestavení vrcholů geometrie, následuje první programovatelná část nazývaná vertex shader [1, 4]. Program je spouštěn pro každý vrchol zvlášť. Vrchol je zde transformován postupně z model prostoru do clip prostoru. Clip prostor je prostor, kde vše, co je mimo rozsahu -1 až 1, bude odebráno v clip kroku.

Výstup vertex shaderu, transformovaný vrchol geometrie, může být vstupem volitelného kroku teselace. Teselace může být přeskočena a dalším krokem je pak geometry shader, který je taktéž volitelný a může být přeskočen.

2.2.3 Proces teselace

Jedním z volitelných kroků po vertex shaderu je teselace. Teselace umožňuje z trojúhelníku vytvořit síť drobnějších trojúhelníků. Tento proces je rozložen do tří kroků hull shader, hardwarová teselace a domain shader. Tato problematika



Obrázek 2.3: Vlevo trojúhelník definovaný proti směru hodinových ručiček. Vpravo trojúhelník definovaný po směru hodinových ručiček.

je zajímavě popsaná v knize od Akenine-Möller Tomas a spol. s názvem Real-Time Rendering [1].

2.2.4 Geometry shader

Volitelný krok po vertex shaderu nebo teselaci je geometry shader. Geometry shader je předchůdce teselace [1]. Geometry shader se používá pro vytváření nových geometrických primitiv např. particle effektů, kde se z jednoho bodu vytvoří tzv. billboard [1].

2.2.5 Clipping

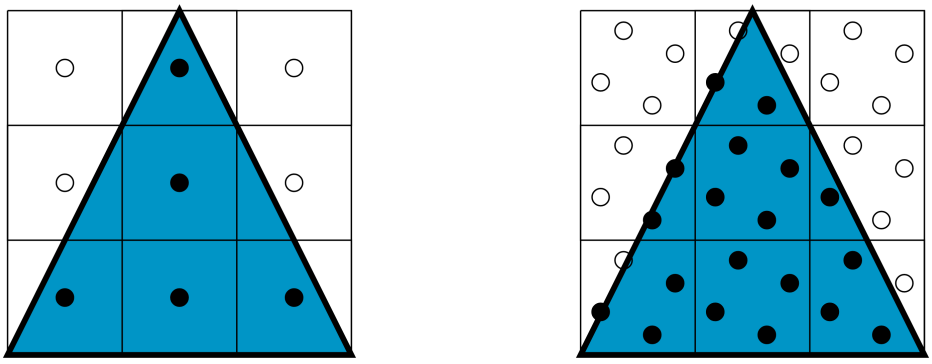
Po transformování a generování trojúhelníků dochází k odebírání trojúhelníků, které se nachází mimo view volume. View volume je ohraničen šesti plochami ve tvaru krychle [1, 5]. Pokud je trojúhelník za plochou, je odebrán. Pokud trojúhelník protíná plochy, je rozložen na více trojúhelníků, tak aby ji neprotínal. Prostor view volume lze zmenšit definováním dalších clip ploch.

Proces může pokračovat perspektivním dělením [1, 5]. To se používá při vykreslování tří rozměrných objektů. Souřadnice x, y a z jsou děleny čtvrtou souřadnicí w. Výsledkem jsou vrcholy v jednotkovém souřadnicovém systému zařízení. V souřadnicovém systému zařízení jsou všechny neodebrané vrcholy v intervalu souřadnic od -1 do 1.

Způsob odebírání trojúhelníků lze dále nastavit i dle jejich orientace [5]. Trojúhelník lze orientovat podle pořadí definování jeho vrcholů (obrázek 2.3). Pořadí vrcholů vyhodnocujeme způsobem po směru hodinových ručiček nebo proti směru hodinových ručiček.

2.2.6 Screen mapping

Trojúhelníky, které nebyly odebrány, jsou přeneseny ze souřadnic zařízení do souřadnic obrazovky. Souřadnice obrazovky se liší podle použitého API (Application Programming Interface). Při použití OpenGL se počátek souřadnic obrazovky nachází v levém dolním rohu obrazovky [1]. Při použití Direct3D se v některých případech počátek nachází v levém horním rohu [1].



Obrázek 2.4: Nalevo bez anti-aliasing, napravo 4 vzorky na pixel.

2.2.7 Rasterizace

Trojúhelník, přímka, či bod je rasterizován. Nejjednodušší způsob rasterizace je vzorkováním pixelů pomocí jednoho vzorku uprostřed pixelu. Pokud se daný vzorek nachází uvnitř tvaru, pak se uvažuje, že daný pixel je uvnitř tvaru [1, 6].

Vzorkování pixelu jedním vzorkem, vytváří velmi zkreslené hrany obrazu. Hrana je velmi zubatá. Pro vyhlazování hran lze použít MSAA (Multi Sample Anti-Aliaisng) [1, 6]. MSAA při rasterizace zvýší počet vzorků pro jeden pixel. Zvýší se tím rozlišení, ale pixel shader je stále volán pouze jednou pro pixel. Na konci vykreslení všech geometrických tvarů se mezi vzorky v pixelu udělá vážený průměr.

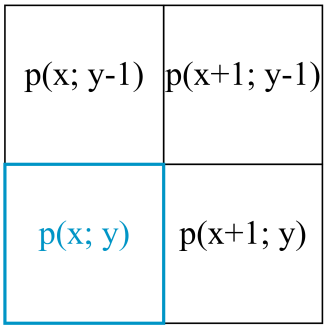
2.2.8 Pixel shader

Po pokrytí geometrického tvaru trojúhelníka body je provedeno nastavení barev bodů. Případně se zde mapují textury na trojúhelník či si zde počítá osvětlení. Tento krok je nazýván rozdílně, dle použitého API. V Direct3D je krok nazýván pixel shader. V OpenGL je krok nazýván fragment shader. V této práci bude použit název pixel shader, protože bude použito API Direct3D.

Tento krok grafické řetězce je programovatelný. Shader je spuštěn pro každý pokrytý pixel nebo vzorek (fragment v OpenGL). Vstupem je vrchol, který je výsledkem interpolace vrcholů trojúhelníka. V případě více vzorků na pixel je interpolace řešena středem pixelu nebo centroidem [6]. Případně je pixel shader pro každý vzorek.

Pro některé texturovací filtry, je třeba gradient některých proměnných [1, 3, 7]. Proto jsou spouštěny tři sousední pixely v pixel shaderu. Přibližný gradient je počítán rozdílem sousedních pixelů.

Výstupem pixel shaderu je typicky barva trojúhelníka. V pixel shaderu lze i upravit hloubku či stencil hodnotu pixelu. Novější API dnes umožňuje provést tzv. alpha test, kde v pixel shadru může být pixel odebrán z procesu, aby se nezobrazil.



$$\frac{\partial p}{\partial x}p(x;y) = p(x+1;y) - p(x;y)$$

$$\frac{\partial p}{\partial y}p(x;y) = p(x;y) - p(x;y-1)$$

Obrázek 2.5: Způsob výpočtu odhadu gradientu v bodě.

2.2.9 Merger

Po pixel shaderu jsou provedeny operace, při kterých se čte a zapisuje do bufferů pro barvu, hloubku (Z-buffer) a stencil hodnotu (stencil buffer). V tomto kroku se definují, jaké testy a jaké operace, které má hardware nad daným vzorkem trojúhelníka a vzorkem již minulého vykreslení provést.

Buffer barvy obsahuje v každé buňce hodnoty RGBA (Red Green Blue Alpha). Buffer hloubky má uloženou vzdálenost neboli hloubku na ose z. Spolu s bufferem hloubky se vytváří i stencil buffer. Stencil hodnota je celé číslo, hloubka bývá float. Buffer hloubky a stencil buffer jsou nepovinné. Buffer je v daném rastru pro každý bod na obrazovce.

Stencil buffer je nepovinný. Stencil hodnoty umožňují hodnotou vytvořit masku a následně ji použít a tím zmenšit oblast vykreslení. Stencil hodnota může být porovnána. Dle výsledku porovnání lze stencil hodnotu přepsat, zvětšit, zmenšit nebo ignorovat. Vytvořená maska stencilu je využita při testu zobrazení bodu a bod případně není vykreslen.

Test hloubky probíhá porovnáním vypočítané hloubky pixelu trojúhelníka a hloubky, která je zapsaná v hloubkovém bufferu. Pokud je menší nebo větší, tak je pixel přepsán novou hloubkou nebo zahozen.

Test stencilu je proveden předem určeným porovnáním stencil hodnoty trojúhelníka se stencil hodnotou již zapsanou v bufferu. Po porovnání je provedena předem definovaná operace mezi těmito hodnotami a následně je zapsána nová hodnota, případně je zahozena.

Pokud pixel projde všemi testy, výsledná barva je smíchána s původní hodnotou dle nastavené rovnice. Hodnota barvy je složena ze čtyř kanálů RGBA. Alpha kanál barvy se často používá jako vyjádření průhlednosti. Alpha hodnota může být využita při MSAA. Při MSAA může být zapnuta technika alpha to cover, která dle velikosti hodnoty vybarví určitý počet vzorků jednoho pixelu.

2.3 Dvourozměrné transformace

Ve vertex shaderu se standardně pracuje s třírozměrnými transformacemi. Mezi zajímavé tituly zabývající se touto problematikou publikují například

Akenine-Möller Tomas a spol. [1] nebo Žára Jiří a spol. [8]. Efektivně lze pracovat i s dvourozměrnými maticemi.

Transformace je možné provádět postupně otočení, zvětšení atd. Tato práce se zabývá transformací posunem, rotací a zvětšení. Pro všechny typy transformací je výhodnější používat matice [8]. Transformace posunem vyžaduje matice třetího řádu. Z tohoto důvodu je tak tento třetí řád použit na všechny typy transformací, jelikož při zpracování dochází k násobení matic. Používá se reprezentace bodů homogenními souřadnicemi $[x_w, y_w, w]$.

Homogenní souřadnice w se nejčastěji volí $w = 1$.

$$\begin{bmatrix} x_w \\ y_w \\ w \end{bmatrix} = \begin{bmatrix} x \cdot w \\ y \cdot w \\ w \end{bmatrix} \tag{2.1}$$

Translace

Transformace posunutím je také nazývána Translací, která posune objekt v hodnotě T_x ve směru osy x a v hodnotě T_y ve směru osy y. V matici je vektor posunutí (T_x, T_y) zapsán na posledním sloupci.

$$\begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ w \end{bmatrix} = \begin{bmatrix} x_w + t_x \\ y_w + t_y \\ w \end{bmatrix} \tag{2.2}$$

Rotace

Rotace neboli otáčení, otočí a posune objekt po kruhové dráze. Rotace se středem v počátečním bodě souřadnicové soustavy lze jednoduše zapsat pomocí matice 2.3.

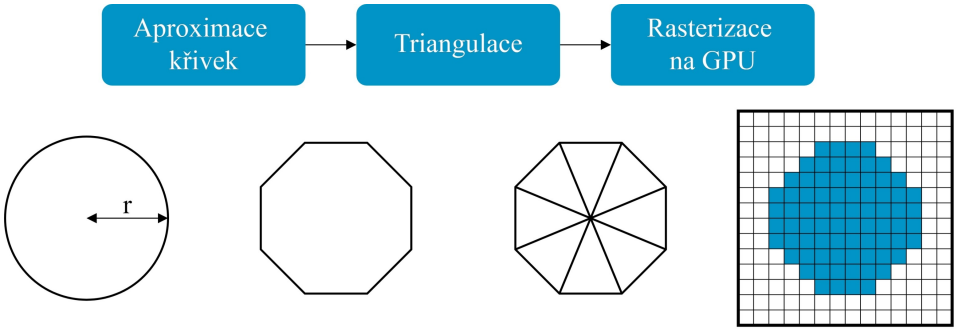
Rotace kolem libovolného bodu lze vyřešit posunutím středového bodu otočením na počátek souřadnicové osy. Následně rotací o daný úhel a inverzí dřívějšího posunutí.

$$\begin{bmatrix} cos(\alpha) & -sin(\alpha) & 0 \\ sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ w \end{bmatrix} = \begin{bmatrix} x_w cos(\alpha) - y_w sin(\alpha) \\ x_w sin(\alpha) + y_w cos(\alpha) \\ w \end{bmatrix} \tag{2.3}$$

Změna měřítka

Transformace zvětšení je také nazýváno Změna měřítka. Principem této transformace je vynásobením souřadnic objektu měřítky S_x a S_y. Pro zvětšení objektu je třeba, aby absolutní hodnoty měřítka byly větší než 1. Pro zmenšení objektu musí být absolutní hodnoty menší než 1.

Při této operaci jsou všechny vrcholy více či méně vzdálené od počátku souřadnicového systému. Jako u rotace i zde lze použitím translace určit vztažný bod změny měřítka.



Obrázek 2.6: Metoda rasterizace aproximace křivek úsečkami.

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ w \end{bmatrix} = \begin{bmatrix} x_w S_x \\ y_w S_y \\ w \end{bmatrix} \tag{2.4}$$

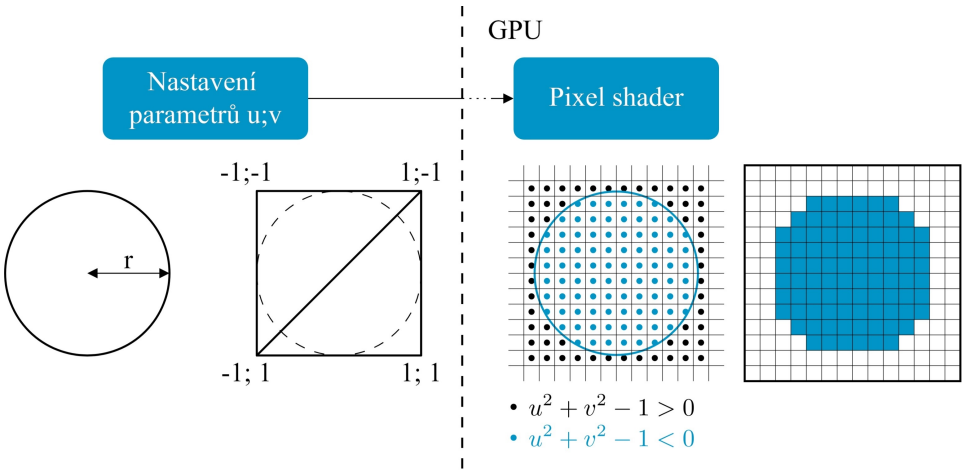
2.4 Způsoby zobrazení křivek

Vektorová grafika je reprezentovaná geometrickými tvary, včetně křivek. Vykreslování křivek pomocí GPU může být realizováno různými způsoby. V této práci budou porovnány dva rozdílné způsoby vykreslování křivek. První způsob vykreslení spočívá v převodu křivek do trojúhelníků tzv. aproximace. Druhý způsob používá rovnice, které určí, zda se daný bod obrazovky nachází vlevo, vpravo nebo na křivce.

2.5 Aproximace křivky

Vykreslování křivek pomocí aproximace bude implementováno knihovnou. Proces aproximace má několik fází. V prvopočátku dochází k aproximaci křivky v lomenou čáru a pokračuje triangulací tvaru uzavřeného lomenou čárou na trojúhelníky. Aproximování křivek úsečkami může být řešeno počítáním několika bodů ležících na křivce. Body křivek jsou spojeny úsečkami a vznikne uzavřený tvar bez křivek. Geometrický tvar je převeden na trojúhelníky tzv. triangulace. Trojúhelníky jsou následně vykresleny pomocí rasterizace na GPU. Proces je vyobrazen na obrázku 2.6.

Způsob vykreslování pomocí aproximace má jednu zřetelnou nevýhodu. Při aproximaci křivky malým počtem úseček, může být při přiblížení viditelná chyba aproximace. Velkou chybu aproximace, lze vyřešit větším počtem úseček. Větší počet úseček může mít za následek pomalejší vykreslování. Proto se často proces aproximace a triangulace opakuje.



Obrázek 2.7: Metoda rasterizace křivek v prostoru pixel shaderu.

