API graficzne – wczoraj, dziś i jutro

W artykule tym przejdziemy szybkim krokiem przez historię API graficznych takich jak DirectX, OpenGL, Vulkan oraz towarzyszącego im na przestrzeni lat rozwoju kart graficznych z jednej strony, a z drugiej – gier video. Nie będziemy się uczyli programowania w żadnym z tych API. Artykuł ten powinien być zrozumiały i zaciekawić może każdego, kto interesuje się grami czy grafiką albo przynajmniej w dzieciństwie grał w gry.

Siłą rzeczy w artykule przedstawiono tylko wybrane fakty i zastosowano pewne uproszczenia. To nie będzie kronika kolejnych dat i wydarzeń w historii rozwoju grafiki komputerowej. Chcę raczej zaprosić czytelnika w podróż, w trakcie której będziemy skakali po osi czasu (aż do czasów starożytnych!), aby jak najlepiej zrozumieć, skąd się wzięła sytuacja, jaka panuje obecnie w dziedzinie programowania kart graficznych, i jak może ona rozwijać się dalej w przyszłości.

Skupimy się na platformach, które służą do grania w gry komputerowe w domu, a więc przede wszystkim na pecetach z systemem Windows. Rozwój grafiki na Linuksie, na urządzeniach mobilnych, na jakichś profesjonalnych stacjach roboczych czy serwerach mógłby stanowić osobną historię.

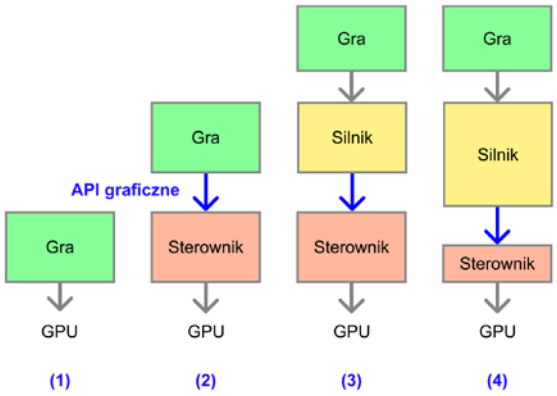
PREHISTORIA

Aby zrozumieć, w jaki sposób i w jakim celu powstały API graficzne, musimy cofnąć się do czasów, kiedy narzędzia te jeszcze nie istniały. Lata 80. to z punktu widzenia współczesnej informatyki prawdziwa prehistoria. Królowały wówczas komputery domowe, takie jak ZX Spectrum, Commodore 64, Atari czy Amiga. Miały one niewielkie możliwości sprzętowe. Na przykład 16-bitowe Atari ST wyposażone było w 256 KB do 1 MB pamięci RAM oraz procesor o częstotliwości 8 MHz.

Już na takie komputery powstawały jednak wciągające gry. Ich grafika była jeszcze dwuwymiarowa, o niskiej rozdzielczości i ograniczonej palecie kolorów. Już wtedy jednak komputery wyposażone były w sprzętowe przyspieszenie (akceleracje) wyświetlania grafiki w postaci rysowania tzw. duszków (ang. *sprites*). Dzięki nim można było wyświetlać poruszające się obiekty (jak postać bohatera czy potworów–przeciwników) dość szybko, aby gracz dostrzegał płynny ruch, co nie byłoby możliwe, gdyby powolny procesor główny musiał je rysować piksel po pikselu.

Programowanie tych komputerów odbywało się bardzo niskopoziomowo, często bezpośrednio w asemblerze. Wyświetlanie grafiki, jak również wszystkie inne funkcje sprzętowe realizowane były bez ustawiania odpowiednich rejestrów czy też zapisów pod określone adresy pamięci. Taki kod był oczywiście specyficzny dla danego modelu komputera, co nie było problemem dopóty, dopóki wszyscy użytkownicy mieli taki sam lub podobny sprzęt (rozszerzony ewentualnie o dodatkową pamięć itp.)

