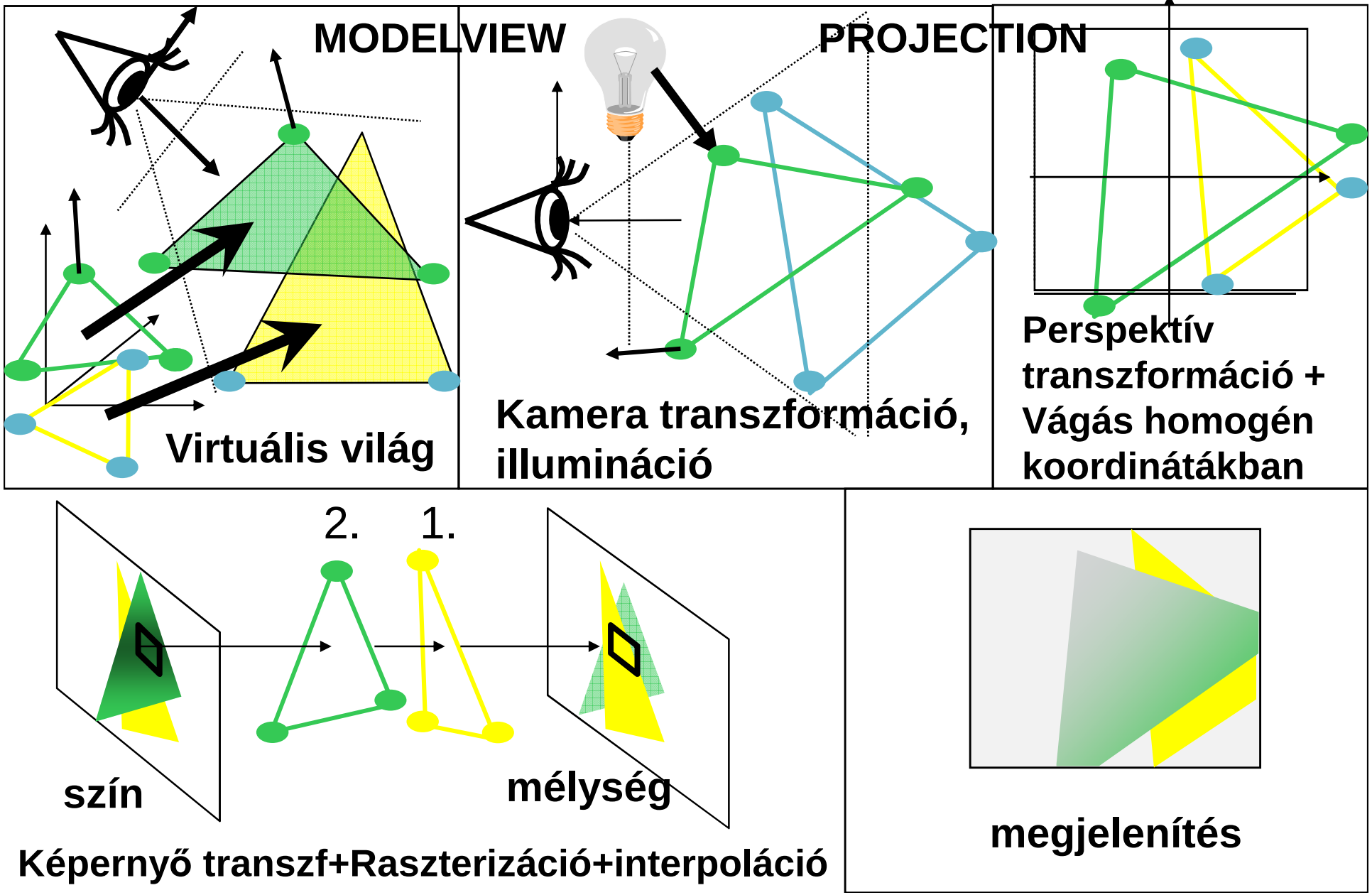
Bevezetés

GPU általános célú programozása

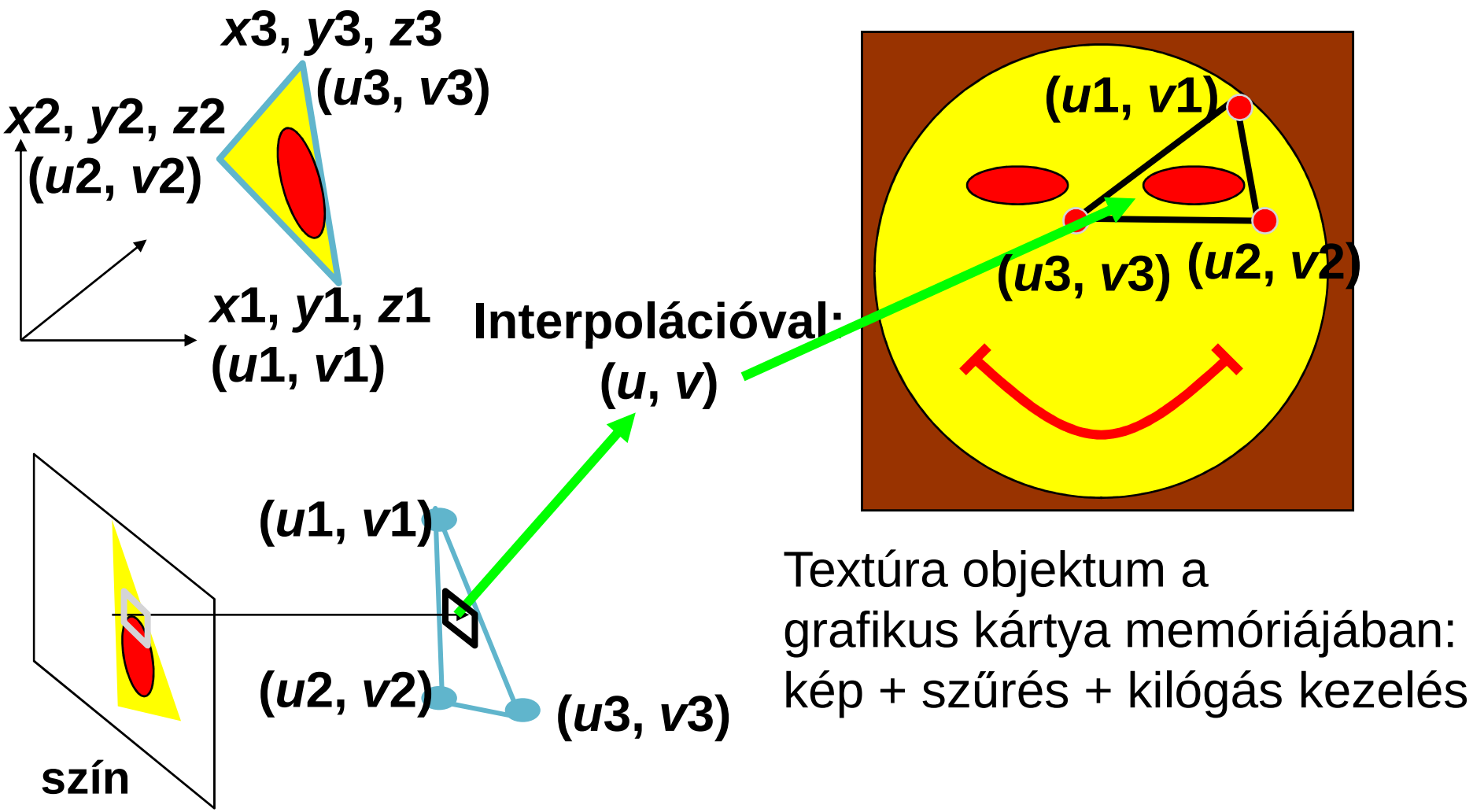
GPU = Graphics Processing Unit

- A számítógépes grafika inkrementális képszintézis algoritmusának hardver realizációja
- Teljesítménykövetelmények:
  - Animáció: néhány nsec / képpont
- Masszívan párhuzamos
  - Pipeline (stream processzor)
  - Párhuzamos ágak

Inkrementális képszintézis



Textúra leképzés



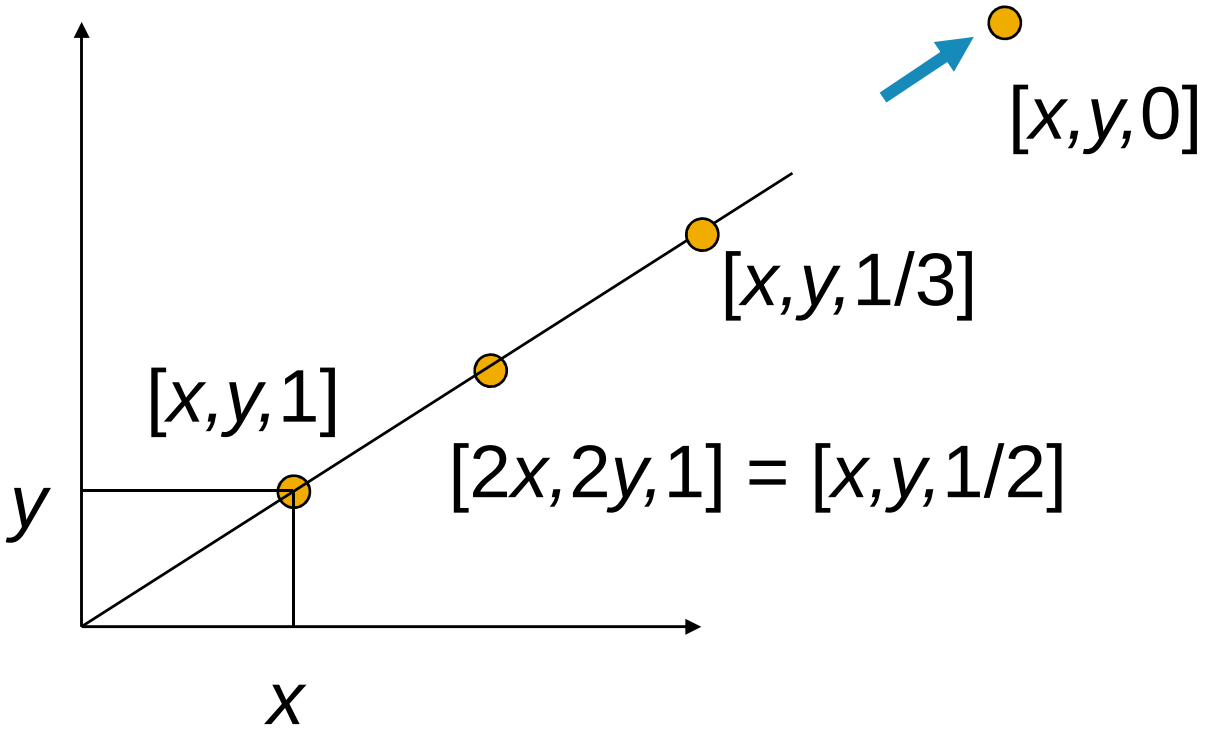
Textúra objektum a grafikus kártya memóriájában: kép + szűrés + kilógás kezelés

