TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2612 – Elektrotechnika a informatika

Studijní obor: 1802R022 - Informatika a logistika

Hardwarový akcelerátor 3D grafických operací

Hardware accelerator of 3D graphical operations

Bakalářská práce

Autor: **Ivo Knejp**

Vedoucí práce: Rozkovec Martin, Ing.

V Liberci 1. 2. 2011

Strana s originálním zadáním a ??

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinností informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat náhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování a předmluva

Tímto bych rád poděkoval vedoucímu práce Ing. Martinovi Rozkovcovi za možnost využití jeho znalostí a zkušeností na poli integrovaných obvodů a za zapůjčení přípravku Xlinx XUPV5-LX110T.

Abstrakt

Tato bakalářská práce se zabývá návrhem hardwarového akcelerátoru 3D grafických operací. První část práce se zabývá konstrukcí programu, který vykresluje 3D obrazce za pomocí transformačních matic a Bresenhamova algoritmu. Vykreslováno je jednoduchou formou wire-framové grafiky, tudíž se nemusí řešit role světla ani viditelnosti. Softwarové řešení slouží k optimalizaci algoritmů a jejich přípravě pro druhou část. Druhá část práce je samotný návrh hardwaru a jeho implementace do hradlového pole. Hardware je navrhovaný v jazyce VHDL a pro všechny komponenty jsou provedeny simulace. Po implementaci jsou provedeny srovnávací testy pro demonstraci rychlosti hardwarového akcelerátoru oproti softwarovým výpočtům. Je diskutována možnost budoucího rozšíření práce v implementaci jiného typu rasterizace, u které by bylo nutno řešit roli světla, viditelnost a přidávání textur na renderovanou scénu.

Klíčová slova

Hardware, VHDL, transformační matice, 3D grafické operace

Abstract

This Bachelor's work concerns designing of the 3D graphical operations hardware accelerator. First part of the work is about software application, which demonstrates Bresenham's drawing algorithm and transformation matrixes. The application draws in simple wire frame graphic, so there is no need to be concerned by the role of light or visibility. The software solution of the problem is there for optimization of all algorithms and their preparation for second part. The second part is about designing whole solution and their implementation into programmable device. Hardware is designed in VHDL and simulations are made for all components. Tests are made to demonstrate speed difference between hardware accelerator and software computation. There is a discussion about possibility of future work expansion by the implementation of other kinds of rendering, where is necessary to solve role of the light, visibility or mapping of the textures.

Keywords

Hardware, VHDL, transformation matrix, 3D graphical operations

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Integrované obvody se stávají běžnou součástí našich životů, aniž by si to většina lidí uvědomovala. Můžeme je nalézt například v každém mobilním telefonu, v GPS v navigacích, nebo v každém počítači a to hlavně díky jejich vysoké spolehlivosti a rychlosti, se kterou pracují. A právě kvůli jejich širokému rozšíření v nich vidím veliký potenciál a rozhodl jsem se zaměřit svou bakalářskou práci na návrh hardwaru pro FPGA obvod. Jako téma bakalářské práce jsem si vybral návrh hardwaru, který realizuje základní 3D grafické operace s výsledkem zobrazeným na monitoru. V první fázi se programuje program, který pomocí wireframové grafiky vykresluje transformované prostorové obrazce pomocí středového promítání. V programu se musí optimalizovat algoritmus pro zobrazování prostorový dat, trojrozměrné geometrické transformace a Bresenhamův vykreslovací algoritmus. Toto vše se pak využije při návrhu hardwaru, kde už není prostor pro debugování dílčích algoritmů spojených do celku. V hardwarovém návrhu realizovaný Bresenhamův vykreslovací algoritmus je v podstatě ta část, díky které lze mluvit o hardwarovém akcelerátoru. Jeho rychlost ve srovnání s rychlostí výpočtů úseček softwarově je snadno demonstrovatelná zamýšleným měřením.

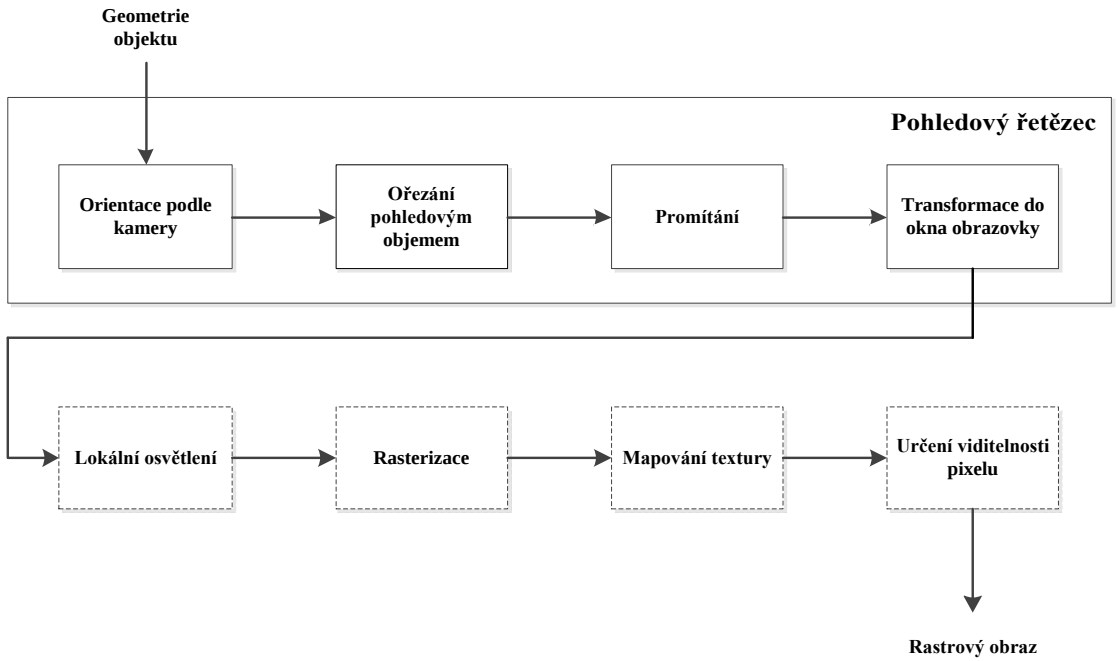
2 Zobrazování prostorových dat

V dnešní době je snaha renderovat co možná nejvíce realistické zobrazení různých prostředí na počítači. Pod pojmem rendering se skrývá velké množství vizualizačních algoritmů a dalo by se říct, že se jedná o celý proces renderování prostorové scény. Od pouhých geometrických informací o objektech, které se na scéně nacházejí, až po rastrový obraz, jež je vykreslován na zobrazovacím zařízení.

Anglické slovo rendering neboli zobrazování, je v našem případě označení pro převod trojrozměrných modelů na dvourozměrnou plochu. Celý proces zobrazování dat pak nazýváme zobrazovací řetězec (angl. Rendering pipeline). V této kapitole jsou vysvětleny základní pojmy z oblasti zobrazování prostorových dat. Jsou zde vysvětleny základní primitiva počítačové grafiky a popsány některé metody vykreslování prostorových scén na počítači. Také jsou v kapitole popsány transformační matice, které se zobrazováním prostorových dat úzce souvisejí.

2.1 Rendering pipeline

Rendering pipeline neboli zobrazovací řetězec, je řešení jednotlivých dílčích úloh, jejichž vstupem je geometrie vykreslovaných objektů a výstupem je rastrový obraz, připravený pro vykreslení. Úplný zobrazovací řetězec (Obrázek 1) zobrazuje jak se ze vstupních informací, které obsahují jen prosté souřadnice vykreslovaných obrazců, stává plnohodnotná scéna připravená k vykreslení. Pokud se zobrazuje ve wireframové grafice, je celý tento řetězec oproštěn od ořezávání pohledovým objemem, řešení otázky lokálního osvětlení, apod. Tytoúlohy jsou na obrázku vyznačeny přerušovanou čarou.



Obrázek 1: Rendering pipeline

V druhé části se řeší osvětlení scény, její rasterizace, mapování textur a úloha pro určení viditelnosti jednotlivých pixelů scény. Výsledkem je pak rastrový obraz scény, u wireframové grafiky se jedná v podstatě pouze o transformovaný drátěný model.

2.2 Základní stavební kameny počítačové grafiky

V následující kapitole jsou popsány základní stavební kameny počítačové grafiky a některé další pojmy, které se zobrazováním dat na počítači úzce souvisí. Podkapitoly jsou seřazeny hierarchicky od nejjednoduššího primitva až po celé objekty a je zde uvedeno související názvosloví.[9]

2.2.1 Vertex

V oblasti počítačové 3D grafiky je vertex bod v prostoru. V počítačové grafice je bod základní primitivum, neboli základní stavební kámen. Všechny ostatní primitiva, jakými jsou polygony apod., se skládají ze dvou či více vertexů. Vertex je nejčastěji definován v Eukleidovském prostoru souřadnicemi (x, y, z). Jelikož je vertex singulární, neboli bezrozměrný, nemá smysl u něj kromě souřadnic zaznamenávat jakékoliv další informace.

2.2.2 Úsečka

Úsečka vzniká vymezením části přímky dvěma body, v našem případě vertexy. Takové dva body se nazývají krajní body úsečky.

2.2.3 Polygon

Polygon (česky mnohoúhelník, nebo n-úhelník) se skládá z minimálně tří bodů (vertexů), z nichž žádné tři sousední body neleží na stejné přímce, a které jsou spojeny rovnými čarami (úsečkami). Polygon je zpravidla uzavřený a jedná se tedy o omezenou část roviny. Úsečky, které spojují dva sousedící body, se nazývají hrany, nebo strany polygonu. Body, ve kterých se střetávají dvě hrany, se nazývají vrcholy polygonu. Úsečky, které spojují nesousedící body polygonu, se nazývají úhlopříčkami.

2.2.4 Objekt

Složením dvou a více polygonů vzniká objekt. Přičemž jednotlivé polygony objektu se musí lišit nejméně v jednom bodě a minimálně jeden bod musí mít sousedící polygony společný. Vymezená část roviny každým polygonem se nazývá stěna objektu.

2.2.5 Wireframe

Wireframe, neboli drátěný model, je zjednodušený model, který úlohu oprošťuje od složitých transformací spojených s vykreslováním plných modelů. V případě, kdy kreslíme pouze pomocí čar a nevyplňujeme stěny obrazců, není nutné zohlednit globální a lokální osvětlení dané scény, problém svázaný s viditelností částí objektů, stíny a textury jednotlivých částí scény. Ve výsledné vyrenderované 3D scéně bude k vidění drátěný model v bíle barvě na černém pozadí (Obrázek 7).

