

GYMNÁZIUM JÍROVCOVA 8

MATURITNÍ PRÁCE

Programování grafické aplikace v C++

René Čakan

vedoucí práce: Dr. rer. nat. Michal Kočer

V Českých Budějovicích 6. února 2026

2025/2026

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně s vyznačením všech použitých pramenů.

V Českých Budějovicích dne podpis

René Čakan

Abstrakt

Klíčová slova

Poděkování

Obsah

Úvod	1
I Teorie k vývoji hry v OpenGL	2
1 Programovací jazyky	3
1.1 Programovací jazyk C	3
1.2 Programovací jazyk C++	4
2 Počitačová grafika	7
2.1 Historie počitačové grafiky	7
2.2 Grafické karty	11
2.2.1 CPU vs. GPU	11
2.2.2 Architektura a paměťová hierarchie GPU	12
2.2.3 Paralelismus	13
2.3 Grafická zobrazovací pipeline	15
2.4 Transformace	16
3 OpenGL	19
3.1 Historie OpenGL	19
3.2 OpenGL pipeline	19
3.3 Shadery	19
3.4 Textury	19
4 to do	20

II Vývoj hry v OpenGL	21
5 Použité knihovny	22
5.1 GLAD	22
5.2 GLFW	22
5.3 stb_image	22
6 Grafika a zvuk	23
6.1 Aseprite	23
6.2 Bosca Ceoil	23
7 Herní scény	24
7.1 Scéna hlavního menu	24
7.2 Scéna pozastavené hry	24
7.3 Scéna hry	24
8 Herní mechaniky	25
8.1 Generování objektů	25
8.2 Pohyb hráče	25
8.3 Detekce kolize	25
Závěr	26
Bibliografie	31
Zkratky	32
Přílohy	35
A Fotky z pokusů	36
B Příloha další	37

Úvod

Část I

Teorie k vývoji hry v OpenGL

1 Programovací jazyky

1.1 Programovací jazyk C

C je středněúrovňový programovací jazyk, tedy jazyk, který je podobou blízko strojovému kódu, ale má už prvky vyššího programovacího jazyka jako jsou funkce, datové struktury nebo to že je strukturovaný. Je komplikaný a statický, což znamená, že se program musí nejdříve přeložit do strojového kódu a až pak se může spustit. Datové typy jsou známy v čase komplikace, proto všechny proměnné musí být v kódu deklarovány, jelikož vkládání vstupních dat do programu probíhá až při běhu programu. Programuje se v něm strukturovaně a procedurálně, tedy kód se píše pomocí řídících struktur (if, while, for atd.) a pomocí funkcí, které umožňují používat části kódu vícekrát. C nemá automatický správce paměti, takže je potřeba uvolňovat paměť manuálně. C má střídmou standardní knihovnu, která obsahuje základní matematické operace a funkce pro práci s pamětí a soubory, takže jakékoli složitější datové struktury či funkce si člověk musí naprogramovat sám. Tato strohost a blízkost ke strojovému kódu z C dělá jeden z nejrychlejších programovacích jazyků. [25, 23, 15, 32, 34]

C bylo vytvořeno Dennisem Ritchiem na počátku 70. let 20. století v AT&T Bell Labs. Jeho předchůdci byly jazyky ALGOL, CPL, BCPL a B. Jeho prvním účelem bylo přepsat operační systém UNIX do použitelnějšího jazyka než Assembly a B. Už koncem 70. let bylo C populární, ale nebylo standardizované a vznikalo mnoho různých variant. Na začátku 80. let tedy American National Standards Institute (ANSI) zahájil práci na formální standardizované verzi. Tu dokončili v roce 1989 a je známa pod jménem C89. V průběhu let vycházely další verze, které jazyk zlepšovaly a modernizovaly. Nejdůležitější verze byly C99, C11 a C17. Norma C23 byla nedávno schválena a teď se implementuje do komplikátorů. V součastnosti mezi nejpoužívanější komplikátory patří GCC, Clang a MSVC. Jelikož bylo C velice populární, ovlivnilo řadu jiných programovacích jazyků, jako C, C#, Java, Rust, Go atd. [4, 18, 25, 23]

C je univerzální programovací jazyk, má tedy širokou škálu využití. Jeho první využití

bylo k napsání UNIXu, který později ovlivnil operační systémy jako Linux, macOS, iOS a Android. Používá se v programování softwaru s omezenou pamětí a výkonem, jako je firmware aut či v zařízeních chytrých domácností. Dále se využívá pro tvoření kompilátorů a interpreterů jako je GCC nebo interpreter Pythonu. Také jsou v něm napsané systémové databáze MySQL a Oracle Database. Kvůli jeho rychlosti jsou v něm napsané knihovny pro jiné programovací jazyky jako je NumPy, OpenGL či GLFW. [4, 25, 47]

Jednoduchý program, který načte ze vstupu počet čísel, která chce uživatel seřídit. Následně daná čísla načte a vytiskne je seřazená:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int compare(const void *a, const void *b)
5 {
6     return (*(int *)a - *(int *)b);
7 }
8
9 int main(void)
10 {
11     int sizeOfArray;
12     scanf("%d", &sizeOfArray);
13     int *array = malloc(sizeOfArray * sizeof(int));
14
15     for(int i = 0; i < sizeOfArray; ++i)
16         scanf("%d", &array[i]);
17
18     qsort(array, sizeOfArray, sizeof(int), compare);
19
20     for(int i = 0; i < sizeOfArray; ++i)
21         printf("%d\n", array[i]);
22
23     free(array);
24     return 0;
25 }
```

Zdrojový kód 1.1: sort_n_numbers.c

1.2 Programovací jazyk C++

Programovací jazyk C++ je v mnoha ohledech podobný jazyku C. Je stejně jako C středně-úrovňový, komplikovaný, statický, má datové typy známé v době komplikace a nemá au-

tomatický správce paměti. V C++ se také programuje strukturovaně a procedurálně, ale na rozdíl od C, také umožňuje programovat objektově. Objektové programování umožňuje používat objekty, které jsou instance tříd. Tyto třídy umožňují dědičnost, polymorfismus a zapouzdření, což dělá kód přehlednější a usnadňuje budoucí rozšiřování a debuggování. Dalším rozdílem je standardní knihovna, kterou má C++ rozsáhlejší. Obsahuje nové kontejnery jako vector, map, a priority_queue, které jsou tvořeny pokročilejšími datovými strukturami jako binární vyhledávací strom nebo heap. Dále obsahuje nové algoritmy, například sort, find nebo count. Kvůli velké podobnosti C a C++ se často může C kód používat v C++, ale není tomu tak vždy. Například tento kód:

