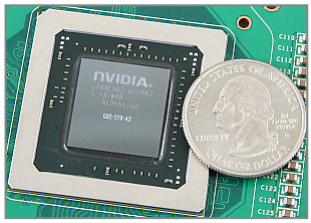
Synteza dźwięku i obrazu

GRAFIKA KOMPUTEROWA

Rozwiązania sprzętowe i programowe



Przyspieszanie sprzętowe

Generowanie obrazu 3D wymaga złożonych obliczeń, szczególnie jeżeli chodzi o generowanie płynnej animacji w grach komputerowych, w czasie rzeczywistym.

Główny procesor komputera nie jest w stanie nadążyć z generowaniem obrazu (zwłaszcza gdy zajmuje się np. obsługą zdarzeń w grze).

Większość operacji została przeniesiona na procesor karty graficznej (GPU).

Przyspieszanie sprzętowe

Przyspieszanie sprzętowe grafiki
(*hardware accelerated graphics*)

- procesor główny (CPU) przesyła wywołanie funkcji graficznej do układu na karcie graficznej
- procesor na karcie graficznej wykonuje sprzętowo operacje tworzenia obrazu, bez udziału CPU
- implementacja sprzętowa procedur graficznych pozwala zwiększyć szybkość tworzenia grafiki i odciąża CPU

Bit blit (BitBLT)

Bit blit to jedna z pierwszych operacji, które zaimplementowano sprzętowo w układach graficznych. Pozwala wykonywać operacje na obrazach rastrowych.

Klasyczne gry 2D:

- stałe tło (zapisane w pamięci),
- ruchomy obiekt (*sprite*) i jego maska

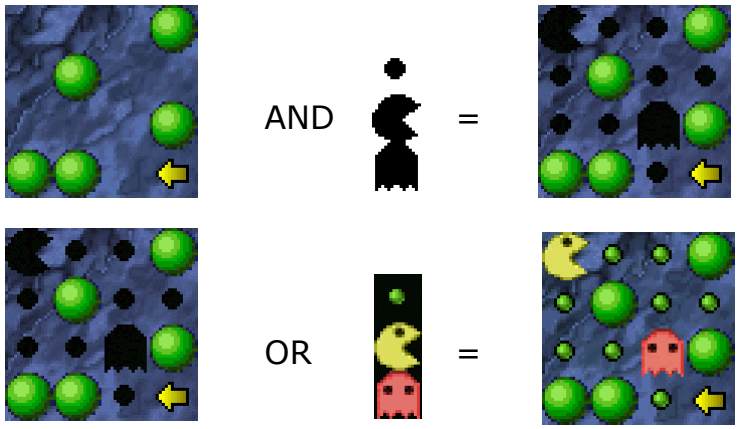
Narysowanie obiektu na tle:

- połączenie maski i tła – operacja OR
- połączenie wyniku ze *spritem* – op. AND

Blitting jest robiony sprzętowo – przyspieszenie.

Bit blit (BitBLT)

Ilustracja *blittingu*:



Podwójne buforowanie

Podwójne buforowanie (*double buffering*)

- Tworzenie grafiki „bezpośrednio na ekranie” powoduje zniekształcenia (np. migotanie) z powodu stałego odświeżania ekranu.
- Podwójne buforowanie:
  - dodatkowy obszar (bufor) pamięci,
  - tworzenie grafiki w tym buforze,
  - przeniesienie zawartości całego bufora na ekran w jednym kroku
  - implementacja sprzętowa – przyspieszenie
  - można użyć dwóch przełączanych buforów

Przyspieszanie grafiki 2D

Przyspieszanie sprzętowe grafiki w kartach graficznych PC:

- wprowadzone w latach 90.
- sprzętowa implementacja rysowania prymitywów 2D, np. prostokątów
- przyspieszało rysowanie np. okienek w systemie operacyjnym (*windows accelerators*)
- w chwili pojawienia się gier 3D przyspieszanie sprzętowe tego typu okazało się niewystarczające

Procesory GPU

GPU – *Graphics Processing* Unit

Wyspecjalizowany procesor graficzny znajdujący się np. na kartach graficznych komputerów PC.

