Paralelní zpracování na úrovni dat

Břetislav Bakala

Důvody použití DLP

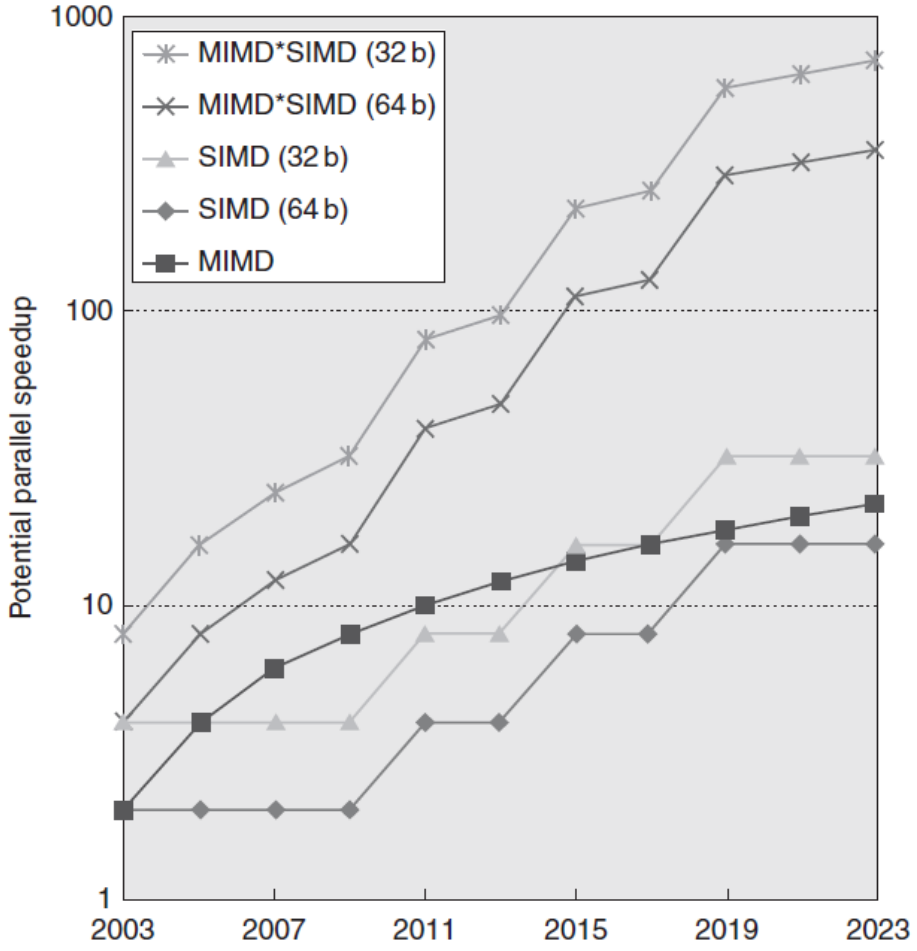
- Jak široká sada aplikací má významný paralelismus na úrovni dat (DLP)?
- Zpracování matematických výpočtů vědecké výpočetní techniky
- Mediální orientace zpracování obrazu a zvuku
- Jedna instrukce může spustit mnoho datových operací
- SIMD je potenciálně energeticky účinnější než MIMD
- SIMD je atraktivní pro osobní mobilní zařízení
- Programátor pokračuje v postupném přemýšlení a přitom dosahuje paralelního zrychlení tím, že má paralelní datové operace.

Varianty SIMD

- vektorové architektury - řízené provádění mnoha datových operací
  Náklady na počet tranzistorů, šířka pásma DRAM – cache.
  vektor je obecnější než multimediální SIMD už pro kompilaci, jsou nadmnožinou
- multimediální zpracování
  architektury x86 byly v roce 1996 zahájeny rozšíření o instrukce SIMD s
  MMX (Multimedia Extensions)
  několik verzí SSE (Streaming SIMD Extensions)
  AVX (Advanced Vector Extensions)
- SIMD v procesorech GPU - sdílejí prvky s vektorovými architekturami, oddělená grafická paměť a GPU mimo sytémový procesor a operační paměť. typ architektury za heterogenní

Vývoj zrychlení

Předpokládá se, že každé dva roky budou přidány dvě jádra na čip pro MIMD a počet operací pro SIMD se bude každé čtyři roky zdvojnásobovat.

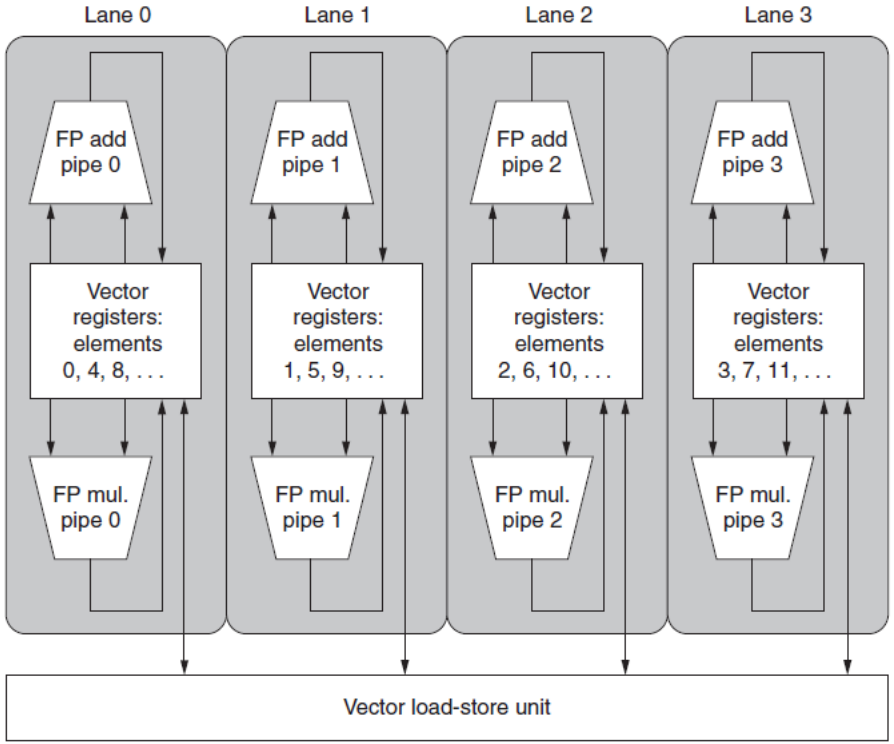


Vektorová architektura

- Vektorové architektury vyberou skupiny datových prvků rozprostřených v paměti, umístí je do velkých sekvenčních souborů registru, pracují s daty v těchto registrových souborech a pak rozešlou výsledky zpět do paměti.
- Jediný příkaz pracuje na vektoru dat, což vede k desítkám operací registr-registr na nezávislých datových prvcích.
- Soubory registru fungují jako vyrovnávací paměti řízené kompilátorem - skrytí latence paměti, využití propustnosti.
- Načítání a ukládání vektorů je plně zřetězeno a v programu proběhne dlouhé zpoždění při práci s pamětí pouze jednou při načtení nebo uložení celého vektoru (na rozdíl od samostatného načtení jednotlivých prvků) a tím ušetří např. zpoždění načtení 64 prvků.

