VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

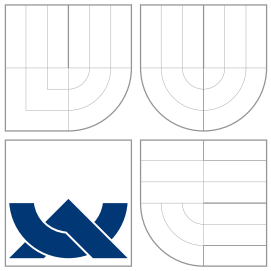
ZPRACOVÁNÍ OBRAZU V FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

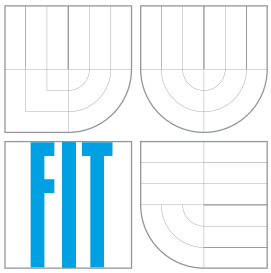
AUTOR PRÁCE
AUTHOR

LUKÁŠ MARŠÍK

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZPRACOVÁNÍ OBRAZU V FPGA

IMAGE PROCESSING IN FPGA

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ MARŠÍK
AUTHOR

VEDOUCÍ PRÁCE — Doc. Dr. Ing. PAVEL ZEMČÍK
SUPERVISOR

BRNO 2008

Abstrakt

Tato bakalářská práce pojednává o hardwarové realizaci grafického algoritmu pro vykreslování objektů popsaných pomocí *3D point clouds* – reprezentace prostorových objektů. Základ pro implementaci funkčních jednotek tvoří *FPGA* (*Field-Programmable Gate Array*) párované s *DSP* (*Digital Signal Processor*). Využitím více párů a s tím spojenou distribucí zátěže vzniká zajímavá možnost zrychlování výpočtů. Vstupními daty jsou takzvané *3D point clouds*, neboli množiny bodů, které jsou pro účel vykreslení převedeny na orientované kružnice promítnuté do *2D* – elipsy. Jako grafická reprezentace se jeví pro spoustu účelů mnohem použitelněji, než nejběžněji používané sítě trojúhelníků. Popsána je i samotná implementace odpovídající návrhu systému.

Klíčová slova

point clouds, rendering, *FPGA*, hardwarová akcelerace, paralelní zpracování

Abstract

This bachelor's thesis presents a hardware realization of graphic algorithm for rendering objects described with *3D point clouds* – a spatial objects representation. An *FPGA* (*Field-Programmable Gate Array*) chip coupled with a *DSP* (*Digital Signal Processor*) creates basement for implementation of function units. Is possible to decrease overall computation time by using more than one of that pair. That mean so simple distribution of load is used. The input graphical data is *3D point clouds* – sets of points which are transformed into oriented circles just for purpose of rendering. Result of projection of that elements are ellipses. Such graphical representation seems to be more suitable for many purposes than the most commonly used triangle meshes. The implementation equivalent to concept is described too.

Keywords

point clouds, rendering, *FPGA*, hardware acceleration, parallel processing

Citace

Lukáš Maršík: Zpracování obrazu v FPGA, bakalářská práce, Brno, FIT VUT v Brně, 2008

Zpracování obrazu v FPGA

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Dr. Ing. Pavla Zemčíka. Další informace mi poskytli Ing. Martin Žádník, Ing. Adam Herout, Ph.D. a Bc. Jiří Havel. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Lukáš Maršík
13. května 2008

Poděkování

Velký dík patří mému vedoucímu práce za obětavost a trpělivost s jakou mi předával všechny potřebné informace a s jakou se dělil o dosavadní poznatky ohledně dané problematiky. Taktéž bych rád poděkoval všem těm, kteří mi poskytli odbornou pomoc a našli si na mě čas ve svém jistě nabitém programu.

Obsah

Kapitola 1

Úvod

V dnešní době se člověk setkává s počítači téměř všude a spousta lidí si život bez nich prakticky nedovede představit. Hlavním důvodem je, že s jejich pomocí lze řešit spoustu úkonů mnohem efektivněji a rychleji. Neustále se vyvíjející technické vybavení umožňuje psaní náročnějších programů, jejichž výstupy jsou o to kvalitnější. Tento trend lze nejlépe pozorovat u počítačové grafiky, konkrétně u zobrazování prostorových scén.

Na počátku éry osobních počítačů sestávaly konfigurace z početně málo výkonných procesorů (*Central Processing Unit – CPU*) doplněných pamětí s malou kapacitou a propustností. Ani monochromatické monitory s nízkým rozlišením *3D* grafice příliš neprospěly. S postupným zlepšováním technologií a s výrobou vyspělejších výkonnějších komponentů nebyl problém řešit pomocí procesoru klasické operace *3D* grafiky, jako jsou transformace, projekce a další. Avšak všechny tyto výpočty *CPU* velice zdržovaly od provádění ostatních operací. Proto byl navržen koncept, kde se o samotnou scénu stará specializovaný procesor osazený na grafické kartě (*Graphics Processing Unit – GPU*). Procesor počítače tak již jen předává povely *GPU*, které řeší všechny úkony spojené s vykreslováním. Od té doby hraje specializovaný *hardware* v úlohách zobrazování *3D* modelů hlavní úlohu a především díky němu je možno generovat řádově desítky až stovky snímků scény za sekundu.

Všechny dnešní grafické procesory jsou striktně orientovány (co se týče hardwarové akcelerace) na polygonální modely. Jako popis prostorové scény ale existuje mnoho dalších reprezentací, které jsou pro jisté aplikace mnohem vhodnější. Jednou z nich je i *3D point clouds*. Tato reprezentace je charakteristická svým specifickým použitím a odlišným způsobem zpracování. Protože v dnešní době chybí podpora *point clouds* na úrovni technického vybavení, je hardwarová implementace grafických algoritmů pro vykreslování a manipulaci s takto reprezentovaným modelem logickým krokem. Výpočty pak lze snadno urychlovat vhodnou distribucí zátěže, paralelním zpracováním úloh, nebo většími a rychlejšími čipy, které vykonávají danou úlohu.

Zadáním projektu je tedy vytvořit aplikaci, která bude funkčnost hardwarové podpory *point clouds* včetně paralelního zpracování demonstrovat. Jedná se o systém sestavený z různorodých částí (software, *DSP*, *FPGA*, ...) jehož úkolem je vykreslit scénu popsanou pomocí *point clouds* a výsledek zobrazit na monitoru nebo *LCD*. Každá část systému se soustředí na vykonávání specifické operace která je jejímu účelu použití vlastní a v níž lze dané operace provádět co možná nejefektivněji.

Cíle projektu jsou následující. V první řadě je to sofistikované řešení systémové architektury s primárně snadnou modifikovatelností nejen pro konkrétní hardware (počet a velikost čipů), ale také pro konkrétní potřeby uživatele – to jest barevná hloubka, rozlišení scény a další. Podstatná je také možnost rozšíření návrhu o nové budoucí myšlenky bez

potřeby výrazně a složitě obměňovat návrh původní. Stěžejní pro celý projekt je akcelerace výpočtů využitím hardwarových čipů naprogramovaných "na míru" pro danou aplikaci a umožnujících těžit z výhod řešení pomocí *pipeline*. Lze také zkrátit celkovou dobu trvání operací (úměrné počtu funkčních jednotek) díky distribuci zátěže a z toho vyplývajícího paralelního zpracování dat. Vedlejším efektem tohoto projektu by mohlo být právě zviditelnění použití *3D point clouds* modelů. Určitě je v nich velký potenciál a jejich podpora v *GPU* by byla do budoucna více než zajímavou a užitečnou novinkou nejen pro aplikace typu *3D CAD*, ale třeba i pro *3D* počítačové hry.

Mračno bodů (v originále *point cloud*) je prakticky množinou vrcholů ve *3D* prostoru. Tyto vrcholy jsou obvykle definovány pomocí souřadnic *X*, *Y*, *Z*. *Point clouds* jsou produktem 3D skenerů, které snímají (proměřují) velké množství bodů na povrchu objektu. Výstupem je množina bodů (*point cloud*) jako datový soubor. Ten reprezentuje viditelný povrch objektu, který byl skenován a digitalizován. *Point clouds* lze využít pro spoustu účelů, včetně vytváření *3D CAD* modelů pro vyráběné součásti, metrologii nebo kvalitativní kontrolu, ale i pro velké množství vizualizací, animací, vykreslování a mnoho zákaznických aplikací. *Point clouds* obecně nejsou samy o sobě ve většině 3D aplikací použitelné přímo, a proto bývají převáděny na model reprezentovaný sítí trojúhelníků, *NURBS surface*, nebo *CAD* modely zpracovávané procesem běžně známým jako reverzní inženýrství. Díky tomu mají široké spektrum použití. Jedna z aplikací, ve které jsou *point clouds* použitelné bez jakýchkoli konverzí, je průmyslová metrologie nebo kontrola. *Point cloud* vyrobené součásti může být převeden na *CAD* model (nebo i na další *point cloud*) a porovnán pro zjištění odlišností. Tyto rozdíly mohou být zobrazeny v podobě barevné mapy, která poslouží jako indikátor odchylek mezi vyrobenou součástí a *CAD* modelem. Geometrické rozměry a tolerance lze z *point cloud* snadno vyčíst také. (Přeloženo z [6].)

Tato bakalářská práce je přehledně členěna do několika kapitol, přičemž jejich struktura a obsah je zhruba následující.

Kapitola 1 uvádí do problému počítačů a zobrazování obecně. Postupně se zaměřuje na zobrazování modelu popsaného pomocí množiny vrcholů (*point clouds*). Také pojednává o základních principech této množiny. Je zde prezentována motivace vedoucí k implementaci projektu a představeny jsou jeho hlavní cíle.

Obsah kapitoly 2 se týká principů renderingu *point clouds* modelů. Je zde krok za krokem popsán postup generování částic ze vstupních dat, tedy jejich promítnutí do *2D* soustavy souřadnic a současný výpočet barevného modelu. Následně je vysvětlena úloha bufferů scény při vykreslování modelu a zápis dat do paměti samotný.

Jelikož je hlavním pilířem projektu hardware, jsou v kapitole 3 stručně popsány charakteristické vlastnosti architektury jednotlivých prvků – *DSP* a *FPGA*. Zmíněna je jak jejich historie, tak parametry současných modelů na trhu hardwarových komponent.

V kapitole 4 je několik zmínek o původní implementaci projektu. Ta je stručně popsána a diskutovány jsou rozdíly vzhledem k nové architektuře. Rozebrány jsou i odlišnosti oproti řešení s využitím *GPU*. Jako závěr ze všech poznatků jsou prezentovány předpokládané výhody a nevýhody.

Kompletní popis architektury systému lze nalézt v kapitole 5. Jsou zde probrány úkoly jednotlivých hardwarových prvků v reálném procesu renderingu.

Kapitola 6 navazuje na kapitolu předchozí a podrobně vysvětluje funkci a řešení jednotlivých částí systému. Také jsou zde diskutovány programovací jazyky vhodné pro implementaci jednotlivých částí. Aby nedošlo k nedorozumění či nepochopení, je popis průběžně

doplňován souvisejícími obrázky a schematy.

Výsledky několika testů výkonnosti a funkčnosti, které by mohli napovědět co můžeme od systému očekávat, jsou shromážděny v kapitole 7.

Poslední závěrečná kapitola 8 shrnuje a diskutuje všechny dosažené cíle. Je zde vysvětleno splnění zadání, zmíněny jsou problémy které nastaly při řešení projektu. Systém je zhodnocen a doplněn je výhled do budoucna.