```
1| int class(int new, int bool);
```

Zdrojový kód 1.2: incompatibility_example.c

V C tento kód vytvoří funkci class, která vrací int a má dva parametry new a bool. V C++ jsou ale class, new a bool klíčová slova, která nelze použít v názvu proměnných a funkcí. Pokud chce programátor napsat C kód, který se bude jednoduše v C++ programech, doporučuje se programovat v C tak, aby daný C kód byl podmnožinou C++. [46, 45]

C++ bylo vytvořeno Bjarnem Stroustrupem v roce 1979 v AT&T Bell Labs. Před vytvořením C++ pracoval Stroustrup s programovacím jazykem Simula 67, který byl objektově orientovaný a sloužil primárně k vytváření simulací. Stroustupovi přišlo objektově orientované programování velmi užitečné, ale Simula 67 byl příliš pomalý pro větší projekty. Rozhodl se vytvořit nadmnožinu jazyka C, která by umožňovala objektově orientované programování a zároveň si zachoval rychlosť C, s názvem C with Classes. V roce 1982 byl Stroustrup se stavem C with Classes zklamán. Nepřišlo mu, že oproti C přináší významné zlepšení a rozhodl se jazyk dále vylepšovat nad rámec objektově orientovaného programování. V roce 1983 byl jazyk přejmenován z C with Classes na C++. Dále bylo C++ v roce 1985 oficiálně vydáno a začalo se používat komerčně. V roce 1998 byla vydána první standardizovaná verze s jménem C++98. Další významné verze, které jazyk modernizovaly a přidávaly mu nové funkce, byly C++03, C++11, C++14, C++17, C++20 a nejnovější verze C++23. C++ se kompiluje pomocí stejných kompilátorů jako C, tedy GCC, Clang a MSVC. [45, 44, 3, 51]

C++ je stejně jako C univerzální programovací jazyk, a využívá se v široké škále odvětví. První využití je ve videoherním průmyslu. V C++ jsou napsané populární hrací enginy jako Unity nebo Unreal Engine. Dále se v něm vytváří aplikace jako Photoshop nebo Blen-

der. Využívá se v částech operačních systémů jako Apple macOS nebo Microsoft Windows OS. Dále se využívá při vytváření internetových prohlížečů, například Firefox nebo Google Chrome. C++ se využívá i ve vědě, například v CERNu nebo v NASA. [46, 16, 43]

Program se stejnou funkcí jako z kapitoly o C, ale napsán v C++

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 int main()
6 {
7     int sizeOfArray;
8     std::cin >> sizeOfArray;
9     std::vector<int> arr(sizeOfArray);
10
11    for(int i = 0; i < sizeOfArray; ++i)
12        std::cin >> arr[i];
13
14    std::sort(arr.begin(), arr.end());
15
16    for(int i : arr)
17        std::cout << i << '\n';
18
19    return 0;
20 }
```

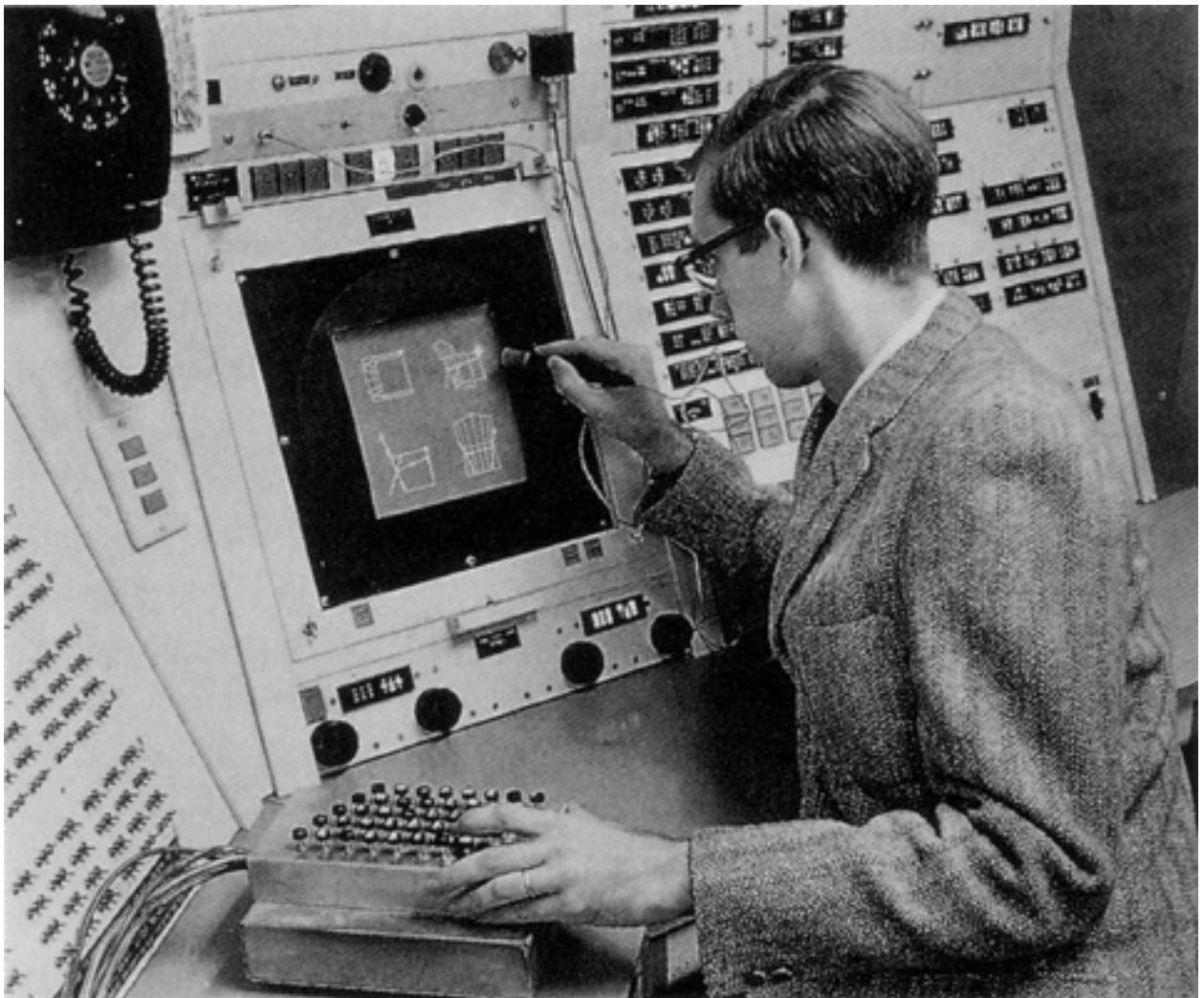
Zdrojový kód 1.3: sort_n_numbers.cpp

2 Počitačová grafika

2.1 Historie počitačové grafiky

Není úplně jasné, který počítač jako první využíval počitačovou grafiku, ale začnu počítačem Small-Scale Experimental Machine (SSEM). Tento počítač byl vytvořen v roce 1948 na Manchesterské univerzitě a jeho tvůrci byli Frederic C. Williams, Tom Kilburn a Geoff Tootill. Tento počítač byl první počítač s uloženým programem, tedy počítač, který měl svůj program uložen ve stejné paměti jako data, se kterými počítač pracoval. Data uchovával na katodové trubici (CRT), které se později začalo říkat Williamsova trubice. Tato trubice si dokázala pamatovat až 2048 bitů, které byly uchovávány jako elektrické náboje. Součástí trubice byl i displej, který na svém fosforovém povrchu promítal oblasti s nábojem. Takto vznikl první počítač s digitálním displejem. [38, 33]

