Dr. Mileff Péter

GRAFIKA PROGRAMOZÁSA

GRAFIKUS PROCESSZOROK GRAFIKUS ÉS JÁTÉKMOTOROK

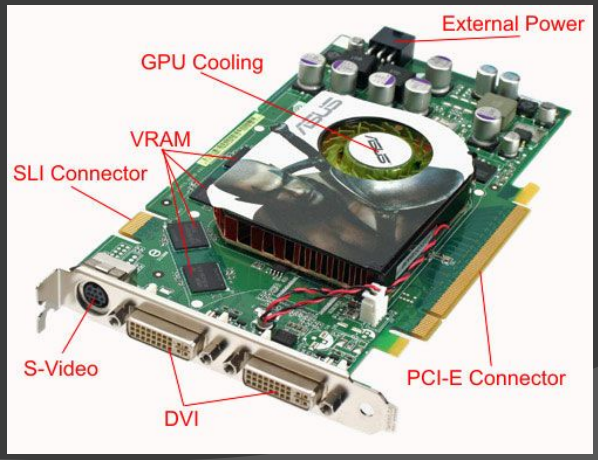
Miskolci Egyetem
Általános Informatikai Tanszék

GPU ÁTTEKINTÉSE…

GPU áttekintése

- A **Graphics Processing Unit (GPU)** a grafikus vezérlőkártya központi egysége
- **Célja:**
  - összetett grafikus műveletek elvégzése
  - A megjelenítés közvetlen gyorsítása
  - **CPU tehermentesítése:**
    - megjelenítéssel kapcsolatos magas szintű feladatok átvétele a CPU-tól
    - annak számítási kapacitása más műveletek elvégzésére legyen felhasználható
- A GPU-k térnyerése párhuzamosan volt jelen a szoftveres renderelés mellett
- Ennek indítója:
  - a hardvergyártók hamar felismerték az üzleti lehetőségeket:
    - a multimédiás alkalmazásokban, a mérnöki rendszerekben és a videojátékokban

Videokártya



Első sikerek

- **A 3dfx cég kiadja 1996-ban kiadta Voodoo I-et**
- **Jellemzői:**
  - Első hardveres gyorsító kártya
  - Hatalmas siker
  - Csak **3D**-s képeket állított elő
  - Szükség volt mellé egy **2D**-s kártyára is.
- **Az ötlet:**
  - a 2D-s leképezéseket egy jó minőségű 2D-s videokártya végzi.
    - Pl. az akkor népszerű Matrox kártya
  - a 3D-s információkat, a Voodoo I-nek adtak át
    - gyors hardvere feldolgozta és elvégezte a szükséges számításokat.

Következmények

- **Ugyanebben az évben:**
  - NVIDIA és az ATI is elindították saját GPU sorozatukat.
  - Nvidia: NV1, RIVA 128, Geforce 256
  - ATI: 3D Rage, Rage Pro, Rage 128
- A gyorsítókártyák rögtön nagyon népszerűek lettek
- **Oka:**
  - Kedvező ár
  - Könnyű beszerezhetőség
  - Játékok szinte azonnali támogatása

GPU ARCHITEKTÚRÁJA…

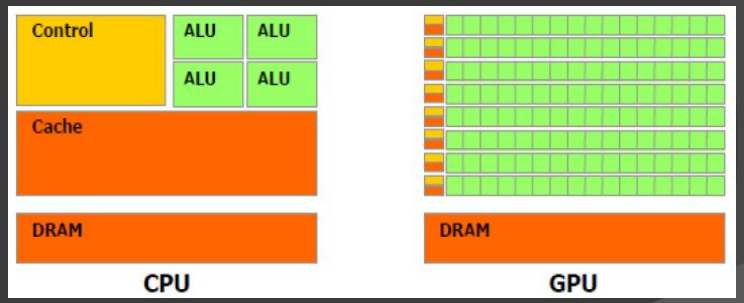
CPU vs GPU

- A GPU felépítése erősen különbözik a központi egység architektúrájától (már a kezdetektől is!)
- **Ok 1:**
  - **speciális célra lettek tervezve:** tipikusan a grafikus számítások gyorsítására
    - A grafikai számítások eltérő igényekkel rendelkeznek, mint a központi egységgel támasztott követelmények.
  - A CPU-nak az általános célokat kell szolgálnia
  - A GPU-k számára kezdetben elég volt csak a grafikai számításokkal kapcsolatos műveletek gyorsítása
- **Ok 2:**
  - a grafikai számítások, a raszterizáció folyamata erősen párhuzamosítható,
  - így a GPU-k fejlődése ebbe az irányba indult el.

