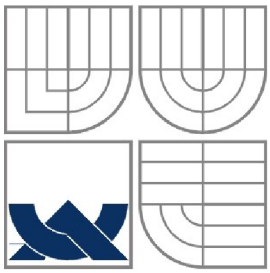
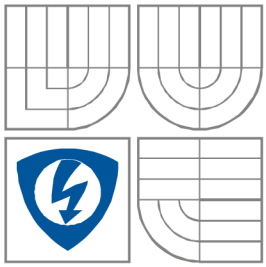


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

IMPLEMENTACE GRAFICKÉ KNIHOVNY AGG NA PROCESOR OMAP

Implementation of graphical library AGG for OMAP procesor

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN PAVLINEC
AUTHOR

VEDOUCÍ PRÁCE — Ing. SOBĚSLAV VALACH
SUPERVISOR

BRNO 2012

Abstrakt

Tato práce se zabývá popisem implementace grafické knihovny AGG na procesor OMAP. Představuje jednotlivé technologie procesoru a jejich možné využití v grafické knihovně. Součástí je také přehled použitých nástrojů a popis jejich využití.

Klíčová slova

OMAP,DSP,ARM, Anti-Grain Geometry,C6EZRun

Abstract

This thesis describes the implementation of the AGG graphics library for OMAP processor.It describes processor technologies and their possible use in the graphic library. A summary description of the specific tools is also included.

Keywords

OMAP,DSP,ARM, Anti-Grain Geometry,C6EZRun

Bibliografická citace:

PAVLINEC, J. Implementace grafické knihovny AGG na procesor OMA*P*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. . Vedoucí diplomové práce byl Ing. Soběslav Valach.

Prohlášení

„Prohlašuji, že svou bakalářskou práci na téma Implementace grafické knihovna na procesor OMAP jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **23. května 2012** ……

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Soběslavu Valachovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: **23. května 2012** ……

podpis autora

Obsah

1 ÚVOD

Tato práce se zabývá implementací grafické knihovny Anti-Grain Geometry na procesory OMAP.
Protože se tyto procesory svou strukturou a určením odlišují od známějších procesorů založených na architektuře x86 je úvodní část věnována podrobnějšímu popisu jednotlivých částí a technologií využitých v těchto procesorech. Dále jsou popsány hlavní části grafické knihovny AGG a některé algoritmy, které tato knihovna využívá.
Další část popisuje jednotlivé SW nástroje použité při implementaci knihovny a objasňuje princip jejich funkce.
Závěr je věnován popisu testů a úprav v jednotlivých částech knihovny AGG.

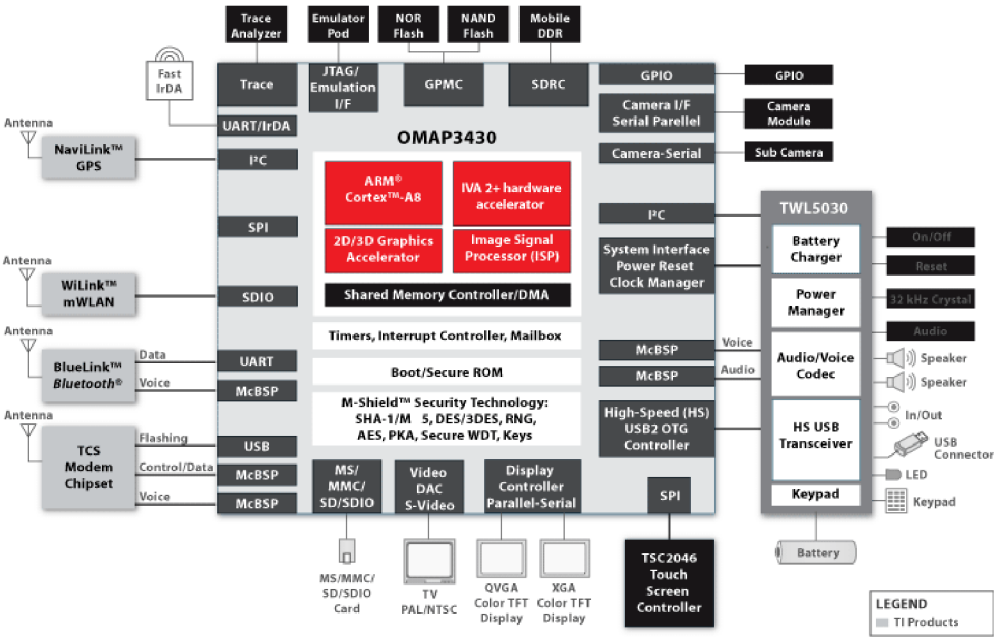
2 PROCESORY OMAP

Procesory OMAP představují kategorii SOC určených pro přenostné a mobilní multimediální aplikace vyráběných firmou Texas Instruments.
Obecným znakem těchto procesorů je výskyt více výpočetních jednotek na jednom čipu, nejčastěji v kombinaci GPP,DSP a GPU. Díky tomuto uspořádání dokáže procesor zpracovávat řadu výpočetně náročných aplikací jako je kódování/dekódování videa v reálném čase, spouštění 3D her atd. Hlavní výhodou tak je, že narozdíl od procesorů architektury x86 k těmto úkonům nepotřebuje žádné přídavné externí karty. Zároveň je díky tomuto uspořádání dosaženo malé spotřeby tak ,aby bylo možné výsledné zařízení napájet z baterie.
Mezi nevýhody této architektury lze zařadit poměrně malá podpora sběrnic mezi které patří především i2c,usb,mmc nebo spi . Chybí například sběrnice pcie, což je ale vzhledek k většinovému využití v mobilních aplikací pochopitelné.
V současné době (rok 2012) je na truhu 5. generace OMAP5, která je určena pro novou generaci mobilních telefonů a tabletů.

3 OMAP3530

Tato práce se zabývá podrobněji procesorem OMAP3530, z tohoto důvodu jsou následující řádky věnovány popisu jeho některých komponent a implementovaných technologií.Uvedený popis lze s určitými vyjímkami zobecnit i na vyšší generace procesorů OMAP, především část o procesoru ARM a DSP.

OMAP3530 tvoří tři základní výpočetní části. Jsou to ARM Cortex-A8,signálový procesor DSP řady TMS320C6x a procesor pro zpracování 3D grafiky PowerVR SGX530.



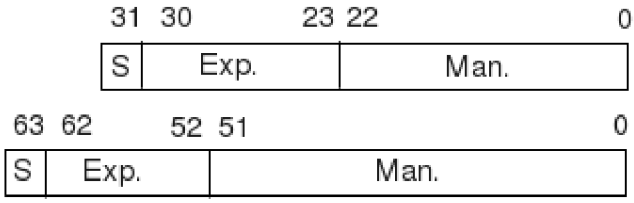
Obr. 1: Schéma struktury procesoru OMAP3430 zdroj [3]

3.1 ARM Cortex-A8

Jedná se o 32 bitový RISC procesor založený na architektuře ARMv7. Z hlediska výkonu běží na frekvenci 600MHz a velikost L1 cache je 32 KB.
Procesor také implementuje technologie VFP, NEON, Thumb-2,Jazzel a TrustZone.
Technologie Jazzel umožňuje přímé vykonání Java bytekodu na procesoru ARM.
TrustZone přidává možnost zabezpečení periferií (paměti, klávesnice,obrazovky, a kryptovacích modulů tak aby bylo zabráněno možným sofwarovým útokům. (Tato technologie je ovšem výrobcem v základu deaktivovaná). Technologie VFP,NEON a částečně i Thumb mají využití v multimediálních aplikacích. Z tohot důvodu jim jsou věnovány následující odstavce.

3.1.1 VFP

VFP implementuje matematický koprocesor, konkrétně v architektuře VFPv3. VFP obsahuje registry pro uchování čísel v plovoucí desetinné čárce v 32bitové a 64bitové přenosti. Pro tyto registry je pak možno využít operací sčítání,odčítání, násobení, dělení, akumulační operace a mocniné operace. VFP poskytuje také instrukce pro konverzi dat v plovoucí desetinné čárce na hodnoty v reprezentaci s pevnou desetinnou čárkou.
Pro reprezentaci čísel v plovoucí desetinné čárce se využívá rozdělení na mantisu a exponent dle standardu IEEE 754.



