VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

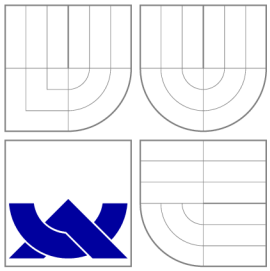
AKCELERACE GRAFICKÝCH OPERACÍ S VYUŽITÍM FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

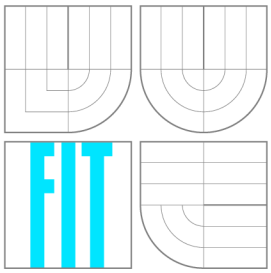
AUTOR PRÁCE
AUTHOR

Bc. LADISLAV ČAPKA

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE GRAFICKÝCH OPERACÍ S VYUŽITÍM FPGA

HARDWARE ACCELERATION OF GRAPHICS ALGORITHMS USING FPGA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. LADISLAV ČAPKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZDENĚK VAŠÍČEK
SUPERVISOR

BRNO 2007

Zadání diplomové práce

Řešitel: **Čapka Ladislav, Bc.**
Obor: Počítačové systémy a sítě
Téma: **Akcelerace grafických operací s využitím FPGA**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte literaturu zabývající se grafickými operacemi a možnostmi jejich akcelerace v hardware.
2. Seznamte se s výukovou platformou FITkit obsahující FPGA.
3. Zpracujte studii na výše uvedená témata.
4. Navrhněte a vhodně implementujte řadič umožňující akceleraci zvolené operace s využitím platformy FITkit, apod.
5. Funkčnost demonstrujte na vzorové aplikaci využívající Vámi navržený řadič.
6. Porovnejte získané výsledky s čistě softwarovou implementací.

Literatura:

- http://www.xilinx.com
- http://www.fit.vutbr.cz/kit
- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Splnění prvních tří bodů zadání

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese
`http://www.fit.vutbr.cz/info/szz`

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Vašíček Zdeněk, Ing.**, UPSY FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technologií Vysokého učení technického v Brně.

Výňatek z licenční smlouvy:

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zvěřejněním díla v databázi přístupné v mezinárodní síti:
   - ■ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací).
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Abstrakt

Tato práce se zabývá rozborem grafického řetězce, pomocí kterého lze vykreslit požadovaný obraz. Dokument je zaměřený na vykreslovací algoritmy, které jsou využívány v rasterizačním bloku. Hlavním cílem této práce je popsat vybrané vykreslovací algoritmy, které jsou vhodné pro implementaci v hardware. Cílovým hardware, na kterém by měly být vykreslovací algoritmy implementovány a testovány, jsou programovatelná hradlová pole FPGA. Jako možná cílová platforma byla zvolena platforma FITkit.

Klíčová slova

grafický hardware, FPGA, rasterizace, rasterizační algoritmy, interpolace, NURBS, Bézier

Abstract

This term project is aimed on analysis of graphic pipeline which can rasterize required picture. Document is specialized to drawing algorithms that are used in rasterization block. Major aim of this project is describing of rasterization algorithms that can be implemented on hardware. Type of aimed hardware is field-programmable gate array FPGA.

Keywords

graphic hardware, FPGA, rasterization, rasterization algorithms, base objects, interpolation, NURBS, Bézier

Citace

Ladislav ČAPKA: Akcelerace grafických operací s využitím FPGA, diplomová práce, Brno, FIT VUT v Brně, 2007

Akcelerace grafických operací s využitím FPGA

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Zdeňka Vašíčka.

.....
Ladislav ČAPKA
22. května 2007

Poděkování

Chtěl bych poděkovat svému školiteli Zdeňkovi Vašíčkovi za jeho cenné rady, svým drahým rodičům a v neposlední řadě své přítelkyni za velkou trpělivost.

Obsah

Kapitola 1

Úvod

Rasterizace obrazu je důležitá ve spoustě odvětví, kde je nutné vizualizovat nejrůznější data. S postupem doby se stále zvyšují nároky na kvalitu a realističnost výsledného obrazu. Kvalita je ovlivněna složitostí vykreslování a dostupným výkonem, který lze na vykreslování využít. Běžnou metodou je zvyšování taktovacích frekvencí vykreslovacích řetězců založených na grafických procesorech. Tento dokument se zabývá metodami a postupy, které je možné navrhnout a využít v grafickém hardware založeném na hradlových polích (obvodech FPGA). Důvodem je myšlenka, že FPGA obvody mohou dosahovat pro určité aplikace a vhodném návrhu architektury vysokého výkonu.

Celý dokument je rozdělen do několika kapitol. V kapitole 2 je čtenář seznámen se základní obecnou koncepcí rasterizačního řetězce. Kapitola seznamuje čtenáře s jednotlivými bloky tohoto řetězce a ukazuje, které z nich jsou nezbytné a které slouží jen ke zvýšení kvality a možností rasterizace. Blok rasterizace, jenž je jádrem rasterizačního řetězce, vždy využívá vhodných algoritmů. Jelikož je tento blok považován za základní, popisuje kapitola 3 podrobněji několik různých přístupů a metod, které mohou tento blok tvořit. Doplňkové algoritmy k těmto metodám popisuje kapitola 4. Uvedené kapitoly popisují teoreticky řešení problému. Od kapitoly 5 se dokument zabývá popisem praktického návrhu a realizace. První kapitolou v této části je kapitola 5. V této kapitole se čtenář seznámen s platformou cílovou platformou, pro které je výsledná architektura navrhována. Čtenář se dozvídá o koncepci a možnostech platformy FITkit. Samostatnou podkapitolou je popis struktury obvodů FPGA. Kapitola 6 přesně popsuje strukturu návrhu jednotlivých bloků architektury. V poslední kapitole 7 jsou vyhodnoceny experimentální výsledky. Hodnocení dosažených výsledků probíhá na více úrovních - porovnání se softwarovou implementací, popis vybraných hodnot výsledků syntézy, atp. Na závěr jsou shodnoceny výsledky, uvedeny možnosti rozšíření a další cíle projektu.

Kapitola 2

Grafický řetězec

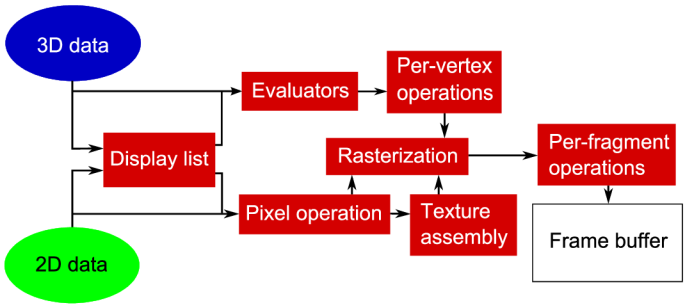
Zobrazovací grafický řetězec zajišťuje grafickou reprezentaci v úlohách, které vyžadují vizualizaci, ať již pro zábavu, lepší orientaci či jednodušší pochopení. Zobrazovací metody lze rozdělit do několika kategorií dle kvality či vzhledu. Nejčastěji se jedná o reprezentaci dat pomocí zobrazení z n-rozměrného prostoru, kde $n \in \{2, 3\}$, do dvourozměrného prostoru. Pro dvou-rozměrnou grafickou reprezentaci se používá zkratka 2D, pro troj-rozměrnou 3D. U obou kategorií je tedy výstupem zobrazení libovolných bodů 2D matice, uvnitř které má každý bod definovanou určitou barvu. V případě 2-rozměrného prostoru je však vstupem do zobrazovacího řetězce 2-rozměrný objekt. Tím mohou být čáry, body, trojúhelníky, křivky apod. Vlastnosti a popis objektů v dvojrozměrném prostoru je jednodušší. Ve většině případů se jedná pouze o jeden atribut – barvu. Řešit stínování či osvětlování z pohledu 2D prostoru nemá smysl. Proto tyto vlastnosti nejsou uvažovány. Velmi důležitou vlastností rasterizačního bloku dvojrozměrných objektů je pořadí vykreslení objektů. Je nutné si uvědomit, že se objekty mohou překrývat.

S trojrozměrným prostorem se lze setkat v oblastech, které vyžadují prostorově vyhlížející výstup. Popis 3D scény je komplikovanější, protože zpravidla vyžaduje přesnější pohled na trojrozměrný prostor, který by měl být pro uživatele co nejrealističtější. Podobně jako ve 2D prostoru mohou být v 3D prostoru objekty modelovány nejrůznějšími technikami jako například kompozicí několika prostorových jednodušších objektů apod. Typicky používaným modelem pro popis 3D scény je model založený na kompozici jednoduchých dvojrozměrných útvarů, kterými jsou typicky trojúhelníky. Výhodou použití trojúhelníků je fakt, že jejich kompozicí lze vytvořit téměř libovolně složitý trojrozměrný objekt. Na obrázku 2.1 je možné vidět, že cesta 3D dat, které charakterizují konkrétní scénu, končí v bloku rasterizace. V tomto bloku probíhá rasterizace převážně 2-rozměrných objektů tvořících výslednou scénu. Stejně jako u 2D prostoru musí mít každý objekt definované atributy, které jsou potřebné pro rasterizaci výsledné scény. Atributem, který je nezbytný pro rasterizaci v 2D i 3D prostoru je barva.

Pro realistické zobrazení trojrozměrné scény je zapotřebí uvažovat alespoň nejjednodušší model osvětlení. Pomocí osvětlení je člověk schopen lépe rozlišit tvar ozářeného objektu a napomáhá mu rozpoznat jeho prostorové souřadnice (například pomocí stínů). Aby bylo možné správně osvětlit objekt 3D scény, který je tvořen kompozicí 2D trojúhelníků, musí být pro každý vrchol definovány normálové vektory a křivka odrazivosti. Křivka odrazivosti

je zpravidla zjednodušena na barvu, kterou ozářený objekt odrazí zpět do prostoru. Normálové vektory jsou vektory kolmé na výslednou plochu. Za jejich pomoci se vypočítává lesk a případně odraz. Další vlastností realistického zobrazení je průhlednost. Bez průhlednosti nelze vykreslit materiály jakými mohou být například sklo či plasty. V neposlední řadě se jedná o pokrytí objektu mapou, které se říká textura. Texturovacím obrázkem se pokrývá objekt, aby jeho výsledná barva byla rozšířena na vizuálně složitější objekt.

Z výše uvedeného popisu plyne, že zobrazení 3D prostoru vyžaduje složitější renderovací řetězec. V grafických kartách domácích počítačů existuje renderovací řetězec (viz obrázek 2.1), který je složen z několika základních částí. Renderovací řetězec má dva vstupní bloky. Do prvního bloku vstupují trojúhelníkové objekty (na obrázku označeny jako **3D data**), které reprezentují prostorová data jednotlivými vrcholy trojúhelníků a jejich vlastnostmi. Druhým vstupem, označeným **2D data**, jsou rastrová data. Tato rastrová data zahrnují bitmapy textur, plošné operace apod. Místem, kde lze data uchovat pro vícenásobné použití, je jednotka nazvaná **display list**. Display list slouží k redukci množství přenášených dat vstupní cestou, což má za následek zrychlení na cestě vstupních 2D i 3D dat. Blok, nazvaný **pixel operation**, slouží k úpravě grafický bodů na straně 2D dat. Pokud je rasterizovaný objekt texturován více texturami složenými přes sebe, využívá se pro předzpracování výsledné textury blok nazvaný **texture assembly**. Na datové cestě prostorových dat je první jednotkou blok nazvaný **evaluators**. Tento blok slouží k výpočtům ploch - například spline povrchů. Za tímto blokem jsou data reprezentována vždy pomocí vrcholů. K modifikacím vrcholů slouží blok nazvaný **per-vertex operations**. Hlavním blokem zobrazovacího řetězce je rasterizační jednotka. Prvním vstupem této jednotky jsou vhodné vrcholy a informace, které se jich týkají. Další informace, které jsou k dispozici pro rasterizační jednotku, jsou rastrová data. Rasterizační jednotka provádí rasterizaci sledu trojúhelníků reprezentovaných vektorovými daty v kombinaci s rastrovými daty. Výstupem rasterizační jednotky je výsledný obraz, který je tvořen mřížkou bodů - fragmentů. Úpravu těchto fragmentů provádí jednotka **per-fragment operations**. Výstup z této jednoty je již zobrazen vhodným zobrazovacím zařízením.



Obrázek 2.1: Renderovací řetězec

Uvedený grafický řetězec umožňuje poměrně kvalitní renderování. Ve většině případů je realizován grafickým procesorem, který má poměrně vysokou spotřebu. Rychlost, kterou vykazuje, není zpravidla dostatečná pro realistické animování složitých scén. Z těchto důvodů je vhodné vytvořit hardware, který by byl schopen v určitých případech zastoupit nyní nejběžnější renderovací řetězec tvořený grafickým procesorem tak, aby jeho spotřeba klesla

a výkon byl co možná nejvyšší. Programovatelná hradlová pole FPGA (viz kapitola 5.1) umožňují složité výpočty efektivně paralelizovat a jejich spotřeba může být oproti grafickým procesorům nižší. V případě, že se navrhovaný hardware optimalizuje pro specifickou aplikaci, lze ve většině případů dosáhnout vyššího výkonu, než v případě využití grafického procesoru. Pro rasterizaci 2D obrazu nebo renderování složitějších 3D scén je nutné využít či navrhnout rasterizační algoritmy na nejnižší úrovni. Mělo by se jednat o algoritmy, které lze paralelizovat a efektivně implementovat v FPGA.

Kapitola 3

Rasterizace

Pro vytvoření grafické reprezentace je nejdůležitější částí grafického zobrazovacího řetězce blok, který se nazývá rasterizační (viz obrázek 2.1). Cílem této kapitoly je seznámit čtenáře s touto jednotkou a možnostmi realizace jejích operací. Úkolem této jednotky je převést číselně či matematicky definovaný tvar na grafickou reprezentaci. Výsledkem je množina bodů doplněných do výsledné matice $m \times n$. Velikost výsledné matice závisí na charakteru zobrazení, tedy na množství informace, která je dostatečná pro grafickou reprezentaci dané úlohy.

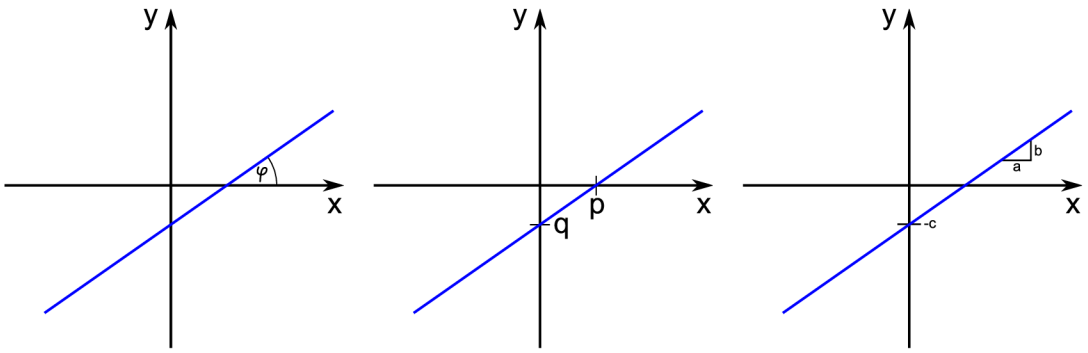
Matice výsledných bodů je z pohledu hardware reprezentována blokem paměti. Jeden grafický bod může reprezentovat jeden byte paměti, avšak může to být více i méně, což závisí na počtu barev, které je nutné zobrazit. Obecně může být pro rasterizace vstupem, který je nutné graficky reprezentovat, libovolný útvar. Pro charakterizaci výsledného tvaru lze užít několik metod. První metodou je matematický popis výsledného tvaru. Matematický popis libovolného tvaru je obtížné vyjádřit. Složitý je i výpočet při rasterizaci. Silnou výhodou FPGA obvodů je možnost velké paralelizace rasterizačního algoritmu. Druhou metodou je popis výsledného útvaru pomocí jednodušších objektů, které se vykreslí v určitém pořadí a vytvoří tak výslednou grafickou reprezentaci. Grafická reprezentace složená z jednoduchých útvarů může ztratit realistický vzhled, ale při dostatečné parametrizaci objektů a kvalitním pokrytí požadovaného tvaru lze dosáhnout velmi kvalitních výsledků. Pro reprezentaci libovolného tvaru lze velmi dobře využít základní prvky, kterými jsou bod, úsečka, kružnice a trojúhelník, pomocí kterých lze charakterizovat téměř všechny graficky zobrazitelné útvary. Objekty, které mají tvar se spoustou zakřivení, je ovšem problematické těmito základními tvary charakterizovat. Řešením mohou být nejrůznější křivky a křivkami tvořené plochy. Obecnou křivkou, jejíž algoritmus lze v hardware realizovat, může být křivka non-uniform rational B-spline (dále jen NURBS), případně Bézierova křivky. V obou případech se jedná o křivky, jejichž definice je jednoduchá, neboť jsou tvořeny několika málo body v prostoru. Poměrně problematickou částí je ovšem definice těchto bodů. Podrobněji se těmito křivkami zabývají podkapitoly 3.4 a 3.5.