2.6 Implicitní křivky

Kroky vykreslování křivek za pomocí implicitních rovnic jsou zobrazeny na obrázku 2.7.

Křivka je nejprve převedena do implicitní rovnice a je provedena jednoduchá triangulace. Poté jsou trojúhelníky rasterizovány na GPU použitím vlastního pixel shaderu. V pixel shaderu je počítána implicitní rovnice, která určí, jestli se pixel nachází vně nebo uvnitř geometrického tvaru. Na obrázku 2.7 jsou vidět jednotlivé kroky pro rasterizaci kruhu.

Implicitní rovnice křivky je ve tvaru rovnice 2.5. Pro každou křivku je napsaná jiná funkce f. Dosazením souřadnice bodu do rovnice se zjistí, jestli se daný bod nachází na křivce. Znaménko vypočtené funkce f určí, zda se bod nachází uvnitř nebo vně geometrického tvaru.

$$f(x, y) = 0 \tag{2.5}$$

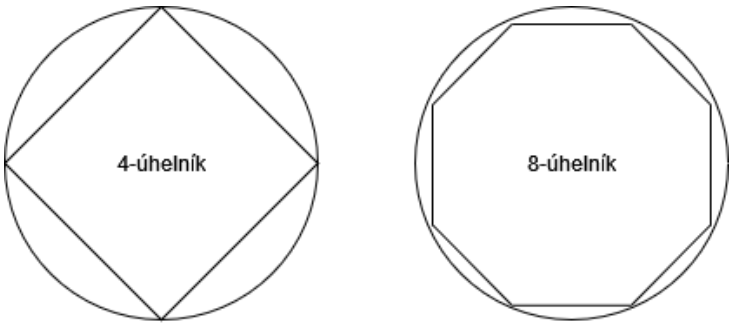
Vektorový obrázek typu SVG (Scalable Vector Graphics) umožňuje vykreslení kruhů, elips a Bézierových křivek. Bézierovy křivky jsou používány k definování cesty. Oblast uzavřená cestou může být vybarvena. Takto uzavřenou cestu rozdělíme na křivky a mnohoúhelníky.

2.6.1 Kruh

Kruh je reprezentován pomocí implicitní nerovnice 2.6, kterou lze počítat v prostoru pixel shaderu. Kruh je tvořen se středem v bodě $[x_0, y_0]$ a poloměrem r. Pak všechny body $[x, y]$, které jsou v kruhu, budou vzdálené maximálně poloměru r od středu kruhu. Implicitní nerovnice pro kruh je ve tvaru:

$$(x - x_0)^2 + (y - y_0)^2 \leq r^2 \tag{2.6}$$

Kružnici lze popsat parametrickou rovnicí 2.7 se středem v bodě $[x_0, y_0]$. Parametrické rovnice zajišťují vypočet vrcholů při případné aproximaci kruhu



Obrázek 2.8: Aproximace kružnice pomocí pravidelných mnohoúhelníků.

pomocí pravidelného n-úhelníku. Přesnost aproximace je možné ovlivnit počtem vrcholů n-úhelníku (viz obrázek 2.8). Pravidelný n-úhelník je konvexní tvar, a proto jsou snadno triangulovatelné. Kruh je možné také aproximovat pomocí uzavřeného tvaru Bézierovými křivkami [9].

$$\begin{aligned} x &= r \cdot cos(t) + x_0 \\ y &= r \cdot sin(t) + y_0 \\ 0 &\leq t \leq 2\pi \end{aligned} \tag{2.7}$$

2.6.2 Elipsa

Elipsa je reprezentovaná pomocí implicitní nerovnice 2.8 podobně jako u kruhu. Elipsa je tvořena se středem v bodě $[x_0, y_0]$ s délkou její hlavní poloosy a a vedlejší poloosy b. Aproximace elipsy je provedena stejným způsobem jako u kruhu rovnicí 2.9.

$$\frac{(x - x_0)^2}{a^2} + \frac{(y - y_0)^2}{b^2} \leq 1 \tag{2.8}$$

$$\begin{aligned} x &= a \cdot cos(t) + x_0 \\ y &= b \cdot sin(t) + y_0 \\ 0 &\leq t \leq 2\pi \end{aligned} \tag{2.9}$$

2.6.3 Bézierova křivka

Ve vektorové grafice se kromě kruhů a elips často používají Bézierovy křivky. Bézierovy křivky jsou používány při popisu cesty. Pokud je cesta uzavřena, může být uzavřená oblast vyplněna barvou.

Parametrická rovnice Bézierovy křivky

Bézierova křivka je definována pomocí Bernsteinových polynomů, ale lze také počítat pomocí De Casteljauova algoritmu. De Casteljauův algoritmus počítá výsledný bod lineární interpolací rekurzivně. Řídící body Bézierovy křivky řádu n jsou $P_i^{(0)}; i = 0, ..., n$.

$$\begin{aligned} P_i^{(j)} &= (1-t)P_i^{(j-1)} + tP_{i+1}^{(j-1)}; i = 0, ..., n-j; j = 1, ..., n \\ C(t) &= P_0^{(n)} \end{aligned} \tag{2.10}$$

Bézierovu křivku lze také vyjádřit parametricky pomocí Bernsteinova polynomu. Na začátku jsou spočítány jednotlivé Bernsteinovy bázové polynomy.

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{2.11}$$

$\binom{n}{i}$ je Binomický koeficient $\frac{n!}{i!(n-i)!}$.

Lineární kombinací Bernsteinových bázových polynomů s řídícími body je získáno parametrické vyjádření Bézierovy křivky.

$$C(t) = \sum_{i=0}^{n} P_i b_{i,n}(t) = \begin{bmatrix} b_{0,n} & ... & b_{n,n} \end{bmatrix} \begin{bmatrix} P_0 \\ ... \\ P_n \end{bmatrix} \tag{2.12}$$

Parametrická rovnice je rozepsána pro křivku druhého (viz rovnice 2.13) a třetího řádu (viz rovnice 2.14). Bézierova křivka n-tého řádu má vždy $n+1$ řídících bodů. U Bézierových křivek je možné křivky nižšího řádu vyjádřit pomocí křivek vyššího řádu. Toto tvrzení platí pouze jednostranně. Křivky vyššího řádu mohou být pouze aproximovány křivkami nižšího řádu.

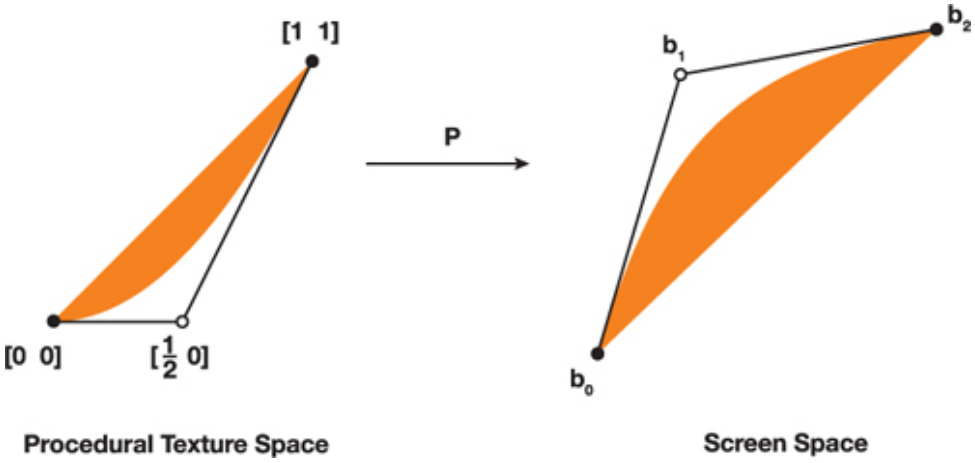
$$\begin{aligned} M_2 &= \begin{bmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{bmatrix} \\ C(t) &= \begin{bmatrix} 1 & t & t^2 \end{bmatrix} \cdot M_2 \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \end{bmatrix} \end{aligned} \tag{2.13}$$

$$\begin{aligned} M_3 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \\ C(t) &= \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \cdot M_3 \cdot \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \end{aligned} \tag{2.14}$$

Implicitní rovnice Bézierovy křivky

Pro zobrazení Bézierovy křivky v prostoru pixel shaderu je potřeba implicitního tvaru rovnice křivky (viz. rovnice 2.5). Implicitní rovnici Bézierovy křivky lze získat eliminováním parametru t z parametrické rovnice 2.13.

Po eliminaci parametru t je rovnice velmi dlouhá a nepraktická již pro Bézierovu křivku druhého řádu. Při použití implicitní rovnice Bézierovy kvadratické křivky 2.15 v prostoru pixel shaderu je možné provést substituci



Obrázek 2.9: Mapování canonické křivky (vlevo) na bézierovu křivku (vpravo) [11].

a některé části z nich předpočítat před vykreslením. V příloze B se nachází implicitní rovnice kvadratické Bézierovy křivky a v příloze C se nachází implicitní rovnice kubické Bézierovy křivky. Implicitní rovnice v příloze B a C byly vypočteny využitím kalkulačky. Implicitní rovnice Bézierových křivek vyšších řádů jsou obsáhlé, princip sestavení je totožný.

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0 \tag{2.15}$$

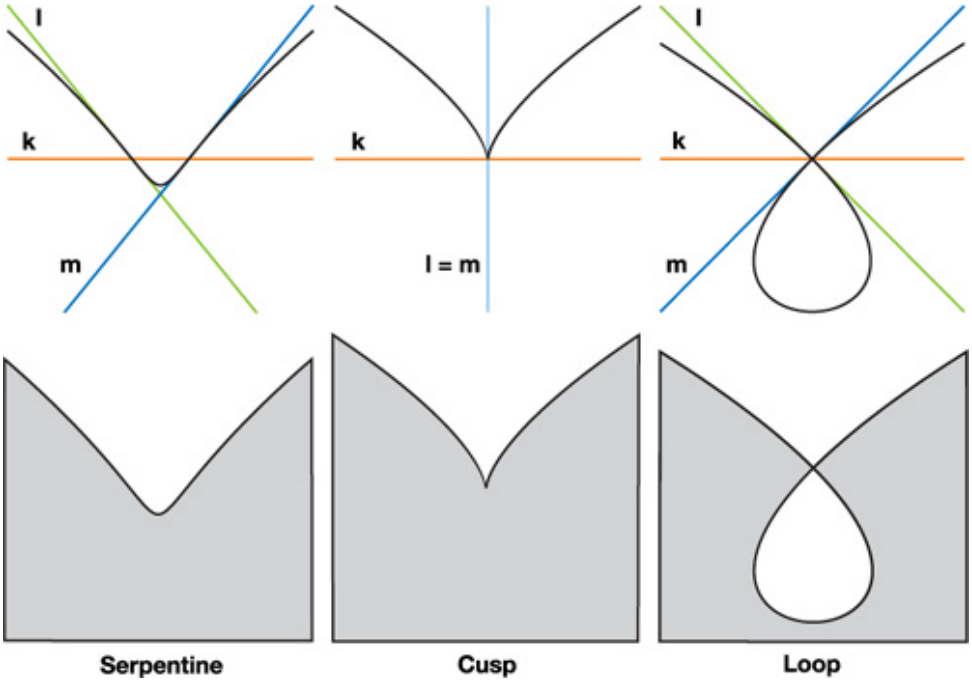
Vykreselní Bézierovy kvadratické křivky

Vykreslování Bézierovy kvadratické křivky pomocí implicitní rovnice 2.15 není praktické. Implicitní rovnice obsahuje příliš mnoho parametrů, které by bylo třeba předat do pixel shaderu. Při dlouhém výpočtu v pixel shaderu může být znatelná ztráta ve výkonu vykreslování.

Pomocí několika matematických úprav může být implicitní rovnice zjednodušena. Jednou z matematických úprav je výpočet transformace na křivku, která má mnohem jednodušší implicitní rovnici 2.16. Transformace do této křivky se vykoná skrz mapování textur (viz obrázek 2.9). Jednotlivým bodům Bézierovy křivky se přiřadí texturovací souřadnice $[0, 0]$, $[\frac{1}{2}, 0]$ a $[1, 1]$. Rovnice, která se následně spočítá ve fragment shaderu, je jednoduchá. Znaménko levé strany rovnice určuje, jestli se daný pixel nachází v oblasti, kde je křivka konvexní nebo konkávní [10]. V případě nulové hodnoty se pixel se nachází na křivce.

$$f(u, v) = u^2 - v = 0 \tag{2.16}$$

Trojúhelník ohraničující Bézierovu křivku může překrývat jinou Bézierovu křivku. V takovém případě je pravděpodobné, že bude zbarvena část oblasti, která zbarvena být nemá. Pro tyto případy je třeba křivky rozdělit, tak aby se jejich trojúhelníky nepřekrývaly.



Obrázek 2.10: Canonické tvary Bézierovy kubické křivky [11].

Vedle implicitní rovnice lze aproximovat Bézierovu křivku také pomocí úseček. Jednotlivé body úseček jsou spočítány pomocí parametrické rovnice Bézierovy křivky nebo De Casteljauova algoritmu. Kvalitu výsledku je možné ovlivňovat zvyšováním počtu úseček. Tímto způsobem se zvyšuje výkon potřebný k vykreslení Bézierových křivek. Po provedení transformací křivky, jako je přiblížení, je často nutno zvýšit počet úseček, aby chyba nebyla tak rozeznatelná.

Jiným řešením je Bézierovu křivku předkreslit do textury, která následně bude zobrazena. Předkreslení je provedeno v daném rozlišení. To má za důsledek, že při dodatečném přiblížení se mohou projevit nedokonalosti.

Vykreslení Bézierovy kubické křivky

Kubické křivky jsou často používané. Lze jimi reprezentovat mnohem více tvarů a zároveň nejsou tolik složité, jako křivky vyšších řádů. Bézierovy kubické křivky mají tři canonické tvary: loop, cusp a serpentinu [12]. Canonické tvary Bézierovy kubické křivky jsou na obrázku 2.10. Tyto tři tvary lze reprezentovat použití jedné rovnice 2.17 [13].

$$k^3 - lmn = 0 \tag{2.17}$$

Algoritmus, který představili Loop Charles a Blinn Jim [10] pro vykreslení kubických křivek v prostoru pixel shaderu je následovný. Nejprve je křivka klasifikována. Dle klasifikace jsou spočteny texturovací souřadnice vrcholů. Následně jsou texturovací souřadnice interpolovány grafickým procesorem a v

pixel shaderu je spočtena rovnice 2.17. Dle znaménka je určeno, která strana má být vybarvena.

Klasifikace kubické křivky může být řešena spočtením parametru, kde se nachází inflexní body křivky [12]. Inflexní bod křivky je bod, kde křivka mění zakřivení. Klasifikace se následně pozná dle počtu kořenů rovnice 2.18. Zde je parametr rovnice $[t, s]$. Parametr parametrické rovnice je $\frac{t}{s}$. Tím to způsobem je docíleno, toho, že inflexní bod, který se nachází v nekonečnu má parametry $[1, 0]$. Parametry d_0, d_1, d_2 a d_3 jsou determinanty 3x3 matic souřadnic bodů kanonické křivky [12]. Souřadnice bodů kanonické křivky jsou získány transformováním bodů Bézierovy křivky maticí M z rovnice 2.14. Determinanty lze zjednodušit použitím vektorového a skalárního součin (viz rovnice 2.19).

$$I(t,s) = d_0 t^3 - 3d_1 t^2 s + 3d_2 t s^2 - d_3 s^3 \tag{2.18}$$

$$\begin{aligned} d_0 &= -p_3 \cdot p_2 \times p_1 \\ d_1 &= p_3 \cdot p_2 \times p_0 \\ d_2 &= -p_3 \cdot p_1 \times p_0 \\ d_3 &= p_2 \cdot p_1 \times p_0 \end{aligned} \tag{2.19}$$

V této práci budou vykreslovány křivky s řídícími body, které budou mít hodnotu souřadnice $w = 1$. Loop Charles a Blinn Jim [10] tvrdí, že se tím zjednoduší rovnice 2.17 ($n = 1$) a rovnice 2.18 ($d_0 = 0$). Zjednodušení rovnice 2.18 se zjednušila také klasifikace, která je popsána v tabulce 2.1.

tvar	kořeny	podmínky
serpentine	3 reálné kořeny	$d_1 \neq 0 \wedge 3d_2^2 - 4d_1 d_3 > 0$
cusp	2 reálné kořeny (1 dvojnásobný)	$d_1 \neq 0 \wedge 3d_2^2 - 4d_1 d_3 = 0$
loop	1 reálný kořen a 2 komplexní kořeny	$d_1 \neq 0 \wedge 3d_2^2 - 4d_1 d_3 < 0$
kvadratická křivka	trojnásobný reálný kořen	$d_1 = d_2 \wedge d_3 \neq 0$

Tabulka 2.1: Tabulka klasifikací kubických křivek.