Obr. 2: Reprezentace čísel v registrech VFP

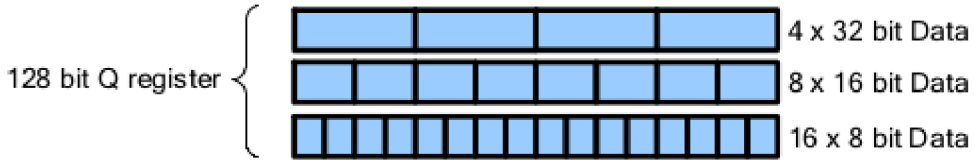
Komprocesor VFP může pracovat celkem ve 4 výpočetních módech:

- Full-compliance mode
- Flush-to-zero mode
- Default NaN mode
- RunFast mode

V modu Full-compliance jsou všechny výpočty v plovoucí desetiné čárce prováděny v souladu s normou IEEE 754. Mód Flush-to-zero zabezpečuje, že v případě operace jejíž výsledekm by bylo subnormální číslo bude výsledek operace nula. Default NaN mode naopak zabezpečuje, že výsledkem operace s číslem NaN bude opět číslo NaN. RunFast mod kombinuje všechny výše popsané módy.

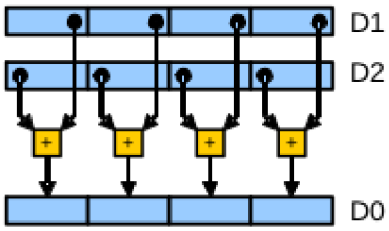
3.1.2 NEON

Tato technologie umoňuje na procesoru ARM současné vykonání matematických a logických operací nad množinou více dat (vektorů) s využitím instrukcí typu SIMD.
SIMD instrukce představují, dle Flynnovy klasifikace, jeden z možných způsobů paralelizace úloh. Daná operace se tak nad vektorem provádí v jednom taktu a díky tomu je možné několikanásobně urychlit výpočet určitého okruhu úloh.
NEON obsahuje celkem 16 128bitových vektorových Q registrů. Q registr se dělí na 64bitové D registry, které lze poté ještě dělit menší 32,16 a 8 bitové registry. Tyto registry jsou společné s VFP koprocesorem a jsou odděleny od ARM registrů.



Obr. 3: Dělení Q registrů, zdroj [2]

Zpracování SIMD instrukcí se provádí odděleně od instrukcí ARM. Z tohoto důvodu existuje fronta pro 16 SIMD instrukcí mezi pipeline ARMu a NEONu. Do doby než je fronta zaplněna nevykonanými instrukcemi tak probýhá zpracování instrukcí NEON i ARM paralelně.



Obr. 4: Vektorové sčítání D registrů , zdroj [2]

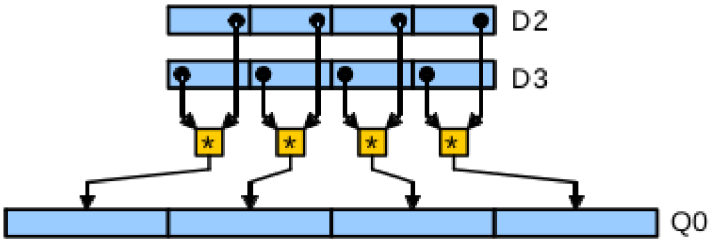
Pro načtení/uložení hodnot do Q registrů a jeho podregistrů slouží instrukce VLD (Vector Load) a VST (Vector Store). Při přesunu hodnot z NEON registrů do ARM registrů je dle dokumentace vždy minimální zpoždění 20 cyklů. Z tohoto důvodu může být kód využívající velkého množství přesunů mezi danými registry pomalejší než kód využívající čistě ARM instrukce.

Kvůli zefektivnění přesunu dat z pracovních registrů ARM do vektorových registrů NEON (a naopak) obsahuje instrukce další pomocné informace. Syntaxe instrukcí SIMD v assembleru má tak následující podobu:

VLD mezera.datovy typ {registry NEONu},adresa

Kde hodnota mezera je číslo 1-4 a přestavuje rozestup načtených dat z paměti (1 až 5 bity). Zle tak např. Načíst pouze jednu složku barvy uloženou v paměti v pořadí RGB. Hodnota datového typu určuje počet bitů pro prvky registru. Může pak obsahovat hodnoty 8,16 a 32. Registry obsahují seznam cílových registrů a hodnota adresa je nejčastěji adresa zdrojové paměti pro čtení, která je uchovávaná v některém z pracovních registrů ARM.
Instrukce provádějící výpočetní operace mají syntaxi podobnou.

VMUL.*datový typ cíl.datový typ zdroj Cílový registr, registr 1,registr 2*



Obr. 5: Vektorové násobení D registrů , zdroj [2]

Podporovány jsou operace jak se stejným cílovým datovým typem jako mají zdrojové registry tak i přetečení do většího datového typu nebo naopak zmenčení cílového datového typu (např. V případě bitového posunu).

3.1.3 Thumb-2

Thumb-2 přestavuje alternativní instrukční sadu k instrukcím ARM. Tato instrukční sada slouží především pro využití v embedded aplikací náročných na velikost výsledného spustitelného kódu.

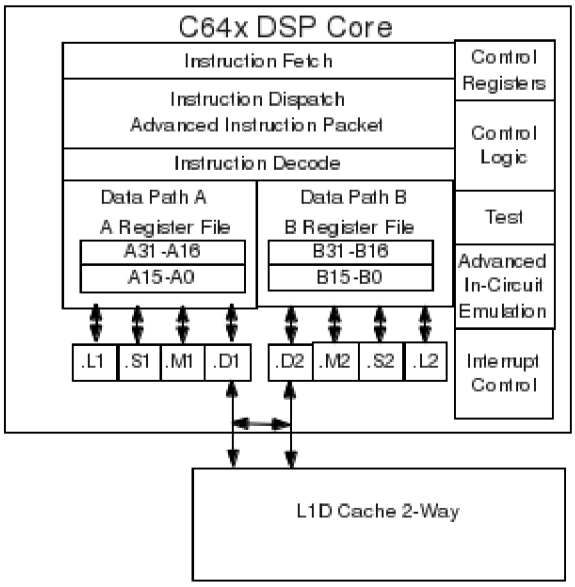
Velikost kódu v instrukční sadě ARM je do značné míry ovlivněna velikostní instrukčního slova. To je dlouhé 32 bitů.

Thumb2 tak zavádí proměnou velikost instrukčního slova (narozdíl od sady Thumb). Nové instrukce vycházejí především z praktických potřeb reálných programů, z tohoto důvodu je poměrně komplikované psát ručně kód využívající tuto instrukční sadu.

Zmenšení velikosti kodu s ohledem na rozunmé prodloužení vykonávaných instrukcí.

3.2 DSP

DSP zle obecně popsat jako procesor, jehož návrh byl optimalizován pro práci s digitálními signály. Tomuto účelu je tak přizpůsobena architektura jádra procesoru i jeho instrukční sada. Procesor OMAP3530 je využit signálový procesor řady TMS320C6x



Obr. 6: Struktura DSP, zdroj [3]

Jádro procesoru se skládá z dvou sad funkčních jednotek .L1 .S1 .M1 a .D1.Dvou souborů registrů, kde každý soubor obsahuje 32 32 bitových registrů pro obecné použití.