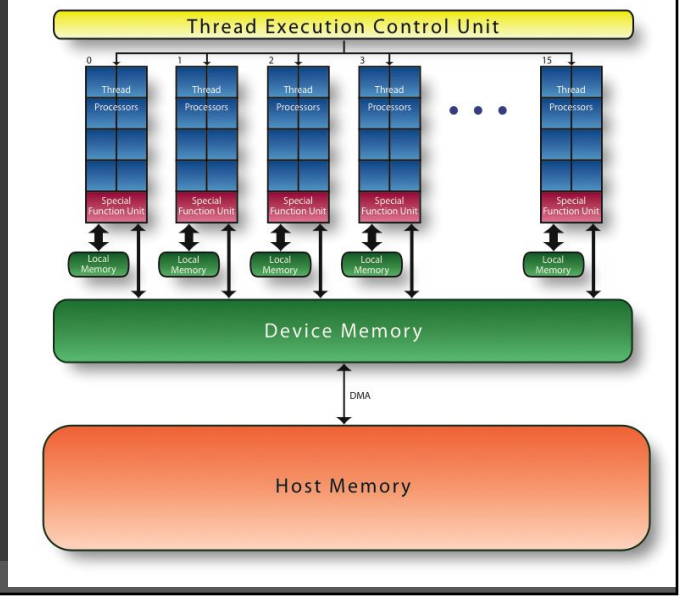
CPU vs GPU

- **CPU:** egy egyszálas számítási architektúrát valósít meg
  - lehetővé teszi több folyamat időosztásos futtatását ezen az egyszálas csővezetéken,
  - az adataikat egyetlen memória interfészen keresztül érik el
- **GPU:** teljesen más architektúrát, az **adatfolyam feldolgozást (stream processing)** követik
  - Hatékony azonos művelet nagy mennyiségű adaton való végrehajtására (SIMD modell)
  - Egy GPU-ban akár több ezer ilyen adatfolyam processzor is van
    - egy dedikált csővezeték processzort (dedicated pipeline processor) kapunk
  - Nincs ütközés és várakozás mint a CPU-nál,
    - az adatfolyam processzorok egy csővezetéket alkotnak

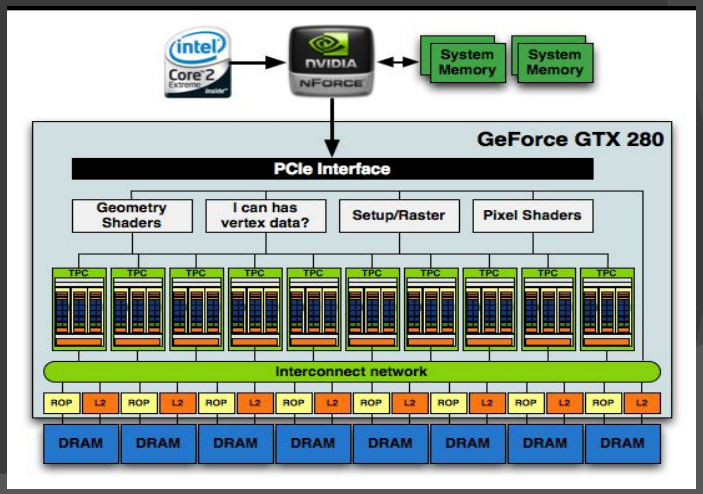
CPU vs GPU



GPU



Geforce GTX 280



CPU vs GPU

- **CPU:** erőforrásai nagy részét a programok vezérlésére, a gyorsabb utasítás-kiválasztásra fordítja.
- **GPU:** erre teljesen alkalmatlan.
  - sok aritmetikai logikai egységekkel (ALU) van felszerelve,
  - így nagyságrenddel gyorsabban képes számolni.
  - **Korlátai:**
    - az, hogy az egyes feldolgozó egységeken azonos utasítások fussanak. **– Adat párhuzamosság!**
- **CPU is támogatja az adatpárhuzamosságot!**
  - utasításkészlet bővítésekkel (pl. SSE, SSE2, SSE3, SSE4, AVX, ALTIVEC, stb),
  - többmagos processzorokkal
  - de összehasonlítva a GPU-val ez még elmarad tőle.

Az adattovábbítási probléma…

Adatbusz

- **Hamar látszott az adattovábbítási probléma**
  - az adatok CPU és GPU közötti megfelelő sebességű továbbítása
  - a szabványos PCI busz sávszélessége továbbra is egy szűk keresztmetszet
    - valósidejű vizualizációval szemben magas elvárások
- **Miért van a növelésre szükség?**
  - Amikor már feltöltöttük az adatot a GPU memóriájában, a vizualizáció megfelelően gyors
  - Van azonban amikor gyakran szeretnénk cserélni vagy módosítani a GPU memóriában lévő adatot
    - Pl. darabokra törő fal, textúra festése, stb

Adatbusz

- 1997-ben kidolgozták az **AGP (Accelerated Graphics Port)** 1.0-ás szabványt
- Nagyságrendekkel gyorsabb adatátvitelt tett lehetővé a grafikus kártya és az alaplap között.
  - Napjainkban még mindig jelen van az AGP szabványa
- Ma már a **PCI Express** átviteli megoldás az egyeduralkodó.

