Cg alapismeretek

- Mi a Cg?
  - C for graphics
- Programozható grafikus hardvert használva
  - Alakzat, megjelenés, mozgásának vezérlése
  - Nagy sebességgel
- Programozási platform
  - Könnyű használni
  - Gyors speciális effekt előállítás
  - Valós idejű mozi minőségű élmény biztosítása

- Nem szükséges a grafikus hardver assembly szintű programozása
  - OpenGl, DirectX, Windows, Linux, Macintosh, Xbox
- CG fejlesztése
  - Microsofttal közösen
  - Kompatibilis
    - OpenGL API
    - High-Level Shading Language (HLSL) DirectX

- A Cg különbözik a C, C++ és Java nyelvektől
    - Nagyon speciális
- Shading/árnyékoló nyelv
    - Fizikai szimuláció és más nem árnyékoló feladatok
    - Cg program
        - Részletes recept egy objektum renderelésére a grafikus hardvert használva

Cg adatfolyam modell

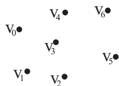
- Grafikára specializálódott
  - Különbözik a többi konvencionális nyelvtől
    - Feldolgozási lépések sorozata az adatokon
  - Vertex-eken és fragmenseken hajt végre műveleteket
  - Minden időpillanatban, amikor egy vertex feldolgozódik vagy egy fragmens jön létre a raszterizálás alatt
    - Vertex/fragmens bemenet
    - Kimenet vertex/fragmens

- CPU
  - Általános célú
  - Alkalmazások végrehajtása
    - C++, JAVA
- GPU
  - Grafikus alkalmazásokra
    - 3D-s színtér
  - Nem képes általános célú programot végrehajtani
    - Speciális
    - Nagy teljesítmény
  - Cg: absztrakt végrehajtási modell

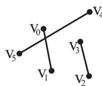
- CPU
  - Helyes program
  - Le lehet fordítani
  - Végre lehet hajtani
    - Operációs rendszer
- GPU
  - Hardware profile-ok
    - Nem minden Cg programot lehet lefordítani egy adott GPU-n
    - Mindegyik profile egy bizonyos GPU architektúrához és grafikus API-hoz tartozik
    - Nem csak helyesnek kell lenni egy programnak
  - Nem képes általános célú programot végrehajtani
  - Korlátozni kell a profile-nak megfelelően

- GPU-k fejlődnek
- Új profile-okat támogat majd a Cg
    - Az új képességekkel rendelkező GPU-okhoz kapcsolódnak
- Idővel a profile-ok nem lesznek olyan fontosak
- A mai Cg programok a jövőbeli profile-okkal gond nélküli fordíthatóak lesznek
    - Superset
- Minél kisebb és hatékonyabb a Cg program, annál gyorsabban fog futni
- A profile-ok nem a Cg korlátozása, hanem a GPU-ké

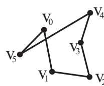
- A csővezeték egy állapotok szekvenciája
    - Párhuzamosan
    - Adott sorrendben
- Mindegyik állapot az előzőből kapja a bemenetét
- A kimenetét pedig a következő állapotba küldi
- Vertexek, geometriai primitívek és fragmensek (lehetséges pixel) sokaságát dolgozza fel