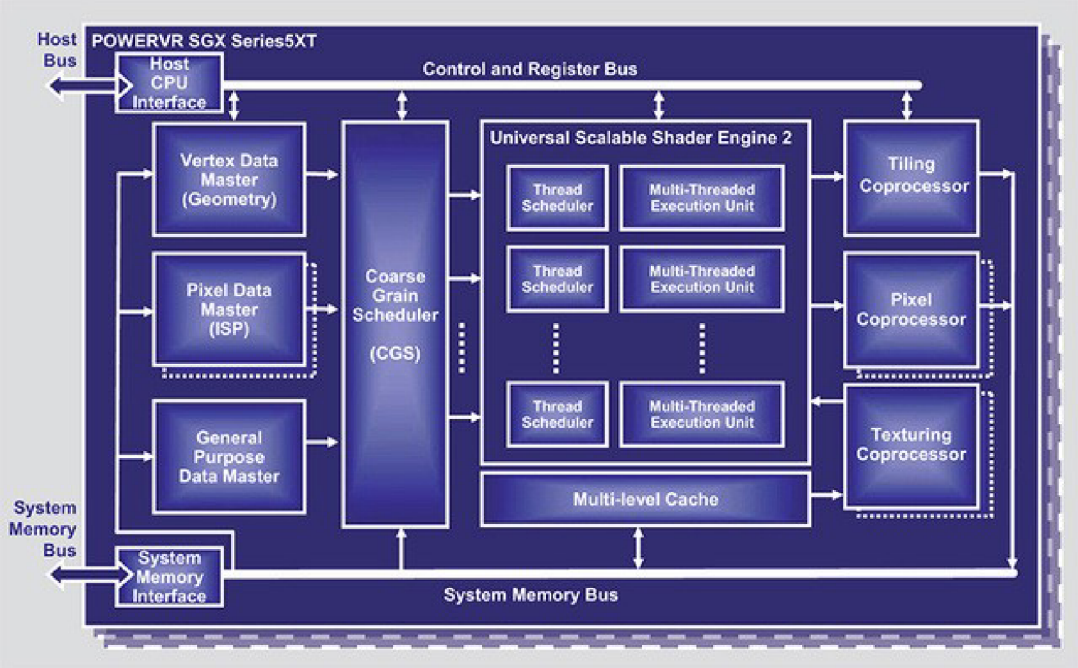
Význam jednotek je následující:

- Jednotka .D slouží pro manipulaci s daty a umožňuje tak v jedné instrukcí načít/uložit 8,16,32 a 64 bitové data do registrů procesoru.
- Jednotka .M představuje rychlou násobičku. Každá z jednotek dokáže zpracovat v jednom hodinovém tiku 2 násobení 16x16 bitových dat, nebo 4 násobení 8x8bitových dat.
- Jednotky .S a .L zajišťují obecné aritmetické, logické apod. funkce, které jsou vykonány v jednom hodinovem cyklu.

3.3 GPU

Jako grafický procesor je používán PowerVR SGX530 vyráběný firmou Imagination Technologies.PowerVR běží na frekvenci 200 MHz a může dostahvat výpočetního výkonu až 1.6 GFLOPS. Cílem této jednotky je urychlení renderování 3D objektů tak, aby podobnými operacemi nemusel být zdržován procesor ARM. Pro renderování 3D objektů je využívána metoda TBDR (Tile-based defferred rendering).

Na rozdíl od procesoru ARM a DSP je specifikace SGX uzvařená. Pro vývojáře jsou dostupné ovladače impelentující podrporu standardu OpenGL es 2. Výrobce poskytuje informace a vývojové prostředky, které mohou pracovat s GPU na nižší úroveni pouze vybraným vývojářům z řad velkých firem. Proto je u této části přiloženou pouze obecné schéma GPU.



Obr. 7: Schéma GPU POWERVR SGX ,zdroj [6]

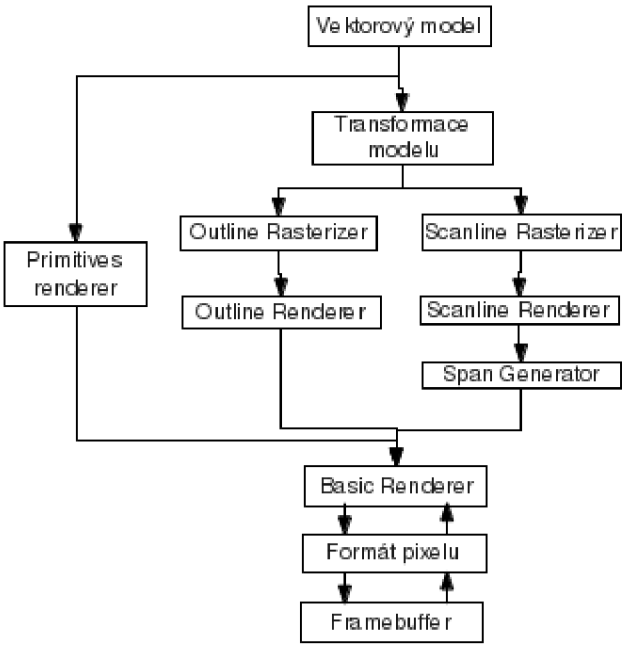
4 GRAFICKÁ KNIHOVNA AGG

AGG je 2D grafická knihovna určená pro práci s vektorovými obrazy. Pro práci s grafikou poskytuje funkce jako je vykreslování, vyplňování a reprezentace obrazové informace. Protože se jedná o čistě softwarovou grafickou knihovnu, která neobsahuje napojení na HW akceleraci, lze ji implementovat na řadu platforem.
Celá knihovna je napsaná v C++. Jednotlivé části jsou odděleně implementovány v různých třídách (renderování, rasterizace,reprezentace pixelu, vektorová reprezentace objektů apod.). Jedná se tedy o modulární systém navržený tak, aby šlo jednotlivé části dle potřeby upravit.Pro tyto účely agg využívá také paramertický polymorfismus (pomocí šablon v c++) tak, aby upravená část (např. Rasterizace) šla bez zásahu do ostatních částí znovu napojit do knihovny.

Mezi hlavní proklamované přenosti je považována vysoká kvalita výstupní grafiky. Knihovna implementuje anti-aliasing a subpixelovou přenost.

Poslední vydaná verze v2.5 je z roku 2006. Celá knihovna je šířena pod licencí GPL v2.

4.1 Části knihovny



Obr. 8: Schéma propojení jednotlivých částí knihovny

4.1.1 Vektorový model a transformace

Vektorový model

Základem každé vektorvé grafiky jsou vektory. Z tohoto důvodu je v knihovně několik tříd, které se umožňují konstrukci a uchování základních geometrických primitiv. Již v této části se uplaňuje využití subpixelové přenosti, protože předávané souřadnice jednotlivých objektů jsou typu double. Takto přesné souřadnice pak slouží především pro rasterizer, který je využívá při anti-aliasing.

Mezi tyto třídy patří path_storage pro tvorbu polygonů, čar, polyčar a křivek. U polygonů/polyline využívá metody move_to,line_to a close_polygon. V případě křivek obsahuje několik metod (curve,curve3,curve4), kde číslo metody určuje počet řídích bodů bézierovy křivky.Pro uchování obecné elipsy je potom určena třída ellipse.

Transformace modelu

Účelem této části knihovny je poskytnout metody pro požadovanou transformaci vektorového modelu. Jednotlivé transformace jsou implementovány v oddělených třídách tak, aby byl zachován modulární princip knihovny.

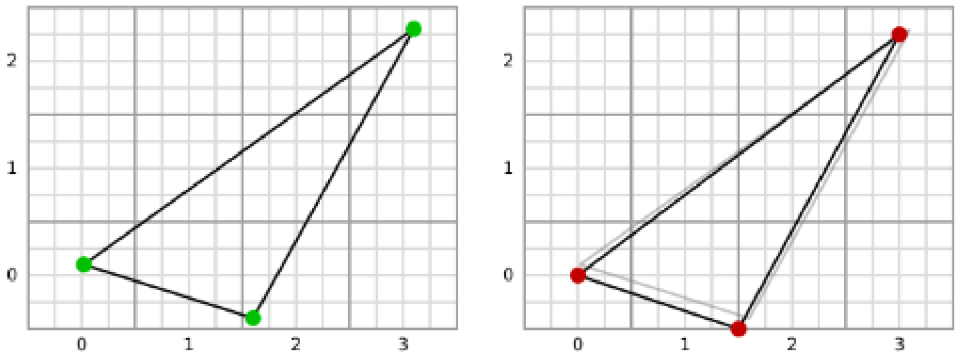
Zpravidla nejvyužívanější jsou třídy trans_affine (určené pro lineární transformace translace, rotace atd), conv_curve (zajišťující převod křivek na úsečky tak, aby je mohla zpracovat rasterizační část) a třída conv_stroke (definující šířku čar a jejich přesné napojení).