Realizuje sprzętowo operacje związane z obrazem:

- przyspieszanie grafiki 2D (płaskie prymitywy)
- tworzenie grafiki 3D (rendering)
- obsługa wideo (np. przechwytywanie ramek)

Akceleracja 3D

Wsparcie sprzętowe dla gier 3D

- Akceleratory 3D – dodatkowe karty graficzne (*3dfx Voodoo*)
- Karty graficzne nowej generacji – GPU realizuje sprzętowo algorytmy grafiki 3D. Najpopularniejsze układy GPU:
  - *GeForce* (firmy *NVidia*),
  - *Radeon* (firmy *ATI*/*AMD*)

Programowanie grafiki

Dawniej producenci gier musieli sami zajmować się generowaniem grafiki na ekranie komputera.

Wprowadzono biblioteki do tworzenia grafiki 2D, będące częścią systemu operacyjnego lub zewnętrzne, np.:

- GDI, GDI+ (Windows)
- Cairo (Linux)
- rozwiązania uniwersalne (wieloplatformowe)

Biblioteki te ułatwiają tworzenie grafiki 2D, ale nie nadają się do grafiki 3D

Programowanie grafiki

Obecnie programiści korzystają z gotowych bibliotek programistycznych (SDK).

Najpopularniejsze systemy do grafiki 3D:

- *Direct3D* (część *DirectX*) – Microsoft, systemy Windows
- *OpenGL* – dostępny dla wielu systemów operacyjnych.

Część funkcji *DirectX* i *OpenGL* jest implementowanych sprzętowo w układach GPU - przyspieszenie tworzenia obrazu, odciążenie CPU.

Direct3D sprzętowo i programowo

Jeżeli twórca gry komputerowej chce uzyskać określony efekt graficzny:

- pisany jest program z wykorzystaniem np. *DirectX*
- system sprawdza czy GPU na karcie graficznej wspiera sprzętowo daną funkcję
- jeżeli tak – GPU realizuje tę funkcję sprzętowo (przyspieszanie)
- jeżeli nie – CPU musi „emulować” te funkcję (brak przyspieszania) – wolniejsze tworzenie grafiki (lub efekt jest wyłączany)

Direct3D - wczesne wersje

Początkowe wersje *DirectX* (do wersji 5.0) zawierały zbiór procedur graficznych (API) do tworzenia grafiki 3D. Możliwości *DirectX* nie były w tym czasie implementowane sprzętowo w kartach graficznych.

Możliwości sprzętowe ówczesnych GPU:

- obsługa siatek wielokątowych,
- filtrowanie tekstur, mipmapping
- bufor głębokości (*z buffer*) i pamięć tekstur

Karty: *NVidia Riva 128; ATI Rage*

DirectX 6

Direct3D 6.0 (1998)

- optymalizacja potoku przetwarzania,
- obsługa wielu tekstur (*multitexture*),
- bufor maski (*stencil buffer*),
- kompresja tekstur (S3 lub DXTC)

W kartach graficznych: obsługa obrazów 32-bitowych, 24-bitowy bufor głębokości, filtracja trójliniowa tekstur.

Karty: *NVidia Riva TNT*, *TNT2; ATI Rage 128*.

DirectX 7

Direct3D 7.0 (1999)

- sprzętowa realizacja *transform & lighting* (przekształcenia i oświetlenie)
- format tekstur .dds
- sprzętowy bufor pamięci wierzchołków
- lepsza obsługa tekstur (*multitextures*)

Karty graficzne: *NVidia GeForce 256*, *Geforce 2*; *ATI Radeon R100*

Transform and Lighting

Transform & Lighting (T&L) – zbiór operacji dotyczących:

- przekształcania modelu 3D (*transform*)
  - przekształcenia wierzchołków siatki,
  - konwersja współrzędnych do widoku 2D
- oświetlenia sceny
  - nakładanie tekstur
  - cieniowanie
  - inne efekty związane z oświetleniem

Transform, Lighting & Clipping

Transform, Lighting & Clipping (TLC)

- rozszerzenie T&L o obcinanie widoku (*clipping*) do obszaru widocznego na ekranie, w tym również usuwanie niewidocznych powierzchni.

