OpenGL ES

Bevezetés


Tanács Attila

Képfeldolgozás és Számítógépes Grafika Tanszék
Szegedi Tudományegyetem

3D grafikai csomagok

- Direct3D, DirectX
  - Csak Microsoft platformon
- OpenGL
  - Silicon Graphics: IRIS GL (zárt kód)
  - → OpenGL (1992)
  - Nyílt szabvány; konzorciumi kezelésben (ma: Khronos Group)
  - C nyelvű függvénykönyvtár asztali PC-kre
  - *Alapfokon ismertnek tételezzük fel a továbbiakban!*
- Unity 3D
  - Multiplatform környezet
  - Magasabb szintű modellezés (OpenGL ES-re épül a kivitelezése)

OpenGL ES

- OpenGL ES (Embedded Systems)
  - Nyílt szabvány; 2000-es évek eleje
  - OpenGL redukált változata beágyazott rendszerekre
  - EGL: közös interfész réteg
  - Sok platform támogatja
    - Android, iOS, Java ME, Symbian, QNX, PlayStation, Blackberry, webOS, ...
    - WebGL (3D grafika böngészőkben) is erre épül!
  - Nincs GLUT és GLU segédkönyvtár!
    - Néhány hasznos függvény azért átkerült

- Specifikáció
  - The Standard for Embedded Accelerated 3D Graphics

OpenGL ES verziók

- **OpenGL ES 1.0**
  - OpenGL 1.3 verzió alapján (ezt tanítjuk BSc-n)
  - **Embedded profilok**
    - **Common lite profile**: csak fixpontos adattípus (nincs lebegőpontos)
      - Ha nincs lebegőpontos CPU támogatás (beágyazott környezetben előfordul)
    - **Common profile**: fixpontos és lebegőpontos adattípus is van
  - **Főbb különbségek az asztali változathoz képest**
    - Nincs közvetlen modellezési lehetőség
      - glBegin() … glEnd() helyett tömbökkel kell dolgozni
    - Fixpontos számábrázolás megjelenése
    - Nem érhetők el az alábbi funkciók
      - GL_QUAD és GL_POLYGON primitívek; csak háromszögek használhatók fel!
      - texgen; line stipple; poligon stipple; polygon_mode
      - Bitmap műveletek; előtér és összegző pufferek
      - Megjelenítési listák és feedback mód
      - Hátlapok anyagjellemzői; felhasználói vágó síkok; …

