Počítačová grafika

Petr Horáček

Obsah

1 Úvod

Hry a grafické anymace jsou interaktivní filmové sekvence s předeme programovanými reakcemi na události. Každý snímek tohoto filmu je rendrován v reálném čase na základě uživatelské reakce, tak aby tvořil dojem realistického dojmu.

Architektura hry je běžně tvořena třemi hlavními částmi:

- **Inicializace hry**
- **Herní smička**
- **Ukončení hry**

Herní smyčka se následně skládá z následujících:

- **Převzetí uživatelského vstupu**
- **Kontrola interních časovačů**
- **Řízení autonomních herních botů**
- **Úprava herního stavu na základě uživatelského vstupu**
- **Vykreslení nového herního snímku**

2 Grafická karta

Grafická karta, často označovaná jako **GPU** (Graphics Processing Unit), je specializovaná elektronická součástka navržená pro rychlé a efektivní zpracování a vykreslování obrazu.

Zatímco CPU (Central Processing Unit) - hlavní procesor počítače - je skvělý v provádění široké škály úloh sériově (jedna po druhé), není ideální pro úlohy, které vyžadují obrovské množství paralelních výpočtů, jako je tomu u grafiky. Vykreslení jednoho snímku ve 3D hře může vyžadovat miliardy výpočtů pro každý pixel a každý vrchol modelu.

Grafická karta je proto navržena tak, aby tyto specifické grafické výpočty prováděla masivně paralelně. Má tisíce menších, specializovaných jader (na rozdíl od několika málo, ale velmi výkonných jader CPU), která dokážou zpracovávat mnoho grafických úloh současně. To je klíčové pro plynulé zobrazení složitých scén.

2.1 Hlavní komponenty Grafické Karty

- **GPU (Graphics Processing Unit)** - Samotný grafický procesor, který je srdcem karty. Provádí veškeré výpočty související s 2D a 3D grafikou, zpracováním videa, a v moderní době i obecnými výpočty (tzv. GPGPU - General-Purpose computing on GPUs).
- **Video paměť (VRAM - Video Random Access Memory)** - Velmi rychlá paměť, která je vyhrazena pouze pro GPU. Slouží k ukládání dat potřebných pro vykreslování - textur, modelů, shaderů, z-bufferů, framebufferů a dalších informací, ke kterým má GPU okamžitý přístup. Rychlost a kapacita VRAM jsou klíčové pro výkon, zejména ve vysokém rozlišení.
- **Video BIOS (VBIOS)** - Malý firmware, který inicializuje grafickou kartu při startu počítače a obsahuje základní informace o kartě.
- **DAC (Digital-to-Analog Converter) / Video Output Interfaces** - Moderní karty používají digitální výstupy, jako jsou HDMI, DisplayPort nebo DVI-D, které převádějí digitální signál z GPU na formát, kterému rozumí monitor. Starší karty mohly mít i VGA (analogový) výstup.
- **Chladicí systém** - Vzhledem k obrovskému množství tepla generovaného GPU a VRAM při intenzivních operacích, jsou grafické karty vybaveny ro-

bustními chladicími systémy - pasivními chladiči (heatsinky) a aktivními ventilátory.

- **Sběrnice (PCI Express)** - Fyzické rozhraní, které umožňuje grafické kartě komunikovat s ostatními komponentami počítače, zejména s CPU a systémovou pamětí. Nejčastěji se používá standard PCI Express (PCIe).

3 Grafické API

API - Aplication Programming Interface je obecně sada postupů, které přesně popisují přístup a ovládání určité softwarové komponenty. **Grafické API** je programové rozhraní mezi programovým kódem a konkrétní grafickou kartou v počítači, které popisuje jak obecně pracovat s grafickými kartami.