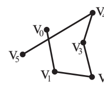
Pontok



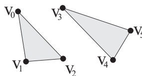
Vonalak



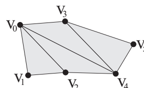
Vonal hurok



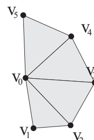
Töredezett vonal



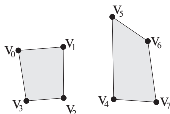
Háromszögek



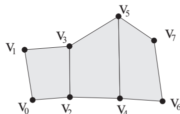
Háromszögsáv



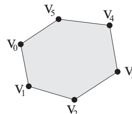
Háromszög-legyező



Négyszögek



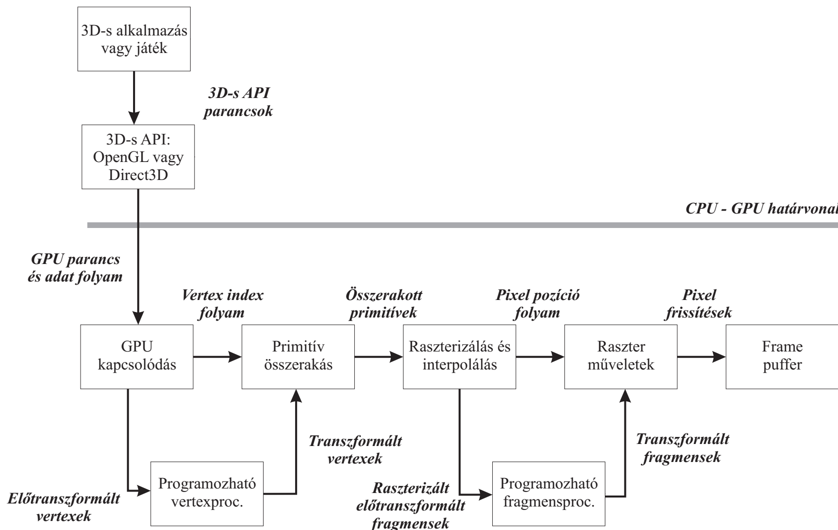
Négyszögsáv



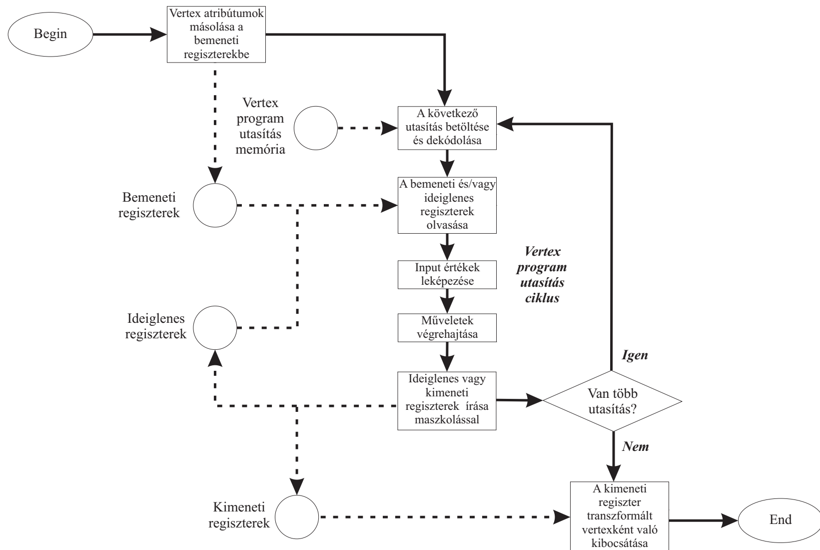
Poligon

- Mindegyik vertex rendelkezik
  - Pozícióval
  - Más attribútumokkal
    - Szín
    - Másodlagos (vagy spekuláris) szín
    - Egy vagy több textúra koordináta halmaz
    - Normál vektorok
    - ...

- Pixel
  - Picture element
  - Frame puffer tartalma egy adott helyen
    - Hasonlóan a szín, mélység és egyéb értékek, melyek ugyanazon a pozícióban találhatóak
- Fragmens
  - Az az állapot, mikor potenciálisan egy bizonyos pixel frissítése szükséges
  - A raszterizálás pixel méretű fragmensekre bontja a geometriai primitíveket
  - Ugyanúgy van pozíciója, mélység értéke, másodlagos színe és egy vagy több textúra koordináta halmaza
  - Potenciális pixel


3D-s alkalmazás vagy játék
3D-s API parancsok
3D-s API: OpenGL vagy Direct3D
CPU - GPU határvonal
GPU parancs és adat folyam
Vertex index folyam
Összerakott primitívek
Pixel pozíció folyam
Pixel frissítések
GPU kapcsolódás
Primitív összerakás
Raszterizálás és interpolálás
Raszter műveletek
Frame puffer
Transzformált vertexek
Transzformált fragmensek
Programozható vertexproc.
Programozható fragmensproc.
Előtranszformált vertexek
Raszterizált előtranszformált fragmensek

Programozható vertex processzor


Begin
Vertex atribútumok másolása a bemeneti regiszterekbe
Vertex program utasítás memória
A következő utasítás betöltése és dekódolása
Bemeneti regiszterek
A bemeneti és/vagy ideiglenes regiszterek olvasása
Input értékek leképezése
Vertex program utasítás ciklus
Ideiglenes regiszterek
Műveletek végrehajtása
Igen
Ideiglenes vagy kimeneti regiszterek írása maszkolással
Van több utasítás?
Nem
Kimeneti regiszterek
A kimeneti regiszter transzformált vertexként való kibocsátása
End

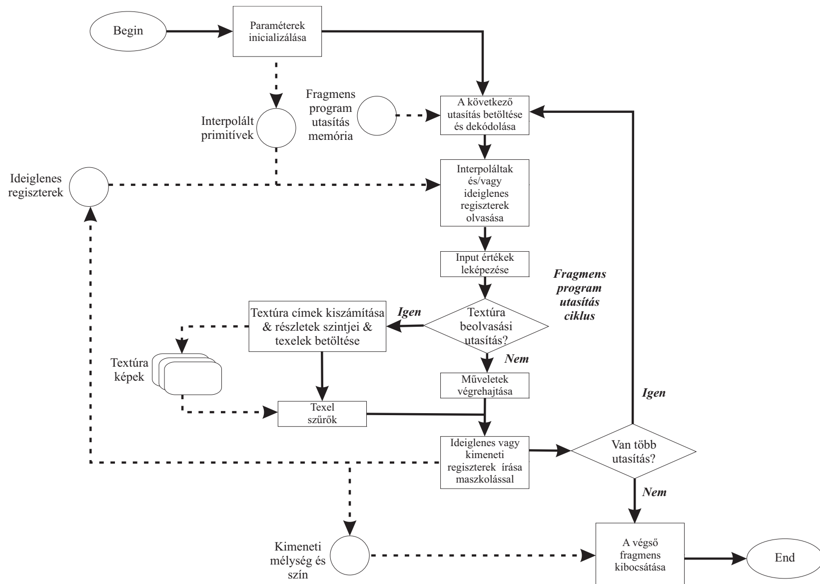
- Vertexek attribútumainak beolvasása a vertex processzorba
  - Pozíció, szín, textúra koordináták, stb.
- A vertex processzor újra meg újra behozza az következő utasítást és addig, amíg a vertex program véget nem ér
- Az utasítások számos különböző regiszter bankok halmazát éri el, melyek vektor értékeket tartalmaznak
  - Pozíció, normál vagy szín