Dalším zajímavým počítačem byl Whirlwind. Americký Office of Naval Research a U.S. Air Force chtěli vytvořit počítač, ve kterém by dokázal běžet letecký simulátor. Tak tedy v roce 1947 začal Jay Forrester pracovat v laboratořích MIT na projektu Whirlwind. Při práci vyvinul Forrester paměť s náhodným přístupem (RAM), tvořenou magnetickými jádrami, skrz které proudil koincidenční proud. Počítač byl dokončen v roce 1951. Využíval CRT na zobrazování výsledků podobně jako SSEM a dokázal na svém displeji řešit rovnice, později i simulovat karetní hru blackjack. Následně se projekt Whirlwind stal součástí projektu Semi-Automatic Ground Environment (SAGE). Ten měl za úkol vytvořit počitačový systém, který by pomocí radarů dokázal odhalovat letadla a řídit obranné síly proti případným letadlům. Projekt SAGE byl jeden z prvních systémů, které využívaly interaktivní ovládání pomocí klávesnice či speciálního světelného pera. Pokud bylo perem namířeno na ikonu letadla, zachytilo světlo z displeje a počítač zobrazil informace o daném letadle jako rychlosť a směr jeho letu. Dalsí počítač, který navazoval na projekt Whirlwind a SAGE, byl počítač TX-2, který byl vytvořen Wesem Clarkem na MIT. Ten byl na rozdíl od Whirlwindu tranzisto-



Obrázek 2.1: Skatchpad Ivana Sutherlanda [22]

rový. Zajímavým projektem, který tento počítač umožnil, byl Sketchpad vytvořen Ivanem Sutherlandem. Tento program byl první, který umožnil interaktivně kreslit na obrazovku. Obrazovka byla velká 7x7 palců s rastrem 1024x1024 bodů a psalo se na ní perem, které zachytávalo světlo z obrazovky. Poloha pera se poté poslala do počítače a na daném místě se vybarvil bod. Druhou rukou ovládal uživatel box s přibližně 40 tlačítky, které měly funkce jako mazání, zoomování či ukládání. Tím vznikl první počítač s interaktivní počítačovou grafikou. [38, 13, 27, 1, 22]

S postupným vývojem počítačů se začaly vytvářet i grafické algoritmy. Jedním z nich je Bresenhamův algoritmus. Vytvořil ho Jack Elton Bresenham v roce 1962 v International Business Machines Corporation (IBM). Tento algoritmus se používá ke kreslení úsečky mezi dvěma body. Jelikož počítačová obrazovka je rozdělena na pixely, pokud daná úsečka není vodorovná ani svislá, nelze ji vykreslit přesně. Algoritmus determinuje, jaký z dvojice pixelů

se více blíží funkci požadované úsečky a tento pixel vybarví. Dalším zajímavým algoritmem je ray casting. První obrázek byl vytvořen v roce 1968. Tento algoritmus funguje tak, že nejdříve uživatel definuje objekty, které chce mít ve své scéně jako matematické rovnice. Pro každý objekt se definuje model osvětlení, což udává jakou má barvu a jak se od něj světlo odráží. Poté se z pohledu kamery vyšle na každý pixel na obrazovce paprsek. Ten putuje, dokud se nezastaví o nějaký námi vytvořený objekt. Následně se z každého místa zastavení vyšle stínový paprsek do zdroje světla. Pokud cestou tento paprsek potká nějaký jiný objekt, daný objekt blokuje světlo, takže tento bod bude ve stínu. [38, 28, 31]

V průběhu 70. let se vyvíjely a vylepšovaly renderovací algoritmy. Jedním z problémů, který bylo potřeba vyřešit, byl hidden surface determination problem. Renderování částí objektů, které nejsou z pohledu kamery vidět zbytečně zatěžuje CPU. Tento problém řeší více rozdílných algoritmů, jako Z-buffer algorithm, Painter's algorithm nebo Binary Space Partitioning. Nejpopulárnější z nich je Z-buffer algoritmus vytvořen Wolfgangem Straßerem v roce 1974. Depth buffer uchovává nejmenší dosud naleznutou hloubku a pokud se přidá nový fragment na daný pixel, algoritmus porovná obě hloubky a pokud je blíže kameře, depth buffer se přepíše na novou hloubku. Další významný pokrok byl v oblasti shadingu. Doposud se využíval k barvení flat shading, takže každý polygon měl svojí vlastní barvu. V roce 1971 vytvořil Henri Gouraud gouraud shading. Tato metoda ukládá barvu polygonů jen do vrcholu daného polygonu, a následně se vypočítává barva každého pixelu váženým poměrem podle vzdálenosti od daného vrcholu. Dalším z důležitých technik je Phong shading, kterou vytvořil Bui Tuong Phong v roce 1974. Po každý pixel se počítá, jak na daný pixel dopadá světlo a jak se od něj odráží. To se následně aplikuje pro výhcozí barvu polygonu. Tato technika je velice náročná a mohla se plně začít využívat až s rovнем GPU. Pokrok se udělal i v realističnosti povrchů. Pokud by povrch měl mýt nějaké výstupky či hrbolky, museli by být dodány v popisu daného polygonu. Tento problém řeší technika zvaná Bump Mapping, vytvořena Jamesem Blinnem v roce 1978. Vytvoří se texturová mapa, která určuje kde je vrcholek a kde prohlubeň. Poté se ve fázi počítání světla místa s vrcholem udělají světlejší a místa s prohlubní tmavější, což dodává efekt hrbolatého povrchu na uplně rovném polygonu. [49, 30, 29, 10, 17]

V 80. letech začaly vytvářet první počítače, které byly dostupné pro veřejnost a začala se využívat GPU. Počítačová grafika se začala využívat ve filmovém průmyslu. V roce 1982 vyšel film Tron, který první masivně využíval CGI. Posun se udělal i v modelování vytvořením matematického modelu Non-Uniform Rational B-Spline (NURBS). Tento model

je schope z bodů a uzlu vytvořit 3D křivku nebo plochu. Další důležitý model byl Cook-Torrancův, který popisuje bidirectional reflective distribution function (BRDF). Tento model je podobný Phong shadingu, ale narodil od něj dodržuje fyzikální korektnost. Odražené světlo nikdy nemůže přesahovat světlo dopadající, címž se vyrendrovaná věc zdá realističtější. Dalším důležitým algoritmem je radiosity algorithm. Tento algoritmus počítá světlo polygonu součtem světla kolik vyzařuje a kolik světla přijímá od ostatních polygonů. Narodil ale od starších algoritmu všechny polygony také nějaké světlo odrážejí, což je dáno povrchem polygonu. Toto odražené světlo se také počítá do celkového osvícení ostatních polygonů. Významné zlepšení proběhlo u Ray Casting algoritmu. Paprsek se místo toho, aby se o polygon zastavil, odrazí a algoritmus se znova rekursivně spustí pro tento paprsek. Toto zlepšení umožňuje vidět v odraze jednoho polygonu druhý. [2, 26, 40, 36, 7]