Vektorová architektura

- Čtveřice linek zřetězených FP jednotek sčítání, násobení, čtení a zápisu



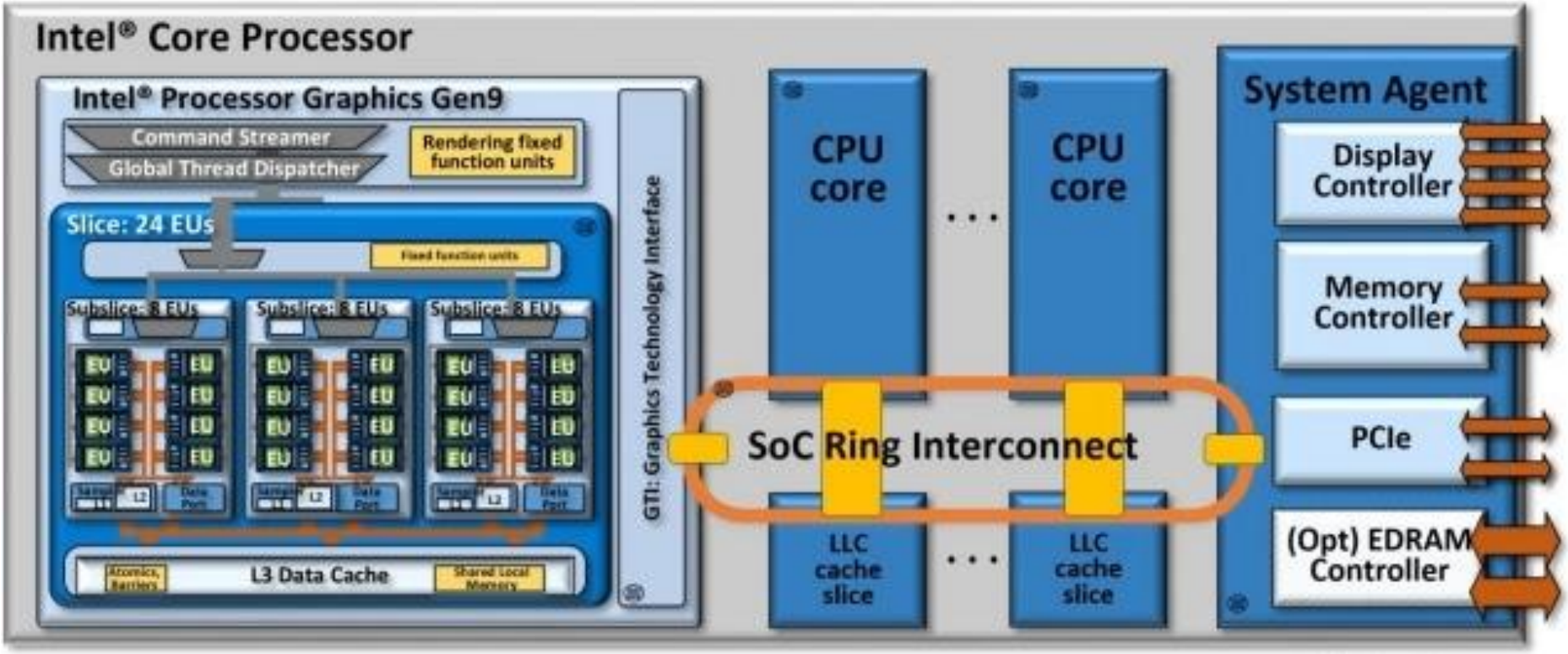
SIMD pro multimedia

- Multimedia pracují s menšími datovými typy (8bit/barva, 8bit průhlednost)
- Postačují menší soubory registrů na rozdíl od plně vektorových architektur
- Nepodporují podmíněné provádění nad jednotlivými prvky multimediálních vektorů
- Pevný počet datových operandů

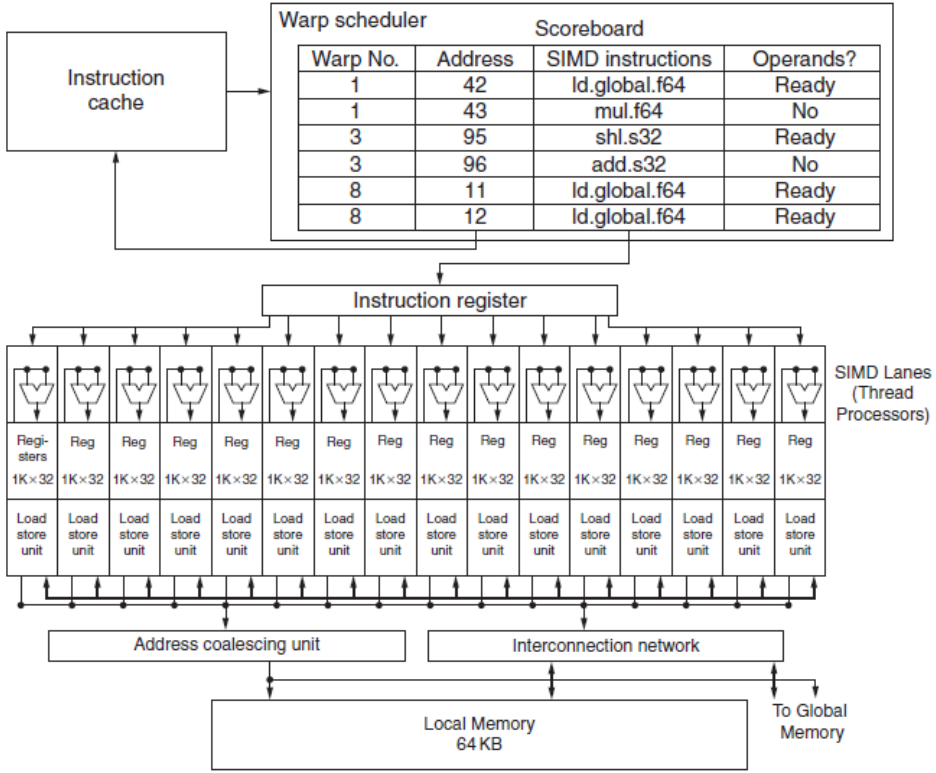
Architektura x86:

- instrukce MMX (Multi Media Extension,1996)
- SSE 1 až 4 (Streaming SIMD Extensions, 1999, 2001, 2004, 2007) čtyři single-precision floating-point operace nebo dvě paralelní double-precision operace. Přidání instrukcí pro navýšení multimediálních funkcí
- AVX (Advanced Vector Extensions, 2010) zdvojnásobení velikosti registrů na 256 bitů s možností až 1024 bitů

Graphics Processing Units - GPU

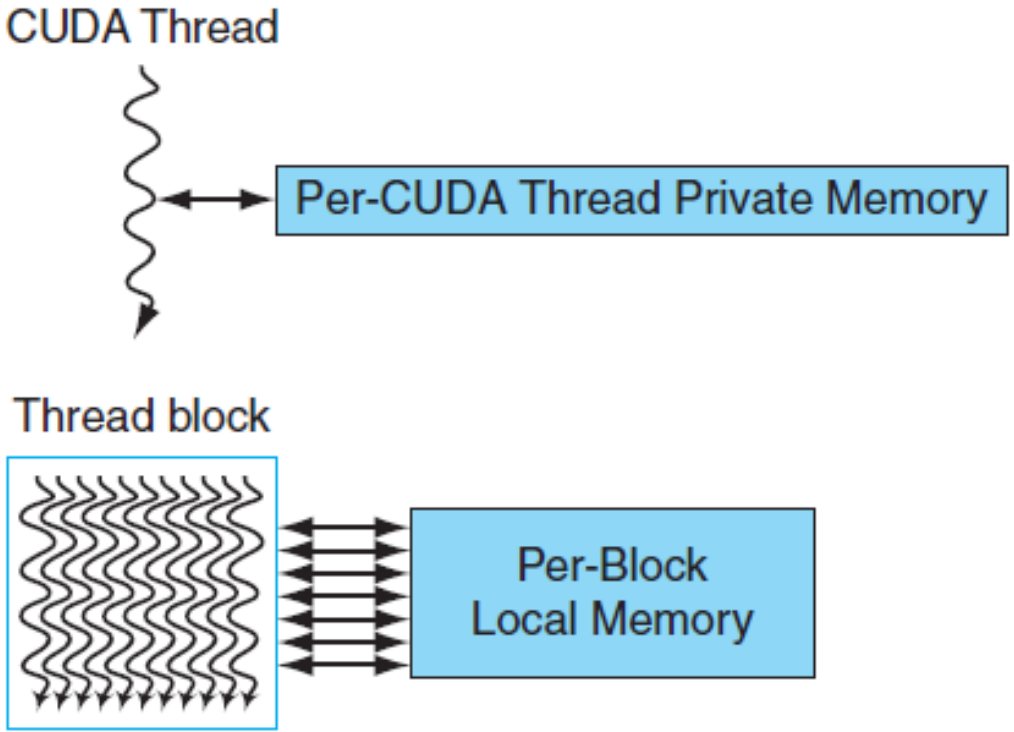


Vícevláknový SIMD Processor



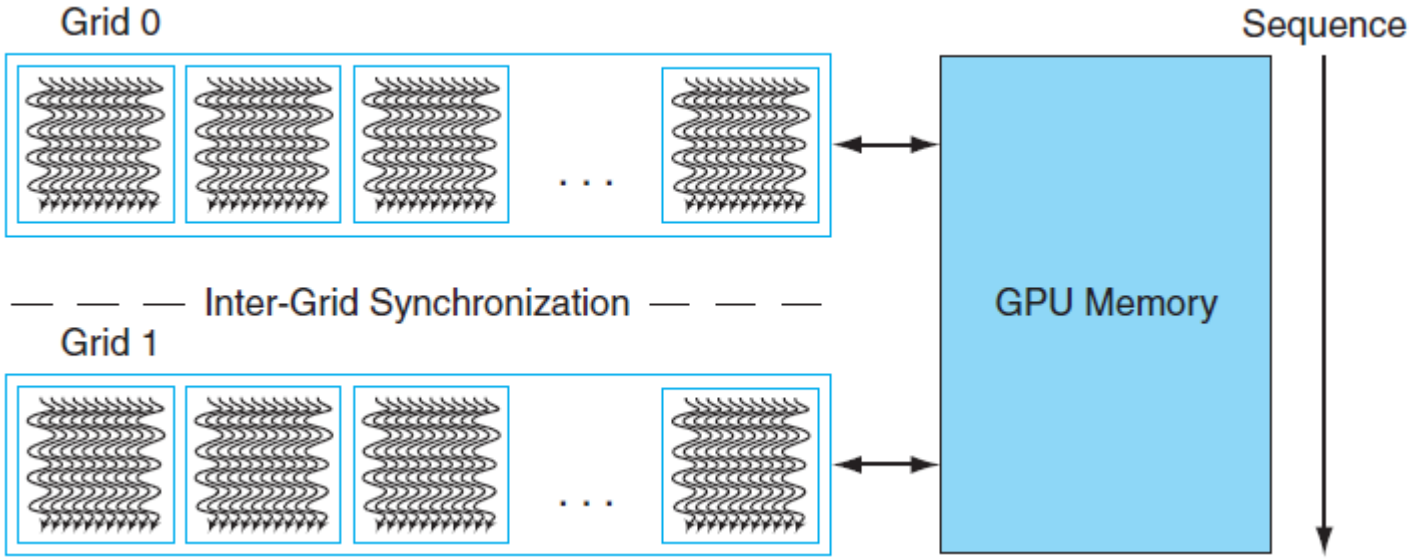
Struktura GPU paměti I.

- Každé vlákno má vlastní paměť na ukládání zásobníku, registrů a lokálních proměnných – paměť na čipu
- Vlastní lokální paměť pro blok podprocesů



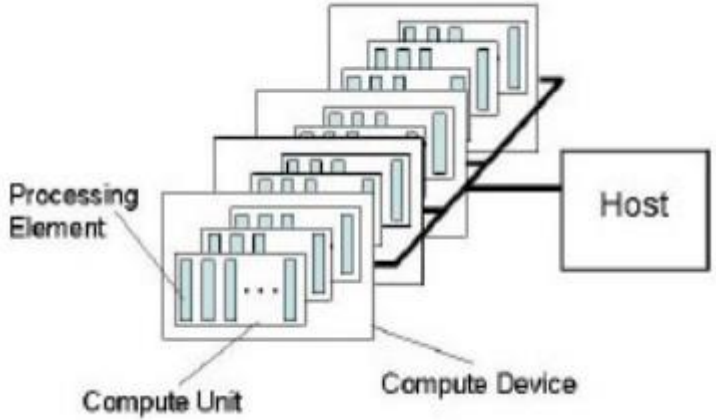
Struktura GPU paměti II.