Aby zrozumieć ewolucję API graficznych, będziemy spoglądać na Rysunek 1 i do niego wracać. Opisaną powyżej sytuację ilustruje pierwsza kolumna, oznaczona jako (1). Widzimy w niej, że gra odwołuje się bezpośrednio do sprzętu, a konkretnie do układu graficznego (**GPU**).



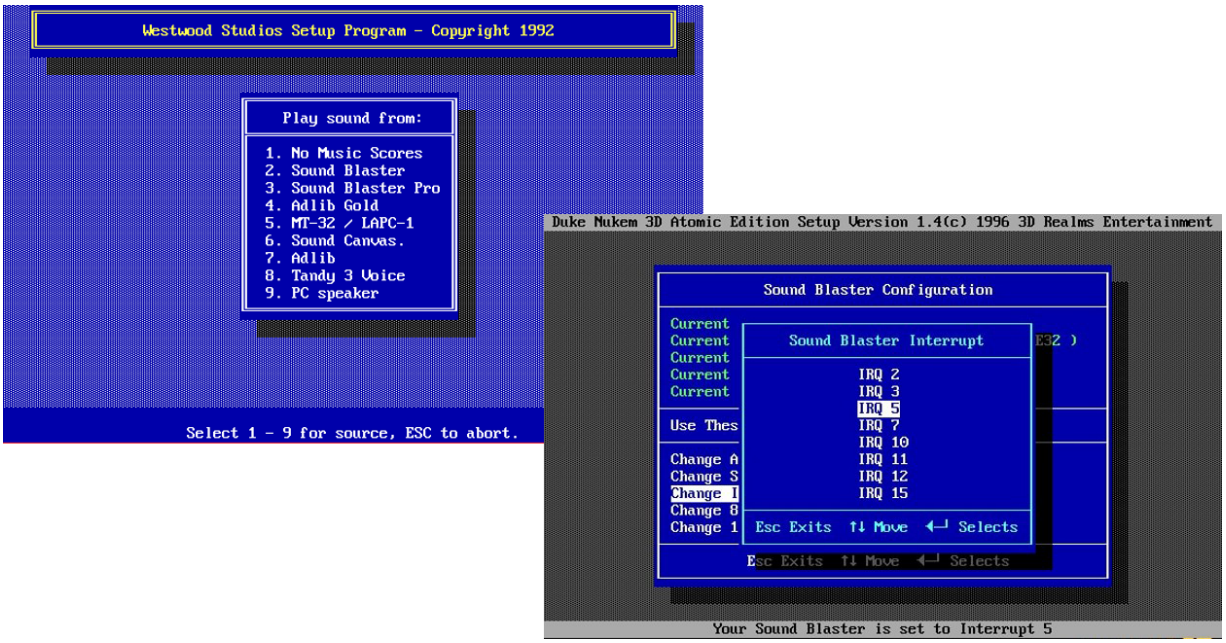
Rysunek 1. Rozwój gier, silników i API graficznych

PIERWSZE PECETY

W drugiej połowie lat 80. i na początku lat 90. dominować zaczęły komputery osobiste typu IBM PC. Architektura ta podbiła rynek m.in. za sprawą swojej modularności i rozszerzalności. Do ustandaryzowanej obudowy i płyty głównej można było montować różnorodne komponenty, np. karty rozszerzeń, a do uniwersalnych portów wejścia–wyjścia podłączać różne urządzenia peryferyjne dostarczane przez wielu producentów.

Dawne pecety miały monitory monochromatyczne z kartami graficznymi takimi jak Hercules, ale później pojawiły się standardy kolorowe, jak VGA. Kto dodatkowo zainstalował w swoim komputerze kartę dźwiękową oraz napęd CD-ROM, mógł pochwalić się, że jego komputer jest „multimedialny". Mógł on wtedy kupić i uruchomić zapisaną na płycie CD „encyklopedię multimedialną", która zawierała nie tylko ogromny zbiór haseł z różnych dziedzin z opisem tekstowym, ale również dołączone do niektórych haseł obrazki, dźwięki, a nawet krótkie sekwencje filmowe, co wówczas robiło ogromne wrażenie.