- A vertex attribútum regiszterek csak olvashatóak
    - Az alkalmazás által meghatározott vertex attribútumok halmazát tartalmazza
- Az ideiglenes regiszterek írhatóak és olvashatóak is
    - Köztes értékek kiszámítására használhatóak
- A kimeneti eredmény regiszterek csak írhatóak
- Amikor a vertex program befejeződik, akkor a kimeneti regiszter tartalmazza a transzformált vertexet
- Az eredmény a raszterizálás és interpolálás után a fragmens processzorhoz kerül

- A legtöbb vertex feldolgozás korlátozott számú műveletet használ
- Vektor műveletek
  - 2, 3, 4 komponensű lebegőpontos vektorok
  - Összeadás, szorzás, szorzás-összeadás, skalár szorzat, minimum, maximum
  - Hardver támogatás
    - Vektor negálás, komponensenkénti „keverés" (swizzle)
    - Vektor műveletek általánosítása (Negálás, kivonás, kereszt-szorzat)

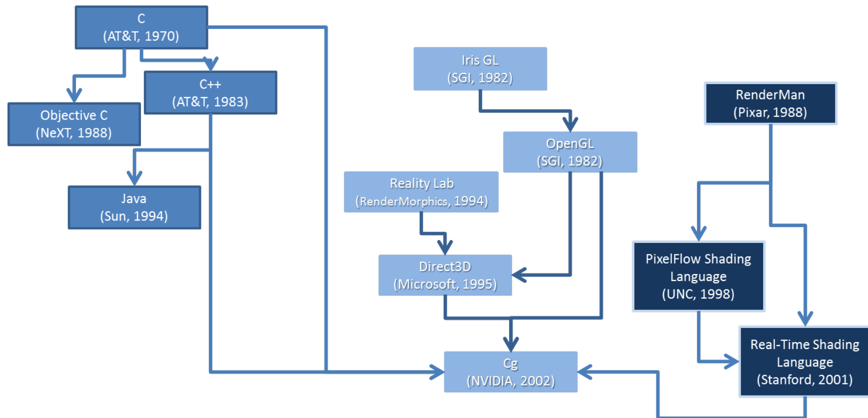
- Komponensenkénti írási maszkolás
  - Műveletek kimenetének szabályozás
- Reciprok és reciprok négyzetgyök kombinálása vektor szorzással és skalár szorzattal
  - Vektor normalizálás
  - Vektor skalárral való osztása
- Exponenciális, logaritmikus és trigonometrikus közelítések
  - Megvilágítás, köd és geometriai számítások elősegítése
- Specializált utasítások
  - Megvilágítás, elnyelődési fv-ek könnyebb kiszámítása

Programozható fragmens processzor


Begin
Paraméterek inicializálása
Interpolált primitívek
Fragmens program utasítás memória
A következő utasítás betöltése és dekódolása
Ideiglenes regiszterek
Interpoláltak és/vagy ideiglenes regiszterek olvasása
Input értékek leképezése
Fragmens program utasítás ciklus
Textúra címek kiszámítása & részletek szintjei & texelek betöltése
Igen
Textúra beolvasási utasítás?
Nem
Textúra képek
Műveletek végrehajtása
Texel szűrők
Igen
Ideiglenes vagy kimeneti regiszterek írása maszkolással
Van több utasítás?
Nem
Kimeneti mélység és szín
A végső fragmens kibocsátása
End

- Hasonló műveletek a programozható vertex processzorhoz
- Textúrázó műveletek
  - Textúra képhez való hozzáférés
    - Textúra koordináták
    - Visszatér a textúra egy szűrt mintájával
- Az új GPU-k támogatják lebegőpontos értékeket használatát
  - A fragmens műveletek hatékonyabbak, ha alacsonyabb pontosságú adat típusoknál

- Sok fragmens feldolgozása egyszerre
  - Nem lehetséges a tetszőleges szétosztás (branching)
- Cg-vel lehetséges ilyen fragmens programokat írni
  - Szimulál
  - Szétoszt és iterál
  - Feltételes „beosztás" operátorok
  - Ciklus „letekerés" (unrolling)
- Bemenő regiszterek
  - Interpolált fragmensenkénti paraméterek
    - Fragmens primitívek vertexenkénti paramétereiből származtatva
- Írható/olvasható ideiglenes regiszterek
- Csak írható kimeneti regiszterek
  - Szín
  - Opcionálisan új mélység érték