2.2.6 Kamera

Promítací úloha řeší ve své podstatě formulaci geometrické situace, tedy určení místa, kde se nachází pozorovatel, určení polohy a orientace průmětny a stanovení směru a cíle pozorování. Kamera má totiž tu úlohu, že pořizuje snímky na průmětnu, která je vždy kolmá na hlavní optickou osu kamery. V této práci je kamera vždy umístěna na stejném místě a úhel jejího natočení se nemění. Se vzdáleností pozorovatele od cíle pozorování samozřejmě souvisí, zda se celý objekt vejde do pohledového objemu kamery. Objekty, které do tohoto pohledového objemu nezapadají, se samozřejmě nerenderují.

2.2.7 Pohledový objem

Pohledový objem je důležitou součástí zobrazovacího řetězce. Zpravidla je prováděn hned po určení orientace kamery, aby se zbytečně nepočítalo s objekty, které jsou mimo scénu. Pohledový objem je určen úhlem záběru a přední a zadní ořezávací rovinou. Vše, co leží uvnitř těchto dvou rovin a je v záběru, je podrobeno zobrazovacímu řetězci a je renderováno. Vše, co naopak leží mimo pohledový objem, je ze scény vyloučeno a dále se s takovými objekty nebo s jejich částmi nepracuje, což zaručuje, že v dalších krocích zobrazovacího řetězce se nebudou výpočty zatěžovat tím, co není v záběru kamery.

2.2.8 Mapování textur

Textura je v podstatě vlastnost povrchu objektu. Při nanesení textury na povrch objektu dochází ke změně povrchu pouze vizuálně, zatímco geometrie objektu zůstává stejná. Jedná se o jednu z konečných fází zobrazovacího řetězce, kdy jsou na objekty nanášeny dvourozměrné nebo trojrozměrné textury. Textura může mít prostý barevný charakter, ale může být reprezentována i složitými obrázky a naprosto tak změnit podobu vykreslovaného objektu.

2.2.9 Lokální a globální osvětlení scény

Modely, které se zabývají odrazem světla od jediného bodu na povrchu objektu, jsou nazývány lokálními osvětlovacími. Tyto modely se nesnaží zachytit celou scénu, ale zaměřují se na jeden konkrétní objekt. Metody pro vykreslování za pomoci lokálního osvětlování jsou oproti globálním metodám výrazně rychlejší, to je ovšem vykoupeno kvalitou renderované scény.

Metody založené na globálním osvětlování scény, se snaží detailně simulovat šíření světla ve scéně. Snaží se modelovat mnohonásobné odrazy světla mezi různými předměty, zlomy světla o části prachu ve vzduchu a útlum světla při kontaktu s objekty. Výsledkem jsou detailní realistické scény, ale s vysokou výpočetní náročností.

2.2.10 Promítání

V počítačové grafice se zobrazují trojrozměrné objekty na zobrazovací zařízení, které jsou pouze dvourozměrné. Proto je nutné všechny objekty transformovat. Jedná se o transformaci, kdy převádíme trojrozměrné objekty do dvourozměrné reprezentace. Taková transformace se nazývá promítání. Každý objekt, který je transformován

(promítán), ztrácí prostorovou informaci a tím může dojít ke zkreslení názoru pozorovatele na skutečný tvar objektu. Proto je při promítání nutno zvolit vhodný typ promítání, a pokud možno jej doplnit o další pravidla a postupy, pro zvýšení reálného vjemu promítaného objektu. Tyto pravidla a doplňující postupy řeší například viditelnost jednotlivých těles při zobrazování, ortogonální průměty, axonometrii, apod..

2.2.11 Promítací paprsek

Promítací paprsek je polopřímka, vycházející z promítaného vertexu. Směr paprsku je přímo závislý na zvolené promítací metodě.

2.2.12 Průmětna

Průmětna (anglicky viewing plane) je plocha v prostoru na kterou promítáme. Na tuto plochu dopadají promítací paprsky a v místě dopadu tak vytvářejí průmět neboli obraz v průmětně. Jelikož se pracuje zpravidla s rovinnou průmětnou, promítají se prostorové úsečky do úseček v rovině. Při transformaci tedy nedochází k zakřivení jednotlivých úseček, a proto není nutné promítat všechny body prostorových objektů. Je-li tedy promítán objekt, stačí transformovat pouze koncové body úseček a ty spojit až v průmětně.[9]

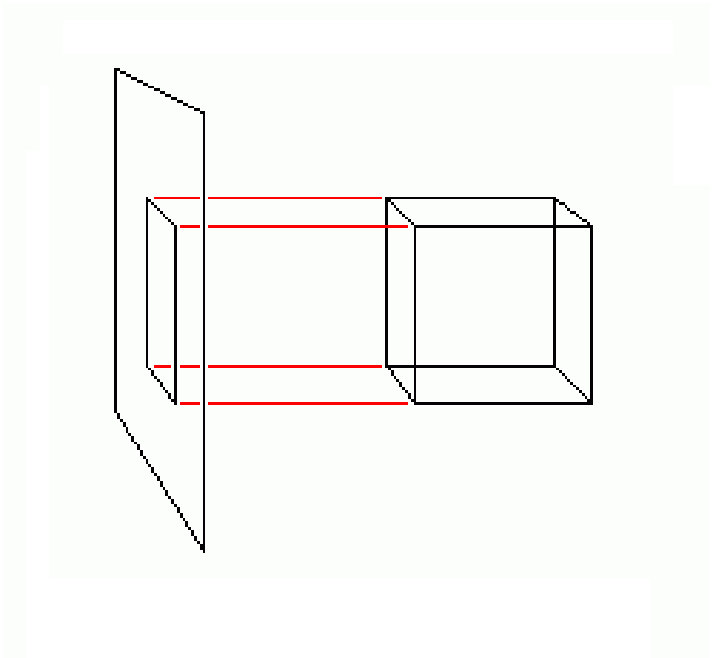
2.3 Typy promítání

V rovinném promítání jsou dvě základní třídy promítání. Jedná se o promítání rovnoběžné neboli paralelní (Obrázek 2), a promítání středové – perspektivní (Obrázek 3).

2.3.1 Rovnoběžné promítání

Rovnoběžné promítání je charakterizováno směrem promítání. To znamená, že všechny promítací paprsky mají stejný směr, ale vycházejí z různých bodů. Na obrázku je vidět, jak na průmětnu dopadají červené promítací paprsky, které vycházejí z bodů promítané krychle.

Je možno položit průmětnu tak, aby byla rovnoběžná s některou z hlavních os, v takovém případě se promítá do roviny ve směru příslušné osy. Budou-li všechny promítací paprsky kolmé na průmětnu, budou tedy mít shodný směr s normálovým vektorem průmětny, nazýváme promítaní promítáním Mongeovým.[2]



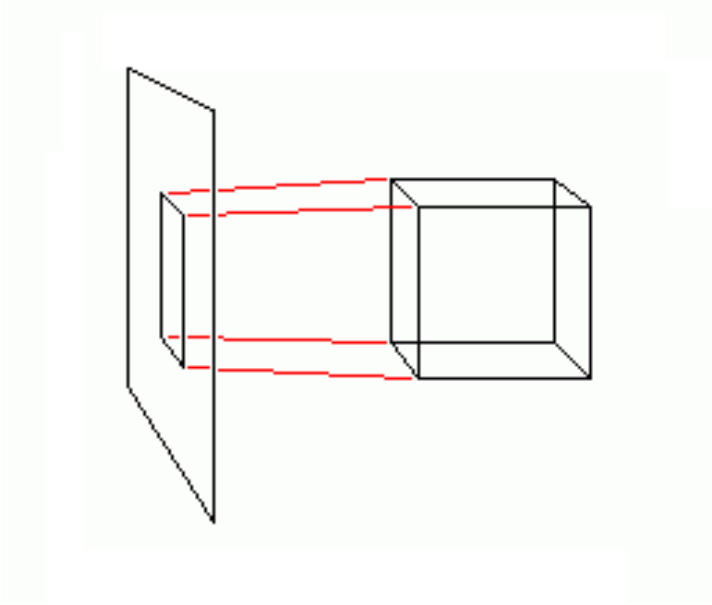
Obrázek 2: Ukázka rovnoběžného Mongeova promítání[4]

Do skupiny rovnoběžného promítání dále řadíme axionometrické promítání, u kterého se využívá průmětna, která není rovnoběžná ani s jednou z hlavních os. Průmětna tak promítá i samotné hlavní osy.

Obecné rovnoběžné promítání je označováno jako kosoúhlé promítání. Jedná se o promítání, ve kterém využíváme elementů z promítání axonometrického a Mongeova.

2.3.2 Středové promítání

Středové promítání je typ promítání, který odpovídá optickému modelu, který reprezentuje lidské vidění reálného světa. Promítací paprsky nejsou rovnoběžné a vycházejí z bodů, jejichž počet je určen druhem perspektivního promítání. Pro tento druh promítání je typické, že modeluje proporcionální zmenšování předmětů při vzrůstající vzdálenosti od pozorovatele. Díky tomu středové promítání poskytuje dobrý prostorový vjem na průmětně. Na obrázku vidíme jednobodové perspektivní promítnutí krychle na průmětnu. Kdyby byly promítací paprsky protaženy dále za průmětnu, došlo by k jejich průniku v právě jednom bodě. Takový bod se nazývá hlavní úběžník.[4]



Obrázek 3: Ukázka jednobodového perspektivního promítání[4]

Dalším typickým rysem perspektivního promítání je fakt, že nezachovává rovnoběžnost jednotlivých úseček. Pokud jsou dvě prostorové úsečky promítnuty, nezachovají si na průmětně svou rovnoběžnost (budou zobrazeny jako mimoběžky), pokud se ovšem nejedná o prostorové úsečky, které leží v rovině rovnoběžné se samotnou průmětnou. V takovém případě by na promítnutém obrazu byly také dvě rovnoběžné úsečky.

Průmětna u středového promítání může mít libovolnou polohu, kvůli praktičnosti se ale rozlišují pouze tři případy, kdy průmětna promítá jednu, dvě, tři souřadnicové osy. Podle toho, kolik os je promítáno, se rozlišuje jednobodová perspektiva, dvoubodová perspektiva a trojbodová perspektiva. Body, které figurují v názvu jednotlivých typů těchto perspektiv, se pak nazývají hlavní úběžníky.