Pro grafický hardware je zapotřebí hledat takové algoritmy, které lze v hardware efektivně implementovat. Požadavkem na takové algoritmy je, aby obsahovaly co možná nejmenší počet složitých výpočetních operací, jakými mohou být například časově náročné dělení, či práce s desetinnými čísly.

Moderní metody grafického hardware založeného na programovatelných hradlových polích FPGA bývají často postaveny na velmi jednoduchých metodách, které lze efektivně paralelizovat a zvýšit tak jejich malý výkon na úroveň výpočetně složitých avšak pomalejších algoritmů. Následující části dokumentu se zabývají metodami, které lze realizovat v hardware a vytvořit pomocí nich většinu požadovaných objektů. Rasterizace bodu není v dokumentu uvažována, jelikož ji lze realizovat jedním zápisem do paměti výstupní obrazové matice. Prvním a nejjednodušším úkolem je tedy rasterizace úsečky.

3.1 Rasterizace úseček

Tato kapitola se zabývá rasterizací úsečky, která tvoří (mimo bod) nejjednodušší útvar, který lze i poměrně snadno rasterizovat. Úsečka je v prostoru reprezentována dvěma body a charakterizuje nejkratší spojnici mezi nimi. Z geometrického hlediska lze úsečku reprezentovat pomocí rovnice přímky. V 2D prostoru je možné využít několik vzájemně ekvivalentních možností popisu přímky.



(a) Směrnice (b) Úsek (c) Obecná rovnice přímky

Obrázek 3.1: Způsoby popisu přímky

Prvním typem popisu přímky je směrnicová rovnice (viz obrázek 3.1(a)). Směrnicová rovnice přímky má tvar $y = k \cdot x + q$, kde $k = tg\varphi$. Směrnicový tvar využívá definice směru přímky pomocí úhlu φ, který svírá osa x s úsečkou. Posunutí přímky je definováno konstantou q.

Druhým možným typem popisu přímky je tzv. úsekový tvar $\frac{x}{p} \cdot \frac{y}{q} = 1$ (viz obrázek 6.7(b)). Podmínkou je, že $p \neq 0 \wedge q \neq 0$. Narozdíl předchozí varianty se jedná o popis pomocí dvou úseků p a q, které charakterizují směr a posunutí přímky. Nevýhodou je, že touto rovnicí nelze popsat přímky rovnoběžné s hlavními osami, jelikož by v takových případech bylo $p = 0$ nebo $q = 0$, což není z hlediska prostoru $\mathbf{R}^2$ přípustné.

Nejvhodnější tvar pro zpracování v hardware je parametrický popis přímky, který je tvořen dvěma rovnicemi (3.1).

$$\begin{aligned} x &= x_1 + (x_2 - x_1) \cdot t \\ y &= y_1 + (y_2 - y_1) \cdot t \\ \text{kde } t &\in [0, 1]. \end{aligned} \tag{3.1}$$

Tento způsob zápisu vyžaduje dva parametry – počáteční bod $[x_1, y_1]$ a koncový bod $[x_2, y_2]$. Modifikací parametru t v rozsahu [0,1] se lze pohybovat po všech bodech, které úsečku tvoří. Složitost hardwarové akcelerace spočívá v možnostech hardware, neboť uvedený způsob vyžaduje práci s desetinnými čísly, což může v některých případech činit komplikace při implementaci. Na druhou stranu výhodou této metody je fakt, že nevyžaduje složité komparace hodnot a vykreslení celé úsečky probíhá změnou jednoho parametru (t). Metoda je tedy jednoduchá.

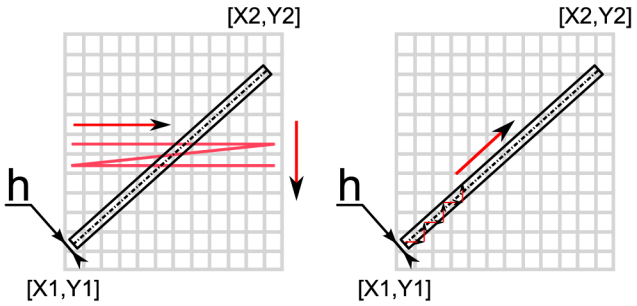
V případech, kdy parametrický tvar není vhodný, může být úsečka efektivně popsána obecnou rovnicí přímky. Obecná rovnice přímky má tvar $ax + by + c = 0$, kde

- a,b,c jsou konstanty, $a \neq 0 \vee b \neq 0$,
- x,y jsou body přímky.

Jelikož rasterizace je defacto diskretizací určité spojité funkce, vyžaduje přístup vykreslování pomocí rovnice obecné přímky jistou toleranci. Pokud by rovnice zůstala ve tvaru $ax + by + c = 0$, mohou být body $[x, y]$ testovány na vlastnost, zda přímce náleží, mírně od hodnoty 0 odchýleny. Proto je vhodné do obecné rovnice zanést toleranční faktor – například jednoduchou nerovnicí (3.2).

$$ax + bx + c \leq h \tag{3.2}$$

Hodnota h je tolerance (tloušťka) čáry. U metody využívající popis úsečky pomocí koeficientů pro obecnou rovnici přímky je nutné projít celý prostor ve kterém se může úsečka nacházet a nalézt body, které splňují definovanou nerovnici, jak je zobrazeno na obrázku 3.2(a). Pro urychlení metody by bylo vhodné najít nejdříve bod, který na dané přímce leží, a postupovat postupně po řádcích přes celou úsečku (viz obrázek 3.2(b)).



(a) Celý prostor (b) Pouze úsečka

Obrázek 3.2: Rasterizace přímky

Pro rasterizaci úsečky lze využít některý z algoritmů založených na uvedených rovnicích přímky. Prvním a nejjednodušším algoritmem je postupné přičítání přírůstku v ose y (algoritmus DDA). Počátečními hodnotami požadovanými pro tento algoritmus jsou výchozí bod, přírůstek $\Delta = \frac{\Delta y}{\Delta x}$ definující vlastně sklon a délka přímky měřená rovnoběžně s osou x. Vykreslování přímky probíhá přičítáním hodnoty 1 k hodně x, která je na počátku

definována jako x_1 a zároveň hodnoty Δ k hodnotě y, která je na počátku definována jako hodnota y_1. Nevýhodou této metody je, že výsledná přímka dosahuje poměrně nízké kvality a podmínkou nutnou pro správné vykreslení je $\Delta \leq 1$. Tedy $\Delta y \leq \Delta x$.

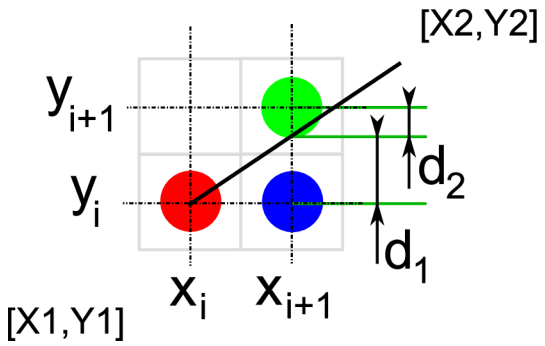
Druhým používaným algoritmem pro rasterizaci úsečky je Bresenhamův algoritmus. Bresenhamův algoritmus vychází ze směrnicového popisu přímky. Důležité rovnice pro tento algoritmus jsou (3.3).

$$\begin{aligned} y &= k \cdot (x_i + 1) + q \\ k &= \frac{\Delta y}{\Delta x} \\ \Delta x &= x_2 - x_1 \\ \Delta y &= y_2 - y_1 \\ \Delta &= \Delta y - \Delta x \end{aligned} \tag{3.3}$$

Průběh algoritmu je dán hodnotami Δ a hodnotou p_i. Hodnota p_i má počáteční hodnotu $p_1 = 2 \cdot \Delta y - \Delta x$. Následující hodnota p_{i+1} vychází z aktuálních hodnot Δ a p_i. Pro $\Delta > 0, p_i > 0$ se využívá výpočet (3.4), pro $\Delta \leq 0, p_i \leq 0$ naopak (3.5).

$$\begin{aligned} y_i - y_{i+1} &= -1 \\ p_{i+1} &= p_i - 2 \cdot \Delta x + 2 \cdot \Delta y \end{aligned} \tag{3.4}$$

$$\begin{aligned} y_i - y_{i+1} &= 0 \\ p_{i+1} &= p_i + 2 \cdot \Delta y \end{aligned} \tag{3.5}$$



Obrázek 3.3: Bresenhamův algoritmus

Výhodou Bresenhamova algoritmu rasterizace přímky je práce pouze s celými čísly. Vzhledem k této vlastnosti se jedná o algoritmus vhodný pro implementaci v hardware. I přes toto zjednodušení vykazuje Bresenhamův algoritmus oproti jiným algoritmům velmi vysokou efektivitu a jeho rychlost může být mnohonásobně vyšší než u algoritmu DDA.

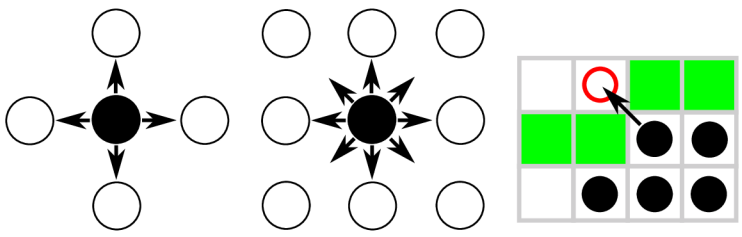
3.2 Rasterizace trojúhelníku

Jak bylo uvedeno na začátku kapitoly 3, lze libovolný trojrozměrný objekt reprezentovat kompozicí jednoduše rasterizovatelných 2D objektů, kterými bývají ve většině případů trojúhelníky. Právě z toho důvodu je rasterizace trojúhelníku velmi důležitým prvkem nejen 2D ale i 3D grafického zobrazování. Z pohledu 3D prostoru se jedná o 2D plochu v prostoru definovanou nejmenším možným počtem řídících bodů, tedy třemi. Jediný rozdíl mezi trojúhelníkem ve 2D prostoru a ve 3D prostoru je ten, že ve 3D prostoru má každý řídící bod definovánu polohu vzhledem ke třem osám a tudíž je nutné nejdříve polohy řídících bodů přepočítat do dvojrozměrné mřížky výsledného rasterizovaného obrazu. Při tomto výpočtu lze uvažovat různé efekty jako například vliv perspektivy apod.

U vykreslování trojúhelníků pro 3D scény je také nutné vytvořit seznam trojúhelníků a na základě tohoto seznamu je vykreslovat. Pořadí v seznamu určuje především vzdálenost od pozorovatele – tedy vykresluje se od nejvzdálenějšího trojúhelníku po nejbližší a je vhodné implementovat i případnou eliminace trojúhelníků, které v renderované scéně nejsou vidět. O efektivním řazení vhodném pro implementaci uvnitř FPGA se lze např. dočíst v [8]. Pro rasterizaci trojúhelníku bylo vytvořeno několik odlišných metod. Úkolem rasterizace je vlastně nalezení bodů trojúhelníku. Každý trojúhelník si lze představit jako dvě doplňující se části. Jednou z nich je obrys trojúhelníku tvořený třemi úsečkami a druhou částí je jeho obsah. Při rasterizaci objektů složených z obrysu a výplně je nutné implementovat vhodný vyplňovací algoritmus, který lze s výhodou využít i u jiných metod. Algoritmů na vyplňování oblastí existuje celá řada. Vhodným algoritmem pro implementaci v FPGA je semínkové vyplňování popsané v následující podkapitole.

3.2.1 Semínkové vyplňování

Nejjednodušším typem vyplňování objektů pevně zvolenou barvou je metoda semínkového vyplňování. Je to softwarově nejméně efektivní vyplňovací algoritmus, který je ovšem z důvodu veliké jednoduchosti dobře implementovatelný v FPGA a také velmi dobře paralelizovatelný. Z těchto důvodů lze v FPGA metodu semínkového vyplňování efektivně využít pro vyplňování oblastí.



(a) čtyř-směrné (b) osmi-směrné (c) Chyba u osmi-směrného

Obrázek 3.4: Semínkové vyplňování

Metoda semínkového vyplňování má dva typy – čtyř-směrová (viz obrázek 3.4(a)) a osmi-směrová (viz obrázek 3.4(b)). Volba typu semínkového vyplňování závisí na typu ohraničení.

Osmi-směrové vyplňování je rychlejší, v některých případech ovšem může způsobit chybu detekce ohraničení a úhlopříčnými směry vygenerovat výplň i mimo zamýšlenou oblast (viz obrázek 3.4(c)). Čtyř-směrové semínkové vyplňování je bezpečnější. Vzhledem k možnostem FPGA, tj. možné paralelizaci úloh, by čtyř-směrové semínkové vyplňování mohlo být dostatečně efektivní.

3.2.2 Metoda hranové detekce

Při rasterizaci trojúhelníku lze vyjít z obecné rovnice přímky. Rozdílem od rasterizace přímky je výpočet hodnoty d ze vzorce $d = Ax + By + C$. U rasterizace trojúhelníku metodou hranové detekce je důležité pouze znaménko hodnoty d.

Trojúhelník si lze představit jako tři úsečky. Pokud jsou souběžně s těmito úsečkami položeny tři přímky, lze o každém bodu a každé přímce říci, zda bod leží nad nebo pod přímkou – tedy v jedné či druhé polorovině, na něž daná přímka celý prostor R^2 rozděluje. Podstatou metody je detekce, zda je daný bod součástí trojúhelníku či nikoliv. Detekce je založena právě na vztahu mezi bodem a jednotlivými polorovinami, které přímky vytvářejí. Nerozlišuje se, zda se jedná o bod patřící do ohraničení trojúhelníku, tedy jeho hrana, nebo do výplně. Pokud by neměl být rasterizovaný trojúhelník ohraničený, bylo by navíc nutné počítat nerovnici $Ax + By + C < h$, kde h by byla tloušťka čáry, a to pro všechny tři hrany trojúhelníku. Pokud by byla splněna podmínka alespoň pro jednu hranu, jednalo by se o hranu trojúhelníku – tedy ohraničení.

Pro výpočet lze využít uvedený vzorec (3.6) pro všechny tři hrany trojúhelníku. Koeficienty A,B a C, které jsou v rovnici použity, definují rovnice (3.7). Tyto koeficienty se v době výpočtu nemění a jsou tedy pro daný trojúhelník konstantní. Této vlastnosti lze s výhodou využít a vypočítat kontanty pouze jednou a to během počáteční fáze rasterizace.

$$Ax + By + C > 0 \tag{3.6}$$

$$\begin{aligned} A &= x_0 \cdot x_1 \\ B &= y_0 - y_1 \\ C &= x_1 - y_0 \end{aligned} \tag{3.7}$$

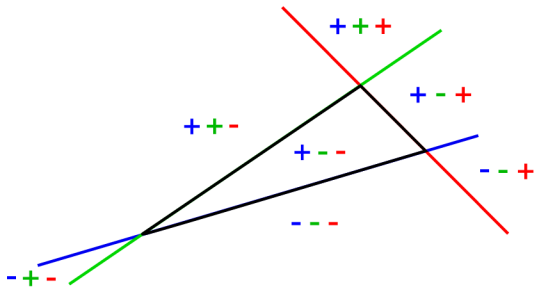
Ve výše uvedených vzorcích (3.6) a (3.7) byla uvažována polorovina určena body $[x_0, y_0]$ a $[x_1, y_1]$. Analogicky lze vytvořit rovnice zbývajících dvou polorovin. Příklad rozložení polorovin vytvořených přímkami souběžnými se stranami trojúhelníku ukazuje obrázek 3.5. Na obrázku je znázorněno ohodnocení polorovin symboly + a − pro všechny tři přímky souběžné s hranami trojúhelníku, které prostor R^2 rozdělují.

$$Q = A \ \cdot x_2 + B \cdot y_2 + C \tag{3.8}$$

Z této úvahy lze vyvodit, že každý bod patřící do trojúhelníku má určitou kombinaci hodnot ve třech uvedených polorovinách. Problémem této metody je určení, pro kterou

hranu trojúhelníku vzít v úvahu nerovnici $Ax + By + C > 0$ a pro kterou $Ax + By + C < 0$. Vhodným cílem je pro všechny hrany získat takové koeficienty, pro které lze pro všechny testované poloroviny využít vždy jednu určitou nerovnici. Tímto by se výpočet jednotlivých bodů, testovaných na vztah s trojúhelníkem, urychlil.