Pontok: homogén koordináták

- Descartes: (x,y,z)
- Homogén: $[X,Y,Z,w] = [xw,yw,zw,w]$
- Miért jó?
  - Eltolás, elforgatás, középpontos vetítés, stb. mind 4x4-es mátrixszal szorzás
  - Több transzformáció egyetlen mátrixban összefogható
  - Párhuzamosok nem okoznak gondot (párhuzamosok ideális pontban metszik egymást)
  - A homogén koordináták lineáris transzformációi: háromszöget háromszögbe visznek át

Homogén koordináták értelmezése

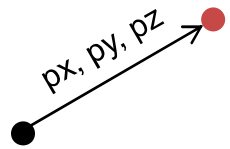
- x, y, z az irány
- w a távolság inverze



Példák homogén lineáris transzformációkra

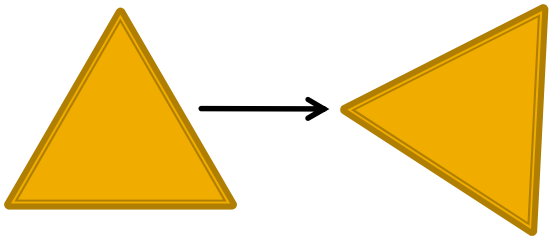
- Eltolás

$$[x,y,z,1]\begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ px, & py, & pz, & 1 \end{bmatrix}$$



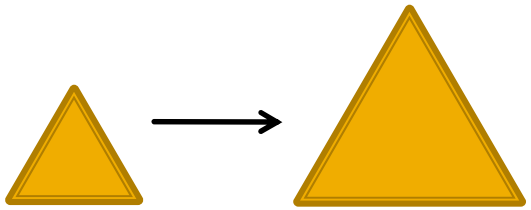
- Elforgatás

$$[x,y,z,1]\begin{bmatrix} \cos\alpha & \sin\alpha & & \\ -\sin\alpha & \cos\alpha & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$



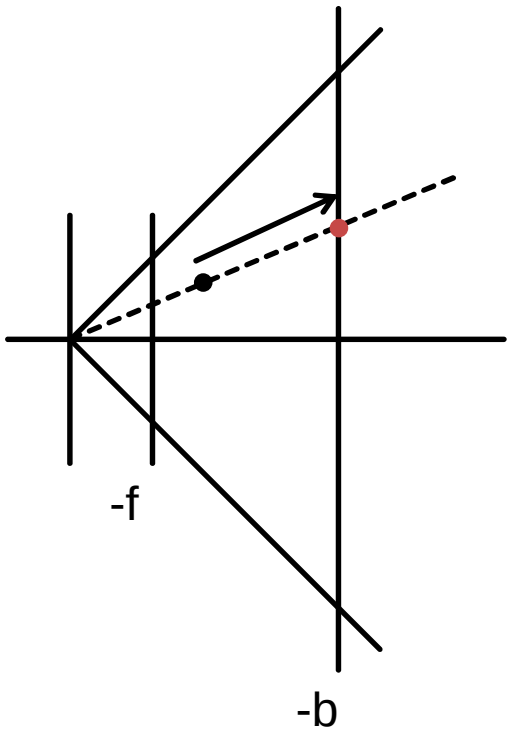
- Skálázás

$$[x,y,z,1]\begin{bmatrix} Sx & & & \\ & Sy & & \\ & & Sz & \\ & & & 1 \end{bmatrix}$$



Példák homogén lineáris transzformációkra

- Vetítés



$$(x,y,z)\to\left(\frac{x}{-z},\frac{y}{-z},-1\right)$$ Nem lineáris

$$z'=-1\ldots 1$$

$$\left.\begin{array}{l} z=-f\ \to z'=-1 \\ z=-b\to z'=1 \end{array}\right\}\quad z'=\alpha z+\beta$$

$$\Downarrow$$

$$(x',y',z',w')=\left(\frac{x}{-z},\frac{y}{-z},\alpha z+\beta,1\right)$$

$$\left(x,y,\underline{-\alpha z^2-\beta},-z\right)$$

Nem lineáris

Példák homogén lineáris transzformációkra

- Vetítés

$$z' = \alpha + \frac{\beta}{z}$$

$$\Downarrow$$

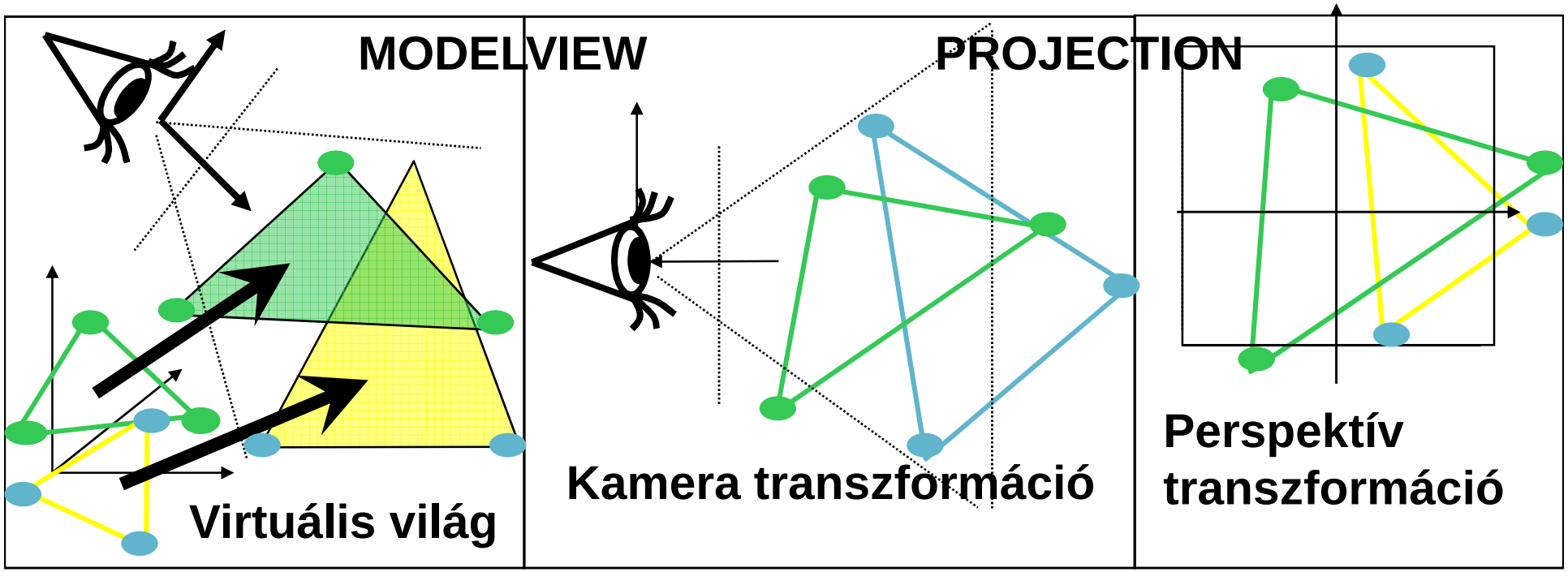
$$(x', y', z', w') = \left(\frac{x}{-z}, \frac{y}{-z}, \alpha + \frac{\beta}{z}, 1 \right)$$