- Off chip DRAM je sdílena celým GPU



Struktura GPU paměti III.

- Systémový procesor se nazývá Host a může číst nebo zapisovat do paměti GPU.
- Lokální paměť není pro hostitele přístupná
- GPU využívá velké množství vláken - lokálních registrů pro překlenutí zpoždění přístupu do DRAM



Host - kernel

- Program, který běží na hostiteli, pracuje se zařízeními pomocí API, které je poskytováno OpenCL standardem. To, co se spouští na zařízeních, je tzv. kernel, což je ta část programu, kterou často chceme paralelizovat.
- Samotné zařízení si můžeme představit jako soubor výpočetních jednotek, které obsahují jeden nebo vice procesních elementů. Zařízení může představovat např. grafická karta popřípadě více-procesorový systém a další.
- Základní jednotkou je vlákno (work-item). Každé vlákno kernelu zpracovává ten samý úsek kódu. Soustava vláken se stává pracovní skupinou (work-group). Vlákna v pracovní skupině pracují nezávisle na sobě, ale mají přístup ke společné pracovní skupinou sdílené paměti.

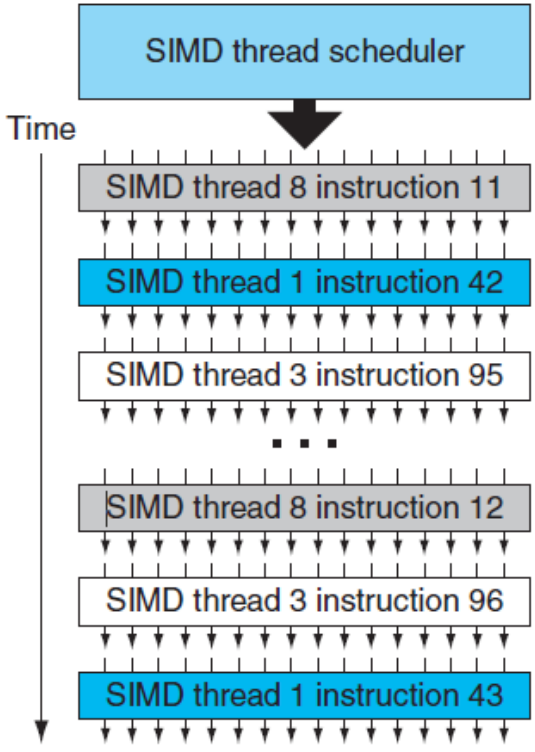
Host - kernel

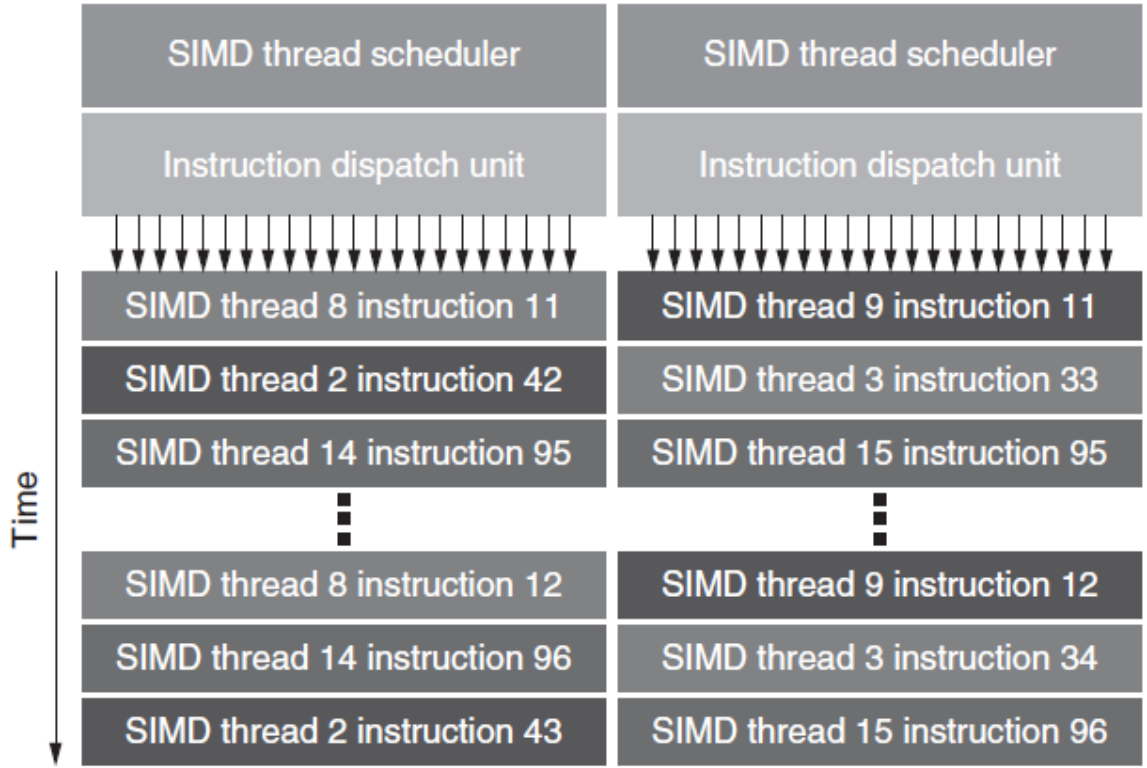
- Host program má za úkol zjistit si informace o platformě a zařízení. Tyto informace jsou poté využity pro vytvoření contextu, což je prostředí, které slouží k samotnému spouštění kernelu. **Context obsahuje informace o tom na jakých zařízeních má dojit ke spouštění, množství volné paměti, která je dostupná, dále správu paměti a harmonogram spouštění.** Po tom, co je context správně nastaven, je možné vytvoření programu. Program se skládá z jednoho čí několika kernelu, konstant či pomocných funkcí.