V průběhu 90. let se významě vylepšovaly domácí počítače, které dokázali spustit i složitější programy a hry. 3D grafika se stala populárnější a vznikal pro ni i software jako Blender nebo 3D Studio. Nevznikly žádné nové revoluční renderovací metody, ale díky zlepšení výkonu počítačů se mohly začít využívat metody dříve vytvořené, které byly příliš náročné pro tehdejší počítače. Velký pokrok zaznamenal filmový průmysl. Terminátor 2 použil CGI na realistického humanoidního robota. V roce 1995 vydal Pixar první celovečerní plně počítačově animovaný film toy story. 90. léta jsou označována jako zlatý věk videoher. První z důležitých her je Wolfenstein 3D. Tato hra využívá ray-casting, takže není 3D, ale vytváří iluzi 3D prostoru. Další důležitou hrou je Doom. Tato hra byla 2.5D, tedy kombinovala prvky 2D a 3D. Zdi a podlaha byly 3D modely, ale příšery byly 2D sprity, kterých bylo více a ukazovaly se podle toho pod jakým úhlem se hráč na příšeru koukal. První plně 3D hra bola Quake. Quake měl pokročilý client-server multiplayer, takže bylo možno hrát ne jen na LAN, ale i přes internet. 3D modely umožnili vývojářům přidat do hry 3. rozměr a tím i více patrové mapy či létející projekty a nepřátele. [5, 14, 11, 21, 19]

V novém tisíciletí se zlepšovalo vše, co se doposud vynalezlo. Renderovací algoritmy mohly být více realistické kvůli zlepšení GPU.

Společně s vývojem počítačové grafiky začaly vznikat i první videohry. Hra Spacewar! byla jedna z prvních, která se rozšířila po laboratořích na amerických univerzitách. Hru vytvořil na MIT Steve Russell a jeho přátelé v roce 1962. Hra byla vytvořena pro dva hráče, každý ovládal svou lod' a jeho cílem bylo sestrelit lod' protihráče. Lodě byly ovládány speciálním boxem, který se dá považovat za předchůdce moderních herních kontrolerů. Hra běžela na počítači PDP-1, který namísto rasterové grafiky využíval grafiku vektorovou. Věci vykreslené

na osciloskopu byly definované matematickou funkcí. Díky tomu, že vektorová grafika nebyla tak náročná na CPU jako rasterová, mohla ve hře být hvězda, která svojí gravitací působila na lodě. [42, 6, 48]

2.2 Grafické karty

2.2.1 CPU vs. GPU

Tradičně běžela většina aplikací sekvenčně, které běží na procesorech, které mají centrální výpočetní jednotku a provádí instrukce jednu po druhé CPU. V průběhu 20. století se výkon CPU významě zlepšoval až na bilion operací za sekundu, čímž se mohli zlepšovat grafické aplikace a využívat náročnější funkce. Na přelomu tisíciletí se ale vývoj začal zpomalovat, kvůli problému se spotřebou energie a odvodem tepla. Proto výrobci začali vytvářet CPU s více jádry, ale ani to nebylo dostatečně výkonné na složité výpočty, a proto už v sedmdesátých a osmdesátých letech začali vznikat počítače které nepracovali sekvenčně, ale paralelně. V devadesátých letech se začali vytvářet mikroprocesory, které se soustředili na paralelní výpočty GPU. V průběhu let se počet jader z jednotek dostal na tisíce, což vytvořilo prostor pro inovaci v grafice. [24]

Prvně bylo GPU pouze v počítačích specializovaných pro 3D hry a vizualizace, ale postupně se stala součástí každého počítače. GPU a CPU se používají v různých případech kvůli rozdílné architektuře. CPU má jednotky až desítky jader a umí dělat jeden krok extrémě rychle kvůli sekvenčnímu zapojení. Velká část čipu je určena pro cache, tedý malu pamět, kam se ukládají data která budou pravděpodobně v budoucnosti potřeba, takže není potřeba komunikovat neustále s RAM. Část čipu je také pro control logiku, která dokáže instrukce z jednoho vlákna dělat paralelně nebo na jiném sekvenčním pořadí, ale zachovat sekvenčnost celého procesu. CPU má také menší propustnost paměti než GPU, proto jsou cache-friendly programy výrazně rychlejší. Narození od toho má GPU stovky až tisíce jader. Kontrolní logika a cache jsou v jádru menší a jednodužší, takže velmi pomalu reaguje na události a po dokončení operaí se většinou data vrací zpět do virtual random access memory VRAM, což je díky velké propustnosti i pro velmi velký objem dat rychlé. [24]

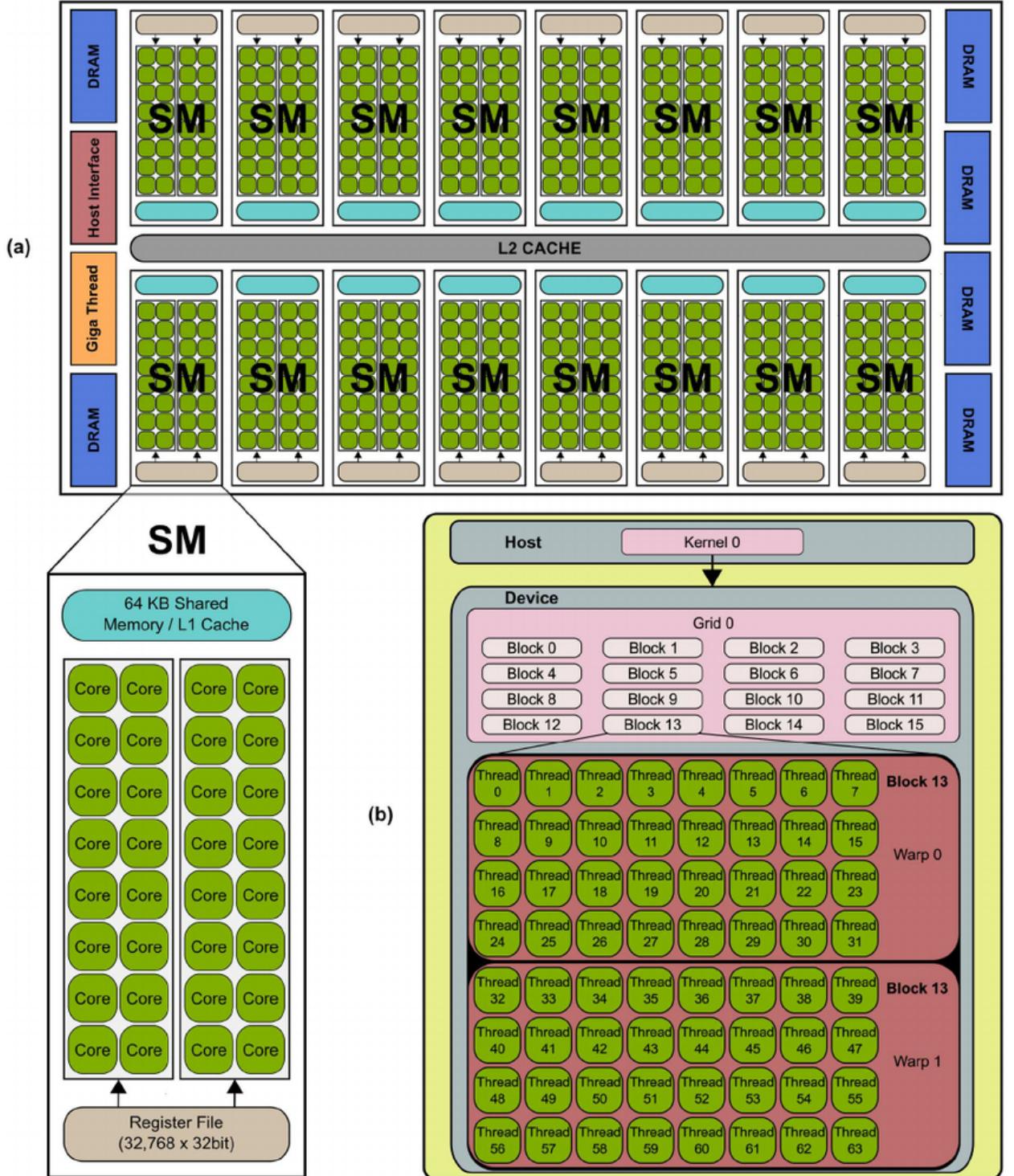
Do roku 2007 se GPU používal převážně pro renderování, ale v roce 2007 přinesla NVIDIA s programovacím modelem pro paralelní výpočty CUDA. Tento model umožňuje psát programy, které beží na GPU a využívají CPU na přenos dat a řízení této programu. Díky