Kapitola 2

Rendering *point clouds* modelu

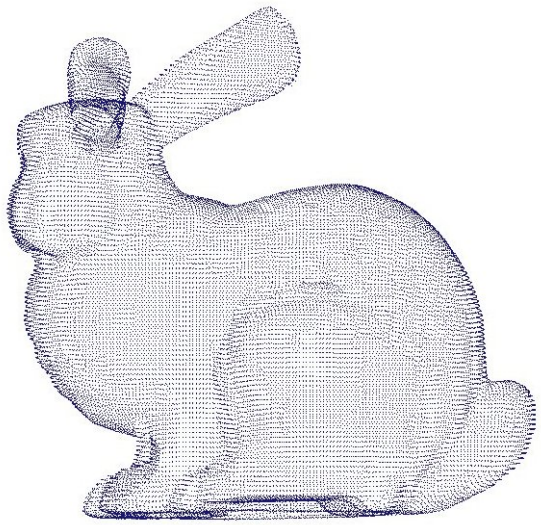
Promítnutí elementů *3D point clouds* modelu do *2D* soustavy souřadnic pro následný rendering je jedním ze složitější úkolů tohoto projektu. Velikou inspirací pro mne byl odborný článek publikovaný předními kapacitami našeho ústavu [1], který se danou problematikou zabývá.

2.1 Vstupní množina dat

Vstupem systému by měly být jednotlivé body modelu. Každý z bodů je definován souřadnicí a normálovým vektorem:

$$\text{Bod} = \{\text{střed}(X_0, Y_0, Z_0), \text{normála}(X_n, Y_n, Z_n)\}$$

Údaje o každém bodu lze doplnit o poloměr částice úměrný hustotě bodů v daném místě (viz níže). Pro účely vykreslování barevného modelu je více než žádoucí přiřadit pro každou částici materiál jejího povrchu, který bude použit pro výpočet *Phongova* osvětlovacího modelu (viz níže).



Obrázek 2.1: Příklad graficky znázorněné vstupní množiny bodů.

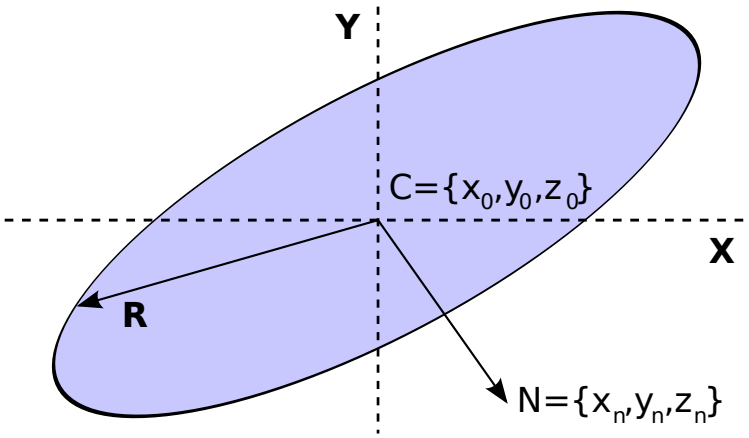
2.2 Zobrazování částic

Protože bod je charakteristický svojí nulovou plochou, je nutné jej převést na částici kterou již bude možno bez potíží vykreslit. V tomto případě se jedná o orientovaný kruh s vypočtenou barvou a poloměrem. Převod na tento typ částice probíhá v následujících krocích:

1. **Poloměr** kruhu je volen podle počtu bodů v jejím bezprostředním okolí. Jinými slovy, čím větší hustota bodů v daném místě, tím menší poloměr kruhů reprezentujících dané body. Totéž platí analogicky i naopak. Protože se jedná o neměnný údaj, lze jej zakomponovat do vstupní množiny bodů. To znamená, že tyto údaje jsou spočítány pouze jednou při generování vstupních dat a není třeba pro tento účel vyhrazovat prostředky ve vlastním procesu vykreslování modelu.

2. Jak již bylo zmíněno výše, každý ze vstupních bodů není charakterizován pouze svým středem, ale také normálovým vektorem určujícím **orientaci** kruhu. To je stěžejní pro výsledný tvar částice i její barvu. Nyní již může částice dostat svou finální geometrickou podobu. Tento proces sestává ze dvou následujících kroků:

   - Promítnutí natočeného kruhu do *2D* soustavy souřadnic. Výsledkem operace je elipsa. Příklad této projekce si lze prohlédnout na obrázku 2.2.
     Před promítnutím samotným je ještě žádoucí změnit poloměr kruhu v závislosti na jeho souřadnici *Z*. Díky tomu se bude její velikost s větší vzdáleností od pozorovatele přirozeně zmenšovat.



Obrázek 2.2: Nákres *2D* projekce částice se středem C, normálou N a poloměrem R

   - Výpočet barvy povrchu částice. Pro konkrétní normálový vektor, zadaný materiál částice a dané světelné zdroje je vypočten *Phongův* empirický osvětlovací model [3].
     Zde má každý světelný zdroj ovlivňující scénu definovány parametry i_d a i_s, což je intenzita (obvykle zadaná hodnotami RGB) difuzní a odrazové složky emitovaného světla. Jediný parametr i_a popisuje světlo přicházející z okolí. Prakticky se jedná o součet světelných přírůstků z okolních zdrojů – rozptýlené světlo, světelný šum.

Velmi důležitým pro výpočet *Phongova* modelu je materiál povrchu, na který světlo dopadá. Je definován těmito parametry:

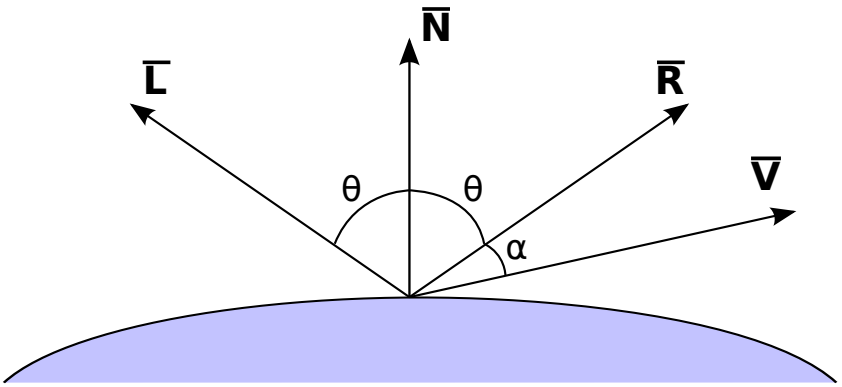
- k_a: konstanta definující poměr reflexe složky rozptýleného světla (*ambient*), tato složka je přítomná ve všech bodech renderované scény
- k_d: konstanta definující poměr reflexe difuzní složky dopadajícího světla (*diffuse*), jedná se o součást původního *Lambertova* osvětlovacího modelu
- k_s: konstanta definující poměr reflexe odrazové složky dopadajícího světla (*specular*)
- n_s: konstanta definující ostrost odlesku materiálu, rozhoduje jak rovnoměrně je světlo odraženo od lesklého bodu (to může vést až k zrcadlícímu povrchu)

Dále je pro *Phongův* model velmi důležitá poloha a natočení světelných zdrojů, povrchu objektu a pozorovatele vůči sobě. Na obrázku 2.3 je celá tato interakce znázorněna pomocí vektorů. $\vec{L}$ reprezentuje vektor světla dopadajícího pod úhlem θ na povrch s normálou $\vec{N}$. Podle zákonu odrazu je světlo odraženo pod stejným úhlem ve směru $\vec{R}$. Umístění pozorovatele (virtuální kamery) je naznačeno vektorem $\vec{V}$, který s paprskem odraženého světla svírá úhel α.

Aplikaci všech těchto parametrů jasně ukazuje rovnice pro výpočet *Phongova* osvětlovacího modelu 2.1. Lze si všimnout, že poloha kamery ($\vec{V}$) neovlivňuje výpočet přírůstku difuzní složky dopadajícího světla k celkové intenzitě světla odraženého směrem k pozorovateli. Je to proto, že odraz difuzní složky je konstantní ve všech směrech z bodu dopadu. Přírůstek odrazové složky je nezanedbatelný pouze v případě, že se směr odraženého světla ($\vec{R}$) blíží směru k pozorovateli ($\vec{V}$), což lze měřit jako kosinus úhlu α mezi nimi. Pokud je parametr n_s (označovaný také jako ostrost odlesku) reprezentován velkou hodnotou (znamená téměř zrcadlící odraz), pak pro jakýkoli paprsek světla odražený ve směru jiném, než přímo k pozorovateli, bude reflexe odrazové složky světla téměř nulová z důvodu velkého umocnění kosinu.

$$I_p = k_a i_a + \sum_{lights} \left(k_d (\vec{L} \cdot \vec{N}) i_d + k_s (\vec{R} \cdot \vec{V})^{n_s} i_s \right) \tag{2.1}$$

V případě barevné reprezentace hodnotami *RGB* je vhodné vypočítat rovnici 2.1 nezávisle pro složky *Red*, *Green* a *Blue*. (Přeloženo z [5])



Obrázek 2.3: Phongův osvětlovací model

3. Posledním krokem generování částice je její **rasterizace** pomocí vhodného algoritmu, neboli převod vektorové reprezentace na ekvivalentní rastrovou formu. Tu již lze pixel po pixelu zapsat do paměti a posléze vykreslit na rastrových zobrazovacích zařízeních, jako je například monitor, nebo *LCD*.

2.3 Vykreslování částic do bufferu

Pro korektní zápis částic do paměti jsou potřebné dva buffery:

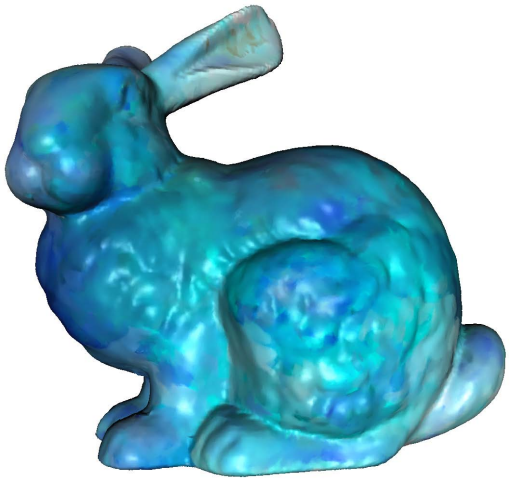
- Buffer pro renderovanou scénu - ***frame buffer***. Obsahuje informace o barvě každého z pixelů.
- Buffer hloubky – ***Z-buffer***. Obsahuje informaci o hloubce pixelu, který je pro příslušnou souřadnici zapsán v paměti renderované scény.

Jednotlivé částice jsou postupně zapisovány do bufferů. Samotný zápis probíhá pixel po pixelu a je potřeba jej řešit ve dvou krocích:

1. Nejprve je porovnána souřadnice Z zapisovaného bodu se souřadnicí zapsanou v bufferu hloubky.
2. Pokud není hloubka zapisovaného pixelu větší, je zapsána jeho barva do *frame bufferu* a zároveň dojde k aktualizaci příslušné hodnoty v bufferu hloubky.

Pokud by tedy byl pixel částice zakrytý jiným pixelem již vykresleného elementu, pak není vůbec zapisován. Tato technika je nazývána *Z-culling* [7] a výsledkem jejího použití je korektně a efektivně vykreslený model.

2.4 Zobrazení renderované scény



Obrázek 2.4: Příklad vykresleného *point cloud* modelu.

Výsledkem postupného provedení všech těchto kroků je *frame buffer* obsahující renderovaný model. Ten je již připraven pro zobrazení na na jakémkoli *2D* rastrovém zobrazovacím zařízení jako například monitor nebo *LCD*.