$$(x, y, -\alpha z - \beta, -z)$$

$$[x, y, z, 1] \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -\alpha & -1 \\ & & -\beta & 0 \end{bmatrix}$$

Lineáris

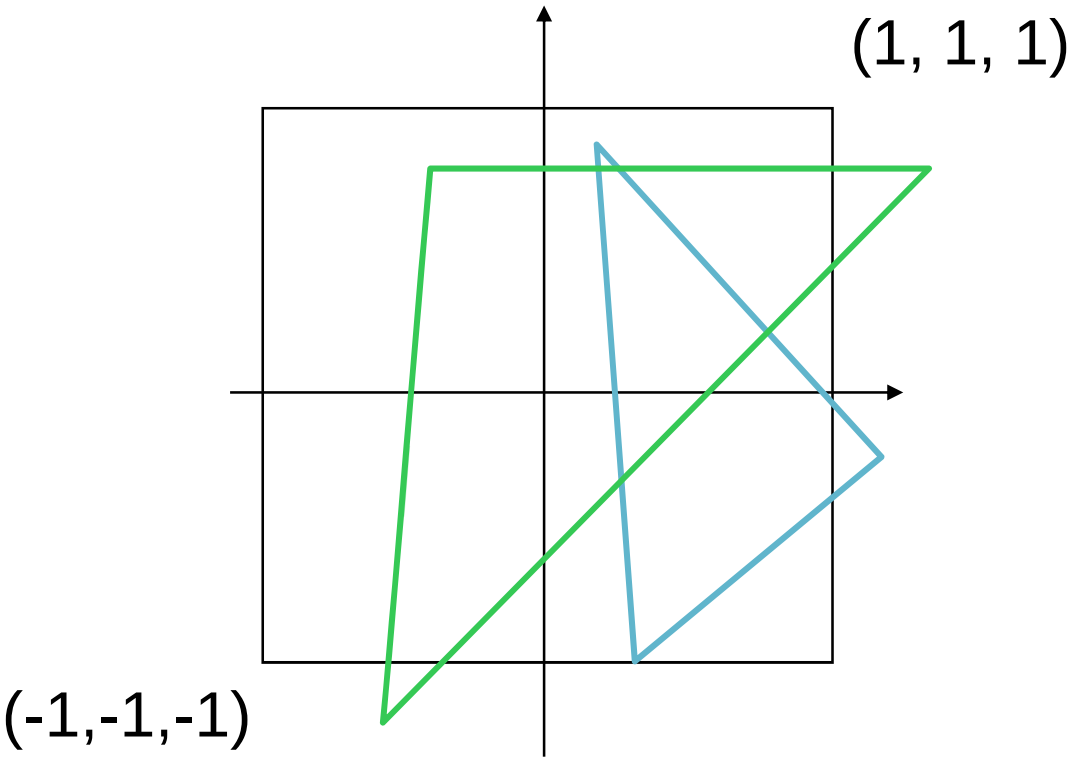
Csúcspontok (vertex) feldolgozása