Pro každý typ je vypočtena matice F. K získání texturovacích souřadnic je třeba matici F vynásobit inversní maticí M. V každém řádku výsledné matice se nachází souřadnice k, l a m pro každý vrchol.

Serpentine

Jedním z klasifikovaných tvarů Bézierovi kubické křivky je serpentine nebo cusp s inflexním bodem v nekonečnu. Tyto stavy lze spojit. V případě serpentine je výpočet matice F následovný.

$$\begin{aligned} (t_l, s_l) &= (d_2 + \tfrac{1}{\sqrt{3}}\sqrt{3d_2^2 - 4d_1 d_3}, 2d_1) \\ (t_m, s_m) &= (d_2 - \tfrac{1}{\sqrt{3}}\sqrt{3d_2^2 - 4d_1 d_3}, 2d_1) \\ (t_n, s_n) &= (1, 0) \end{aligned} \tag{2.20}$$

$$F = \begin{bmatrix} t_l t_m & t_l^3 & t_m^3 & 1 \\ -s_m t_l - s_l t_m & -3 s_l t_l^2 & -3 s_m t_m^2 & 0 \\ s_l s_m & 3 s_l^2 t_l & 3 s_m^2 t_m & 0 \\ 0 & -s_l^3 & -3 s_m^3 & 0 \end{bmatrix} \tag{2.21}$$

Pokud je d_1 záporné, musí být znaménka parametru k a l převrácena. Tímto převrácením se zajistí, aby se vyplněná část nacházela vždy na levé straně křivky [10].

Cusp

Dalším tvarem Bézierovi kubické křivky je cusp. Speciálně cusp s vrcholem v nekonečnu. Tento tvar křivky má dvojnásobný kořen v nekonečnu. Výpočet matice F je shodný jako v případě serpentine, ale parametry t a s jsou jiné. U tohoto tvaru křivky není třeba nikdy převracet parametry k a l [10].

$$\begin{aligned} (t_l, s_l) &= (d_3, 3d_2) \\ (t_m, s_m) &= (1, 0) \\ (t_n, s_n) &= (1, 0) \end{aligned} \tag{2.22}$$

Loop

Poslední tvar Bézierové kubické křivky je loop. U tohoto typu je třeba zjistit, jestli se některý z bodů $\frac{td}{sd}$ nebo $\frac{te}{se}$ nenachází v intervalu $[0, 1]$ [10]. Pokud by se nacházel, mohlo by to mít za následek chybné vykreslení nebo chybné zjištění, jestli se daný bod nachází na levé nebo pravé straně křivky. Proto je třeba v takovém případě křivku rozdělit na více Bézierových kubických křivek tvaru loop.

$$\begin{aligned} (t_d, s_d) &= (d_2 + \tfrac{1}{\sqrt{3}}\sqrt{4d_1 d_3 - 3d_2^2}, 2d_1) \\ (t_e, s_e) &= (d_2 - \tfrac{1}{\sqrt{3}}\sqrt{4d_1 d_3 - 3d_2^2}, 2d_1) \end{aligned} \tag{2.23}$$

Následně se dle $d_1 H(\cdot)$ zjistí, jestli se nemá převrátit znaménko parametrů k a l. Výpočet matice F je následovný:

$$F = \begin{bmatrix} t_d t_e & t_d^2 t_e & t_d^2 t_e & 1 \\ -s_e t_d - s_d t_e & -s_e t_d^2 - 2 s_d t_e t_d & -s_d t_e^2 - 2 s_e t_d t_e & 0 \\ s_d s_e & t_e s_d^2 + 2 s_e t_d s_d & t_d s_e^2 + 2 s_d t_e s_e & 0 \\ 0 & -s_d^2 s_e & -s_d s_e^2 & 0 \end{bmatrix} \tag{2.24}$$

Kvadratická

Při počítání kubické křivky degradované na kvadratickou je třeba rovnicí 2.17, která je počítána v pixel shaderu, vyjádřit kvadratickou rovnici 2.16. Dosažení kvadratické rovnice je řešeno dosazení do parametru m a k totožných hodnot. Matematickými úpravami lze následně vyjádřit kvadratickou křivku.

Způsob dosazení parametrů k, m a l a následný výpočet je podobným jako u kvadratické Bézierovy křivky.

$$\begin{aligned} 0 &= k^3 - lm \quad k = m \\ 0 &= k^3 - lk \\ 0 &= k(k^2 - l) \end{aligned} \tag{2.25}$$

2.6.4 Anti-aliasing

Anti-aliasing lze řešit několika způsoby. Loop Charles a Blinn Jim [10] ukázali způsob použití vzdálenosti se znaménkem nebo použitím dalších trojúhelníků za hranou křivky. Knihovna Direct2D používá menší trojúhelníky na hraně aproximovaného tvaru. Je použit výpočet interpolace přes hranu. Tímto způsobem se lze vyhnout MSAA.

Nejjednodušší způsob je použití signed distance. Použitím gradientu v prostoru pixel shaderu. Získáme gradient křivky v pixelu. Rovnice 2.26 nám umožní zjistit, jak daleko se pixel nachází od hrany křivky. Funkce f v rovnici reprezentuje levou stranu implicitní rovnice křivky. Gradient je počítán parciální derivací osy x a y. Lze získat aproximaci gradientu, která je zmíněna v sekci 2.2.8 Pixel shader. Následně může být namíchána průhlednost dle vzdálenosti od hrany.

$$d(x,y) = \frac{f(x,y)}{\|\nabla f(x,y)\|} = \frac{f(x,y)}{\sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}} \tag{2.26}$$

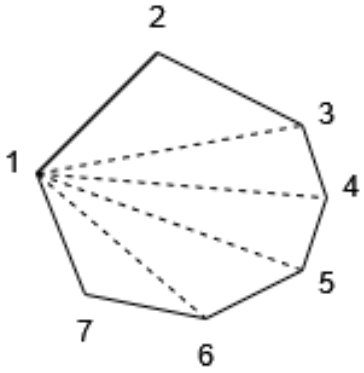
2.7 Triangulace

Vektorové obrázky obsahují kromě křivek, také obdélníky a mnohoúhelníky. Tyto tvary je nutné rozdělit do trojúhelníků, aby je bylo možné zobrazit pomocí rasterizace na GPU. Proces rozdělení mnohoúhelníků na trojúhelníky se nazývá triangulace. Mnohoúhelník může být triangulován různými algoritmy dle jeho typu. Pokud je mnohoúhelník konvexní triangulace je primitivní. Pokud mnohoúhelník není konvexní, lze jej triangulovat pomocí ear clipping algoritmu nebo rozdělit na monotónní mnohoúhelníky a následně triangulovat monotónní mnohoúhelníky [14, 15].

2.7.1 Konvexní mnohoúhelník

Nejjednodušší je triangulace konvexního mnohoúhelníku. Konvexní mnohoúhelník je mnohoúhelník, kde všechny přímky, které prochází skrz mnohoúhelník, mají přesně dva průsečíky. Další možností, jak konvexní mnohoúhelník rozeznat, je pomocí velikosti vnitřních úhlů vrcholů, které nesmí přesáhnout 180 stupňů.

Možností, jak triangulovat konvexní mnohoúhelník, je mnoho. Každý z nich dosáhne různých parametrů trojúhelníků. Pro reprezentování mnohoúhelníku



Obrázek 2.11: Triangulace konvexního sedmiúhelníku

trojúhelníky stačí vybrat jeden libovolných vrchol mnohoúhelníku. Z toho bodu vedeme všechny diagonály. Například triangulace sedmiúhelníku může vypadat způsobem zobrazeným na obrázku 2.11.

2.7.2 Ear clipping triangulace

Pokud mnohoúhelník není konvexní, jeho triangulace může být řešena Ear clipping triangulací. Ear clipping triangulace funguje na principu postupném odstranění trojúhelníků nazvaných uši [16, 17, 18].

Algoritmus využívá Two Ear theorem, který říká, že každý jednoduchý mnohoúhelník s více než třemi vrcholy má alespoň dvě uši. Uši jsou tři sousední vrcholy P_i, P_{i+1} a P_{i+2}, pro které platí následná pravidla. Vnitřní úhel mnohoúhelníku při vrcholu P_{i+1} je menší než 180 stupňů. Dále musí platit, že hrana mezi vrcholy P_i a P_{i+2} se nachází uvnitř mnohoúhelníku.

Nalezené ucho je jedním z výsledných trojúhelníků. Tento trojúhelník se odstraní odebráním vrcholu P_{i+1}. Proces se opakuje, dokud mnohoúhelník má více než 3 vrcholy.

Algoritmus lze rozšířit pro řešení mnohoúhelníků s dírami. Nicméně pro vykreslování mnohoúhelníků s dírami bylo v této práci využito vykreslování mnohoúhelníků pomocí stencil bufferu.

2.7.3 Vykreslování mnohúhelníka pomocí stencil bufferu

Vykreslování pomocí stencil bufferu funguje na principu, vytvoření šablony (pokrytí pixelů) a následné vykreslení tvaru s barvou.

Prvním krokem je jednoduchá triangulace, kdy je zvolen libovolný bod a směr triangulace. Následně je sestaven trojúhelník ze zvoleného vrcholu a dvou sousedních vrcholů ve zvoleném směru. Další trojúhelník je sestaven ze zvoleného vrcholu, vrcholu předchozího trojúhelníka a jeho souseda ve směru triangulace.

Trojúhelníky jsou následně vykresleny do stencil bufferu s nastavenou operací XOR (Exclusive or) nebo sčítání odčítání dle orientace trojúhelníka.

Výsledkem je stencil buffer s nenulovými hodnotami v pixelech, které mnohoúhelník pokrývá. V následujícím kroku je mnohoúhelník s použitím stencil bufferu vykreslen na obrazovku.

Tento způsob vykreslení mnohoúhelníku, lze použít při vykreslování uzavřeného tvaru s křivkami [19]. Mnohoúhelník a křivky jsou vykresleny do stencil bufferu. Následně je celý tvar vykreslen na obrazovku. Pro řešení antialiasingu lze použít tzv. alpha to cover [19].

Tato metoda bude použita pro vykreslení uzavřených tvarů s křivkami. Metoda byla zvolena kvůli její jednoduchosti na implementaci.

Kapitola 3
Implementace

Tato kapitola popisuje použitý programovací jazyk, použité knihovny a SDK (Software Development Kit). Pokračuje popisem dílčích částí implementace programu. Při implementaci se v prvé řadě bylo třeba seznámit s prostředím. Dále bylo zapotřebí vyřešit zobrazení vektorové grafiky tradičním způsobem. Následovalo vykreslení kruhu, čtverce a uzavřené cesty. Nesmíme však opomenout ovládání a načítání samotných SVG obrázků, které byly v průběhu implementace vylepšovány.

3.1 Programovací jazyk

Byl zvolen programovací jazyk C++ standardu 20. V programovacím jazyce C++ lze dobře řídit uvolňování paměti. Programový kód v tomto jazyce se překládá přímo do instrukcí procesoru. Program pak může být rychlejší.

Kompilátorem jazyka C++ byl zvolen MSVC (Microsoft Visual C++). Zvolení kompilátoru MSVC bylo z důvodu použití kolekce knihoven Windows SDK.

3.2 Použité knihovny

Direct3D 11

K rychlému vykreslování grafických objektů byla zvolena technologie DirectX s knihovnou Direct3D ve verzi 11, která umožňuje komunikaci s grafickým hardwarem. Na technologii je přistupováno přes Windows SDK, který obsahuje programové knihovny pro práci v Direct3D ve verzi 11. Součástí SDK je také knihovna pro kompilaci programovatelných shaderů. Programovatelné shadery jsou tvořeny v jazyce HLSL (High-Level Shader Language).

Direct2D

Pro vykreslování křivek pomocí jejich aproximace a triangulace je použita knihovna Direct2D. Vykreslení probíhá buď softwarově na procesoru počítače nebo hardwarově na GPU s použitím Direct3D. Pro všechna testování bylo vhodné zajistit stejné podmínky neboli stejnou grafickou knihovnu (Direct3D).

Od aktualizace Windows 10 Creators Update dovede knihovna načítat a vykreslovat SVG obrázky.

Windows SDK

Knihovny DirectX jsou součástí Windows SDK. Z Windows SDK jsou použity další knihovny k obsluze okna programu a dalším funkcím.

GLM (OpenGL Mathemantics)

Knihovna GLM nám pomůže provádět výpočty, které byly nutné provést na CPU (Central Processing Unit). Mezi ně patří například výpočet matice pro transformaci bodů ve vertex shaderu.

RapidXml

SVG soubory jsou ve formátu XML (Extensible Markup Language). Načítání a parsování XML bylo zajištěno knihovnou RapidXml.

3.3 Sestavení a spuštění programu

Pro sestavení programu byl zvolen program CMake a kompilátor MSVC. Všechny C++ zdrojové soubory jsou přidány pomocí příkazu `file(GLOB_RECURSE ...)`. Zdrojové soubory HLSL jsou konvertovány pomocí CMake skriptů do C++ zdrojových souborů. Pro spuštění programu, tedy není nutné řešit načítání shader souborů.

Spustit program lze dvojklikem nebo přes možnost otevřít v programu nad souborem vektorového obrázku. Spuštění skriptů pro měření rychlosti vykreslování lze zadáním parametrů programu přes příkazový řádek. Parametrem *--help* lze zjistit nápovědu programu.

Program při otevření spustí dotazové okno, kde uživatel zvolí grafický adaptér pro vykreslování. Dalším dotazem zvolí vektorový obrázek, který má být vykreslován.

3.4 Řešení načtení shaderů programu

Spouštění programu ve vývojovém prostředí a mimo něj je program spuštěn v jiném adresáři. Tento problém byl vyřešen přesunem shaderů do zdrojového kódu C++.

Vložení zdrojových kódů shaderu do stringových konstant C++ nebylo výhodné. Při editaci zdrojových kódů shaderu nebyla vývojovým prostředím podporována nápověda. Neustálé kopírování změn kódu shaderu do stringových konstant, nebylo také praktické. Někdy se stávalo, že některé zdrojové kódy byly opomenuty a nebyly zkopírovány.

Rozhodl jsem se vytvořit CMake skripty, které automaticky převedou zdrojové soubory HLSL do konstant C++ kódu. Soubor může být načten

binárně a vložen do bytového pole nebo načten jako text a vložen jako Raw String Literal. Při použití Raw String Literal je použita vlastní sekvence znaků pro naznačení začátku a ukončení textu. Pro způsob vložení souboru do zdrojového souboru bylo zvoleno bytové pole. Při vkládání souboru do bytového pole je potřeba soubor načíst binárně. Tento způsob vložení souboru do C++ je vhodný pro textové a binární soubory. V budoucnu tedy tento skript bude možné použít i pro binární soubory.

Soubor bylo třeba načíst a převést do formátu inicializace bytového pole v C++. Principem převodu je načtení souboru v hexadecimálním formátu a následná úprava hexadecimálních dvojic. Načtení souboru proběhlo použitím funkce *file(READ ... HEX)*. Dále byla provedena úprava spuštěním funkce *string(REGEX REPLACE ...)*.