Wracając jednak do pecetów najdawniejszych, systemem operacyjnym był tam MS-DOS, a programowanie aplikacji nadal odbywało się niskopoziomowo, przez bezpośredni dostęp do sprzętu. W modularnej architekturze PC rodziło to problemy z zapewnieniem kompatybilności z różnymi urządzeniami. Nie było wówczas architektury Plug&Play. Każda gra czy program musiał być osobno przystosowany do pracy z wieloma popularnymi modelami urządzeń. Na Rysunku 2 pokazano zrzuty ekranu z gry „Duke Nukem 3D", w której ręcznie skonfigurować trzeba było posiadany model karty dźwiękowej i numer przerwania IRQ, aby w grze pojawił się dźwięk.



Rysunek 2. Konfiguracja dźwięku w grze „Duke Nukem 3D"

WINDOWS I STEROWNIKI

O ile DOS był systemem jednozadaniowym, pojawienie się systemu Windows zapewniło możliwość pracy z wieloma aplikacjami naraz. Wraz z nią dostępny stał się graficzny interfejs użytkownika (GUI) oparty na okienkach, które mogły wzajemnie się przysłaniać. Początkowo model współpracy między uruchomionymi programami był kooperatywny (aplikacja musiała oddać sterowanie następnej). To jednak sprawiało, że źle napisany program mógł łatwo zawiesić cały komputer. W nowszych wersjach Windowsa, jak 95, 98, a tym bardziej wersjach opartych na jądrze NT (Windows 2000, XP itd.), aplikacje nie mogły już tak łatwo zawiesić całego systemu czy uzyskać dostępu do pamięci innego procesu.

To wymagało także odizolowania aplikacji od bezpośredniego dostępu do sprzętu. Taka „wirtualizacja", w połączeniu z wielozadaniowością, a także potrzebą zapewnienia kompatybilności różnych aplikacji z różnymi modelami klawiatur, myszek, drukarek, monitorów itd. spowodowała, że niezbędnym pośrednikiem stał się **sterownik** (ang. *driver*). Dostarczony przez producenta danego urządzenia, zainstalowany w systemie, zajmował się kontrolowaniem swojego urządzenia, podczas gdy aplikacje przeprowadzały wszelkie operacje (np. wyświetlania okienek, odtwarzania dźwięków, drukowania) poprzez API systemowe (WinAPI).

POWSTANIE DIRECTX

Taki stan był dobry dla aplikacji okienkowych, jak edytory tekstu, ale nie był idealny dla gier. Na Rysunku 3 pokazano zrzut ekranu z gry „SimCity 2000", która w wersji na Windows 3.1 wyświetlała swoją grafikę w ramach zwykłego okienka, korzystając z systemowego menu i innych kontrolek. Jak nietrudno się domyślić, twórcy gier woleliby wyświetlać całą grafikę we własnym zakresie, przejmując kontrolę nad pełnym ekranem. To nie tylko pozwala grom wyglądać bardziej atrakcyjnie, ale też zapewnia lepszą wydajność, gdyż bezpośredni dostęp do bufora ramki omija komponowanie nakładających się okienek i związane z tym dodatkowe kopiowanie danych.

Rysunek 3. Gra „SimCity 2000" w wersji na Windows 3.x

W odpowiedzi na te potrzeby Microsoft stworzył interfejs **DirectX**. To nowe API, przeznaczone do gier i innych aplikacji multimedialnych, zapewniało wydajniejszy dostęp do funkcji sprzętowych, jak na przykład wejście w tryb pełnoekranowy (ang. *exclusive fullscreen*). Słowo „direct" nie oznacza tu jednak bezpośredniego dostępu do sprzętu w sensie tak dosłownym, jak to było w dawnych komputerach. W systemie nadal działał sterownik, a także inne aplikacje, tak że wciśnięcie Alt+Tab powodowało opuszczenie trybu pełnoekranowego.