PCIe 1.0	PCIe 2.0	PCIe 3.0	PCIe 4.0
250 MB/s	500 MB/s	984.6 MB/s	1969.2 MB/s

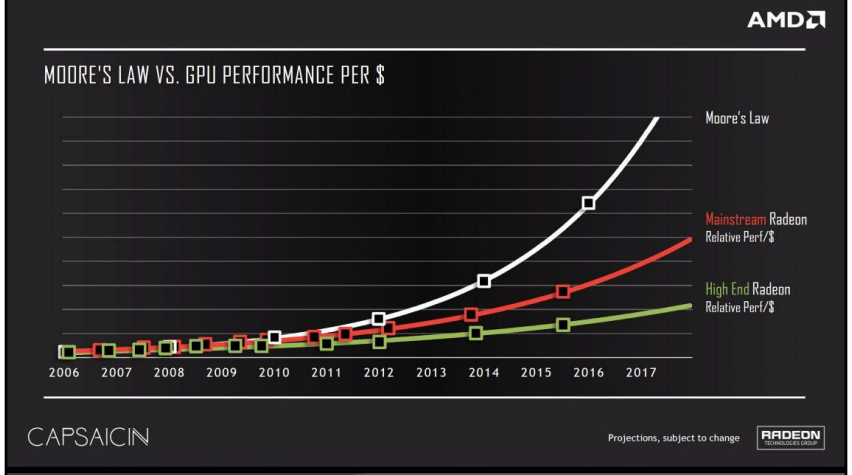
PCIe 4.0 - végleges specifikáció kb 2017-re

Tendencia

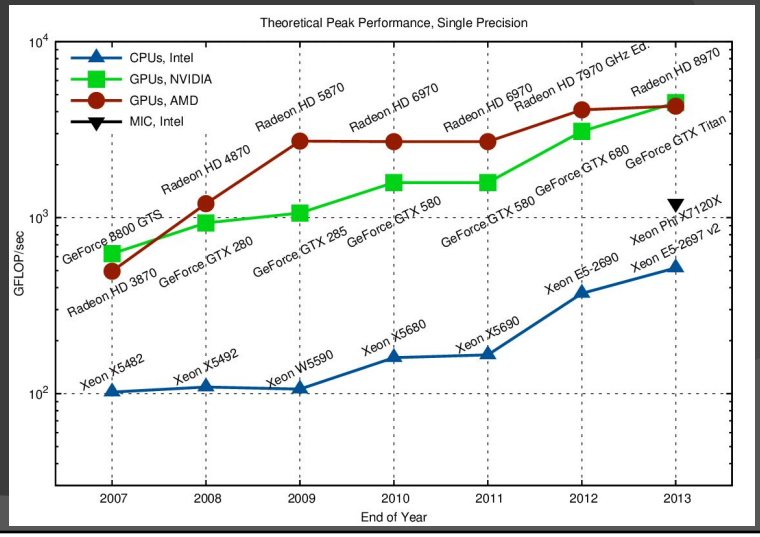
- A GPU-k a fejlődési sebessége messze meghaladja a CPU-k fejlődését
- **Moore törvény:** az egységi felületre elhelyezhető tranzisztorok száma duplázódik minden 12 hónapban
  - CPU: a sebesség lelassult 18 hónapra
  - GPU: duplázódás sebessége 6 hónapra csökkent
- Példa:

  ATI Radeon HD 3800 GPU család:
  - 320 adatfolyam processzor
  - 666 millió tranzisztor
  - Teljesítmény > 1 terraFLOPS
- Intel Core 2 Quad CPU
  - 582 millió tranzisztor
  - 9.8 gigaFLOPS

Tendencia



Tendencia



2014 08 16 [date stamp from Google search, but it must be earlier than 2014 11 17 according to page 49 of the PDF]