Název C++ konstanty byl vytvořen z názvu souboru pomocí funkce *string(MAKE_C_IDENTIFIER ...)*.

```
# zjednodušená verze implementace

string(MAKE_C_IDENTIFIER  ${hlsl_file_name} id)

# načtení souboru hexadecimálně formátu
file(READ ${hlsl_file_name} content HEX)

string(REGEX REPLACE # $\label{code:hlsl_regex}$
    "([0-9a-f][0-9a-f])" "0x\\1, "
    content ${content}
)

# bytové pole
file(APPEND ${cpp_file_name} "const char ${id}_data[]{\n")
file(APPEND ${cpp_file_name} "    ${content}0x00\n")
file(APPEND ${cpp_file_name} "};\n")

# velikost bytového pole
file(APPEND ${cpp_file_name}
    "const long ${id}_size = sizeof(${id}_data) - 1;\n"
)
```

Program je možné úspěšně spustit v jakémkoliv adresáři. Není ani podstatné, kde se nachází shader soubory programu. Shader soubory programu jsou součástí programu.

3.5 Grafický debugger

Významnou část implementace probíhalo opravování chyb. Nejsilnějším nástrojem se zde projevil grafický debugger. V grafickém debuggeru byla nalezena většina chyb vykreslení. Jako dobrá prevence před chybami se ukázalo nastavení celého grafického řetězce před spuštěním rasterizace. Velmi užitečné se ukázalo nastavení jmen objektům Direct3D. Název objektu lze pak vidět

v grafickém debuggeru. Nastavení jména Direct3D objektu lze pomocí funkce *SetPrivateData(WKPDID_D3DDebugObjectName, ...)*.

Některé grafické debuggery nedokázaly spustit program kompilovaný pro 64bitový systém. Bylo nutné změnit konfiguraci a kompilovat program pro 32bitový systém. Například Visual Studio Graphics Debugger nedokázal spustit program pro 64bitový systém.

Při testování sestavení programu pro spuštění na jiném počítači, je třeba nezapomenout odebrat většinu debuggovacích nástrojů. Některé počítače nemusí mít nainstalované SDK pro debuggování programu.

3.6 Vykreslení pomocí knihovny Direct2D

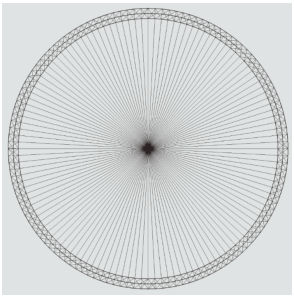
Pomocí knihovny Direct2D lze vykreslení křivky provést převedením křivky pomocí aproximace na úsečky a vykreslit. Knihovna Direct2D umožňuje takto vykreslit geometrické tvary například úsečku a kruh. Bližším prozkoumáním knihovny a porovnáním se zadáním úlohy byla nalezena metoda *ID2D1DeviceContext5::DrawSvgDoucment*. Metoda dělá přesně postup, který bychom jinak museli programovat od parsování SVG, převodu křivek na úsečky a vykreslení. Cílem nebylo programovat metodu vykreslení pomocí aproximace, a proto byla zvolena pro tuto metodu vykreslování křivek funkce z dané knihovny.

Ověření, že metoda *ID2D1DeviceContext5::DrawSvgDoucment* vykresluje daným způsobem, bylo provedeno pokusy naprogramováním vykreslení základních geometrických tvarů. Kontrolovalo se, jak jsou jednotlivé grafické prvky rasterizovány. Bylo zde vidět sestavení grafického prvku do dílčích trojúhelníků.

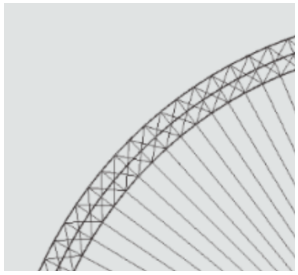
Pokusem vykreslení kruhu v SVG bylo vidět, že před rasterizací na GPU byl kruh aproximován úsečkami a triangulován. Po zvětšení kruhu byla vidět menší chyba aproximace. V instrukcích pixel shaderu, nebyla počítána žádná implicitní rovnice. Antialiasing byl řešen pomocí malých trojúhelníků, které se nacházely na hraně geometrického tvaru. Pomocí interpolace barev vykreslených trojúhelníků na hraně geometrického tvaru bylo řešeno vyhlazení hran geometrického tvaru (viz obrázek 3.1).

Při pokusu vykreslení uzavřeného tvaru s Bézierovými křivkami uvedeného v SVG byla rasterizace na GPU provedena do tzv. cover mapy. Cover mapa je mapa pixelů, které jsou pokryty vykreslovaným tvarem. Detailnějším nahlédnutím na rasterizaci tvaru do cover mapy v grafickém debuggeru bylo vidět velké množství trojúhelníků (viz obrázek 3.2). Trojúhelníky tvořily aproximovaný tvar v SVG. Při náhledu do instrukcí pixel shaderu nebyla počítána žádná implicitní rovnice. Bylo provedeno další vykreslení pomocí GPU, kde byl rasterizovaný tvar v cover mapě překreslen do výsledného obrazu. Zde byl řešen antialiasing použitím texturovacích filtrů při překreslování do výsledného obrazu.

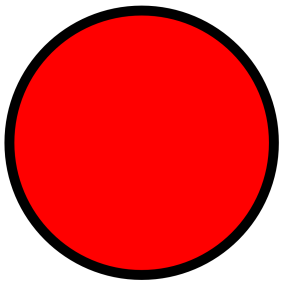
V některých případech nebylo možné grafický debugger spustit či nahlédnout na způsob rasterizace. Důvodem byl způsob vytvoření objektu *ID2D1DeviceContext5*. Při vytvoření objektu z objektu *ID3D11Device* ná-



(a) : Triangulovaný tvar z grafického debuggeru

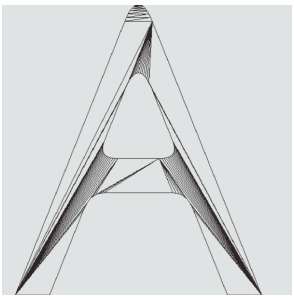


(b) : Přiblížení triangulovaného tvaru



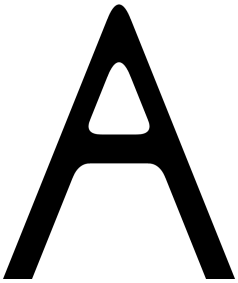
(c) : Výsledek vykreslení

Obrázek 3.1: Vykreslení kruhu pomocí knihovny Direct2D.



(a) : Triangulovaný tvar z grafického debuggeru

(b) : Vykreslená cover mapa



(c) : Výsledek vykreslení

Obrázek 3.2: Vykreslení geometrického tvaru ve tvaru písmene "A"použitím knihovny Direct2D.

sledujícím způsobem v programu, bylo možné spustit grafický debugger a nahlédnout na způsob vykreslení.

1. Konverze *ID3D11Device* na *IDXGIDevice*.
2. Vytvoření *ID2D1Device* pomocí metody *IDXGIFactory5::CreateDevice*
3. vytvoření *ID2D1DeviceContext* pomocí metody *ID2D1Device::CreateDeviceContext*.

Dle použitého typu kontextu Direct2D je obrázek rozdílně umístěn na obrazovce. Při použití kontextu, který je vytvořen z okna programu, při různých změnách nastavení operačního systému, je obrázek umístěn různě. Bylo vypozorováno, že se to děje minimálně změnou parametru zvětšení ikon a textu.

Nikde nebylo zjištěno použití implicitních křivek k vykreslení. Metoda *ID2D1DeviceContext5::DrawSvgDoucment* byla použita pro vykreslení vektorových obrázků způsobem aproximace křivek úsečkami.

3.7 Implementace vykreslení kruhu

Vykreslení kruhu s vyhlazením hran je řešeno v souboru pixel shaderu *shaders/circle-aa.hlsl*. Kruh je obalen do dvou trojúhelníků. Oblasti trojúhelníků, které nepatří do kruhu, jsou v prostoru pixel shaderu odebrány.

Vrcholy trojúhelníků obsahují texturovací souřadnice. Texturové souřadnice jsou v souřadnicovém systému, kde počátek souřadnicového systému se nachází uprostřed kruhu.

Do pixel shaderu je předávána pomocí constant bufferu barva výplně a barva obrysu kruhu. Také je předán poloměr kruhu a poloměr kružnice, oddělující výplň od okraje kruhu.

V prostoru pixel shaderu je spočítána vzdálenost texturovacích souřadnic od kružnic, které jsou definované výše zmíněnými poloměry se středem v počátku souřadnicového systému texturovacích souřadnic.

Následně jsou spočítány gradienty těchto vzdáleností a je vypočtena znaménková vzdálenost od kružnic dle rovnice 2.26 v sekci 2.6.4 Anti-aliasing. Dle vzdálenosti je vypočtena průhlednost, která má lineární průběh přes hranu do vzdálenosti jednoho pixelu.

Pro míchání poloprůhledných barev je nastavena následující rovnice.

$$C_{result} = C_{source} * A_{source} + C_{destination} * (1 - A_{source}) \tag{3.1}$$

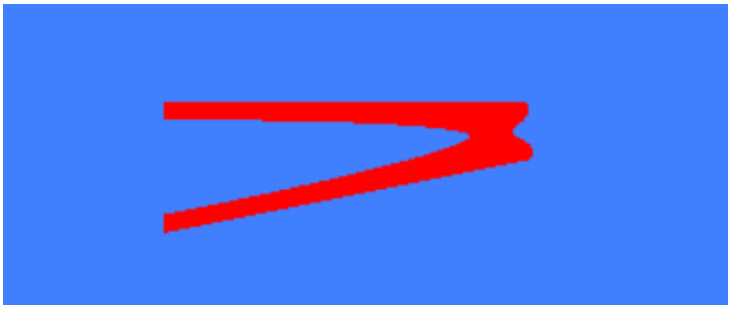
3.8 Některá zrychlení vykreslování uzavřené cesty

V průběhu implementace vykreslování uzavřených cest bylo vykreslování s vyhlazováním hran velmi pomalé. Bylo přistoupeno ke zrychlení vykreslování.

Jednou z myšlenek bylo nespouštění rasterizace geometrických tvarů, které se nenachází na obrazovce. Test, jestli se geometrický tvar nenachází na obrazovce, byl řešen pomocí obdélníka. Obdélník byl vytvořen, tak aby obsahoval celou uzavřenou cestu. Souřadnice obdélníka jsou spočteny nalezením minima a maxima x-ových a y-ových souřadnic všech bodů uzavřené cesty. Souřadnice obdélníka jsou následně transformovány maticemi transformace. Pokud je zjištěno, že obdélník se nenachází žádnou částí na obrazovce, rasterizace objektu na GPU není spuštěna. Tento způsob vykreslení se projevil na rychlosti vykreslení vektorového obrázku při přiblížení či posunu.

Obdélník obsahující uzavřenou křivku byl následně použit k dalšímu zrychlení. Při vykreslování pomocí implicitních rovnic velkého množství menších křivek je vykreslování velmi pomalé. V kroku překreslení šablony do výsledného obrazu jsou počítány pixely, které se nenachází v obdélníku. Jelikož obdélník obsahuje celou uzavřenou cestu, pixely jsou zbytečně počítány pomocí šablony (stencil testu).

Prostor vykreslování na GPU lze oříznout funkcí `ID3D11DeviceContext::RSSetScissorRect`. Oříznutím prostoru vykreslení nebudou počítány pixely mimo nastavený scissor obdélník. Jako obdélník oříznutí byl nastaven výše vypočítaný obdélník, který obsahuje celou cestu.



Obrázek 3.3: Chybné vykreslení kvadratické křivky s širokým okrajem.

Zmenšením prostoru vykreslení, byla zvýšena rychlost vykreslení uzavřených cest. Nejvíce se zrychlení projevilo při vykreslování velkého množství malých uzavřených cest s vyhlazenými hranami.

3.9 Vykreslení obrysu tvaru uzavřeného cestou

Implementování vykreslení obrysu tvaru uzavřeného cestou probíhalo podobným způsobem jako vykreslování antialiasingu. Vypočítala se vzdálenost od křivky. Následně se vzdálenost porovnala s šířkou vykreslovaného okraje. Tato implementace fungovala, dokud nebyly testovány některé křivky. Například na obrázku 3.3 je vidět vykreslení širokého okraje kvadratické křivky.

Rozhodl jsem se vykreslení okraje cesty převést na vykreslení výplně. Problémem je v definici nové cesty. Bézierovu křivku v původním tvaru nelze jednoduše posunout. Křivka musí být reprezentována křivkou mnohem vyššího řádu. A proto jsem se rozhodl aproximovat okraj Bézierovými křivkami. Program Inkscape umožňuje převést okraje cesty aproximací na vyplnění cesty pomocí Bézierových křivek třetího řádu, proto bylo toto řešení využito.

3.10 Řešení maticových transformací definovaných uvnitř SVG souboru

Některé SVG obrázky obsahovaly matice transformace jednotlivých geometrických tvarů. Vzhledem ke složitosti parsování transformací v SVG souboru, není parsování implementováno v programu.

Řešení transformací je řešeno aplikováním transformací v programu Inkscape. Obrázek je upraven v programu Inkscape pomocí následujících kroků:

1. Nastavení možnosti *Upravit* → *Předvolby* → *Chování* → *Transformace* → *Uložení transformace: Optimalizováno* zapne aplikování transformací.
2. Funkce *Upravit* → *Vybrat vše* vybere všechny objekty ve vektorovém obrázku.
3. Funkce *Objekt* → *Seskupit* je spuštěna, aby mohla být spuštěna funkce na zrušení seskupení.

4. Funkce *Objekt* → *Zrušit seskupení* zruší seskupení a aplikuje transformace na objekty.

5. Opakovat krok 4, dokud nejsou zrušena všechna seskupení.

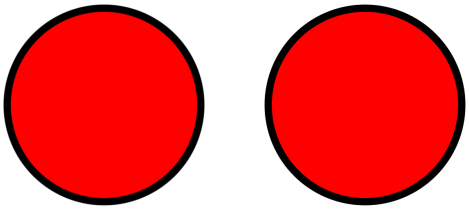
Program neumožňuje zobrazit lineární gradienty. Toto nebylo v zadání. Cílem zadání bylo porovnání rychlostí vykreslování mezi aproximací křivek a následnou rasterizací a rasterizací pomocí implicitních rovnic v prostoru pixel shaderu. Cílem práce nebylo podporovat, veškerou funkčnost, co vektorový obrázek typu SVG může využívat. Je možné při pokračování této problematiky se tímto směrem zabývat.

Kapitola 4

Výsledky

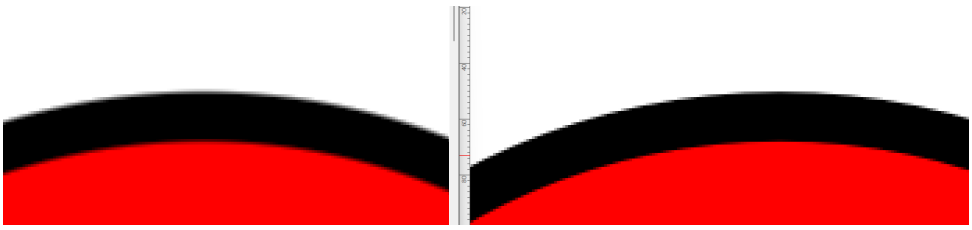
Výsledkem implementace je spustitelný program, který dokáže načíst a vykreslit vektorové obrázky. Rychlost vykreslení těchto obrázků dokáže také měřit.

Program dokáže zobrazit kruh s okrajem i bez něj pomocí implicitních rovnic počítaných v prostoru pixel shaderu. Styl kruhu lze popsat v SVG souboru pomocí XML atributů nebo style atributu použitím CSS (Cascading Style Sheets) syntaxe. Na obrázku 4.1 lze vidět vykreslený kruh pomocí implicitních rovnic a pomocí knihovny Direct2D.

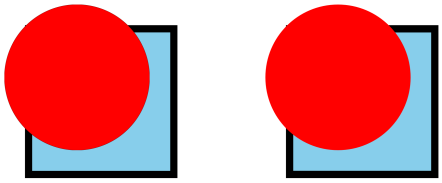


Obrázek 4.1: Nalevo vykreslení kruhu pomocí implicitních rovnic, napravo vykreslení kruhu použitím knihovny Direct2D.

