Mobil 3D grafika

Android platformon


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; …

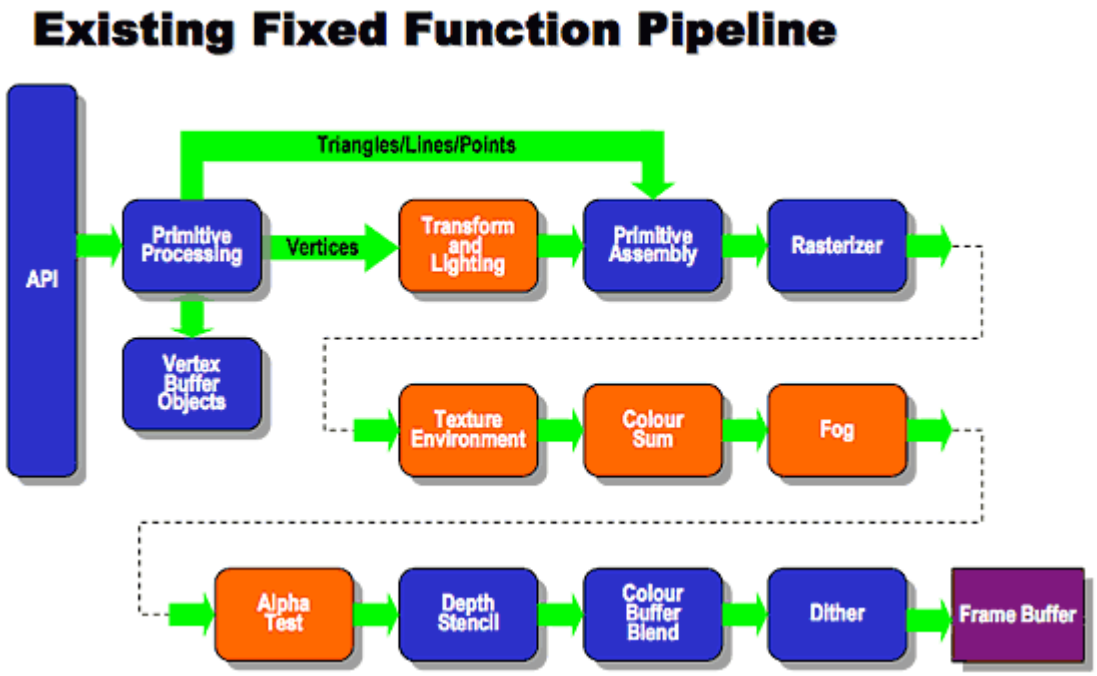
OpenGL ES verziók

- 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!
  - Nincs visszafelé kompatibilitás az 1.1 verzióval!