GPU Motivation (I): Performance Trends

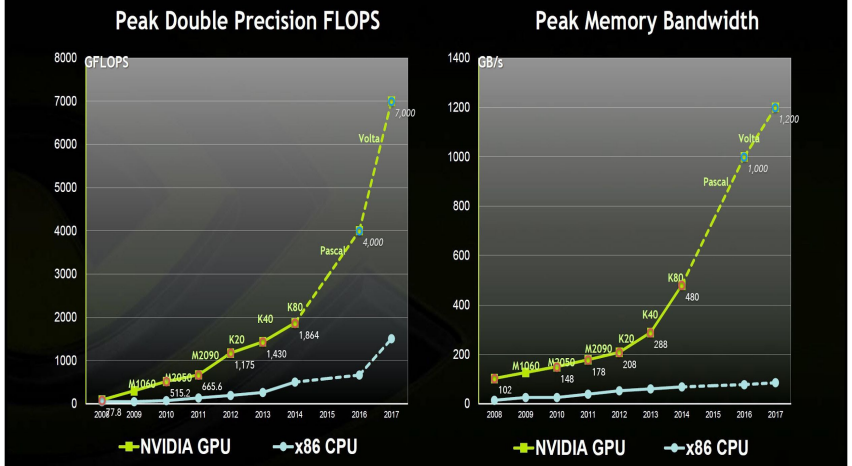


Table overlays: Theoretical DP GFLOPS and bandwidth of NVIDIA Tesla cards. *Light grey italic text* represents my guesses.

http://www.ecmwf.int/sites/default/files/HPC-WS-Posey_0.pdf (page 7)

GPU PROGRAMOZÁSA…

Programozási felületek

- A videokártyák fejlődésével párhuzamosan készültek különböző alacsony szintű programozási felületek (API).
  - Hardvergyártók erős befolyása alatt
- **Első ismert API: Glide API**
  - 3dfx fejlesztette a saját Voodoo kártyáihoz
  - OpenGL-hez hasonló felület
  - Teljesítmény és funkcionalitás szempontjából a játékokat célozta meg.
  - 1990-es évek közepén abszolút domináns a játékiparban
  - 2000-ben Nvidia felvásárolja a 3dfx-et

DirectX vs OpenGL

- **DirectX**
  - Microsoft grafikus API-ja
  - Kizárólag a MS platformok grafikus gyorsítása
    - Windows, Xbox, Windows Phone
  - A legkedveltebb grafikus API a játékfejlesztők körében.
- **Népszerűségének oka:**
  - a fejlesztése tökéletesen lépést tart a grafikus hardver fejlődésével
  - Rendelkezik beépített magasabb szintű megoldásokkal is. Pl.
    - Optimalizált matematikai lehetőségek: pl. mátrix, vektor, stb
    - Saját X modell formátum
  - Egyéb kiegészítő magasabb szintű API-k: DirectDraw, DirectInput, DirectSound, stb

Direct3D vs OpenGL

- **OpenGL (Open Graphics Library):**
  - Platformfüggetlen 2D és 3D grafikai API szabvány specifikációja
  - 1992-ben jelent meg a Silicon Graphics Inc által
  - A szabvány fejlesztéséért az ARB (Architecture Review Board) konzorcium volt a felelős,
    - tagjai a legfontosabb szoftverfejlesztő és hardver gyártó cégek (ATI. NVIDIA. Intel. Microsoft, stb.)
  - 2006-ban a **Khronos Group** konzorcium vette át.
  - **Lassabb fejlesztés:** a specifikáció fejlesztése lassú folyamat, amely jelentősen hátráltatja a grafika-intenzív alkalmazások fejlesztőit.
- **OpenGL Extension Library:** orvosolja a lassú fejlesztést
  - A hardver gyártó és szoftverfejlesztő cégek készíthetnek kiterjesztéseket,
    - tartalmazzák az új funkció használatához szükséges függvényeket és konstansokat.

Direct3D vs OpenGL

- Igazi konkurensek a játékfejlesztés területén
- Mindkét API-nak megvan a maga előnye illetve hátránya
  - De elsősorban csak strukturális különbségek vannak a kettő között, funkcionalitásában a két API majdnem azonos.
    - Alapvetően ugyanannak a hardvernek a funkcióihoz biztosítanak interfészt
- **Az OpenGL előnye (a jövő):**
  - Platformfüggetlenség: lehetőséget adva így tetszőleges eszközön való alkalmazására
  - A beágyazott rendszerek számára az OpenGL egy szűkített változatát, az **OpenGL ES**-t dolgozták ki.
  - A mai virágzó mobil eszközök GPU világában így egyre inkább egyeduralkodóvá válik az OpenGL a DirectX-el szemben.
    - TEGRA, ARM Mali, Adreno, AMD Z430, PowerVR sorozatok

WebGL

- Az informatikai eszközök fejlődése:
  - **HTML5 + Javascript** alapú technológiák kerültek előtérbe
  - 2011-ben kialakult a **WebGL** (Web-based Graphics Library)
- **Egy programkönyvtár**, amely
  - kiegészíti a JavaScript-et
  - OpenGL alapú hardveresen gyorsított grafikát nyújt,
  - futási környezete a böngésző
- **Előnyei:**
  - nincs szükség fordító programra, egy szövegszerkesztő elegendő hozzá
  - Az alkalmazás kódját valós időben szerkeszthetjük a böngészők Javascript/Debug konzolainak segítségével
  - Kiválóan alkalmas különböző demonstrációs alkalmazások készítésére, egy-egy grafikai koncepció bemutatására