Úvahou byl algoritmus obohacen o úpravu koeficientů tak, aby bod v trojúhelníku mohl být vždy detekován rovnicí $Ax + By + C > 0$. Nejdříve je nutné vypočítat hodnoty A, B a C, které jsou dále upraveny jednoduchou podmínkou. Výpočet se provede postupně pro koeficienty všech přímek a dosazením bodu $[x, y]$, který definuje třetí řídící bod trojúhelníku. Třetím bodem se myslí bod, který nebyl využit pro výpočet koeficientů A, B a C. Výsledkem rovnice, která navazuje na výše uvedené příklady s dosazenými body $[x_0, y_0]$ a $[x_1, y_1]$, je hodnota Q v následujícím vzorci s třetím řídícím bodem $[x_2, y_2]$.



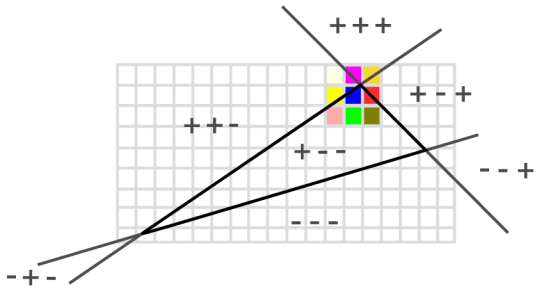
Obrázek 3.5: Příklad rozložení polorovin

Pokud je výsledek Q výše uvedeného vzorce (3.8) záporný, je nutné všechny koeficienty A, B i C pro danou přímku nastavit na hodnoty s opačným znaménkem. Analogickým postupem lze upravit hodnoty A, B a C u zbývajících přímek. Algoritmus je tedy možné provádět vždy s nerovnicí $Q > 0$. Bod trojúhelníku je vždy určen kladnými hodnotami Q. Platí tedy $Q > 0$ (případně $Q \geq 0$). Rasterizaci touto metodou by mělo být možné pomocí FPGA realizovat s využitím jen malé části plochy FPGA obvodu.

3.2.3 Paralelní verze hranové detekce

Metoda detekce hrany trojúhelníku je založena na ohodnocení každého bodu tak, že lze říci, zda daný bod do trojúhelníku patří či nikoliv. Každý bod je vyhodnocován nezávisle na okolních bodech. Této vlastnosti lze efektivně využít pro paralelizaci v FPGA. Pro rychlejší vyhodnocení a vykreslení je vhodné detekci, zda je bod součástí trojúhelníku, implementovat v FPGA vícekrát a vyhodnocovat tak například 8 bodů současně (viz obrázek 3.6).

Z pohledu hardware je nutné rasterizaci rozložit na více paralelních bloků, kde každý blok rasterizuje určitou jen část plochy výsledného obrazu. Plocha FPGA obvodu, kterou zaplní metoda hranové detekce, je u paralelní metody nejméně n-násobkem, kde n je počet bloků. Omezením této metody je rychlost zpracování výsledků, které jsou paralelně touto metodou získávány. Každý bod vyhodnocovaného trojúhelníku, jež je jeho součástí, musí být rasterizován, či dále zpracován.



Obrázek 3.6: Paralelizace pomocí sub-prostorů

3.3 Rasterizace kruhu

Z objektů uvedených v předchozích částech kapitoly není vhodné tvořit kruhové útvary. Kruhové útvary lze rozdělit na dva typy. Jedním z nich jsou kruhy a druhým kružnice. Rasterizace kružnice spočívá ve vykreslení ohraničení kruhového útvaru, tedy bez vyplnění vnitřní oblasti. Kruh lze poté definovat jako vyplněnou kružnici. Pro rasterizaci kružnice i kruhu lze využít několik algoritmů. Kružnice i kruh mají definovaný střed v bodě $[x, y]$ a poloměr r. Jednodušším tvarem je kružnice, a proto se jí budeme zabývat jako první. Vzorec definující kružnici je (3.9).

$$(x - x_0)^2 + (y - y_0) = r^2 \tag{3.9}$$

Nejjednodušší možností rasterizace kružnice může být jednoduchý algoritmus postavený na přičítání úhlu a s využitím funkcí *sin* a *cos*. Algoritmus je tvořen cyklem přes hodnotu α v intervalu $\alpha \in < 0, \pi >$. Hodnota α je doplněna do rovnic (3.10).

$$\begin{aligned} x &= r \cdot cos(\alpha) \\ y &= r \cdot sin(\alpha) \end{aligned} \tag{3.10}$$

Výsledky rovnic vždy udávají bod, který je součástí kružnice. Lze tedy postupně projít n bodů kružnice a body rasterizovat. Na volbě počtu bodů závisí kvalita vykreslení kružnice. Pokud by byl počet zvolených bodů n malý, nebyla by výsledná kružnice (rasterizována jako čarové úseky) spojitá. Naopak pokud by byl počet bodů n příliš velký, probíhal by mnohokrát zápis do paměti na stejné místo a algoritmus by nebyl efektivní. Výhodným koeficientem je $\Delta\alpha = 1/r$. Jistým zkvalitněním nedostatku spojitosti u uvedeného algoritmu je možnost vypočítat vždy dva po sobě jdoucí body a spojit je čárou. Kvalita vykreslení kružnice při menším počtu bodů klesá, ale problém s nespojitostí výsledné čáry kružnice je eliminován.

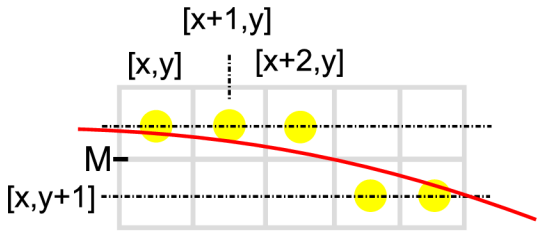
Jiný efektivnější a pro hardware vhodnější algoritmus je, podobně jako u rasterizace přímky, Bresenhamův algoritmus rasterizace kružnice. Podobně jako u Bresenhamova algoritmu pro rasterizaci přímky se jedná o celočíselný algoritmus. Výpočet se provádí jen na osmině kruhu a každý bod se kreslí na zbývajících sedmi symetrických místech. Je tedy výpočetně rychlejší.

Proměnné u tohoto algoritmu nabývají na počátku hodnot: $D_0 = 1.25 - r$ (případně $D_0 = 1 - r$). Algoritmus rozhoduje o vykreslení bodu dle znaménka výsledku D_n, který určuje, zda bod leží pod středem či nad středem M (viz obrázek 3.7).

Pokud platí, že $D_n < 0$, je výpočet veden podle vzorce (3.11). Pokud tomu tak není a platí tedy $D_n > 0$ jsou použity vzorce (3.12).

$$\begin{aligned} D_{n+1} &= D_n + 2x + 3 \\ y_{n+1} &= y_n \\ x_{n+1} &= x_n + 1 \end{aligned} \tag{3.11}$$

$$\begin{aligned} D_{n+1} &= D_n + 2x - 2y + 3 \\ y_{n+1} &= y_n - 1 \\ x_{n+1} &= x_n + 1 \end{aligned} \tag{3.12}$$



Obrázek 3.7: Bresenhamův algoritmus rasterizace kružnice

Výsledky Bresenhamova algoritmu dosahují poměrně vysoké kvality při nízké režii. Algoritmus lze implementovat v FPGA. Výhodou je jeho celočíselná forma.

Druhým možným rasterizovaným útvarem může být kruh. Jak již bylo zmíněno výše, kruh lze definovat jako vyplněnou kružnici. Z této úvahy lze vyvodit postup, jak rasterizovat kruh. V podkapitole 3.2.1 je vysvětlen jednoduchý algoritmus vyplňování vhodný pro implementaci v FPGA. Při rasterizace kruhu lze tedy postupovat složením dvou metod: a to rasterizace kružnice, která vyžaduje definici stejných hodnot jako rasterizace kruhu, a aplikací semínkového vyplňování například od středu rasterizované kružnice. Je zřejmé, že výsledkem kombinace těchto dvou metod bude kruh. Jinou možností je přímo testovat, zdali bod leží uvnitř kružnice.

3.4 NURBS křivky

Útvary, které byly popsány v předchozích kapitolách, pokrývají svými možnostmi širokou škálu tvarů, které je možné složit. Zaoblené tvary lze vytvořit kompozicí částí kružnic. Ty ovšem nejsou schopny jednoduše a dostatečně pokrýt širokou škálu tvarů, které mají mnoho zaoblených míst, která na sebe mohou i navazovat. Z těchto důvodů je možné využít rasterizace libovolné křivky. Složitost vyjádření určité křivky ovšem stoupá s množstvím

zaoblených míst a složitostí tvaru geometrickou řadou. Obecné vyjádření křivky tedy není vhodné uvažovat jako rychlou rasterizační metodu. Tyto zaoblené tvary lze rasterizovat a modelovat pomocí určitých B-Spline křivek. B-Spline křivky jsou založeny na obecné rovnici a $p+1$ kontrolních bodech $(P_0..P_p)$. B-Spline křivka je tvarem blízká sledu polohy bodů v prostoru. Tvoří ji $p-(n-1)$ segmentů $\{Q_n, Q_{n+1}, ..., Q_p\}$, kde $p \geq n$. Dále je možné pro každý segment definovat parametr t s hodnotami v intervalu $t \in [0,1]$. Pro každý segment Q_i je hodnota t v intervalu $[t_i, t_{i+1}]$, kde $n \leq i \ \leq p$. Každý segment Q_i je ovlivněn pouze kontrolními body $P_{i-n}..P_i$. Mezi dvěma segmenty Q_i a Q_{i+1}, kde každé $i \geq n$, existuje spoj s hodnotou t_i parametru t. Pro každý B-Spline existuje $p-n-2$ spojů. B-Spline křivky lze rozdělit do několika kategorií. Pokud platí $\forall i \in [n,p] \wedge t_{i+1} - t_i = t_{i+2} - t_{i+1}$, jedná se o B-Spline uniformní (uniform). Pokud není tato podmínka splněna, jedná se o B-Spline neuniformní (non-uniform). Druhou kategorizací je rozdělení na racionální a neracionální. Racionální B-Spline křivky jsou takové, že body křivky jsou definovány racionálními polynomy. Z těchto dvou kategorizací lze tedy B-Spline křivky rozdělit do 4 kategorií. Nejběžnějším typem jsou B-Spline neuniformní racionální křivky, pro něž se často používá zkratka NURBS (non-uniform rational B-Spline). O NURBS křivkách pojednává [10].

Pro vyjádření křivky NURBS ve 2D prostoru R^2 lze matematicky vycházet ze vzorce (3.13).

$$Q(t) = \frac{\sum_{i=0}^{p} B_{i,n}(t) \cdot P_i \cdot w_i}{\sum_{i=0}^{p} B_{i,n}(t) \cdot w_i} \tag{3.13}$$

kde P_i je kontrolní bod, w_i je jeho váha a $B_{i,n}$ je bázová funkce, která je rekurzivně definovaná jako:

$$B_{i,0}(t) = \begin{cases} 1 & t_i \leq t < t_{i+1} \\ 0 & jinak \end{cases}$$

$$\forall k > 0, B_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} \cdot B_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} \cdot B_{i+1,k-1}(t) \tag{3.14}$$

Výše uvedený vzorec (3.14) je obecnějšího tvaru a platí jak pro B-Spline křivky racionální, tak neracionální. Pro křivky NURBS, tedy neracionální, lze výše uvedený vzorec zjednodušit na tvar (3.15).

$$Q(t) = \sum_{i=0}^{p} B_{i,n}(t) \cdot P_i \tag{3.15}$$

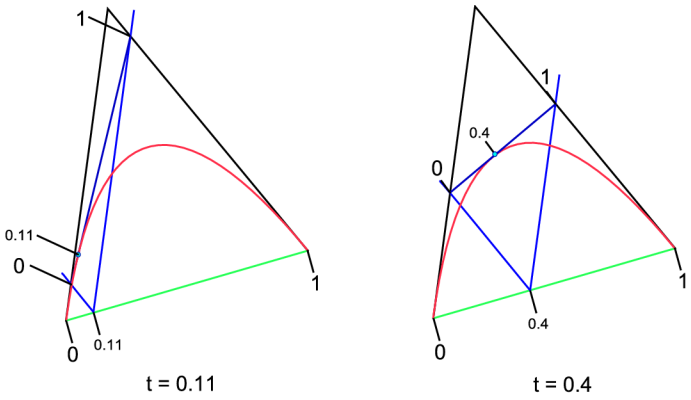
Vyjádření NURBS křivek je poměrně složité, ale jako metoda rasterizace v hardware je realizovatelná. Další informace k rasterizaci B-Spline křivek a práci s nimi je možné nalézt v [5] případně [7].

3.5 Bézierovy křivky

Další možností jak rasterizovat zaoblení jsou kromě NURBS křivek, popsaných v podkapitole 3.4, křivky Bézierovy. Bézierovy křivky jsou obecné parametrické křivky. Vzhledem k matematickému popisu lze křivku popsanou jednoduchou rovnicí vykreslit v nejrůznějších rozlišeních, což přináší vysokou kvalitu zobrazení grafickými zařízeními. Parametrizace křivek dává možnost jednoduše popsat objekt složitého tvaru. Stejně tak jako NURBS křivky, které mohou být plošné, mohou být i Bézierovy křivky rozšířeny na Bézierovy plochy (viz podkapitola 3.5.1). Bézierova křivka je definována body (P_0, P_1, P_2, … P_n), pomocí kterých se křivka vypočítá. Křivka vždy začíná v prvním definovaném bodě P_0 a končí v posledním bodě P_n. Ostatní body definují zakřivení Bézierovy křivky. Zakřivení křivky se chová tak, jako by byla křivka k danému bodu přitahována. Pro n bodů se jedná o Bézierovu křivku stupně n. Obecná rovnice Bézierovy křivky je dána vzorcem (3.16).

$$\begin{aligned} B(t) &= \sum_{i=0}^{n} \binom{n}{i} P_i \, (1-t)^{n-i} \, t^i \\ t &\in [0,1] \end{aligned} \tag{3.16}$$

Do rovnice Bézierovy křivky je dosazen parametr t, dle kterého lze vypočítat výsledek pro určitou část křivky, kde $t = 0$ udává počáteční bod a $t = 1$ koncový. Výpočet bodu křivky v určitém bodě daném hodnotou t je ukázán na obrázku 3.8.



Obrázek 3.8: Vykreslování Bézierovy křivky

Rovnici lze dále zjednodušit na následující tvar (3.17).

$$\begin{aligned} B\,(t) &= \sum_{i=0}^{n} P_i b_{i,n}\,(t) \\ t &\in [0,1] \end{aligned} \tag{3.17}$$

.

Polynum $b_{i,n}$ je pro rovnici (3.17) definován rovnicí (3.18).

$$\begin{aligned} b_{i,n}(t) &= \binom{n}{i} t^i (1-t)^{n-i} \\ i &\in [0,n] \end{aligned} \tag{3.18}$$

Pro zvýšení možností Bézierových křivek může být každý bod obohacen o váhu. Bézierovy křivky, kde body mají navíc definovánu váhu, se nazývají Bernsteinova forma Bézierových křivek. Vyšší váha udává, že je k tomuto bodu křivka více přitahována než k bodu, který má váhu s hodnotou nižší. Rovnice (3.19) Bernsteinovy formy Bézierovy křivky je doplněna o váhy w_0, w_1, ... w_i.

$$B(t) = \frac{\sum_{i=0}^{n} b_{i,n}(t) P_i w_i}{\sum_{i=0}^{n} b_{i,n}(t) w_i} \tag{3.19}$$

Výhodou Bézierových křivek je, že pomocí nich lze velmi dobře vykreslit, jak základní tvary jako například úsečky, tak složité spojité hladké tvary.