$$[x,y,z,1] \cdot T_{\text{modelviewproj}} = [X,Y,Z,w]$$

Z_{camera}

Vágás

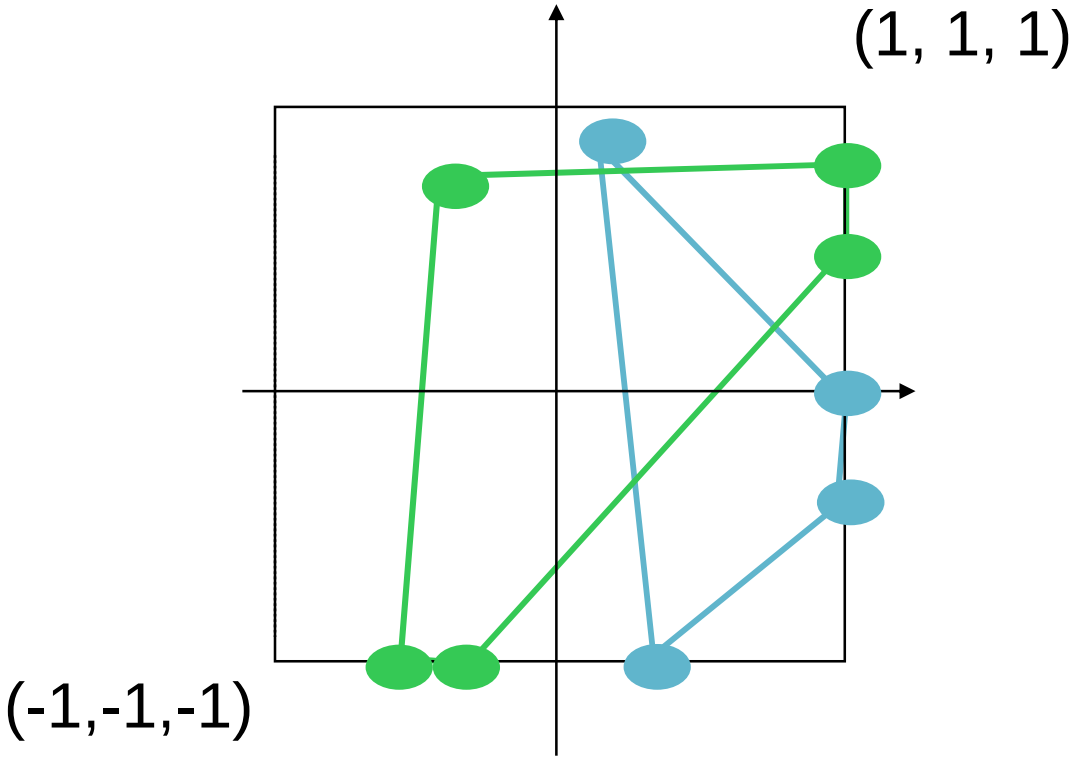


$$-1 < x < 1$$
$$-1 < y < 1$$
$$-1 < z < 1$$
$$w > 0$$

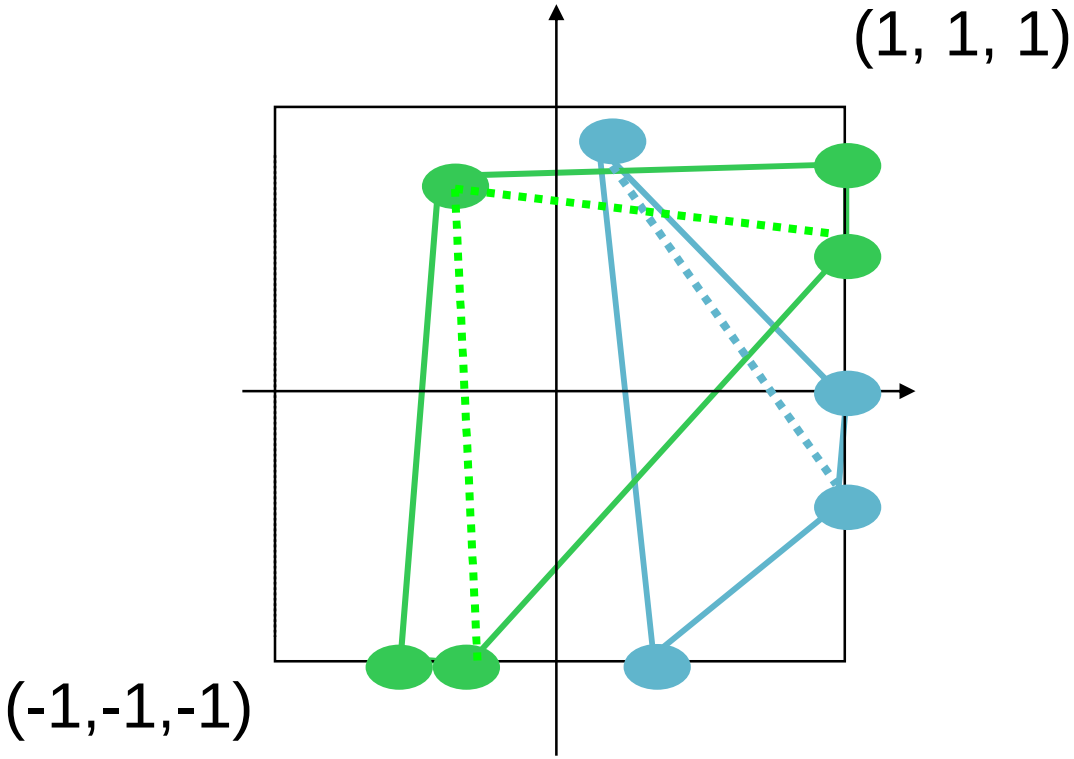
$$-w < X < w$$
$$-w < Y < w$$
$$-w < Z < w$$