WebGL

- **Hátrányai:**
  - a WebGL-el készült alkalmazás kódja és a különböző tartalmai publikusak
    - nem lehet védeni attól, hogy valaki megnézze a weboldal forráskódját
  - A fejlesztők nem feltétlenül szeretnék forráskód szinten terjeszteni alkalmazásukat.
    - egy komplexebb alkalmazás kódszintű megértése nagyon sok erőfeszítést igényel, de ettől még a készítők aggodalma jogos lehet
  - a sebesség sokszor nem kielégítő
  - nehezebb debugolni, ami jelentősen megnövelheti a fejlesztési időt
  - Az adatokat le kell tölteni a kliens gépére, amely lassítja az (első) indulást

GRAFIKUS ÉS JÁTÉKMOTOROK...

Áttekintés

- Napjainkban egyre hangsúlyosabb szerephez jutnak a grafikus és játékmotorok
- Nem véletlen a két csoport megkülönböztetése.
  - **Alapja a Funkcionalitás**
- **Grafikus motorok:** kizárólag a képi megjelenítésben nyújtanak segítséget
- **Játékmotorok:** egy nagyobb alrendszer halmazt foglalnak magukban.
  - pl. Hang, Hálózat, Eszközkezelés, Input, stb.
  - támogatva ezzel a játékfejlesztés igényelte sokrétűséget.

Játékmotor feladata

- **A játékmotor célja 1:** egy eszközrendszer biztosítása a fejlesztők (programozó, designer, teszter, stb.) számára,
  - Pl. szerkesztők, futtató környezet, hálózat, hang, stb.
- Hatékony, kényelmes és gyors játékfejlesztés válik lehetővé.
- Egy réteg az operációs rendszer és a játék logika között.
- Egyszerűsíti a rutin programozási feladatokat:
  - amiket egyébként minden játék készítésekor a programozóknak kellene kifejleszteni.
  - Pl. ablakfeldobás, audio, videó lejátszás, modellek betöltése, stb.
- **A játékmotor célja 2:** a megfelelő technikai színvonal képviselése
  - mind a grafikai minőségben mind pedig sebességben.

Játékmotor felépítése

- A játékfejlesztés egy komplex informatikai tudást igénylő folyamat!
  - Egy motor ennek támogatására komplex funkcionalitással kell rendelkezzen.

Ezeket jól körülhatárolt alrendszerekbe szervezik:

- **Core alrendszer:** a fő rendszermag, összefogja a többi egységet. Platformfüggetlenség biztosítása, események továbbítása más alrendszerek felé.
- **Megjelenítő alrendszer:** a képi megjelenítésért felelős rendszer. Tipikusan valamely API (OpenGL, DirectX)-ra épül.
  - Modellek megjelenítése, fények, effektek, utófeldolgozás (post processing), részecske rendszer, stb.

Játékmotor felépítése

- **Hang és zene alrendszer:** hangok és zenék támogatása
- **Mesterséges intelligencia alrendszer**
- **Hálózati alrendszer:** hálózati kapcsolatok támogatása
- **Inputkezelő és eseménykezelő alrendszer:** beviteli egységek, események kezelése
- **Szkriptrendszer:** szkript vezérlés támogatása
- **Erőforrás-kezelő alrendszer:** hozzáférés az erőforrásokhoz
- **Fizikai alrendszer:** fizikai szimulációk támogatása
- **Egyéb alrendszerek:** számításokhoz, videó lejátszáshoz, stb.

Játékmotor felépítése

- **Az alrendszereknek célszerű cserélhetőnek lennie**
- Előfordul, hogy egy-egy alrendszer nem saját fejlesztésű
  - a cégek dönthetnek úgy, hogy egy már létező és jól működő technológiát vásárolnak meg.
  - Ha az új alrendszer kifejlesztése többe kerülne mint egy már meglévő beszerzése.
    - Tipikus példa a Havok fizikai rendszer integrálása
    - Külső GUI megoldások (pl. CEGUI)
- A mai főbb játékmotorokat megvizsgálva jól látszik a modularitás.
  - A főbb komponenseket dll-ben készítik el (leginkább C++-ban),
  - A játéklogikát pedig egy magasabb szintű nyelven
    - Pl. C# -ban dll-ként használva az adott komponenseket.

Játékmotor felépítése

- **Miért nem jó mindenhová a C++?**
  - Valójában megfelelő, de:
    - a nyelv komplexitása miatt magasabb programozói tudást igényel
    - C#, Java-ban könnyebb kódolni
      - Csökken a hibák kijavításához szükséges átlagos idő
    - A fejlesztő környezetek (pl. Eclipse, IntellJ) kiforrottabbak,
      - számos esetben több segítséget nyújtanak ha ilyen nyelveken programozunk
    - A játékiparban az egyik legfontosabb cél, hogy a szoftver minél gyorsabban készüljön el a lehető legkevesebb hibával.
      - Magasabb szintű nyelvek ehhez pedig jól hozzájárulnak

VEZETŐ FEJLESZTÉSEK, MAI TRENDEK...