tomu mohli zacit vznikat aplikace, ktere normalne bezi na CPU, ale pri potrebe narocnejcich pocetnich operaci mohly bezet na GPU, a tim zvysit vykon. Pro ucel vysvetleni programovacich modelu pro paralelni vypocty budu vyuzivat model CUDA, pricemz existuji i jine podobne modely jako OpenCL nebo SYCL. [24]

2.2.2 Architektura a paměťová hierarchie GPU

GPU s podporou CUDA je organizovano do pole paralelnich streamovacich multiprocesoru SM. Kazdy SM se sklada z nekolika streamovacich procesoru SP, ktere jsou zakladni vypoctni jednotkou, na ktere probihaji aritmeticke operace. Dalsi soucasti jsou ridici jednotky, ktere nacitaji instrukce pro dany SM. Ridici jednotky taky obsahuji malou cache na instrukce, aby se pro ne meuselo porad sahat do RAM. Dale je unvintr SM warp scheduler, ktery ridi ktere zkupiny vlaken budou v kazdem cyklu pouzity. [24]

Vetsina grafickych karet ma svoji vlastni dynamic random access memory DRAM, ktere se rika globalni pamet. V grafickych kartach je specificky typ DRAM zvany VRAM. Tato pamet ma jednotky az desitky gigabytu. Dochazi v ni k vymene dat mezi host memory. Rychla vymena dat mezi host memory a VRAM je zarucena velkou propustností, ktera je zpusobena nacitanim dat blokove namisto sekvencne a take posilanim vetsiho objemu dat najednou. VRAM je zdilena mezi vsemy SM, ale pristup k ni je relativne pomaly. Dalsi pameti je shared memory, ktera ma vetsinou desitky kilabajtu. Tuto pamet ma kazdy SM, a kazdy SP si do ni muze sahat. V kazdem SM se takze vyskytuje L1 cache, ktera slouzi k uchovavani dat, ze ktercyh se nedavno cetlo nebo byla zmenena. Dale existuje L2 cache, ktera je vetsi nez L1 cache, a je jedna pro vsechny SM. Pouziva se pro uchovavani dat nedavno ziskana nebo poslana do VRAM. Cast VRAM je take urcena pro dva jine typy pameti a to texture a constant memory. Obe pameti maji k sobe prirazenou svoji cache a jsou read-only. Tim ze jsou read-only, muzeme casto pouzivane hodnoty uchovavat v cachu bez obavy z toho, ze by se dana promena mohla v prubehu zmenit. Nejmensi, ale take nejrychlejsi casti pamety jsou register files. Tato pamet primo komunikuje s SP a ukladaji se do ni docasne promene a mezivypocty. [24, 8, 50]



Obrázek 2.2: Typická architektura NVIDIA GPU [35]

2.2.3 Paralelismus

Paralelismus je výpočetní technika, která umožnuje vykonávání více operací najednou. Paralelismus se dá rozdělit do dvou kategorií. První z nich je ulohový paralelismus. Tento typ nasleduje Multiple Instructions, Multiple Data (MIMD) model. Tento model rozdělí různé

ulohy mezi vlaken nebo i jader. Tento model se využívá, pokud na sobě nejsou ulohy závislé. Pokud by se například chtěla vypočítat suma arraye, může každé vlakno počítat jednu polovinu a dva výsledky se sestavit. MIMD se využívá v simulacích nebo na webových aplikacích. [24, 9]

Druhou metodou je data parallelismus, který je definován Single Instruction, Multiple Data (SIMD) modelem. Tento model aplikuje jednu úlohu pro velké množství dat a využívá se v GPU. Používá například při násobení matic, kde se dana matice dá rozdělit na menší části nebo při vykreslování, kdy se jeden shader aplikuje pro každý pixel. V grafických kartách se jednotlivá vlakna sdružují do warpu, ty dalej do bloku, které plní SM. Do každého warpu jsou načteny stejné instrukce a postupně se mu dodávají data, na kterých dané instrukce provádí. Díky tomu lze i velké množství dat zpracovávat velmi rychle, protože se dodávají s velkou propustností a instrukce na nich jsou prováděny současně na celém množství warpu. [24, 9]

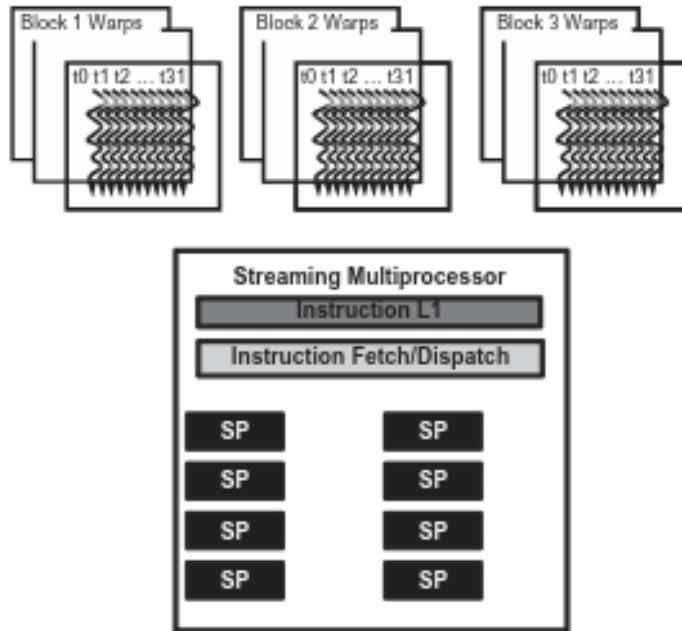


FIGURE 3.13

Blocks are partitioned into warps for thread scheduling.