Litera „X” w nazwie DirectX oznaczała z kolei „coś”, czym mógł być jeden z wielu komponentów wchodzących w skład tego API. Stare wersje DirectX dostarczały bowiem osobną część służącą do wyświetlania grafiki 2D (DirectDraw), grafiki 3D (Direct3D), odtwarzania dźwięku (DirectSound), muzyki (DirectMusic), obsługi klawiatur, myszek i innych kontrolerów (DirectInput), odtwarzania video (DirectShow) i innych. Do naszych czasów przetrwał jedynie **Direct3D**. Pozostałe biblioteki, jeśli wciąż działają dla zapewnienia kompatybilności wstecznej, są już jednak niezalecane, gdyż zostały wyparte przez inne, nowocześniejsze. Obecnie określenie DirectX jest więc synonimem Direct3D.

Jeszcze inną ciekawostką może być geneza nazwy konsoli **Xbox**. Oznacza ona bowiem nic innego, jak „DirectX Box”, czyli pudełko z DirectXem. Microsoft, tworząc swoją konsolę, zdecydował się bowiem „upakować” tam komputer ze specjalną wersją systemu Windows i właśnie DirectXem jako API przeznaczonym do obsługi grafiki i wszelkich multimediów przez gry tworzone na tę platformę. Tak jest do dzisiaj, gdzie tworzenie gier na nowe generacje Xbox wymaga używania DirectXa (ze specyficznymi dla konsoli rozszerzeniami).

CZYM JEST API?

W tym miejscu warto doprecyzować, co mamy na myśli, pisząc **API**. Skrót ten oznacza „Application Programming Interface”. Interfejs to ogólnie pewien łącznik między czymś a czymś. Tak jak graficzny interfejs użytkownika (GUI) jest łącznikiem pozwalającym użytkownikowi sterować aplikacją, tak API stanowi dla programisty protokół komunikacji między częściami oprogramowania, np. między pisanym programem a jakąś biblioteką. Tak naprawdę API to definicja funkcji, struktur, stałych i innych elementów kodu, zapisana np. w postaci dokumentacji lub pliku nagłówkowego .h dla języka C.

API graficzne są natomiast interfejsem pomiędzy aplikacją (najczęściej grą) a sterownikiem graficznym. API takie jak Direct3D stanowi standardowy protokół komunikacji pozwalający wydawać komendy rysowania grafiki na ekranie, podczas gdy sterownik zajmuje się ich tłumaczeniem na niskopoziomowe komendy specyficzne dla danego modelu GPU (dyskretnej karty graficznej bądź też układu graficznego zintegrowanego z procesorem). Sytuację tą ilustruje kolumna (2) na Rysunku 1.

PREKURSORZY 3D

Pojawienie się grafiki trójwymiarowej w grach nie było wydarzeniem nagłym. Wskazać można wiele ogniw tej ewolucji. Większość z nich związana jest z rozwojem gier z gatunku strzelanek z perspektywy pierwszej osoby (ang. **FPS** – *First Person Shooter*). Jedną z pierwszych popularnych gier tego typu był „Wolfenstein 3D” firmy id Software. O ile trójwymiarowe ściany z nałożonymi teksturami robiły wrażenie, o tyle – ze względu na sposób renderowania grafiki w tej grze – podłogi i sufity musiały pozostać płaskie i jednokolorowe. Dopiero następna strzelanka tej firmy – „Doom” – zniosła to ograniczenie, pozwalając uzyskać na mapie schody.

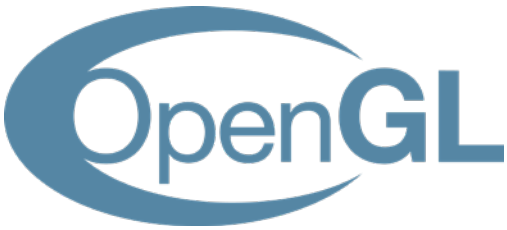
Wskazać można kilka firm developerskich i ich tytułów, które stanowiły kamienie milowe w historii rozwoju gier 3D, w tym m.in. id Software („Wolfenstein 3D”, „Doom”, „Quake”), Epic MegaGames („Unreal”) czy 3D Realms („Duke Nukem 3D”). Historię rozwoju gier FPS świetnie opowiada 4,5-godzinny film „FPS: First Person Shooter Documentary” (2023) [1].