2.4 Metody užívané v počítačové grafice

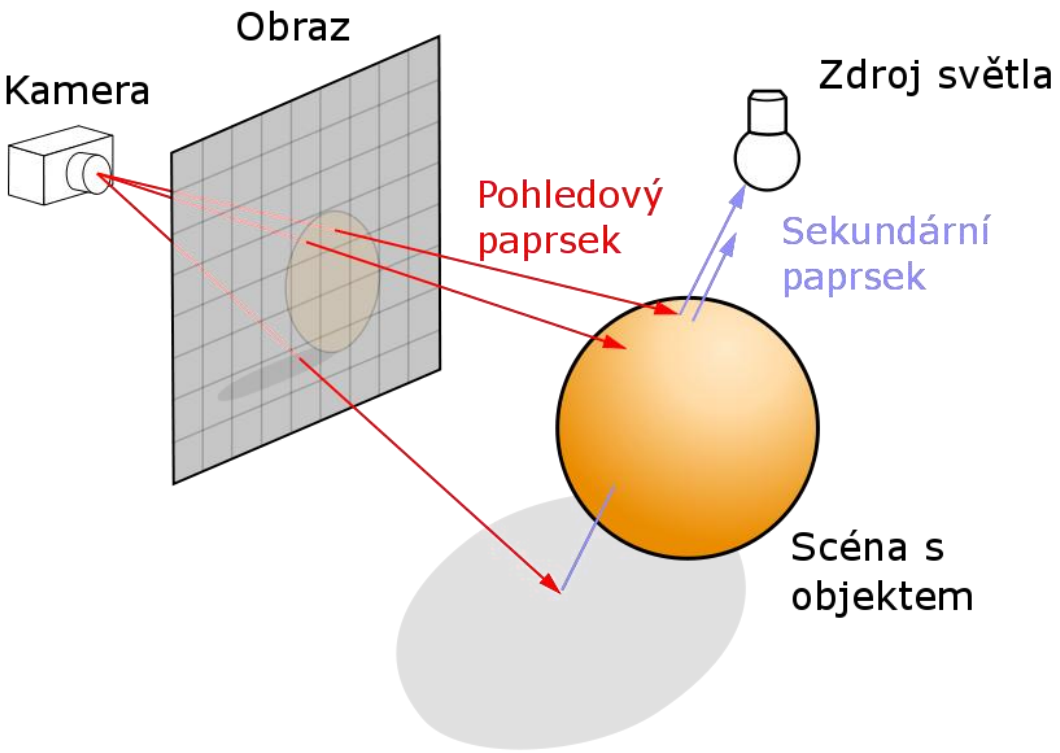
Moderní počítačová grafika, se větví na několik přístupů jak přistupovat k vykreslování 3D scény, jak vnímat paprsky světla, řešit otázku osvětlení tak, aby byla vyrenderovaná scéna co nejvěrohodnější. Většiny těchto algoritmů může být považována za simulaci fyzikálního šíření světla ve skutečném světě. Některé vybrané metody globálního osvětlování scény jsou popsány dále v této kapitole.[1]

2.4.1 Ray Casting

Ray Casting je metoda, která se dříve používala pro vykreslování prostorové scény. U této metody dochází k vysílání pomyslných paprsků z kamery (od pozorovatele) směrem na scénu. Každý paprsek se sleduje do doby, než protne nejbližší z objektů renderované scény, neboť další objekty nejsou z místa pozorovatele viditelné, pokud ovšem není první protnutý objekt průhledný. V místě kontaktu pomyslného paprsku s povrchem renderovaného objektu se provede výpočet světelných poměrů, podle dalších pomyslných paprsků, které směřují k jednotlivým světelným zdrojům. Pokud je zdroj světla z místa průsečíku viditelný, určí se barva na základě odchylky od normálového vektoru, nebo pokud je zdroj zakrytý, vytvoří se stín.

2.4.2 Ray Tracking

Metoda Ray Tracking je v podstatě rozšíření metody Ray Casting. Tato metoda umožňuje dodávat renderovaným scénám velmi reálnou podobu. Přidáním efektů reflexe (odrazy světla) a refrakce (lámání světla) lze vyrenderovat scény, které jsou od fotografie takřka k nepoznání. Toho všeho je dosaženo tím, že kromě pomyslných paprsku zmíněných u předchozí metody, jsou sledovány další sekundární paprsky, jejich počet může být prakticky nekonečný. Sledují se odrazy hlavně odrazy paprsků od objektů a jejich další průsečíky s dalšími renderovanými objekty. Do algoritmu pro vykreslování se přidává maximální počet odrazů, za účelem snížení náročnosti výpočtu. Za pomocí této metody je možno vykreslit velice realistické prostorové scény, ale samozřejmě s o mnoho vyššími výpočetními nároky. Mezi hlavní nevýhody této a zároveň i předchozí metody patří například ostré vykreslení stínů, bodové zdroje světla, nebo fakt, že při jakékoliv změně ve scéně (např. změna pozice pozorovatele) je nutné celou scénu renderovat znovu.[1]



Obrázek 4: Ukázka principu Ray Trackingu[6]

2.4.3 Photo Mapping

Další použitelný algoritmus pro vykreslení prostorové scény je Photo Mapping. Dalo by se říci, že se jedná o dvou průchodový algoritmus, protože algoritmus nepočítá pouze s paprsky, které jsou vyslány z kamery, ale také s paprsky, které pocházejí ze světelného zdroje. Při splnění všech podmínek algoritmu je v případě propojení obou paprsků spočítána hodnota záření a určena barva daného bodu. Tato metoda je vhodná pro vykreslování scén, kde světelné paprsky prochází průsvitnými objekty, jako jsou například sklo, nebo voda. Oproti předchozím metodám má také velikou výhodu v tom, že je zde možnost vzájemné reflexe mezi osvětlovanými objekty. To znamená, že světlo je schopno osvětlit další objekty i po průchodu průhledným objektem. Jedná se o výrazně náročnější metody, u které zatím nejsou možnosti real-timového vykreslování.

2.4.4 Radiozita

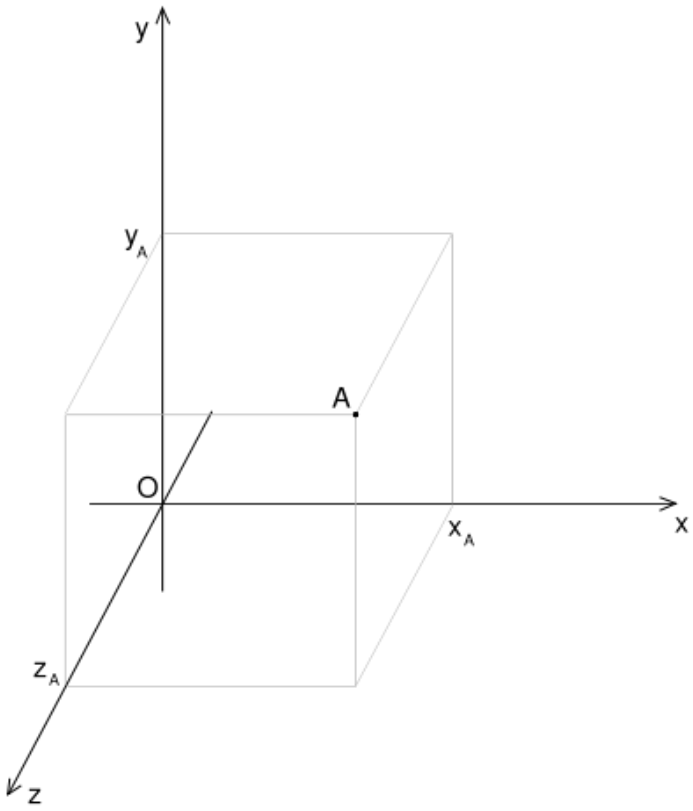
Další známa metoda pro vykreslení prostorové scény v počítačové grafice se nazývá radiozita. Protože je tato metoda založena na zákonu zachování energie, vyžaduje, aby renderovaná scéna byla energeticky uzavřená. Proto s touto metodou nejde vykreslovat

scény, které obsahují průhledné objekty, zrcadla, nebo objekty s texturou. Vykreslovaná scéna navíc musí být reprezentována polygonálním modelem.

2.5 Definice bodu v prostoru

Všechny body, se kterými pracujeme, jsou definovány souřadnicemi, které jednoznačně určují jejich polohu na všech třech osách. Je-li dán tedy bod trojrozměrného prostoru A (Obrázek 5), souřadnice, které polohu toho bodu jednoznačně určují, jsou X_A, Y_A a Z_A.

Platí tedy že: $A = [X_A, Y_A, Z_A]$



Obrázek 5: Bod v prostoru

2.5.1 Homogenní souřadnice

Abychom si zjednodušili výpočet jednotlivých transformací, budeme pro reprezentaci jednotlivých bodů používat homogenní souřadnice. Což nám umožní vyjádřit používané lineární transformace pomocí jedné matice. Skládání transformací pak lze realizovat

násobením jednotlivých transformačních matic, včetně transformační matice pro perspektivní promítání. Čtveřice čísel [x, y, z, w] pak představuje homogenní souřadnice bodu P, který má své kartézské souřadnice [X, Y, Z] v trojrozměrném prostoru, za podmínky že platí:

$$X = \frac{x}{w}, Y = \frac{y}{w}, Z = \frac{z}{w}, w \neq 0$$

Význam homogenních souřadnic je hlavně u výpočtu translace jednotlivých objektů, kde by se bez homogenních souřadnic nedošlo ke správnému výsledku. Velký význam mají dále homogenní souřadnice u řešení problému viditelnosti jednotlivých bodů.[9]

2.6 Trojrozměrné geometrické transformace

Geometrické transformace můžeme rozdělit na lineární a nelineární. V jednoduché čárové grafice si vystačíme pouze s lineárními transformacemi, jako jsou posunutí, rotace, či změna měřítka. Všechny tyto transformace používáme současně s převodem trojrozměrné scény do roviny obrazu. My budeme transformovat bod P, který má kartézské souřadnice [X, Y, Z] v našem trojrozměrném prostoru. Po aplikaci dané transformace tak získáme bod P' se souřadnicemi [X', Y', Z'].

Obecnou matici 4 x 4 reprezentující lineární transformaci bodu P = [x, y, z, w] na bod P′ = [x′, y′, z′, w′] budeme označovat A.

Transformaci souřadnic můžeme zapsat takto:

$$\mathrm{P'} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = A_{4x4} \cdot P = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

2.6.1 Posunutí

Posunutí v prostoru závisí na vektoru posunutí $\vec{p} = (X^P, Y^P, Z^P)$, přičemž transformační matice posunutí má pak následující tvar:

$$P = P(X^P, Y^P, Z^P) = \begin{bmatrix} 1 & 0 & 0 & X^P \\ 0 & 1 & 0 & Y^P \\ 0 & 0 & 1 & Z^P \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.6.2 Rotace

Při rotaci v prostoru se jedná o jeden ze tří případů otáčení kolem jednotlivých souřadnicových os. Matice R_x tak představuje otáčení okolo osy *x* o úhel α:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotace okolo osy *y* o úhel β a *z* o úhel γ jsou pak sestaveny obdobně:

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.6.3 Změna měřítka

Změnu měřítka pak popisuje transformační matice S(s_x, s_y, s_z), kde nenulové koeficienty s_x, s_y, s_z určují změnu ve směru příslušné souřadnicové osy:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

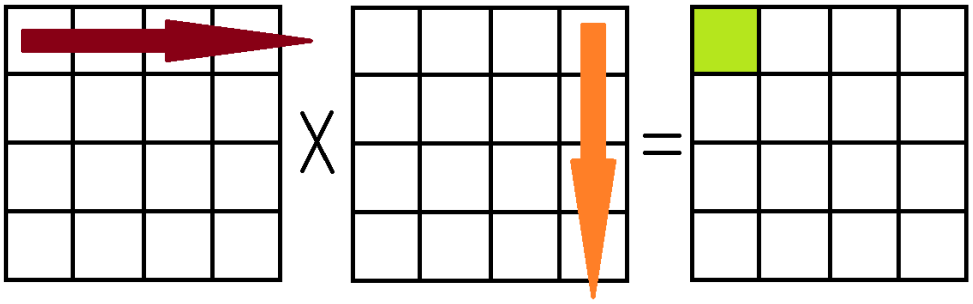
2.6.4 Vzájemné násobení dvou matic

Pokud složíme čtyři body do jedné matice, je možné násobit pak všechny tyto čtyři body najednou s transformační maticí, které vzniká vynásobením jednotkové matice jednotlivými, výše popsanými, transformačními maticemi. Výsledkem pak jsou čtyři transformované body. Algoritmus pro vzájemné vynásobení dvou matic je následující.