3.5.1 Bézierovy plochy

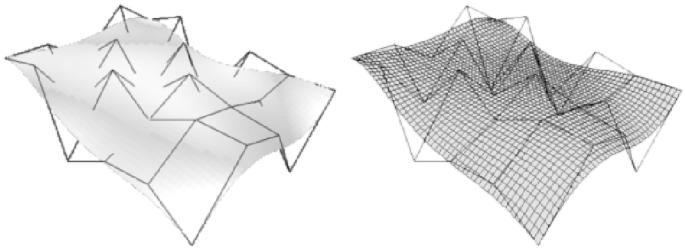
Bézierovy křivky, popsané v předchozí části a demonstrované ve 2D prostoru, lze rozšířit do vícerozměrného prostoru. Typicky se jedná o prostor se třemi rozměry. Body Bézierovy křivky P_0 až P_n jsou rozšířeny na matici $m \times n$ bodů, tedy $P_{i,j}$, kde $i \in 0, 1, ...n, j \in 0, 1, ...m$. Zobrazením takové křivky je Bézierova plocha (viz obrázek 3.9).

Rovnice (3.20) definuje Bézierovy plochy s maticí kontrolních bodů $u \times v$, $(u, v) \in [0, 1]^2$.

$$P(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u) B_j^m(v) k_{i,j} \tag{3.20}$$

kde $B_i^n(u)$ vyjadřuje Bernsteinův polynom daný rovnicí (3.21)

$$B_i^n(u) = \binom{n}{i} u^i (1-i)^{n-i} \tag{3.21}$$



Obrázek 3.9: Bézierova plocha

Může se zdát, že sada útvarů stále není kompletní, avšak Bézierovy plochy i NURBS křivky popsané v podkapitole 3.4, je možné rasterizovat postupně složením více menších křivek a vytvořit tak téměř libovolný tvar. Rasterizaci NURBS křivek i Bézierových ploch je vhodné provádět pomocí sady trojúhelníků. Řídící body trojúhelníků, které mají být využity pro rasterizaci, je nutné vhodně vypočítat. Metoda převodu obecné křivky na trojúhelníky se nazývá teselace. O těchto metodách pojednává následující podkapitola 3.6.

3.6 Teselace

Provádět rasterizaci ploch definovaných křivkami, popisovanými v podkapitolách 3.4 a 3.5.1, je poměrně složitý úkol. Výsledný obraz po rasterizaci je vždy diskrétní. Provedení rasterizace plošné křivky pomocí průchodu prostoru a rasterizací jednoduchých bodů, které jsou součástí plochy, by bylo velmi neefektivní a problematické. Z těchto důvodů se používá metoda, nazývaná teselace, která umožňuje převádět spojitou plošnou křivku na jednoduché útvary. Výsledkem teselace je sled trojúhelníků, které svou vzájemnou návazností co nejvíce pokrývají tvar spojité křivky. Trojúhelníky je nutné vhodně seřadit. O řazení, které využívají paralelní rasterizační řetězce, je napsán článek [8]. Výsledná rasterizace složená ze sledu setříděných trojúhelníků je již mnohem jednodušší. Často se jako teselace využívá algoritmu subdivision-loop. O hardwarové implementaci tohoto algoritmu se lze dočíst v [11].

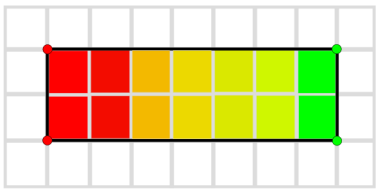
Kapitola 4

Doplňkové algoritmy

Všechny rasterizační metody, uvedené v kapitole 3, generují informaci o poloze rasterizovaných bodů. Buď algoritmus generuje postupně polohy bodů, a nebo se testují všechny body na vlastnost, zda patří či nepatří rasterizovanému tvaru. Žádná metoda nezahrnuje zpracování informací o barvě rasterizovaného bodu. U řídících bodů, které vstupují do rasterizační jednotky, je barva definována. Ovšem typicky má objekt méně řídících bodů (například úsečka dva, trojúhelník tři, ...), než je počet všech bodů, které jsou rasterizovány vhodnou metodou. Pro tyto body je nutné barvu dopočítat z barvy řídících bodů. Dle počtu řídících bodů se zavádí metoda interpolace barev, o které pojednává podkapitola 4.1. Při rasterizaci dochází často k určitým nedostatkům. Jelikož jsou polohy bodů rasterizovaného objektu získány s výsledkem *bod patří/bod nepatří* objektu, dochází k chybě, které se říká aliasing. Vizuálně tvar působí dojmem, že jeho okraj je hranatý. Tento jev lze do jisté míry potlačit pomocí anti-aliasingu, který je popsán v podkapitole 4.2.

4.1 Interpolace barev

Požadavkem rasterizace objektů je automatický výpočet barev pro body, které náleží danému tvaru. Jelikož každý tvar (mimo bod) má alespoň dva řídící body, kde každý z nich má definovánu barvu, je nutné využít vhodný algoritmus pro výpočet barevné informace. Často využívanou metodou je lineární přechod mezi barvami řídících bodů, tedy lineární interpolace (viz obrázek 4.1). Interpolaci lze obecně provádět mezi libovolným počtem vstupních barev. Pro dva body se metoda nazývá lineární interpolace, pro tři body bilineární interpolace.



Obrázek 4.1: Příklad interpolace barev

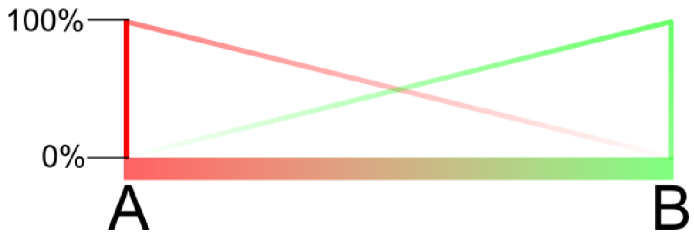
Lineární interpolaci barev lze provádět mezi dvěma libovolnými barvami. Výsledkem

lineární interpolace je barva s určitou mírou řídící barvy **A** a s určitou mírou řídící barvy **B**. Intenzita barevné hodnoty klesá od jedné barvy ke druhé s lineární závislostí a to od 100% do 0% (viz obrázek 4.2).

Vzorec lineární interpolace barvy t mezi dvěma barva A a B je uveden pod (4.1).

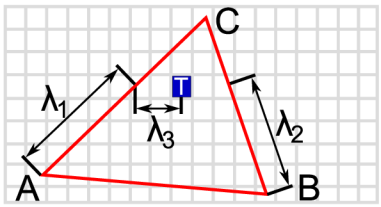
$$t = \lambda \cdot A \ + (1 - \lambda) \cdot B \tag{4.1}$$

Hodnota λ definuje relativní vzdálenost od místa absolutní barvy A k místu absolutní barvy B a to v intervalu $\lambda \in < 0, 1 >$, kde pro $\lambda = 1$ je výsledkem interpolace absolutní barva A a pro hodnotu $\lambda = 0$ je výsledkem interpolace absolutní barva B.



Obrázek 4.2: Interpolace mezi dvěma barvami

Interpolaci barev lze využít u libovolných rasterizačních úloh. Typickým využitím může být interpolace barev pro anti-aliasing (viz podkapitola 4.2) při rasterizaci přímky. Podobně lze interpolaci barev využít při rasterizaci ostatních útvarů (kružnice, kruh, ...). Pokud se jedná o tvar se třemi řídícími body, mezi kterými je rasterizován výsledný tvar (trojúhelník), lze využít pro výpočet barvy jednotlivých rasterizovaných bodů bilineární interpolaci.



Obrázek 4.3: Výpočet lineární interpolace

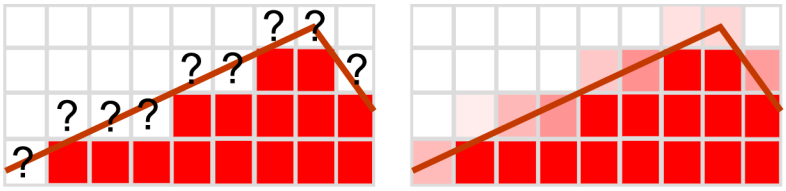
Bilineární interpolace se skládá z dvojitého výpočtu (4.2) lineární interpolace (4.1) mezi dvěma body. Tuto interpolaci lze vypočítat ve dvou krocích. V prvním kroku jsou vypočítány interpolace barvy na hranách trojúhelníku, které jsou následně v druhém kroku využity jako krajní body pro interpolaci barvy v daném bodě T (viz obrázek 4.3). Ve všech výpočtech pro lineární interpolaci je využita přímá lineární závislost mezi dvěma krajními body. Každý vrchol trojúhelníku má definovanou určitou barvu, jejíž intenzita s lineární závislostí klesá k černé. Jedná se o závislost barvy na vzdálenosti od vrchnolu, kterém tato barva náleží. Minimální intenzity (tedy černé barvu) dosahuje v místě ostatních vrcholů.

$$
\begin{aligned}
x &= \lambda_1 \cdot A + (1 - \lambda_1) \cdot C \\
y &= \lambda_2 \cdot B + (1 - \lambda_2) \cdot C \\
T &= \lambda_3 \cdot x + (1 - \lambda_3) \cdot y
\end{aligned} \tag{4.2}
$$

Na hraně trojúhelníku jsou vypočítány barvy v určité vzdálenosti od dvou vrcholů přilehlých k dané straně a pomocí aditivního skládání barev jsou tyto dvě barvy sloučeny do výsledné barvy odpovídající bilineárně interpolované barvě. Interpolační metody nejsou omezeny jen na lineární a bilineární interpolace. Další možností je například bikubická interpolace, o které je v souvislosti s hardware napsáno v [9].

4.2 Anti-aliasing

Při rasterizaci dochází k převodu spojitých, rovnicemi definovaných, tvarů na diskrétní body. Vlivem této diskretizace dochází často k velkým nepřesnostem, které u výsledného obrazu vytváří nepříjemné nedostatky na kvalitě obrazu (viz obrázek 4.4(a)).



(a) Chyby zobrazení bez interpolace (b) Aplikace interpolace bodů

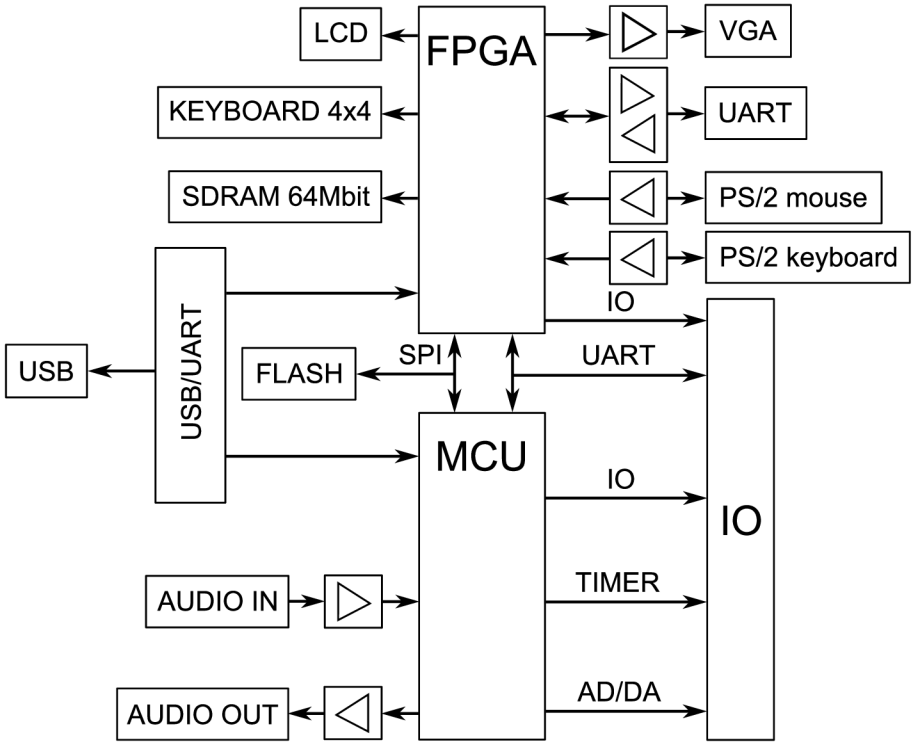
Obrázek 4.4: Interpolace bodů v trojúhelníku

U rasterizačních metod lze tyto jevy potlačit pomocí tzv. anti-aliasingu. Anti-aliasing mění styl uvedené diskretizace tak, že ruší dvou stavovou hodnotu **bod je součástí objektu/bod není součástí objektu**. Navíc definuje pro každý bod, jak velikou částí zasahuje do objektu. Při rasterizace lze přímo této úvahy využít a i body, které jen částečně zasahují do objektu, zahrnout do rasterizace. Tyto body je nutné vykreslit vhodně upravenou barvou (viz obrázek 4.4(b)). Barvu bodů lze vypočítat metodou interpolace barev, která je popsána v podkapitole 4.1. Interpolace vyžaduje alespoň dvě hodnoty, mezi kterými barevná interpolace probíhá. U anti-aliasingu je jedna barva definovaná rasterizovaným objektem a jako druhou lze s výhodou využít původní barvu rasterizovaného bodu, než bude nahrazena barvou aktuálně vypočítanou. Jinou možností, kterou lze vytvořit zobrazení s nižší úrovní aliasingu, jsou metody nazývané obecně super-sampling. Tyto metody jsou založeny na rasterizaci několikanásobně více bodů, než je výsledná rasterizační mřížka. Zpravidla to bývá například dvojnásobné rozlišení, ale rasterizovat lze i jiné násobky výsledného rozměru. Poté se provádí metoda super-sampling, která vždy z několika bodů vytvoří jeden výsledný. Například pokud je rozlišení dvojnásobné, výsledný bod tvoří matice 2x2 bodů. Vlastním algoritmem, který vypočítá výslednou barvu bodu, může být například bikubická či bilineární interpolace [6].

Kapitola 5

Hardwarové možnosti, FITkit

Metody popisované v tomto dokumentu jsou směřované na libovolný hardware obsahující obvod FPGA, který je vybavem dostatečným množtvím paměti a analogovým převodníkem. Grafickým výstupem může být například grafický port VGA. Tyto dva předpoklady splňuje platforma FITkit. Platforma FITkit je vzkvétajícím projektem, který umožňuje studentům fakulty informačních technologií Vysokého učení technického v Brně rozvíjet své praktické zkušenosti i v oblasti hardware, který je úzce spojen s informačními technologiemi. FITkit také poskytuje možnost nahlédnout na problematiku vestavěných systémů.



Obrázek 5.1: Blokové schéma FITkit

Na platformě FITkit je umístěn obvod FPGA, tedy programovatelné hradlové pole. Aby byly možnosti platformy co možná nejširší, obsahuje FITkit také výkonný mikrokontrolér s nízkým příkonem (dále jen MCU) a řadu periférií [3]. Mezi periferiemi lze najít i grafický

port VGA, k němuž je možné připojit řadu grafických zařízení, mezi které patří monitory, projektory a další. FITkit obsahuje také dvě sběrnice, které je možné využít pro libovolný účel k rozšíření této platformy. Základní blokové schéma je uvedeno na obrázku 5.1. Programování platformy FITkit a základní komunikace probíhá terminálovým protokolem přes USB. Připojení platformy FITkit k napájecímu zdroji způsobí nakonfigurovaní FPGA a start zařízení, to vše provádí MCU. Prvotním úkolem je inicializace terminálu a nahrávání konfigurace hradlového pole FPGA. Pro statické uložení konfigurace může být využito paměti FLASH. Po těchto základních úkonech je možné s platformou FITkit komunikovat pomocí příkazů z terminálu, a nebo provádět naprogramovanou úlohu mikrokontrolérem MCU a obvodem FPGA. Pro účely akcelerace grafickým hardware je možné implementovat v obvodu FPGA vhodné metody uvedené v kapitole 3. Mikrokontrolér lze přitom využít pro generování požadavků na rasterizaci. Jako paměť pro grafické operace lze použít rychlou blokovou paměť zabudovanou v obvodu FPGA, případně paměť s několikanásobně vyšší kapacitou typu SDRAM, která je taktéž připojena k obvodu FPGA. Programovatelné hradlové pole FPGA, které je umístěno na platformě FITkit, je vyrobeno firmou Xilix Inc. [4]. Jedná se o FPGA typu Spartan 3 XC3S50 PQ208. O tomto obvodu blíže pojednává samostatná podkapitole 5.1. Mikrokontrolér MCU platformy FITkit je z řady obvodů MSP430F1xx. Pro zápis programového kódu je možné využít překladač MSPGCC [1], který umožňuje překlad z jazyka C do vhodného binárního kódu včetně programování pomocí zvoleného portu.