4.1.2 Rasterizace a rendrování

Pro rasterizaci lze v AGG v zásadě použít tři různé třídy. Jedná se o scanline rasterizaci,outline rasterizaci a rasterizaci základní primitiv jako jsou elipsa, čáry, apod. . Všechny třídy implementují odlišné algoritmy.

Scanline rasterizace

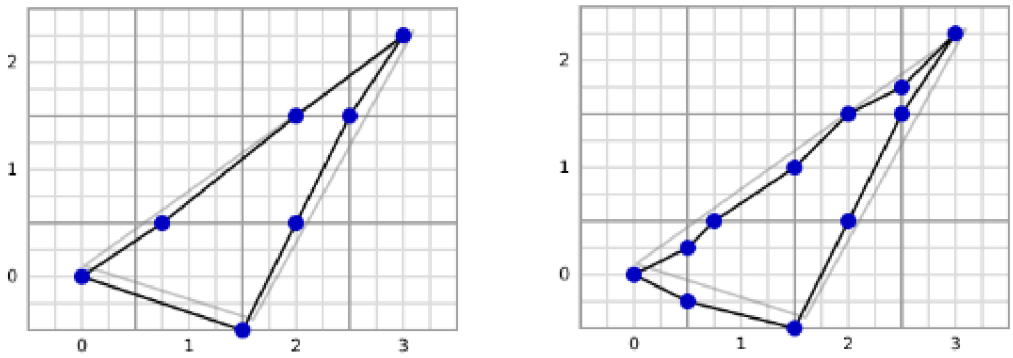
Nejpoužívanějším algoritmem pro rasterizaci v agg je scanline rasterizace. Tento algoritmus v sobě implementuje využití subpixelové přesnos a antialiasingu.



Obr. 9: Scanline algoritmus (zarovnání na subpixelovou mřížku), zdroj [7]

AGG používá subpixelovou přenost 1/256, každý pixel je tak rozdělen na 256 oblastí. V příkladu je pro přehlednost využita pouze přenost 1/4.

Při rasterizaci algoritmus postupuje tak, že nejdříve zarovná koncové body polygonu na subpixelovou mřížku. (obr 9).

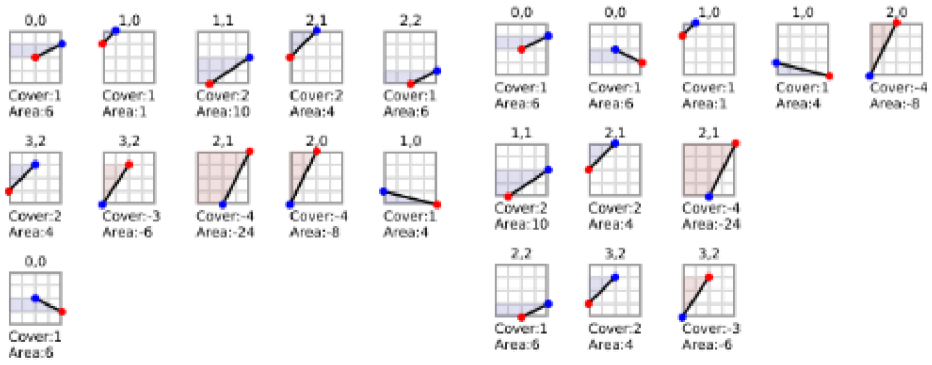


Obr. 10: Scanline rasterizace (zarovnání průsečíků na subpixelovou mřížku), zdroj [7]

V dalším kroku zarovná průsečíky s pixelovou mřížkou na jejich subpixelové ekvivalenty (nejdřívě podle osy x a poté podle osy y (viz obr. 10).

Na obr. 10 souřadnice (0,1) lze vidět ztrátu přenosti, která je při tomto zpracování způsobena. Tyto chyby jsou v AGG potlačovány využitím subpixelové přenosti 1/256 a tak je jejich projev v praxi minimální.

Po tomto zpracování vznikne 11 čar, které tvoří původní trojúhelník.



Obr. 11: Scanline rasterizace (zpracování subpixelové mřížky) ,zdroj [7]

U všech segnemtů se vypočte tvz. hodnoty cover a area. Ty platí pro jednotlivé pixely. Hodnota cover značí výšku čáry v subpixelové mřížce a hodnota area udává počet pokrytých subpixelů v jednotlivém pixelu od levého okraje po čáru. Podle orientace čáry je pak určeno znaménko pro obě hodnoty (cover i area).

Po seřazení jednotlivých části podle hodnoty Y a X se sečtou hodnoty cover a area, které mají stejné souřadnice. Z výsledných hodnot se pro každý segment vypočte dle vzorce šířka* hodnota cover – hodnota area. Výsledkem je maska pokrytí pixelu, které se mapuje na hodnotu 0-256.

Výsledný obraz je potom předán ve scanline kontajneru (např. scanline_u8) k dalšímu zpracování, kde se polygon vybarví nebo se s použitím span generatoru aplikuje složitější grafické pokrytí.

“Span Generator” slouží pro případy, kdy je potřeba vyplnit objekty např. Texturou, gradientem apod.

Rasterizace primit

Pod tímto pojmem jsou myšleny metody obsažené v třídě renderer_primitives. Jejím hlavní cílem je poskytnout metody pro rychlého kreslení základních objektů jako jsou čáry, elipsy, čtverce, apod. . Jedná se o čisté implementace bresehmanova algoritmu,

které nepodporují anti-aliasing. Jejich využití je především v aplikacích, které potřebují co největší rychlost vykreslování.

Reprezentace pixelů

Pro všechny výše uvedené části knihovny není podstatný způsob uchování pixelů v paměti. Tyto informace jsou totiž odděleně uchovávány ve třídě pixfmt_rgb,pixfmt_rgba apod.

To se využije především v situaci, kdy máme zařízení, které používá jiný formát reprezentace barev než rgb (např. Gbr apod.) Při zápis do bufferu se tak nemusí ztrácet čas konverzí rgb->gbr , protože se využije třída s podporou příslušného formátu.

AGG používá barevný prostor RGB, z tohoto důvodu neobsahuje třídy pixfmt s podporou barevného prostoru CMYK,YUV apod.. Při potřebě využít jiný barevný prostor je proto nutné vypořádat se s konverzí (a faktem že některé barvy z CMY nebudou obsahovat odpovidajici barvu RGB) posvém.

Framebuffer a Basic renderer

Poslední částí knihovně je Framebuffer a basic renderer. Framebuffer představuje místo v paměti popř. HW rozhranní umožňující akceleraci vykreslování. Basic renderer uchovává infomrace o velikosti, šířce a výšce zvoleného framebufferu stejně jako jeho umístění v paměti. Výhodou třídy basic_renderer je také možnost rozdělení framebufferu do více oddělených částí.

5 VÝVOJOVÁ PLATFORMA

Pro testování implementace byla použita „vývojová deska“ Beagleboard rev C.

Ta má následující parametry:

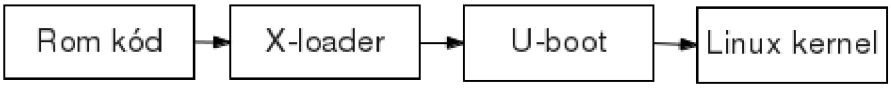
- Procesor TI OMAP3530
- 256 MB LPDDR RAM
- 256 MB NAND Flash

Jako testovací operační systém byla použita distribuce Ångström. Jedná se o linuxouvou distribuci určenou pro embedded zařízení využívající procesory ARM. Součástí této distrubuce je také linuxové jádro z vývojové větve Linux OMAP kernel.

Další odstavce se věnují operacím, které jsou prováděny při spuštění systému a jsou nutné pro jeho správnou inicializaci.

5.1 Zavaděč systému u procesorů OMAP

O správnou inicializaci procesoru a periferií se stará zavaděč. U procesorů OMAP je celá zaváděcí sekvence rozdělena do 4 základních kroků. Každý krok má na starosti odlišný zavaděč u kterého obecně platí, že vykonává nezbytné množství inicializačních operací tak, aby bylo šlo přejít k další části zaváděcí sekvence.