Na trhu existuje velké množství grafických karet různých výrobců a každá grafická karta má určitý způsob ovládání. To znamená, že pro použití dané grafické karty je třeba vytvořit specifický postup řízení v programu, který s touto kartou pracuje. Jakmile by se ale daný program použil na počítači s jinou grafickou kartou, došlo by tomu, že by program nefungoval správně, protože jiná karta vyžaduje jiný specifický způsob ovládání. Daný program tak není přenositelný na jiný stém a vyžaduje speciální hardwarovou konfiguraci. Aby grafické programy byly přenositelné mezi systémy s různými grafickými kartami vzniklo tzv. grafické API, které přesně popisuje jednotný způsob ovládání grafických karet.

Grafické API vytváří neviditelnou mezi-vrstvu mezi grafickou kartou a programem, který s grafickou kartou potřebuje pracovat. V této neviditelné mezi-vrstvě se pak skryje způsob jak se s konkrétní grafickou kartou pracuje a vytvoří se přesně pojmenované příkazy, které mají za úkol provést přesně danou činnost. To ve výsledku znamená, že pokud grafické API zpřístupní seznam operací které budou fungovat na libovolné grafické kartě, je možné z nich sestavit komplexní grafický program a není třeba se starat o to jakým způsobem to grafické API dělá.

3.1 Úkoly grafického API

- **Správa stavu** - nastavení a míchání barev
- **Správa paměti GPU** - Nahrávání vertex dat, textur, shaderů do paměti grafické karty
- **Vykreslování geometrie**
- **Programování shaderů**
- **Správu textur**
- **Nastavení viewporu a projekce** - Definování, jak se 3D scéna mapuje na 2D okno
- **Synchronizaci** - Zajištění správného pořadí operací mezi CPU a GPU

3.2 Příklady grafických API

Existuje více v současné době používaných grafických API. Je to důsledek historického vývoje jak grafického hardwaru tak požadavků uživatelů a konkurence softwarových společností jako je Apple a Microsoft a open source komunity. Každé z těchto grafických API má své pro i proti a jsou optimální pro určité případy použití. Mezi nejčastější grafická API patří:

- **OpenGL - Open Graphics Library** - Starší, ale stále široce používané API od Khronos Group. Je možné jej používat na všech současných platformách - Ms Windows, Apple a GNU Linux.
- **Direct3D** - Grafické API od Microsoftu, exkluzivní pro Windows a Xbox. Je součástí DirectX.
- **Vulkan** - Moderní, nízkoúrovňové, explicitní API od Khronos Group. Dává programátorovi mnohem větší kontrolu nad hardwarem a je navrženo pro moderní vícevláknové procesory a více GPU. Je multiplatformní (Windows, Linux, Android, částečně macOS přes MoltenVK).
- **Metal** - Grafické API od Applu, exkluzivní pro Apple platformy (macOS, iOS, iPadOS, tvOS). Velmi nízkoúrovňové, optimalizované pro Apple hardware.

3.3 Grafické API a herní engine

Grafické API přináší stále vysokou kontrolu nad použitým GPU. To znamená, že většina grafických operací jako je vykreslení základních geometrických objektů nebo texturování není přímočaré a skládá se z určité posloupnosti dílčích příkazů jejichž výsledkem je požadovaná akce. To činí programování grafických aplikací čistě pomocí grafického API zdlouhavé a technicky stále relativně náročné.

Aby vývoj grafických aplikací jako jsou photoshopy nebo hry bylo rychlejší a méně náročné na použití, jsou nad grafickým API postaveny grafické knihovny vyšší úrovně jako jsou například herní enginy. Grafické knihovny a enginy (jako Unity nebo Unreal Engine) výrazně zjednodušují práci s grafikou, protože fungují jako chytrá vrstva abstrakce nad samotnými grafickými API (jako OpenGL nebo Vulkan). Dělají to dvěma hlavními způsoby:

- **Zjednodušují složité operace** - operace, které by bylo nutné provést pomocí mnoha dílčích příkazů grafického API je skryto pod jedním komplexním příkazem grafické knihovny.