Vezető fejlesztések

- A technológiának köszönhetően a grafikus és játék motorok már pazar látványt képesek nyújtani.
- A játékok egyre komplexebbek, egyre több cinematikus elemet tartalmaznak.
  - Nem véletlen tehát ha egy modern motorért ma már magas árat kell fizetni. Pl.
    - Unity Pro – 125$ per seat / month
    - Unigine Engine – 1495$/seat, 9995$/10 seat, etc
    - CryEngine 3 – ???
    - Unreal engine - 5% royalty after the first $3,000 of revenue per product per quarter
- Cserébe: a fejlesztők több évnyi tapasztalatot kapnak implementált algoritmusok formájában

Vezető játékmotorok

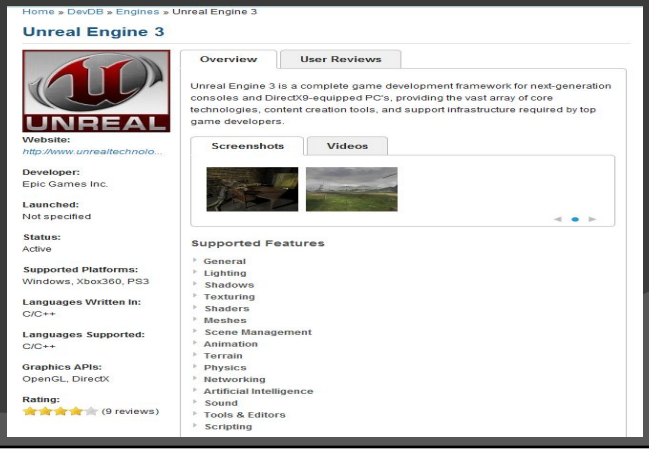
- Unreal Engine 4 - Epic Games
- Unigine Engine - UNIGINE Corp
- ID Tech 6 – ID Software
- Frostbite - EA Digital Illusions CE
- Cryengine 3 - Crytek
- Source – Valve
- Unity Engine - Unity Technologies
- ShiVa 3D - Stonetrip
- C4 Engine - Terathon Software
- Lumberyard Engine - Amazon

Új trendek

- **Új piac a motorok számára:** a mobil és táblagépek piaca.
  - Előtérbe került a **OpenGL ES**-t használó játékmotorok támogatása is.
- Számos fejlesztőeszköz jelent meg erre a platformokra
- Több nagy motort képviselő cég elkészítette programjának mobil változatát
  - pl. Unity, Unreal Engine, ShiVa 3D.
- Eszközrendszerük nagyon hatékony:
  - a fejlesztés akár PC-n folyhat, a mobil teszteléseket pedig az auto-deploy biztosítja.

Motorok adatbázisa

- http://devmaster.net/devdb/engines
- Részletes keresési lehetőség:
  - Licensz, Programnyelv, API, Funkcionalitás, Platform, stb



A JÁTÉKMOTOR ALAPJAI…

Játékmotorok alapjai

- A grafikai vizualizációnak két nagyobb csoportja különíthető el:
  - két és háromdimenziós megjelenítés
- **Két dimenziós vizualizáció:**
  - két dimenziós objektumokkal dolgozik. Két koordináta: *x* és *y*
  - A két dimenziós megjelenítés egyszerűbb és sokszor gyorsabb is (bizonyos esetekben kevesebb transzformáció).
- **Háromdimenziós leképzés**: szükség van a harmadik, *z* koordinátára is.
  - A PC iparban a játékok többsége a 3D irányba tolódott el,
- A 2D továbbra is fontos szerepet tölt be:
  - mobil és tábla eszközökön pedig különösen domináns
  - Oka: a gyengébb hardver eszközök

Platformfüggőség kérdése

- Bármilyen megjelenítő szoftver valójában egy, az operációs rendszere épülő új réteg.
- **Fontos feladata:** grafikus ablak és inputkezelés biztosítása
  - illeszkednie kell az operációs rendszer adta lehetőségekhez
- **Platformfüggőség:**
  - az „ablak feldobás” funkciója és az eseménykezelés minden operációs rendszeren egyedi megoldást kíván meg
  - A játékot, vagy a grafikus motort fejlesztő programozónak kell elkészítenie.
    - Ez plusz feladat
    - Ez az oka annak, hogy a mai játékok zöme csak egy platformot (Windows) támogat.
  - Függ az alkalmazott programnyelvtől. Pl. Java vs C++