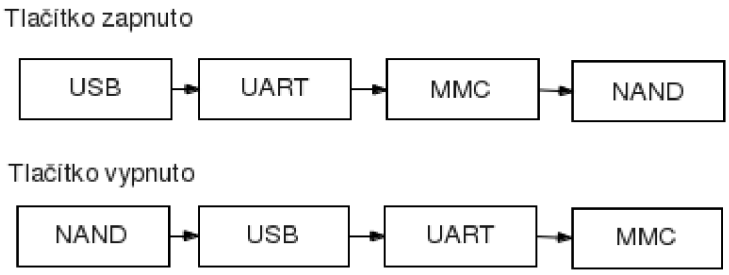
Obr. 12: Schéma postupného volání jednotlivých zavaděčů

5.1.1 ROM kód

Pří spuštění provede ROM kód konfiguraci hodin, paměti a základních periferií. Poté prohledá jednotlivé zařízení zda obsahují platný obraz x-loaderu.

Pak může kód uložený v ROM paměti pokračovat v boot sekvenci načtením x-loaderu z několika zařízení jako jsou UART,MMC ,NAND paměť nebo USB. Pořadí bootovaní z těchto zařízení je určeno na základě konfigurace SYSBOOT pinů. Nastavení pinů lze přečíst z fyzické adresy 0x480022f0.

U platformy Beagleboard lze konfiguraci této sekvence měnit pomocí tzv. Uživatelského tlačítka. V závislosti na jeho aktivaci tak může uživatel ovlivnit pořadí prohledávání jednotlivých zařízení.



Obr. 13: Priorita prohledávání zařízení

Poté co je nalezen vhodný obraz x-loaderu na některém ze zařízení, načte jej ROM loader do vnitřní 16Kbytové paměti SRAM a spustí jej.

Způsob jakým bude prováděna komunikace mezi ROM loaderem a zvoleným zařízením je popsán v následujících odstavcích.

UART

Při bootování ze seriové linky je nejdřív vysláno identifikační číslo přes seriovou linku. V případě že hostitel odpoví během definovaného intervalu začne program z ROM číst data do paměti SRAM. V případě, že během přenosu není detekována žádná chyba, spustí zavaděč kód uložený v paměti SRAM.

SD Card Boot

ROM zavaděč zkontroluje, zda lze přistoupit z řadiče MMC k obsahu SD karty. V případě, že je karta nalezena zkontroluje zda je její první oddíl ve formátu FAT32. Po této kontrole vyhledá soubor s názvem MLO. Tento soubor obsahuje hlavičku s informacemi o umístění bootloadru v paměti, jeho velikost a spustitelný kód zavaděče x-loader. Na základě informací z hlavičky je pak načten x-loader do paměti SRAM a poté spuštěn.

NAND Boot

U zařízení s NAND pamětí je nejdříve načten její první sektor. V případě že sektor obsahuje kód, nahraje jej zavaděč do paměti SRAM a předá mu kontrolu. V situaci, kdy je první sektor prázdný nebo poškozený se pokusí zavaděč načíst další sektory (maximálně 4) Neuspěje-li ukončí slou činnost.

Protože Beagleboard Rev C obsahuje 256 MB NAND pamět, je v dalších částech paměti uložen také u-boot, konfigurační soubor pro u-boot a linuxové jádro.

5.1.2 X-loader

Protože velikost paměti SRAM je omezena, je hlavním úkolem x-loaderu inicializace paměti SDRAM, hodin,multiplexerů a seriové konzole. Po úspěšné inicializaci nahraje do paměti u-boot a spustí jej.

5.1.3 U-boot

Ve třetí fázi zaváděcího procesu se provede inicializace grafického subsystému a dalších částí v závislosti na vývojové desce a jejich periferii. Dalším důležitým krokem je přečtení konfiguračního souboru uEbv.txt, který obsahuje parametry předávané jádru. Jedním z těchto paramtrů je argument mem.Ten určuje velikost a fyzickou adresu paměti využitelnou jádrem.
V tomto okamžiku, lze určit sdílenou operační paměť mezi procesorem ARM a DSP.

Jako příklad lze uvést následující hodnotu:
mem=99M@0x80000000 mem= 128M@0x88000000
Ta v paměti vytváří okno velikosti 29 MB, které může sloužit jako sdílená paměť mezi procesory.
Podobným argumentem je také omapfb, který definuje nastavení grafického subsystému OMAP.

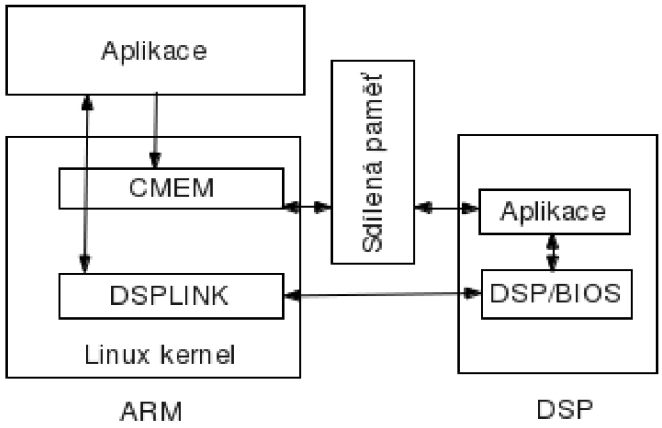
Příklad použitého argumentu:
omapfb.video_mode=800x400MR-24@60
Ten nastaví hodnotu rozlišení na 800x400 pixelů,24bitovou barevnou hloubku a obnovovací frekvenci na 60 HZ.
Po úspěšném nastavení argumentů uboot načte jádro do paměti SDRAM a předá mu řízení.

5.2 Operační systém

Z hlediska zavádění celého systému zastává jádro linux dvě stěžejní úlohy.
Prvním je inicializace LCD,HDMI,I2C,zvukového systému apod. Druhou je připojení souborového systému a poskytnutí rozhranní pro práci s DSP a GPU načtením potřebných jaderných modulů.

5.2.1 Linux a DSP/BIOS

Z hlediska systému na procesorech OMAP představuje nejčastější využití scénář, kdy operační systém a aplikace běží na procesoru ARM a při potřebě vykonání výpočetně náročných úloh se využije DSP. Oba procesory spolu poté komunikují prostřednictvím IPC zpráv a pro přesun dat využívají sdílenou paměť. V praxi tyto činnosti zajišťují jaderné moduly DSPLINK a CMEM (popř. DSPBRIDGE). Tuto situaci zobrazuje následující schéma.



Obr. 14: Schéma komunikace mezi aplikací běžící na ARM a DSP

Modul DSPLink poskytuje aplikacím běžícím v uživatelském prostoru na procesoru ARM jednotnou sadu funkcí zajišťující komunikaci s DSP. Tyto funkce umožňují následující operace:

- Inicializace komunikace s procesorem DSP
- Načtení systému DSP/BIOS a aplikace do DSP
- Spuštění kódu na DSP
- Zápis a čtení do paměti procesoru DSP
- Ověřování úspěšného vykonání operací (spuštění aplikace, zápis apod.)

Na straně DSP běží operační systém DSP/BIOS (někdy také označován pouze názvem BIOS) poskytující podobnou funkcionalitu aplikacím běžícím na straně DSP. DSP/BIOS je proprietární RTOS vyvinutý firmou TI. Mezi hlavní udávané přenosti patří škálovatelnost, rychlost a nízké latence. S jeho využitím lze na DSP přepínat mezi několika běžícími aplikacemi s minimálním zpožděním. Toto přepíná se nejčastěji využívá při přehrávání videa, kdy jedna aplikace dekóduje obraz a další zvuk.

Aby bylo možné tuto funkcionalitu využít je potřeba, aby byla aplikace pro DSP napsána dle standardu XDAIS (eXpressDsp Algorithm Interoperability Standard).

Pro výměnu větších objemů dat slouží sdílená paměť. Jejím hlavím účelem je minimalizace kopírování,to by zbytečně zdržovalo práci obou procesorů.

Fyzické umístění této paměti se určuje při zavádění operačního systému argumentem mem. Adresa takto umístěné paměti je důležitá především pro aplikace běžící na straně DSP. U moderních OS aplikace pracují s virtuálními adresami a využívají pokročilé techniky jako je stránkování paměti apod. Z tohoto důvodu je pravděpodobné, že by při alokování paměti programem běžícím na straně ARMu byla paměť (z hlediska fyzického umístění) byla různě fragmentována. Tato situace by představovala jeden problém, další nepříjemností by byla skutečnost, že fyzická adresa alokované paměti by se různě měnila.