Kapitola 3

Hardwarové prvky

Nejefektivnější cesta ke zrychlení složitých algoritmů je jejich implementace do specializovaného hardware. Ze současné široké nabídky hardwarových prvků na trhu se pro konkrétní použití v tomto projektu se nejvíce hodí dva – *DSP* a *FPGA*.

3.1 DSP

Takzvaný signálový procesor (*Digital Signal Processor – DSP*) je specializovaným mikroprocesorem navrženým právě pro digitální zpracování signálů, zpravidla se jedná o *real-time* výpočty. (Obsah sekce přeložen z [4])

Typické vlastnosti

Signálové procesory jsou charakteristické následujícími vlastnostmi:

- Jsou navrženy pro *real-time* zpracování.
- Mají optimální výkon pro zpracování streamovaných dat.
- Jejich paměť pro program a data je oddělena.
- Podporují speciální instrukce pro *SIMD* (*Single Instruction, Multiple Data*) operace.
- Nemají hardwarovou podporu pro multitasking.
- Jsou schopné účastnit se komunikace jako *DMA* řadiče.
- Zpracovávají digitální signály převedené (pomocí *A-D* převodníku) z analogových. Výstupy jsou konvertovány *D-A* převodníkem zpět na analogový signál.

Rysy architektury

Digitalizované signály mohou být zpracovány i konvenčními procesory (*CPU*, ...). Avšak *DSP* je navrženo s potřebnými optimalizacemi přímo v architektuře, což ve výsledku znamená nejen rychlejší zpracování, ale i nižší cenu čipu, včetně nízkého zahřívání a spotřeby energie.

Typické rysy architektury signálových procesorů jsou zhruba následující:

- Co se týká **toku programu**, tak *floating-point* jednotka je integrována přímo do cesty toku dat. Architektura využívá řešení pomocí *pipeline* a v neposlední řadě obsahuje paralelní sčítačky a násobičky. Speciální hardware zajišťuje provádění cyklů a smyček s minimální režií.

- **Paměťová architektura** může být buď *Harvardská*, nebo modifikovaná *von Neumannova*. Specifická bývá svou schopností načíst více dat či instrukcí najednou. Neobvyklé není ani použití *DMA* řadiče pro přístup do paměti, nebo speciálního obvodu pro výpočet adresy.

- **Datové operace** jsou evidentně přizpůsobeny pro práci se signály. Je použita speciální aritmetika, kde operace s libovolnou hodnotou může vést až k maximu respektive minimu (saturaci), ale nikdy nedojde k přetečení respektive podtečení. Často je využívána *fixed-point* aritmetika pro urychlení zpracování. Provádění operací v jednom strojovém cyklu zvyšuje výhody řešení využitím *pipeline*.

- **Instrukční sada** se vyznačuje podporou operací pro sčítání matic urychlujících konvoluce (filtrování), skalární součin, ... Vyjímkou není použití instrukcí za účelem zlepšení paralelního zpracování jako jsou *SIMD*[1], *VLIW*[2] a superskalární architektura[3]. Používají se i instrukce pro modulo adresování v kruhových bufferech a bitově převrácený adresovací mód pro křížové odkazování při rychlé Fourierově transformaci (FFT). Lze se i setkat s časově neměnným kódováním za účelem zjednodušení hardware a zvýšení efektivity zakódování.

Historie

V minulosti se na trhu vystřídalo několik generací signálových procesorů. Mezi první lze zařadit například *Intel 2920* z roku 1978, který měl *D-A/A-D* převodníky integrované přímo na čipu. V roce 1979 firma *AMI* vypustila model *S2811* navržený jako periferie procesoru, ten ale nebyl příliš úspěšný.

S rokem 1979 se objevil první samostatný čip od *Bell Labs*, konkrétně *Mac 4 Microprocessor*. Tou dobou se začaly vyrábět procesory které využívaly výhod řešení využitím *pipeline*.

V roce 1983 přišla na trh s vlastním *DSP* i firma *Texas Instruments (TI)*. Bylo to *TMS32010* s Harvardskou architekturou a speciální instrukční sadou. 16-ti bitová čísla zpracovalo operací *MAC*[4] za $390\,ns$. *TI* se od té doby stal největším výrobcem v oblasti těchto specializovaných procesorů.

Po 5-ti letech, se s druhou generací objevily paměti pro uložení dvou operandů současně. Nový byl vestavěný hardware pro urychlování nekonečných smyček a nově se objevila i jednotka starající se o adresování v cyklu. Některé procesory měly 24 bitové proměnné a obvykle bylo potřeba $21\,ns$ pro operaci *MAC*. Typický představitel pro tuto generaci byly *AT&T DSP16A* nebo *Motorola DSP56001*.

Nejdůležitější vylepšení ve třetí generaci bylo nasazení aplikačně specifických jednotek (včetně instrukcí) do cesty toku dat, nebo někdy i jako koprocesory. Tyto jednotky umožňovaly přímou hardwarovou akceleraci velice specifických a komplexních matematických úloh.

[1] *Single Instruction, Multiple Data* – zpracování většího množství dat jednou instrukcí

[2] *Very Long Instruction Word* – instrukční slovo v sobě nese více primitivních instrukcí, které se vykonají paralelně

[3] provádí se více než jedna instrukce během jednoho strojového cyklu

[4] *Multiply-ACcumulate* – maticová operace

Například se jednalo o *Fourierovu transformaci* nebo maticové operace. Některé čipy, jako například *Motorola MC68356*, byly složeny z více jader, která pak pracovala nezávisle. Další konkrétní modely z roku 1995 jsou *TI TMS320C541* nebo *TMS 320C80*.

Čtvrtou generaci lze nejlépe charakterizovat změnami v instrukčních sadách a v jejich kódování a dekódování. *SIMD* a *MMX*[5] rozšíření byla zakomponována do návrhu. Objevila se i *VLIW* a superskalární architektura. Jako vždy se zvýšila rychlost, $3\,ns$ na operaci *MAC* nyní není problém.

Současnost

Dnešní signálové procesory poskytují mnohem větší výkon. Je to důsledek vylepšení v architekturách a technologiích jako je širší sběrnice, rychlá dvouúrovňová vyrovnávací paměť, řadič *(E)DMA* a další. Pochopitelně ne všechna *DSP* poskytují stejný výkon. Existuje spousta druhů, z nichž každý se hodí pro jinou specifickou úlohu. Jejich ceny se pohybují mezi $1,50$\$ až 300\$. Výrobců je celá řada (*Freescale*, *Analog Devices*), přičemž nejznámější *Texas Instruments* má na trhu spoustu úspěšných modelů, z nichž například série *C6000* je taktována na $1.2\,GHz$, má oddělené vyrovnávací paměti pro instrukce a data (8MB v druhé úrovni) a jeho vstup–výstupní rychlost je obrovská díky čtyřiašedesáti *(E)DMA* kanálům. Vrcholné řady mají výkon až 8000 *MFLOPS*[6], používají *VLIW* kódování umožňující 8 operací během strojového cyklu. Většina sérií je navíc kompatibilní s velkým množstvím periferií a sběrnic.

Spousta modelů signálových procesorů používá aritmetiku s pevnou řádovou čárkou, protože při zpracování reálných signálů není potřebný tak velký rozsah jako poskytuje čárka plovoucí. Díky zjednodušení hardware se zvýší jeho výpočetní výkon a sníží složitost. Naopak *DSP* počítající v plovoucí řádové čárce jsou nepostradatelná v aplikacích, kde je obrovský dynamický rozsah hodnot potřebný. Vývojáři je také mohou využít k snížení ceny a složitosti software na úkor ceny hardware, protože algoritmy jsou obecně lépe implementovatelné ve *floating point* aritmetice.

Konvenční *CPU* si vypůjčily od signálových procesorů některé z myšlenek, jako je například rozšíření instrukční sady *MMX* nebo *SSE*[7]. Tyto koncepty se pak objevily v instrukční sadě architektury společnosti *Intel* známé jako *IA-32*.

Obecně jsou *DSP* velmi složité a dedikované integrované obvody, nicméně jejich funkcionalita může být realizována v čipech *FPGA*.

3.2 FPGA

FPGA (Field-Programmable Gate Array) [8] je polovodičový obvod obsahující programovatelné komponenty zvané "logické bloky" a jejich konfigurovatelné propojení. Každý ze základních prvků může být naprogramován na provádění základních logických funkcí jako *AND* a *XOR*, nebo z nich lze sestavit komplexnější matematické funkce a dekodéry. Ve většině *FPGA* jsou do logických bloků vloženy i paměťové prvky, které mohou zastávat funkci jako obyčejný klopný obvod, nebo je z nich možno vystavět paměť.

[5] *MultiMedia eXtension* – obdoba *SIMD*

[6] *Milion FLoationg point Operations Per Second* – počet milionů operací v plovoucí řádové čárce za sekundu, jednotka pro měření výkonu

[7] *Stream SIMD Extension* – navazuje na rozšíření *MMX*

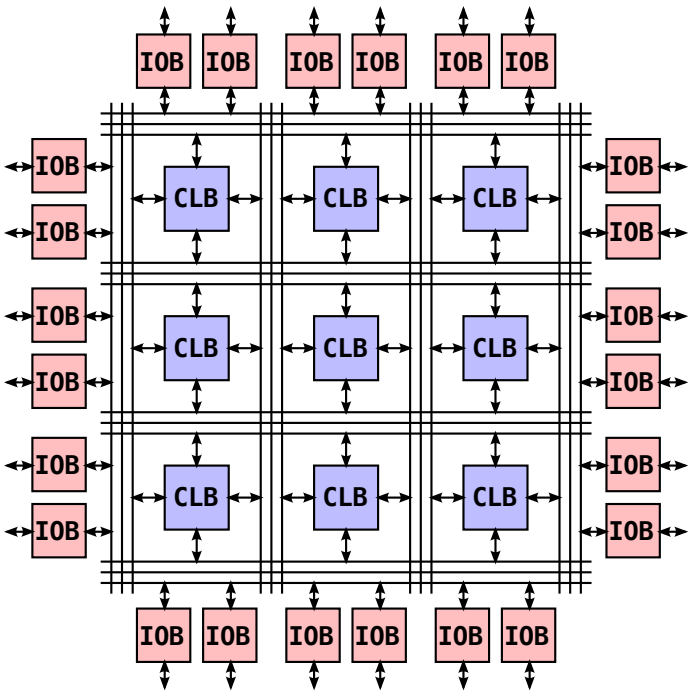
Typické vlastnosti

FPGA jsou charakteristická zhruba následujícími vlastnostmi:

- Hierarchie konfigurovatelného propojení umožňuje pospojování logických bloků podle potřeb návrháře.
- Jsou obvykle pomalejší než aplikačně specifické integrované obvody (*ASIC*), nemohou pojmout tak komplexní návrh a mají i větší spotřebu energie.
- Na rozdíl od *ASIC* lze vnitřní strukturu změnit přeprogramováním. Vhodné pro ladění a výzkumné účely.
- Jejich programování probíhá na základě popisu specializovanými *HDL* jazyky.
- Pomocí specializovaného software je zdrojový kód převeden (syntetizován) na soubor, který je do *EEPROM* nahrán – dochází ke konfiguraci *FPGA*.