Obrázek 2.3: Rozdělení vláken do warpu v SM [24]

2.3 Grafická zobrazovací pipeline

Hlavní funkci graficke zobrazovaci pipeline je vyrenderovat dvourozmerny obraz z trojrozmernych objektu, pozice virtualni kamery, zdroju svetla atd., který je schopen byt zobrazen na napriklad monitoru. Pipeline se sklada z nekolika fazi. Tyto faze probihaji paralelne v ramci jednotlivych faz, tedy jedna faze probiha pro vice dat najednou, tak i mezi jednotlivymi fazemi, takze pro pruchodu jednoho bloku dat jednou fazi prechazi do dalsi faze, i kdyz pro jina data jeste predchozi faze nemusi byt ukoncena. Hlavni ctyri faze jsou aplikacni faze, geometricke zpracovani, rasterizace a zpracovani pixelu. [2]

Aplikacni faze je rizena samotnou aplikaci a je vetsinou implementovana jako software bezici na CPU. Tato faze nema zadne podprvky, a proto muze bezet paralelne na ruznych jadrech. V teto casti se zpracovavaji vztupy od uzivatele, ktere napriklad pozmenuji matici k posunuti, otoceni nebo zmeny velkosti ruznych objektu. V teto casti take dochazi detekce kolizi, akceleracni algoritmy, fyzikalni simulace a dalsi vypocty, ktere pripravují data pro dalsi scenu. [2]

Dalsi casti pipeliny je geometricke zpracovani. Ta jo zodpovedna za vetsinu operaci pro kazdy trojuhelniky a vrcholy, ktere tvori objekty. Rozdeluje se na ctyri podfaze, vertex shading, projekce, clipping a screen mapping. Vertex shading vypocitava pozici vrcholu a pripravuje jeho data jako barvy a textury. Objekt je transformovan z modeloveho prostoru do svetovych souradnic, tedy souradnic relevantni pro vsechny objekty ve scene. Dale se to transformuje do kameroveho prosotru, ve kterym je kamera v bode nula. Pote dochazi k projekci, kdy dochazi k transformaci z kameroveho prostoru do clip spacu, aby bylo mozne provezt clipping. Clipping urcuje, jaký objekt se budou renderovat. Pokud by napriklad cely objekt byl mimo zorne pole sceny, není renderovan, aby zbytecne nezatezoval GPU. Posledni casti je mapovani na obrazovku, ktera prevede souradnice z clip spacu do souradnic obrazovky, pripravene pro rasterizaci. Soucasti vertex shadingu jsou take volitelne faze vertex zpracovani. Tessellation generuje vhodne trojuhelniky pro zakrivené povrchy, aby vypadali realisticneji. Geometry shader umoznuje podle potreby vytvaret nove vrcholy z existujicich trojuhelniku nebo bodu, coz se pouziva napriklad u malych castic, aby byli lepe videt na obrazovce, takze se z nej udela napriklad maly ctverec. Posledni volitelna faze je stream output, ktera umoznuje namisto poslani vrcholu do dalsi faze ulozit vrcholy do bufferu, na kterych CPU muze delat dalsi vypocty ci simulace. [2]

Nasledujici faze je restarizace, jejiz ucel je nalezt vsechny pixely uvnitr vykreslovaneho trojuhelniku. Restarizace se deli na pripravu trojuhelniku a pruchod trojuhelnikem. Pri priprave se vypocitavaji diferencialy, rovnice hran a jina podobna data, ktera jsou vyuzivana k zjistovani zda dany pixel lezi uvnitr trojuhelniku. Pri pruchodu se zjistuje, zda stred pixelu lezi uvnitr trojuhelniku a pokud ano, vytvori se pro dany pixel fragment. Vlastnosti, jako barva, hloubka atd. jednotlivych fragmentu se ziskavaji interpolaci mezi vrcholy daneho trojuhelniku a nasledne se ukladaji do specificych bufferu. Vysledne fragmenty jsou nasledne predany do faze pixeloveho zpracovani. [2]

Pixelove zpracovani se deli na pixelove stinovani a slucovani. Ve fazi pixeloveho stinovani se provadeji vypocty stinovani jednotlivych pixelu z stinovacich dat. Vysledkem je jeden nebo vice barevnych vystupu. V teto fazi se pouziva hodne specifickych technik, jako napriklad texturovani. Pixelove stinovani je narozeno od ostatnich faz, ktere jsou dany architekturou GPU, dano programovatelnymi jadry GPU. Pri slucovani dochazi k kombinaci vyztupu pixeloveho stinovani a ulozenyh barev v bufferech. V teto fazi se take resi viditelnost, aby se vykresloval pro dany pixel fragment s nejmensi hloubkou. [2]

2.4 Transformace

V prubehu renderovani je casto potreba s objekty ruzne pohybovat. Mohli bychom pozmenovat souradnice puvodnich vrcholu a opetovne konfigurovat jejich buffery, ale to je vypocetne naroce. Misto toho se vyuzivaji transformace pomocni matic. Matice jsou obdelnikove pole matematicickych vyrazu. Souradnice objektu se ukladaji do 4D vektoru, a transformace do 4x4 matice, ktere se mezi sebou vynasobi.

$$\begin{pmatrix} m_{00} \cdot x + m_{01} \cdot y + m_{02} \cdot z + m_{03} \cdot w \\ m_{10} \cdot x + m_{11} \cdot y + m_{12} \cdot z + m_{13} \cdot w \\ m_{20} \cdot x + m_{21} \cdot y + m_{22} \cdot z + m_{23} \cdot w \\ m_{30} \cdot x + m_{31} \cdot y + m_{32} \cdot z + m_{33} \cdot w \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

Mezi zakladni operace patri translace, rotace a skalovani. Translace je operace, pri ktore se k puvodnim vektoru price jiny vektor, cimz se puvodni objekt posune. Potrebujeme translacni hodnoty nasobit slozkou w vektoru, ktera je nastavena na 1, aby neupravila tyto hodnoty.

Proto pouzivame 4D vektor, protoze s 3D vektorem by translace nebyla mozna.

$$\begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Dalsi operaci je skalovani. Pri teto operaci se dane souradnice nasobi konstantou, cimz se muze dany objekt zmensit nebo zvetsit.

$$\begin{pmatrix} S_x \cdot x \\ S_y \cdot y \\ S_z \cdot z \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Posledni dulezita operace je otaceni. Pri teto operaci zavisi na tom, podle jake osy se ma objekt otacet a o kolik radianu se ma otocit.