Z tohoto důvodu je proto nutné vybrat před startem systému umístění sdílené paměti tak, aby jej systém nevyužíval. O potřebnou alokaci paměťi se poté stará modul CMEM.

6 VÝVOJOVÉ NÁSTROJE

Následující část je věnována představení jednotlivých vývojových nástrojů využitých při implementaci knihovny AGG. Pro vývoj na straně DSP byl použit nástroj C6EZRun a pro vývoj na straně ARMu byl použit kompilátor CodeSourcery G++ Lite.

6.1 C6EZRun

C6EZRun je sada nástrojů, poskytovaná firmou Texas Instruments, určená ke zrychlení vývoje a prototypování aplikací které kombinují využití procesorů ARM a DSP u platformy OMAP. Hlavní část, tvořena kompilátorem TI C6000, slouží k překladu zdrojového kódu v jazyce C do objektového souboru pro DSP. Kromě samotného překladu se nástroje starají o implementace rozhranní DSPLink, využití sdílené paměti a tvorbu spustitelných částí souborů nebo knihoven.

V rámci C6EZRun tak existuje nástroj C6RunApp (určený pro tvorbu aplikací) a nástroj C6RunLib pro tvorbu knihoven.

6.1.1 C6RunApp

Tento nástroj lze využít v případě, že chceme aby naše aplikace běžela pouze na straně DSP. Výstupem C6RunApp je poté spustitelný soubor pro procesor ARM, který se postará o inicializaci DSP, načtení kódu do DSP a obsloužení případných systémových ze strany kódu běžícího na DSP.

Pro tyto účely se využívá nástroj s názvem c6runapp-cc. Při spuštení mu je jako parametr předán název souboru se zdrojovým kódem naší aplikace určené pro DSP.

V prvník kroku je zkompilován zdrojový kód aplikace a slinkován s částí knihovny C6run určené pro DSP. Výstupem této operace je spustitelný kód pro DSP ve formátu TI COFF. Tento kód se poté ještě minimalizuje pomocí nástroje strip6x.
Takto upravený kód je uložen v byte poli ve zdrojovém kódu tzv. C6run loaderu. Ten je poté přeložen standartním kompilátorem pro procesor ARM a tvoří spustitelný soubor.

Výsledný soubor pro ARM zajišťuje při spuštění Inicializaci C6Run loaderu. Tato operace zahrnuje následující kroky:

1. Inicilizace modulu CMEM.Pomocí CMEM je alokován zásobník o velikosti spustitelného kódu pro DSP. Aplikace poté tento kód do daného zásobníku překopíruje.
2. Inicilizace modulu DSPLink voláním funkcí (PROC_setup, PROC_attach, POOL setup, MSGQ setup).
3. Načtení spustitelného kódu pro DSP (PROC_load) do sdílené paměti
4. Spuštění DSP pomocí volání PROC_start
5. Uvolnění alokovaného místa pro zásobník voláním CMEM (po spuštění DSP už není zásobník potřebný)
6. Nastavení IPC pomcí fronty pro IPC zprávy
7. Vytvoření vlánka pro obsluhu IO operací ze strany DSP

Výsledná aplikace pro ARM tak využívá dvě vlákna. První vlákno kontroluje frontu zpráv a čeká na zprávu EXIT, která značí ukončení běhu programu na straně DSP. Druhé vlánko se stará o obsluhu IO požadavků ze strany DSP.

Po přijetí zprávy EXIT aplikace ukončí činnost DSP voláním odpovídající funkce v modulu DSPLink.

6.1.2 C6RunLib

C6RunLib slouží k vytvoření staticky linkované knihovny, kdy část kódu běží na procesoru ARM a kritická část na procesoru DSP. Výsledná knihovna může být poté využívána v aplikacích běžících na procesoru ARM obdobně jako je tomu u jiných knihoven.

Stejně jako u C6RunApp je nejdříve vytvořen objektový soubor pro DSP ze zdrojového kódu. Poté jsou pomocí specialního skriptu vybrány názvy použitých fukncí. Na jejich základě je vytvořeno rozhranní pro knihovnu na straně procesoru ARM. Tato část knihovny poté slouží k slinkování s ARM aplikacemi. Její inicilizační a řídící funkce jsou pak obdobné jako v případě aplikací z C6RunApp.

Součástí C6RunLib jsou dva nástroje:

- C6runlib-cc - se používá pro kompilování zdrojového kódu v jazyce C. Jeho výstupem je objektový soubor ve formátu C6000 a další pomocné soubory popisující rozhranní výsledné knihovny. Pří kompilaci je možné používat podobnou sadu přepínačů jako u klasického kompilátoru GCC
- C6runlib-ar - slouží k vytvoření knihovny z vygenerovaných souborů aplikace C6runlib-cc .

6.2 Kompilátor ARM

Pro kompilaci knihovny AGG pro procesor ARM byl využíván kompilátor CodeSourcery G++ Lite. Tento kompilátor je součástí sady nástrojů CodeSourcery ARM GNU/Linux toolchain vyvíjených firmou Mentor Graphics .

Tyto nástroje dovolují tzv. Křížovou kompilaci, která umožňuje kompilovat zdrojové soubory v jazyce C/C++ pro procesory ARM na procesorech x86. Využitím těchto nástrojů bylo docíleno oddělení vývojové a testovací platformy. Tento kompilátor zároveň podporuje EABI (embedded application binary interface), které je využíváno v systému Angstrom.

6.2.1 ABI

ABI (application binary interface) představuje popis nízkoúrovňového rozhraní mezi aplikacemi a knihovnami (případně OS). Cílem je definovat např. datové typy, jejich velikost,zarovnání, konvence pro volání funkcí (způsob předávání argumentů a návratových hodnot), čísla systémových volání apod.

Součástí toho popisu je i způsob reprezentace a zpracování čísel v pohyblivé řádové čárce. Protože některé procesory ARM neobsahjí matematický koprocesor je potřeba tuto skutečnost zohlednit před kompilací. V případě kompilátoru gcc pro procesory ARM je možné volit mezi třemi způsoby pomocí přepínače mfloat-abi.

Možné hodnoty mfloat-abi a jich význam

- soft – emuluje veškeré operace v pohyblivé řádové čárce
- softfp – využívá matematického koprocesoru, ale argumenty jsou předávány pomocí jednotky pro celočíselné argumenty
- hardfp - jsou plně využity možnosti matematického koprocesoru včetně předávání argumentů

V případě využití emulace FPU je spustitelný kód univerzálnější a lze je provozovat na větší škále zařízení. Výhodou hardfp je pak rychlejší kód pracující s čísly v pohyblivé řádové čárce.

7 IMPLEMENTACE KNIHOVNY

Celý proces implementace byl rozdělen na několik částí. První část spočívala ve zporoznění knihovny na procesoru ARM. Součástí tohoto kroku bylo vytvoření třídy agg_test_suite, která slouží pro testování scanline rasterizace a rasterizace primitiv.

Druhá část se zabývala možností využití některých specifických technologií pro ARM Cortex-A8 a DSP.

7.1 Testovací třída

Jak již bylo řečeno v úvodu, testovací třída agg_test_suite poskytuje rozhranní pro testování rasterizace. Součástí je několik testovacích metod, které využívají třídy z knihovny agg. Použitá verze knihovna agg je 2.5.

Třída se skládá z následujících metod:

- agg_test_suite(int width,int height)
- ~agg_test_suite()
- draw_circle(const circle_obj& circle)
- test_circles_norm(int count)
- test_fireworks()
- test_lines_norm(int count)
- test_primitives_dsp()
- test_primitives_neon()
- test_primitives_norm()
- test_triangles_norm(int count)
- write_to_file(const std::string &file_name)
- write_to_framebuffer()

7.1.1 Konstruktor a destruktor

Konstruktor slouží k alokaci paměti, která je využíváná jako kreslící plátno pro knihovnu. Tato paměť je tak dána vstupní výškou šířkou a velikosti formátu pro reprezentaci pixelů. Standartně je používána hodnota 8 bytů pro jednu složku barvy (24 bytů pro pixel). Úkolem destruktoru je poté tuto paměť uvolnit k dalšímu použití.