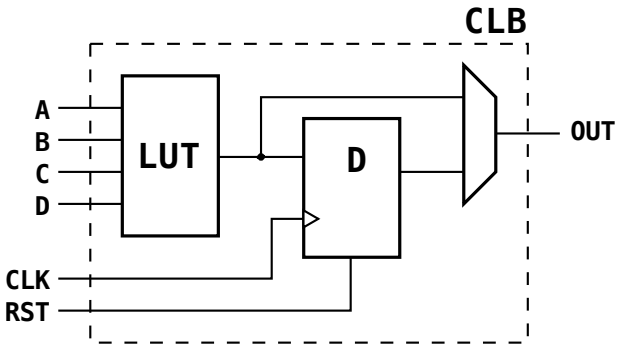
Rysy architektury

Současné největší obvody *FPGA* obsahují až 6 milionů ekvivalentních hradel (typické dvouvstupové hradlo *NAND*). Obvyklou strukturu *FPGA* čipu znázorňuje obrázek 3.1.



Obrázek 3.1: Vnitřní struktura *FPGA* čipu.

Bloky *IOB* (*Input/Output Block*) jsou vstup-výstupními obvody pro každý pin pouzdra čipu. Tyto bloky obvykle obsahují registr, budič, multiplexer a ochranné obvody. Bloky *CLB* (*Configurable Logic Block*) představují vlastní programovatelné logické bloky, které mohou být mezi sebou libovolně propojeny (naprogramováním) globální propojovací maticí. Nejpoužívanější struktura konfigurovatelného logického bloku je znázorněna na obrázku 3.2.



Obrázek 3.2: Typická struktura *CLB*.

FPGA obvykle umožňují propojit některé signály logických bloků přímo se sousedním bez nutnosti využívat globální propojovací matici. Takovéto spoje mají mnohem menší zpoždění a umožňují tak realizovat například rychlé obvody šíření přenosu, což je nezbytné pro sčítačky nebo násobičky.

Kromě bloků znázorněných na předchozích obrázcích integrují výrobci do *FPGA* další prvky. Většina moderních čipů obsahuje několik bloků rychlé synchronní statické paměti *RAM*. Velmi často obvody obsahují *PLL* (*Phase Locked Loop*) nebo *DLL* (*Delay Locked Loop*) pro obnovení charakteristik hodinového signálu, případně pro násobení nebo dělení jeho frekvence. Nejnovější hradlová pole často obsahují bloky vhodné pro vytváření složitých systémů pro číslicové zpracování signálů jako jsou například hardwarové násobičky nebo dokonce mikroprocesory.

Historie

Historické kořeny *FPGA* lze nalézt v *CPLD* (Complex Programmable Logic Device) z poloviny osmdesátých let. Spoluzakladatel společnosti *Xilinx*, Ross Freeman vyvinul programovatelné hradlové pole v roce 1984. Prakticky první *FPGA*, které se objevilo na trhu, byl roku 1985 model *XC2064*. Později se nová řada čipů *XC4000* z roku 1991 se stala první opravdu rozšířenou a přinesla firmě *Xilinx* zisk z jejich prodeje. Rodina čipů *Virtex* (rok 1998) byl jedním z největších pokroků v oblasti architektury *FPGA*. Dalším milníkem bylo představení architektury *Spartan-3* (rok 2003). Tento model se stal vůbec prvním levným *FPGA* čipem vyrobený 90 *nm* technologií.

Současnost

V současné době má firma *Xilinx* na trhu modely řady *Virtex-II*, které jsou vyrobeny nejmodernějšími technologiemi. V závislosti na velikosti čipu (nejmenší *XCV40*, největší *XCV8000*) má tato řada zhruba následující vlastnosti:

- počet systémových hradel v rozmezí 40 tisíc až 8 milionů, vnitřní hodiny taktovány na 420MHz, vstup-výstupní rychlost přenosu až 840MB/s
- výkonné rozhraní pro externí paměť (*DRAM*, *SRAM*, *CAM*)
- dedikovaná blok pro násobičku 18-bit × 18-bit, až 12 *DCM* modulů

- čtvrtá generace segmentové propojovací struktury, minimální zpoždění propojení nezávislé na větvení
- kompatibilita s *PCI* (33 a 66 MHz), *PCI-X* (66 a 133MHz) sběrnicemi (3.3V), využití diferenčního signálu (*LVDS*)

Kompletní výčet vlastností této řady lze najít v [8].

3.3 Využití kombinace *DSP* a *FPGA*

Kombinace dvou výše uvedený hardwarových komponent snadno najde uplatnění v některém z multimediální projektů, kde je potřeba hardwarově akcelerovat výpočty. Potom je nezbytné navrhnout speciální technické vybavení, na kterém aplikace poběží. Na ukázku poslouží obrázek hardwarového řešení společnosti *CAMEA*

Obrázek 3.3: Speciální hardwarová *PCI* deska firmy *CAMEA*.

Kapitola 4

Návrh zobrazovacího systému

Úkolem této práce bylo vytvoření systému pro rendering modelů popsaných pomocí *3D point clouds*. Pro řešení této netriviální úlohy se jako velice rozumné jevilo použití nápadů a technik z publikace [1], která se právě problematikou hardwarové akcelerace renderingu těchto specifických modelů zabývá. Jedná se o prakticky první funkční hardwarovou implementaci podobného systému na Ústavu Počítačové Grafiky a Multimédií FIT VUT v Brně. Postupem času a s příchodem nového technického vybavení však vznikl prostor pro další nápady a vylepšení, na něž především je tento projekt orientován.

4.1 Současný stav implementace

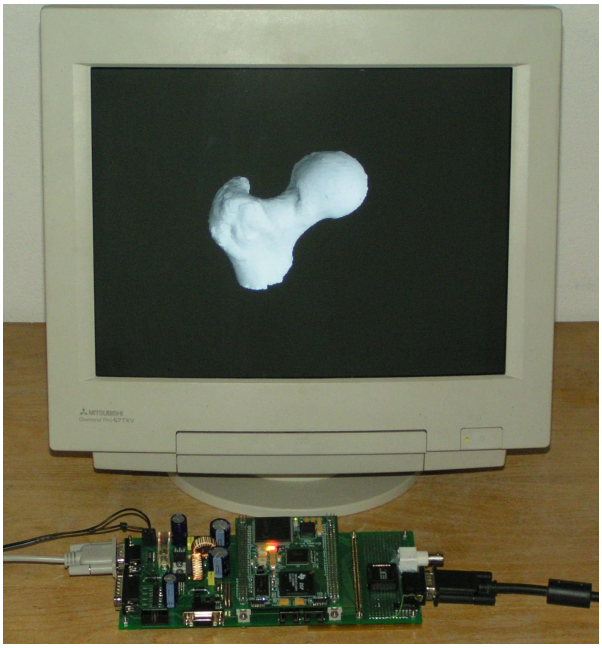
V současné době je na Ústavu Počítačové Grafiky a Multimédií v provozu speciální hardwarová implementace. Na rozdíl od té nové, pro kterou je připraven univerzální *PC* se speciálním hardware, stávající běží na unikátní platformě vyvinuté právě pro účel renderingu *3D point clouds* modelů.

Architektura systému sestává z paměti typu flash, signálového procesoru (*DSP*), *FPGA* čipu, *DMA* řadiče, *DRAM*, videopaměti a speciálního obvodu pro řízení zobrazování na *LCD*.

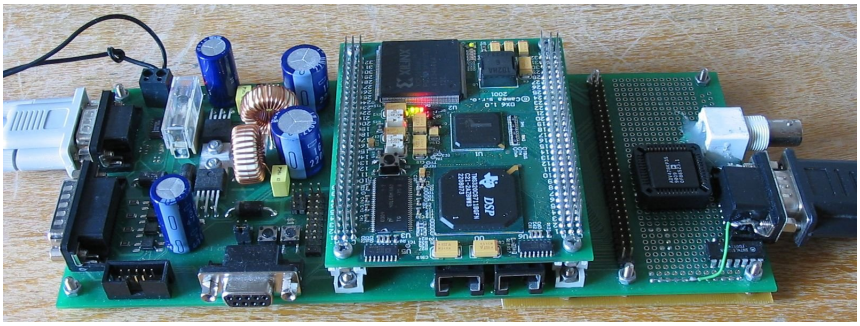
Flash paměť obsahuje informace o jednotlivých bodech. Ty jsou čteny pomocí *DSP*, které na jejich základě z tabulek naplněných dopředu vypočítanými a zakódovanými údaji (tvar, barva), vytvoří specifické kódové slovo. V *DRAM* jsou alokovány speciální buffery do nichž jsou jednotlivá slova ukládána v závislosti na souřadnici *Y*. Poté je naprogramován řadič *DMA*, který se postará o překopírování dat z *DRAM* do *FPGA*, v němž dochází k zapisování částic do *frame bufferu* a *Z-bufferu*. Díky vertikálnímu seřazení částic mohou být již neaktivní řádky kopírovány do videopaměti, odkud je speciální subsystém zobrazí na monitoru. Alokovaných bufferů je v *DRAM* několik skupin. Po startu systému se každá z nich naplní kódovými slovy transformovanými pro určitou polohu kamery. Signálový procesor pak již žádná data nezpracovává, ale pouze v cyklu dává povely řadiči *DMA*, který do *FPGA* přenáší obsahy jednotlivých částí paměti. To znatelně urychlí zobrazovanou animaci.

Co se týče výše zmiňovaného zobrazovacího subsystému, byl nejspíše tou nejsložitější a nejvíce problémovou částí (generovaní časování pro zobrazovací zařízení, ...). Problémů tohoto typu bude nový návrh ušetřen vzhledem k použití univerzálního *PC*, kde je k zobrazení scény na *LCD* využito služeb operačního systému. Díky tomu je bez problémů možné změřit se na dodržení hlavních cílů uvedených v úvodu.

Podrobnější popis experimentálního systému (obrázky 4.1 a 4.2) lze najít v [1].



Obrázek 4.1: Původní hardwarová implementace systému..



Obrázek 4.2: Hardwarová deska původního systému..

4.2 Odlišnosti proti původnímu návrhu

Protože nová implementace využívá stejnou základní myšlenku jako ta původní, je zřejmé, že si budou systémy v mnoha ohledech velmi podobné. Zajímavější bude, zaměřit se na hlavní rozdíly:

1. Hardware:
   - ∗ Nový systém poběží na speciální kartě vsazené do *PCI* slotu základní desky univerzálního *PC*. Karta samotná potom obsahuje konektory pro připojení modulů se specifickými komponenty.
   - † Původní systém potřeboval pro běh pouze samotnou speciální hardwarovou desku včetně složitých zobrazovacích obvodů.
2. Paralelní zpracování:

   * Specifický hardware je navržen tak, aby bylo možno použít více modulů obsahujících pár *DSP – FPGA* a paměť *DRAM*. Lze tak snadno zrychlit rendering pomocí distribuce zátěže mezi více modulů.
   † Hardwarová deska pro původní projekt žádné dodatečné úpravy neumožňuje. K dispozici je pouze jeden pár *DSP – FPGA* a bez nového designu a výroby desky není jakékoli paralelní zpracování (ve smyslu distribuce zátěže uvedené výše) možné.

3. Velikost částic:
   * Koncept nového systému počítá s teoreticky neomezenou velikostí částice která má být vykreslena.
   † Původní omezení je 8×8.

4. Barevný model:
   * Nově se počítá zjednodušený *Phongův* osvětlovací model pro všechny barevné složky – *Red*, *Green*, *Blue*. Součástí vstupních dat pro každý bod je i materiál povrchu částice.
   † Původně měly všechny částice stejný materiál a *Phongův* osvětlovací model byl vypočítáván pouze pro jasovou složku bílého světla. Výsledkem bylo zobrazení ve stupních šedi.