Matice *A* má rozměr *m* x *n* a matice *B* má rozměr *n* x *k*. Součin těchto dvou matic bude mít ve výsledku rozměr *n* x *k*. Výsledná matice C je definována následujícím vztahem:

$$C = (A \cdot B)_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj}$$

Což platí pro všechny prvky (*i*,*j*) výsledné matice *C*.



Obrázek 6: Součin dvou matic

2.7 Středové promítání

Středové (perspektivní) promítání je počítačovou paralelou vyjadřující vidění reálného světa lidským okem. Vnáší do zobrazování perspektivu, zmenšování objektů s rostoucí vzdáleností od pozorovatele. Pozorovatel je umístěn do počátku souřadnicového systému, čímž se průmětna stává kolmá na osu z. Z toho vyplývá, že objekty, které leží za průmětnou, jsou při zobrazování zmenšeny a objekty, které leží naopak před průmětnou, jsou ve stejném poměru zvětšeny. Objekty, které leží přímo na průmětně, si pak svojí velikost zachovají.

2.7.1 Lineární transformace středového promítání

Na obrázku 1.1 vidíme bod $P[x, y, z, 1]$ promítnut na bod $P_p[x_p, y_p, z_p, 1]$. Bod P je na bod P je na bod P_p transformován pomocí:

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w_p \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_0} & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = T_{perspektiva} \cdot P$$

To znamená, že perspektivní promítání lze vyjádřit v homogenních souřadnicích za pomocí lineární transformace.

Je-li středové promítání vyjádřeno v homogenních souřadnicích pomocí lineární transformace, pozorovatel je uvažován v bodě $z = z_0$ a průmětna je přemístěna do počátku souřadnicového systému (je ztotožněna s rovinou *xy*), pak musí platit:

$$x_p = x \cdot \frac{z_0}{z_0 + z} = \frac{x}{1 + \frac{z}{z_0}}, y_p = y \cdot \frac{z_0}{z_0 + z} = \frac{y}{1 + \frac{z}{z_0}}$$

3 Vlastní řešení

3.1 Softwarové řešení

Jelikož se jedná o komplexnější práci, je nutné ověřit správnost transformačních matic a dílčích výsledků, než se začne s návrhem designu a jeho implementací, aby se předešlo časově náročnému ladění při práci s integrovaným obvodem.

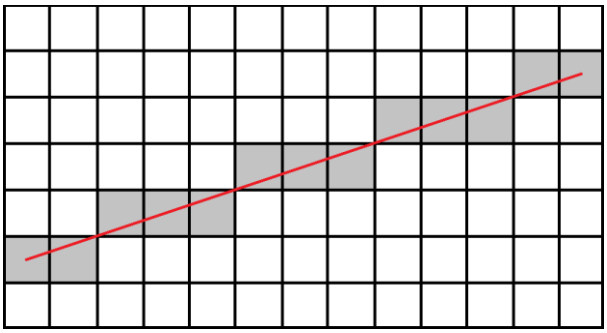
Celý problém proto nejprve řešíme softwarově a to konkrétně v jazyce C/C++ ve spojení s knihovnou SDL, která umožňuje snadné vykreslování a jednoduché testování vlastního vykreslovacího algoritmu. Je zde prostor i pro porovnání algoritmů DDA a Bresenhemova algoritmu (Obrázek 7). Na obrázku jsou vykresleny dvě krychle. Krychle nalevo je vykreslena pomocí Bresenhamova algoritmu, kdežto krychle napravo je vykreslena pomocí algoritmu DDA. Bresenhemův algoritmus počítá s takzvaným chybovým členem a určuje vždy pixel, který je bližší k reálné nerasterované úsečce. Dále zde provádíme optimalizaci vhodnějšího Bresenhamova algoritmu, vybrání optimálního postupu násobení matic apod..

Obrázek 7: Vlevo použit Bresenhamův algoritmus, na pravo algoritmus DDA

Získané algoritmy jsou dále samozřejmě využity, jelikož transformace bodů jsou prováděny softwarově.

3.1.1 Bresenhamův algoritmus

Bresenhamův algoritmus zjišťuje, který bod má být na mřížce vybarven, aby celková reprezentace čáry byla co nejbližší rovné čáře vedené mezi dvěma body. Běžně se tento algoritmus využívá ke kreslení čar v počítačové grafice. Všechny výpočty Bresenhamova algoritmu jsou přitom provedeny celočíselnou arimetrikou.



Obrázek 8: Ilustrace výsledku Bresenhamova algoritmu pro kreslení úseček

3.1.1.1 Pseudokód Bresenhamova algoritmu

Pokud je absolutní hodnota (*y1-y0*) větší než absolutní hodnota (*x1-x0*) tak
se signál *steep* nastaví do log. jedničky
Pokud je *steep* v logické jedničce tak
    se prohodí souřadnice *x0* s *y0* a *x1* s *y1*
Pokud je *x0* větší než *x1* tak
    se prohodí souřadnice *x0* s *x1* a *y0* s *y1*
deltaX se nastaví na hodnotu *x1-x0*
deltaY se nastaví na absolutní hodnotu (*y1-y0*)
error se nastaví na hodnotu *deltaX* / 2

Na začátku se uloží do *y* hodnota *y0*
Pokud je *y0* menší než *y1 tak*
    Se do *ystep* uloží 1, jinak -1
Po úvodním nastavení a prohození je spuštěn cyklus hodnota *x* se v každém cyklu inkrementuje od *x0* do x1
        pokud je *steep* roven log. jedničce tak
            se vykreslí prohozené souřadnice *y,x*
            jinak se vykreslí neprohozené souřadnice *x,y*
        od hodnoty *error* se odečte *deltaY*
        a pokud je *error* menší než 0 tak
            k *y* se přičte *ystep*
            k *error* se přičte *deltaX*

3.1.2 DDA (Digital differential algorithm) algoritmus

DDA je jednoduchý přírůstkový algoritmus pro výpočet bodů úsečky. Ve směru jedné osy s větším přírůstkem inkrementuje o krok jedna a ve směru druhé osy o krok menší než jedna podle poměru délky úsečky ve směru osy *x* a ve směru osy *y*. DDA algoritmus počítá v oboru reálných hodnot, což si vyžaduje použití FPU, která výpočty reálných čísel značně urychluje. Pokud FPU není k dispozici, bývá tento problém řešen ve formátu fixed-point.

3.1.2.1 Pseudokód DDA algoritmu

deltaX se nastaví na hodnotu *X1-X0*
deltaY se nastaví na hodnotu *Y1-Y0*
koef se nastaví na hodnotu *deltaX / deltaY*
Pokud *koef* leží v intervalu <-1 , 1> tak
    Dx se nastaví na hodnotu 1
    Dy se nastaví na hodnotu *koef*
    Souřadnice pro další krok jsou vypočítány následovně
        $X_{i+1} = X_i + Dx = X_i + 1$
        $Y_{i+1} = Y_i + Dy = Y_i + k$
Pokud *koef* leží v intervalu $(-\infty, -1)$ nebo v intervalu $(1, \infty)$ tak
    $Dx = 1/koef$
    $Dy = 1$
    Souřadnice pro další krok jsou vypočítány následovně
        $X_{i+1} = X_i + Dx = X_i + 1/koef$
        $Y_{i+1} = Y_i + Dy = Y_i + 1$

3.2 Návrh hardware

3.2.1 Programovatelná hradlová pole

Programovatelná hradlová pole, anglicky Field Programmable Gate Array (FPGA), jsou obvody s architekturou patřící do skupiny elektricky reprogramovatelných PLD obvodů. FPGA obvody mají architekturu založenou na malých generátorech logických funkcí s pamětí (takzvaných LUT tabulkách), klopných obvodech a spoustě vertikálních a horizontálních propojení. V současně největších programovatelných hradlových polích se nalézají stovky tisíců LUT tabulek a klopných obvodů. Obvody od firmy Xilinx, využívající technologii SRAM, lze je tak libovolně krát reprogramovat. Kromě výše zmíněných částí je možno uvnitř FPGA obvodů naleznout ještě další elementy, jako jsou například vložené bloky násobiček, nebo vložené bloky paměti. Tyto bloky se

dají nakonfigurovat podle potřeby do různých paměťových funkcí, jako jsou například RAM, ROM, nebo FIFO.[5]

3.2.2 Xilinx: XUPV5-LX110T

Xilinx XUPV5-LX110T je všestranná vývojová deska, která je osazena FPGA obvodem Virtex-5. Přípravek nabízí spoustu různých možností například v oblastech vyhodnocování získaných informací a vývoje průmyslových zařízení. Na přípravku nalezneme většinu průmyslově standardních rozhraní. V práci je využita pouze FPGA Virtex-5 včetně statické paměti ve vnořených blocích BRAM, paměť DDR2 a čip DVI s příslušným konektorem.[8]

3.2.2.1 Virtex-5 LX FPGA

Jedná se o velmi výkonný FPGA obvod z rodiny Virtex předposlední generace. Frekvence systému je nastavena na 125MHz. Avšak správnou optimalizací by se dala tato hodnota ještě navýšit. Toto FPGA má kromě standardních struktur vestavěné funkce, které ještě zvyšují možnosti zařízení.[8]

3.2.2.2 Obvod DVI

Monitor je připojen k přípravku přes DVI konektor, který je přímo napojen na DVI obvod. Ten využívá Chrontel CH7301C, který je schopný vykreslit rozlišení až 1600 x 1200, v 24-bitové hloubce barev. DVI obvod umí řídit digitální i analogový signál, takže je možné připojit starší VGA monitor za použití DVI – VGA adaptéru. Pro VGA a DVI signál je ovšem nutné specificky nastavit DVI čip a proto nelze při stejném nastavení libovolně prohazovat zobrazovací zařízení.