Przed *Direct3D 7.0* operacje TLC musiał wykonywać procesor główny (CPU).

Przesunięcie tych operacji do GPU stanowiło prawdziwą rewolucję w grafice komputerowej.

DirectX 8

Direct3D 8.0 (2000)

Wprowadzenie programowalnych jednostek:

- *vertex shader* – operuje na wierzchołkach (werteksach) siatki wielokątowej;
- *pixel shader* –operuje na pikselach obrazu (wyznacza ich barwę).

Ponadto sprzętowe algorytmy: *bump mapping*, *texture mapping*, efekt mgły

Karty graficzne: *NVidia GeForce 3*, *GeForce 4* (sprzętowy antyaliasing); *ATI Radeon R200*.

Shader

Shader – zestaw instrukcji programu, za pomocą którego programista wpływa na przebieg renderingu obrazu 3D.

Shadery pozwalają programiście kontrolować potok renderingu w procesorze graficznym.

Inaczej mówiąc, shadery pozwalają programistom pisać programy na GPU.

DirectX udostępnia trzy rodzaje shaderów: vertex, pixel, geometry.

Vertex shader

Vertex shader – dowolny algorytm, który wykonuje operacje na siatce wielokątowej – modyfikuje współrzędne werteksów. Każdy werteks można modyfikować oddzielnie.

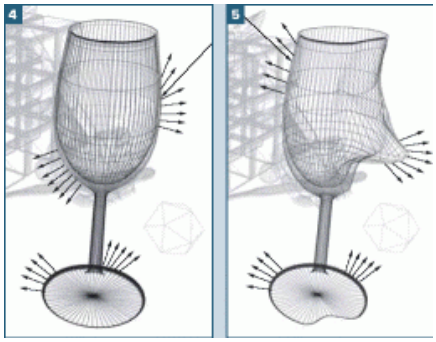
Możliwe jest modyfikowanie współrzędnych werteksu (x, y, z), barwy, tekstury, sposobu oświetlenia (modyfikacja wektora normalnego).

Algorytm jest realizowany sprzętowo, przy użyciu jednostki *vertex processor*.

Vertex shader

Jednostki *vertex shader* umożliwiają uzyskiwanie różnorodnych efektów, m.in.

- zniekształceń obiektów (np. uszkodzeń),
- ruchu powierzchni wody,
- mimiki twarzy, itp.



Pixel shader

Pixel shader (fragment shader) to algorytm pozwalający w dowolny sposób modyfikować barwę każdego z pikseli obrazu podczas rasteryzacji.

Działanie tych algorytmów jest zwykle powiązane z zastosowaniem modelu oświetlenia podczas renderingu sceny.

Algorytmy są realizowane sprzętowo, przy użyciu jednostki *pixel processor*, który zwykle musi wykonać znacznie więcej obliczeń niż jednostka *vertex processor*.

Pixel shader

Zastosowanie jednostek *pixel shader* pozwala uzyskać efekty związane z oświetleniem, takie jak:

- chropowatość powierzchni (*bump mapping*),
- fale na wodzie,
- odbicia w wodzie,
- cienie,
- efekty eksplozji,
- tonowanie barwy

DirectX 9

Direct3D 9 (2002-2005)

- udoskonalenia shaderów
- język programowania shaderów – HLSH (*high level shader language*)
- modele oświetlenia o wysokiej rozdzielczości
- tekstury zmiennoprzecinkowe
- udoskonalenie bufora werteksów (indeksy)
- efekty *Aero* w Windows Vista

Karty: *NVidia GeForce FX* (5), 6, 7, 8;
ATI Radeon R300; R420; R520

Udoskonalenia kart graficznych

Różnice pomiędzy kartami graficznymi kolejnych serii (np. GeForce FX, 6, 7, 8):

- większa częstotliwość procesora, więcej pamięci,
- nowsze jednostki shader (*shader model*),
- optymalizacja algorytmów (np. lepsze filtrowanie tekstur),
- dodatkowe algorytmy (SLI, *antialiasing*),
- zwiększenie liczby potoków renderingu.