Vykreslené obrázky se liší především v místech přechodu barev mezi červenou a černou a mezi černou a bílou. Důvodem je použití různých technik vyhlazování hran. Výsledné obrázky proto byly otevřeny v nástroji Microsoft Paint a přiblíženy. Přiblížené obrázky ukázaly kvalitu vyhlazení hran obou způsobů vykreslení (viz obrázek 4.2).



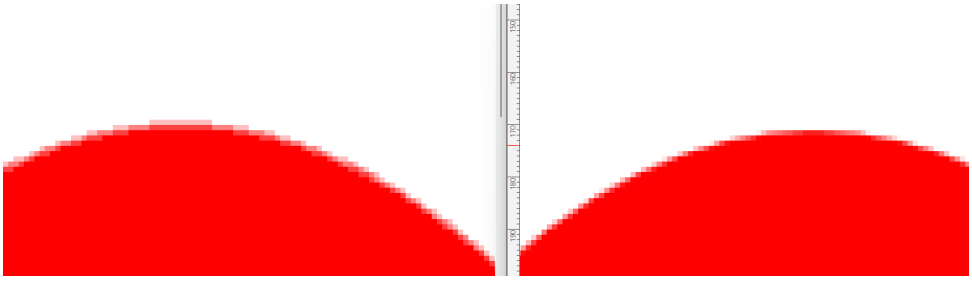
Obrázek 4.2: Porovnání vyhlazení přechodů vykreslovaného kruhu pomocí implicitních rovnic a použitím knihovny Direct2D.



Obrázek 4.3: Program vykreslil obrázek, kde kruh překrývá obdélník. Nalevo je vidět vykreslený obrázek pomocí implicitních rovnic, napravo je obrázek vykreslený použitím knihovny Direct2D.

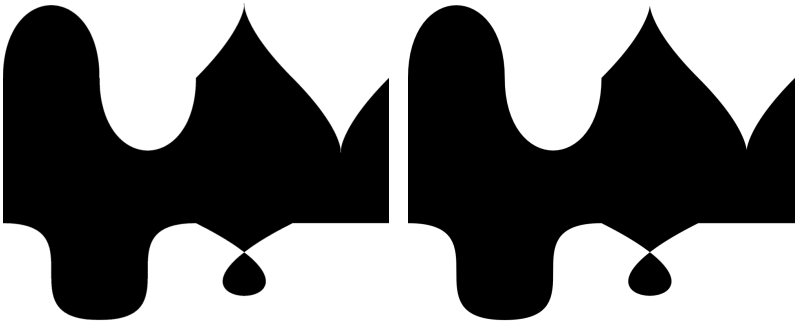
Program také dokáže zobrazit obdélník s okrajem i bez okraje. Program respektuje pořadí vykreslení. Vykreslované objekty se mohou překrývat a jednotlivé typy objektů při vykreslování lze střídat. Na obrázku 4.3 lze vidět vykreslený obdélník a kruh.

Obrázek 4.4: Nalevo je vykreslený geometrický tvar s Bézierovými křivkami pomocí implicitních rovnic. Napravo je vykreslený geometrický tvar použitím knihovny Direct2D.



Obrázek 4.5: Nalevo je vidět vyhlazení hrany kvadratické Bézierovy křivky, vykreslené pomocí implicitních rovnic. Napravo je vykreslena Bézierova křivka použití knihovny Direct2D.

V programu je implementována funkčnost vykreslovat uzavřené cesty s výplní. Dokáže vykreslit cestu s pravidlem výplně principem Odd - Even. Na obrázku 4.4 je vidět vykreslený vektorový obrázek, který obsahuje kvadratické křivky. Zde byl použit jiný způsob vyhlazení hran, vykreslené obrázky jsou



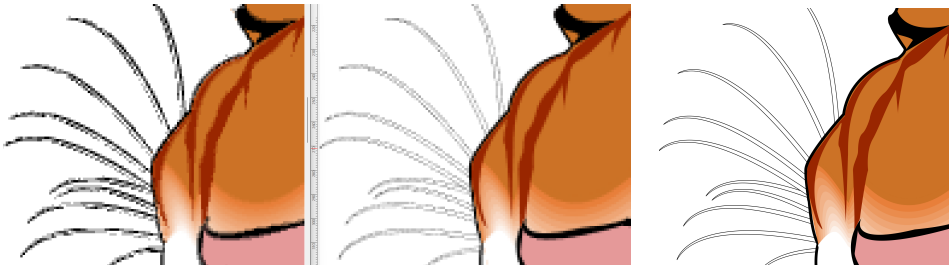
Obrázek 4.6: Vykreslování různých kubických křivek. Nalevo je obrázek vykreslen použitím implicitních rovnic, napravo je obrázek vykreslen použitím knihovny Direct2D.

Obrázek 4.7: Vykreslení obrázku tigra nalevo pomocí implicitních rovnic a napravo použitím knihovny Direct2D.

přiblíženy pro lepší názornost. Na obrázcích je vidět, že vykreslovaná křivka pomocí implicitních rovnic není, tak kvalitně vyhlazená, jako vykreslený kruh na obrázku 4.5. Program umožňuje vykreslit různé kubické křivky. Příklady některých kubických křivek jsou vidět na obrázku 4.6.

Při testování vykreslení složitějších obrázků byly nalezeny další rozdíly ve výsledku vykreslení. V rámci jednoho obrázku nastala situace, kdy v oblasti hrubšího vykreslení rozdíly nebyly tak markantní, jako u vykreslení detailu, drobných čar apod. Příkladem je obrázek hlavy tigra (viz obrázek 4.7) [20]. V menším rozlišení nastaly viditelné rozdíly ve výsledku vykreslení. Viditelný rozdíl je ve vykreslení malých geometrických tvarů o velikosti přibližně pixelu (např. vykreslené chloupky tigra viz obrázek 4.8).

Program dokáže vykreslit SVG obrázky oběma způsoby, a to jak vykreslení pomocí implicitních rovnic, tak i vykreslení aproximací křivek. K vykreslování



Obrázek 4.8: Přiblížení výsledku vykreslení tigra. Nalevo vykreslení pomocí implicitních rovnic. Uprostřed vykreslení použitím knihovny Direct2D. Napravo vykreslení ve větším rozlišení pomocí knihovny Direct2D.

pomocí aproximací křivek byla použita knihovna Direct2D. Oba způsoby vykreslení produkují velmi podobný výsledný obraz. Rozdíly výsledného obrazu se nacházely na hranách geometrických tvarů. Oba způsoby vykreslení vyhlazovaly hrany. Rozdíly obrazu byly minimální a rychlosti jejich vykreslení byly porovnány.

4.1 Porovnání rychlostí

Porovnání rychlostí zobrazení vektorových obrázků bylo prováděno na různých počítačích pro různá rozlišení. V průběhu vykreslování byly s obrázky prováděny různé transformace.

Měření probíhalo na počítačích s různými GPU. Předpokládá se, že na rychlost vykreslení bude mít největší vliv počet výpočetních jader GPU a architektura GPU. Seznam specifikací počítačů, na kterých bylo prováděno vykreslování je v tabulce 4.1.

GPU	shader jednotek	CPU
NVIDIA GeForce RTX 4080	9728	Intel Core i9-10900X
NVIDIA GeForce RTX 2080 Ti	4352	Intel Core i9-10900X
NVIDIA Quadro RTX 5000	3072	Intel Core i9-10900X
NVIDIA GeForce GTX 1050 Ti	768	Intel Core i3-8100
Intel Iris Xe Graphics 80EU	640	Intel Core i5-1235U
NVIDIA GeForce GT 1030	384	Intel Core i3-4170
NVIDIA GeForce GT 730	384	AMD Ryzen 5 1600
NVIDIA GeForce GT 710	192	AMD Phenom II X4 960T
Intel HD Graphics 4400	160	Intel Core i3-4170

Tabulka 4.1: Specifikace počítačů, na kterých bylo prováděno vykreslování.

Vykreslování proběhlo v několika rozlišení. Předpokládá se, že velikost rozlišení bude mít velký vliv na rychlost vykreslení obrázku. Ověření v závislostí rozlišení probíhalo od velikosti 50x50 a postupně až do velikosti rozlišení 4400x4400.

Měření rychlostí vykreslování probíhalo na různých SVG obrázcích:

- obrázek kruhu,
- obrázek s kvadratickou Bézierovou křivkou,
- obrázek s kubickou Bézierovou křivkou,
- obrázek hlavy tigra.

Vedle různého rozlišení obrázku měření také probíhalo s různými manipulacemi s obrázky. Manipulací je myšleno přibližování během vykreslování popřípadě posunem přemístění obrázku na jinou pozici, například zleva doprava. Při tomto experimentu nebyla měřena rychlost vykreslení prvního snímku. Toto rozhodnutí bylo z důvodu velkého počtu neznámých. Vzhledem k tomu, že vykreslení daného objektu bylo několikrát opakováno v jednom procesu, vynechání prvního snímku v konečném důsledku celkovou rychlost neovlivňuje.

Měření, která proběhla byla následující:

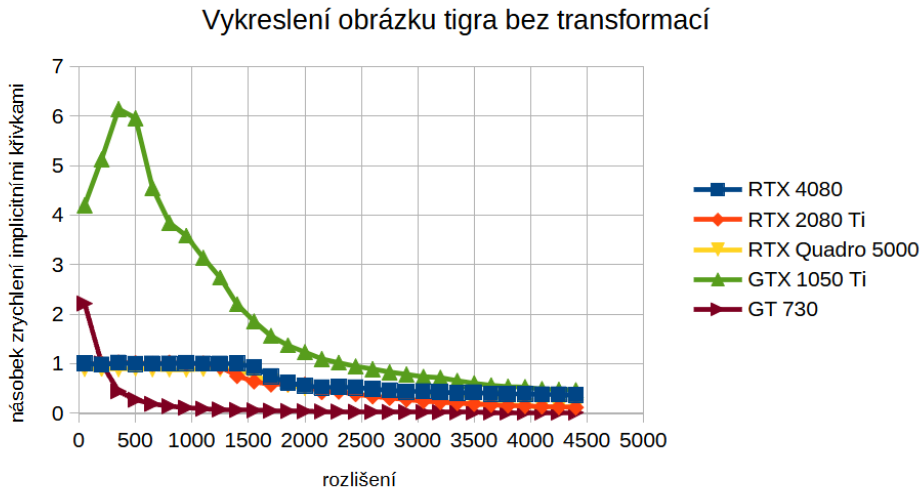
- Rychlost vykreslení obrázku bez transformací v závislosti na rozlišení.
- Rychlost vykreslení obrázku v závislosti na rozlišení a změně velikosti obrázku.
- Rychlost vykreslení obrázku v závislosti na rozlišení při posunu.
- Rychlost vykreslení obrázku v závislosti na použitém HW (hardware).

Vykreslení vektorového obrázku při nižších rozlišeních probíhalo rychleji metodou implicitních rovnic. Při vyšších rozlišeních bylo vykreslení rychlejší s metodou aproximací křivek úsečkami. Na některých moderních GPU byla rychlost vykreslování obou metod v nízkém rozlišení stejná (viz obrázky 4.9 a 4.10 a obrázky v příloze D).

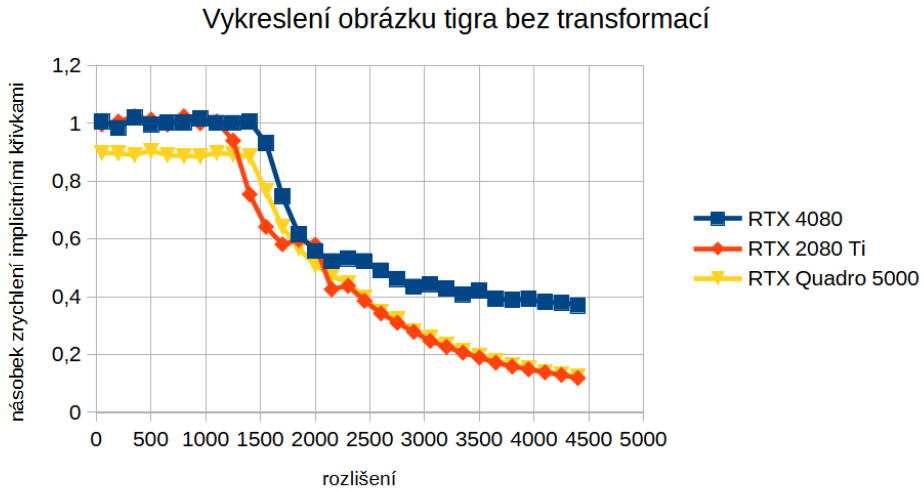
Vykreslování vektorového obrázku se změnou velikosti obrázku bylo rychlejší metodou implicitních rovnic. Ve vyšších rozlišení se rychlosti vykreslení obou metod přibližovaly (viz obrázek 4.11).

Rychlost vykreslování vektorového obrázku se změnou pozice byla vyšší než rychlost vykreslení statického vektorového obrázku. Důvodem vyšší rychlosti zřejmě byla skutečnost, že na některých snímcích vektorový obrázek nebyl celý viditelný (viz obrázek 4.12).

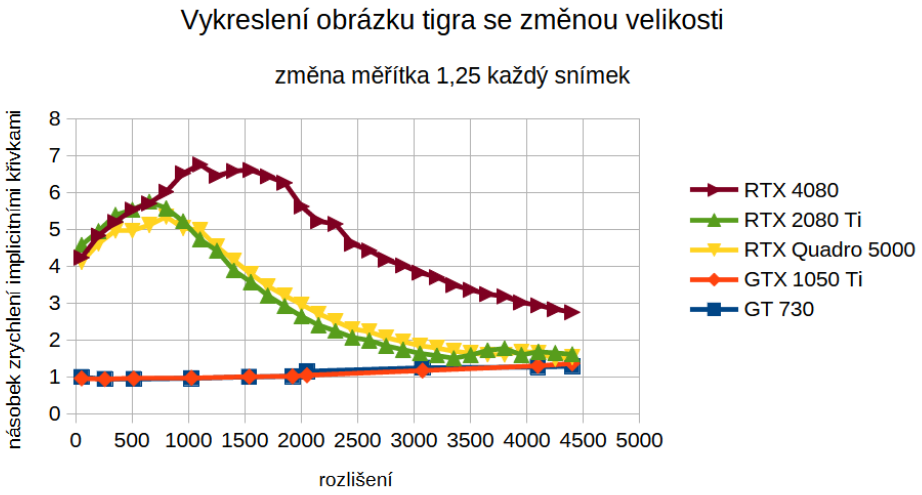
Rychlost vykreslování vektorových obrázků pomocí metody implicitních rovnic, byla rychlejší při vykreslování na nižším rozlišení než metoda vykreslení pomocí aproximace křivek. Toto chování je nejspíš zapříčeněno složitostí pixel shaderu a operací vykonávaných pro pixel či vzorek pixelu. Metoda vykreslení pomocí implicitních rovnic byla rychlejší v případě změn velikosti obrázku než metoda vykreslení pomocí aproximace křivek. Od určitého přiblížení je potřeba vektorový obrázek znovu aproximovat s menší či větší chybou.



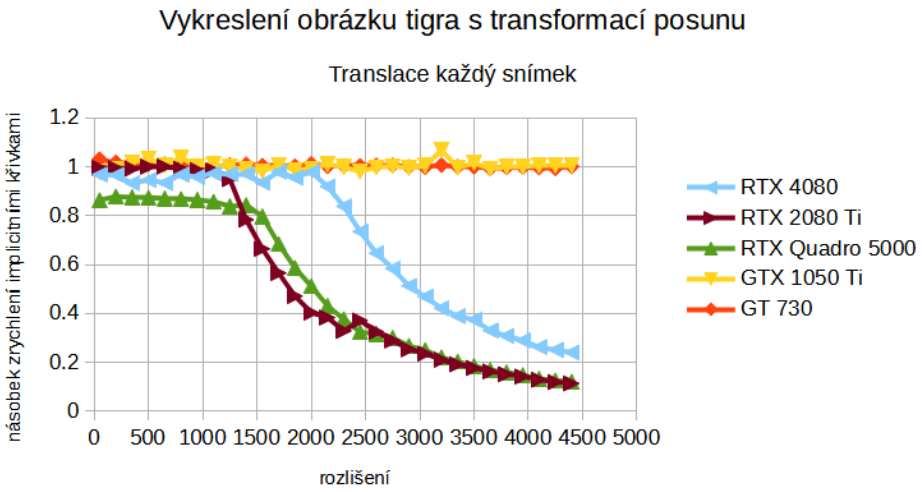
Obrázek 4.9: Zrychlení vykreslení obrázku tigra bez transformací použitím implicitních rovnic.