OpenGL ES verziók

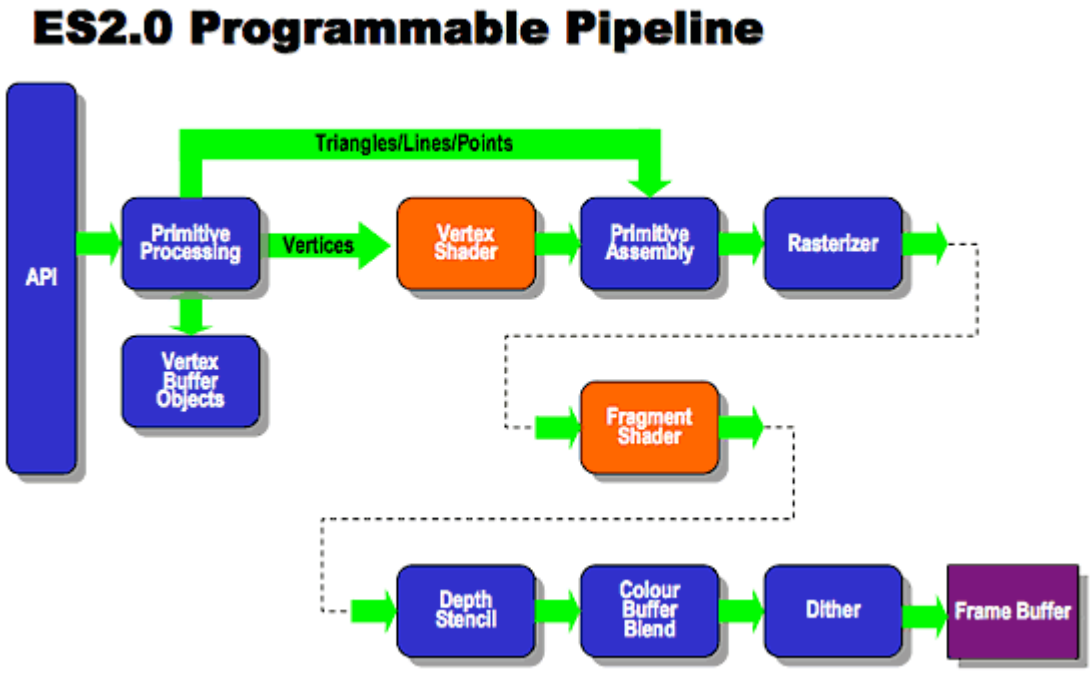
- OpenGL ES 3.0 (2012. augusztus)
  - Teljes kompatibilitás az OpenGL 4.3 verzióval
  - Visszamenőlegesen kompatibilis az OpenGL ES 2.0 verzióval
  - Újdonságok
    - Új textúra formátumok, tömörítés
    - Hatékonyabb megjelenítés
    - Új GLSL ES shader programozási nyelv
- OpenGL ES 3.1 (2014. március)
  - 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
    - 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
    - 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);
```

OpenGL ES Android-on

- OpenGL ES 1.0
  - android.opengl (API Level 1; Android 1.0)
  - Rögzített műveletsor
- OpenGL ES 1.1
  - android.opengl.GLES11 (API Level 4; Android 1.6)
  - Rögzített műveletsor
- OpenGL ES 2.0
  - android.opengl.GLES20 (API Level 8; Android 2.2)
  - Programozható műveletsor (shader-ek)
- OpenGL ES 3.0
  - (API Level 18; Android 4.3 – ha a gyártó támogatja)
- OpenGL ES 3.1
  - (API Level 21; Android 5.0 – ha a gyártó támogatja)
- Elérhető Java API-n vagy NDK-n keresztül (C nyelven)

Android OpenGL ES információk

- OpenGL ES (Android Developer)
- Grafika OpenGL ES 2.0-val (Android Developer)
  - OpenGL ES környezet beállítása
  - Alakzatok definiálása
  - Alakzatok rajzolása
  - Projekciós és kamera nézeti transzformációk megadása
  - Animáció
  - Reagálás érintés eseményekre
- NeHe Android portja
  - http://insanitydesign.com/wp/projects/nehe-android-ports/
  - http://code.google.com/p/nehe-android/
    - (OpenGL Demos és NeHe Lessons a Play Áruházban)
- Fountain OpenGL Application Walkthrough (CodeProject)
  - http://www.codeproject.com/Articles/269727/Fountain-OpenGL-Application-walkthrough

Android OpenGL ES

- Fő irányelvek
  - `<uses feature>` a `Manifest.xml` fájlban
  - Rajzolás `SurfaceView`-ra, összekapcsolás OpenGL ES kontextussal
  - Modellezés külön szálon fusson!

- Megvalósítás
  - Android segédosztályokkal (`GLSurfaceView` és társai)
    - Egyszerűbb, de kevesebb kontroll
  - Kontextus létrehozása manuálisan

GLSurfaceView + Renderer

- GLSurfaceView
  - `GLSurfaceView` osztály példányosítása (interakció nélküli esetben)
  - Leszármaztatás (ha szükséges interakció)
  - Az objektumpéldány `setRenderer()` függvényének hívása, amelynek átadjuk a saját `Renderer` modellező osztályunk objektumpéldányát
- **`GLSurfaceView.Renderer`** interfész implementálása
  - `onSurfaceCreated(GL10 unused, EGLConfig config)`
    - Kezdeti beállítások
  - `onSurfaceChanged(GL10 gl, int width, int height)`
    - Vetítő mátrix definiálása a beérkező méretek alapján
  - `onDrawFrame(GL10 gl)`
    - Modellezés helye
    - FloatBuffer, ByteBuffer, IntBuffer osztályok: Java tömbből OpenGL ES tömbökbe konverzió!
  - `onPause()`, `onResume()` függvények kezelése!
    - A `GLSurfaceView` objektumpéldány `onPause()`, `onResume()` függvényeit is meg kell hívni, mivel a modellezés külön szálon fut!

Pufferek létrehozása