- **Abstrahují rozdíly mezi API a platformami** - různá grafická API stejnou věc umožňují realizovat jiným způsobem nebo jinými příkazy. Grafické knihovny umožňují skrýt tyto detaily a stejnou operaci můžou realizovat pomocí různých grafických api a uživateli umožňuje přepínat mezi tím jaké grafické API použít.

4 GAL

Grafická Abstraktní Vrstva (Graphics Abstraction Layer) nebo také Renderovací Abstraktní Vrstva (Render Abstraction Layer) je vrstva kódu, která leží mezi aplikačním kódem (např. herním enginem) a nízkoúrovňovými grafickými API (Vulkan, OpenGL nebo Direct X). Je to v podstatě rozhraní, které sjednocuje přístup k různým grafickým API, a umožňuje psát grafický kód jednou a nechat ho běžet na jakémkoli podporovaném API a platformě.

4.1 Důvody použití GAL

- **Multiplatformní kompatibilita** - Umožňuje spustit stejnou grafickou aplikaci na Windows (s Direct3D nebo Vulkan/OpenGL), Linuxu (s Vulkanem/OpenGL), macOS (s Metalem) nebo mobilních zařízeních, aniž by bylo nutné přepisovat renderovací kód.
- **Modularita a flexibilita** - Kód aplikace je oddělen od specifických detailů grafického API. Pokud se objeví nové API nebo bude potřeba podporovat starší hardware, stačí implementovat novou backendovou vrstvu pro GAL, aniž by se měnil zbytek enginu.
- **Zjednodušení vývoje** - GAL abstrahuje složité a často odlišné nízkoúrovňové detaily různých API, což vede k čistšímu a snadněji udržovatelnému kódu na úrovni aplikace.

4.2 Funkce GAL

Dobře navržená GAL by měla mít následující vlastnosti:

1. **Abstrakce klíčových grafických konceptů** - Měla by poskytovat jednotná rozhraní pro základní prvky grafického pipeline, které jsou společné pro všechna API:
   - *Zařízení (Device)* - Reprezentace grafické karty.
   - *Kontext (Context/Command Queue)* - Rozhraní pro odesílání vykreslovacích příkazů.
   - *Buffery (Buffers)* - Správa vertex dat, indexů, uniformních proměnných.
   - *Textury (Textures)* - Správa obrazových dat.

- *Shadery (Shaders)* - Abstraktní rozhraní pro programovatelné jednotky na GPU.
- *Pipeline State Objects* - Zahrnuje nastavení jako blending, depth testing, culling, které jsou v moderních API často sjednocené.
- *Draw Calls* - Příkazy k vykreslení geometrie.
- *Render Targets/Framebuffers* - Kam se vykresluje (na obrazovku, do textury).

2. **Nízká režie (Low Overhead)** - Měla by přidávat minimální výkonovou režii. To znamená, že by neměla provádět příliš mnoho dodatečných kontrol nebo konverzí, které by zpomalily běh programu. Cílem je být co nejblíže nativnímu výkonu základního API.
3. **Flexibilita a rozšiřitelnost** - Měla by umožňovat přístup k pokročilým nebo API-pecifickým funkcím, pokud je to potřeba. Někdy je nutné "prolomit" abstrakci a získat nativní handle na objekt (např. GLuint z OpenGL nebo VkDevice z Vulkanu), aby bylo možné volat specifické funkce daného API. To na druhou stranu může rozbít nebo zkomplikovat přenositelnost mezi různými grafickými API.

4.3 Abstrakce vykreslovacího řetězce

Grafické API jako OpenGL, Vulkan, Direct3D nebo Metal nabízejí odlišné přístupy k vykreslování grafiky. Přestože slouží stejnému účelu - vykreslení obrazu na obrazovku - jejich filozofie, způsob práce s pamětí, správou stavu a řízením průběhu vykreslování se výrazně liší.