Rotace podle osy X:

$$\begin{pmatrix} x \\ \cos \theta \cdot y - \sin \theta \cdot z \\ \cos \theta \cdot y + \cos \theta \cdot z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotace podle osy Y:

$$\begin{pmatrix} \cos \theta \cdot x + \sin \theta \cdot z \\ y \\ -\sin \theta \cdot x + \cos \theta \cdot z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Rotace podle osy Z:

$$\begin{pmatrix} \cos \theta \cdot x - \sin \theta \cdot y \\ \sin \theta \cdot x + \cos \theta \cdot y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Pri provadeni transformaci zalezi na poradi. Pokud je pro transformaci jednoho objektu vice matic, pronasobi se matice mezi sebou, a pouziva se vysledna matice. Pri tvoreni takove

matice se ale postupuje v opacnem poradi nasobeni. Pokud by jsme pro vektor \mathbf{V} chteli udelat operace a , b a c , tak to lze udelat nasledovne:

$$\mathbf{V}' = \mathbf{C} \cdot (\mathbf{B} \cdot (\mathbf{A} \cdot \mathbf{V}))$$

nebo

$$\mathbf{M} = \mathbf{C} \cdot \mathbf{B} \cdot \mathbf{A}$$

$$\mathbf{V}' = \mathbf{M} \cdot \mathbf{V}$$

3 OpenGL

3.1 Historie OpenGL

3.2 OpenGL pipeline

3.3 Shadery

3.4 Textury

4 to do

mozna pridat trochu historie compileru u C a CPP

celá část hirstorie počítačové grafiky je scuffed, chybí tam části o gpu, a nejspis to bude potreba zkratit a udelat rozdeleni do subsectionu podle let

kdyz introdusuju zkratku, dej ji velka pismena, at ctenar vi ze je to zkratka
predelat pomlcky z - na tu dinvou co pouziva chat a chce ji pazi
odendej vsechny cislovky z textu a nahrad je slovama

Část II

Vývoj hry v OpenGL

5 Použité knihovny

5.1 GLAD

5.2 GLFW

5.3 stb_image

6 Grafika a zvuk

6.1 Aseprite

6.2 Bosca Ceoil

7 Herní scény

7.1 Scéna hlavního menu

7.2 Scéna pozastavené hry

7.3 Scéna hry

8 Herní mechaniky

8.1 Generování objektů

8.2 Pohyb hráče

8.3 Detekce kolize

Závěr

Bibliografie

1. *A Critical History of Computer Graphics and Animation* [Online]. [B.r.]. Dostupné také z: <https://web.archive.org/web/20070405181508/http://accad.osu.edu/%7Ewaynec/history/lesson2.html>. [citováno 2025-11-07].
2. AKENINE-MÖLLER, Tomas et al. *Real-Time Rendering*. 4th. A K Peters/CRC Press, 2018.
3. ALBATROSS. *History of C++* [Online]. [B.r.]. Dostupné také z: <https://cplusplus.com/info/history/>. [citováno 2025-10-17].
4. BANAHAN, Mike; BRADY, Declan; DORAN, Mark. *The C Book: Featuring the ANSI C Standard(Instruction Set)*. 2nd. Addison-Wesley, 1991. ISBN 9780201544336.
5. BHATTACHEJEE, Souktik. *A timeline of 3D softwares* [Online]. [B.r.]. Dostupné také z: <https://www.re-thinkingthefuture.com/career-advice/a2944-a-timeline-of-3d-softwares/>. [citováno 2025-12-02].
6. BRITANNICA. *vector graphics* [Online]. 2025. Dostupné také z: <https://www.britannica.com/technology/vector-graphics/additional-info#history>. [citováno 2025-11-12].
7. CCGOMENZ. *Radiosity Algorithm* [Online]. [B.r.]. Dostupné také z: <https://ccgomezn.github.io/vc/docs/workshops/rendering/radiosity>. [citováno 2025-11-17].
8. COMPUTE, Arc. *Memory Hierarchy of GPUs* [Online]. 2023. Dostupné také z: <https://www.arccompute.io/arc-blog/gpu-101-memory-hierarchy>. [citováno 2025-12-13].
9. CSBRANCH. *Types of Parallelism: Data vs. Task Parallelism* [Online]. 2024. Dostupné také z: <https://csbranch.com/index.php/2024/10/26/types-of-parallelism-data-vs-task-parallelism/>. [citováno 2025-12-15].

10. DIGITAL ARTS, Copenhagen Academy of. *WHAT IS PHONG SHADING? (GUIDE WITH EXAMPLES)* [Online]. [B.r.]. Dostupné také z: <https://cada-edu.com/guides/what-is-phong-shading>. [citováno 2025-11-15].
11. DOUGC. *What was the deal with 2.5D games like DOOM* [Online]. 2002. Dostupné také z: <https://boards.straightdope.com/t/what-was-the-deal-with-2-5d-games-like-doom/105737/3>. [citováno 2025-12-02].
12. ENGEL, Wolfgang. *The History of the GPU - New Developments*. 1st. Springer International Publishing, 2023. ISBN 9783031140464.
13. ENGINEERING; WIKI, Technology History. *Milestones:Whirlwind Computer, 1944-59* [Online]. 2024. Dostupné také z: https://ethw.org/Milestones:Whirlwind_Computer,_1944-59. [citováno 2025-11-06].
14. FRONT, Wolf. *Development History* [Online]. [B.r.]. Dostupné také z: <https://wolfenstein3d.nl/development-history/>. [citováno 2025-12-02].
15. GEEKSFORGEEKS. *Features of C* [Online]. 2025. Dostupné také z: <https://www.geeksforgeeks.org/c/features-of-c-programming-language/>. [citováno 2025-10-12].
16. GEEKSFORGEEKS. *Top 25 C++ Applications in Real World[2025]* [Online]. 2025. Dostupné také z: <https://www.geeksforgeeks.org/blogs/top-applications-of-cpp-in-real-world/>. [citováno 2025-10-22].
17. GLOSSARY, Computer Graphics. *Bump Mapping - Definition & Detailed Explanation - Computer Graphics Glossary Terms* [Online]. 2025. Dostupné také z: <https://pcpartsgeek.com/bump-mapping/>. [citováno 2025-11-15].
18. HARBISON, Samuel P.; STEELE, Guy L. *C, a Reference Manual*. 5th. Prentice-Hall, 2002. ISBN 9780130895929.
19. HICKMAN, Zachary. *Quake Engine Analysis* [Online]. [B.r.]. Dostupné také z: <https://zhickman.com/analysisfinal.pdf>. [citováno 2025-12-02].
20. HUGHES, John F. et al. *Computer Graphics: Principles and Practice*. 3rd. Addison-Wesley Professional, 2013.