C
(AT&T, 1970)
C++
(AT&T, 1983)
Objective C
(NeXT, 1988)
Java
(Sun, 1994)
Iris GL
(SGI, 1982)
OpenGL
(SGI, 1982)
Reality Lab
(RenderMorphics, 1994)
Direct3D
(Microsoft, 1995)
RenderMan
(Pixar, 1988)
PixelFlow Shading
Language
(UNC, 1998)
Real-Time Shading
Language
(Stanford, 2001)
Cg
(NVIDIA, 2002)

- Általános célú programozási nyelvek
  - GPU-kra specializálódva
- Nem valós idejű árnyaló nyelvek
  - Valós időre optimalizálva
- Programozhat GPU-k és 3D API-k
  - Magas szintű nyelvi támogatás

- Örökség
  - Általános célú C programozási nyelv
  - Offline árnyékoló nyelvek
    - Pl. Renderman árnyaló nyelv
  - Grafikus funkcionalitás
    - OpenGL
    - Direct3D

Cg környezet

- A Cg csak egy komponense a szoftver és hardver infrastruktúrának
    - Összetett 3D-s látvány előállítása programozható GPU-okon valós időben

- Régebben egy PC-n a CPU kezelte le az összes vertex transzformációkat és „pixel-pushing” feladatokat
- A grafikus hardver csak a pixel puffert biztosította
  - A hardver a képernyőn jelenítette meg
- Saját 3D-s megjelenítő algoritmusok
- 3D-s alkalmazások
  - OpenGL
  - Direct3D

OpenGL

- SGI 1991
- OpenGL architecture Review Board (ARB)
- Eredetileg csak erős UNIX munkaállomásokon fut
- Microsoft alapító ARB tag
  - Windows NT
- Multi platformos programozási felület

Direct3D

- Microsoft 1995
  - Direct3D - DirectX
- Windows-s PC-k
- Xbox konzolok
- DirectX 10
  - Windows Vista
  - Aero felület

- Windows-os PC-ken
  - OpenGl vs. Direct3D
    - Melyik a jobb?
    - Melyik lesz az egyeduralkodó?
- A verseny folytatódik
  - Mind a két programozási felület előnyére válik
    - Javul a teljesítményük
    - Javul a minőségük
    - Javuk a funkcionalitásuk

- GPU programozhatóság
- Cg szempontjából összehasonlítható képességekkel bírnak
- Ugyanazon a GPU-n fut mind a kettő
    - Meghatározza a képességeket
- Csekély előny OpenGl esetén
    - A hardware gyártók jobban tudják megmutatni a teljes jellegzetességük halmazát OpenGl-en keresztül
    - A gyártó specifikus kiterjesztések egy kicsit bonyolultabbak a fejlesztők számára
- Mind a két programozási felület támogatja a Cg-t

A Cg fordító és Runtime

- Egyetlen egy GPU sem képes a Cg programot szöveges formában futtatni
  - A fordítás során a Cg programot olyan formátumba kell fordítani, melyet a GPU végre tud hajtani

- Először a Cg program olyan formába kerül, melyet a 3D-s programozási felület elfogad
  - OpenGl
  - Direct3D
- Az alkalmazás továbbítja a Cg program fordítását a GPU-nak
  - Megfelelő OpenGl vagy Direct3D utasításokkal
- Az OpenGl vagy Direct3D meghajtó hajtja végre az utolsó fordítást
  - Hardveren végrehajtható forma

- A fordítás részletei a GPU kombinált képességei és a 3D programozási felülettől függnek
- A köztes OpenGl vagy Direct3D formátum a GPU generációjától függ
    - Előfordulhat, hogy a GPU nem támogat egy érvényes/helyes Cg programot a GPU korlátai miatt

Hagyományos

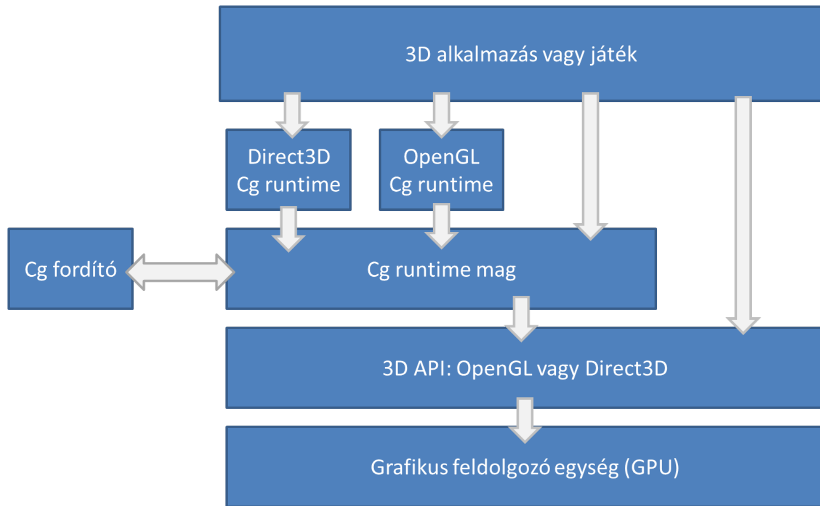
- A fordítás offline eljárás
    - A fordító a programot a CPU futható formátumúvá alakítja
    - A fordítás után nincs szükség újra fordításra
        - Programkód megváltozik
        - Másik platformon akarjuk futtatni