Architektura Fermi GPU

- Každé jádro má vlastní plánovač podprocesů a vydávání instrukcí
- Dvojitý plánovač vybírá dvě vlákna SIMD instrukcí do dvou sad 16 SIMD zřetězení, 16 load/store jednotek nebo 4 speciální funkce
- Vlákna jsou nezávislé a není nutná kontrola závislosti dat – obdoba vícevláknového vektorového procesoru, který může provádět vektorové instrukce ze dvou nezávislých podprocesů

Architektura Fermi GPU





Fermi GPU inovace

- Fast Double-Precision Floating-Point Arithmetic
- Cache v GPU čipu – skrytí zpoždění v DRAM paměti, L1datová a instrukční cache, L2 768 KB pro každé SIMD jádro GPU
- 64 bit adresace a Unified Address Space for All GPU Memories – zjednodušuje použití ukazatelů v programu

Fermi GPU blokové schema

- Každé vlákno má zřetězení pro výpočet FP, celočíselný výpočet, logiku pro pro práci s instrukcemi a operandy a zásobník mezivýsledků

- SFU – jednotky zvláštních funkcí provádí druhou odmocninu, převrácenou hodnotu, sinus, cosinus

Fermi streaming multiprocessor (SM)

Fermi GPU inovace

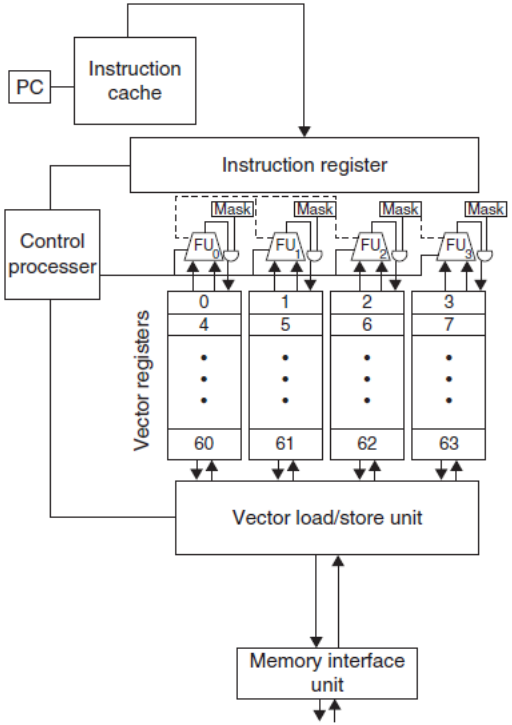
- Error Correcting Codes – detekce a oprava chyb v paměti a v registrech (ECC)
- Faster Context Switching – rychlejší hardwarová podpora pro přepínání kontextu (až 10x než předchozí modely)

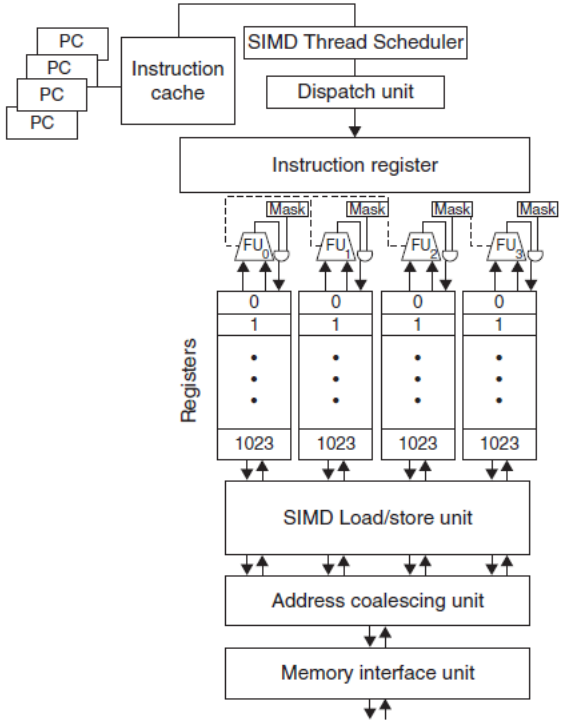
Vektorová architektura verzus GPU

- Obě architektury jsou navrženy tak, aby prováděly paralelní výpočty na úrovni dat, ale používají různé cesty
- Procesor SIMD se jeví jako vektorový procesor
- Více procesorů SIMD v GPU fungují jako nezávislé MIMD jádra
  Např. NVIDIA GTX 480 má 15 jader s HW podporou vícevláken, každé jádro má 16 vláknových linek
- Především využitím vícevlákna se liší GPU od většiny vektorových procesorů
- Soubor registrů o vektorového procesoru obsahuje souvislý blok 64 dvojic, u GPU je jediný vektor dodáván do registrů všech SIMD cest

Vektorová architektura verzus GPU

- Oba procesory mají 4 cesty (většinou v GPU je cest mnohem více)
- Instrukce Parallel Thread Execution PTX je nejblíž k instrukci vektoru





Vektorová architektura verzus GPU

Feature	Multicore with SIMD	GPU
SIMD processors	4 to 8	8 to 16
SIMD lanes/processor	2 to 4	8 to 16
Multithreading hardware support for SIMD threads	2 to 4	16 to 32
Typical ratio of single-precision to double-precision performance	2:1	2:1
Largest cache size	8 MB	0.75 MB
Size of memory address	64-bit	64-bit
Size of main memory	8 GB to 256 GB	4 to 6 GB
Memory protection at level of page	Yes	Yes
Demand paging	Yes	No
Integrated scalar processor/SIMD processor	Yes	No
Cache coherent	Yes	No

GPGPU frameworky

- GPGPU - General Purpose computing on Graphics Processing Units
  - Paralelní výpočty na GPU

- CUDA(Compute Unified Device Architecture)
  - jedná z nevýhod CUDA je závislost pouze na výrobci grafických karet - Nvidia

- OpenCL(Open Computing Language)
  - je otevřený standard

- DirectCompute - DirectX
  - Omezení pro platformu OS

OpenCL - Open computing language

- Představuje standard, který umožňuje programování heterogenních počítačových systémů - architektura paralelních procesorů v grafických kartách se hodně liší, ale díky abstrakci s nimi stále můžeme prostřednictvím OpenCL pracovat. (Abstrakce - umožňuje zjistit obecné vztahy a vlastnosti)
- Prvotní návrh OpenCL byl vytvořen firmou Apple.
- Specifikace byla vydána koncem roku 2008