3.2.2.3 Paměť DDR2

Použitý přípravek disponuje pamětí DDR2 (Double Data Rate), jejíž velikost je 256MB. DDR2 jsou implementovány v SODIMM modulech, přes jejichž vývody je komunikace s pamětí realizována. Paměť DDR2 je navíc oproti starší paměti DDR až dvakrát rychlejší, protože sběrnice, kterou tyto moduly využívají je taktována dvakrát rychleji.

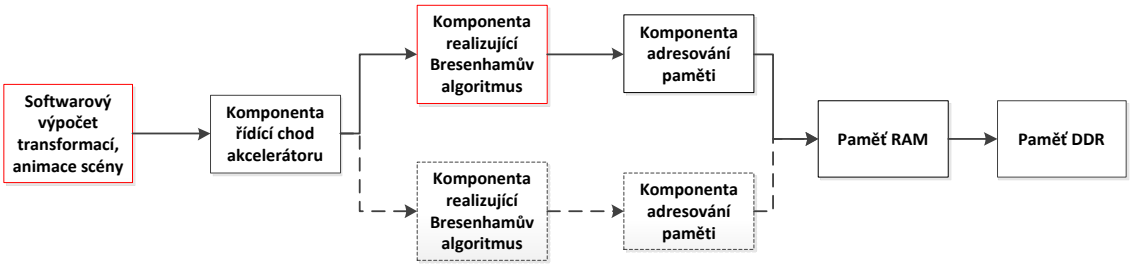
3.2.2.4 Procesor MicroBlaze 8.1

Vývojové prostředí Embedded Processor Development Kit (EDK) umožňuje implementovat 32-bitový softwarový procesor MicroBlaze. Samotný procesor

umožňuje řadu nastavení, čímž se zvětšuje pole možností jeho využití. Mezi stálé části procesoru patří třicet dva 32-bitových registrů, 32-bitové instrukční slova se třemi operady a dvěma adresníma módama, 32-bitová adresová sběrnice a samostatnou datovou linku.[8]

3.2.3 Popis systému

Celý systém lze rozložit na menší funkční bloky, které v systému plní konkrétní úlohy. Jelikož by kompletní návrh celého systému a jeho uvedení do provozu bylo velmi náročné, práce se zabývá jen vybranými částmi systému, které jsou po dokončení zasazené do již fungujícího systému. Na schématu systému (Obrázek 9) jsou červeně vyznačeny bloky, jejichž vývoj je hlavní prací této práce. Všechny funkční bloky jsou vysvětleny v následujícím textu.



Obrázek 9: Schéma systému

3.2.3.1 Softwarový výpočet transformací, animace scény

Prvním funkčním blokem je ta část systému, která se stará o výpočet všech 3D grafických transformací a o animaci scény. Toto vše je řešeno softwarově v soft procesoru Microblaze 8.1, který byl popsán výše. V tomto funkčním bloku také dochází k výpočtu Bresenhamova algoritmu, pokud nevyužíváme k výpočtu bodů úseček hardwarový akcelerátor.

3.2.3.2 Komponenta řídící chod akcelerátoru

Druhý funkční blok je komponenta, která řídí chod dalších komponent, především se jedná o komponentu realizující Bresenhamův algoritmus. Tato komponenta vysílá signály o tom, že na vstupu komponenty realizující Bresenhamův algoritmus jsou správné souřadnice a může začít výpočet jednotlivých bodů. Pokud by byl výpočet bodů úseček paralelizován, o obsluhu přidaných komponent, by se opět starala, tato komponenta. Komponenta se také stará o komunikaci s procesorem MicroBlaze.

3.2.3.3 Komponenta realizující Bresenhamův algoritmus

Tento funkční blok se stará o výpočet jednotlivých bodů úseček, jejichž souřadnice do komponenty vstupují. Celá komponenta je popsána dále v textu.

3.2.3.4 Komponenta adresování paměti

Z předchozí komponenty do toho funkčního bloku přicházejí souřadnice jednotlivých bodů úsečky a signál o tom, že je možno tyto souřadnice zapsat do paměti RAM. Vypočítané souřadnice ukazují na přesně jeden bit v paměti, kde je potřeba vykreslit bod úsečky, a proto je zde nutnost převést souřadnice na adresu. Paměť je rozdělena do čtyř sloupců (čtyři instance). Celková velikost je rovna 19 bitovému číslu, protože takové číslo stačí na zápis celého vyrenderovaného obrazu, samozřejmě s určitou rezervou na další informace o konfiguraci apod.. Jelikož paměť RAM je nastavena jako dual portová, je možno ji adresovat binárně a po bajtech, což umožňuje adresaci 4-bytového (čtyři instance paměti) slova – přistupuje se z 32-bitové sběrnice.

3.2.3.5 Paměť RAM

FPGA Virtex-5 obsahuje celkem 5,328Mbit (666KB) statické paměti ve vnořených blocích BRAM. Jelikož vykreslujeme scénu v rozlišení 800 x 600 a pouze ve dvou barvách (černo-bílá scéna), což odpovídá velikosti přibližně 480Kb, vejde se celá vyrenderovaná scéna do RAM paměti najednou. Pro vyšší rozlišení nebo při použití více barev, by bylo nutné využít alternativní práce s pamětí, o těchto možnostech se píše dále v kapitole Alternativní řešení práce s pamětí. Pro smazání scény, v okamžiku kdy jsou vyrenderovány všechny segmenty scény, dochází k vynulování celé RAM paměti.

3.2.3.6 Paměť DDR2

Z paměti RAM, je vyrenderovaná scéna nahrána do paměti DDR2. Odkud je dále vykreslována na obrazovku pomocí DVI obvodu. Dalo by se říci, že paměť RAM v systému funguje jako určitě vyrovnávací paměť, aby nedocházelo ke ztrátě informací.

3.2.3.7 Možná paralelizace části systému

Jak již bylo zmíněno výše, je možné dosáhnout většího výkonu systému, využije-li se možnosti paralelizace některých části. Konkrétně se jedná o zvýšení počtu komponent, které realizují bresenhamův algoritmus pro počítání souřadnic bodů jednotlivých čar. Každá další taková komponenta by si samozřejmě vyžádala přidání navazujících

komponent pro adresaci RAM paměti a přidání časového multiplexingu, čímž by se zamezilo pokusu o nahrávání více souřadnic do paměti najednou. Na schématu systému (Obrázek 9) je možná paralelizace znázorněna přerušovanou čarou, přičemž je by samozřejmě bylo možné přidat větší počet daných komponent.

3.2.4 Bresenhamův vykreslovací algoritmus

V práci je použit Bresenhamův vykreslovací algoritmus pro vykreslování úseček, který po zadání souřadnic počátečního a konečného bodu spočítá jednotlivé body úsečky. Tento algoritmus je pro naší úlohu vhodnější než například algoritmus DDA. Jedním z důvodů je ten, že algoritmus využívá takzvané predikce, a proto vypadají výsledné vykreslené úsečky více spojitě. Laicky řečeno algoritmus hledá body, které leží nejblíže skutečné úsečce. Druhá výhoda je, že vše navíc počítá pouze pomocí celočíselné aritmetiky, což zvyšuje rychlost výpočtů a to je další důvod, proč se v počítačové grafice používá právě tento vykreslovací algoritmus a proč je tento algoritmus použit v této práci.

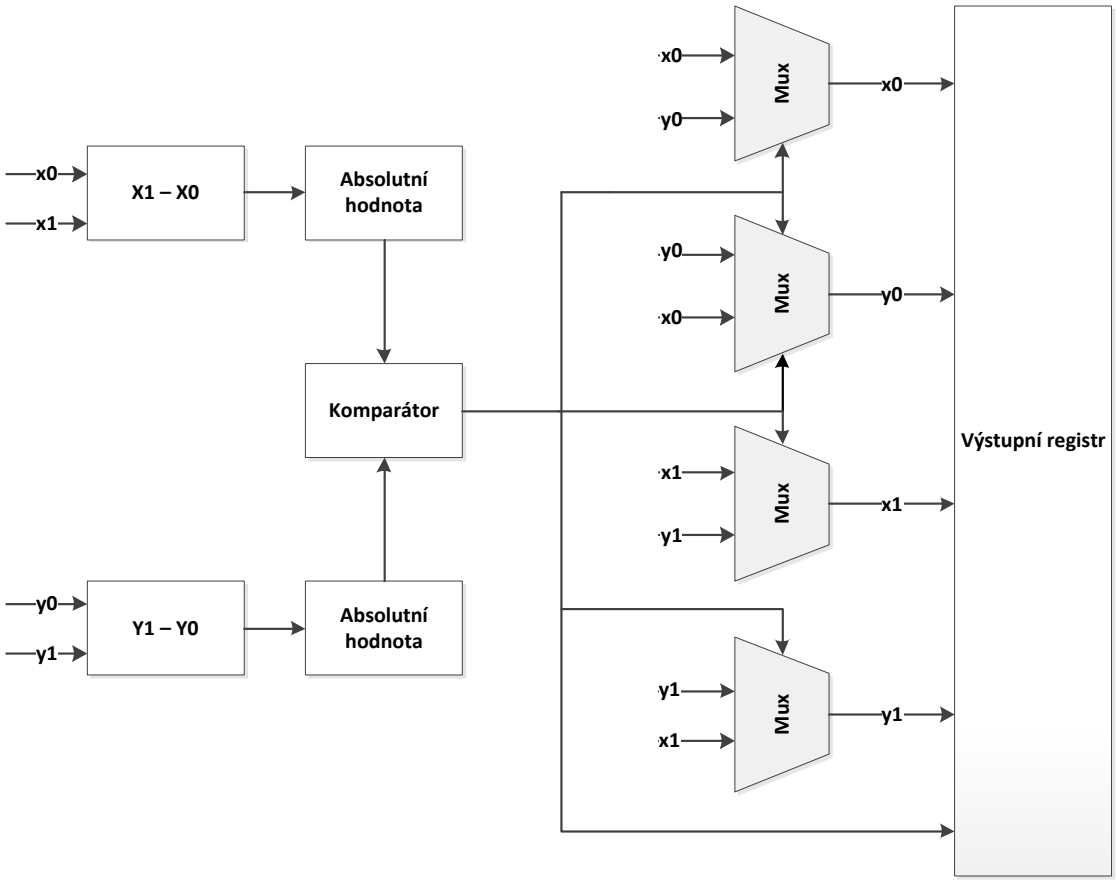
3.2.4.1 Popis

Komponenta, která realizuje Bresenhamův vykreslovací algoritmus je rozdělena na tři funkční bloky. V prvním funkčním bloku se v případě potřeby prohazují x-ové a y-ové souřadnice, aby se iterovalo po delší ose úsečky, je nutné uchovat informaci o tom, jestli se toto prohození provedlo, protože v takovém případě je nutné prohodit vypočítané souřadnice zpět. V druhé části se prohazují pouze počáteční a koncové souřadnice, aby se počítalo od nižší souřadnice k vyšší. Třetí část je opatřena stavovým automatem a zajišťuje samotný iterační výpočet souřadnic jednotlivých bodů úsečky. Samotná komponenta má pak vstupy – hodinový signál, reset a souřadnice počátečního a koncového bodu úsečky. Výstupy komponenty jsou vypočítané souřadnice jednoho bodu úsečky, signál umožňující zápis do paměti RAM a signál informující o dokončení práce komponenty. Komponenta je řízena signálem, který říká, kdy může začít s výpočtem souřadnic. Podrobný popis všech funkčních bloků je uveden v následujícím textu.