Cg

- Dinamikus fordítás
  - Támogatja a statikust is
- A fordító nem egy különálló program
  - Cg runtime könyvtár része
- Az alkalmazásokat össze kell szerkeszteni a Cg futásidejű könyvtárral
- Az alkalmazás használja a Cg-t, majd meghívja a Cg futásidejű rutinokat cg prefix-szel
- Optimalizált Cg program bizonyos GPU-ra

- OpenGl → CgGl → `CcgGL`
- Direct3D → CgD3D → `CcgD3D`
- Egyszerre a kettőt nem lehet használni!
- Összehasonlítva a Cg futásidejű könyvtár magjával, amely tartalmazza a Cg fordítót
    - CgGl és CgD3D könyvtárak viszonylag kicsik
    - Csak a megfelelő OpenGl illetve Direct3D hívásokat tartalmazza, melyek Cg programok végrehajtásának a beállítására szolgálnak
    - Hasonló rutinok

Cg alapismeretek
Hogyan illeszkedik a Cg futásidejű könyvtár az alkalmazásokhoz?



```
struct C2E1v_Output {
  float4 position : POSITION;
  float4 color    : COLOR;
};

C2E1v_Output C2E1v_green(float2 position : POSITION)
{
  C2E1v_Output OUT;

  OUT.position = float4(position, 0, 1);
  OUT.color    = float4(0, 1, 0, 1);  // RGBA green

  return OUT;
}
```

```
struct C2E1v_Output {
  float4 position :
      POSITION;
  float4 color    :
      COLOR;
};
```

- Kimeneti értékek
- Korlátozott kimentek
- Hasonló a C/C++-hoz
- Szemantika
  - POSITION
  - COLOR

- Betűvel kezdődő
  - Betűvel vagy számmal folytatódik
  - Tartalmazhat _ (aláhúzás) karaktert
  - Nem lehet kulcsszó
- Más azonosítók
  - Függvény név
  - Függvény paraméter név
  - Helyi változó
  - stb.

- C, C++-ban nincs natív vektor típus
  - Skalár értékek tömbje
- A vertex és fragmens feldolgozásban alapvető adat típus a vektorok
  - GPU beépített vektor támogatás
  - Cg-ben vektor adat típus
- float4
  - Nem foglalt szó
  - Alap típus definíció Cg Standard Könyvtárban

- Előredefiniált vektor adattípus
    - `float2`, `float3`, `float4`
    - Biztosítják a hatékony vektor feldolgozást a GPU-okon
- `float x[4]` $\neq$ `float4 x`
- Pakolt tömb
    - `data[3]` hatékony
    - `data[i]` nem hatékony

- Natív támogatás
  - `float4x4`
  - `half3x2`
  - `fixed2x4`
- Hasonlóan inicializálhatók, mint C-ben
- Szintén hatékonyak

- Ragasztó, mely hozzáköt egy Cg programot a hátralévő grafikus csővezetékhez
- Megmutatja azt a hardver erőforrást, amelyik feltölti a kimeneti struktúrát a Cg program visszatérésekor
- `POSITION`
  - Transzformált vertex vágási területen lévő pozíciója
- `COLOR`
  - Diffúz vertex szín
- Jelzi, hogy a megelőző változók hogyan kacsolódnak a grafikus csővezeték maradék részéhez
- Nem mindegyik szemantika érhető el az összes profile-ban
- Lehet saját szemantika neveket is létrehozni

- Függvények deklarációja hasonlóan történik, mint C-ben
  - Visszatérési érték
  - Név
  - Vesszővel elválasztott paraméter lista zárójelek között
  - Függvénytörzs
- Belépő vagy belső függvények

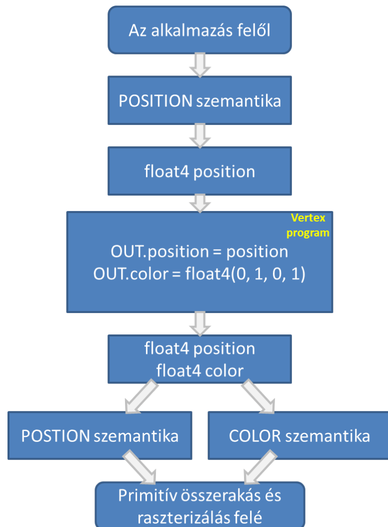
Belépő függvények

- Vertex vagy fragmens program definiálása
  - Analóg a `main` függvényre
  - A program végrehajtása ebben a belépő függvényben kezdődik
  - Amikor egy bemenő paraméter nevét kettőspont és szemantika név követ, akkor ez azt jelzi, hogy szemantika az bemenő paraméterhez van kötve
    - A vertex processzor inicializálja ezt a paramétert az alkalmazás által meghatározott összes vertex pozíciójával