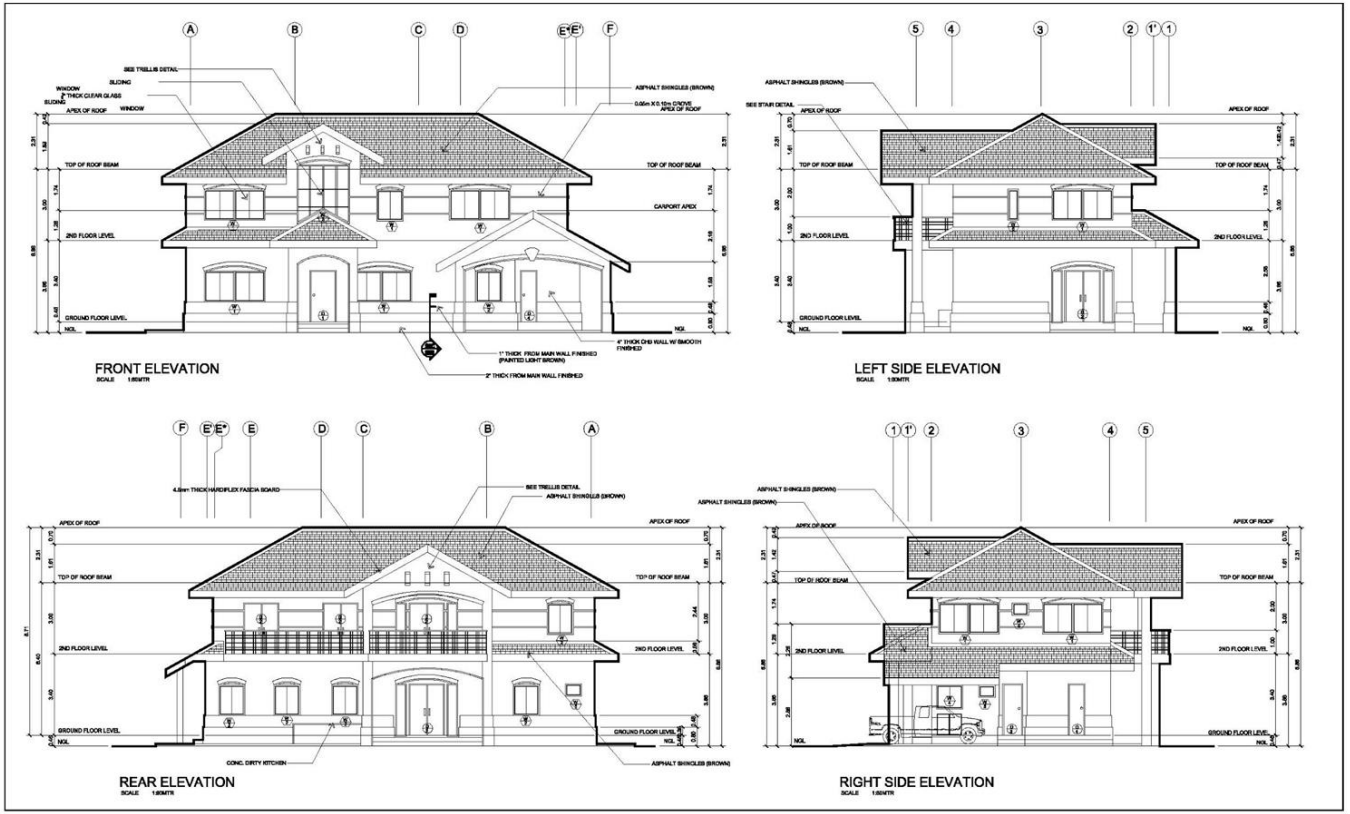
Využití GPU

- Zobrazování, renderování 3D grafiky
- Masivní paralelní výpočty
  - Simulace
  - Strojové učení
  - Zpracování videa a zvuku
  - Specifické úlohy (např. Bitcoin miner)

Pouze ilustrativní obrázek

Jednotlivé objekty scény jsou definovány obrazně řečeno “stavebními plány”, které definují jejich tvar v prostoru, textury, barvy atp.

CPU s těmito objekty pracuje a dále je předává grafické jednotce k jejich vizuálnímu zobrazení.