4.4 Vykreslování

na vyšší úrovni vykreslovaní, to znamená v grafickém enginu se pak pro každý vykreslovaný objekt, ať už je to pixe, linka, trojúhelník nebo 3D objekt vytvoří objekt Mesh, který interně obsahuje vertex buffer. Objekt Mesh se následně interně uloží do pole vykreslovaných objektů, který je následně sekvenčně vykreslovaný.

5 Okenní systém

Grafické API (jako OpenGL, Vulkan, Direct3D nebo Metal) je sada instrukcí, které umožňují komunikovat s grafickou kartou (GPU). Popisuje, jak provádět grafické výpočty, operace a jak pracovat s grafickou pamětí. Samotné API neřeší, jak se výsledek těchto výpočtů zobrazí na monitoru nebo jak se vytvoří okno, do kterého se má vykreslovat. To je práce pro operační systém a jeho **okenní systém**.

Každý operační systém (Windows, Linux, macOS) má svůj vlastní okenní systém. Ten je zodpovědný za všechno, co je zobrazeno na obrazovce a s čím daný program interaguje:

- **Vytváření a správa oken** - Vytváří okna, ikony, tlačítka a veškeré uživatelské rozhraní.
- **Zpracování uživatelského vstupu** - Reaguje na kliknutí myši, stisky kláves, dotyky.
- **Správa grafických bufferů** - Přiděluje paměťové oblasti (buffery), kam mohou aplikace vykreslovat.
- **Komunikace s ovladači displeje** - Posílá hotové pixely z těchto bufferů na monitor.

5.1 Běžné okenní systémy

- **Win32 API (pro Windows)**
- **X11 (X Window System) nebo Wayland (pro Linux)**
- **Cocoa (pro macOS)**

5.2 Napojení grafického API na nativní okno

Než je možné cokoli zobrazit pomocí grafického API, je třeba nejprve vytvořit spojení mezi konkrétním grafickým API a oknem na daném systému. Tento proces se nazývá napojení nebo integrace (někdy se mluví o Window System Integration - WSI u Vulkanu).

Základní kroky propojení jsou pro všechna grafická API a okenní systémy podobné, liší se jen konkrétní API volání:

- **Vytvoření Nativního Okna** - Nejprve je třeba pomocí funkcí okenního systému vytvořit standardní systémové okno, se kterým může uživatel interagovat.
- **Získání "Vykreslovací Plochy"** - Z okna je třeba získat odkaz na jeho vykreslovací plochu (např. Device Context (HDC) na Windows, Drawable ID na X11 nebo wl_surface na Waylandu). Tato plocha je pro grafické API cílem vykreslování.
- **Nastavení Pixel Formátu** - Systému je třeba sdělit, jaké vlastnosti má mít tato vykreslovací plocha (např. barevná hloubka, podpora dvojitého bufferování, hloubkový a stencil buffer). Tím se připraví buffer, do kterého bude GPU kreslit.
- **Vytvoření Grafického Renderovacího Kontextu** - Na základě vybraného pixel formátu a vykreslovací plochy se ovladač grafické karty požádá o vytvoření grafického renderovacího kontextu. Toto je stavový stroj na GPU, který uchovává všechna aktuální nastavení pro vykreslování.
- **Aktivace Kontextu** - Sdělení operačnímu systému, že daný konkrétní grafický kontext je aktivní pro aktuální vlákno programu. Teprve pak systém propojí volání funkcí grafického API konkrétnímu oknu.
- **Načtení Funkcí API (Loadery)** - Jelikož většina moderních funkcí grafických API není přímo součástí systémové knihovny, je třeba je dynamicky načíst za běhu. Teprve poté je možné volat funkce grafického API.
- **Výměna Bufferů (SwapBuffers)** - Po dokončení vykreslování jednoho snímku do skrytého (back) bufferu, je třeba dát okennímu systému příkaz k prohození tohoto back bufferu s aktuálně zobrazovaným (front) bufferem. Tím se vykreslený snímek zobrazí na monitoru a zabrání se blikání (**Flickering**).