4.3 Porovnání s *GPU*

Za zmínku jistě stojí i srovnání s konvenčními grafickými procesory. Hned na první pohled se nabízí několik rozdílů:

- Vzhledem k podpoře jednoho konkrétního algoritmu má *point clouds* zobrazovací systém mnohem menší složitost.
- Použitím páru *DSP* a *FPGA* je spotřeba oproti *GPU* mnohem menší a to i při případném zapojení více jednotek paralelně.
- Nový systém lze díky důmyslnému konceptu provozovat na specializovaném hardware i bez *PC*.
- Výkon je zhruba srovnatelný, ale co se týče počtu částic za jednotku času je vítězem jasně nový zobrazovací systém.

4.4 Výhody a nevýhody nového konceptu

Z několika odlišností uvedených v sekci 4.2 vyplývají jak výhody, tak i nevýhody. Nejprve se zaměřme na výhody:

+ Díky možnosti propojení celého systému se software počítače se zjednodušuje spousta úkonů, jako je například zásobování systému vstupními daty, zobrazení renderované scény, ale i ladění systému samotného.

+ Pro další zrychlování výpočtů lze využít moduly s rychlejšími signálovými procesory nebo většími a rychlejšími *FPGA* čipy. Také s větším počtem modulů se čas potřebný pro vykreslení scény úměrně snižuje.

+ Zobrazování již není limitováno omezenou velikostí částic a v případě potřeby je bez problémů možno vykreslit částici větší než 8×8.

+ Dokonalejší a věrnější implementace osvětlovacího modelu a to včetně využití všech tří jasových složek *RGB* a materiálu povrchu modelu. Vykreslený model se pak zdá realističtější.

A nyní nevýhody:

– Dražší hardware z důvodu použití více signálových procesorů, ale především kvůli osazení většího počtu drahých *FPGA* čipů. Cena je úměrná počtu modulů pro paralelní zpracování.

Jak vidno proti novému návrhu hovoří pouze to, že nový speciální hardware je nákladnou záležitostí. Pokud ale vezmeme v úvahu, že koncepce nového hardware počítá s univerzálním použitím téměř pro jakýkoli multimediální projekt s požadavkem na akceleraci algoritmů, tak nový systém oproti původnímu prakticky žádné hendikepy nemá. Naopak výhod je dostatečné množství, aby stálo za to se tomuto problému důkladně věnovat.

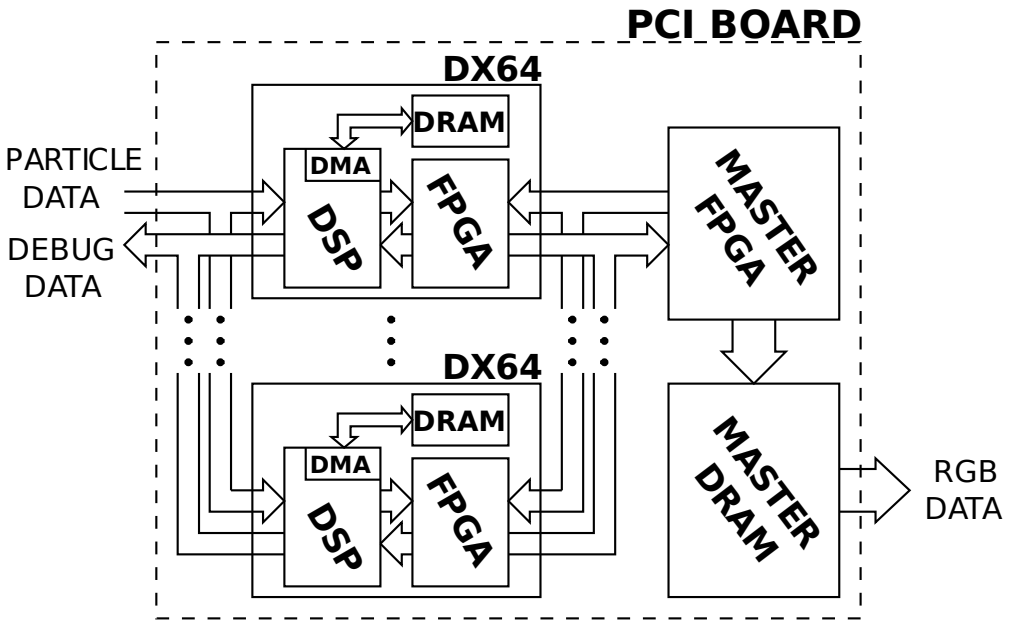
Kapitola 5

Systémová architektura

Základ architektury byl s drobnými změnami převzat ze systému původního. Stále se tedy jedná o dvojici *DSP* – *FPGA*, která se stará o běh algoritmů pro rendering. Kompletní výčet rozdílů mezi návrhem původním a novým lze najít v kapitole 4.2.

Každý z prvků hardwarové architektury se stará o určitou část procesu renderingu. Neboli má na starost provedení jisté sady úkonů, jež je prvkem samotným dostatečně efektivně proveditelná. Například relativně málo nákladné *DSP* se hravě zhostí konvenčních výpočtů a lineárních transformací, čímž jsou ušetřeny zdroje v *FPGA*, které mohou být použity za účelem implementace algoritmů pro rychlé paralelní operace. Tyto není prakticky možné v signálovém procesoru efektivně implementovat.

V následujících kapitolách lze nalézt popis funkce jednotlivých prvků hardware, přičemž jejich propojení zjednodušeně vystihuje obrázek 5.1. Popsány jsou jen relevantní prvky, ale pro běh systému je jich potřeba více (*CPLD* obvod pro *boot*, *FPGA* jako arbitr paměti, ...).



Obrázek 5.1: Zjednodušené schéma hardwarové architektury.

5.1 Software jako vstupní bod

O vstupní data systému se stará software – program používající pro zpřístupnění registrů *DSP* speciální funkce. Navíc se s jeho pomocí inicializuje speciální *PCI* karta, která nese všechny ostatní hardwarové prvky potřebné pro implementaci systému. Pomocí tohoto programu lze v neposlední řadě také vyčíst stavové registry z ostatních jednotek.

Hlavním úkolem, podstatným pro danou aplikaci, je především zásobování signálových procesorů vstupními daty – body popsanými středovou souřadnicí, normálou, velikostí a materiálem. V případě více než jednoho páru *DSP* – *FPGA* se software stará o vhodnou distribuci bodů mezi jednotlivé jednotky. Pomocí speciálních knihovních funkcí získává přístup do adresního prostoru, kde se nachází registry *DSP*. Zápisem a čtením určitých z nich komunikuje (pomocí *challenge-handshake*) se signálovým procesorem. Urychlení přenosu dat ze systémové paměti do *DRAM* signálního procesoru lze dosáhnout naprogramováním *busmaster* arbitra, který se o kopírování dat po sběrnici stará bez dalších zásahů software. Stejným způsobem je vhodné ošetřit i přenos renderované scény z hlavní *DRAM* paměti. Po skončení *busmaster* transakce má program data připravena v požadovaném paměťovém místě a může je zobrazit.

V procesu renderingu, tak jak byl teoreticky rozebrán, se software stará o množinu popsanou v sekci 2.1.

5.2 *DSP* – generátor kódů

Úlohou *DSP* je zajistit zásobování *FPGA* jasnými kódy nesoucími v sobě všechny potřebné informace o částici. Komunikace může probíhat přímo, pomocí 16-ti bitového rozhraní *EMIFB*, nebo naprogramováním řadiče *DMA*, kdy jsou data přenášena z *DRAM* bez účasti procesoru. Na počátku provádění instrukčního kódu si signálový procesor alokuje paměť pro tabulky kódových slov. Ty jsou naplněny předem vypočtenými hodnotami pro konkrétní normálový vektor a velikost částice. Dopředu vypočítán tak může být tvar (projekce kružnice a následně rasterizace elipsy) i parametry světelných zdrojů pro osvětlovací model částice.

Signálový procesor tedy zpracovává data , kterými je z vnějšku plněna paměť v jeho adresním prostoru. Vstupní data jsou uložena do paměti a nadále je *DSP* z vnějšku zásobováno pouze transformačními maticemi. Pro konkrétní parametry bodu (normála a velikost) je z obsahů tabulek složeno kódové slovo, které je posléze doplněno o další údaje (souřadnice částice a materiál). Zakódované částice jsou zapisovány do bufferů alokovaných v *DRAM* a rozdělených podle souřadnice *Y*. Po zpracování celé množiny dat je dán příkaz řadiči *DMA*, který zajistí odeslání kódů do další části systému – *FPGA*. Řadič již autonomně řídí celý přenos dat a signálový procesor se může věnovat zpracování množiny dat s ohledem na novou transformační matici (jedná se o pohyb kamery).

Úloha *DSP* v procesu renderingu odpovídá sekci 2.2.

5.3 *FPGA* a vykreslování do bufferů

Samotný rendering do bufferů je úkolem *FPGA*. Obsahuje tabulku materiálů, kterou je možné kdykoli naplnit potřebnými hodnotami. V případě více jednotek, kdy každá vykresluje určitou množinu náhodně rozdělených částic, je potřebné předávat si obsahy pamětí mezi sebou. Dochází tak vlastně k postupnému dokreslování scény – distribuci zátěže. Jako

prostředník slouží takzvaný *master FPGA* čip, jenž zajišťuje propojení sousedních jednotek mezi sebou.

FPGA přijímá sadu kódů na jejichž základě zapisuje barevná data do jednotlivých bufferů. Každá částice se rozkóduje a pro její povrch je dopočten zjednodušený osvětlovací model na základě předaných parametrů světelných zdrojů a indexu do tabulky materiálových konstant. Zápis samotný probíhá paralelně, takže se vždy zapisuje velké množství pixelů na jednou. V paměti hloubky dojde nejprve k vyřešení viditelnost pixelu, poté je případně jeho barva zapsána do *frame-bufferu* a zároveň dojde k aktualizaci hodnoty v *Z-bufferu*. Aktivních pro zápis je vždy 8 řádků (takzvaný pruh) a ostatní se kopírují (v originále *flush*) do *master FPGA*, které je odesílá na místo určení. Již zkopírované neaktivní řádky jsou připraveny pro zápis. Znovu se aktivují po určitém počtu posunutí aktivního pruhu v obraze.

V případě většího počtu jednotek nastává další problém v podobě předávání obsahu bufferů. Pro korektní vykreslení je třeba s pamětí scény předávat i paměť hloubky. První jednotka je specifická tím, že vždy inicializuje buffery daty z hlavní *DRAM* (načtení pozadí scény). S postupným posouváním aktivního pruhu se kopírují neaktivní řádky, a to včetně odpovídajícího řádku *Z-bufferu*, do dalšího *FPGA*. Jednotky vždy čekají na naplnění aktivního pruhu (8 řádků *frame-bufferu* i *Z-bufferu*) jednotkou předchozí a poté mohou začít s vykreslováním. Neaktivní řádky jsou kopírovány stejným způsobem jako u první jednotky a směřují vždy do následujícího *FPGA*.

V procesu renderingu se *FPGA* stará o část popsanou v sekci 2.3.

5.4 Propojení skrze *master FPGA*

Propojovací síť je implementována v hlavním čipu na *PCI* kartě, a sice v *master FPGA*.