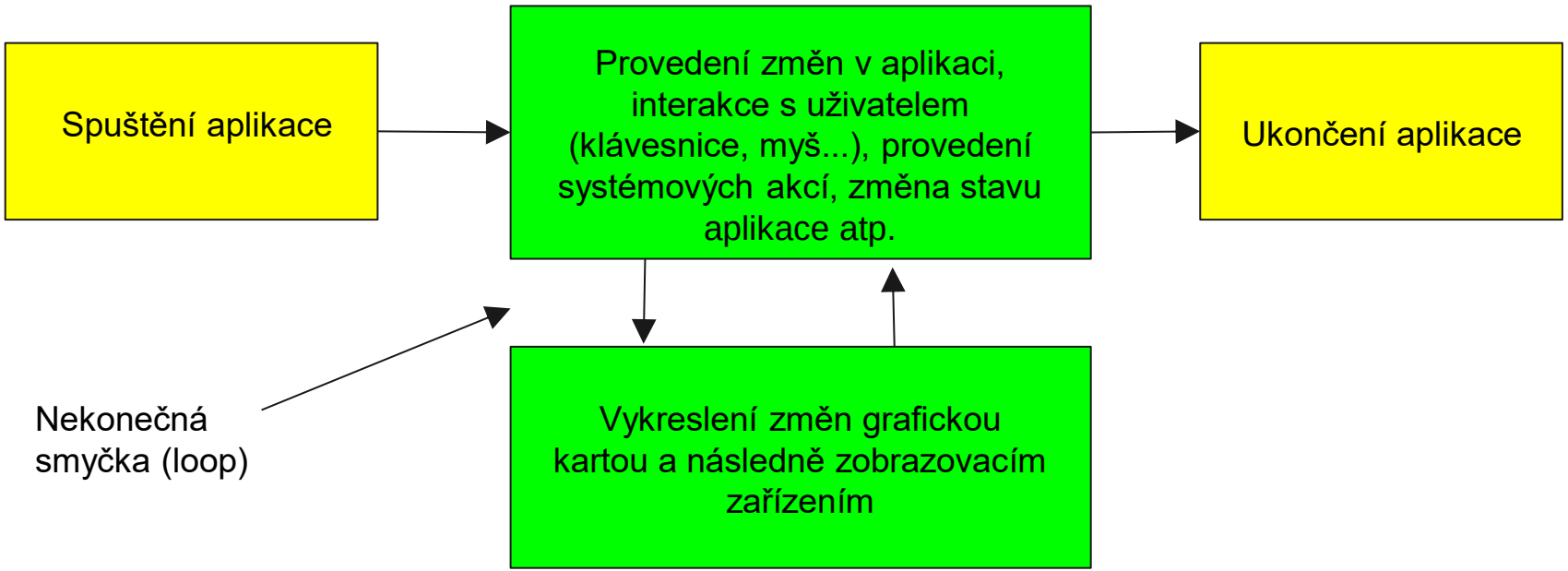


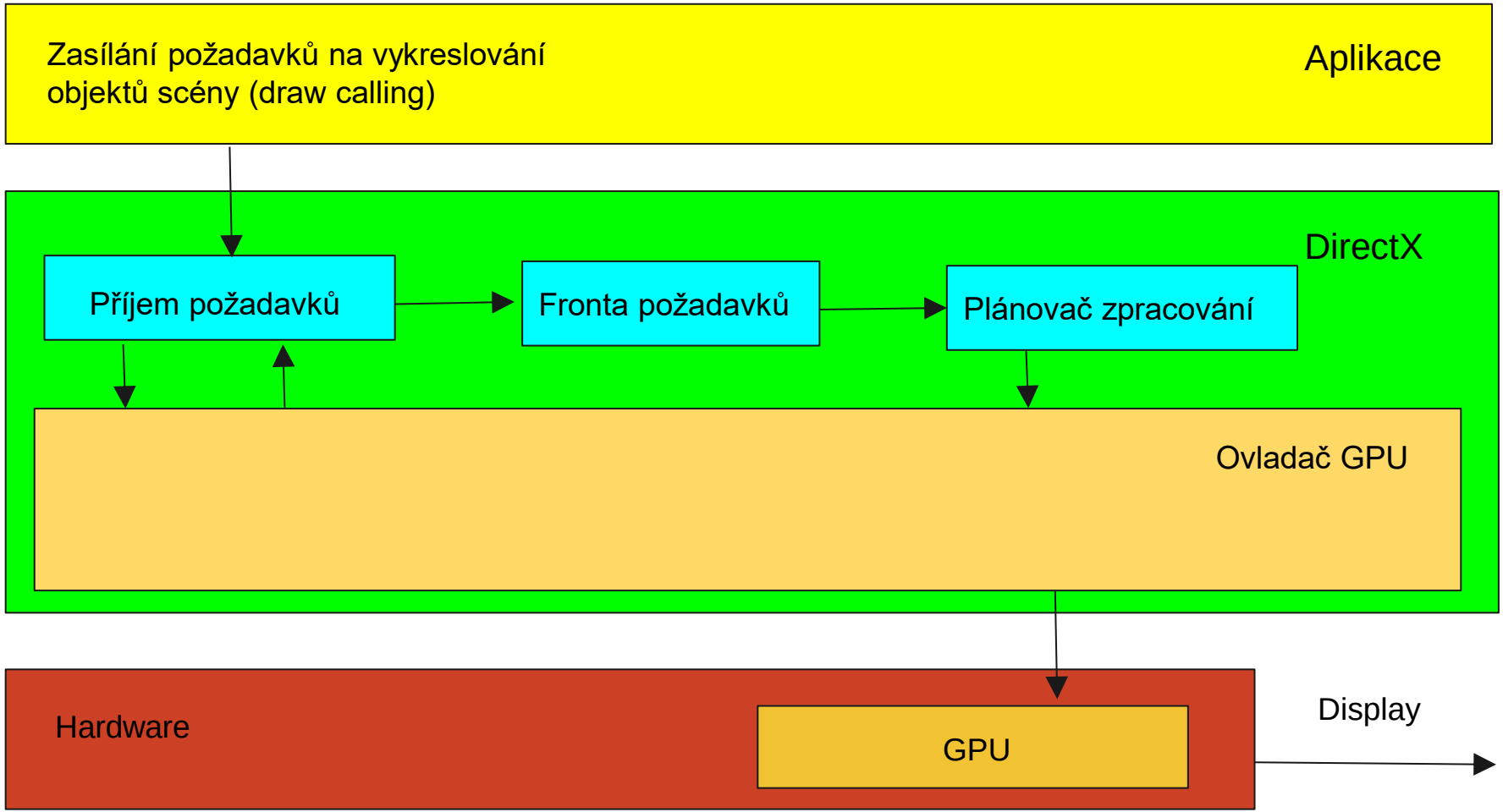

0
H
B
L
A

Výkon 3D - FPS (Frame per second)

- Udává počet snímků, které je grafická jednotka schopna v daný okamžik vygenerovat
- Počet reálně vygenerovaných snímků je závislý na složitosti renderované scény - množství a detailnosti jednotlivých 3D objektů

Základní schéma běhu aplikace




Zasílání požadavků na vykreslování objektů scény (draw calling)
Aplikace
DirectX
Příjem požadavků
Fronta požadavků
Plánovač zpracování
Ovladač GPU
Hardware
GPU
Display

DirectX I.

- DirectX je skupina softwarových rozhraní (API) určených pro zjednodušení programování multimediálních aplikací - obzvláště počítačových her pro operační systém Microsoft Windows.

- Umožňuje programátorovi velice rychlou komunikaci s dostupným hardwarem bez nutnosti znát specifické informace o každém zařízení. DirectX je tedy Microsoftem vytvořená abstraktní vrstva nad hardwarem, ke kterému výrobci zařízení dodávají ovladače s DirectX kompatibilní.

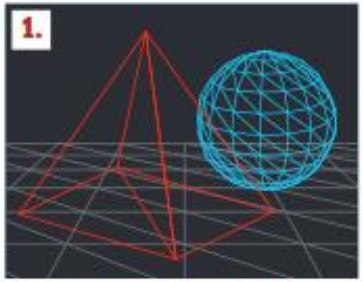
DirectX II.

- Grafické jednotky mohou být od různých výrobců a mohou tak používat pro vykreslení scény rozdílné příkazy a kódy.
- Proto bylo nutné vytvořit mezi samotnou aplikací a grafickou jednotkou ještě univerzální rozhraní, které by (kromě jiného) na svém vstupu od aplikace převzalo příkazy na vykreslení scény a podle druhu grafické jednotky by pak tyto příkazy převedlo do jazyka, kterému daná grafická jednotka (karta) rozumí a scénu nakonec zobrazí.

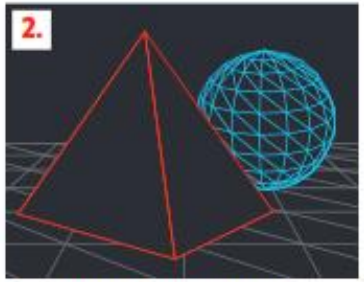
DirectX III.

- Implementován již do Windows 95
- Součásti DirectX:
  - Direct3D
  - DirectWrite - zobrazování typů písem
  - DirectSound3D - přehrávání 3D zvuků
  - DirectX Media - přehrávání multimédií; animace; interaktivní aplikace
  - DirectCompute - využití pro GPU výpočty
  - DirectX Media Objects - podpora pro streamování ….

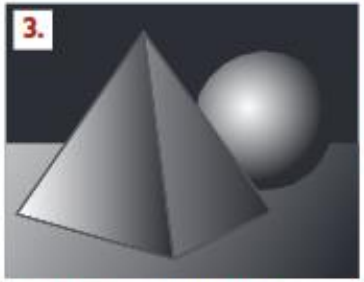
Grafická pipeline



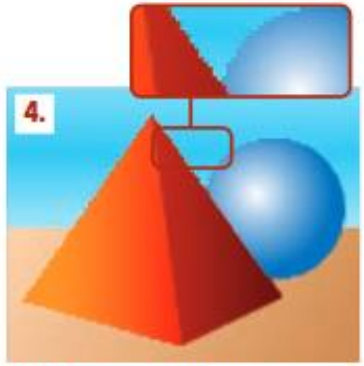
Vertex shader vytvoří ze souřadnic a vektorů 3D scénu.