21. JPIOLHO. [2021 Re-release] *How multiplayer works* [Online]. 2021. Dostupné také z: <https://steamcommunity.com/sharedfiles/filedetails/?id=2624343184>. [citováno 2025-12-02].
22. KAY, Alan. *Vision & Reality of Hypertext and Graphical User Interfaces* [Online]. [B.r.]. Dostupné také z: https://mprove.de/visionreality/text/3.1.2_sketchpad.html. [citováno 2025-11-07].
23. KERNIGHAN, Brian W.; RITCHIE, Dennis M. *The C Programming Language*. 2nd. Pearson, 1988. ISBN 9780131103627.
24. KIRK, David B.; HWU, Wen-mei W. *Programming Massively Parallel Processors: A Hands-on Approach*. 3rd. Morgan Kaufmann, 2016.
25. KOCHAN, Stephen G. *Programming in C*. 4th. Pearson Education, 2015. ISBN 9780321776419
26. KONOW, David. *Putting the Original Tron's Special Effects Together* [Online]. 2015. Dostupné také z: <https://web.archive.org/web/20180512181521/http://www.tested.com/art/movies/520562-putting-original-trons-special-effects-together/>. [citováno 2025-11-16].
27. LABORATORY, Lincoln. *SAGE: SEMI-AUTOMATIC GROUND ENVIRONMENT AIR DEFENSE SYSTEM* [Online]. [B.r.]. Dostupné také z: <https://www.ll.mit.edu/about/history/sage-semi-automatic-ground-environment-air-defense-system>. [citováno 2025-11-06].
28. LABS, BCA. *Vision & Reality of Hypertext and Graphical User Interfaces* [Online]. [B.r.]. Dostupné také z: <https://bcalabs.org/subject/bresenhams-line-algorithm-in-computer-graphics>. [citováno 2025-11-11].
29. LEE, Sarah. *Mastering Gouraud Shading Techniques* [Online]. 2025. Dostupné také z: <https://www.numberanalytics.com/blog/mastering-gouraud-shading-techniques>. [citováno 2025-11-15].
30. LOGAN. *Z-Buffer Algorithm* [Online]. 2025. Dostupné také z: <https://www.onlycode.in/z-buffer-algorithm/>. [citováno 2025-11-15].
31. LONDON, Imperial College. *Ray Tracing* [Online]. [B.r.]. Dostupné také z: <https://www.doc.ic.ac.uk/~bkainz/graphics/notes/GraphicsNotes1011.pdf>. [citováno 2025-11-12].

32. MORTENSEN, Peter. *What's the difference between a low-level, midlevel and hight-level language* [Online]. 2023. Dostupné také z: <https://stackoverflow.com/questions/3468068/whats-the-difference-between-a-low-level-midlevel-and-high-level-language>. [citováno 2025-10-13].
33. NAPPER, Brian. *The Manchester Small Scale Experimental Machine – "The Baby"* [Online]. 1999. Dostupné také z: <https://web.archive.org/web/20120604211339/http://www.computer50.org/mark1/new.baby.html>. [citováno 2025-11-05].
34. NOLLE, Tom. *What is structured programming(modular programming)* [Online]. 2023. Dostupné také z: <https://www.techtarget.com/searchsoftwarequality/definition/structured-programming-modular-programming>. [citováno 2025-10-14].
35. NVIDIA. *Accelerating Fibre Orientation Estimation from Diffusion Weighted Magnetic Resonance Imaging Using GPUs* [Online]. 2013. Dostupné také z: https://www.researchgate.net/figure/Typical-NVIDIA-GPU-architecture-The-GPU-is-comprised-of-a-set-of-Streaming_fig1_236666656. [citováno 2025-12-11].
36. OPENGL, Learn. *Theory, BRDF* [Online]. [B.r.]. Dostupné také z: <https://learnopengl.com/PBR/Theory>. [citováno 2025-11-16].
37. PEDDIE, Jon. *The History of the GPU - Eras and Environment*. 2nd. Springer Nature, 2023. ISBN 9783031135811.
38. PEDDIE, Jon. *The History of the GPU - New Developments*. 1st. Springer International Publishing, 2023. ISBN 9783031140464.
39. PEDDIE, Jon. *The History of the GPU - Steps to invention*. 1st. Springer International Publishing, 2023. ISBN 9783031109676.
40. RHINOCEROS. *What are NURBS* [Online]. [B.r.]. Dostupné také z: <https://www.rhino3d.com/features/nurbs/>. [citováno 2025-11-16].
41. SEVO, Daniel. *HISTORY OF COMPUTER GRAPHICS 1990-99* [Online]. [B.r.]. Dostupné také z: https://www.danielsevo.com/hocg/hocg_1990.htm. [citováno 2025-12-03].
42. SMITH, Ryan P. *How the First Popular Video Game Kicked Off Generations of Virtual Adventure* [Online]. 2018. Dostupné také z: <https://www.smithsonianmag.com/smithsonian-institution/how-first-popular-video-game-kicked-off-generations-virtual-adventure-180971020/>. [citováno 2025-11-12].

43. SRUTHY. *What Is C++ Used For? Top 12 Real-World Applications And Uses Of C++* [Online]. 2025. Dostupné také z: <https://www.softwaretestinghelp.com/cpp-applications/>. [citováno 2025-10-22].
44. STROUSTRUP, Bjarne. *Design and Evolution of C++, The*. 1st. Addison-Wesley Professional, 1994. ISBN 9780201543308.
45. STROUSTRUP, Bjarne. *Programming Principles and Practice Using C++*. 2nd. Pearson Education, 2014. ISBN 9780321992789.
46. STROUSTRUP, Bjarne. *The C++ Programming Language*. 4th. Pearson Education, 2013. ISBN 9780321563842.
47. TECH, WsCube. *Top Applications of C programming* [Online]. 2024. Dostupné také z: <https://www.wscubetech.com/resources/c-programming/applications>. [citováno 2025-10-09].
48. TILLEY, Thomas. *Spacewar! Controllers* [Online]. 2015. Dostupné také z: <https://tomtilley.net/projects/spacewar/>. [citováno 2025-11-12].
49. TODAY, Everything Explained. *Hidden-surface determination explained* [Online]. [B.r.]. Dostupné také z: https://everything.explained.today/Hidden-surface_determination_explained. [citováno 2025-11-15].
50. UNIVERSITY, Cornell. *Memory Types* [Online]. [B.r.]. Dostupné také z: https://cvw.cac.cornell.edu/gpu-architecture/gpu-memory/memory_types. [citováno 2025-12-13].
51. VOLLE, Adam. *C++ computer language* [Online]. 2025. Dostupné také z: <https://www.britannica.com/technology/C-computer-language>. [citováno 2025-10-17].

Zkratky

CPU Central processing unit. 9, 11, 12, 15

CRT Cathode ray tube. 7

DRAM Dynamic random access memory. 12

GPU Graphics processing unit. 9–16, 33

RAM Random access memory. 7, 11, 12

SM Streaming multiprocessor. 12, 14, 33

SP Streaming processor. 12

SSEM Small-Scale Experimental Machine. 7

VRAM Video random access memory. 11, 12

Seznam obrázků

2.1	Skatchpad Ivana Sutherlanda [22]	8
2.2	Typická architektura NVIDIA GPU [35]	13
2.3	Rozělení vláken do warpu v SM [24]	14

Seznam tabulek

Přílohy

A Fotky z pokusů

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

B Příloha další