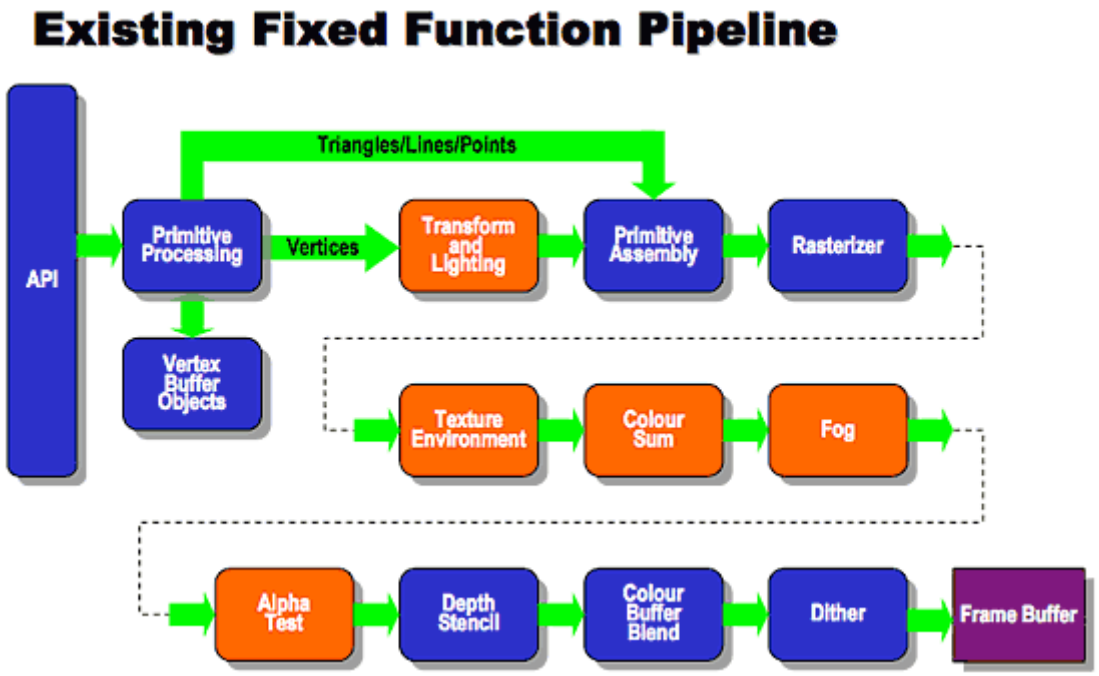
OpenGL ES verziók

- OpenGL ES 1.1
  - OpenGL 1.5 verzió alapján
  - Újdonságok
    - Multitexture támogatás; mipmap generálás
    - VBO (vertex buffer object)
    - Állapotlekérések; felhasználói vágósíkok
- OpenGL ES 2.0 (2007. március)
  - OpenGL 2.0 verzió alapján
  - Programozható műveletsor megjelenése
    - Vertex és fragment shader-ek
    - Nincs alapértelmezett transzformáció sorozat és megvilágítás típusok!
    - Nagyobb programozói szabadság – több munka!
      - Úgy oldjuk meg, ahogy akarjuk/tudjuk…
  - Nincs visszafelé kompatibilitás az 1.1 verzióval!

OpenGL ES verziók

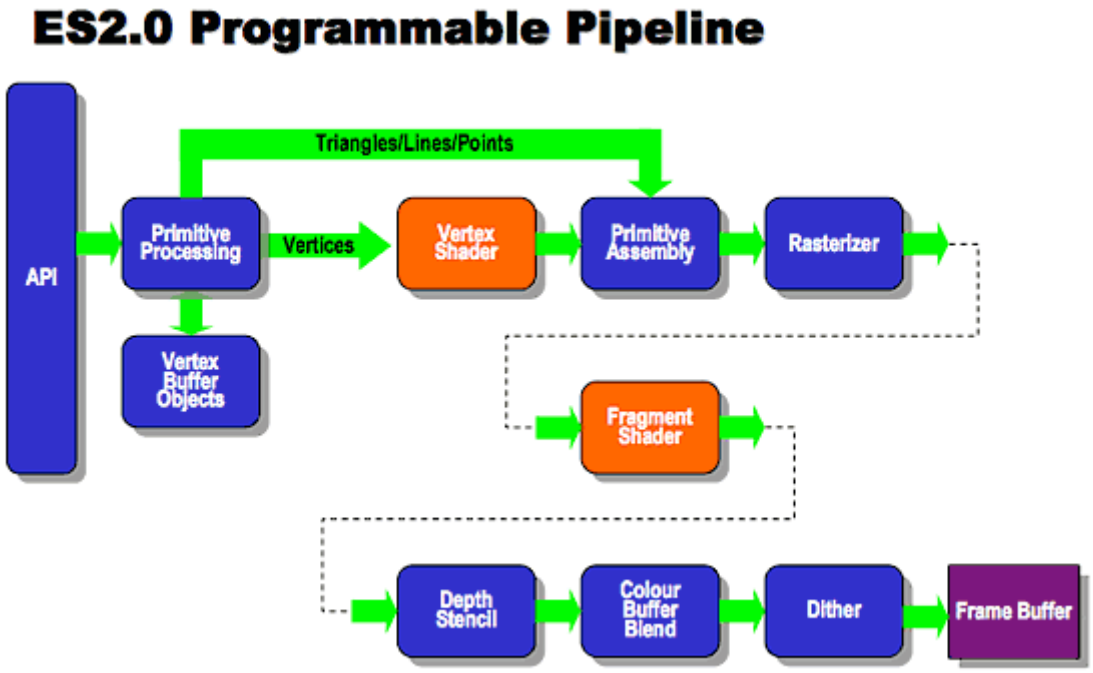
- OpenGL ES 3.0 (2012. augusztus)
  - Teljes kompatibilitás az OpenGL 4.3 verzióval
  - Visszamenőlegesen kompatibilis az OpenGL ES 2.0 verzióval
  - Újdonságok
    - Új textúra formátumok, tömörítés
    - Hatékonyabb megjelenítés
    - Új GLSL ES shader programozási nyelv
- OpenGL ES 3.1 (2014. március)
  - Indirekt rajzoló parancsok, shader frissítések
  - Visszamenőlegesen kompatibilis az OpenGL ES 3.0 és 2.0 verziókkal
- OpenGL ES 3.2 (2015. augusztus)
  - Geometria és tesszelláló shader-ek, lebegőpontos render, ...