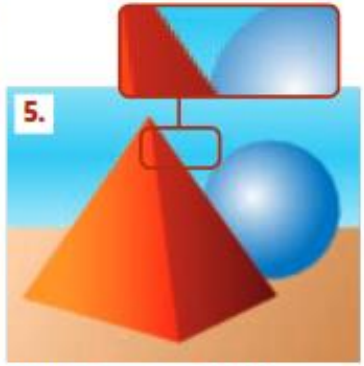
Neviditelné prvky jsou odstraněny ze scény.



Scéna je nasvícena jedním nebo více světelnými zdroji.



Pixel shader a texturovací jednotky obarví jednotlivé objekty scény.



Anti-aliasingem jsou vyrovnány hrubé a ostré hrany objektů.

(zjednodušené schéma)

Grafické zřetězení - pipeline II.

- Je základem každého 3D grafického čipu
- Představuje souslednost po sobě jdoucích funkcí prováděných pro vykreslení scény
- Na začátku do pipeline vstupují data aktuálních modelů scény, ta jsou po jednotlivých vrstvách upravována a na konci jsou ve výsledné podobě zpracována na úroveň grafického signálu pro monitor

Renderování 3D scény

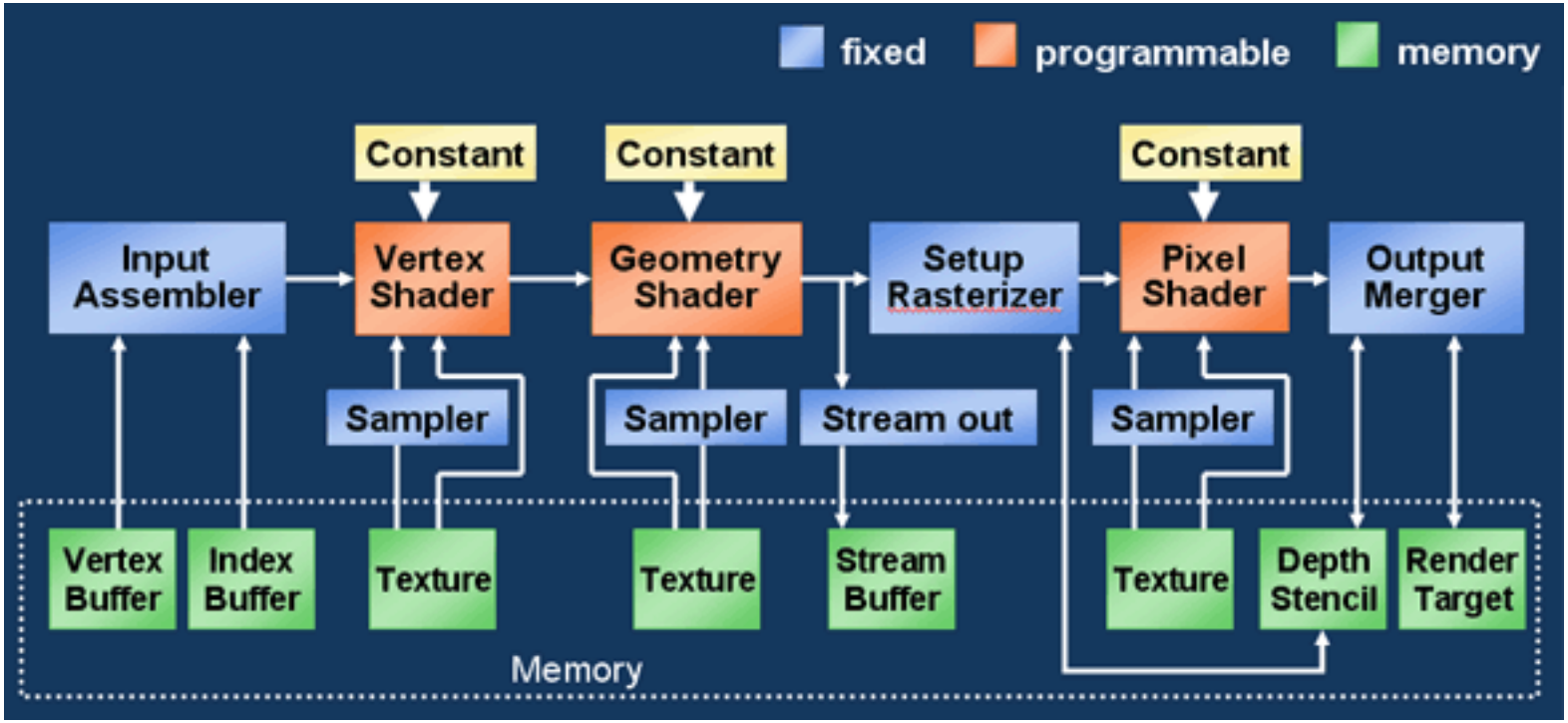
- Převzetí vstupních dat - modelů jednotlivých objektů
- Aplikace předá pokyn k překreslení scény
- Přípravnou fázi dat realizuje na grafické kartě tzv. Input assembler
- Základním stavebním prvkem každého 3D modelu je trojúhelník umístěný v prostoru resp. celý model je tvořen množstvím trojúhelníků
- Každý kompletně popsaný bod v 3D prostoru je dále nazýván **vertex** (vrchol).
- Každý vertex má celou řadu parametrů (umístění v prostoru, barvu, měřítko a umístění textury, osvětlení, vazbu na další vertexy)

Renderování 3D scény

- Vertex shadery - funkce - mohou měnit pozici jednotlivých vertexů
- Rasterizér - příprava scény - z vektorového modelu je vytvořen obraz, který bude v konečné scéně viditelný (převod 3D modelu na konkrétní 2D pohled)
- Pixel shadery - zjednodušeně řečeno pokrývají "kostru" našeho virtuálního modelu odpovídajícími materiály - včetně stanovení vlastností těchto materiálů (jako jsou druhy textur a jejich umístění, míra průhlednosti, použité efekty...).
- Složení výstupních dat pro zobrazovací zařízení

Anti-aliasing

- Technika, kterou jsou zjemňovány vizuálně zubaté okraje grafických objektů (vyhlazování)


fixed
programmable
memory
Constant
Constant
Constant
Input Assembler
Vertex Shader
Geometry Shader
Setup Rasterizer
Pixel Shader
Output Merger
Sampler
Sampler
Stream out
Sampler
Vertex Buffer
Index Buffer
Texture
Texture
Stream Buffer
Texture
Depth Stencil
Render Target
Memory