Belső függvények

- A belépő függvények és belső függvények által meghívható függvények
  - Cg Standard Könyvtár
  - Saját belső függvény

- Ugyanaz a nevük
    - Mégis különböznek
        - Alkalmazás által meghatározott vertex pozíciók
        - Vágási területen lévő vertex pozíció
        - Hardver raszterizáló
- A grafikus csővezeték különböző helyén lévő pozíciók
    - Az első programban ez változás nélkül van tovább küldve
    - Nem változik



- A lényeg található a függvény törzsében
- Deklarálni kell a visszatérési értéket tartalmazó változót
- Kimeneti struktúra

```
{
  C2E1v_Output OUT;

  OUT.position = float4(position, 0, 1);
  OUT.color    = float4(0, 1, 0, 1);      // RGBA zöld

  return OUT;
}
```

- `OUT.position = float4(position, 0, 1);`
  - A két komponensű vektor a kimeneti pozíció vektornak megfelelő struktúrává konvertálódik
- `OUT.color = float4(0, 1, 1, 1);`
  - Constructor vektorok és mátrix számára
- `return OUT;`

- Cg futásidejű könyvtár
  - Betöltés és fordítás
  - Meg kell adni
    - A belépő függvény nevét
    - A profile nevét a belépő függvény lefordításához

Profile neve	Programozási felület	Leírás
arbvp1	OpenGL	Alap multivendor-os vertex programozhatóság
vs_1_1	DirectX 8	Alap multivendor-os vertex programozhatóság
vp20	OpenGL	Alap NVIDIA vertex programozhatóság
vs_2_0 vs_2_x	DirectX 9	Fejlett multivendor-os vertex programozhatóság
vp30	OpenGL	Fejlett NVIDIA vertex programozhatóság

Hagyományos hibák

- Szintaktikus
- Szemantikus
- Hagyományos fordítói hibák

Profile függő hibák

- Szintaktikailag és szemantikailag hibátlan
- Nem támogatja a megadott profile

Adottság

- Fragmens program
  - Textúra elérés
- Vertex program nem ér el ilyen adatot
- Nem megengedett olyan program lefordítása, amelyet nem lehet végrehajtani

Környezet

- Hibás olyan vertex programot írni, amely nem tér vissza pontosan egy `POSITION` szemantikával rendelkező értékkel
  - A vertex pozícióval rendelkeznek a hátralévő grafikus csővezeték állapotában
  - Fordítva: a fragmens program nem térhet vissza `POSITION` szemantikával rendelkező értékkel

Kapacitás

- A GPU képességeinek a korlátaiból származik
  - 4 textúra elérése egy renderelési menetben

Nem nyilvánvaló az, hogy mi haladja meg a GPU képességét

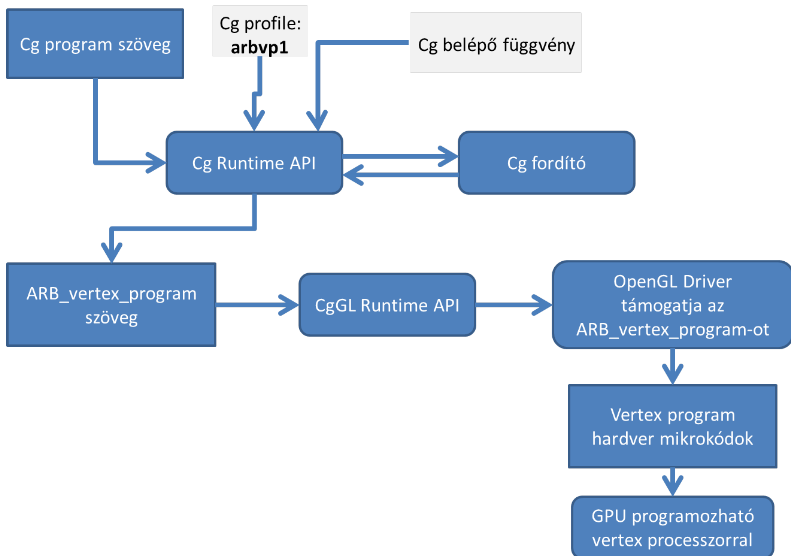
Hibák megelőzése

- Megfelelően nagy képességű profile választása
- Ismerni kell a határokat

- Cg programok gyűjteménye
  - Egy vertex és egy fragmens program
  - Cg programok fordítása futási időben
  - Új Cg programok generálása az alkalmazás futása közben