5.1 Architektura FPGA

Architektura FPGA obvodů od firmy Xilinx Inc. je v poslední době špičkou na trhu programovatelných obvodů. Jádrem těchto obvodů je rekonfigurovatelný prostor složený z menších elementů. Strukturu obvodu znázorňuje obrázek 5.2. Tyto rekofigurovatelné bloky bývají označovány zkratkou CLB (configurable logic blok). Jednotlivé bloky bývají v obvodu uloženy ve tvaru čtvercové mřížky. Jejich počet udává kapacitu a zároveň cenu FPGA obvodu. Každý blok CLB se skládá z několika dalších menších struktur, které bývají označovány SLICE. Počet bloků SLICE v buňkách CLB se v jednotlivých FPGA obvodech liší. Blok SLICE představuje obecnou buňku, která obsahuje několik základních komponent, již lze využít při návrhu výsledné architektury obvodu. Jedná se o look-up-table (LUT), klopný obvod typu D a multiplexory. Počet vstupních signálů LUT bývá zpravidla čtyři, ale například obvody firmy Xilinx Inc. řady Virtex Pro mohou mít až šest vstupních signálů pro LUT. Klopný obvod D je možno využít pro realizaci sekvenčních obvodů a uložení bitových hodnot, které jsou výstupem LUT.

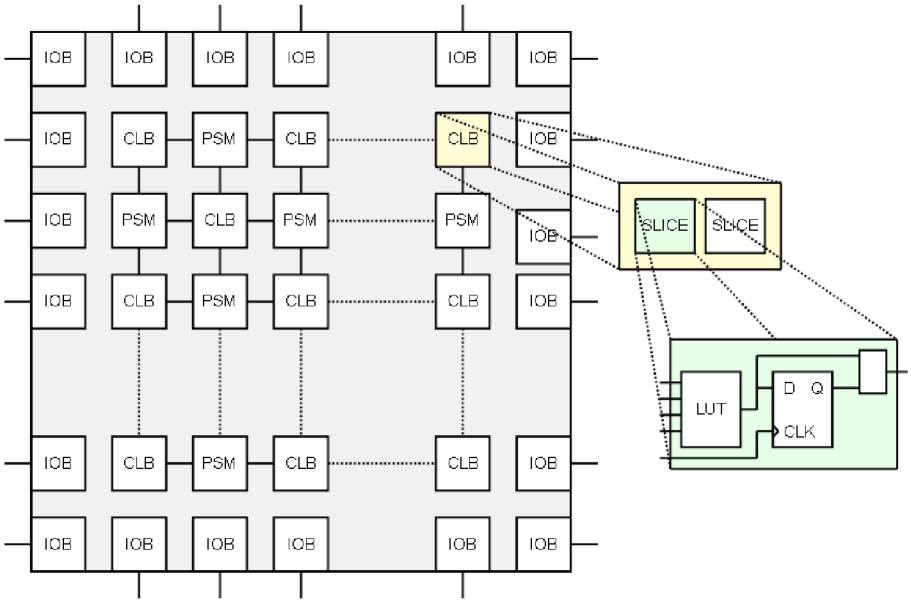
Vnitřní struktura složená z CLB bloků je propojena sofistikovanou propojovací sběrnicí, která tvoří přibližně devadesát procent objemu obvodu. Tuto sběrnici lze rozdělit do několika kategorií.

První kategorií jsou sběrnice rozvádějící hodinový signál. Jedná se vodiče, které vykazují velmi malá zpoždění. Jejich využití je vždy jen pro řídící hodinový signál. Přístup k této sběrnici umožňují zvláštní komponenty nazývané ve VHDL popisu jako GBUF.

Druhou částí jsou rozvody dat po čipu. Jedná se již o pomalejší typy vodičů, které

umožňují rozvod dat mezi jednotlivým částmi obvodu. Zpravidla je lze rozdělit na dlouhé a krátké. Delší cesty rozvádějí signály mezi vzdálenějšími bloky CLB a kratší propojují vždy několik bližších buněk. V běžných obvodech FPGA tyto sběrnice umožňují nabývat hodnot logického signálu H a L, ale v případě dražších obvodů lze na sběrnici pracovat i se signálem vysoké impedance Z.

Celá mřížka CLB včetně sběrnice rozvodů je připojena k externím pinům pouzdra buňkami, které bývají označeny IOB. Buňky IOB umožňují zpracování vhodných úrovní signálů, jakými mohou být různé úrovně logických signálů (TTL, CMOS, ...), vytvoření třístavového portu pro realizaci vstupu s vysokou impedancí a podobně.



Obrázek 5.2: Zjednodušená struktura obvodů FPGA

Hodinový signál, který je přenášen speciální sběrnicí s malým zpožděním, je typicky předzpracován a modifikován. Předzpracováním se rozumí průchod částí tzv DCM obvodu. DCM blok umožňuje korekce hodinového signálu, násobení frekvencí určitým koeficientem, fázový posun atd. Tyto hodnoty je možné vhodně nastavit a vytvořit uvnitř FPGA čipu i několik hodinových domén.

Mimo tyto základní prvky obsahují obvody FPGA i další často využívané komponenty. Jejich ruční vytvoření z CLB bloků by bylo náročné na plochu a dosahovaly by horších výsledků (menší pracovní frekvence, vyšší zpoždění). Typicky se jedná o vestavěné blokové paměti RAMB, násobičky, zabudované procesory (PowerPC) a podobně. Jejich počet se liší dle zvoleného FPGA obvodu.

5.1.1 Spartan XC3S50 PQ208-4

FPGA obvody s označením **XC3S**50 PQ208-4 patří do řady Spartan, kterou lze zařadit do třídy nejlevnějších FPGA obvodů firmy Xilinx. Další část označení XC3S**50** PQ208-4 udává nepřímo kapacitu obvodu. Tento obvod je nejmenším a nejlevnějším dostupným obvodem řady Spartan. Pouzdro obvodu nese označení PQ208 a udává jeho tvar i počet fyzických

pinů obvodu (208). Nejdůležitější částí označení z pohledu výkonu je koncovka **-4**. Toto číslo udává rychlost obvodu. Čím vyšší číslo, tím menší zpoždění mají jednotlivé komponenty obvodu.

Obvod XC3S50, který je cílovým pro navrhovanou grafickou akceleraci, obsahuje 1700 logických programovatelných buněk CLB, dva hodinové generátory DCM a čtyři 18b násobičky násobičky sdílné s blokovými pamětmi RAMB.

5.1.2 Ostatní obvody FPGA Xilinx

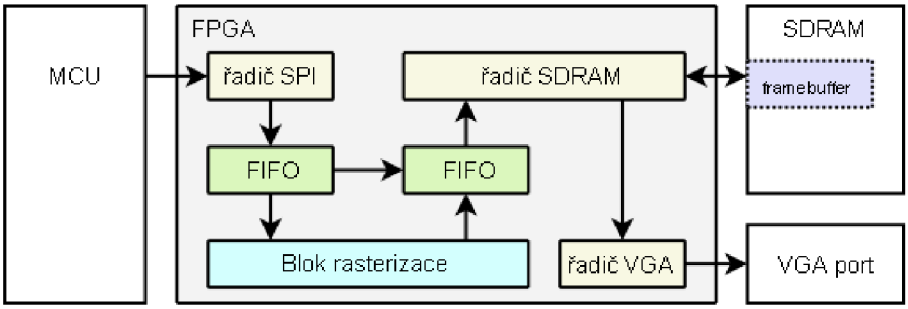
Obvod XC3S50, který je součástí platformy FITkit, je nejmenší FPGA obvod z vyráběné řady Spartan firmy Xilinx Inc. Lze tedy očekávat, že pro některé architektury již jeho kapacita nebude dostačovat. V kapitole 7, kde jsou zhodnoceny dosažené experimentální výsledky, jsou zmíněné i další FPGA obvody, pro které byla provedena syntéza. Jedná se o obvody XC3S200, XC3S250, XC2V250 a XC2VP2.

Obvody XC3S200 a XC3S250 patří da řady Sparat a lze je řadit k levnějším. Liší se z hlavně kapacitou, tedy počtem CLB a počtem vestavěných obvodů (např. násobičky, paměti, ...). Obvody s označením XC2V250 a XC2VP2 náleží řadě Virtex (resp. Virtex Pro II), která nepatří k levným modelům. Jejich přínosem k navrženým architekturám je menší zpoždění jejich vnitřních hradel oproti obvodům řady Spartan. Navíc také obsahují další vestavěné obvody jako například procesory PowerPC apod. Tyto komponenty ovšem efektivitu navržené architektury neovlivní, protože nejsou součástí návrhu a zůstávají nevyužité.

Kapitola 6

Praktická realizace

Cílová platforma popsaná v kapitole 5 může být využita pro jednoduché grafické aplikace. Na dostupném FPGA obvodu byl vytvořen kompletní základní rasterizační řetězec, který je primárně určen právě pro tuto platformu. Prakticky navržený řetězec demonstruje dva typy problémů – rasterizaci čar (kapitola 7.1) a rasterizaci trojúhelníků (kapitola 7.2). Z kapacitných důvodů se jedná o dvě nezávislé jednotky, které jsou založeny na velmi podobné generické architektuře, kterou znázorňuje obrázek 6.1.



Obrázek 6.1: Blokové schéma rasterizační architektury

Z důvodu kapacity hradlového pole se navržená architektura skládá pouze z nejnutnějších bloků, které řeší rasterizaci. Rasterizaci řídí mikrokontrolér přes sběrnici SPI (sériová linka). Proto je vstupním blokem řadič SPI spolu s adresovým dekodérem. Tento řadič obsluhuje tok dat sběrnicí SPI a umožňuje zprostředkovat data odeslaná mikrokontrolérem k dalšímu zpracování. Druhým blokem je paměť tvořící frontu (FIFO - first in first out), která je popsána v kapitole 6.2. Blok realizující vlastní proces rasterizace je vysvětlen v kapitole 6.4 pro rasterizaci čar a v kapitole 6.5 pro rasterizace trojúhelníku. Koncepce těchto architektur se liší pouze ve vstupní části, neboť množství dat se pro tyto dva objekty liší v šířce dat řadiče SPI a paměti FIFO. Výstupem bloku rasterizace je sled bodů definovaných pozicí a barvou. Tato data jsou již pro oba typy navrženého rasterizačního bloku stejná, a proto se zbývající část architektury neliší. Hodnoty rasterizace je nutné uložit do vhodného bloku paměti tvořícího tzv. framebuffer. K tomuto účelu lze na platformě FITkit využít paměť SDRAM (synchronous dynamic random access memory). Jelikož paměť SDRAM obsahuje pouze jeden port a vyžaduje složitější řízení než statické typy pamětí

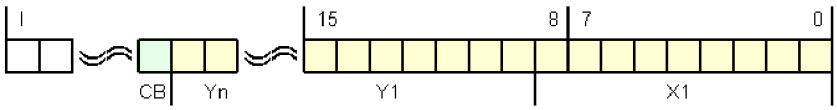
(SRAM), je mezi řadič této paměti a výstup rasterizačního bloku vložena druhá paměť typu FIFO. Za touto pamětí již následuje sdílený řadič paměti SDRAM, který je podrobněji popsán v kapitole 6.3. Platforma FITkit obsahuje grafický port VGA, jenž umožňuje zobrazovat nejrůznější data. Součástí navržené architekture je proto řadič grafického portu VGA. Grafický port vyžaduje přesné časování a řadič tohoto portu musí mít k dispozici v přesně definovaný okamžik platná data. Je tedy, stejně jako výše uvedená fronta FIFO, připojen k řadiči SDRAM, ovšem s vyšší prioritou. Následující kapitoly jsou rozděleny od návrhu k realizaci ve směru toku dat, tedy od řídícího mikrokontroléru po výstupní grafický port VGA. Každá podkapitola popisuje část řešeného problému. Poslední podkapitoly se zabývají experimentálními výsledky.

6.1 Řízení rasterizace

V předchozí kapitole 6 bylo uvedeno, že mikrokontroler generuje požadavky na rasterizaci. Jelikož smyslem práce bylo vytvořit akcelerátor, měl by mikrokontrolér zpracovávat co nejmenší možnoství dat a stejně tak množství dat poslanené pomocí komunikačního protokolu SPI. Množství zpracovávaných matematických výpočtů by mělo být řešeno až v navrhované architektuře FPGA. Ať už se jedná o řízení rasterizace čar či trojúhelníků, je nezbytně nutné předat mezi mikrokontrolérem a hradlovým polem FPGA alespoň informace o všech vrcholech útvaru, barvě a případně několik vhodných doplňkových bitů se specifickým účelem.

Jednoduché aplikace, které si vystačí s rasterizací útvaru o určité barvě, vyžadují minimálně jeden doplňkový bit. Tímto bitem je požadavek mikrokontroléru na vymazání frame-bufferu.

Adresový dekodér FITkitu umí pracovat s daty o délce N-bytových (8-bitových) bloků. V FPGA mohou být jednotlivé bity dostupné přímo za sebou v bitovém toku. Realizovat v mikrokontroléru spojení číselných hodnot do delšího bytového bloku není náročné, a proto byl implementován jednoduchý převod vstupních hodnot na N-bytové pole. Toto pole je následně odesláno do FPGA. Strukturu dat naznačuje obrázek 6.2.



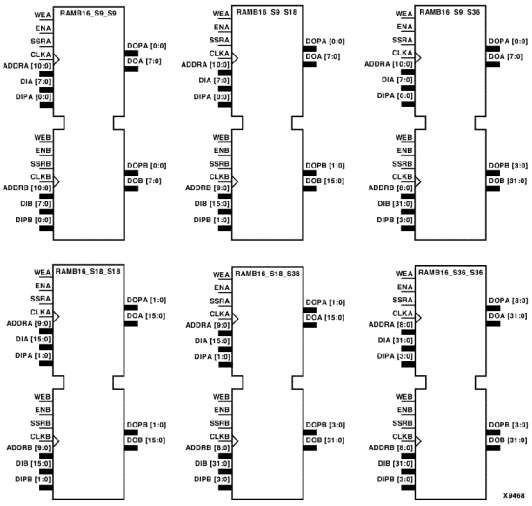
Obrázek 6.2: Struktura dat přenášených z MCU do FPGA

Z uvedeného obrázku je možné vidět, že jednotlivé hodnoty jsou složeny bitově za sebe a nejvyšší bit označený CB určuje, zda má být provedeno vymazání frame-bufferu či nikoliv.

6.2 Blok fronty - FIFO

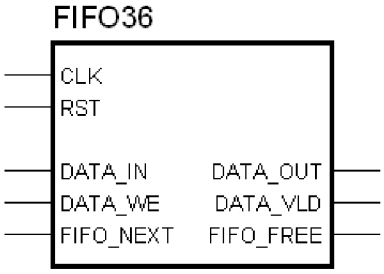
Navržená architektura vyžaduje realizovat na dvou místech frontu typu FIFO (viz obrázek 6.1). Jedná se o následující dva případy – fronta požadavků na rasterizaci a fronta vypočtených obrazových bodů rasterizovaného obrazce. Frontu FIFO lze realizovat dvěmi způsoby.

Prvním způsobem je realizace pomocí registrů FPGA. Tato metoda může být vhodná pro fronty s menší datovou šířkou. Jejím kladem je rychlost a hlavně dostupnost ve kterékoliv části FPGA obvodu. Druhou možností je využití blokové paměti RAMB16. Její hlavní výhodou oproti předchozímu řešení je možnost využití větší šířky dat beze ztráty zdrojů FPGA. Sběrnice vyžadující frontu FIFO mají poměrně velkou šířku dat. Vstupní fronta pro rasterizaci čar vyžaduje minimální šířku dat 45 bitů (4x9 bitů vektory polohy, 1x8 bitů barva, 1 bit extra) a pro rasterizaci trojúhelníku 63 bitů (6x9 bitů vektory polohy, 1x8 bitů barva, 1 bit extra). Fronta za rasterizačním blokem musí pojmout pozici bodu rasterizovaného útvaru a jeho barvu, což vyžaduje minimální šířku 26 bitů (2x9 bitů vektory polohy, 1x8 bitů barva). Bloková paměť FPGA RAMB16 umožňuje nanejvýš 36 bitů, a to pomocí portu DIA/DIB včetně využití paritních bitů portu DIPA/DIPB (viz obrázek 6.3).