Jeho úkolem je distribuce obsahů bufferů mezi jednotlivými jednotkami. Vždy příchozí řádky z první jednotky předává jako vstupní data jednotce druhé, výstup jednotky číslo dvě jednotce třetí a tak dále. První *FPGA* zásobuje pozadím scény a z posledního přijímá již kompletní renderovanou scénu včetně bufferu hloubky, který je pro další postup zbytečný. Je proto odstraněn a zbylá *RGB* data jsou zapsána do hlavní *DRAM* karty.

5.5 Zpracování výstupu v software

Druhým úkolem již zmiňovaného programu je přečtení renderované scény a její zobrazení na *LCD*.

RGB data jsou čtena z hlavní paměti *DRAM* speciální hardwarové karty a jsou dále zpracována v software tak, aby bylo možné je zobrazit (konverze bitové hloubky nebo barevného modelu). Poté je výsledek vykreslen pomocí služeb operační systému na obrazovku.

Tento proces odpovídá zobrazení renderované scény z kapitoly 2.4.

Kapitola 6

Implementace systému

Kapitola 5 stručně popisuje funkci jednotlivých částí hardwarové architektury systému. Úkolem kapitoly této je rozvinout a podrobně popsat jejich funkci a zaměřit se na implementaci řešení.

6.1 Software

Jak již bylo zmíněno, úkolem programu je především zásobování signálových procesorů vstupními daty – množina bodů, transformační matice. V konečné fázi renderingu se také stará o zobrazení dat výstupních. Software vlastně obsluhuje vstupní a výstupní body spojující vnitřnosti systému (speciální hardware) s běžným programátorským rozhraním

Implementace je vhodná prakticky v jakémkoli programovacím jazyce, který má snadnou vazbu na hardware. Ideální je jazyk *C++*, který po svém předchůdci (jazyk *C*) zdědil nízkoúrovňové vlastnosti ale zároveň lze program snadno vytvořit jako "klikací" okenní aplikaci, což je podstatné pro demonstrační účely a nezbytné pro zobrazování.

Distribuce zátěže

Kritická pro běh systému je hned na počátku distribuce vstupních dat mezi jednotlivé *DSP*.

Jako nejoptimálnější se jeví rozdělení vstupní množiny na tolik částí, jako je počet paralelních jednotek. Každá podmnožina by měla obsahovat pokud možno stejné množství bodů se shodnou souřadnicí *Y*. Tím by se zajistilo, že žádná z vykreslujících jednotek nebude nijak výrazně nečině čekat na předání bufferů z jednotky předchozí. Prakticky ale takovéto rozdělení není možné a to z důvodu pohybu kamery. Jinými slovy – body rotují v prostoru a nelze je předem podle žádného klíče rozdělit. Nejvhodnější je proto body distribuovat čistě náhodně pomocí *randomize* funkcí.

Za běhu systému již žádné problémy s distribucí dat nenastávají, protože pro jeden snímek scény je do všech *DSP* předávána vždy stejná transformační matice.

Zobrazení scény

Jako nejlepší a nejrychlejší řešení pro zobrazení výstupu se jevilo využití služeb operačního systému. Konkrétně se jedná o *DirectDraw* – součást *DirectX API* [2] vyvinutého společností *Microsoft*.

Důvodem volby je jednoduchost použití, kvalitní dokumentace a příklady na webu, ale především velká rychlost vykreslování díky přímému přístupu do videopaměti, což oceníme

při vysokých počtech snímků za sekundu. *DirectDraw* využívá hardwarovou akceleraci v *GPU*, takže nakonec i konvenční grafický procesor najde v tomto projektu uplatnění.

6.2 DSP

Úkolem signálového procesoru je předzpracování vstupních dat pro samotné vykreslení v *FPGA*. Celý program pro *DSP* je napsán v jazyce *C*, který je hojně využíván pro programování specializovaných procesorů a mikrořadičů. Díky svým nízkoúrovňovým vlastnostem jej lze snadno převést na assembler příslušného zařízení. Výhodou ale je přímá podpora cyklů a dalších základních programovacích konstrukcí, které se v assembleru implementují velmi těžkopádně a naprogramovat v něm takto složitý funkční kód je prakticky nemožné.

Kódování tvaru

Nejefektivnější způsob pro popis tvaru částice je zakódování řádků její rastrové formy tak, jak bylo prezentováno v [1].

Každý řádek může být buď prázdný, nebo se na něm nachází vždy souvisle vyplněné seskupení pixelů s různou délkou. Pro popis jednoho řádku částice je zapotřebí dvou kódů:

- ***Start of Scan*** – souřadnice pixelu od kterého se začne s vyplňováním.
- ***Length of Scan*** – délka souvisle vyplněného seskupení pixelů.

Prázdný řádek je potom kódován tak, že *Start of Scan* a *Length of Scan* po sečtení dají hodnotu větší než 7. Takový údaj je pro zobrazení neplatný a můžeme jej pro tento účel perfektně využít.

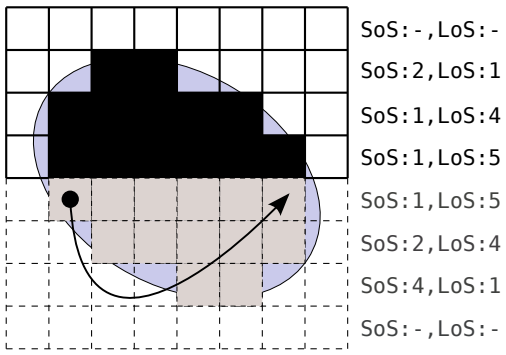
Maximální možná plocha do které lze částici zapsat je čtverec o rozměru 8×8 pixelů. Z toho jasně vyplývá, že pro dvojici kódů popisující vyplněnou oblast na řádku stačí 2×3 bity. Tvar celé částice je tedy možno popsat pomocí 8 párů *Start of Scan – Length of Scan*. Díky vlastnostem elipsy je zde ale prostor pro optimalizaci.

- Částice **menší než** 8×8 pixelů

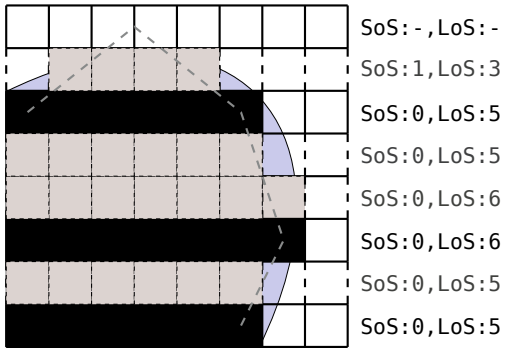
  Elipsa má tu vlastnost, že je středově symetrická. Proto je možné zakódovat pouze její polovinu a druhou část lze jednoduše dopočítat (viz obrázek 6.1). Můžeme tak ušetřit polovinu bitů, což jistě velmi oceníme při přenosu dat do *FPGA*. Pro popis tvaru celé částice rázem stačí pouze 24 bitů.

- Částice **větší než** 8×8 pixelů

  Protože požadavek na systém byl, aby padlo omezení na vykreslování částic o maximálním rastru 8×8 pixelů, je potřeba podporovat zakódování i částic větších. Ty jsou ale vždy rozděleny na kaskádu rastrů 8×8. Taková částice je potom přenášena do *FPGA* po jednotlivých částech. Jelikož nyní nejsou fragmenty elipsy samy o sobě symetrické, je potřeba odlišné zakódování. Pro jednotný přístup jak ke klasickému symetrickému, tak k interpolovanému zakódování je třeba se vejít do již zmiňovaných 24 bitů. Proto není možné přenášet všechny řádky částice. Předem určené z nich se vynechají a později se v *FPGA* zpětně interpolují (viz obrázek 6.2).



Obrázek 6.1: Zakódování rastru částice menšího než 8×8 – symetrie.

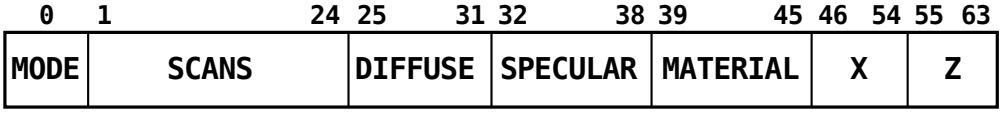


Obrázek 6.2: Zakódování rastru částice většího než 8×8 – interpolace.

Kódové slovo

Pro předávání informací o jednotlivých částicích do *FPGA* slouží takzvaná kódová slova. Je to vlastně finální produkt transformace bodu na objekt který lze vykreslit. Kódové slovo má unifikovanou strukturu (obrázek 6.3):

- ***Mode*** [1 bit] – přepíná mezi symetrickým (bit má hodnotu 0) a interpolovaným (hodnota 1) módem zakódování
- ***Scans*** [24 bit] – tvar částice zakódovaný v jednom ze dvou módů
- ***Diffuse term*** [7 bit] – parametr popisující difuzní složku světelných zdrojů
- ***Specular term*** [7 bit] – parametr popisující odrazovou složku světelných zdrojů
- ***Material index*** [7 bit] – index do tabulky materiálů
- ***X*** [9 bit] – středová souřadnice X
- ***Z*** [9 bit] – souřadnice Z (hloubka)



Obrázek 6.3: Standardní kódové slovo.

Pro komunikaci s *FPGA* je vyhrazena jen jedna datová cesta. Čas od času je ale potřeba předat jádru, starajícímu se o vykreslení, nějaký specifický příkaz. Za účelem komunikace byla vyhrazena **speciální kódová slova**, jejichž zasláním se uvnitř *FPGA* vyvolá příslušná událost. Nezbytná kódová slova jsou následující:

- ***start of load*** – jeho zasláním je *FPGA* informováno, že následující data poslouží k naplnění tabulky materiálových konstant
- ***end of load*** – ukončuje blok záznamů pro tabulku materiálových konstant
- ***shift N lines*** – posunutí o N řádků, neboli na tento počet řádků v dané jednotce nebude prováděn zápis
- ***ambient term*** – zapíše do speciálního registru složku rozptýleného světla ve scéně (součást kódového slova).

Speciální kódové slovo musí být jednoznačně identifikovatelné a rozpoznatelné od kódů reprezentujících částice. Prvních osm bitů slova pro rozlišení bohatě postačí. Potom, začíná-li šestnáctkovou hodnotou *0xFF*[1], je jasně rozpoznáno jako příkaz. Další dva bity jej již identifikují konkrétně a v případě, že je potřeba mít pro jeho provedení k dispozici i parametry, jsou tyto zakódovány ve zbývající délce slova. Konkrétně se jedná o příkaz *shift N lines*, kde je předán počet řádků o které se aktivní pruh posouvá, nebo kódové slovo *ambient term*, jež obsahuje složku rozptýleného světla scény.

Výpočet zjednodušeného osvětlovacího modelu

V kapitole 2.2 byl prezentován *Phongův* osvětlovací model. Pro implementaci v hardware je ale velice složitý a pro efektivní rychlý běh bylo nutné udělat několik kompromisů.

Jedním omezením je předpoklad bílých zdrojů, čímž odpadá nutnost přenášet do *FPGA* všechny tři barevné složky parametrů popisujících vlastnosti světla, ale postačí pouze složka jasová. Rozptýlené světlo (*ambient*) je všude ve scéně konstantní, a proto může být příslušný parametr nahrán do registru v *FPGA*, kde je poté využíván při dopočítávání osvětlovacího modelu.

Dalšímu omezení podléhá parametr definující míru odlesku, ten je nastaven pevně na hodnotu 1. Důvod je ten, že logika realizující výpočet mocniny je velmi složitá a proces renderingu by neúměrně zdržovala.