Obrázek 4.10: Zrychlení vykreslení obrázku tigra bez transformací použitím implicitních rovnic (graf přiblížen k výkonějším GPU).



Obrázek 4.11: Zrychlení vykreslení obrázku tigra se změnou měřítka 1,25 každým snímkem.



Obrázek 4.12: Zrychlení vykreslení obrázku tigra s transformací posunu v každém snímku.

Kapitola 5

Závěr

Implementoval jsem program umožňující načtení a vykreslení vektorových obrázků. Vykreslení křivek vektorové grafiky bylo provedeno jejich implicitní podobou. Implicitní rovnice křivek byly počítány v prostoru pixel shaderu na GPU.

Nejtěžší částí implementace bylo implementování vykreslení kubických Bézierových křivek, které reprezentovaly hranu geometrického tvaru. Vykreslení geometrických tvarů, které obsahovaly kubické Bézierovy křivky, bylo úspěšné.

Implementované vykreslení bylo následně porovnáno s metodou vykreslení využitím aproximace křivek na úsečky. Metoda aproximací křivek byla řešena knihovnou Direct2D.

Vykreslení pomocí metody implicitních křivek bylo v některých případech rychlejší než vykreslení aproximací křivek. Metoda implicitních křivek byla rychlejší v nižších rozlišeních a při neustálé změně velikosti vektorového obrázku.

Literatura

1. AKENINE-MÖLLER, Tomas; HAINES, Eric; HOFFMAN, Naty; PESCE, Angelo; IWANICKI, Michael; HILLAIRE, Sébastien. *Real-time rendering*. Fourth edition. Boca Raton: CRC Press/Taylor & Francis Group, 2018. ISBN 978-1-1386-2700-0 (cit. na s. 3, 5–7, 9).
2. WIKIPEDIA CONTRIBUTORS. *Shader — Wikipedia, The Free Encyclopedia* [online]. 2024. [cit. 2024-05-16]. Dostupné z: `https://en.wikipedia.org/w/index.php?title=Shader&oldid=1211435920` (cit. na str. 4).
3. MARSCHNER Steve a Shirley, Peter. *Fundamentals of Computer Graphics*. 4. vyd. CRC Press, 2015. ISBN 9781498785907. Dostupné také z: `https://ebookcentral.proquest.com/lib/cvut/detail.action?docID=4710787#` (cit. na s. 5, 7).
4. WIKI, OpenGL. *Rendering Pipeline Overview — OpenGL Wiki* [online]. 2022. [cit. 2024-05-16]. Dostupné z: `http://www.khronos.org/opengl/wiki_opengl/index.php?title=Rendering_Pipeline_Overview&oldid=14914` (cit. na str. 5).
5. WIKI, OpenGL. *Vertex Post-Processing — OpenGL Wiki* [online]. 2021. [cit. 2024-05-16]. Dostupné z: `http://www.khronos.org/opengl/wiki_opengl/index.php?title=Vertex_Post-Processing&oldid=14828` (cit. na str. 6).
6. MICROSOFT. Rasterization Rules [online]. [B.r.] [cit. 2024-05-16]. Dostupné z: `https://learn.microsoft.com/en-us/windows/win32/direct3d11/d3d10-graphics-programming-guide-rasterizer-stage-rules` (cit. na str. 7).
7. WIKI, OpenGL. *Sampler (GLSL) — OpenGL Wiki* [online]. 2020. [cit. 2024-05-16]. Dostupné z: `http://www.khronos.org/opengl/wiki_opengl/index.php?title=Sampler_(GLSL)&oldid=14716` (cit. na str. 7).
8. ŽÁRA, Jiří; LIMPOUCH, Aleš; BENEŠ, Bedřich; WERNER, Tomáš. *Počítačová grafika principy a algoritmy /: principy a algoritmy*. Praha: Grada, 1992. ISBN 80-85623-00-5 (cit. na str. 9).

9. RIŠKUS, Aleksas. Approximation of a cubic bezier curve by circular arcs and vice versa. *Information Technology and Control*. 2006, roč. 35 (cit. na str. 12).
10. LOOP, Charles; BLINN, Jim. Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics - TOG*. 2005, roč. 24. Dostupné z DOI: 10.1145/1073204.1073303 (cit. na s. 14–18).
11. NGUYEN, Hubert et al. *Gpu gems 3*. First. Addison-Wesley Professional, 2007. ISBN 9780321545428. Dostupné také z: https://developer.nvidia.com/gpugems/gpugems3/contributors (cit. na s. 14, 15).
12. BLINN, Jim. *Jim Blinn's corner: notation, notation, notation*. Amsterdam: Morgan Kaufmann Publishers, 2003. The Morgan Kaufmann series in computer graphics and geometric modeling. ISBN 978-1-55860-860-3 (cit. na s. 15, 16).
13. SALMON, George. *A Treatise on the Higher Plane Curves*. Dublin: Hodges & Smith, 1852. Dostupné také z: http://name.umdl.umich.edu/ABQ9497 (cit. na str. 15).
14. SEIDEL, Raimund. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry*. 1991, roč. 1, č. 1, s. 51–64. ISSN 0925-7721. Dostupné z DOI: https://doi.org/10.1016/0925-7721(91)90012-4 (cit. na str. 18).
15. O'ROURKE, Joseph. *Computational Geometry in C*. 2. vyd. Cambridge University Press, 1998. ISBN 0521640105, ISBN 0521649765 (cit. na str. 18).
16. MEISTERS, G. H. Polygons Have Ears. *The American Mathematical Monthly* [online]. 1975, roč. 82, č. 6, s. 648–651 [cit. 2024-01-17]. ISSN 00029890, ISSN 19300972. Dostupné z: http://www.jstor.org/stable/2319703 (cit. na str. 19).
17. WIKIPEDIA CONTRIBUTORS. *Polygon triangulation — Wikipedia, The Free Encyclopedia* [online]. 2024. [cit. 2024-01-18]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Polygon_triangulation&oldid=1194965482 (cit. na str. 19).
18. EBERLY, David. Triangulation by Ear Clipping. *Geometric Tools* [online]. 2002 [cit. 2024-01-17]. Dostupné z: https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf (cit. na str. 19).
19. KOKOJIMA, Yoshiyuki; SUGITA, Kaoru; SAITO, Takahiro; TAKEMOTO, Takashi. Resolution independent rendering of deformable vector objects using graphics hardware. In: *ACM SIGGRAPH 2006 Sketches*. Boston, Massachusetts: Association for Computing Machinery, 2006, 118–es. SIGGRAPH '06. ISBN 1595933646. Dostupné z DOI: 10.1145/1179849.1179997 (cit. na str. 20).

20. COMMONS, Wikimedia. *File:Ghostscript Tiger.svg* [online]. 2009. [cit. 2024-05-24]. Dostupné z: `https://commons.wikimedia.org/wiki/File:Ghostscript_Tiger.svg` (cit. na str. 31).

Seznam zkratek

2D 2 dimensions

3D 3 dimensions

API Application Programming Interface

CPU Central Processing Unit

CSS Cascading Style Sheets

GLM OpenGL Mathemantics

GPU Graphics Processing Unit

HLSL High-Level Shader Language

HW hardware

MSAA Multi Sample Anti-Aliaisng

MSVC Microsoft Visual C++

PC Personal Computer

RGBA Red Green Blue Alpha

SDK Software Development Kit

SVG Scalable Vector Graphics

XML Extensible Markup Language

XOR Exclusive or

Příloha A

Seznam příloh

- *cmake/* - CMake skripty
- *data/* - naměřená data, grafy a skripty pro práci s daty
- *shaders/* - HLSL zdrojové kódy
- *src/* - C++ zdrojové kódy
- *svg/* - vykreslované vektorové obrázky
- *CMakeLists.txt* - soubor sestavení projektu
- *VectorGraphicsWithSupporForImplicitCurves.exe* - implementovaný program

Příloha B

Implicitní rovnice Bézierovy křivky 2. řádu

$$0 = f(x,y) = Ax^2 + By^2 + Cxy + Dx + Ey + F$$

$$A = P_{2,y}^2 - 4P_{2,y}P_{1,y} + 2P_{2,y}P_{0,y} + 4P_{1,y}^2 - 4P_{1,y}P_{0,y} + P_{0,y}^2$$

$$B = P_{2,x}^2 - 4P_{2,x}P_{1,x} + 2P_{2,x}P_{0,x} + 4P_{1,x}^2 - 4P_{1,x}P_{0,x} + P_{0,x}^2$$

$$C = -2P_{2,x}P_{2,y} + 4P_{2,x}P_{1,y} - 2P_{2,x}P_{0,y} + 4P_{1,x}P_{2,y} - 8P_{1,x}P_{1,y} + 4P_{1,x}P_{0,y} - 2P_{0,x}P_{2,y} + 4P_{0,x}P_{1,y} - 2P_{0,x}P_{0,y}$$

$$D = 2P_{2,x}P_{2,y}P_{0,y} - 4P_{2,x}P_{1,y}^2 + 4P_{2,x}P_{1,y}P_{0,y} - 2P_{2,x}P_{0,y}^2 + 4P_{1,x}P_{2,y}P_{1,y} - 8P_{1,x}P_{2,y}P_{0,y} + 4P_{1,x}P_{1,y}P_{0,y} - 2P_{0,x}P_{2,y}^2 + 4P_{0,x}P_{2,y}P_{1,y} + 2P_{0,x}P_{2,y}P_{0,y} - 4P_{0,x}P_{1,y}^2$$

$$E = -2P_{2,x}^2P_{0,y} + 4P_{2,x}P_{1,x}P_{1,y} + 4P_{2,x}P_{1,x}P_{0,y} + 2P_{2,x}P_{0,x}P_{2,y} - 8P_{2,x}P_{0,x}P_{1,y} + 2P_{2,x}P_{0,x}P_{0,y} - 4P_{1,x}^2P_{2,y} - 4P_{1,x}^2P_{0,y} + 4P_{1,x}P_{0,x}P_{2,y} + 4P_{1,x}P_{0,x}P_{1,y} - 2P_{0,x}^2P_{2,y}$$

$$F = P_{2,x}^2P_{0,y}^2 - 4P_{2,x}P_{1,x}P_{1,y}P_{0,y} - 2P_{2,x}P_{0,x}P_{2,y}P_{0,y} + 4P_{2,x}P_{0,x}P_{1,y}^2 + 4P_{1,x}^2P_{2,y}P_{0,y} - 4P_{1,x}P_{0,x}P_{2,y}P_{1,y} + P_{0,x}^2P_{2,y}^2$$

Kde $P_i; i \in \{0,1,2\}$ jsou souřadnice řídících bodů.

Příloha C

Implicitní rovnice Bézierovy křivky 3. řádu

$$0 = f(x,y) = Ax^3 + By^3 + Cx^2y + Dxy^2 + Ex^2 + Fy^2 + Gxy + Hx + Iy + J$$

$$A = -P_{0,y}^3 + 9P_{1,y}P_{0,y}^2 - 9P_{2,y}P_{0,y}^2 + 3P_{3,y}P_{0,y}^2 - 27P_{1,y}^2P_{0,y} - 27P_{2,y}^2P_{0,y} - 3P_{3,y}^2P_{0,y} + 54P_{1,y}P_{2,y}P_{0,y} - 18P_{1,y}P_{3,y}P_{0,y} + 18P_{2,y}P_{3,y}P_{0,y} + 27P_{1,y}^3 - 27P_{2,y}^3 + P_{3,y}^3 + 81P_{1,y}P_{2,y}^2 + 9P_{1,y}P_{3,y}^2 - 9P_{2,y}P_{3,y}^2 - 81P_{1,y}^2P_{2,y} + 27P_{1,y}^2P_{3,y} + 27P_{2,y}^2P_{3,y} - 54P_{1,y}P_{2,y}P_{3,y}$$

$$B = P_{0,x}^3 - 9P_{1,x}P_{0,x}^2 + 9P_{2,x}P_{0,x}^2 - 3P_{3,x}P_{0,x}^2 + 27P_{1,x}^2P_{0,x} + 27P_{2,x}^2P_{0,x} + 3P_{3,x}^2P_{0,x} - 54P_{1,x}P_{2,x}P_{0,x} + 18P_{1,x}P_{3,x}P_{0,x} - 18P_{2,x}P_{3,x}P_{0,x} - 27P_{1,x}^3 + 27P_{2,x}^3 - P_{3,x}^3 - 81P_{1,x}P_{2,x}^2 - 9P_{1,x}P_{3,x}^2 + 9P_{2,x}P_{3,x}^2 + 81P_{1,x}^2P_{2,x} - 27P_{1,x}^2P_{3,x} - 27P_{2,x}^2P_{3,x} + 54P_{1,x}P_{2,x}P_{3,x}$$

$$C = 3P_{0,x}P_{0,y}^2 - 9P_{1,x}P_{0,y}^2 + 9P_{2,x}P_{0,y}^2 - 3P_{3,x}P_{0,y}^2 - 18P_{0,x}P_{1,y}P_{0,y} + 54P_{1,x}P_{1,y}P_{0,y} - 54P_{2,x}P_{1,y}P_{0,y} + 18P_{3,x}P_{1,y}P_{0,y} + 18P_{0,x}P_{2,y}P_{0,y} - 54P_{1,x}P_{2,y}P_{0,y} + 54P_{2,x}P_{2,y}P_{0,y} - 18P_{3,x}P_{2,y}P_{0,y} - 6P_{0,x}P_{3,y}P_{0,y} + 18P_{1,x}P_{3,y}P_{0,y} - 18P_{2,x}P_{3,y}P_{0,y} + 6P_{3,x}P_{3,y}P_{0,y} + 27P_{0,x}P_{1,y}^2 - 81P_{1,x}P_{1,y}^2 + 81P_{2,x}P_{1,y}^2 - 27P_{3,x}P_{1,y}^2 + 27P_{0,x}P_{2,y}^2 - 81P_{1,x}P_{2,y}^2 + 81P_{2,x}P_{2,y}^2 - 27P_{3,x}P_{2,y}^2 + 3P_{0,x}P_{3,y}^2 - 9P_{1,x}P_{3,y}^2 + 9P_{2,x}P_{3,y}^2 - 3P_{3,x}P_{3,y}^2 - 54P_{0,x}P_{1,y}P_{2,y} + 162P_{1,x}P_{1,y}P_{2,y} - 162P_{2,x}P_{1,y}P_{2,y} + 54P_{3,x}P_{1,y}P_{2,y} + 18P_{0,x}P_{1,y}P_{3,y} - 54P_{1,x}P_{1,y}P_{3,y} + 54P_{2,x}P_{1,y}P_{3,y} - 18P_{3,x}P_{1,y}P_{3,y} - 18P_{0,x}P_{2,y}P_{3,y} + 54P_{1,x}P_{2,y}P_{3,y} - 54P_{2,x}P_{2,y}P_{3,y} + 18P_{3,x}P_{2,y}P_{3,y}$$