Obrázek 6.3: Typy komponent RAMB16

Cena při volbě metodou registrů by byla příliš velká, a proto navržený blok realizující frontu využívá blokovou paměť RAMB16_S36_S36. Jedná se o dvou portovou blokovou paměť RAMB16 s šířkou dat celkem 36 bitů. V případě vstupních front, kdy je požadována šířka 45, resp. 63 bitů, jsou do fronty vždy ukládány dva záznamy za sebou. Takto je vyřešen problém s příliš velkou šířkou vstupních dat. Efektivní využití takové fronty je 87,5 %.



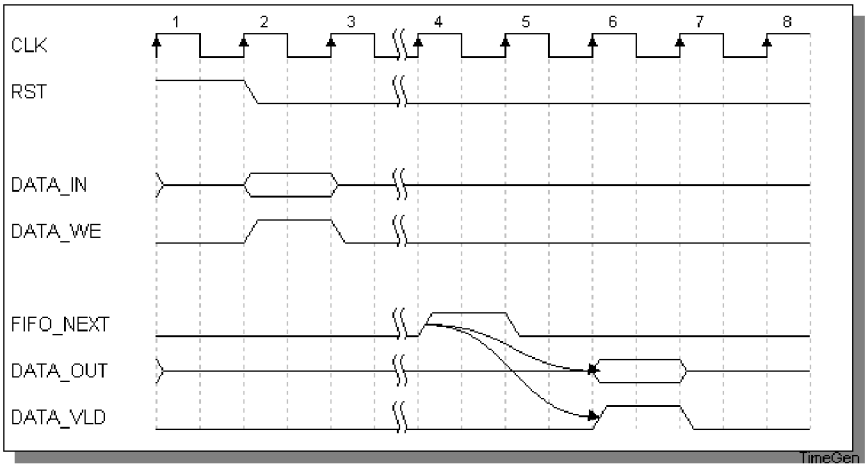
Obrázek 6.4: Rozhraní navržené fronty FIFO36

V případě výstupní fronty, kde je nutné ukládat 26 bitů, je do fronty s šířkou 36 bitů vždy uložen celý záznam obsahující pozici bodu a barvu. Vyšší bity se nevyužívají. Účinné

využití takové fronty je 72 %. Toto využití není příliš efektivní, ale komponenta s nejbližší menší šířkou, tedy RAMB16_S18_S18, je již pro 26-bitová data příliš malá.

Délka fronty je dána kapacitou použité paměti, která je pro uvedené FPGA 2 kB. Pro tuto velikost a šířku dat 36 bitů je délka fronty 512 záznamů. Rozhraní bloku realizujícího navrženou frontu je ukázáno na obrázku 6.4.

Vstupní/výstupní data jsou připojena na sběrnici DATA_IN (resp. DATA_OUT). Data jsou do fronty uložena s náběžnou hranou v případě, že má signál DATA_WE hodnotu H. Čtení probíhá nastavením signálu FIFO_NEXT na H. Jakmile jsou data na sběrnici DATA_OUT platná, je na jeden takt nastavena hodnota H na signálu DATA_VLD. Signál FIFO_FREE umožňuje detekovat zaplnění fronty. Do stavu L je vystaven v případě, že je fronta zaplněna a setrvává v něm po dobu, dokud není přečten alespoň jeden blok. Časový diagram funkce fronty je ukázán na obrázku 6.5.



Obrázek 6.5: Časový diagram bloku FIFO36

Navržený blok architektury pro rasterizaci čar po syntéze obsahuje 28 registrů a 16 LUT. Maximální frekvence tohoto bloku je dána nejdelší cestou a odpovídá 142 MHz.

6.3 Řadič paměti SDRAM

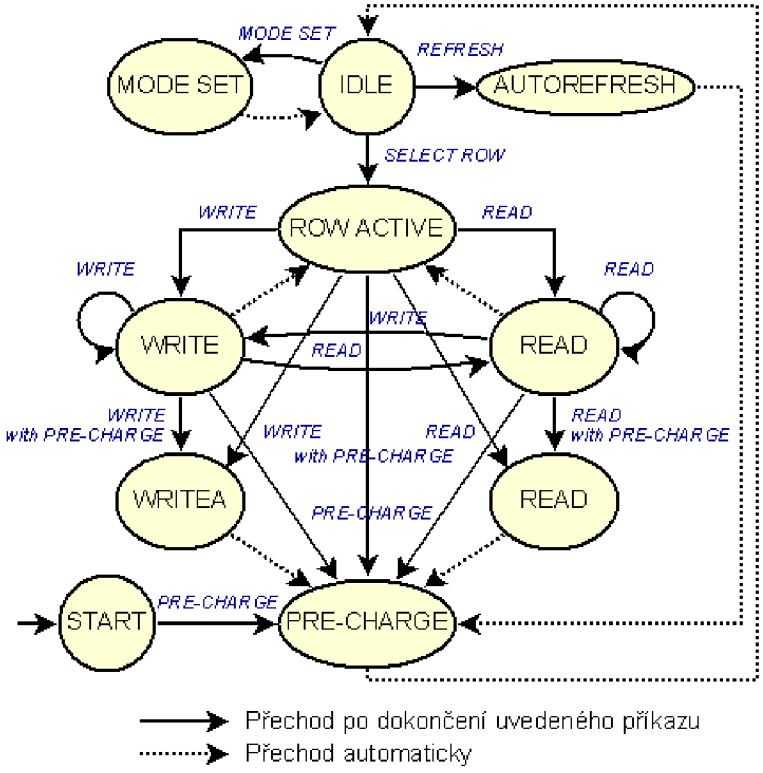
Jednou ze složitějších součástí architektury, je řadič synchronní dynamické paměti SDRAM. Konkrétně se jedná se paměť typu GM72V66841ET7K. Na základě uvedeného typu je možné zjistit, že se jedná o 3,3 V paměť SDRAM s kapacitou 64 Mbitů. Tato paměť je rozložena do 4 banků. Tyto banky jsou nezávislé části paměti umožňující podporu prokládaného čtení/zápisu. Každý bank obsahuje 4096 řádků po 512 sloupcích. Šířka dat na jednotlivých adresách je 8 bitů. Pro tuto paměť je dále charakteristické provádět 4096 obnovovacích cyklů za čas 64 ms, aby nedošlo ke ztrátě dat. Nutnost obnovacích cyklů je dána konstrukcí paměti. Paměť je konstruována z malých tranzistorů a kondenzátorů, které umožňují držet po určitou dobu náboj a tím i informaci o stavu daného bitu. Vlivem vybití náboje kondenzátorů dochází ke ztrátě uložených dat. Celá paměť typu SDRAM funguje na principu

konečného automatu uvedeného na obrázku 6.6. Při řízení paměti je možné využít modifikovanou verzi takového automatu.

Přechod mezi jednotlivými stavy automatu zajišťují jednotlivé příkazy. Každý příkaz trvá určitý počet taktů, ale paměť umožňuje využít některé optimalizace prokládání přístupu na úrovni jednotlivých banků paměti a tím zvýšit efektivitu práce. Z výše popsaných vlastností je patrné, že řízení tohoto typu paměti vyžaduje sofistikovaný způsob přístupu, který umožní v co nejvyšší míře využít dostupné optimalizace.

6.3.1 Řízení pamětí SDRAM

Jak bylo popsáno v předchozí části dokumentu, paměti SDRAM se chovají podle určitého stavového automatu. Tento automat ve zjednodušené podobě ukazuje obrázek 6.6. Nejefektivnějším a pravděpodobně nejjednodušším způsobem, jak správně řídit tyto paměti, je vytvořit architekturu s automatem, který určitým způsobem kopíruje stavy automatu SDRAM. Automat řadiče proto mimo několika dalších stavů obsahuje také stavy identické se stavy paměti. Jedná se o stavy Idle, LMR (load mode register), Select, Read, Write, Charge a Refresh.



Obrázek 6.6: Zjednodušený automat řízení paměti SDRAM

Každý z těchto stavů trvá přesně definovaný počet cyklů vstupního hodinového signálu CLK a jejich návaznost určují přechody automatu 6.6. Přesné doby jednotlivých příkazů a další důležité hodnoty jsou definovány výrobcem konkrétní paměti. Detailní informace o časování pro paměť dostupnou na platformě FITkit je možné nalézt v [2].

6.3.2 Obnovovací cykly

Komplikací, která nastává při řízení tohoto typu paměti, jsou obnovovací cykly. Obnovací cykly musí být vystřídány po určitém počtu příkazů. Tato vlastnost může být pro některé aplikace do jisté míry limitujícím faktorem, protože mohou požadovat neustálou dostupnost dat z paměti SDRAM po určitý interval. V takovém případě se naskýtá jediné řešení a to vložení obnovovacích cyklů do určitých časových pásem, kde aplikace přístup k paměti nevyžaduje. Pro jiné typy je výhodnější zajistit obnovovací cykly řadičem paměti SDRAM automaticky ve vhodných intervalech. Navržená architektura řadiče proto umožňuje výběr typu řízení obnovovacích cyklů a to buď automaticky nebo manuálně vystavením požadavku na vstupu řadiče.

V případě automatického řízení obnovovacích cyklů je řadič navržen pro frekvenci hodinového signálu $CLK = 50MHz$. Vnitřní čítač obnovovacích cyklů je řízen tímto hodinovým signálem CLK a nastaven tak, aby za čas 64 ms provedl obnovu všech 4096 dostupných řádků paměti ve všech bancích a to tak, že provádí obnovovací cyklus v intervalu 15.360 μs případně v nejbližší možný čas po dokončení prováděného příkazu. Samostatný obnovovací cyklus trvá sedm taktů hodinového signálu CLK. Pokud je před tímto příkazem aktivní nějaký bank, je nutné provést deaktivaci banků, která trvá tři takty hodinového signálu CLK. Reaktivace banku nakonec vyžaduje čtyři takty hodinového signálu CLK. Kompletní obnovovací cyklus tedy znemožní přístup k paměti SDRAM na dobu čtrnáct taktů hodinového signálu. Pokud aplikace není závislá na přesném časování a nevyžaduje data s určitou maximální latencí, jsou takto obnovovací cykly lineárně rozděleny v čase. Paměť je tudíž dostupná poměrně rychle v kterýkoliv čas a nejhorším možným stavem je prodloužená latence způsobená probíhajícím obnovovacím cyklem.

Při využití řadiče SDRAM, jako například zdroje dat pro řadič grafického portu, je nutné řídit obnovovací cykly manuálně ve vhodných časových pásmech. V takovém případě lze využít příkaz obnovy nastavený na sběrnici CMD potvrzený signálem CMD_WE. Protože je možné provádět tento příkaz bez problémů sekvenčně, lze tak vyhradit učitý časový blok pro obnovovací cykly paměti SDRAM.

Navržený řadič dále umožňuje aktivovat řádky postupně pro více banků, umí využít optimalizace typu read-after-read, write-after-write a read-after-write. Není tedy nijak omezen pro konkrétní aplikaci a naopak je navržen pro široké využití.

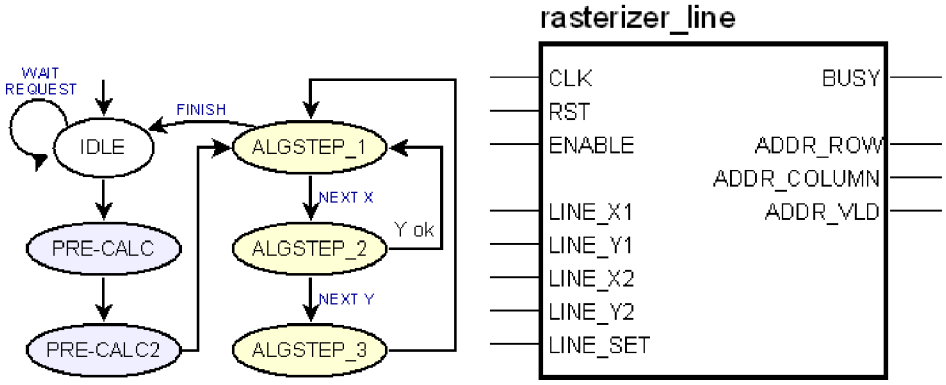
Řadič paměti SDRAM po syntéze obsahuje 109 registrů a 162 LUT. Maximální frekvence tohoto bloku dána nejdelší cestou odpovídá 76 MHz.

6.4 Blok rasterizace čar

Prvním blokem, který umožňuje vykreslovat jednoduché útvary, je blok rasterizace čar 6.7. Jádrem této architektury je Bresenhamův algoritmus popsaný v kapitole 3.3. Algoritmus je řízen jednoduchým konečným automatem, který obsahuje šest stavů. Tento automat je zobrazen na obrázku 6.7(a). Po inicializaci komponenty signálem RESET a náběhu hodinového signálu se nachází automat ve stavu Idle. Rasterizace je spustěna ihned na následující náběžné hraně, kdy je hodnota signálu LINE_SET (viz obrázek **??**) v log. 1.

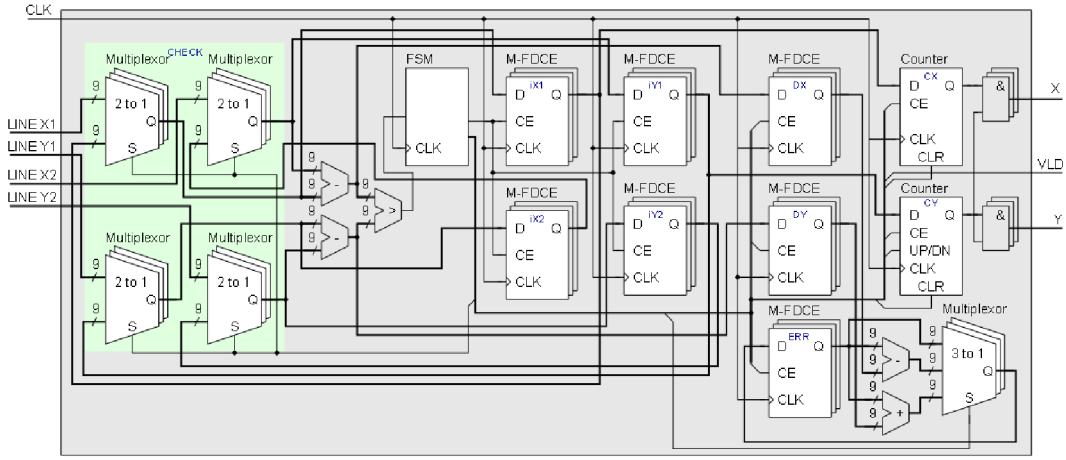
Všechny čtyři hodnoty vstupních vektorů LINE_X1/X2/Y1/Y2 jsou na této náběžné

hraně uloženy do vnitřních registrů a automat přechází do stavu Pre-calc. V tomto stavu jsou odečteny hodnoty $X1 - X2$ a $Y2 - Y1$, s jejichž pomocí je možné zjistit, který z těchto rozdílů hodnot je větší, a vhodně prohodit body $[X1, Y1]$ a $[X2, Y2]$ tak, aby vždy platilo $X1 \leq X2 \wedge |X2 - X1| > |Y2 - Y1|$.



(a) Automat řízení bloku rasterizace čar (b) Rozhraní bloku rasterizace čar

Z tohoto stavu automat přechází do stavu Pre-calc2, ve kterém jsou odečteny nové hodnoty $Y1$ a $Y2$. Výsledkem je hodnota registru udávající kladný či záporný krok na souřadnici Y. Ze stavu Pre-calc2 automat přechází do stavu Algstep_1, ve kterém jsou aktuální hodnoty vnitřních registrů X a Y vystaveny signály rozhraní ADDR_ROW a ADDR_COLUMN. Jejich platnost je potvrzena signálem ADDR_VLD. Z tohoto stavu přechází automat do stavu Algstep_2. Po dokončení rasterizace čáry se automat vrací zpět do stavu čekání, tedy Idle.



Obrázek 6.7: Vnitřní struktura bloku rasterizace čar

Navržená architektura bloku rasterizace čar sestává po syntéze z 95 registrů a 320 LUT. Maximální frekvence tohoto bloku daná nejdelší cestou je 57 MHz.

6.5 Blok rasterizace trojúhelníku

Rasterizace čar není dostatečnou ukázkou a demonstrací využití možností hradlového pole Spartan FITkitu. Rasterizace čar navíc sama nedostačuje pro vykreslení většiny útvarů. Z těchto důvodů byl navržen blok, pomocí kterého lze rasterizovat jiný základní prvek grafiky, kterým je trojúhelník.