$$[X, Y, Z, w] = [xw, yw, zw, w]$$

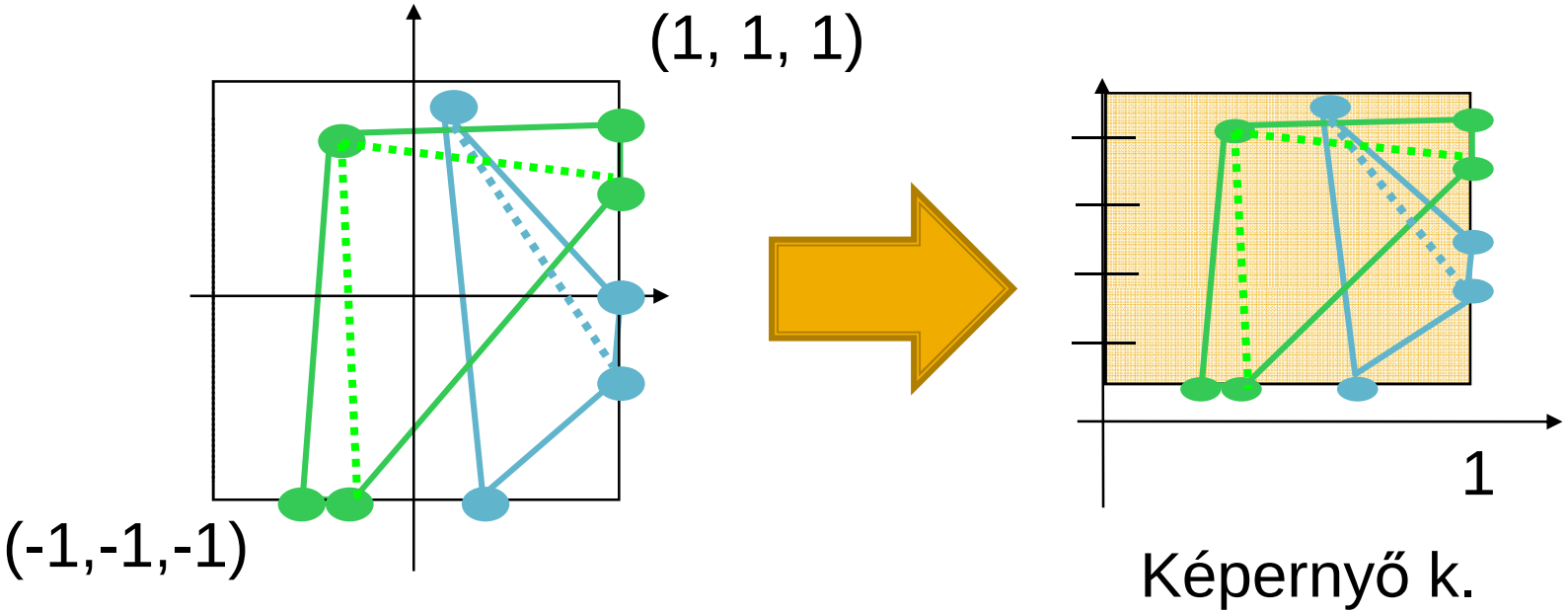
Vágás



Vágás



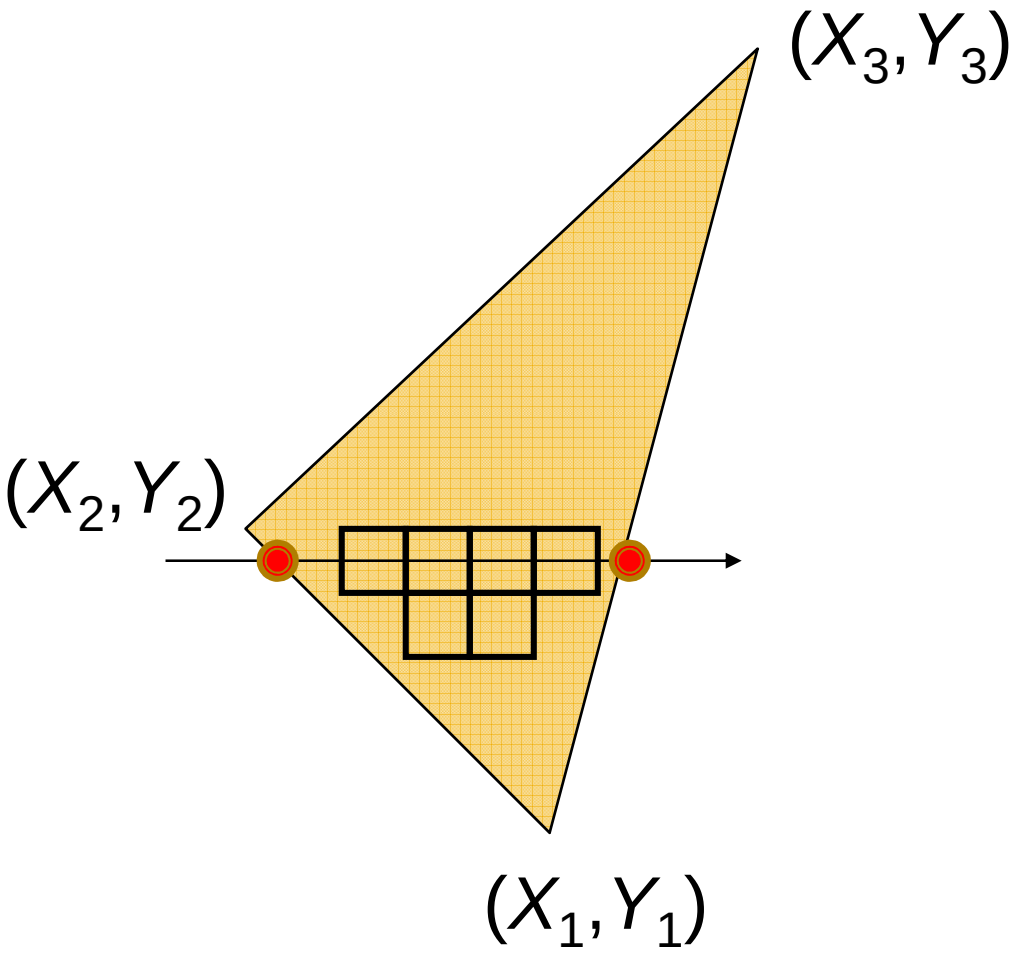