6 Grafická pipeline

Vykreslování 3D scény na 2D obrazovku je komplexní proces, který zahrnuje mnoho fází. **Grafická** nebo také **vykreslovací pipeline (Graphics pipeline)** je metaforické označení pro řadu kroků a operací, kterými prochází geometrická data (body, čáry, trojúhelníky) a další informace (barvy, textury) od chvíle, kdy opustí CPU, až po zobrazení výsledného pixelu na monitoru. Vykreslovací pipeline je abstraktní model, který popisuje, jak jsou 3D modely zpracovány a přeměněny na 2D obraz.

6.1 Jak fuguje grafická pipeline

Princip fungování grafická pipeliny spočívá v sekvenčním zpracování dat, kde výstup z jedné fáze se stává vstupem pro fázi následující. Data tečou jednosměrně. Některé fáze jsou pevně dané (fixed-function) a jejich chování je řízeno nastavením, zatímco jiné jsou programovatelné, což znamená, že jejich chování je definováno kódem, který se nazývá **shadery**.

V moderních grafických API (Vulkan, Direct3D 12, Metal) se kompletní stav grafické pipeliny (včetně všech shaderů a jejich nastavení) zapouzdří do jednoho objektu nazývaného **Pipeline State Object (PSO)**. To GPU umožňuje efektivnější práci, protože všechny potřebné informace jsou dostupné najednou.

6.2 Fáze grafické pipeliny

Vykreslovací pipeline se typicky dělí na několik klíčových fází, které lze kategorizovat na **geometrické fáze (vertex processing)** a **rasterizační fáze (pixel processing)**.

6.2.1 Geometrické Fáze (Vertex Processing)

Tyto fáze se zabývají zpracováním 3D geometrie a transformací vrcholů (bodů, které definují tvary).

- **Vstupní sestavovač (Input Assembler)** - Načítá data o vrcholech (pozice, barvy, texturové souřadnice atd.) a indexy z paměti GPU. Sestavuje z nich základní grafické primitivum, jako jsou body, čáry nebo trojúhelníky.

- **Vertex Shader** - Je volán pro každý jednotlivý vrchol. Jeho hlavní úkol je transformovat 3D pozici vrcholu z lokálního prostoru objektu do 2D prostoru obrazovky (přesněji "clip space") pomocí transformačních matic (model, view, projection). Může také počítat barvy vrcholů, transformovat normály pro osvětlení, nebo předávat další data do následujících fází. Jedná se o programovatelnou část grafické pipeliny.
- **Fáze teselace (Tessellation Stages - volitelná fáze)** - Tyto fáze (obvykle se skládají z Tessellation Control Shaderu, Pevné jednotky Tessellator a Tessellation Evaluation Shaderu) slouží k dynamickému generování geometrie. Dokážou z jednoduššího vstupu (např. patch) vytvořit mnohem detailnější síť trojúhelníků, často na základě vzdálenosti od kamery (blíž detailnější, dál hrubší). Jedná se o programovatelnou část grafické pipeliny.
- **Geometry Shader (volitelná fáze)** - Přijímá celá primitiva (body, čáry nebo trojúhelníky) a může je buď modifikovat, nebo generovat nová primitiva. Například z jednoho bodu může vytvořit celý trojúhelník nebo rozdělit trojúhelník na menší. Jedná se o programovatelnou část grafické pipeliny.
- **Klipování (Clipping)** - Odstraňuje části primitiv, které jsou mimo viditelnou oblast (frustum kamery). Tím se zajišťuje, že se zpracovávají a vykreslují jen ty části scény, které jsou skutečně viditelné.