3.2.4.2 Bresenham: První funkční blok – zajištění strmosti úsečky

V tomto funkčním bloku se zjišťuje příkrost dané úsečky. To, jestli je její délka na souřadnicové ose *x* delší než délka na souřadnicové ose *y*. To znamená, že se volí, zda

se bude v dalších výpočtech iterovat pomocí x-ových nebo y-ových souřadnic. Porovnává se absolutní hodnosta rozdílů souřadnic na jednotlivých osách. Z komparátoru je signál přiveden do čtyř multiplexorů, které prakticky realizují samotné prohození souřadnic, je-li signál *steep* nastaven na logickou jedničku. Porovnané a případně prohozené počáteční i konečné souřadnice úsečky, spolu s hodnotou signálu *steep* uložíme do registru a vyšleme signál, o tom, že první funkční blok (Obrázek 10) proběhl v pořádku a souřadnice jsou připraveny pro vstup do druhého funkčního bloku.

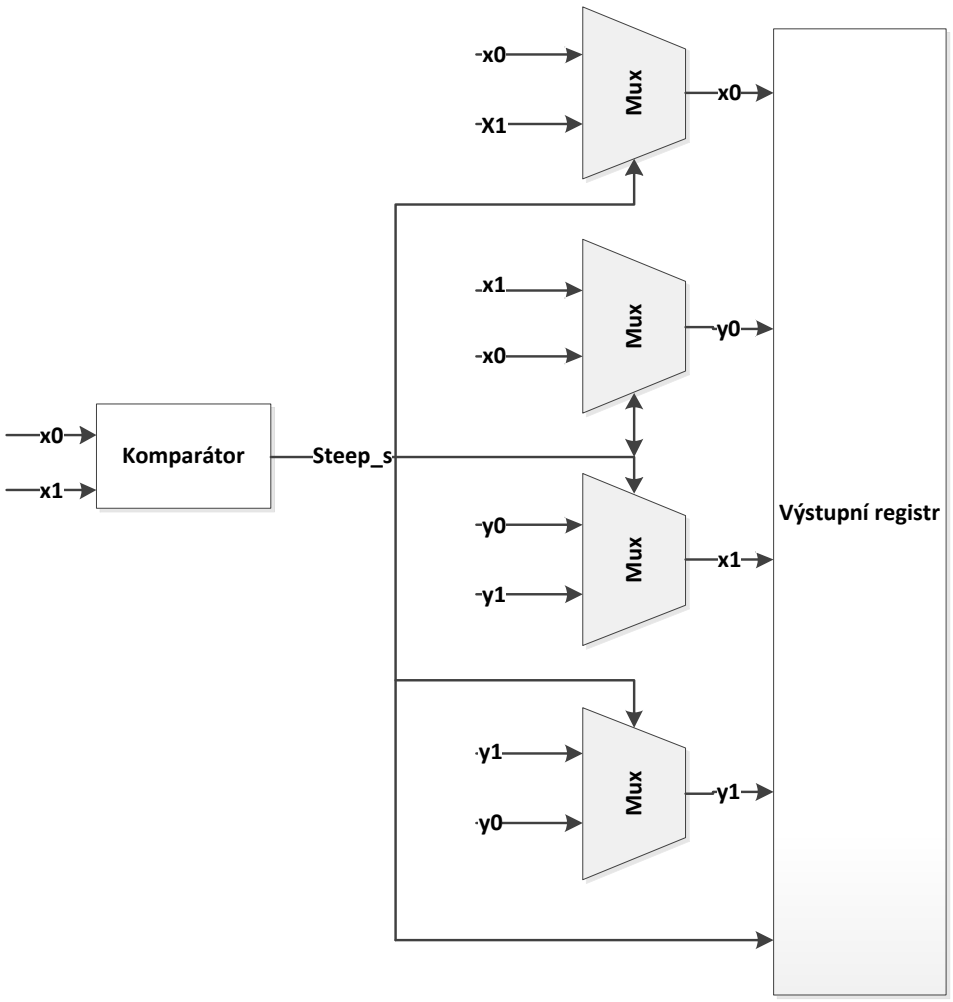


Obrázek 10: Schéma prvního funkčního bloku Bresenhamova algoritmu

3.2.4.3 Bresenham: Druhý funkční blok – zajištění směru inkrementace

V druhé funkční části komponenty (Obrázek 11) se porovnávají x-ové souřadnice, aby se vykreslovalo od zdola nahoru, zde se musí dát pozor, jelikož x-ová souřadnice roste směrem od horního levého rohu obrazovky, a proto je-li počáteční souřadnice výš na obrazovce. Je-li signál vedoucí z komparátoru do multiplexorů nastaven v logické jedničce, vzájemně prohodíme x-ové souřadnice a zvlášť y-ové souřadnice. Toto prohození je opět realizované multiplexorem, který je na ně přiveden z komparátoru, co

by adresní bit. Ať už prohozené souřadnice, či nikoliv jsou poté ukládány do registru, kde jsou připravené pro závěrečnou část této komponenty, která realizuje samotné počítání jednotlivých souřadnic. S tím je opět spojeno nastavení signálu, vedoucího do třetí části komponenty, do logické jedničky, což znamená, že se druhá část provedla v pořádku a souřadnice jsou připraveny pro další použití.



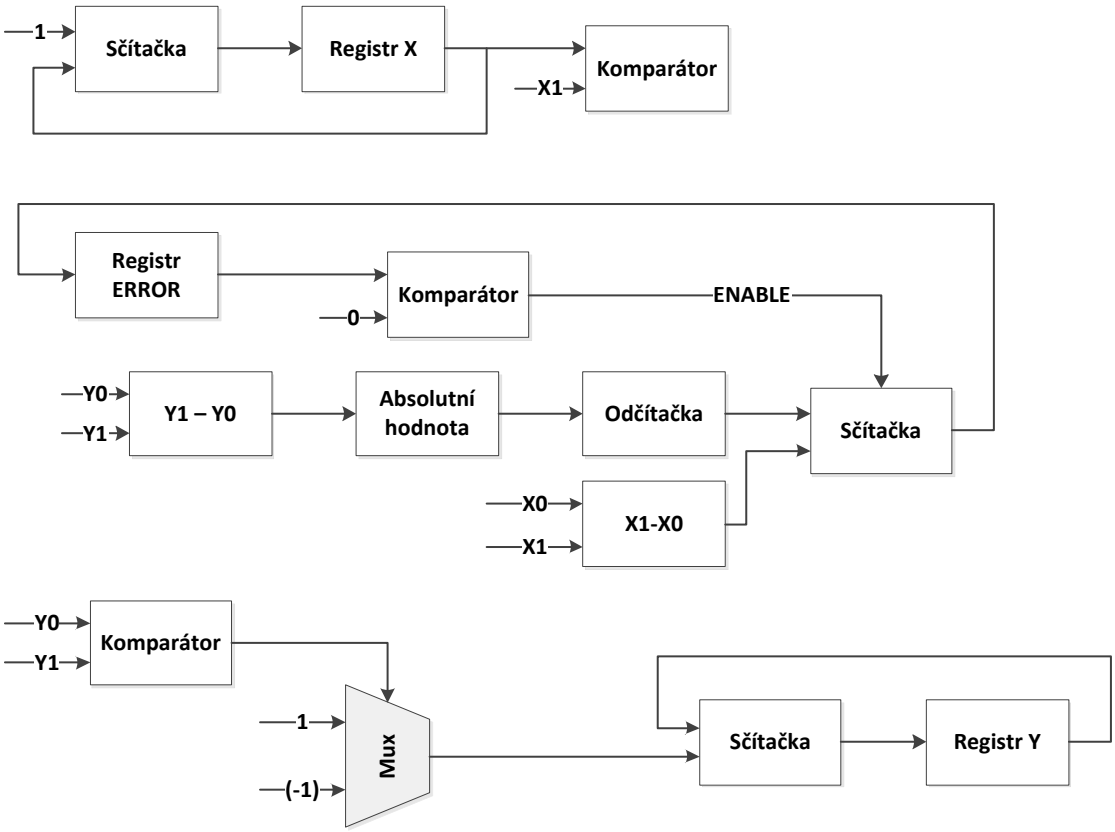
Obrázek 11: Schéma druhého funkčního bloku Bresenhamova algoritmu

3.2.4.4 Bresenham: Třetí funkční blok – inkrementační fáze a stavový automat

Třetí funkční blok (Obrázek 12) je nejsložitější, jelikož realizuje samotný výpočet souřadnic jednotlivých bodů úsečky. Chod tohoto bloku je řízen jednoduchým dvoustavovým automat. V prvním stavu automatu, se čeká, až se dokončí předchozí část a presetují se registry, se kterými se dále pracuje. Je-li automat v druhém stavu, funkční blok iterační metodou počítá souřadnice jednotlivých pixelů úsečky.

Do funkčního bloku vstupují připravené souřadnice, řídící signály z automatu a signál *steep*, který je opět připojen do multiplexorů, které řeší případné, zpětné prohození jednotlivých souřadnic. Výpočet souřadnic probíhá do té doby, než se iterovaná x-ová souřadnice nerovná konečné x-ové souřadnici, což znamená, že jsou všechny body úsečky spočteny a může se cyklus ukončit. Co se týče y-ové souřadnice, je zde komparátor, který porovnává počáteční a konečnou souřadnici y. Je-li počáteční souřadnice menší, než konečná, tak se souřadnice y inkrementuje, je-li naopak konečná souřadnice větší, souřadnici y se dekrementuje. Toto se ovšem děje pouze za předpokladu že signál jdoucí z komparátoru, který porovnává signál *error* s nulou, je nastaven do logické jedničky. Signál *error* je v podstatě rozdíl počáteční a konečné x-ové souřadnice vydělený dvěma. Od tohoto signálu v každém taktu navíc odečítáme absolutní hodnotu rozdílu y-ových souřadnic a společně se změnou hodnoty y v registru k signálu *error* přičítáme rozdíl x-ových souřadnic.