Cg alapismeretek

Vertex és fragmens programok letöltése és konfigurálása



```
struct C2E2f_Output {
  float4 color : COLOR;
};

C2E2f_Output C2E2f_passthru(float4 color : COLOR)
{
  C2E2f_Output OUT;
  OUT.color = color;
  return OUT;
}
```

- Ez a program nem csinál semmit
- Az output ugyanaz az értek, amelyet a raszterizáló állított elő
- A GPU raszter műveletekért felelős hardvere ezt a színt használja a frame puffer frissítésére, amennyiben a fragmens túlélte a különböző raszter műveleteket

```
struct C2E2f_Output {
  float4 color : COLOR;
};
```

- A fragmens program csak szín értéket frissít a frame pufferben
  - Néhány fejlettebb profile-ban további adat is módosítható pl. a mélység érték
- A `COLOR` szemantika azt jelenti, hogy a color tag egy szín, amely értékét használja majd a frame puffer frissítésénél

- Belépő függvény deklaráció
  - `C2E2f_Output C2E2f_passthru(float4 color : COLOR)`
- Visszatérő struktúra
  - `C2E2f_Output`
  - Négy komponensű vektor
- A függvény törzse

```
{
  C2E2f_Output OUT;
  OUT.color = color;
  return OUT;
}
```

Profile neve	Programozási felület	Leírás
ps_1_1 ps_1_2 ps_1_3	DirectX 8	Alap multivendor-os fragmens programozhatóság
fp20	OpenGL	Alap NVIDIA fragmens programozhatóság NV_texture_shader-nek és NV_register_combiners-nek megfelelően
arbfp1	OpenGL	Fejlett multivendor-os fragmens programozhatóság
ps_2_0 ps_2_x	DirectX 9	Fejlett multivendor-os fragmens programozhatóság
fp30	OpenGL	Fejlett NVIDIA fragmens programozhatóság NV_fragment_program-nak megfelelően

- Olyan egyszerű, hogy bármelyikkel fragmens profile-lal lefordítható lenne
- Cg fordító használata
    - cgc
    - Tesztelés
    - IDE
        - MS Visual C++

OpenGL

```
glBegin(GL_TRIANGLES);
    glVertex2f(-0.8,  0.8);
    glVertex2f( 0.8,  0.8);
    glVertex2f( 0.0, -0.8);
glEnd();
```

Direct3D

```
static const MY_V3F triangleVertices[] = {
    { -0.8f,  0.8f, 0.0f },
    {  0.8f,  0.8f, 0.0f },
    {  0.0f, -0.8f, 0.0f }
pDev->DrawPrimitive (D3DPT_TRIANGLELIST, 0, 1);
```

```
myCgContext = cgCreateContext();
myCgVertexProgram =
cgCreateProgramFromFile(
  myCgContext,          /*Cg runtime környezet*/
  CG_SOURCE,            /*A program olvasható formában van*/
  myVertexProgFName,    /*A programot tartalmazó file neve*/
  myCgVertexProfile,    /*Profile: OpenGL ARB vertex program*/
  myVertexProgName,     /*Belépő függvény neve*/
  NULL);                /*Nincs extra fordító opció*/

cgGLLoadProgram(myCgVertexProgram);

cgGLEnableProfile(myCgVertexProfile);
cgGLBindProgram(myCgVertexProgram);

cgGLDisableProfile(myCgVertexProfile);

cgDestroyProgram(myCgVertexProgram);
cgDestroyContext(myCgContext);
```

```
static void checkForCgError(const char *situation)
{
  CGerror error;
  const char *string = cgGetLastErrorString(&error);

  if (error != CG_NO_ERROR) {
    printf("%s: %s: %s\n",
    myProgramName, situation, string);
    if (error == CG_COMPILER_ERROR) {
      printf("%s\n", cgGetLastListing(myCgContext));
    }
    exit(1);
  }
}
```

- Uniform paraméterek
  - Az előző példák kiterjesztése
  - `OUT.color = float4(1.0, 0.41, 0.7, 1.0);`
  - Nem lehet minden színre külön Cg programot írni
  - A program általánosítása paraméter átadásával

```
struct C3E1v_Output {
  float4 position : POSITION;
  float4 color    : COLOR;
};

C3E1v_Output C3E1v_anyColor(float2 position : POSITION,
                            uniform float4 constantColor)
{
  C3E1v_Output OUT;

  OUT.position = float4(position, 0, 1);
  OUT.color = constantColor;  // Some RGBA color

  return OUT;
}
```

- Jelzi a változók kezdő értékének a forrását
  - Amikor `uniform` változóként van deklarálva egy változó, akkor külső környezetből kapja az iniciális értékét
- Olyan vertex program generálódik, amely a GPU konstans regiszteréből kapja az iniciális értékét
- Cg runtime használatakor
  - A 3D-s alkalmazás le tudja kérni egy paraméter kezelőt a Cg programon belül

Kezelő lekérdezése

```
Cgparemeter myParameter = cgGetNamedParameter(program,
    ``myParameter``);
```