Platformfüggőség kérdése

- Mivel a mai grafikus motorok többsége a gyorsaság kihasználása miatt **C,C++**-ban íródik.
  - Legalábbis a magja
- A nyelv lehetőséget ad az egyes platformok kezelésére:
  - előre definiált makrókkal, melyek az OS-t szimbolizálják
  - a programozó képes a fordítás folyamatát platformfüggően vezérelni.
  - A makró elnevezések függenek az éppen alkalmazott fordítótól (pl.: GCC, CLANG, INTEL, BORLAND, MICROSOFT, stb)

Platformfüggőség kérdése

- Főbb platformok makrói:

Makró	Platform
_WIN32	Windows 32 és 64 bit
_WIN64	Windows 64 bit
_WIN32_WCE	Windows CE
__linux__	Linux rendszerek
__APPLE__	Mac OS X
__FreeBSD__	Free BSD
__BEOS__	BeOS
__amigaos__	Amiga OS
__unix__	Unix
__sun	Solaris Sun OS

A makró pontos nevéről célszerű először a fordító dokumentációjából tájékozódni

Platformfüggőség kérdése

- A platformfüggőség fordítás lehetősége (C):
  - Példa Linux és Windows keresztfordításra

```
#ifdef WIN32
#include <windows.h>
#include "KeysWin.h"
#endif
 ….
#ifdef __linux__
#include "KeysLinux.h"
#endif
```


KIEGÉSZÍTŐ API-K…

Kiegészítő API-k

- Az alkalmazás számára szükség van egy, az operációs rendszer által biztosított **ablak**ra,
  - amelyben, vagy annak egy részében fog történni a vizualizáció.
- Ehhez társul az **eseménykezelés**: ablak mozgatása, egér kattintás, szálkezelés, stb.
- A programkód ezen részei tipikusan platformfüggők, egyedik
- **Megvalósítása történhet:**
  - a programozó által, felhasználva az operációs rendszer ablakozó és eseménykezelő API-ját (Win32, Linux – X11/GLX, stb).
  - Itt pontosan ismerni kell az operációs rendszer működését.
  - Egy újabb platform bevezetése esetén újabb „alacsony” szintű rutinok írása szükséges
- Évek során több kiegészítő API alakult ki
  - Támogatják az OS függő részeket

Kiegészítő API-k

- **SDL (Simple DirectMedia Layer):** ingyenes platformfüggetlen multimédia könyvtár.
  - Támogatása szerteágazó: audio, egér, billentyűzet, szálkezelés, 3D, 2D, stb.
  - Platformok: Linux, Windows, Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, QNX
  - Különösen kedvelt a játékfejlesztők körében

Példa SDL-el készült programokra:

- Unreal Tournament
- Doom3
- Quake4
- Quake Wars
- Civilization: Call to Power
- CryEngine Linux port

http://www.libsdl.org/

Kiegészítő API-k

- **SFML (Simple and Fast Multimedia Library):**
  - ingyenes C++ alapú multimédia könyvtár.
  - Célja az SDL-el egyezik.
  - Szintén gazdag funkcionalitással rendelkezik, az SDL fő vetélytársa.
  - Magasabb szintű mint az SDL

http://www.sfml-dev.org/

- **GLUT (The OpenGL Utility Toolkit):**
  - ablakozó rendszer független OpenGL-t segítő programcsomag.
  - **Célja:** az ablakfeldobás és eseménykezelés egyszerű támogatása
  - Főleg kisebb példa programok készítésére használják,
    - nagyobb szoftverek, játékok készítésére nem javasolt.

http://www.opengl.org/resources/libraries/glut/

Kiegészítő API-k

- **GLFW:** a GLUT-hoz hasonlóan egy kis méretű C függvénykönyvtár
  - OpenGL kontextus létrehozására, ablakok és események támogatására.
  - Rendelkezik kép, illetve hang támogatással is.

http://www.glfw.org/

- **GLEW (The OpenGL Extension Wrangler Library):**
  - Egy platformfüggetlen, nyílt forráskódú kiterjesztés függvénykönyvtár az OpenGL-hez
  - Különböző operációs rendszereken is azonos módon tudjuk elérni az OpenGL kiterjesztéseket
  - Alkalmazható a fenti API-kkal együtt is.

http://glew.sourceforge.net/

Kiegészítő API-k

- A megfelelő döntéshez célszerű kipróbálni mindegyik API-t.
- A NEHE oldalán egy-egy példa alkalmazás számos különböző verziója tölthető le
  - Pl. Linux, Windows, OSX, BEOS, SDL, MASM, JOGL, stb.