Řídící stavový automat je typu Moore, který má celkem dva stavy, tři vstupní signály a jeden výstupní. Nachází-li se automat v nultém stavu, výstupní bit je nastaven do logické nuly. Automat přechází do stavu jedna, když se na vstupu, na který je veden signál nesoucí informaci o ukončení druhého bloku, objeví logická jednička. V tomto stavu automat setrvává do té doby, než je na vstup přivedena z komparátoru porovnávající hodnotu inkrementované souřadnice x a konečné souřadnice x logická jednička. Na výstupní bit se opět uloží nula a znovu se čeká, až budou souřadnice další úsečky připravené k výpočtům.[5]



Obrázek 12: Schéma třetího funkčního bloku

3.2.5 Alternativní řešení práce s pamětí

Kdyby bylo potřeba vykreslovat například ve vyšším rozlišení a nestačila by paměť RAM pro vyrenderování celé scény uvnitř paměti, muselo by se přistoupit k alternativnímu řešení, kdy by se obrazový prostor rozdělil na bloky o vhodné velikosti. Postupně by se pak vykreslovali do RAM paměti jednotlivé bloky, které by se spojily v celý obraz scény až později v DDR paměti, kde už je samozřejmě prostor pro uložení celého obrazu a odkud je scéna vykreslována fyzicky na obrazovce monitoru. Existuje několik přístupů k tomuto přístupu, například vzor Scan-Line, nebo Tiled Rendering. Oba dva tyto přístupy jsou stručně popsány v následujícím textu.

3.2.5.1 Scan-Line

Jedná se o přístup, kdy do paměti vykreslujeme po jednotlivých řádcích, nebo sloupcích. Nepříjemností tohoto přístupu je, že je nutné u každého řádku zjišťovat, které z úseček jsou na něm skutečně renderované, a které naopak do této části obrazu nezasahují.

3.2.5.2 Tiled Rendering

Tiled Rendering, neboli „Dlážděné vykreslování“ je v dnešní době velmi často využíváno (například v mobilních technologiích), pro snížení náročnosti co do velikosti vnitřních pamětí. Jak je z názvu patrné, tento přístup vykreslování do paměti, dělí renderovanou scénu na „dlaždice“, které se vykreslují do paměti postupně. Opět je zde nižší náročnost na paměť vykoupena nutností hledat průsečíky jednotlivých úseček s okraji jednotlivých bloků scény.

3.2.6 Endianita

Při adresování paměti je také nutné dát si pozor na to, jak jsou do ní ukládána čísla, protože je-li vyvíjen komplexnější systém, může dojít k nekompatibilitě jednotlivých komponent, což ve výsledku může vést ke špatnému chodu, nebo k pádu celého systému. Pokud se tedy například v jednotlivých komponentách ukládají data do registrů a tyto komponenty jsou pak vzájemně propojeny, je nutné si před návrhem ujasnit, jaký způsob ukládání se zvolí.

Pojem endianita znamená, jakým způsobem jsou ukládána čísla, přesněji řečeno, v jakém pořadí jsou ukládány bajty číselného datového typu. Endianitu je možno také označit jako pořadí bajtů.

3.2.6.1 Little Endian

Pokud je na místo v paměti s nižší adresou uložen nejméně významný bajt a dále jsou pak uloženy další bajty až po nejvíce významný bajt, jedná se o architekturu pracující na principu Little Endianu. Mezi takové architektury se řadí například Intel X86. Na následujícím příkladu (Tabulka 1) je vidět uložení 32-bitového čísla 0x1A2B3C4D v paměti na adresu 50:

Tabulka 1: Uložení čísla 0x1A2B3C4D na adresu 50, podle Litte Endianu

	50	51	52	53	
…	4D	3C	2B	1A	…

3.2.6.2 Big Endian

Pokud je na místo v paměti s nižší adresou uložen nejvíce významný bajt a dále jsou uloženy další bajty až po nejméně významný bajt, jedná se o architekturu pracující na principu Big Endianu. Mezi takové architektury patří například SPARC, Motorola 68000. Na následujícím příkladu (Tabulka 2) je vidět uložení 32-bitového čísla 0x1A2B3C4D v paměti na adrese 50.

Tabulka 2: Uložení čísla 0x1A2B3C4D na adresu 50, podle Big Endianu

	50	51	52	53	
…	1A	2B	3C	4D	…

3.3 Měření rychlosti hardwarového akcelerátoru

Pro demonstraci funkčnosti a rychlosti hardwarového akcelerátoru se provádí několik měření a testů. Měření je prováděno s využitím systémového časovače, takže je nutné provést převod na standardní jednotky času.

Při současném nastavení procesor pracuje na frekvenci 125MHz, která odpovídá periodě 8ns.

Výsledky všech testů jsou uloženy na přiloženém CD.

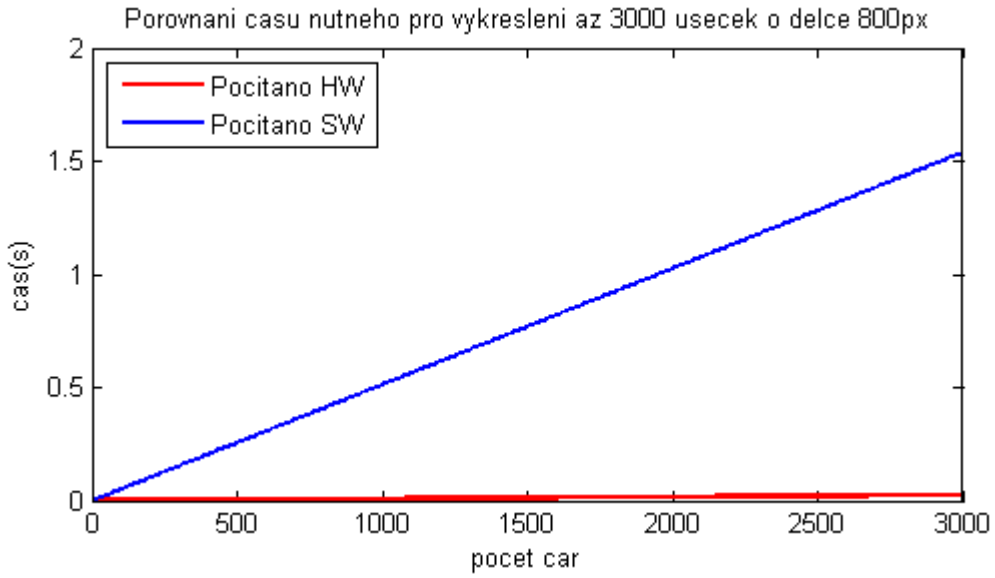
3.3.1 Rychlost vykreslení rostoucího počtu úseček

První provedené měření se týká rychlosti vykreslení rostoucího počtu čar. Počet čar se lineárně zvyšuje po deseti. Začíná se na desíti čarách a končí na třech tisících. Jelikož se čas vykreslení lineárně zvyšuje s počtem čar, mohou se naměřené hodnoty zakreslené do grafu proložit. Jsou provedeny testy pro různé délky úseček.

3.3.1.1 Úsečky o délce 800 pixelů

Nejprve jsou vykreslovány úsečky o délce 800p, jedná se tedy o maximální možnou délku vodorovných rovnoběžných úseček.

Z grafu (Obrázek 13) je na první pohled patrné, že hardwarový akcelerátor, v grafu zakreslen červenou barvou, počítá, v porovnání se softwarovým výpočtem (v grafu zakreslen modrou barvou) o mnoho rychleji. Je-li výpočet hardwarově akcelerován, je možné vykreslit tři tisíce úseček o délce 800px přibližně za 0,0248s. Pokud je výpočet prováděn čistě softwarově, dojde k vykreslení tří tisíc úseček o délce 800px přibližně za čas 1,5337s.

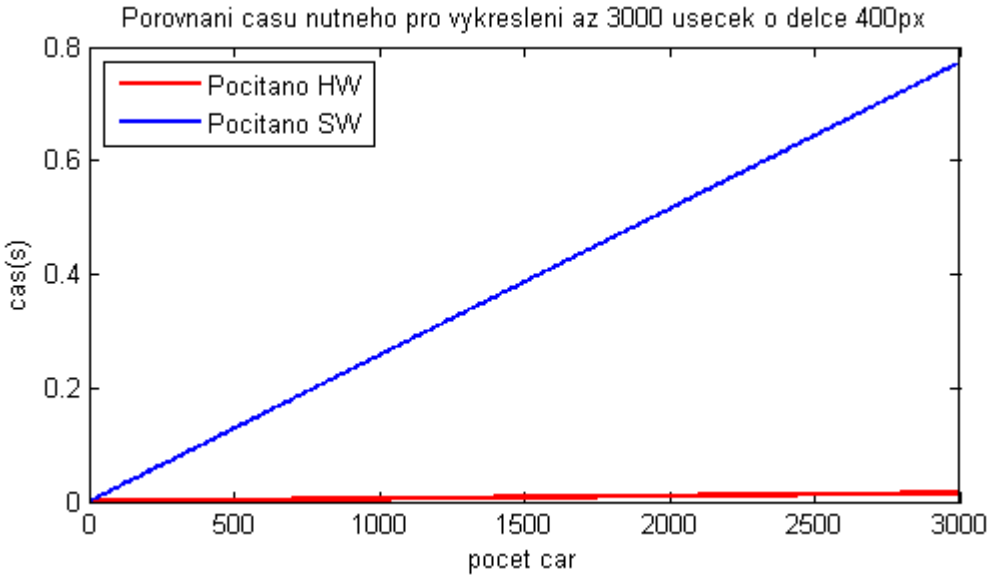


Obrázek 13: Graf závislosti času na počtu úseček dlouhých 800px

3.3.1.2 Úsečky o délce 400 pixelů

Druhé měření je provedeno při vykreslování úseček o poloviční velikost velikosti z předchozího testu.

Z grafu (Obrázek 14) je opět patrné, že hardwarový akcelerátor vykreslil všechny čáry řádově rychleji. Za pomoci hardwarového akcelerátoru bylo vykresleno tři tisíce úseček o délce 400px přibližně za 0,0151s. Softwarový výpočet vykreslil stejný počet totožných úseček za čas 0,0770s.