Képernyő (viewport) transzformáció



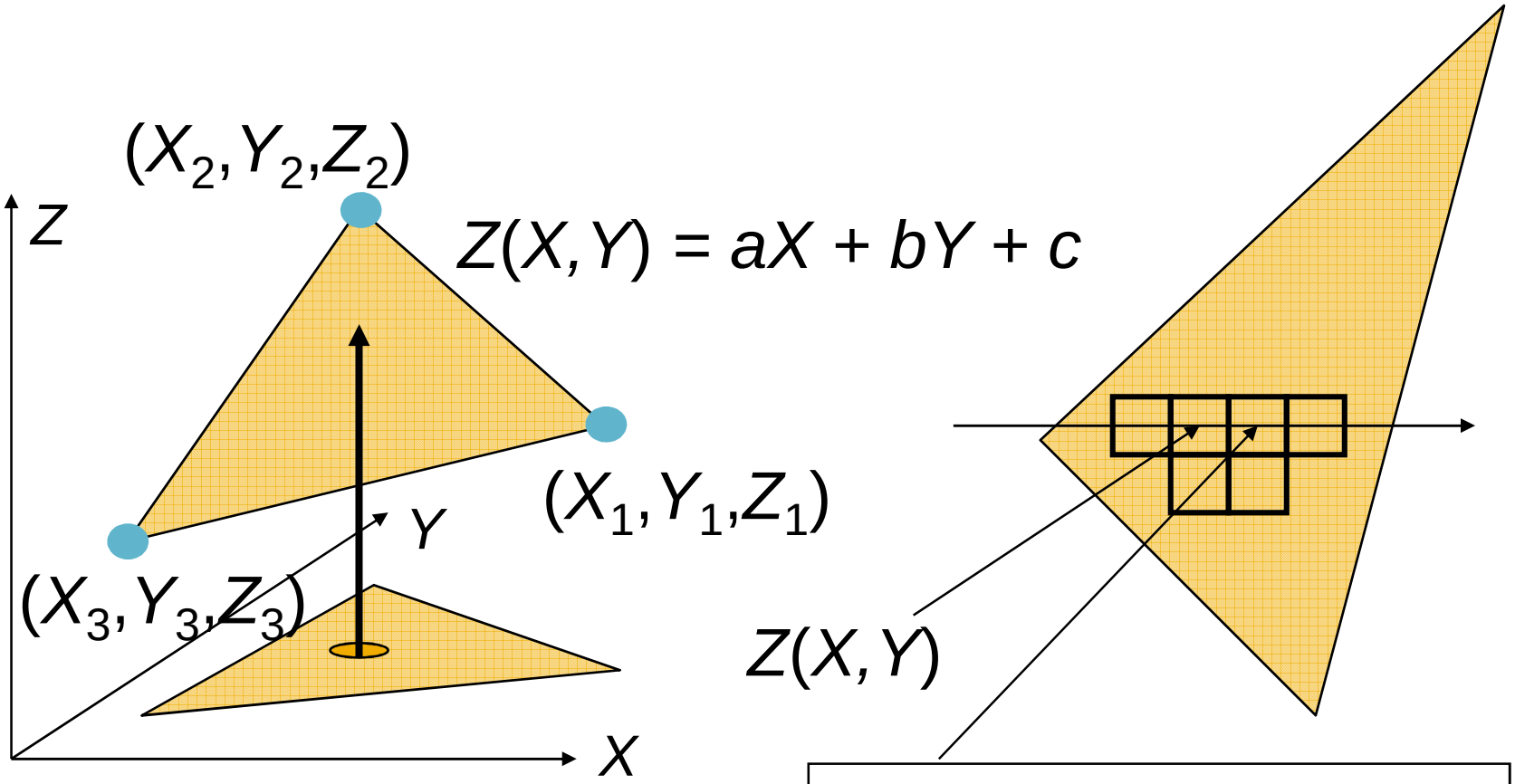
Z(-1,1)

(0,1)