http://nehe.gamedev.net/

PÉLDÁK…

Egyszerű SDL példa

```
#include <stdio.h>
#include <SDL2/SDL.h>

int main(int argc, char* argv[])
 {
    int quit = 0;
    SDL_Window *window;
    SDL_Event event;

    SDL_Init(SDL_INIT_VIDEO);  // Initialize SDL2

    window = SDL_CreateWindow("SDL2 Window", SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED, 800, 600, SDL_WINDOW_OPENGL);
```


Egyszerű SDL példa

```
        while (!quit) {
        SDL_WaitEvent(&event);
        switch(event.type) {
            case SDL_QUIT:
              quit = 1;
              break;
               }

        // Close and destroy the window
        SDL_DestroyWindow(window);

        // Clean up
        SDL_Quit();
        return 0;
}
```

Egyszerű SFML példa

```
#include <SFML/Window.hpp>

int main(int argc, char* argv[])
    sf::Window window(sf::VideoMode(800, 600), "My window");

    while (window.isOpen()) {
        // check all the window's events that were triggered since the
        sf::Event event;
        while (window.pollEvent(event)) {
                if (event.type == sf::Event::Closed)
                window.close();
        }
    }
    return 0;
}
```


A GLM könyvtár...

GLM könyvtár

- A **GLM** (OpenGL Mathematics) egy C++ nyelvi alapokon fekvő matematikai függvény könyvtár
  - GLSL (OpenGL Shading Language) specifikációja alapján készült
- **Célja:**
  - matematikai formulák biztosítása, amelyek elnevezési konvenciói és funkciói azonosak a GLSL-ben specifikáltakkal
  - Támogatja a mátrix transzformációkat, vektorokat, véletlen számokat, zajt, és még számos fontos algoritmust, vagy funkciót
- **Miért lehet/van rá szükség?**
  - az OpenGL 3.0-tól a matematikai lehetőségek a fix funkcionalitású csővezetékkel együtt már nem részei az OpenGL szabványnak
    - pl. glRotatef, glTranslate, glScale, stb.
  - A GLM ezt pótolja
  - profi célokra is kiválóan alkalmas
  - úgynevezett “header only library”, nem igényel külön build-elést

GLM példa

```
#include <glm/vec3.hpp>
#include <glm/vec4.hpp>
#include <glm/mat4x4.hpp>
#include <glm/gtc/matrix_transform.hpp>

glm::mat4 camera(float Translate, glm::vec2 const & Rotate)
{
	glm::mat4 Projection = glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.f);
	glm::mat4 View = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f,-Translate));

	View = glm::rotate(View, Rotate.y, glm::vec3(-1.0f, 0.0f, 0.0f));
	View = glm::rotate(View, Rotate.x, glm::vec3(0.0f, 1.0f, 0.0f));

	glm::mat4 Model = glm::scale(glm::mat4(1.0f), glm::vec3(0.5f));

	return Projection * View * Model;
}
```

Használjunk vagy készítsünk?

Használjunk vagy készítsünk?

- Ma már számos különböző API áll rendelkezésre a fejlesztés segítésére
  - Vannak általános keretet adók (SDL, SFML, stb)
  - Vannak specifikusak: csak bizonyos funkcióra koncentrálnak
    - képformátumok betöltése. Pl. DevIL
    - hangok kezelése. Pl. OpenAL, Bass
    - betűk kezelése. Pl. FreeType
    - mesterséges intelligencia, stb
- **Kezdők kedvencei**
  - Az OpenGL szűk funkcionalitása miatt a kezdők előszeretettel alkalmazzák az API-kat
  - ezekből építkezve gyorsabban megtudják valósítani a kitűzött céljukat

Használjunk vagy készítsünk?

- A profi játékipart megfigyelve a fejlesztők másképpen gondolkodnak
  - **OKA:** nagyon megnőhet a játékunk más API-któl való függősége
- Példa: szeretnénk készíteni egy platformfüggetlen két dimenziós fizikai szimuláció alapú játékot
  - Grafikus API: OpenGL
  - Matematikai függvénykönyvtár: GLM
  - Keretrendszer: SDL
  - Textúrák betöltése: DevIL library
  - Audió támogatása: Bass library
  - Fizika támogatása: Box2D
  - Fontok kezelése: FreeType2
  - Szükség esetén GUI library: CEGUI

8 darab library

Használjunk vagy készítsünk?

- Belátható, hogy aki egy nagyobb projektben gondolkozik, csökkentenie kell a függőségeket
- Az API-k ha lassan is, de folyamatosan változnak
- Egy-egy változás hibákat, bizonyos részek újraírását idézheti elő.
- Mindezek mellett az egyes felhasznált lib-ek licencei bizonyos esetekben korlátozhatják a fejlesztőket.
  - Például ha kommerciális termékké érik a projekt
- A függőségek kikerülése azonban jelentős többletmunkát igényel
  - A gyakorlatban még a legnagyobb cégek is meghagynak bizonyos függőségeket

GAME OVER