$$D = -3P_{0,y}P_{0,x}^2 + 9P_{1,y}P_{0,x}^2 - 9P_{2,y}P_{0,x}^2 + 3P_{3,y}P_{0,x}^2 + 18P_{1,x}P_{0,y}P_{0,x} - 18P_{2,x}P_{0,y}P_{0,x} + 6P_{3,x}P_{0,y}P_{0,x} - 54P_{1,x}P_{1,y}P_{0,x} + 54P_{2,x}P_{1,y}P_{0,x} - 18P_{3,x}P_{1,y}P_{0,x} + 54P_{1,x}P_{2,y}P_{0,x} - 54P_{2,x}P_{2,y}P_{0,x} + 18P_{3,x}P_{2,y}P_{0,x} - 18P_{1,x}P_{3,y}P_{0,x} + 18P_{2,x}P_{3,y}P_{0,x} - 6P_{3,x}P_{3,y}P_{0,x} - 27P_{1,x}^2P_{0,y} - 27P_{2,x}^2P_{0,y} - 3P_{3,x}^2P_{0,y} + 54P_{1,x}P_{2,x}P_{0,y} - 18P_{1,x}P_{3,x}P_{0,y} + 18P_{2,x}P_{3,x}P_{0,y} + 81P_{1,x}^2P_{1,y} + 81P_{2,x}^2P_{1,y} + 9P_{3,x}^2P_{1,y} - 162P_{1,x}P_{2,x}P_{1,y} + 54P_{1,x}P_{3,x}P_{1,y} - 54P_{2,x}P_{3,x}P_{1,y} - 81P_{1,x}^2P_{2,y} - 81P_{2,x}^2P_{2,y} - 9P_{3,x}^2P_{2,y} + 162P_{1,x}P_{2,x}P_{2,y} - 54P_{1,x}P_{3,x}P_{2,y} + 54P_{2,x}P_{3,x}P_{2,y} + 27P_{1,x}^2P_{3,y} + 27P_{2,x}^2P_{3,y} + 3P_{3,x}^2P_{3,y} - 54P_{1,x}P_{2,x}P_{3,y} + 18P_{1,x}P_{3,x}P_{3,y} - 18P_{2,x}P_{3,x}P_{3,y}$$

$$E = 3P_{3,x}P_{0,y}^3 - 9P_{2,x}P_{1,y}P_{0,y}^2 - 18P_{3,x}P_{1,y}P_{0,y}^2 - 18P_{1,x}P_{2,y}P_{0,y}^2 + 54P_{2,x}P_{2,y}P_{0,y}^2 - 9P_{3,x}P_{2,y}P_{0,y}^2 - 3P_{0,x}P_{3,y}P_{0,y}^2 + 27P_{1,x}P_{3,y}P_{0,y}^2 - 54P_{2,x}P_{3,y}P_{0,y}^2 + 21P_{3,x}P_{3,y}P_{0,y}^2 + 27P_{1,x}P_{1,y}^2P_{0,y} + 54P_{3,x}P_{1,y}^2P_{0,y} - 54P_{0,x}P_{2,y}^2P_{0,y} + 162P_{1,x}P_{2,y}^2P_{0,y} - 54P_{2,x}P_{2,y}^2P_{0,y} + 27P_{3,x}P_{2,y}^2P_{0,y} - 21P_{0,x}P_{3,y}^2P_{0,y} + 54P_{1,x}P_{3,y}^2P_{0,y} - 27P_{2,x}P_{3,y}^2P_{0,y} + 3P_{3,x}P_{3,y}^2P_{0,y} + 27P_{0,x}P_{1,y}P_{2,y}P_{0,y} - 81P_{1,x}P_{1,y}P_{2,y}P_{0,y} - 81P_{2,x}P_{1,y}P_{2,y}P_{0,y} - 27P_{3,x}P_{1,y}P_{2,y}P_{0,y} - 9P_{0,x}P_{1,y}P_{3,y}P_{0,y} - 27P_{1,x}P_{1,y}P_{3,y}P_{0,y} + 153P_{2,x}P_{1,y}P_{3,y}P_{0,y} - 63P_{3,x}P_{1,y}P_{3,y}P_{0,y} +$$

$$63P_{0,x}P_{2,y}P_{3,y}P_{0,y}-153P_{1,x}P_{2,y}P_{3,y}P_{0,y}+27P_{2,x}P_{2,y}P_{3,y}P_{0,y}+9P_{3,x}P_{2,y}P_{3,y}P_{0,y}-27P_{0,x}P_{1,y}^3-54P_{3,x}P_{1,y}^3+54P_{0,x}P_{2,y}^3+27P_{3,x}P_{2,y}^3-3P_{0,x}P_{3,y}^3-81P_{0,x}P_{1,y}P_{2,y}^2-81P_{1,x}P_{1,y}P_{2,y}^2-81P_{3,x}P_{1,y}P_{2,y}^2+9P_{0,x}P_{1,y}P_{3,y}^2-54P_{1,x}P_{1,y}P_{3,y}^2+18P_{2,x}P_{1,y}P_{3,y}^2+18P_{0,x}P_{2,y}P_{3,y}^2+9P_{1,x}P_{2,y}P_{3,y}^2+81P_{0,x}P_{1,y}^2P_{2,y}+81P_{2,x}P_{1,y}^2P_{2,y}+81P_{3,x}P_{1,y}^2P_{2,y}-27P_{0,x}P_{1,y}^2P_{3,y}+54P_{1,x}P_{1,y}^2P_{3,y}-162P_{2,x}P_{1,y}^2P_{3,y}+54P_{3,x}P_{1,y}^2P_{3,y}-54P_{0,x}P_{2,y}^2P_{3,y}-27P_{2,x}P_{2,y}^2P_{3,y}+27P_{0,x}P_{1,y}P_{2,y}P_{3,y}+81P_{1,x}P_{1,y}P_{2,y}P_{3,y}+81P_{2,x}P_{1,y}P_{2,y}P_{3,y}-27P_{3,x}P_{1,y}P_{2,y}P_{3,y}$$

$$F=3P_{3,y}P_{0,x}^3+3P_{3,x}P_{0,y}P_{0,x}^2+18P_{2,x}P_{1,y}P_{0,x}^2-27P_{3,x}P_{1,y}P_{0,x}^2+9P_{1,x}P_{2,y}P_{0,x}^2-54P_{2,x}P_{2,y}P_{0,x}^2+54P_{3,x}P_{2,y}P_{0,x}^2+18P_{1,x}P_{3,y}P_{0,x}^2+9P_{2,x}P_{3,y}P_{0,x}^2-21P_{3,x}P_{3,y}P_{0,x}^2+54P_{2,x}^2P_{0,y}P_{0,x}+21P_{3,x}^2P_{0,y}P_{0,x}-27P_{1,x}P_{2,x}P_{0,y}P_{0,x}+9P_{1,x}P_{3,x}P_{0,y}P_{0,x}-63P_{2,x}P_{3,x}P_{0,y}P_{0,x}-27P_{1,x}^2P_{1,y}P_{0,x}-162P_{2,x}^2P_{1,y}P_{0,x}-54P_{3,x}^2P_{1,y}P_{0,x}+81P_{1,x}P_{2,x}P_{1,y}P_{0,x}+27P_{1,x}P_{3,x}P_{1,y}P_{0,x}+153P_{2,x}P_{3,x}P_{1,y}P_{0,x}+54P_{2,x}^2P_{2,y}P_{0,x}+27P_{3,x}^2P_{2,y}P_{0,x}+81P_{1,x}P_{2,x}P_{2,y}P_{0,x}-153P_{1,x}P_{3,x}P_{2,y}P_{0,x}-27P_{2,x}P_{3,x}P_{2,y}P_{0,x}-54P_{1,x}^2P_{3,y}P_{0,x}-27P_{2,x}^2P_{3,y}P_{0,x}-3P_{3,x}^2P_{3,y}P_{0,x}+27P_{1,x}P_{2,x}P_{3,y}P_{0,x}+63P_{1,x}P_{3,x}P_{3,y}P_{0,x}-9P_{2,x}P_{3,x}P_{3,y}P_{0,x}+27P_{1,x}^3P_{0,y}-54P_{2,x}^3P_{0,y}+3P_{3,x}^3P_{0,y}+81P_{1,x}P_{2,x}^2P_{0,y}-9P_{1,x}P_{3,x}^2P_{0,y}-18P_{2,x}P_{3,x}^2P_{0,y}-81P_{1,x}^2P_{2,x}P_{0,y}+27P_{1,x}^2P_{3,x}P_{0,y}+54P_{2,x}^2P_{3,x}P_{0,y}-27P_{1,x}P_{2,x}P_{3,x}P_{0,y}+81P_{1,x}P_{2,x}^2P_{1,y}+54P_{1,x}P_{3,x}^2P_{1,y}-9P_{2,x}P_{3,x}^2P_{1,y}-54P_{1,x}^2P_{3,x}P_{1,y}-81P_{1,x}P_{2,x}P_{3,x}P_{1,y}-18P_{1,x}P_{3,x}^2P_{2,y}-81P_{1,x}^2P_{2,x}P_{2,y}+162P_{1,x}^2P_{3,x}P_{2,y}+27P_{2,x}^2P_{3,x}P_{2,y}-81P_{1,x}P_{2,x}P_{3,x}P_{2,y}+54P_{1,x}^3P_{3,y}-27P_{2,x}^3P_{3,y}+81P_{1,x}P_{2,x}^2P_{3,y}-81P_{1,x}^2P_{2,x}P_{3,y}-54P_{1,x}^2P_{3,x}P_{3,y}+27P_{1,x}P_{2,x}P_{3,x}P_{3,y}$$

$$G=54P_{2,y}^2P_{0,x}^2+21P_{3,y}^2P_{0,x}^2-27P_{1,y}P_{2,y}P_{0,x}^2+6P_{0,y}P_{3,y}P_{0,x}^2+9P_{1,y}P_{3,y}P_{0,x}^2-63P_{2,y}P_{3,y}P_{0,x}^2-6P_{3,x}P_{0,y}^2P_{0,x}+54P_{1,x}P_{1,y}^2P_{0,x}-81P_{2,x}P_{1,y}^2P_{0,x}-27P_{3,x}P_{1,y}^2P_{0,x}-81P_{1,x}P_{2,y}^2P_{0,x}-108P_{2,x}P_{2,y}^2P_{0,x}+27P_{3,x}P_{2,y}^2P_{0,x}-63P_{1,x}P_{3,y}^2P_{0,x}+9P_{2,x}P_{3,y}^2P_{0,x}+6P_{3,x}P_{3,y}^2P_{0,x}-9P_{2,x}P_{0,y}P_{1,y}P_{0,x}+45P_{3,x}P_{0,y}P_{1,y}P_{0,x}+9P_{1,x}P_{0,y}P_{2,y}P_{0,x}-45P_{3,x}P_{0,y}P_{2,y}P_{0,x}-81P_{1,x}P_{1,y}P_{2,y}P_{0,x}+243P_{2,x}P_{1,y}P_{2,y}P_{0,x}-45P_{1,x}P_{0,y}P_{3,y}P_{0,x}+45P_{2,x}P_{0,y}P_{3,y}P_{0,x}+81P_{1,x}P_{1,y}P_{3,y}P_{0,x}-180P_{2,x}P_{1,y}P_{3,y}P_{0,x}+45P_{3,x}P_{1,y}P_{3,y}P_{0,x}+126P_{1,x}P_{2,y}P_{3,y}P_{0,x}+81P_{2,x}P_{2,y}P_{3,y}P_{0,x}-45P_{3,x}P_{2,y}P_{3,y}P_{0,x}-54P_{2,x}^2P_{0,y}^2-21P_{3,x}^2P_{0,y}^2+27P_{1,x}P_{2,x}P_{0,y}^2-9P_{1,x}P_{3,x}P_{0,y}^2+63P_{2,x}P_{3,x}P_{0,y}^2-81P_{2,x}^2P_{1,y}^2-54P_{3,x}^2P_{1,y}^2+108P_{1,x}P_{3,x}P_{1,y}^2+81P_{2,x}P_{3,x}P_{1,y}^2+81P_{1,x}^2P_{2,y}^2+81P_{1,x}P_{3,x}P_{2,y}^2-54P_{2,x}P_{3,x}P_{2,y}^2+54P_{1,x}^2P_{3,y}^2-27P_{1,x}P_{2,x}P_{3,y}^2-54P_{1,x}^2P_{0,y}P_{1,y}+81P_{2,x}^2P_{0,y}P_{1,y}+63P_{3,x}^2P_{0,y}P_{1,y}+81P_{1,x}P_{2,x}P_{0,y}P_{1,y}-81P_{1,x}P_{3,x}P_{0,y}P_{1,y}-126P_{2,x}P_{3,x}P_{0,y}P_{1,y}+81P_{1,x}^2P_{0,y}P_{2,y}+108P_{2,x}^2P_{0,y}P_{2,y}-9P_{3,x}^2P_{0,y}P_{2,y}-243P_{1,x}P_{2,x}P_{0,y}P_{2,y}+180P_{1,x}P_{3,x}P_{0,y}P_{2,y}-81P_{2,x}P_{3,x}P_{0,y}P_{2,y}+27P_{3,x}^2P_{1,y}P_{2,y}-243P_{1,x}P_{3,x}P_{1,y}P_{2,y}+81P_{2,x}P_{3,x}P_{1,y}P_{2,y}+27P_{1,x}^2P_{0,y}P_{3,y}-27P_{2,x}^2P_{0,y}P_{3,y}-6P_{3,x}^2P_{0,y}P_{3,y}-45P_{1,x}P_{3,x}P_{0,y}P_{3,y}+45P_{2,x}P_{3,x}P_{0,y}P_{3,y}-108P_{1,x}^2P_{1,y}P_{3,y}-81P_{2,x}^2P_{1,y}P_{3,y}+243P_{1,x}P_{2,x}P_{1,y}P_{3,y}-9P_{2,x}P_{3,x}P_{1,y}P_{3,y}-81P_{1,x}^2P_{2,y}P_{3,y}+54P_{2,x}^2P_{2,y}P_{3,y}-81P_{1,x}P_{2,x}P_{2,y}P_{3,y}+9P_{1,x}P_{3,x}P_{2,y}P_{3,y}$$

$$H=-3P_{3,x}^2P_{0,y}^3+9P_{3,x}^2P_{1,y}P_{0,y}^2+18P_{2,x}P_{3,x}P_{1,y}P_{0,y}^2-27P_{2,x}^2P_{2,y}P_{0,y}^2+18P_{3,x}^2P_{2,y}P_{0,y}^2+36P_{1,x}P_{3,x}P_{2,y}P_{0,y}^2-54P_{2,x}P_{3,x}P_{2,y}P_{0,y}^2+81P_{2,x}^2P_{3,y}P_{0,y}^2+3P_{3,x}^2P_{3,y}P_{0,y}^2-27P_{1,x}P_{2,x}P_{3,y}P_{0,y}^2+6P_{0,x}P_{3,x}P_{3,y}P_{0,y}^2-27P_{1,x}P_{3,x}P_{3,y}P_{0,y}^2-27P_{2,x}P_{3,x}P_{3,y}P_{0,y}^2-27P_{3,x}^2P_{1,y}^2P_{0,y}-54P_{1,x}P_{3,x}P_{1,y}^2P_{0,y}-81P_{1,x}^2P_{2,y}^2P_{0,y}+54P_{0,x}P_{2,x}P_{2,y}^2P_{0,y}+54P_{0,x}P_{3,x}P_{2,y}^2P_{0,y}-162P_{1,x}P_{3,x}P_{2,y}^2P_{0,y}+54P_{2,x}P_{3,x}P_{2,y}^2P_{0,y}-3P_{0,x}^2P_{3,y}^2P_{0,y}-81P_{1,x}^2P_{3,y}^2P_{0,y}+27P_{0,x}P_{1,x}P_{3,y}^2P_{0,y}+27P_{0,x}P_{2,x}P_{3,y}^2P_{0,y}+27P_{1,x}P_{2,x}P_{3,y}^2P_{0,y}-6P_{0,x}P_{3,x}P_{3,y}^2P_{0,y}-27P_{3,x}^2P_{1,y}P_{2,y}P_{0,y}+81P_{1,x}P_{2,x}P_{1,y}P_{2,y}P_{0,y}-54P_{0,x}P_{3,x}P_{1,y}P_{2,y}P_{0,y}+81P_{1,x}P_{3,x}P_{1,y}P_{2,y}P_{0,y}+81P_{2,x}P_{3,x}P_{1,y}P_{2,y}P_{0,y}+54P_{1,x}^2P_{1,y}P_{3,y}P_{0,y}-81P_{2,x}^2P_{1,y}P_{3,y}P_{0,y}+9P_{0,x}P_{2,x}P_{1,y}P_{3,y}P_{0,y}-162P_{1,x}P_{2,x}P_{1,y}P_{3,y}P_{0,y}+9P_{0,x}P_{3,x}P_{1,y}P_{3,y}P_{0,y}+108P_{1,x}P_{3,x}P_{1,y}P_{3,y}P_{0,y}+$$