Look-up tabulky

Je zřejmé, že projekce kružnice do *2D*, následná rasterizace elipsy a stejně tak evaluace parametrů osvětlovacího modelu jsou tak náročné výpočty, že je prakticky nelze provádět

[1] je zřejmé, že nyní nelze použít pro reprezentaci prázdného řádku hodnoty *Start of Scan*= 7 a zároveň *End of Scan*= 7

za plného běhu systému. Proto je velice vhodné tyto operace provést před vlastním procesem renderingu a výsledky uložit do tabulek.

Pro konkrétní normálové vektory a velikosti částice je dopředu vypočítána třírozměrná tabulka (velikost, a dvě složky normálového vektoru, které jej plně popisují). Ta je naplněna kódovými slovy, která zatím obsahují pouze mód, tvar a konstanty osvětlovacích modelů. Během renderingu je pro konkrétní bod načten příslušný záznam z tabulky a do kódového slova jsou doplněny (logickou operací posunu a *OR*) zbylé údaje, jako je index materiálu, souřadnice středu X a souřadnice hloubky Z. Všechna slova jsou v závislosti na souřadnici Y ukládána do bufferů, odkud jsou kopírovány řadičem *DMA* do *FPGA*.

Pro zakódování větších částic je nutné použít odpovídající množství tabulek. Například pro částici s rastrem větším než 8×8, ale zároveň nepřesahujícím 16×16, jsou zapotřebí 4 tabulky s dopředu připravenými kódovými slovy (bit mód je nastaven na hodnotu 1 – interpolovaný). Každá z tabulek pak souvisí s jednou částí (čtverec 8×8 pixelů) větší částice. Pro jeden bod dojde k vybrání po jednom slovu z každé tabulky a nezbytné je přepočtení středové souřadnice (X a Y) pro jednotlivé úseky. Za předpokladu, že jsou kódová slova rozdělena do dvou polí s 32-bitovými záznamy, je opravdu nutné mít 4 tabulky obsahující tvar, ale tabulku druhou, která by měla obsahovat část z konstant osvětlovacího modelu, lze využít společnou pro oba módy. Stejný je i postup pro větší částice, jen s tím rozdílem, že například pro rozměr 32×32 potřebujeme obdobných tabulek (obsahujících zakódovaný tvar) 16. Jinými slovy – se zvětšováním rastru částice jejich počet kvadraticky roste.

Pohyb kamery

Za účelem animace, nebo dynamického chování scény je potřeba vstupní data, nahraná programem do *DRAM*, nějakým způsobem transformovat. Pro každý zobrazovaný snímek je *DSP* předána transformační matice a před převedením bodů na zakódované částice (vyhledání v tabulkách) dojde k transformaci každého z bodů podle předané matice, což ve výsledku vede k pohybu objektu ve scéně.

6.3 FPGA

Za účelem implementace jádra pro rendering v *FPGA* byl od začátku favoritem některý z *HDL* jazyků vyvinutých speciálně pro návrh hardware a použitelných při následném programování hradlových polí. Díky předchozím zkušenostem a dostupným nástrojům byl nakonec vybrán jazyk *VHDL*.

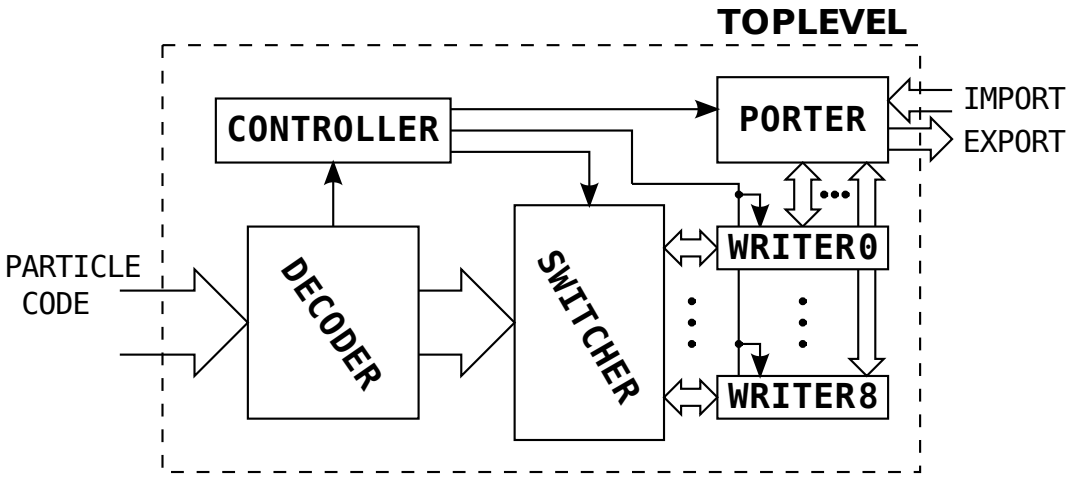
Framework

Vnitřní struktura jádra starajícího se o rendering je připojena na speciální *framework*. Ten poskytuje jednotné rozhraní, na které je pak možné připojit aplikačně specifickou jednotku tak, že se přenos bloků z *DSP* do *FPGA* a nazpět pro ni jeví transparentní. *Framework* zajišťuje i generování hodin, obsluhu konfiguračního rozhraní a také umožňuje důmyslné zřetězené propojení jednotek.

Příchozí data z *DSP* (rozhraní *EMIFB*) jsou po blocích dočasně ukládána do paměti, odkud jsou poté předávána ostatním jednotkám implementovaným na čipu.

Toplevel architektura

V důsledku omezených zdrojů v *FPGA* čipu není možné držet v paměti celou scénu. V jeden moment se uchovává pouze několik (v tomto konkrétním případě 16) řádků, z nichž je právě 8 aktivních (do nich jsou zapisovány částice). Zbytek jsou neaktivní řádky, přičemž z části se exportují buffery do další paralelní jednotky a do zbylých z nich jsou naopak data importována.



Obrázek 6.4: Schéma toplevel architektury *FPGA*.

Decoder

První jednotkou na vstupu jádra renderingu je právě *Decoder*. Jeho úkolem je převést kódové slovo popisující částici na sadu signálů, kterými jsou ovládány jednotky další.

Data přicházejí vždy po 16-ti bitech a proto je nutné nejprve nashromáždit 64-bitů, neboli jedno kódové slovo, jehož jednotlivá pole (obrázek 6.3) jsou rozkódována.

Před dekódováním samotným je ještě logikou ověřeno, zda se nejedená o speciální kódové slovo. Pakliže ano, může na jeho základě dojít k naplnění tabulky materiálových konstant (index – materiál) daty, které přicházejí v bloku mezi dvěma uvozujícími slovy (*start of load – end of load*). S osvětlovacím modelem souvisí i registr složky rozptýleného světla scény, který je po restartu vynulován a při příchodu kódového slova *ambient term* se naplní příslušnou hodnotou. Jedná li se naopak o příkaz posunu, je extrahována hodnota reprezentující počet řádků, který je předán na výstup jako řídící údaj pro *controller* jádra.

Pakliže se nejedná o kódové slovo speciální, je nejprve určen mód, na jehož základě je doplněna zbývající polovina kódů *Start of Scan – End of Scan* a to buď za pomocí symetrie, nebo interpolací vynechaných řádků (principy vysvětleny v sekci 6.2). Na základě indexu do tabulky materiálů je dopočítán zjednodušený *Phongův* model. Materiálové konstanty jsou vynásobeny s parametry popisujícími světelné zdroje (součást kódového slova, registr složky *ambient*) a sečteny po složkách *RGB*. Výsledkem je barva povrchu odpovídající osvětlovacímu modelu. Jako poslední je z kódového slova extrahována středová souřadnice X a hloubka Z.

Relevantní rozkódované nebo dopočtené údaje a stejně tak řídící signály jsou potom výstupem dekodéru.

Controller

Řídící jednotkou celého jádra je *controller*. Jeho primárním úkolem je pomocí jednoduchého stavového automatu řídit posun aktivního pruhu v obraze a také je nezbytný při importu a exportu bufferů. Informace o posunu a jeho velikosti dostává od dekodéru.

Řídící jednotka uchovává tři typy aktuálních informací:

1. **Vertikální souřadnice** v obraze na které se nachází aktivní pruh. Mění se s příkazem posunu.
2. **Počet naimportovaných** řádků, včetně osmi aktivních (s renderingem se vyčká na naplnění). Jejich počet se zvyšuje s příchozími daty z paralelních jednotek a naopak je snižován s příkazem posunu.
3. **Počet** řádků bufferu **připravených pro export**. Jejich počet se zvyšuje s příkazem posunu a naopak je snižován s každým již vyexportovaným řádkem.

Na základě těchto údajů jsou řízeny jednotky (přesněji jejich multiplexery) *switcher* a *porter*. *Controller* navíc zajistí pozastavení renderingu v případě že není naplněn aktivní pruh (není importováno alespoň 8 řádků). Proces znovu spustí jakmile je načten dostatečný počet řádků.

Switcher

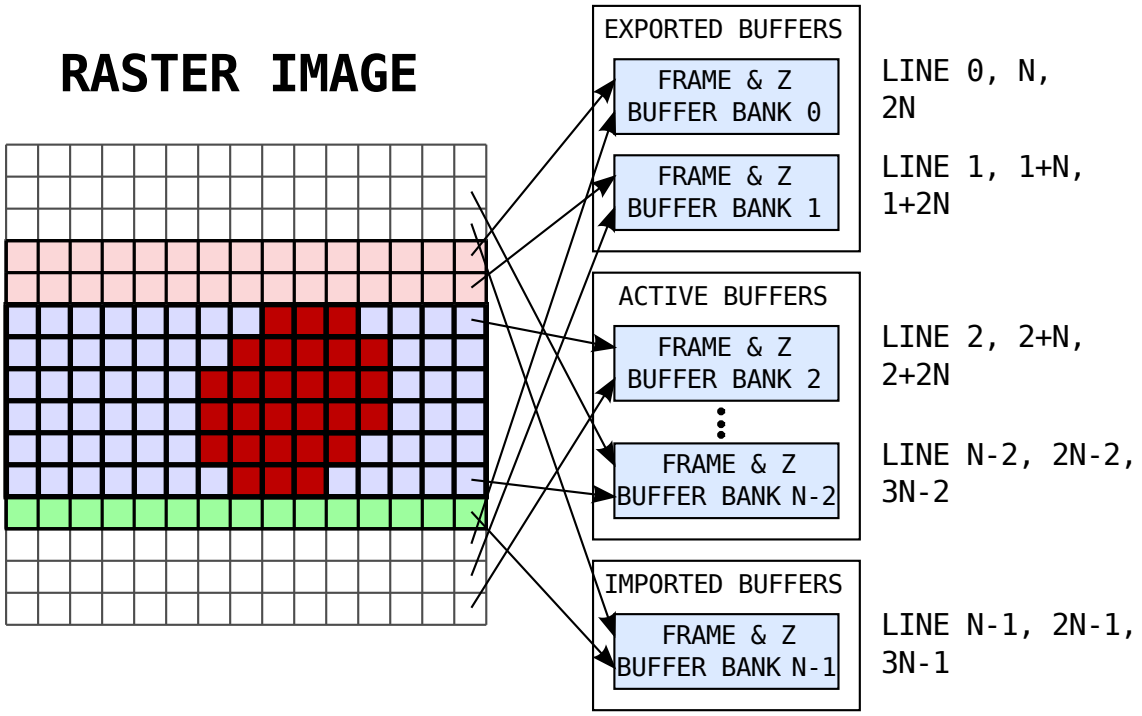
Signály na výstupu dekodéru je třeba vhodně přepínat mezi jednotky starající se o zápis. Pro tento úkol obsahuje jádro jednotku *switcher*, která je zkomponována ze sady multiplexerů.