```
public class Triangle {

    private FloatBuffer vertexBuffer;

    // number of coordinates per vertex in this array
    static final int COORDS_PER_VERTEX = 3;
    static float triangleCoords[] = {   // in counterclockwise order:
             0.0f,  0.622008459f, 0.0f, // top
            -0.5f, -0.311004243f, 0.0f, // bottom left
             0.5f, -0.311004243f, 0.0f  // bottom right
    };

    // Set color with red, green, blue and alpha (opacity) values
    float color[] = { 0.63671875f, 0.76953125f, 0.22265625f, 1.0f };

    public Triangle() {
        // initialize vertex byte buffer for shape coordinates
        ByteBuffer bb = ByteBuffer.allocateDirect(
                // (number of coordinate values * 4 bytes per float)
                triangleCoords.length * 4);
        // use the device hardware's native byte order
        bb.order(ByteOrder.nativeOrder());
        // create a floating point buffer from the ByteBuffer
        vertexBuffer = bb.asFloatBuffer();
        // add the coordinates to the FloatBuffer
        vertexBuffer.put(triangleCoords);
        // set the buffer to read the first coordinate
        vertexBuffer.position(0);
    }
}
```

Rajzolás

```
TriangleSmallGLUT(float size) {
        if (size != 1.0f) {
            for (int vertex = 0; vertex < vertices.length; vertex++ ) {
                vertices[vertex] *= size;
            }
        }

        mVertexBuffer = getFloatBufferFromFloatArray(triangleCoords);
        mColorBuffer = getFloatBufferFromFloatArray(color);
    }

    void draw(GL10 gl) {
        gl.glFrontFace(GL10.GL_CW);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mVertexBuffer);
        gl.glNormal3f(0f, 0f, 1f);

        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);

    }

    void drawColorful(GL10 gl) {
        gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
        gl.glColorPointer(4,GL10.GL_FLOAT, 0, mColorBuffer);
        draw(gl);
        gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
    }
```

Handler-ek

- Kommunikáció GUI és OpenGL szál között
  - Példák
    - OpenGL szál adatot írna egy Android vezérlőelembe (pl. FPS szöveg)
    - GUI szál által fogadott interakció továbbítása 3D vezérléshez
  - Handler objektumokkal

```
public final Handler mHandler = new Handler();
public void toggleFPSDisplay() {
        showFPS = !showFPS;
        if (!showFPS) {
            mHandler.post(new Runnable() {
                public void run() {
                    mFPSText.setText("FPS off (press 'f' to turn on)");
                }
            });
        }
    }
```

Android OpenGL ES 2.0

- GLSurfaceView konstruktorában
  - **`setEGLContextClientVersion(2);`**
  - Renderer létrehozása előtt
- OpenGL függvényhívások

```
@Override
public void onDrawFrame(GL10 unused) {
    if (!initialized) {
        return;
    }
    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT);
    GLES20.glUseProgram(shaderProgram);
    GLES20.glVertexAttribPointer(0, 3, GLES20.GL_FLOAT, false, 12,
verticesBuffer);
    GLES20.glEnableVertexAttribArray(0);
    GLES20.glDrawArrays(GLES20.GL_TRIANGLES, 0, 3);
}
```

Shader-ek kezelése