7.1.2 Testovací metody

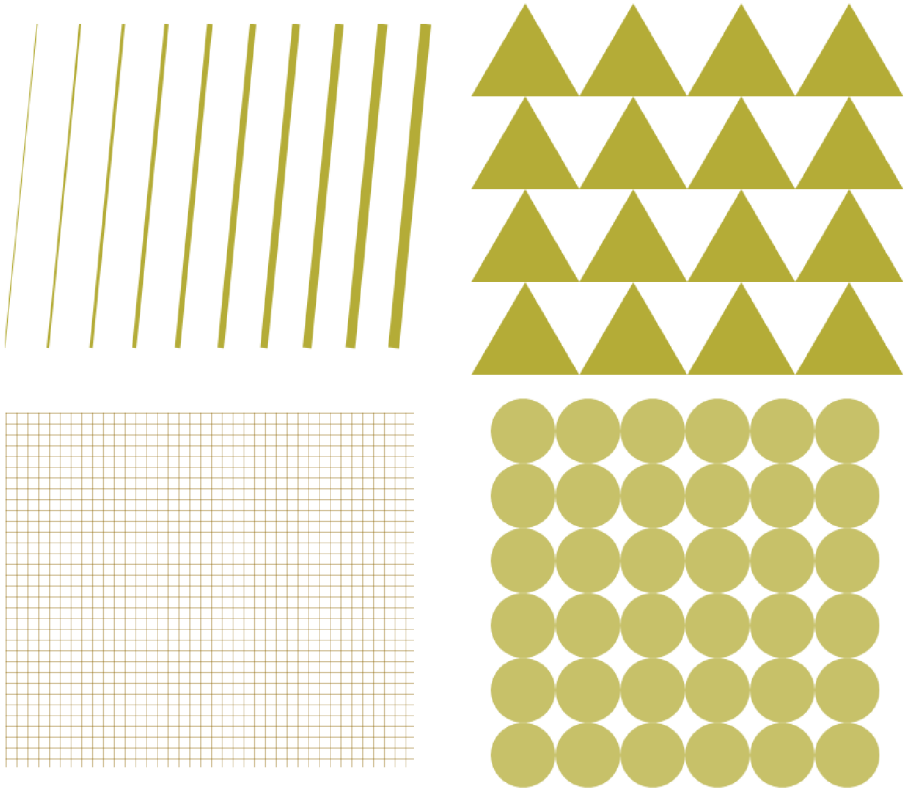
Metody draw_circles, test_circles_norm,test_fireworks a test_triangles_norm slouží k testování scanline rasterizace. Jednotlivé metody začínají deklarací potřebných částí knihovny jako je formát pixelů, určení velikosti alokovaného bufferu pro kreslení, apod.

Všechny metody také obsahují kód(využívající časovač systému) pro měření doby běhu jednotlivých funkcí nutných k rasterizaci. Výstupem těchto metod je ve formátu double a značí počet milisekund nutných k rasterizaci testu do paměťového bufferu.

Metody test_primitives slouží k testování algoritmů z třídy renderer_primitives. Protože metoda line, z třídy renderer_primitives obsahuje chybu, která způsobí dělení vstupních souřadnic hodnotou 256, stará se také o odstranění této chyby úpravou vstupních parametrů.

7.1.3 Ostatní metody

Metoda write_to_file slouží k zapsání obsahu bufferu do souboru ve formátu PPM. Metoda write_to_framebuffer přistupuje k zařízení /dev/fb0, které je součástí grafického subsystému procesoru OMAP. Jejím úkolem je překopírovat obsah paměťového bufferu do tohoto zařízení.



Obr. 15: Příklad vykreslených testů

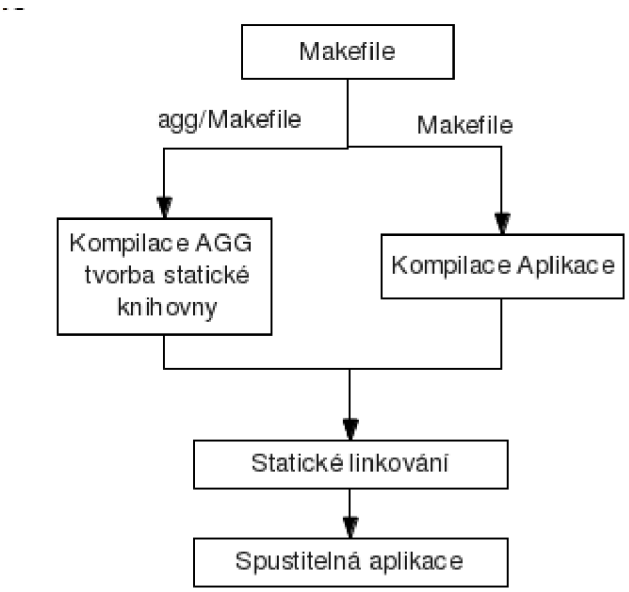
7.2 Implementace pro ARM

První část implementace se zabývala způsobem kompilace knihovny pro platformu ARM. Cílem bylo upravit soubory Makefile (sloužící ke kompilaci a sestavení

knihovny) tak, aby bylo možné využít křížové kompilace a zvolit vhodné nastavení parametrů překladače.

7.2.1 Překlad knihovny a aplikací

Z důvodu přehlednosti jsou jednotlivé součástí knihovny AGG rozděleny do řady souborů a složek. Pro překlad takto fragmentovaných částí je používán program make. Ten se postará, na základě informací v souboru Makefile, o volbu překladače, jeho vhodné nastavení a statické linkování knihovny. Následující schéma znároňuje tento postup včetně tvorby spustitelné aplikace využívající knihovnu AGG.



Obr. 16: Schéma sestavení knihovny a testovací aplikace

AGG obsahuje několik souborů Makefile. Při překladu je využíván Makefile ve složce src.Ten obsahuje názvy kompilovaných souborů a jejich umístění.Pro definici proměnných a nastavení překladače je pak využit soubor Makefile.in.Linux ve výchozím adresáři knihovny.

Obsah souboru Makefile.in.Linux:

AGGLIBS= -lagg

AGGCXXFLAGS = -O3 -march=armv7-a -mcpu=cortex-a8 -mfpu=neon -ftree-vectorize -mfloat-abi=softfp -I/usr/X11R6/include -L/usr/X11R6/lib

CXX = arm-none-linux-gnueabi-g++

C = arm-none-linux-gnueabi-gcc

LIB = ar cr

.PHONY : clean

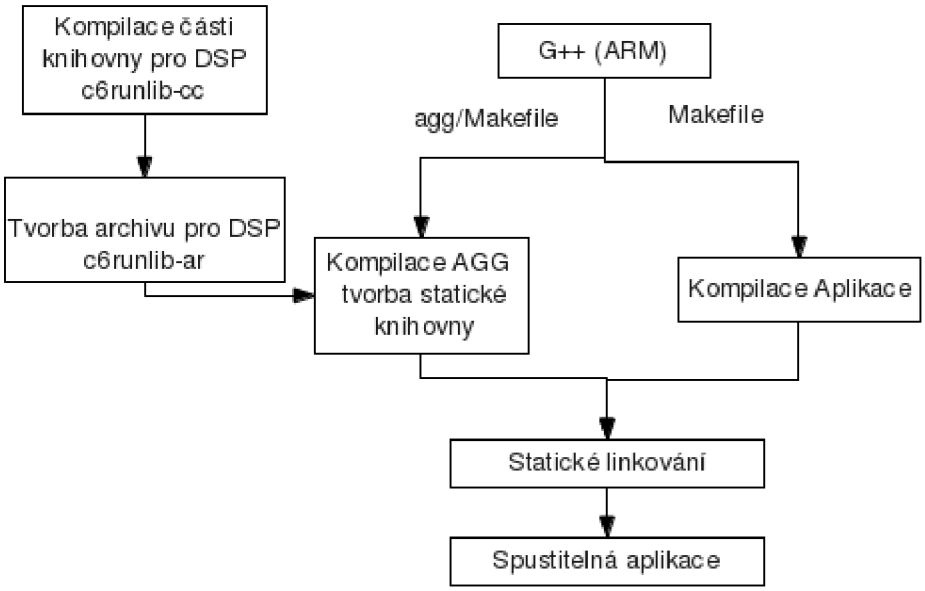
Mezi nejdůležitější proměnné tohoto souboru patří AGGXXFLAGS, která obsahuje nastavení překladače a CXX obsahující zvolený překladač.