Návrh architektury bloku rasterizace trojúhelníků je složitější než návrh architektury bloku rasterizace čar a to z důvodu dostupné kapacity cílového FPGA. Při návrhu bylo zapotřebí zohlednit a prakticky testovat více možných realizací tak, aby cílový návrh zabíral co nejméně plochy čipu FPGA a umožňoval dosáhnout co možná nejvyšší řídící frekvence hodinového signálu. Následující části kapitoly pojednávají o těchto možnostech návrhu.

6.5.1 Algoritmus

Algoritmus, který byl použit v bloku rasterizace trojúhelníku, je popsán v kapitole 3.2.2. U každého bodu je nutné vyhodnotit, zda bod náleží či nenáleží trojúhelníku a na základě toho vygenerovat bod či nikoliv.

Uvedený algoritmus lze ve VHDL popsat několika způsoby, které se ve výsledku navzájem liší nejdelší cestou a použitou plochou, kterou tento blok v FPGA zabere. Způsob popisu algoritmu z velké míry závisí na způsobu průchodu prostorem. Algoritmus vyžaduje výpočet jednoho násobení a dvou odečtů pro všechny hrany trojúhelníku. Tyto hodnoty udávají šest koeficientů, které jsou po dobu rasterizace daného trojúhelníku konstantní.

Jakmile jsou vypočítány uvedené koeficienty A až C pro všechny tři strany trojúhelníka, je každý bod testován rovnicí (3.6) opět pro všechny tři strany trojúhelníku. Tato rovnice obsahuje dvě operace násobení a dvě operace sčítání.

Pro uvedené násobení je vhodné využít vestavěnou násobičku tzv. hard-makro. Tato násobička umožňuje násobit dvě 18-ti bitová čísla se znaménkem v dvojkovém doplňku. Hradlové pole platformy FITkit obsahuje čtyři vestavěné násobičky a čtyři blokové paměti RAMB16. Nevýhodou je, že každá dvojice násobička-sčítačka spolu sdílí určité vývody. Navržená architektura vyžaduje dvě fronty FIFO popsané v části 6.2, které zabírají dvě blokové paměti RAMB16. Pro blok rasterizace trojúhelníku lze tedy vyhradit maximálně dvě vestavěné násobičky.

Metoda se sdílením vestavěných násobiček

První možností je testovat jednotlivé body přesně podle vzorce (3.6). Toto lze realizovat násobením polohy testovaného bodu s vhodným koeficientem pomocí dvou volných vestavěných násobiček. Výsledek je možné jednoduše sečíst a získat tak v každém taktu ohodnocení daného bodu pro jednu rovnici s latencí jeden takt řídícího hodinového signálu. Násobičky je nutné sdílet pomocí multiplexorů. Multiplexory musí realizovat výběr z vysokého počtu hodnot – násobičky jsou nutné pro výpočet tří koeficientů A a dále pro tři rozdílné testované přímky. Složité multiplexory již na první pohled udávají, že výsledná nejdelší cesta bude pravděpodobně daná sdílením jednotlivých hodnot.

Metoda s využitím akumulátoru

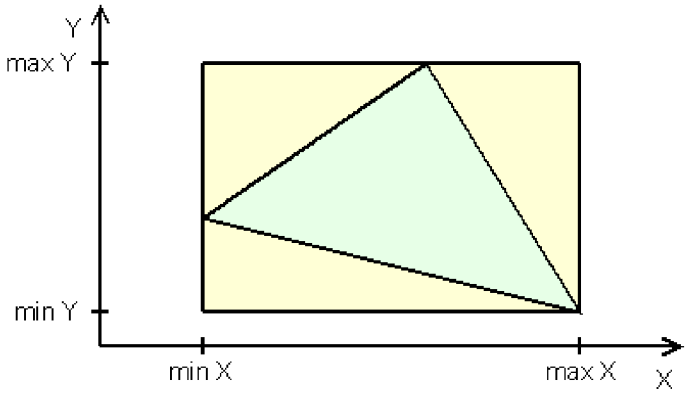
Druhou možností je sekvenční testování hodnot s využitím akumulátoru. Rovnice (3.6) udává, že koeficient A lze připočítávat k akumulátoru při každé změně souřadnice X. Podobně lze říci, že koeficient B lze postupně připočítávat k akumulátoru vždy s každou změnou souřadnice Y. Z této úvahy plyne, že je možné obě násobení nahradit přičítáním hodnoty k akumulátoru a to za předpokladu, že bude testovaný prostor procházen vhodným způsobem.

Vhodným způsobem průchodu se myslí například sekvenční průchod zleva doprava a postupně po řádcích shora dolů. Tímto lze eliminovat složitě sdílené vestavěné násobičky na dva méně sdílené bloky a to násobičku využitou při počátečním výpočtu krajního bodu a dvě sčítačky pro postupné přičítání A k akumulátoru hodnoty X a B k akumulátoru hodnoty Y. Tato možnost vyžaduje vyšší míru využití registrů, kterých je ovšem pro realizaci volných dostatečné množství.

Další výhodou tohoto přístupu je možnost vytvořit plně sekvenční architekturu. Je totiž možné vytvořit více sčítaček a otestovat tak určitý bod v jednom taktu řídícího hodinového signálu.

6.5.2 Modifikace průchodu prostorem

Způsob průchod prostoru, který potenciálně obsahuje rasterizovaný útvar, je hlavním omezujícím faktorem a má značný vliv na efektivitu zvoleného algoritmu. Jelikož je možné testovat body v jednom taktu metodou popsanou v 6.5.1, bylo by vhodné omezit počet testovaných bodů na minimum. Je-li například celková velikost prostoru 512×512 bodů, je nutné otestovat 262144 bodů. Za předpokladu, že je frekvence řídícího hodinového signálu 50 MHz, je jeden trojúhelník vyrasterizován za čas 5,24 ms. Tato hodnota udává maximální rychlost rasterizace 190 trojúhelníků za sekundu.

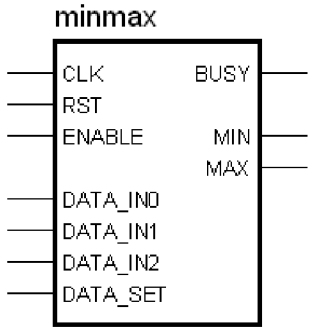


Obrázek 6.8: Limitace procházeného prostoru

Omezení prostoru lze jednoduše definovat jako dva intervaly – od nejmenší hodnoty X po nejvyšší hodnotu X a od nejmenší hodnoty Y po nejvyšší hodnotu Y z vrcholů $[X1, Y1]$, $[X2, Y2]$ a $[X3, Y3]$. Tedy nejmenší obdélník okolo výsledného trojúhelníku (viz obrázek 6.8). Za předpokladu, že je průměrný trojúhelník malý, například pokrývá plochu

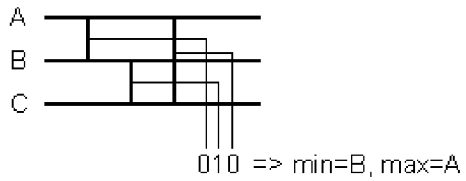
25x25 bodů, je počet testovaných bodů jen 625, což při výše uvedené frekvenci 50 MHz udává dobu testování celé plochy 12 μs a tedy celkový výkon 80000 trojúhelníků za sekundu.

Je více než patrné, že rozdíl efektivit daný různými průchody prostorem je propastný. Z tohoto důvodu byla navržena jednoduchá komponenta, která testuje vstupní hodnoty a hledá minimální a maximální hodnoty X a Y. Rozhraní této komponenty je uvedeno na obrázku 6.9. Vstupními hodnotami jsou tři čísla, kterým odpovídají signály DATA_IN0, DATA_IN1 a DATA_IN2. Výpočet začíná s nástupnou hranou, na které je hodnota signálu DATA_SET v logické 1.



Obrázek 6.9: Rozhraní komponenty výběru minima a maxima

Struktura této komponenty by mohla odpovídat řadicí sítí a nebo její částí. Jelikož však výsledkem komponenty má být pouze minimální a maximální hodnota ze tří vstupních hodnot, byl navržen postup, který vyžaduje pouze tři porovnání pro dosažení výsledku. Řadicí síť, která umí pracovat zřetězeně, vyžaduje v každém kroku tolik registrů, kolik je řazených hodnot. V případě platformy FITkit se tento přístup jeví jako nevhodný, jelikož vyžaduje příliš mnoho zdrojů. Tento blok má být vložen do místa, kde se předpočítávají hodnoty pro rasterizace trojúhelníku a po dobu rasterizace se nemění. Není proto důvod tvořit plně zřetězenou architekturu. Porovnání, které je nutné provést, je uvedeno na obrázku 6.10. Výsledky jednotlivých srovnání jsou uloženy do registru, ve kterém každý bit odpovídá jednomu porovnání.



Obrázek 6.10: Struktura vnitřních porovnání

Experimentální výsledky ukázaly, že s ohledem na výslednou plochu je vhodné do jisté míry sdílet funkční bloky (sčítačky, násobičky, porovnání, ...) pomocí multiplexorů. V tomto případě lze popsat vstupní signály pro porovnání vždy dvěmi různými hodnotami, a proto je vhodné popsat strukturu jediným funkčním blokem realizujícím porovnání. Tabulka 6.1 ukazuje vyhodnocení výsledků výsledného vnitřního registru MINMAX_REG. Výsledná architektura této komponenty zabírá poměrně málo prostředků. Její vložení před rasterizační

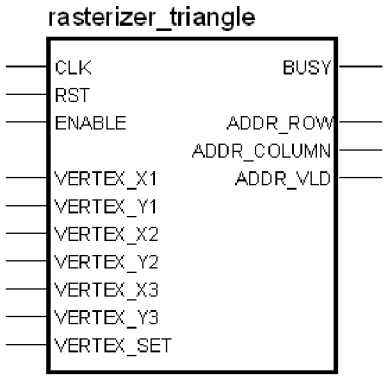
blok ovšem zásadně ovlivňuje efektivitu navrženého rasterizačního bloku. O konkrétních výsledcích a srovnáních pojednává kapitola 7.2.

MINMAX_REG	MIN	MAX
000	VX2	VX0
010	VX1	VX0
011	VX1	VX2
100	VX2	VX1
101	VX0	VX1
111	VX0	VX2

Tabulka 6.1: Výběr výstupních hodnot z registru výsledků porovnání

6.5.3 Architektura bloku rasterizace trojúhelníku

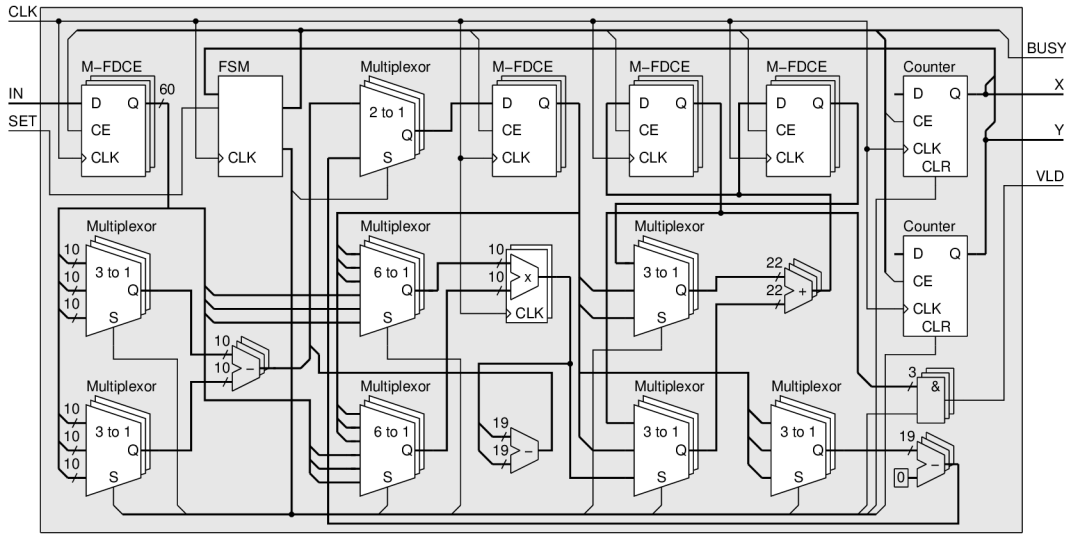
Z rozboru uvedeným v kapitolách 6.5.1 a 6.5.2 vyplývá, že pro efektivní implementaci bloku umožnující akceleraci rasterizace trojúhelníka (s důrazem na výslednou plochu) je vhodné využít algoritmus s třemi paralelními sčítačkami a limitací průchodu prostorem alespoň na nejmenší možný obdélník. Vnější navržené rozhraní tohoto bloku ukazuje obrázek 6.11. Z obrázku je patrné, že je vnější rozhraní velmi podobné rozhraní bloku rasterizace úsečky, které je popsáno v kapitole 6.4. Jeho vnější obsluha je tedy stejně jednoduchá jako v případě předchozího navrženého bloku a všechny důležité výpočty se provádějí uvnitř tohoto bloku architektury.



Obrázek 6.11: Rozhraní bloku rasterizace trojúhelníku

Podobně jako v případě ostatních bloků architektury provádějících určité zpracování dat, je výpočet započat s nástupnou hranou, kde je hodnota signálů VERTEX_SET v úrovni log. 1. Výpočet začíná přepočítáním hodnot koeficientů a jejich korekcí. Při výpočtu koeficientu A jsou využity vestavěné znaménkové násobičky 18x18 bitů. Koeficienty jsou uloženy v registrových polích. Paralelně s touto částí výpočtu lze v případě dostatečné kapacity FPGA provádět výpočet minim a maxim za využití navrženého bloku popsaného v kapitole 6.5.2. Výpočet koeficientů trvá 18 až 27 taktů řídícího hodinového signálu. Uvedený rozptyl je dán nutností korigovat některé vypočítané koeficienty. Jakmile jsou koeficienty

rasterizace známy, přechází řídící automat do stavu, ve kterém probíhá akumulace hodnot a tím i vyhodnocování jednotlivých bodů obrazu. Vnitřní strukturu znázorňuje obrázek 6.12.



Obrázek 6.12: Vnitřní struktura bloku rasterizace trojúhelníku

Navržená architektura pro rasterizaci trojúhelníku obsahuje po syntéze 309 registrů a 854 LUT. Maximální pracovní frekvence tohoto bloku daná nejdelší cestou je 85 MHz. V případě, že je architektura obohacena o limitaci procházeného prostoru, zvýší se počet registrů na 316 a počet LUT na 925.

Kapitola 7

Experimentální výsledky

Obecnou architekturu složenou z bloků uvedených v předchozí kapitole, lze jednoduše sestavit, provést syntézu a její funkčnost otestovat prakticky. Protože blok rasterizace trojúhelníku zabírá větší plochu obvodu a pomocí trojúhelníku lze přibližně rasterizovat i úsečku, byly praktické experimenty rozděleny na hodnocení dvou různých architektur. První část, kapitola 7.1, samostatně pojednává o rasterizaci úseček. Druhá část, kapitola 7.2, se zabývá vyhodnocením architektury obvodu rasterizace trojúhelníku.

7.1 Rasterizace čar

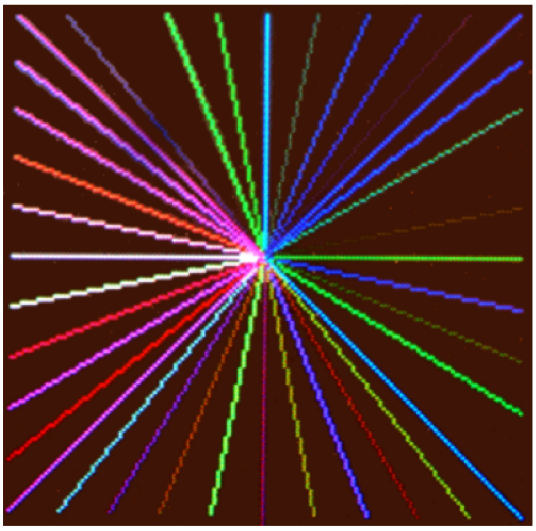
Dle obrázku 6.1 byla z popsaných komponent vytvořena architektura zahrnující blok rasterizace čar. Výsledky syntézy pro FPGA čip dostupný na platformě FITkit ukázaly, že počet potřebných registrů je roven 396 a počet využitých čtyřvstupových look-up tabulek je 635. Celkový počet zabraných slice FPGA obvodu je 468, což odpovídá 60 % FPGA. Maximální frekvence, na které je tento obvod schopen korektně pracovat, je 54.445 MHz. Doba potřebná pro rasterizaci úsečky pomocí navržené architektury závisí na délce úsečky, jejím směru a dále na frekvenci řídícího hodinového signálu.