6.2.2 Rasterizační Fáze (Pixel Processing)

Tyto fáze převádějí 2D geometrii na pixely a rozhodují o jejich finální barvě a vlastnostech.

- **Rasterizátor (Rasterizer)** - Vezme 2D primitiva (trojúhelníky) a převádí je na sadu fragmentů. Fragment je potenciální pixel, který leží uvnitř primitivy. Pro každý fragment rasterizátor také interpoluje data (např. barvy, UV souřadnice, normály) z vrcholů, aby byly hodnoty plynulé přes celý povrch.
- **Fragment Shader (Pixel Shader)** - Je volán pro každý jednotlivý fragment generovaný rasterizátorem. Zde se provádí většina vizuálních výpočtů: aplikují se textury, počítá se per-pixel osvětlení, provádí se komplexní efekty materiálu. Jeho výstupem je finální barva fragmentu. Jedná se o programovatelnou část grafické pipeliny.

- **Per-Fragment Testy a Míchání (Tests and blending)** - Poslední série testů a operací, které rozhodnou, zda a jak se fragment zapíše do barevného bufferu (framebufferu) a hloubkového bufferu.

7 Shader

Shadery jsou základním stavebním kamenem moderní grafické pipeline, bez kterých by současné 3D hry a vizualizace nebyly možné. **Shader** je v podstatě malý program napsaný ve speciálním programovacím jazyce, který běží na GPU. Na rozdíl od běžných programů pro CPU, které vykonávají instrukce sekvenčně, shadery běží na GPU paralelně na tisících, někdy i miliónech vláken najednou. Tento masivní paralelismus je klíčová pro rychlé zpracování obrovského množství grafických dat.

Primárním účelem shaderů je definovat, jak se má geometrie a barva objektů zobrazit na obrazovce. Mění pozice vrcholů, počítají barvy pixelů, simulují světlo a stín, a aplikují textury.

7.1 Účel shaderů

- **Realistické osvětlení a stíny** - Simulace, jak světlo dopadá na povrchy, jak se odráží a vytváří stíny.
- **Materiály a textury** - Aplikace obrázků na povrchy objektů (texturování) a definování vlastností materiálů (např. lesk, drsnost, průhlednost).
- **Speciální efekty** - Od ohně a kouře, přes vodu, déšť, sníh, až po pokročilé post-processing efekty (jako bloom, depth of field, motion blur).
- **Procedurální generování** - Vytváření geometrie nebo textur přímo na GPU bez nutnosti je předem ukládat.
- **Animace** - Deformace modelů pro animace (např. skinning, morphing)

7.2 Druhy shaderů v grafické pipelině

Moderní grafická pipeline se skládá z několika fází, přičemž některé z nich jsou programovatelné pomocí shaderů. Nejčastěji se setkáváme s těmito typy:

- **Vertex Shader (Vrcholový Shader)** - Shader, která má za úkol zpracovat každý jednotlivý vrchol předané geometrie. Provádí například transformaci pozic pro zobrazení na obrazovku a výpočet dat do další fáze.
- **Fragment Shader (Fragmentový Shader) / Pixel Shader** - Přebírá data z vertexovho shaderu a má za úkol zpracovat každý jednotlivý fragment (ekvivalent pixelu), který se má vykreslit na obrazovce. Má za úkol

vypočítat finální barvu pro daný fragment. To znamená aplikaci textur, výpočet osvětlení, odlesků a dalších materiálových vlastností.
- **Geometry Shader (Geometrický Shader)** - Volitelný shader pro pokročilé grafické techniky
- **Tessellation Shaders**

7.3 Programování shaderů v různých grafických API

Každé grafické API používá svůj vlastní jazyk ve kterém se popisují shadery. Například v OpenGL se používá jazyk GLSL (OpenGL Shading Language), v Metal je to MSL (Metal Shading Language) nebo v DirectX HLSL (High-Level Shading Language).

8 OpenGL

9 Vulkan