DirectX 10

Direct3D 10 (2006), 10.1 (2008)

- *geometry shader*
- zintegrowane trzy rodzaje *shaderów*
- macierze tekstur (podmiana tekstur przez GPU)
- udoskonalenia jednostek *shader*, języka programowania i innych czynników

Dostępny tylko dla Windows Vista.
Karty: *NVidia GeForce* 8, 9 GTX200;
ATI Radeon R600, R700

Geometry shader

Geometry shader pozwala modyfikować siatkę wielokątową obiektu:

- dodawanie nowych werteksów
- usuwanie werteksów
- sprzętowe zwiększanie rozdzielczości siatki (interpolacja siatki)
- operuje głównie na grupach werteksów (np. na pojedynczym trójkącie)
- wykorzystanie np. do tworzenia cieni (*shadow volume*) i mapowania tekstur w technice *cube mapping*.

Unified Shader Architecture

Do tej pory trzeba było programować oddzielnie każdy z typów shadera, nieraz przy użyciu różnych narzędzi programistycznych.

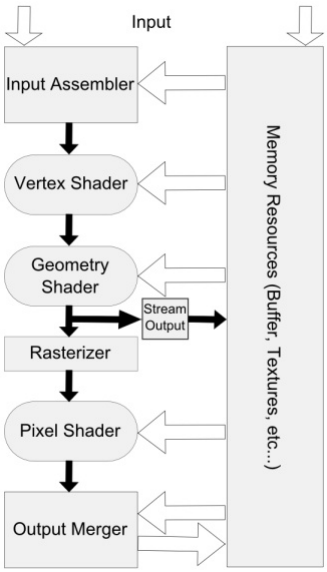
Unified Shader Architecture – wprowadzona w *DirectX* 10 zintegrowana jednostka „3 w 1”: *vertex*, *pixel* i *geometry shader*, działająca w trybie zmiennoprzecinkowym.

Pozwala programować wszystkie shadery w jednakowy sposób.

Stosuje się również termin *Shader Model 4.0*. Obsługa również z poziomu OpenGL.

Potok renderingu w Direct3D 10

- *Input Assembler* - dostarcza dane
- *Vertex shader* - transformacje wierzchołków siatki
- *Geometry shader* - przetwarzanie prymitywów (grup werteksów)
- *Stream output* - zachowanie stanów pośrednich w pamięci
- *Rasterizer* - przekształcanie obiektów 3D w obraz 3D, przycinanie
- *Pixel shader* - operacje na pikselach obrazu 2D (modyfikacje barwy)
- *Output merger* - łączy wyniki działania różnych procedur w obraz końcowy



OpenGL

OpenGL jest systemem wieloplatformowym, dla różnych systemów operacyjnych.

Jest to tylko specyfikacja – zbiór funkcji i opis ich działania. Producenci sprzętu muszą zadbać o poprawną implementację tych funkcji.

OpenGL definiuje bibliotekę funkcji (API) niskiego poziomu. Wymaga od programisty podania kolejnych kroków renderingu obrazu.

OpenGL ARB (*Architecture Review Board*) – konsorcjum kierujące pracami nad standardem.