Paraméter értékének beállítása

```
cgGLSetParameter4fv(myParameter, value);
```

Paraméter definiálása

```
float4 myParameter
```

Explicit iniciális érték megadás

```
float4 green = float4(0, 1, 0, 1);
```

Szemantika használata

```
float4 position : POSITION;
```

Profile függő

```
float whatever; //nem definiált vagy 0
```

- Hasonló hatása van, mint C vagy C++-ban
    - Korlátozza a változó használatát a programban
    - A bizonyos érték nem változhat meg soha
    - Hiba üzenet generálódik ellenkező esetben

- A bemeneti adatokon elvégzett számítások
  - Operátorok
  - Beépített függvények a Cg standard könyvtárban
- A Cg támogatja ugyanazokat az aritmetikai, relációs és más operátorokat, amelyeket a C és C++-ban használhatunk
- A Cg mégis különbözik a C és C++-tól
  - Beépített támogatás a vektor mennyiségeken végzendő aritmetikai műveletek esetén
  - C++-ban operátor overloading (túlterhelés) megoldható
  - Amikor skalár az egyik operandus, akkor az adott skalárt egy vektorrá konvertálja

- Folytonos adattípusok ábrázolása
  - `float`, `half`, `double`
  - Csak a Cg-ben `half`: 16 bites fél pontosságú lebegőpontos érték
- A GPU általában nem rendelkezik hardveres támogatással annyi alap adat típusra, mint a CPU
  - Pl. nem támogatja a pointer adat típust
  - Nem támogatják a természetüktől fogva diszkrét mennyiségeket sem
    - Alfa-numerikus karakterek
    - Bit maszkok

- A folytonos mennyiségek nincsenek korlátozva az egész értékekre
  - Fragmens szinten egy szűk intervallumra vannak korlátozva
    - $[0, 1]$ vagy $[-1, +1]$ (szín, normál vektorok)
    - Ezen intervallum korlátozott adattípusok fix-pontos adattípus néven is ismertek
- A `float` nem mindig lebegő-pontos értéket jelent az összes profile-ban, az összes összefüggésben

- Trigonometrikus
- Exponenciális
- Vektor mátrixok
- Textúrák
- Viszont nincs
  - I/O
  - String műveletek
  - Memória foglalás

- A Cg standard könyvtár túlterheli a függvényeket
  - A rutinok sok adattípuson értelmezettek
  - Többszörös megvalósítás
  - Ugyanazon néven több fajta paraméter listával
    - Mindig a megfelelő verziójú függvény hívódik meg
  - Saját belső túlterhelt függvények
- Különböző megvalósítása ugyanannak a rutinnak más profile-ok számára
  - Pl. egy fejlett vertex profile-ban van sin és *cos*
    - De egy alap vertex profile-ban valamilyen módon közelíteni kell az adott értékeket
    - Profile függő függvény túlterhelés
    - Két függvény, mely speciális profile-t követel meg

- A Cg Standard könyvtár matematikai és más műveltеket hatékonyabbak és pontosabbak
    - Speciális GPU utasítások

Saját függvény

```
float myDot(float3 a, float3 b)
{
   return a[0]*b[0] + a[1]*b[1] + a[2]*b[2]
}
```

Beépített függvény (Gyorsabb)

```
dot(a, b)
```

- `out` minősítő jelzi, amivel a rutinnak vissza kell térnie
  - Kezdetben az értéke nem definiált
  - call-by result (copy out)
- `in` minősítő érték szerinti paraméter átadás
  - Az értéket eldobja a Cg
  - Ha `out` minősített is akkor nem
- `inout`
- Az `in` minősítő alapértelmezett
- Nincs funkcionális különbség a két módszer között
- Kombinálhatóak

- Vektor komponensek átrendezése
- Suffix-ek
  - x, y, z, w
  - r, g, b, a
- Diszjunkt halmazok
- Fordított swizzling
  - vec1.xw = vec2

Példák

```
float4 vec1 = float4(4.0, -2.0, 5.0 3.0);
float2 vec2 = vec1.yx;
float scalar = vec1.w
float3 vec3 = scalar.xxx; //smearing
```

- `._m<row><col>`

Példák

```
float4x4 myMatrix;
float myFloatScalar;
float4 myFloatVec4;

myFloatScalar = myMatrix._m32;
myFloatVec4   = myMatrix._m00_m11_m22_m33;
```

- Struktúrák
  - Előző példák
- Tömbök
  - Nincs pointer típus
  - Tömb szintaxist kell használni
- Folyam szabályozás
  - Függvények és a `return` utasítás
  - `if-else`
  - `for`
  - `while` és `do-while`
- Profile specifikusak
  - Pl. a ciklusoknál az iterációk számát előre meg kell tudni határoznia
- `goto` és `switch`
  - Foglaltak
  - Nem támogatják egyelőre

Összefoglalás

- Programozható vertex és fragmens processzor
  - A csővezeték standard folyamatának a megváltoztatása
  - Profile függő vertex és fragmens programok
  - Szemantikák
  - Speciális műveletek vektorokon
  - Dinamikus fordítás program futása alatt
- Példák