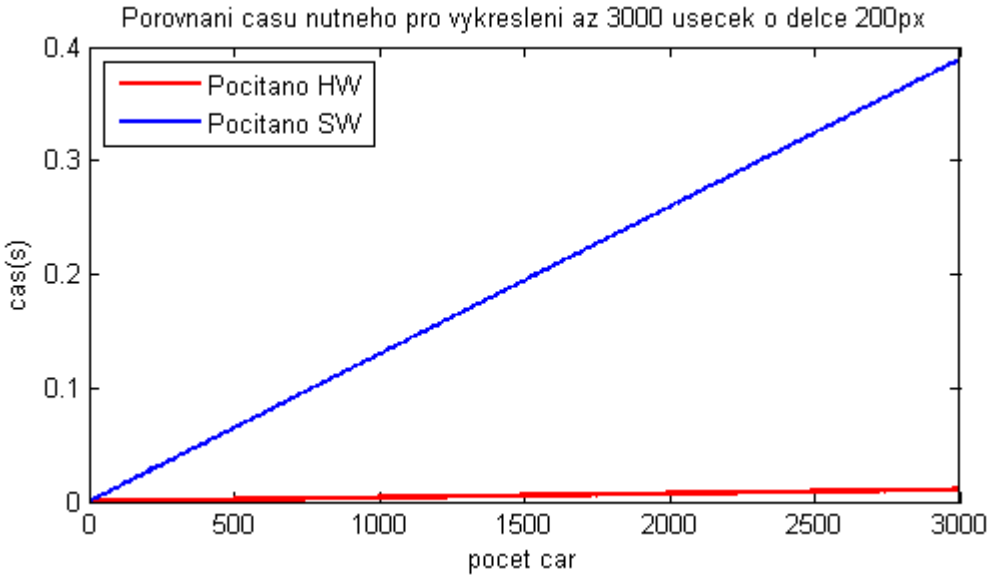


Obrázek 14: Graf závislosti času na počtu úseček dlouhých 400px

3.3.1.3 Úsečky o délce 200 pixelů

Třetí měření je provedeno při vykreslování úseček opět o poloviční velikosti.

Výsledek tohoto měření je opět ilustrován v následujícím grafu (Obrázek 15). Hardwarový akcelerátor vykreslil tři tisíce úseček o délce 200px za čas 0,0107s. Oproti tomu softwarový výpočet trval 0,3837s.



Obrázek 15: Graf závislosti času na počtu úseček dlouhých 200px

3.3.1.4 Shrnutí

Využití hardwarového akcelerátoru nemusí být vždy tou výhodnější variantou. Nutnost použití paměti RAM, do které je obraz renderován, způsobuje určité zpoždění. Při nižší délce vykreslovaných úseček, je tedy výhodnější počítat vykreslované úsečky softwarově.

Tabulka 3: Shrnutí výsledků prvního měření

Délka úseček[px]	SW[s]	HW[s]	*K=SW/HW*
800	1,5337	0,0248	61,9057
400	0,7700	0,0151	50,9708
200	0,3837	0,0107	36,4032

Z hodnot koeficientu *K* (Tabulka 3) je snadno pozorovatelné, že efektivita využití hardwarové akcelerátoru klesá společně s délkou vykreslovaných úseček.

3.3.2 Počet vykreslených čar za jednu sekundu

Další provedené měření, má za úkol zjistit, kolik čar je možno vykreslit za jednu sekundu pokud je použit hardwarový akcelerátor, či nikoliv. Z výše uvedeného výpočtu je známo, že jeden takt trvá 8ns. Za jednu sekundu procesor stihne „tiknout“ 125 000 000krát. Hledání počtu čar je prováděno tak dlouho, dokud není nalezen horní počet čar N_h a dolní počet čar N_d, pro který platí:

$$N_h = N_d + 1$$

A současně platí pro čas vykreslení horního počtu čar t_h a pro čas vykreslení dolního počtu čar t_d následující vztah:

$$t_d \leq 10^9 ns \leq t_h$$

Výsledek měření opět potvrdil, že urychlení výpočtu použitím hardwarového akcelerátoru nám umožňuje vykreslit řádově vyšší počet čar. Naměřené hodnoty jsou uvedeny v následující tabulce (Tabulka 4).

Tabulka 4: Počet vykreslených čar za jednu sekundu

Výpočet proveden	N_d	N_h	$t_d[ns]$	$t_h[ns]$
Hardwarově	121107	121108	999996936	1000005064
Softwarově	1950	1951	999681760	1000163464

3.3.3 Rychlost vymazání referované scény

Rychlost vymazání je u hardwarového akcelerátoru ovlivněna faktem, že je potřeba vynulovat paměť RAM, do které je scéna vykreslována, ale předtím se musí obsah celé paměti nahrát do paměti DDR, aby nedošlo ke ztrátě dat. A právě tento fakt, že je v podstatě potřeba „mazat“ dvě paměti na místo jedné, je nulování scény jediný případ, kdy je hardwarový akcelerátor pomalejší, než nulování prováděné softwarově. V následující tabulce opět vidíme naměřené hodnoty pro vymazání obrazu (Tabulka 5).

Tabulka 5: Čas potřebný pro vymazání obrazu

Výpočet proveden	Průměrný čas vymazání [ns]
Hardwarově	8280632
Softwarově	3164936

3.3.4 Rychlost výpočtu 3D grafických transformací

Vývojové prostředí Xilinx Platform Studio disponuje velkým množstvím nejrůznějších možností, jak optimalizovat procesor MicroBlaze za účelem zvýšení rychlosti a efektivnosti obvodu.

V testu se měří čas výpočtu geometrických transformací. V těchto transformacích se počítají hodnoty goniometrických funkcí sinus a cosinus třech úhlů. Dále zde dochází k násobení jednotkové matice s rotačními maticemi, translační maticí, maticí pro změnu měřítka a transformační maticí. Výsledná matice je pak aplikována na jednotlivé body krychle.

V tomto testu byly naměřeny celkové časy výše zmíněných výpočtů se třemi různými nastaveními procesoru. První test byl při optimálním nastavení, kdy bylo v prostředí Xilinx Platform Studio umožněno využití takzvané Floating Point Unit a v procesoru byly povoleny standardní cache (instrukční i data). Druhý test byl proveden bez podpory Floating Point Unit a třetí test byl proveden při zákazu obou cache i Floating Point Unit. Zprůměrované hodnoty všech vycházejí následovně:

Tabulka 6: Výsledky výpočtů 3D grafických transformací

	FPU i cache	**Pouze cache**	**Vše vypnuté**
Čas výpočtu [s]	0,012705	0,012874	0,116602

Z tabulky je patrné, že zákaz cache i FPU výrazně navýší čas výpočtů grafických transformací a to zhruba o 100ms. FPU je velmi výkonný obvod, který značně urychluje výpočty reálných čísel. Ze zprůměrovaných hodnot je vidět, že využití FPU urychlí výpočet grafických transformací přibližně o 0,1ms.

3.3.4.1 Instrukční cache

Použití instrukčních cache je čistě volitelné. Jejich využití však znamená veliké urychlení chodu procesoru, pokud je v běhu program, který se dostává mimo LMB (Local Memory Bus) adresní šíři.

3.3.4.2 Data cache

Další volitelnou položkou při nastavování procesoru je zapnutí/vypnutí data cache. Předpoklad pro využití data cache je, že rozsah cachované paměti nezasahuje do adres po celé LMB adresní šíři.

3.3.4.3 Floating Point Unit

Floating Point Unit, dále jen FPU, je pro použití ve spojení s MicroBlazem založena na základně standardu IEEE 754. FPU pracuje s takovým reálným formátem, s definicí nuly, se stavem Not-a-number a s definicí nekonečna, přesně tak, jak jsou stanoveny v tomto standardu. FPU podporuje všechny běžné matematické operace od sčítání, přes násobení až například po druhou odmocninu.

4 Závěr

V práci jsou popsané základní algoritmy a pojmy z oblasti 3D počítačové grafiky, včetně vysvětlení základních trojrozměrných geometrických transformací. Ke všem geometrickým transformacím jsou přidány odpovídající transformační matice.

Jedním z výsledků práce je program napsaný v jazyce C++ ve spojení s grafickou knihovnou SDL. Tento program sloužil k ozkoušení a následnému vybrání vhodných metod pro počítání trojrozměrných transformací. Program za pomoci knihovny SDL dále sloužil k otestování vybraných algoritmů pro vykreslování úseček, jakými jsou Bresenhamův algoritmus nebo algoritmus DDA.

Bresenhamův algoritmus, který se ukázal jako vhodnější pro další práci, byl následně převeden do návrhu hardware a funkčnost této komponenty byla řádně odsimulována. Celá komponenta byla přidána do již funkčního systému, kde plnila funkci hardwarového akcelerátoru pro výpočet bodů úsečky.

Funkčnost celého systému byla experimentálně ověřena vyrenderováním animace prostorové scény v podobě rotující krychle. Systém byl nakonec podroben několika testům, které ukázaly například efektivitu využití hardwarového akcelerátoru nebo také fakt, že vývojové prostředí od firmy Xilinx nabízí celou řadu možností pro optimalizaci navrhovaných systémů. Jen možnosti nastavení procesoru MicroBlaze jsou natolik rozsáhlé, že by o nich bylo možné napsat celou samostatnou práci.

Možnost rozšíření práce je určitě v optimalizaci vyhotoveného systému, tak aby bylo možno dosáhnout na vyšší frekvenci systému. Ta je nyní na hodnotě 125MHz. Při optimalizaci by se mohla frekvence zvýšit ke 300Mhz.

Další reálná cesta v rozšiřování této práce je možná v implementaci rozdílného typu rasterizace. Lze uvažovat o zvýšení hloubky barev, klidně až do True Colors (24-bitová hloubka barev), což by souviselo se změnou grafiky z wire-framové na grafiku, která by zahrnovala například mapování textur na jednotlivé objekty, anebo by to mohlo vést až k přechodu k některé z pokročilých metod vykreslování prostorové scény na počítači, jako je například Ray Tracking.

Použitá literatura

[1] AKENINE-MOLLER, Tomas; HAINES, Eric; HOFFMAN, Naty. *Real-Time Rendering*. Wellesley : A K Peters, 2008. 1027 s. ISBN 978-1-56881-424-7.

[2] HEARN, Donald; BAKER, M. Pauline. *Computer Graphics*. 2nd edition. [s.l.] : [s.n.], 1996. 662 s.

[3] HEROUT, Pavel. *Učebnice jazyka C - 1. díl*. České Budějovice : Kopp, 2006. 280 s. ISBN 80-7232-220-6

[4] IVANCHEV, Mihail; THÖRNQUIST, Hans. *DevMaster.net* [online]. 30. 11. 2003 [cit. 2011-05-15]. Software Rendering School, Part II: Projection. Dostupné z WWW: <http://www.devmaster.net/articles/software-rendering/part2.php>

[5] PINKER, Jiří; POUPA, Martin. *Číslicové systémy a jazyk VHDL*. Praha : Ben, 2006. 352 s.

[6] *Svět Hardware* [online]. 2011 [cit. 2011-05-17]. Dostupné z WWW: <http://www.svethardware.cz/>

[7] VOJÁČEK, Jakub. *Matematika pro každého* [online]. 24. 5. 2008 [cit. 2011-05-15]. Analytická geometrie - Úvod. Dostupné z WWW: <http://maths.cz/clanky/analyticka-geometrie-uvod.html>

[8] *Xilinx* [online]. 2011 [cit. 2011-05-15]. Dostupné z WWW: <http://www.xilinx.com/>

[9] ŽÁRA, Jiří, et al. *Moderní počítačová grafika : kompletní průvodce metodami 2D a 3D grafiky*. 2. přepracované vydání. Brno : Computer Press, a.s., 2010. 609 s. ISBN 80-251-0454-0.