Význam zvolených přepínačů překladače:

- -march=armv7-a -určení architektury procesoru
- -mcpu=cortex-a8 -jméno cílového proceru na jehož základě překladač volí použitou instrukční sadu
- -ftree-vectorize – překladač se pokusí přeložit kód, tak aby využíval instrukce SIMD (součástí je řada algoritmů pro úpravu kódu do vektorvé podoby
- -mfloat-abi – nastavení způsobu zpracování čísel v pohyblivé řádové čárce

7.3 Implementace pro DSP

Při implementaci na DSP byla využita technologie c6runlib. Cílem bylo otestovat rychlost kreslení čar v třídě renderer_primitives.
Tato třída byla zvolena z několika důvodů. Prvním důvodem byl fakt, že kompilátor TI C6000 dokáže překládat pouze zdrojový kód v jazyce C. Proto bylo potřeba vybrat vykreslovací algoritmus, který využíval co nejmenší množství dalších vrstev/tříd knihovny AGG. Dalším požadavkem bylo, aby vykreslování probýhalo přímo do bufferu umístěného ve sdílené paměti mezi procesory ARM a DSP.
Na základě těchto požadavků byla v knihovně cfft.h implementována verze Bresenhamova algoritmu pro kreslení čar a pomocná funkce pro volbu barvy pixelu. Program poté pomocí synchronního volání spustí funkci a předá jí umístění bufferu,rozměry a souřadnice čáry.
Následující schéma popisuje sestavení jednotlivých částí aplikace.



Obr. 17: Schéma sestavení knihovny a testovací aplikace

Před samotnou kompilací je nutné nastavit velikost,umístění a rozdělení sdílené paměti kterou bude využívat aplikace běžící na straně DSP.

Obsah konfiguračního souboru config.mak:
PLATFORM=beagleboard
C6RUN_INSTALL_DIR=C6Run_0_98_03_03
IPC=dsplink
DSPOS=dspbios5
GPPOS=linux
SHAREDMEM=cmem
DSP_REGION_BASE_ADDR=0x86300000
DSP_REGION_CMEM_SIZE=0x01000000
DSP_REGION_CODE_SIZE=0x00D00000

Pro sdílenou paměť je vyhrazeno 29MB RAM. Z toho je 16MB určeno pro sdílení dat mezi částí knihovny běžící na DSP a ARM. Zbylých 13MB je určeno pro program, zásobník, haldu apod. Nastavení kompilátoru musí být totožné s parametry které jsou předávány jádru systému při zaváděcí sekvenci. V opačném případě by pokus o zavedení knihovny do DSP skončil chybou.

7.3.1 Omezující faktory DSP

Využití této části knihovny je podmíněno dodržením několika nutných podmínek. První skutečností je fakt, že v případě využití technologie c6run nelze na DSP spouštět více programů naráz. To je dáno skutečností, že programy přeložené pomocí těchto nástrojů neimplementují standard XDAIS.
Dalším omezujícím faktorem je potřeba načtení modulů DSPLink a CMEM před spuštěním této knihovny a restartování DSP. Reset DSP je vyžadován především proto, že v cache paměti DSP může zůstat předchozí načtený program. V takovém případě může nastat situace, kdy starý program stále běží na straně DSP. To může způsobit vyvolání chyby při pokusu o načtení nového programu do DSP.

7.4 Testování knihovny

Pro praktické měření na procesoru OMAP byl vytvořen program aggtest s ovládáním přes příkazovou řádku. Program využívá třídu agg_test_suite a její metody pomocí kterých je vytvořen jednoduchý testovací scénář.
Před samotným měřením bylo, pomocí programu rrtest, určeno rozlišení použitého časovače na 0.319 ms.

Na základě takto naměřených dat byla sestavena tabulka udávající čas potřebný pro vykreslení jednoho testu.

	ARM [ms]	Thumb [ms]	Neon [ms]	ARM+DSP [ms]
Lines	22.003	22.094	21.802	-
Circles	56.118	57.521	57.057	-
Triangles	30.225	30.336	30.033	-
Primitives	1337.420	1337.322	1336.961	1850.257

Tabulka 1: Naměřené hodnoty jednotlivých testů programem aggtest

Jednotlivé položky v tabulce mají násjedující význam:

- ARM – při překladu knihovny byla použita instrukční sada ARM
- Thumb – pře překladu knihovny byla použita instrukční sada Thumb
- Neon – byl použit přepínač free-vectorize, který se naží upravovat kód pro lepší využití SIMD instrukcí
- ARM+DSP – jedná se o část knihovny s implentací kreslení čar na DSP

Výsledná velikost knihovny využívající instrukční sadu ARM byla 482 Kb. Pro sadu Thumb 412 Kb . Pří použití sady Thumb tedy došlo ke zmenšení výsledného souboru o 17% .

Při využití části knihovny na procesoru DSP byl naměřen nejpomalejší čas testu. Na této skutečnosti se podílí pravděpodobně několik faktorů. Prvním důvodem je pravděpodobně režie vznikající při komunikaci mezi procesory ARM a DSP. Druhým důvodem je skutečnost, že algoritmus na straně DSP není optimalizovaný pro využitý specializovaných SIMD instrukcí, které v řadě případů představují největší výpočetní přínos u toho typu procesorů.

8 ZÁVĚR

8.1 Dosažené výsledky

V průběhu práce byly analyzovány specifika procesorů OMAP a jejich možné využití pro grafické operace. Po prostudování knihovny AGG byly určeny rasterizační části a využité algoritmy. Na základě těchto zjištění se práce zaměřila na optimalizace knihovny na straně procesoru ARM a DSP.
Pro testování výkonu jednotlivých úprav v knihovně byla vytvořena testovací třída agg_test_suit a aplikace pro testování agg_test.

8.2 Další vývoj

Při pokusu o implementaci části knihovny na DSP se projevilo omezení nástrojů C6EZRun, které umožňují překlad pouze zdrojového kódu v jazyce C. Z tohoto důvodu se v ukázala idea implementovat celou knihovnu na DSP jako značně komplikovaná (vzhledem ke struktuře knihovny AGG).
Z hlediska zlepšení výkonu celé knihovny se tak jeví jako nejefektivnější úprava částí algoritmů do vektorové podoby tak, aby šlo využít technologie NEON.

Literatura

[1] Procedure call standard for ARM architecture [online] [cit. 23.5.2012] Dostupné na URL:<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0042d/index.html>

[2] ARM NEON support in the ARM compiler [online] [cit. 23.5.2012] Dostupné na URL:<http://www.arm.com/files/pdf/neon_support_in_the_arm_compiler.pdf>

[3] Cortex-A8 Technical Reference Manual [online] [cit. 23.5.2012] Dostupné na URL:< http://infocenter.arm.com/help/.../DDI0344D_cortex_a8_r2p1_trm.pdf>

[4] RealView Compilation Tools Assembly Guide [online] [cit. 23.5.2012] Dostupné na URL:< http://infocenter.arm.com/help/.../DUI0204I_rvct_assembler_guide.pdf>

[5] OMAP35x Technical Reference Manual (Rev. V) [online] [cit. 23.5.2012] Dostupné na URL:< http://www.ti.com/product/omap3530>

[6] POWERVR Series5 Graphics [online] [cit. 23.5.2012] Dostupné na URL:<http://www.imgtec.com/powervr/insider/docs/PowerVR%20Series5%20Graphics.SGX%20architecture%20guide%20for%20developers.1.0.8.External.pdf >

[7] The cl-aa algorithm [online] [cit. 23.5.2012] Dostupné na URL:<http://projects.tuxee.net/cl-vectors/section-the-cl-aa-algorithm >

[8] C6EZRun [online] [cit. 23.5.2012] Dostupné na URL:< http://processors.wiki.ti.com/index.php/C6EZRun>

Seznam příloh

Příloha 1. CD se zdrojovými kódy