Rögzített műveletsor



Kép forrása: Khronos Group

Programozható műveletsor



Kép forrása: Khronos Group

Shader-ek

- Vertex shader
  - Csúcspontokra vonatkozó transzformációk
    - Geometriai transzformációk alkalmazása
    - Megvilágítási egyenletek kiértékelése
    - ...
  - Speciális új adattípusok
    - attribute: pontonkénti adatok (pl. vertextömbök) tárolása
    - uniform: a vertex shader által használt konstans érték
    - sampler: textúrákat reprezentációhoz.
    - varying: shader interpolált kimenete fragmes shader felé
- Fragmens shader
  - Poligonok belső pontjaira vonatkozó transzformációk
    - Textúrázás
    - Fényhatások
    - ...
- A shaderek ESSL/GLSL nyelven íródnak
  - Embedded System / OpenGL Shading Language
    - Alacsony szintű, C-szerű szintaktika

Modellezés tömbökkel

- C programozási nyelven bemutatva
  - „Desktop” OpenGL szintaxis
- glBegin() … glEnd() hiányzik!
  - glDrawArrays() vagy glDrawElements() függvényekkel
  - glEnableClientState() és glDisableClientState() függvények
- Tömb típusok
    - glVertexPointer(): csúcspontok koordinátái
    - glNormalPointer(): csúcspontokhoz tartozó normálvektorok
    - glColorPointer(): csúcspontokhoz tartozó színek (RGB)
    - glIndexPointer(): csúcspontokhoz tartozó színek (színindex)
    - glTexCoordPointer(): csúcspontokhoz tartozó textúra koordináták
    - glEdgeFlagPointer(): élkirajzolás szabályozása
- Példa
  - http://www.songho.ca/opengl/gl_vertexarray.html

Függvények

- Csúcspontok koordinátái
  - glVertexPointer(GLint size, GLenum type, GLsizei stride, const GLvoid* pointer)
    - size: csúcspont koordináták száma
      - 2: 2D; 3: 3D
    - type: GL_FLOAT, GL_SHORT, GL_INT vagy GL_DOUBLE
    - stride: csúcspont adatok közötti távolság a tömbben (bájtban)
    - pointer: tömb kezdőcíme a memóriában
- Normálvektor adatok
  - glNormalPointer(GLenum type, GLsizei stride, const GLvoid* pointer)
    - type: GL_FLOAT, GL_SHORT, GL_INT vagy GL_DOUBLE
    - stride: normálvektor adatok közötti távolság a tömbben (bájtban)
    - pointer: tömb kezdőcíme a memóriában

Függvények

- Kirajzolás #1
  - glDrawArrays(GLenum mode, GLint first,GLsizei count);
    - mode: GL_TRIANGLE, …
    - first: tömb első feldolgozandó indexe
    - count: hány csúcspontot kell modellezni
- Hátrány
  - Az ismétlődő csúcsokat egyenként meg kell adni

Függvények

- Példa (kocka lapjainak modellezése)

```
GLfloat vertices[] = {...}; // 36 of vertex coords
...
// activate and specify pointer to vertex array
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);

// draw a cube
glDrawArrays(GL_TRIANGLES, 0, 36);

// deactivate vertex arrays after drawing
glDisableClientState(GL_VERTEX_ARRAY);
```

Függvények

- Kirajzolás #2
  - glDrawElements(Glenum mode, GLsizei count, Glenum type, const GLvoid * indices);
    - mode: GL_TRIANGLE, ...
    - count: csúcspontok száma
    - type: az csúcspontok indextömbjének típusa (pl. GL_UNSIGNED_BYTE)
    - indices: csúcspont indextömb memóriacíme

Függvények

- Példa (kocka lapjainak modellezése)

```
GLfloat vertices[] = {...};           // 8 of vertex coords
GLubyte indices[] = {0,1,2, 2,3,0,    // 36 of indices
                     0,3,4, 4,5,0,
                     0,5,6, 6,1,0,
                     1,6,7, 7,2,1,
                     7,4,3, 3,2,7,
                     4,7,6, 6,5,4};
...
// activate and specify pointer to vertex array
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, vertices);

// draw a cube
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_BYTE, indices);

// deactivate vertex arrays after drawing
glDisableClientState(GL_VERTEX_ARRAY);
```

Feladatok

- Modellezzünk kockát tömbökkel!
  - `glDrawElements()` függvényt használjuk
  - Háromszögekből építsük fel
    - A GPU háromszögekkel dolgozik, a sokszögek úgyis felbontásra kerülnek
    - OpenGL 3.1 felett csak háromszögekkel lehet dolgozni, megszűnt a `GL_QUADS`, `GL_QUAD_STRIP` és a `GL_POLYGON`
- Rendeljünk szín információt a csúcspontokhoz!
  - Külön szín tömb használatával
  - A felső (Y koordináta szerint) pontok vörösek, az alsók zöldek legyenek
- A csúcspont és szín információt közös tömbben adjuk meg!
  - Figyeljünk a `stride` és `pointer` paraméterek helyes megadására!
- Adjunk normálvektort a kocka csúcspontjaihoz!
  - Ezt már nem ússzuk meg 8 csúcsponttal, kell mind a 24...