Řídící obvod, tedy mikrokontrolér, provádí program nezávisle na obvodu FPGA. Využití mikrokontroléru tímto kleslo jen na minimální dobu a to odesílání řídících dat do obvodu FPGA. Ušetřený čas lze využít například ke zpracování dat či řízení periférií. Grafické výsledky praktické realizace ukazuje obrázek 7.1.

7.1.1 Porovnání s čistě softwarovým řešením

Čistě softwarové řešení vyžaduje použít určitý sled instrukcí. Každá instrukce trvá u procesorů s redukovanou instrukční sadou (RISC) přibližně jeden takt hodinového signálu, kterým je daný mikrokontrolér řízen. Výpočet začíná v případě architektury FPGA i v případě softwarového řešení výpočtem koeficientů. Tuto dobu můžeme vzhledem k její složitosti zanedbat. Vlastní algoritmus je dán rovnicemi (3.4) a (3.4), které lze realizovat dle 7.1. Pokud by byl procesor schopen realizovat každou instrukci v jednom taktu, tak by dva takty trvala podmínka a volání na druhém řádku. Další takt (za ideálního předpokladu, že hodnoty budou v registrech) by vyžadovalo sčítání a jeden takt by trvala podmínka na řádku čtyři.

Toto je celkem sedm taktů za předpokladu, že podmínka na čtvrtém řádku není splněna. Pokud by byla tato podmínka splněna, bylo by nutné provést ještě další dvě operace – tedy dva takty. Vyhodnocení a přičtení X do dalšího kola smyčky vyžaduje jeden takt – pokud by byla využita instrukce LOOP. Softwarová realizace ideálně trvá osm až deset taktů dle podmínky.



Obrázek 7.1: Ukázka praktického experimentu

V případě navržené architektury je situace poněkud jiná. Podmínka na řádku dvě je realizována vystavením vhodné logické hodnoty na signál ADDR_VLD a paralelně s tímto probíhá vyhodnocení konce algoritmu, tedy $x = x1$ v uvedeném algoritmu 7.1. Třetí řádek trvá vždy jeden takt a ve stejném taktu je vyhodnocena podmínka na řádku čtyři. Pokud není podmínka na řádku čtři splněna, trvá jeden průchod cyklem dva takty. Pokud je podmínka splněna, je navíc proveden jeden takt, kde je paralelně přepočítána hodnota proměnných y a *error* uvedených na řádcích pět a šest. Navržená hardwarová realizace trvá dva až tři takty.

```
1       for x = x0 to x1
2          if (steep) setpixel(y, x) else setpixel(x, y)
3          error = error + deltay
4          if error > 0 then
5              y = y + ystep
6              error = error - deltax
```

Výpis 7.1: Algoritmus rasterizace úsečky

Z výše uvedených úvah je patrné, že rasterizace je v hardwarové realizaci zrychlena 3.3 až 4-krát. Přesná hodnota záleží na sklonu rasterizované přímky a tedy četnosti splnění podmínky na řádku čtyři.

7.2 Rasterizace trojúhelníků

Architektura bloku rasterizace trojúhelníku je implementována stejným způsobem jako architektura rasterizace čar 7.1. Jediným rozdílem je větší šířka vstupních dat. Při realizaci této architektury již nedostačovala kapacita dostupného obvodu FPGA platformy FITkit a architektura rasterizace trojúhelníku musela být prakticky testována bez bloku počítajícího limity procházeného prostoru 6.5.2. Tabulka 7.1 shrnuje výslednou plochu architektury a frekvenci řídícího hodinového signálu, na které lze tuto architekturu maximálně provozovat. Přestože nebylo možné prakticky vizuálně ověřit funkčnost architektury s limitací prostoru, byl blok podroben syntéze i pro jiné cílové obvody. Získané výsledky shrnuje tabulka 7.2

Typ	Registrů	LUT4	Slice	Max. frekvence[1]
Bez limitace prostoru	716	1182	766	63.508
S limitací prostoru	722	1260	796	-[2]

Tabulka 7.1: Výsledky syntézy architektury rasterizace trojúhelníku

Obvod	Registrů	LUT4	Slice	Max. frekvence[3]
XC3S200	719/3840	1312/3840	729/1920	62.543
XC3S250	719/4896	1299/4896	716/2448	65.125
XC2V250	719/3072	1331/3072	734/1536	79.460
XC2VP2	719/2816	1319/2816	727/1408	114.090

Tabulka 7.2: Výsledky syntézy architektury rasterizace trojúhelníka s limitací prostoru pro jiné FPGA obvody

Obrázek 7.2 ukazuje několik obrazovek, které byly získány z aplikace testující navrženou architekturu na platformě FITkit. Obrázky znázorňují barevnou plochu složenou z trojúhelníků. Výsledek byl zobrazen na monitor a vyfotografován.

7.2.1 Porovnání s čistě softwarovým řešením

Podobně jako blok rasterizace čar lze porovnat i blok rasterizace trojúhelníků. Několik taktů tvořících přípravu před vlastním rasterizačním cyklem opět zanedbáme. V případě softwarového řešení je příprava tvořena desítkami cyklů, kdežto v případě navržené architektury je to maximálně dvacetdva cyklů hodinového signálu. Vzhledem k celkovému počtu taktů je to ovšem zanedbatelná hodnota.

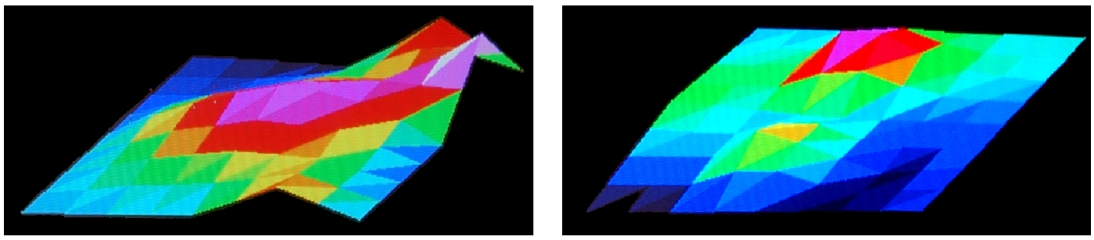
Vlastní algoritmus realizovaný metodou akumulace (viz kapitola 6.5.1) znázorňuje pseudokód 7.2. Jeho realizace je založena na minimalizaci počtu složitých operací. Metoda je založena na průchodu a testování všech bodů, které tvoří minimální obdélník okolo výsledného

[1] Maximální frekvence pro obvod XC3S50-4

[2] Nedostatečná kapacita obvodu

[3] Předpokládaná frekvence - jedná se pouze o výsledky syntézy, nikoliv place&route

trojúhelníku – tedy ekvivalent k architektuře obsahující minimalizaci průchodu prostorem (viz kapitola 6.5.2).



Obrázek 7.2: Ukázka několika obrazovek z testovací aplikace

V případě softwarového řešení jsou řádky dva až čtyři provedeny H-krát, kde $H = |y_0 - y_1| + 1$. Vnitřní podmínka na řádku šest je provedena P-krát, kde $P = H * W$ a $W = |x_0 - x_1| + 1$. Stejně tak vnitřní akumulace hodnot A_n se provádí P-krát. Pokud bude procesor pracovat ideálně jako RISC popisovaný v kapitole 7.1.1, budou řádky 2 až 4, 7 až 9 a 11 až 13 trvat minimálně jeden takt. Řádky 2 až 4 a 11 až 13 by potom vyžadovali $6 * N$ taktů řídícího hodinového signálu. Řádky ve vnitřním cyklu, tedy 7 až 9, jsou provedeny za $3 * M * N$ taktů. Podmínku je nutno rozložit na tři porovnání, kde každé z nich vyžaduje minimálně jeden takt. Podmínce tedy připadá $3 * M * N$ taktů. Celkový počet taktů, které trvá rasterizace trojúhelníku touto metodou v ideálním procesoru je $6 * (M * N + N)$.

```
1    for y = y0 to y1
2      Z1 = Z1a      // Z1a - na počátku hodnota C1
3      Z2 = Z2a      // Z2a - na počátku hodnota C2
4      Z3 = Z3a      // Z3a - na počátku hodnota C3
5      for x = x0 to x1
6        if ((Z1 >= 0) & (Z2 >= 0) & (Z3 > 0)) setpixel(x, y)
7        Z1 = Z1 + A1;
8        Z2 = Z2 + A2;
9        Z3 = Z3 + A3;
10
11     Z1a = Z1a + B1;
12     Z2a = Z2a + B2;
13     Z3a = Z3a + B3;
```

Výpis 7.2: Algoritmus rasterizace trojúhelníku

Navržená architektura pracuje plně paralelně. Vnitřní smyčku, kterou popisují řádky 5 až 9, realizuje navržená architektura během jednoho jediného taktu. To je tedy celkem $M * N$. Řádky 2 až 4 a 11 až 13 jsou realizovány také paralelně v jednom taktu. Celkový čas rasterizace tedy připadá době $(M * N + N)$. Z uvedené úvahy pro ideální využitý procesor plyne, že navržená architektura obvodu FPGA je 6-krát rychlejší.

V praktické realizaci této metody je nutno brát ohled na to, že ne všechny hodnoty jsou uloženy ve vnitřních registrech procesoru a je nutné pracovat s hodnotami v paměti. Instrukce, které umožňují akumulaci hodnot pouze na základě vstupních paměťových adres, nebývají ve většině případů k dispozici a čas nutný k provedený algoritmu prudce stoupá. Tato čísla nelze zanedbat, protože pro trojúhelník o velikosti minimálního obdélníku například 100 x 100 bodů se s každou instrukcí navíc prodlužuje čas minimálně o sto dalších taktů (v případě vnitřní smyčky dokonce o 10 000 taktů) řídícího hodinového signálu.

Kapitola 8

Závěr

V této práci byl popsán grafický řetězec a algoritmy, které mohou tvořit rasterizační blok grafického hardware založeného na hradlových polích FPGA. V praktické části práce popsala několik architektur realizujících rasterizaci v obvodu FPGA, pro které byly vytvořeny i vhodné aplikace, demonstrující funkčnost.

Navržené architektury jednotlivých bloků a z nich vytvořené demonstrační aplikací ukazují, jak lze prakticky realizovat grafický řetězec s využitím obvodu FPGA. Porovnání navržených architektur se softwatovým řešením ukázalo, že navržené architektury vykazují několikanásobně vyšší výkon než tradiční procesory, které by vykonávaly stejný algoritmus. V případě rasterizace čar je to v popsaném případě 3.3 až 4-krát. Tento rozsah je dán realizací vybraného algoritmu a přímo jej udává sklon výsledné přímky. Architektura bloku rasterizace trojúhelníku je vzhledem k danému algoritmu urychlena v obvodu FPGA 6-krát. Obě porovnání jsou vzhledem k určitému typu procesoru, jehož vlastnosti jsou popsány v příslušných kapitolách. V běžné praxi může být urychlení více násobné, protože procesory běžně zpracovávají více instrukcí, než je uvažováno.

Architekturu je možné dále rozšiřovat o vhodné grafické doplňky. Zajímavým doplňkovým blokem navržené architektury by mohl být blok interpolace barev. Tento blok by sloužil k rasterizaci výsledných trojúhelníků s interpolovanou barvou. Metoda interpolace trojúhelníka je v tomto textu teoreticky popsána. Kapacita obvodu FPGA platformy FITkit je ovšem pro takovou architekturu již nedostatečná a bylo by nutné při realizaci využít alespoň nejbližší vyšší model FPGA obvodu. Tímto modelem může být například Spartan XC3S200 Xilinx Inc., jež je přímým následníkem v řadě Spartan.

Dodatek A

Manuál

Cílem práce bylo navrhnout, realizovat a prakticky demonstrovat funkčnost rasterizačního bloku v obvodu FPGA. Výsledkem praktické části jsou dvě aplikace. Obě aplikace se skládají z programu pro řídící mikrokontroler MCU a z architektury pro obvod FPGA. Aplikace jsou přímo určeny pro platformu FITkit, jednotlivé navržené řadiče a bloky ovšem nejsou platformě závislé a lze je využít v libovolné jiné aplikace a na jiném FPGA obvodu.

Ke zprovoznění jednotlivých aplikací pro platformu FITkit je nutné mít k dispozici následující programy:

- GCC toolchain for MSP430 (dostupný na adrese http://mspgcc.sourceforge.net - březen 2007),
- Xilinx ISE WebPack (dostupný po registraci na http://www.xilinx.com) či jiný podobný nástroj pro vytvoření konfiguračního řetězce obvodu FPGA,
- terminálový program. Dostačuje i Windows Hyperterminál.

Zprovoznění probíhá v několika krocích, které jsou společné pro obě aplikace. Obě aplikace rozlišuje jen vybraný adresář, který je v následující postupu vždy uveden.

1. Přeložení aplikace příkazem make v adresáři „/Source/app/vga_line_rasterizer/sw“ (resp. „/Source/app/vga_triangle_rasterizer/sw“).
2. Vytvoření konfiguračního řetězce pro FPGA příkazem make v adresáři „/Source/app/vga_line_rasterizer/top“ (resp. „/Source/app/vga_triangle_rasterizer/top“).
3. Nahrání programu do mikrokontroléru FITkitu příkazem make s parametrem load v adresáři „/Source/app/vga_line_rasterizer/sw“
   (resp. „/Source/app/vga_triangle_rasterizer/sw“).
4. Propojení propojky J6 na platformě FITkit v případě, že není propojena.
5. Připojení platformy FITkit k USB.
6. Spuštění terminálového programu a navázání spojení s konfigurací:
   Rychlost 460800 Bd, osm datových bitů, bez parity, jeden stopbit, DTR a RST zapnuto.

7. Zadat do terminálu příkaz „flash w fpga“.

8. Pomocí protokolu XMODEM-1KCRC nahrát soubor konfigurace do FPGA (typicky soubor output.bin v podadresáři „top“ aplikace).

9. Provést reset mikrokontroléru příkazem „reset mcu“, po kterém bude provedena konfigurace FPGA obvodu a spuštěna demostrační aplikace.

10. Příkazy, které generují demostrační obrázek jsou:
„gl“ pro aplikaci rasterizace čar, „gt“ a „load“ pro aplikaci rasterizace trojúhelníků.

Po zadání příkazu „gl“ je na připojenou obrazovku vykreslena úsečka mezi dvěma náhodnými body. Po zadání příkazu „gt“ druhé aplikace je na připojenou obrazovku vykreslen náhodný trojúhelník. mezi třemi body. Příkaz „load“ odešle do obvodu FPGA definice vrcholů trojúhelníků, tvořících jednoduchý barevný výškový graf (viz obrázky 7.2).

Literatura

[1] *GCC toolchain for the Texas Instruments MSP430 MCUs* [online]. [cit. květen 2007].

[2] *SDRAM GM72V66841ET datasheet* [online]. [cit. květen 2007].

[3] *Výuková platforma FITkit* [online]. [cit. květen 2007].

[4] *Xilinx Inc.* [online]. [cit. květen 2007].

[5] Foley, J., van Dam, A., Feiner, S. K., aj.: *Computer Graphics. Principles and Practice. 2nd Edition in C.* Addison-Wesley, 1996, ISBN 0-201-84840-6.

[6] Gribbon, T., K., Bailey, G., D.: *A Novel Approach to Real-time Bilinear Interpolation.* IEEE computer society, DELTA, 2004, ISBN 0-7695-2081-2.

[7] Macri, D.: *Using NURBS Surfaces in Real-time Applications* [online]. [cit. květen 2007].

[8] Molnar, S., Cox, M., Ellsworth, D., aj.: *A Sorting Classification of Parallel Rendering.* IEEE Comput. Graph. Appl., 1994.
Dostupné z: `<citeseer.ist.psu.edu/article/molnar94sorting.html>`

[9] Nuño-Maganda, M., Aras-Estrada, M.: *Real-time FPGA-based Architecture for Bicubic Interpolation: An Application for Digital Image Scaling.* IEEE computer society, ReConFig, 2005, ISBN 0-7695-2456-7.

[10] Piegl, L., Tiller, W.: *The Nurbs Book.* Springer, 1995, ISBN 3-540-61545-8.

[11] del Rio, A., Boo, M., Amor, M., aj.: *Hardware Implementation of the Subdivision Loop Algorithm.* IEEE computer society, EUROMICRO, 2002.
Dostupné z: `<citeseer.ist.psu.edu/delrio02hardware.html>`