```
private int loadAndCompileShader(int shaderType, int shaderId)
        throws Exception {
        InputStream inputStream =
            AndroidGL2Activity.this.getResources().openRawResource(shaderId);
        String shaderCode = inputStreamToString(inputStream);
        int shader = GLES20.glCreateShader(shaderType);
        if (shader == 0) {
            throw new Exception("Can't create shader");
        }
        // hand the code over to GL
        GLES20.glShaderSource(shader, shaderCode);
        // compile it
        GLES20.glCompileShader(shader);
        // get compile status
        int[] status = new int[1];
        GLES20.glGetShaderiv(shader, GLES20.GL_COMPILE_STATUS, status, 0);
        if (status[0] == 0) {
            // failed
            Log.e(DEBUG_TAG, "Compiler Failure: "
                + GLES20.glGetShaderInfoLog(shader));
            GLES20.glDeleteShader(shader);
            throw new Exception("Shader compilation failed");
        }
        return shader;
    }
```

Shader-ek kezelése #2

```
private int shaderProgram = 0;

        private void initShaderProgram(int vertexId, int fragmentId)
            throws Exception {
            int vertexShader =
                loadAndCompileShader(GLES20.GL_VERTEX_SHADER, vertexId);
            int fragmentShader =
                loadAndCompileShader(GLES20.GL_FRAGMENT_SHADER, fragmentId);
            shaderProgram = GLES20.glCreateProgram();
            if (shaderProgram == 0) {
                throw new Exception("Failed to create shader program");
            }
            // attach the shaders to the program
            GLES20.glAttachShader(shaderProgram, vertexShader);
            GLES20.glAttachShader(shaderProgram, fragmentShader);
            // bind attribute in our vertex shader
            GLES20.glBindAttribLocation(shaderProgram, 0, "vPosition");
            // link the shaders
            GLES20.glLinkProgram(shaderProgram);
            // check the linker status
            int[] linkerStatus = new int[1];
            GLES20.glGetProgramiv(shaderProgram, GLES20.GL_LINK_STATUS,
                linkerStatus, 0);
            if (GLES20.GL_TRUE != linkerStatus[0]) {
                Log.e(DEBUG_TAG, "Linker Failure: "
                    + GLES20.glGetProgramInfoLog(shaderProgram));
                GLES20.glDeleteProgram(shaderProgram);
                throw new Exception("Program linker failed");
            }
            GLES20.glClearColor(0.5f, 0.5f, 0.5f, 1);
        }
```

Shader-ek #3

- simple_vertex.shader

```
attribute vec4 vPosition;
void main()
{
    gl_Position = vPosition;
}
```

- simple_fragment.shader

```
precision mediump float;
void main()
{
    gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
}
```

Összetett modellek

- Célszerű külső modellező programmal készíteni
  - Pl. Blender
- Nyílt formátum
  - Wavefront OBJ
  - Könnyen olvasható

További példák

- Android Wireless Application Development könyv
  - SimpleOpenGL példaprogram
    - Kontextus manuális létrehozása
    - Kommunikáció a UI és az OpenGL szálak között
    - Összetettebb objektumok modellezése
    - Fények, textúrák alkalmazása
  - SimpleOpenGL2 példaprogram
    - OpenGL ES 2.0 használata
  - SimpleNDK példaprogram
    - OpenGL ES C kódból
- NDK samples
  - GL2JNIActivity példaprogram
- Android SDK samples
  - APIDemo grafika példái

Magasabb szintű modellezés

- Unity 3D
  - Multiplatform 3D játékmotor
    - iOS, Android, Windows Store, Windows Phone, BlackBerry, asztali PC és web platformokra kódgenerálás
  - Windows és MacOS rendszeren érhető el
  - Ingyenes verzió + Pro előfizetés ($75 havonta)
  - Jellemzők
    - Fejlett grafikus funkciók
    - 2D és 3D képek/modellek importálása
      - 3DS Max, Maya, Blender, …
    - Shader programozás
    - Fizikai modellezés
      - PhysX motor

  - AngryBots példaprogram
  - UnityStudent: Unity és UnityScript tananyagok