Na základě modulo dělení údaje o aktuální vertikální souřadnici jsou k dispozici vektory (vytvořeny kombinační logikou) pro jednotlivé multiplexery. *Switcher* s jejich pomocí přepne jednotlivá výstupní rozhraní rozhraní obsahující informace o řádku částice (začátek, délka, barva, hlobka a offset) na vstupy konkrétních jednotek *writer* v závislosti na posunu aktivního pruhu v obraze.

Writer

Jednotkou, která zajišťuje samotný rendering je *writer* a v jádře jich je právě 8. Každá jednotka obsahuje dva řádky bufferu, ty ale nesmějí být oba najednou v aktivním pruhu (ideálně 1. a 9., 2. a 10., ...), protože pak by nebylo možné zápis provádět paralelně. Oba řádky jsou řešeny proložením poloviny jedné *BRAM* (primitivum *FPGA* čipu se se dvěma porty, kde jeden lze využít pro čtení a druhý pro zápis) polovinou paměti druhé, takže vždy sudé buňky patří první a liché druhé. V případě většího čipu lze takovýchto pamětí zakomponovat i více, vždy v závislosti na počtu řádků bufferu. Díky proložení pamětí lze zápis do paměti provést částečně paralelně a úměrně snížit dobu potřebnou pro zápis částice. Paměťové místo na svých 36-ti bitech obsahuje jak údaje o hloubce, tak 24-bit barvu bodu obrazu. Buffery hloubky a scény jsou tedy sloučeny do jednoho. Pro lepší představu si lze prohlédnout obrázek 6.5, blok vyznačený jako paměťový *bank* se skládá ze dvou polovin *BRAM*, které jsou vzájemně proloženy.

Zápis do paměti probíhá následovně. Jednotka předem ví, jak dlouhý souvislý pruh bodů má zapsat. Pro každý pixel je jedním portem paměti načtena původní souřadnice Z a na základě porovnání s hloubkou aktuálního pixelu se rozhodne o tom, zda se má zapsat. Pokud ano, je pro zápis použit druhý port. Načtení hodnot ze *Z-bufferu* musí proběhnout



Obrázek 6.5: Rozložení N řádků do jednotlivých bufferů.

v předstihu, ale díky zřetězení nijak výrazně dobu zápisu částice do paměti neovlivňuje. Vlastní zápis pixelů tedy probíhá sériově, což je ale zrychleno současným zapsáním sudého a lichého bodu obrazu.

Writer také zajišťuje import a export dat z bufferů. Protože neaktivní řádky používají stejné *BRAM* a dokonce stejné porty jako ty aktivní, je nutné zabránit kolizím. Lze k tomu využít buď volné cykly při předčítání z hodnot z bufferu hloubky, nebo tento transfer probíhá ve chvílích, kdy nejsou k dispozici žádná data pro rendering. Celý tento proces je jednotkou *writer* autonomně řízen.

Porter

Stejně jako jednotka *switcher* byla vlastně sestavou multiplexerů, tak i *porter* pracuje na podobném principu. Je připojen přímo na rozhraní propojené s *master FPGA* a prakticky zprostředkovává toto komunikační rozhraní příslušné jednotce *writer*.

Na základě řídících signálů od kontroléru (*controller*) spojí jednu určenou zapisovací jednotku s rozhraním pro odesílaní (export) a druhou s rozhraním pro čtení (import). Tyto propojení se dynamicky mění v závislosti na počtu nevyexportovaný a importovaných řádků.

Kapitola 7

Výsledky

Vzhledem k tomu, že dnes již existuje funkční implementace se kterou lze tento projekt porovnat, očekává se jisté minimální zlepšení výsledků.

Původní systém (tak jak dnes běží) je schopný generovat 5 milionů částic za sekundu. Nově, za předpokladu délky periody $10\,ns$ ($105\,MHz$), potřeby dvou hodinových taktů na předčtení souřadnice Z a čtyřech taktů na zapsání elementu, je výsledek $60\,ns$ na částici, což odpovídá přibližně 17 milionům částic za sekundu. To je více než dobré zlepšení. Ale tento konkrétní design je omezen velikostí *FPGA* čipu (*XC2V250*), kdy kvůli příliš malému počtu *BRAM* není možno zajistit úplné paralelní zapsání celé částice. Při použití většího z rodiny programovatelných hradlových polí, například *XC2V1000*, lze jednoduše realizovat design tak, že zápis celé částice trvá jeden takt (3 takty včetně předčtení *Z-bufferu*), výsledkem čehož je doba pro vykreslení $30\,ns$ na element a a počet částic za sekundu je dvojnásobný. V úvahu musíme vzít ještě možnost paralelního zpracování, kdy například při použití čtyřech modulů osazených *XC2V1000* je systém schopen (ideální případ neuvažující režii při předávání bufferů) generovat až 133 milionů částic za sekundu.

Co se týká přenosu kódových slov z paměti *DSP* (model *TI TMS320C6416*) do *FPGA* za účasti *DMA*, tak s drobnou režií (jeden až dva takty) probíhá přenos 16-ti bitů za jeden hodinový takt. Z toho vyplývá, že na přenos jednoho elementu do *FPGA* je potřeba $40\,ns$, což by znamenalo neustálé zásobování vstupními daty bez prodlev. Reálná rychlost přenosu je pak přibližně $200\,MB/s$

Distribuce bodů pomocí *randomize* funkcí se ukázala jako nejlepší řešení. Body jsou rovnoměrně rozděleny mezi jednotlivé moduly a v drtivé většině případů nedochází k výraznějším čekáním, jaké by způsobilo zahlcení jednoho ze zařízení.

Kapitola 8

Závěr

V úvodu bakalářské práce bylo stanoveno několik cílů. Vzhledem k tomu, že se podařilo navrhnou flexibilní strukturu zobrazovacího systému, který je schopen realizovat paralelní zpracování a zároveň efektivně akceleruje výpočty za pomocí hardware, považuji vytyčené cíle za splněné.

Zadáním této bakalářské práce bylo pět jasně definovaných bodů, které byly splněny takto:

1. Co se prvního bodu týče, tak nastudování potřebné literatury na téma zpracování obrazu a struktury *FPGA* obvodů jsem provedl ještě před započetím prací na projektu jako takovém. Některé ze získaných poznatků jsou uvedeny v úvodních kapitolách, přičemž prameny, kterými jsem se inspiroval, jsou uvedeny na konci této práce.

2. Úkol skrývající se pod bodem číslo dvě, totiž vytypovat (za asistence vedoucího) vhodný algoritmus pro implementaci v hardware, byl také proveden ještě před započetím vlastní práce na projektu. Na základě zkušeností se stávající implementací systému zobrazujícího *point clouds* bylo rozhodnuto o implementaci modifikované verze, která naplno využije speciálně vyvinutého hardware a odstraní některé neduhy a omezení té původní. V úvodních kapitolách je tento algoritmus představen a později jsou v kapitole 4 diskutovány požadované inovace.

3. Implementace systému (tedy bod 3) byla pro každou jeho část provedena ve vhodném programovacím jazyce na základě připraveného návrhu. Všechny programy byly za účelem ladění a pozdějšího hladkého běhu v hardware simulovány. Veškeré poznatky ohledně implementace jsou uvedeny v kapitolách 5 a 6

4. Bod 4, neboli demonstrace funkčnosti a výkonnosti. V hardware byla provedena série zátěžových testů, které napoví, jaké výsledky lze od systému po odladění očekávat. Také byl na úrovni simulací otestován proces renderingu samotný včetně teoretických předpokladů ohledně distribuce zátěže. Tyto poznatky jsou shromážděny v kapitole 7.

5. Diskuze dosažených výsledků (bod 5) proběhla na základě srovnání stanovených cílů před započetím prací na projektu a cílů dosažených po jeho dokončení. Vše je podrobně rozebráno v kapitole 8.

Z mého pohledu se zobrazování *point clouds* modelů jeví jako velice zajímavý problém. V oblasti hardware, kde je v komerční sféře podpora této grafické reprezentace prakticky

nulová, by do budoucna mohly některé ze zde uvedených nápadů najít reálné uplatnění. Myslím si, že vyrábět a prodávat specializované procesory pouze pro výpočet *point clouds* scén nemá smysl, ale integrace tohoto konceptu do *GPU* by byla více než vhodná a dle mého názoru není do budoucna nereálná. V každém případě tento projekt odkryl další potenciály *point clouds* a v oblasti počítačové grafiky tuto reprezentaci zase o něco více zviditelnil.

Jako obtížnější se jevilo oživování systému v hardware. Celek se velmi špatně ladí, hlavně proto, že není možné použít ladící nástroj jako je například *debugger* v případě software. Hardware je velmi nevyzpytatelný a je potřeba si dát velký pozor při návrhu a pečlivě vše odsimulovat, protože jakákoli nepřesnost může způsobit problémy (například zatuhnutí). Systém se skládá z různých částí naprogramovaných v odlišných jazycích a zajistit občas hladký běh interakcí mezi nimi je nadlidský úkon. Velmi mi pomohly rady a zkušenosti kolegů, kteří již některé multimediální systémy na stejné platformě oživovali.

Ve srovnání s původní implementací si lze všimnout znatelného nárůstu výkonu už při použití jediného modulu a malého *FPGA*. Výkon v počtu částic za sekundu se zvýšil více než třikrát (17 milionů elementů za sekundu). Při použití čtyřech paralelních jednotek s velkými čipy dochází k teoretickému nárůstu až 133 krát.

Co se týče výhledu do budoucna, rád bych ještě zapracoval na optimalizaci procesu zobrazení a tím mírně zrychlil běh systému. Také bych chtěl tento zobrazovací systém přizpůsobit pro zprovoznění složitější animace (demo), která by na první pohled zaujala i problematiky neznalého člověka.

Jako možnost se jeví i přesunutí části implementované v *DSP* do programovatelného hradlového pole, což by odstranilo závislost na signálovém procesoru. Také by bylo možné všechny zmíněné algoritmy zakomponovat do grafických procesorů a záleží pouze na výrobcích, jak se k tomuto problému postaví.

Literatura

[1] Adam Herout and Pavel Zemčík. Animated particle rendering in dsp and fpga. In *SCCG 2004 Proceedings*, pages 237–242. Slovak University of Technology in Bratislava, 2004.

[2] Microsoft, USA. *DirectX*, 2008. [Online at http://www.microsoft.com/directx; accessed 13-May-2008].

[3] Bui-Tuong Phong. Illumination for Computer Generated Pictures. 18(6):311–317, 1975.

[4] Wikipedia. Digital signal processor — Wikipedia, the free encyclopedia, 2008. [Online; accessed 09-May-2008].

[5] Wikipedia. Phong shading — Wikipedia, the free encyclopedia, 2008. [Online; accessed 02-May-2008].

[6] Wikipedia. Point cloud — Wikipedia, the free encyclopedia, 2008. [Online; accessed 12-April-2008].

[7] Wikipedia. Z-buffering — Wikipedia, the free encyclopedia, 2008. [Online; accessed 26-April-2008].

[8] Xilinx, USA. *Virtex-II Platform FPGAs*, ds031-4 (v3.5) edition, November 5 2007. [Online at http://www.xilinx.com; accessed 12-May-2008].