Współzałożyciele tych firm, jak John Carmack w id Software i Tim Sweeney w Epicu, stali się postaciami kultowymi w środowisku programistów grafiki i gier. Ówcześnie sami byli głównymi programistami swoich gier. Obecnie każdy z nich jest raczej pełnym sukcesu przedsiębiorcą, choć obaj pozostają blisko związani z technologią. Tim Sweeney nadal zarządza firmą Epic, nadzorując rozwój silnika Unreal Engine i gry „Fortnite”, podczas gdy John Carmack skupił się na tematyce sztucznej inteligencji.

Sprzętowa akceleracja grafiki 3D też nie pojawiła się natychmiast. Pierwsze gry 3D renderowane były w pełni programowo – przez procesor główny CPU. Pewnym „brakującym ogniwem ewolucji” była karta 3dfx Voodoo, która stanowiła dodatkową kartę rozszerzeń przeznaczoną wyłącznie do grafiki 3D, która musiała być zamontowana w pececie obok zwykłej karty graficznej. Do jej programowania służyło własne API o nazwie Glide. Dopiero potem zwykłe karty graficzne w pecetach zintegrowały sprzętowe wsparcie dla rasteryzacji trójkątów, które stało się standardowym sposobem na przedstawianie obiektów 3D i takim pozostaje do dzisiaj.

Tymczasem gry z gatunku strzelanek pierwszoosobowych nadal wyznaczają nowe ścieżki w jakości grafiki, fotorealizmie i wykorzystaniu nowych, zaawansowanych technik graficznych. Razem z nimi w parze często idą wysokie wymagania sprzętowe, które niejednego gracza skłaniają do zakupu nowej konsoli, nowego komputera czy przynajmniej nowszej karty graficznej, aby zagrać w nowo wydany tytuł. Takimi grami wyznaczającymi nowy standard jakości były swego czasu m.in. „Crysis”, „Killzone 2”, „STALKER”, a także wychodzące regularnie nowe gry z serii „Call of Duty”.

CO Z OPENGL?



Uwagę dotychczas poświęciliśmy interfejsowi DirectX, a tymczasem opisując API graficzne, wypada wspomnieć też o jego konkurencie: **OpenGL**. Można wskazać wiele różnic między nimi, jak też i wiele podobieństw. Oba rozwijały się w ciągu wielu dekad, dając dostęp do wydajnego renderowania grafiki 3D. Oba dostępne są do użycia w systemie Windows i choć większość gier korzysta z DirectX, wydanych zostało też wiele dużych gier opartych na OpenGL (w tym gry firmy id Software).

Trudno powiedzieć z całą pewnością, by jedno z tych API było obiektywnie lepsze od drugiego, dostarczało większych możliwości czy działało z większą wydajnością. Można oczywiście rozważać wiele różnic i niuansów, jednak koniec końców to są tylko interfejsy zapewniające dostęp do tych samych funkcji sprzętowych karty graficznej.

K-202: SYNDROM IKARA · APLIKACJE SAMOCHODOWE ANDROID AUTO – PROJEKTOWANIE GUI APLIKACJI

INDEX: 285358

www.programistamag.pl

Magazyn programistów i liderów zespołów IT

programista

4/2025 (119)

Cena 32.90 zł (w tym VAT 8%)

API GRAFICZNE WCZORAJ, DZIŚ I JUTRO

ISSN 2084-9400
04
9 772084 940503

Bootstrapping aplikacji React Native

Zbuduj własnego Linuxa z Yocto i Raspberry Pi

Repozytorium bez chaosu
Zaawansowane techniki pracy z Git

Wprowadzenie do testów automatycznych dla ASP.NET