Programowanie w DirectX i OpenGL

Direct3D

- programowanie za pomocą DirectX API (języki C++, C#, VB, itp.)
- istnieją też biblioteki pomocnicze (np. *Allegro*)

OpenGL

- najczęściej korzysta się z pomocniczych bibliotek - nakładek (np. *GLU*, *GLUT, WGL*)
- alternatywne implementacje - np. *Mesa3D*
- wsparcie dla większości języków programowania

Programowanie shaderów

Programowanie jednostek *shader* (karty graf.) – specjalistyczne języki programowania:

- Cg (NVidia) – Direct3D, OpenGL
- HLSL – *High Level Shader Language* (Microsoft) – Direct3D
- GLSL – *OpenGL Shading Language* (OpenGL ARB)

Można również stosować asembler.

Program główny (dla CPU) pisany jest w wybranym języku programowania, natomiast programowanie GPU - w jednym z ww. języków.

Przetwarzanie równoległe

Pojedynczy potok renderingu (*rendering pipeline* lub *pixel pipeline*) jest w stanie obliczyć kolor tylko jednego piksela obrazu w danej chwili.

Zwiększenie liczby potoków pozwala obliczać kilka pikseli obrazu jednocześnie. Współczesne karty graficzne:

- od 4 potoków (np. *GeForce 6200)*
- 24 potoki (np. *GeForce 7900 XT*)
- najszybsze karty: 240 potoków (*GeForce 280 GTX*)

Modele kart z danej serii mogą się różnić liczbą potoków (np. *GeForce GS, GT, GTX*).

Najszybsze karty graficzne

Styczeń 2009 – najszybsze karty graficzne na rynku:

- NVidia
  - GeForce GTX 260, GTX 280
  - GeForce GTX 285, GTX 295
- ATI/AMD
  - Radeon 4850, 4870

Nowoczesne karty graficzne mogą być łączone w pary (SLI, Dual GPU).

Przetwarzanie równoległe na GPU (GPGPU)

Moc GPU można wykorzystać nie tylko do grafiki, ale również do przyspieszania obliczeń równoległych, wykorzystując shadery:

- początkowo - konieczność „oszukiwania” (tworzenia sztucznych shaderów)
- *NVidia CUDA*:
  - darmowe SDK (język C), pozwala wykonywać obliczenia na GPU
  - pluginy do Matlaba
  - wymaga karty graficznej GeForce 8800 lub nowszej albo dedykowanego sprzętu (*NVidia Tesla*)

Game engine

Game engine („silnik gry”) – część oprogramowania wykorzystywana do tworzenia gry. Obejmuje m.in.:

- rendering,
- modele fizyczne,
- modele sztucznej inteligencji,
- skrypty,
- obsługę dźwięku i muzyki,
- interfejs użytkownika

Modele fizyczne

Model fizyczny (*physics engine*) służy do realistycznego odwzorowania praw fizyki w grach.

- Modele fizyczne w grach są z konieczności uproszczone (praca w czasie rzeczywistym).
- Zwykle wykorzystuje się uproszczone siatki wielokątowe obiektów do wykrywania zderzeń i modyfikowania struktury obiektów.
- Model fizyczny „zwalnia" programistę od przejmowania się prawami fizyki w komputerowym świecie.

PhysX

PhysX – model fizyczny opracowany przez firmę *Ageia*, obecnie własność firmy *NVidia*.

Implementacja:

- sprzętowo – specjalistyczny procesor PPU (*Physics Processing Unit*)
- na procesorze GPU (w ramach *CUDA*)
- programowo (realizowane przez CPU).

Typowe efekty:

- eksplozje, odłamki obiektów
- efekt cienkiego materiału
- dym i mgła, reakcja na ruch postaci

Havok Physics

Havok – inny model fizyczny (*physics engine*) wykorzystywany we współczesnych grach

- opracowany przez firmę *Havok*, obecnie własność firmy *Intel*,
- lansowany przez AMD w kartach ATI
- wykorzystywany w wielu grach, dostępny na wielu platformach (PC, konsole, itp.)
- gra *Second Life* – model uruchamiany na serwerach, nie na komputerach graczy
- może współpracować z innymi modułami *Havok*, np. modelem zachowania się postaci