$$9P_{2,x}P_{3,x}P_{1,y}P_{3,y}P_{0,y}+81P_{1,x}^2P_{2,y}P_{3,y}P_{0,y}-54P_{2,x}^2P_{2,y}P_{3,y}P_{0,y}-9P_{0,x}P_{1,x}P_{2,y}P_{3,y}P_{0,y}-108P_{0,x}P_{2,x}P_{2,y}P_{3,y}P_{0,y}+162P_{1,x}P_{2,x}P_{2,y}P_{3,y}P_{0,y}-9P_{0,x}P_{3,x}P_{2,y}P_{3,y}P_{0,y}-9P_{1,x}P_{3,x}P_{2,y}P_{3,y}P_{0,y}+27P_{3,x}^2P_{1,y}^3+54P_{0,x}P_{3,x}P_{1,y}^3-27P_{0,x}^2P_{2,y}^3-54P_{0,x}P_{3,x}P_{2,y}^3+3P_{0,x}^2P_{3,y}^3+81P_{0,x}P_{1,x}P_{1,y}P_{2,y}^2+81P_{0,x}P_{3,x}P_{1,y}P_{2,y}^2+81P_{1,x}P_{3,x}P_{1,y}P_{2,y}^2-18P_{0,x}^2P_{1,y}P_{3,y}^2+27P_{1,x}^2P_{1,y}P_{3,y}^2+54P_{0,x}P_{1,x}P_{1,y}P_{3,y}^2-36P_{0,x}P_{2,x}P_{1,y}P_{3,y}^2-9P_{0,x}^2P_{2,y}P_{3,y}^2-18P_{0,x}P_{1,x}P_{2,y}P_{3,y}^2-81P_{0,x}P_{2,x}P_{1,y}^2P_{2,y}-81P_{0,x}P_{3,x}P_{1,y}^2P_{2,y}-81P_{2,x}P_{3,x}P_{1,y}^2P_{2,y}+81P_{2,x}^2P_{1,y}^2P_{3,y}-54P_{0,x}P_{1,x}P_{1,y}^2P_{3,y}+162P_{0,x}P_{2,x}P_{1,y}^2P_{3,y}-54P_{0,x}P_{3,x}P_{1,y}^2P_{3,y}-54P_{1,x}P_{3,x}P_{1,y}^2P_{3,y}+27P_{0,x}^2P_{2,y}^2P_{3,y}+54P_{0,x}P_{2,x}P_{2,y}^2P_{3,y}+27P_{0,x}^2P_{1,y}P_{2,y}P_{3,y}-81P_{0,x}P_{1,x}P_{1,y}P_{2,y}P_{3,y}-81P_{0,x}P_{2,x}P_{1,y}P_{2,y}P_{3,y}-81P_{1,x}P_{2,x}P_{1,y}P_{2,y}P_{3,y}+54P_{0,x}P_{3,x}P_{1,y}P_{2,y}P_{3,y}$$

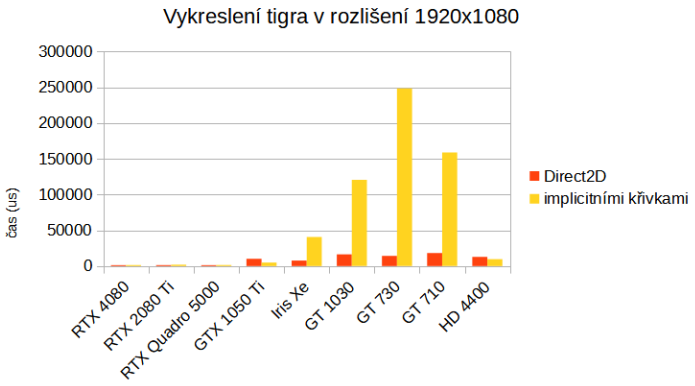
$$I=3P_{3,y}^2P_{0,x}^3+27P_{2,x}P_{2,y}^2P_{0,x}^2-81P_{3,x}P_{2,y}^2P_{0,x}^2-9P_{1,x}P_{3,y}^2P_{0,x}^2-18P_{2,x}P_{3,y}^2P_{0,x}^2-3P_{3,x}P_{3,y}^2P_{0,x}^2+27P_{3,x}P_{1,y}P_{2,y}P_{0,x}^2-6P_{3,x}P_{0,y}P_{3,y}P_{0,x}^2-36P_{2,x}P_{1,y}P_{3,y}P_{0,x}^2+27P_{3,x}P_{1,y}P_{3,y}P_{0,x}^2-18P_{1,x}P_{2,y}P_{3,y}P_{0,x}^2+54P_{2,x}P_{2,y}P_{3,y}P_{0,x}^2+27P_{3,x}P_{2,y}P_{3,y}P_{0,x}^2+3P_{3,x}^2P_{0,y}^2P_{0,x}+81P_{2,x}^2P_{1,y}^2P_{0,x}+81P_{3,x}^2P_{1,y}^2P_{0,x}-54P_{1,x}P_{3,x}P_{1,y}^2P_{0,x}-81P_{2,x}P_{3,x}P_{1,y}^2P_{0,x}+81P_{1,x}P_{3,x}P_{2,y}^2P_{0,x}+54P_{2,x}P_{3,x}P_{2,y}^2P_{0,x}+27P_{1,x}^2P_{3,y}^2P_{0,x}+27P_{1,x}P_{2,x}P_{3,y}^2P_{0,x}-27P_{3,x}^2P_{0,y}P_{1,y}P_{0,x}+9P_{2,x}P_{3,x}P_{0,y}P_{1,y}P_{0,x}-54P_{2,x}^2P_{0,y}P_{2,y}P_{0,x}-27P_{3,x}^2P_{0,y}P_{2,y}P_{0,x}-9P_{1,x}P_{3,x}P_{0,y}P_{2,y}P_{0,x}+108P_{2,x}P_{3,x}P_{0,y}P_{2,y}P_{0,x}-27P_{3,x}^2P_{1,y}P_{2,y}P_{0,x}-81P_{1,x}P_{2,x}P_{1,y}P_{2,y}P_{0,x}+162P_{1,x}P_{3,x}P_{1,y}P_{2,y}P_{0,x}-162P_{2,x}P_{3,x}P_{1,y}P_{2,y}P_{0,x}-54P_{2,x}^2P_{0,y}P_{3,y}P_{0,x}+6P_{3,x}^2P_{0,y}P_{3,y}P_{0,x}+54P_{1,x}P_{2,x}P_{0,y}P_{3,y}P_{0,x}-9P_{1,x}P_{3,x}P_{0,y}P_{3,y}P_{0,x}+9P_{2,x}P_{3,x}P_{0,y}P_{3,y}P_{0,x}+54P_{1,x}^2P_{1,y}P_{3,y}P_{0,x}+162P_{2,x}^2P_{1,y}P_{3,y}P_{0,x}-81P_{1,x}P_{2,x}P_{1,y}P_{3,y}P_{0,x}-108P_{1,x}P_{3,x}P_{1,y}P_{3,y}P_{0,x}+9P_{2,x}P_{3,x}P_{1,y}P_{3,y}P_{0,x}-54P_{2,x}^2P_{2,y}P_{3,y}P_{0,x}-81P_{1,x}P_{2,x}P_{2,y}P_{3,y}P_{0,x}-9P_{1,x}P_{3,x}P_{2,y}P_{3,y}P_{0,x}+27P_{2,x}^3P_{0,y}^2-3P_{3,x}^3P_{0,y}^2+18P_{1,x}P_{3,x}^2P_{0,y}^2+9P_{2,x}P_{3,x}^2P_{0,y}^2-27P_{2,x}^2P_{3,x}P_{0,y}^2-27P_{1,x}P_{2,x}P_{3,x}P_{0,y}^2-27P_{1,x}P_{3,x}^2P_{1,y}^2-81P_{1,x}^2P_{3,x}P_{2,y}^2-27P_{1,x}^3P_{3,y}^2-81P_{1,x}P_{2,x}^2P_{0,y}P_{1,y}-54P_{1,x}P_{3,x}^2P_{0,y}P_{1,y}+18P_{2,x}P_{3,x}^2P_{0,y}P_{1,y}+54P_{1,x}^2P_{3,x}P_{0,y}P_{1,y}+81P_{1,x}P_{2,x}P_{3,x}P_{0,y}P_{1,y}+36P_{1,x}P_{3,x}^2P_{0,y}P_{2,y}+81P_{1,x}^2P_{2,x}P_{0,y}P_{2,y}-162P_{1,x}^2P_{3,x}P_{0,y}P_{2,y}-54P_{2,x}^2P_{3,x}P_{0,y}P_{2,y}+81P_{1,x}P_{2,x}P_{3,x}P_{0,y}P_{2,y}+81P_{1,x}P_{2,x}P_{3,x}P_{1,y}P_{2,y}-54P_{1,x}^3P_{0,y}P_{3,y}+54P_{2,x}^3P_{0,y}P_{3,y}-81P_{1,x}P_{2,x}^2P_{0,y}P_{3,y}+81P_{1,x}^2P_{2,x}P_{0,y}P_{3,y}+54P_{1,x}^2P_{3,x}P_{0,y}P_{3,y}-54P_{1,x}P_{2,x}P_{3,x}P_{0,y}P_{3,y}-81P_{1,x}P_{2,x}^2P_{1,y}P_{3,y}+54P_{1,x}^2P_{3,x}P_{1,y}P_{3,y}+81P_{1,x}^2P_{2,x}P_{2,y}P_{3,y}$$

$$J=27P_{0,y}P_{3,y}^2P_{1,x}^3+81P_{3,x}P_{0,y}P_{2,y}^2P_{1,x}^2-27P_{0,x}P_{1,y}P_{3,y}^2P_{1,x}^2-54P_{3,x}P_{0,y}P_{1,y}P_{3,y}P_{1,x}^2-81P_{2,x}P_{0,y}P_{2,y}P_{3,y}P_{1,x}^2+27P_{3,x}^2P_{0,y}P_{1,y}^2P_{1,x}-81P_{0,x}P_{3,x}P_{1,y}P_{2,y}^2P_{1,x}-27P_{0,x}P_{2,x}P_{0,y}P_{3,y}^2P_{1,x}+9P_{0,x}^2P_{2,y}P_{3,y}^2P_{1,x}-18P_{3,x}^2P_{0,y}^2P_{2,y}P_{1,x}-81P_{2,x}P_{3,x}P_{0,y}P_{1,y}P_{2,y}P_{1,x}+27P_{2,x}P_{3,x}P_{0,y}^2P_{3,y}P_{1,x}+54P_{0,x}P_{3,x}P_{1,y}^2P_{3,y}P_{1,x}+81P_{2,x}^2P_{0,y}P_{1,y}P_{3,y}P_{1,x}+9P_{0,x}P_{3,x}P_{0,y}P_{2,y}P_{3,y}P_{1,x}+81P_{0,x}P_{2,x}P_{1,y}P_{2,y}P_{3,y}P_{1,x}+P_{3,x}^3P_{0,y}^3-27P_{0,x}P_{3,x}^2P_{1,y}^3+27P_{0,x}^2P_{3,x}P_{2,y}^3-P_{0,x}^3P_{3,y}^3-54P_{0,x}P_{2,x}P_{3,x}P_{0,y}P_{2,y}^2+3P_{0,x}^2P_{3,x}P_{0,y}P_{3,y}^2+18P_{0,x}^2P_{2,x}P_{1,y}P_{3,y}^2-9P_{2,x}P_{3,x}^2P_{0,y}^2P_{1,y}+27P_{2,x}^2P_{3,x}P_{0,y}^2P_{2,y}+81P_{0,x}P_{2,x}P_{3,x}P_{1,y}^2P_{2,y}+27P_{0,x}P_{3,x}^2P_{0,y}P_{1,y}P_{2,y}-27P_{2,x}^3P_{0,y}^2P_{3,y}-3P_{0,x}P_{3,x}^2P_{0,y}^2P_{3,y}-81P_{0,x}P_{2,x}^2P_{1,y}^2P_{3,y}-27P_{0,x}^2P_{2,x}P_{2,y}^2P_{3,y}-9P_{0,x}P_{2,x}P_{3,x}P_{0,y}P_{1,y}P_{3,y}+54P_{0,x}P_{2,x}^2P_{0,y}P_{2,y}P_{3,y}-27P_{0,x}^2P_{3,x}P_{1,y}P_{2,y}P_{3,y}$$

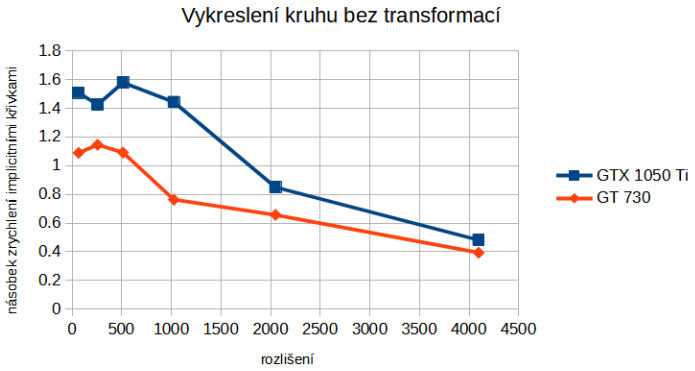
Kde $P_i; i \in \{0,1,2,3\}$ jsou souřadnice řídících bodů.

Příloha D

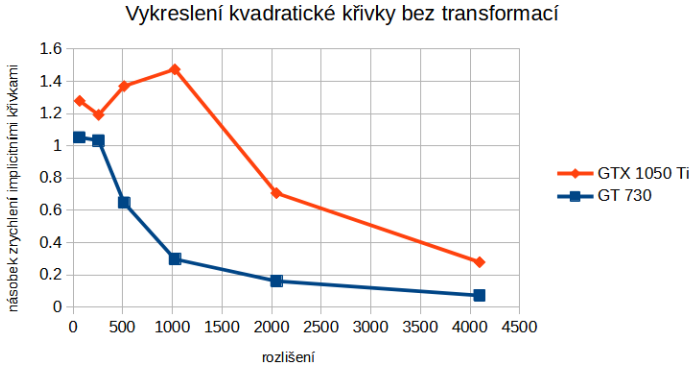
Grafy rychlostí vykreslení obrázků



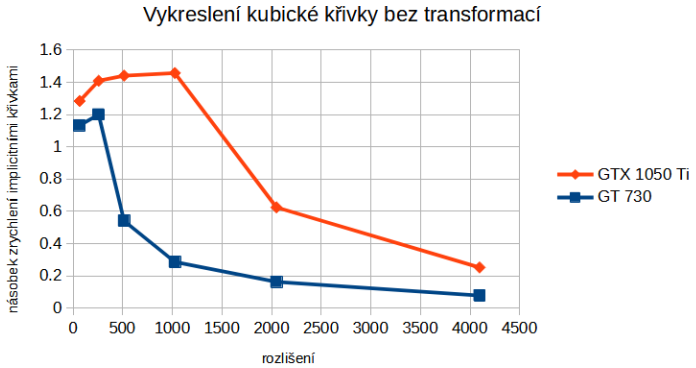
Obrázek D.1: Vykreslení obrázku tigra v rozlišení 1920x1080 pomocí implicitních rovnic a použitím knihovny Direct2D.



Obrázek D.2: Zrychlení vykreslení kruhu bez transformací použitím implicitních rovnic.



Obrázek D.3: Zrychlení vykreslení kvadratické křivky bez transformací použitím implicitních rovnic.



Obrázek D.4: Zrychlení vykreslení kubické křivky bez transformací použitím implicitních rovnic.