Raszterizáció

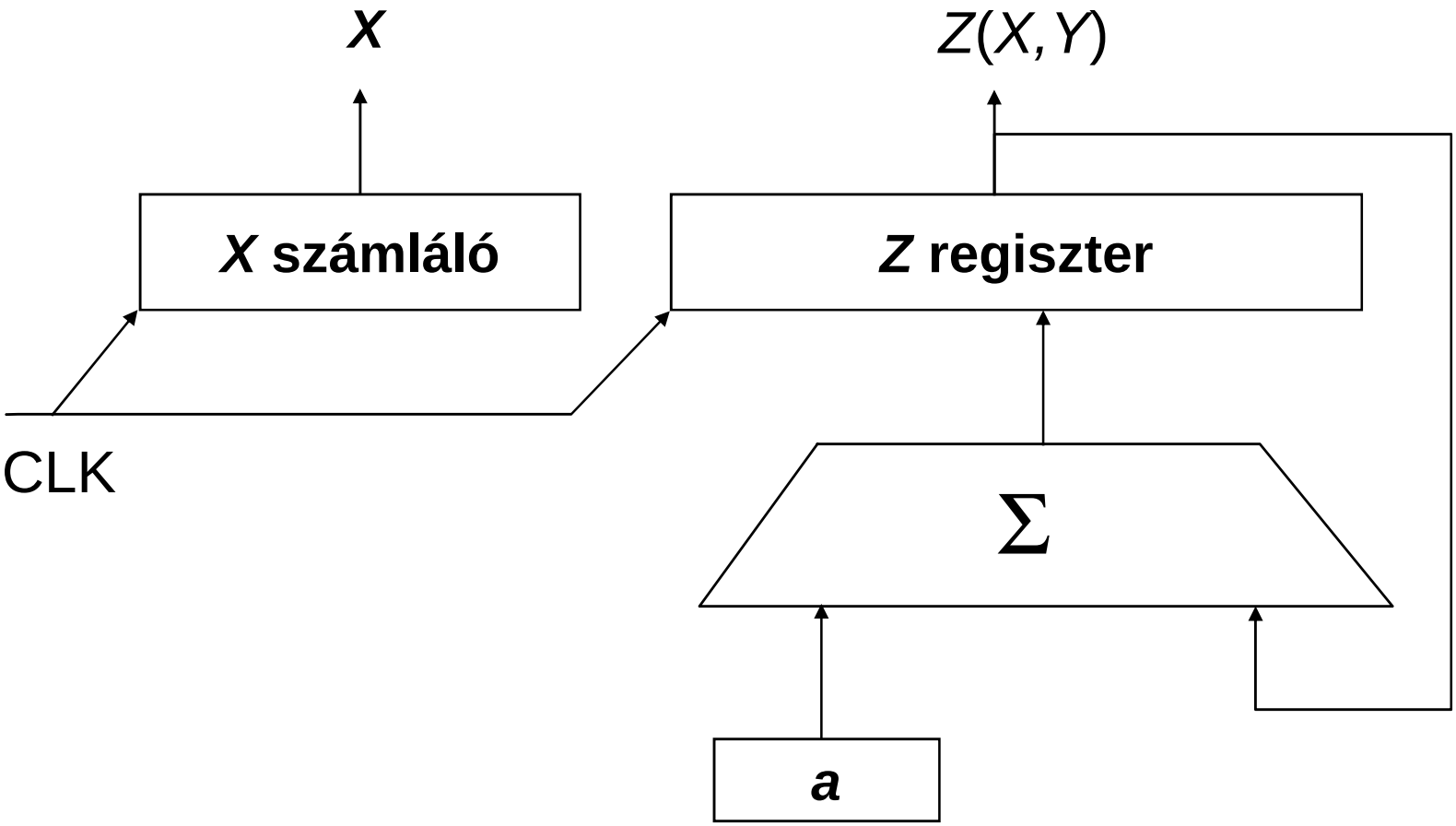


Tulajdonságok interpolációja

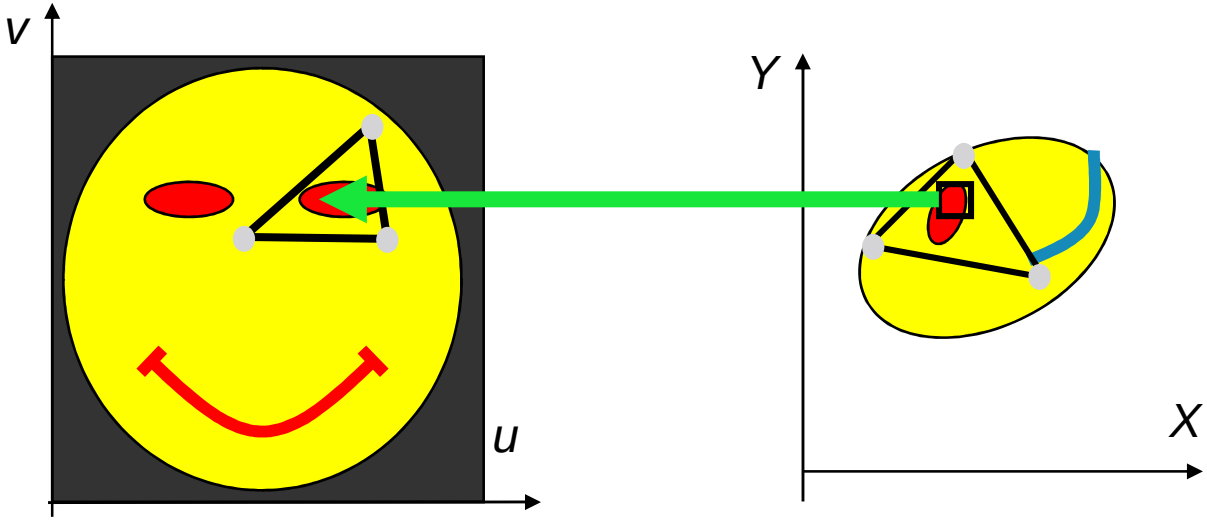


$$Z(X+1,Y) = Z(X,Y) + a$$

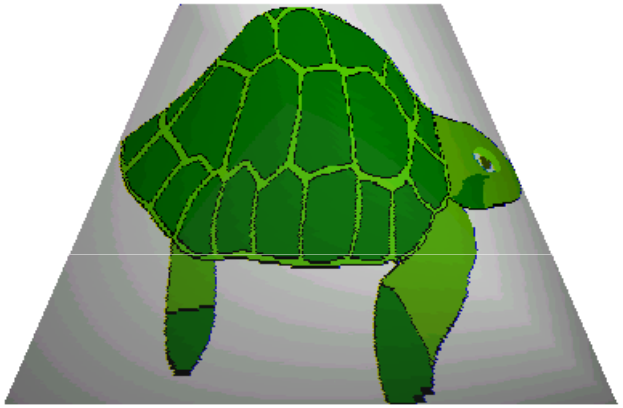
Interpolációs hardver



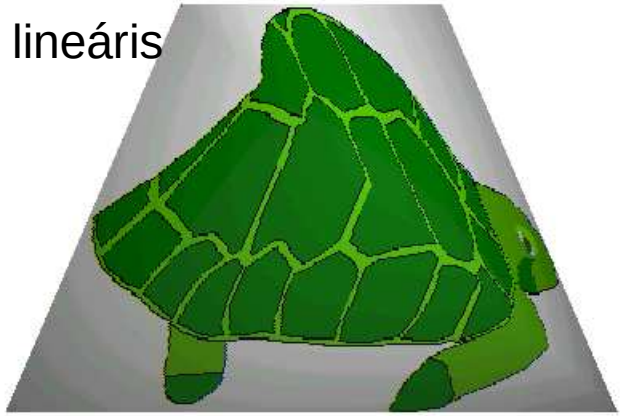
Perspektív helyes interpoláció



Perspektív helyes



lineáris



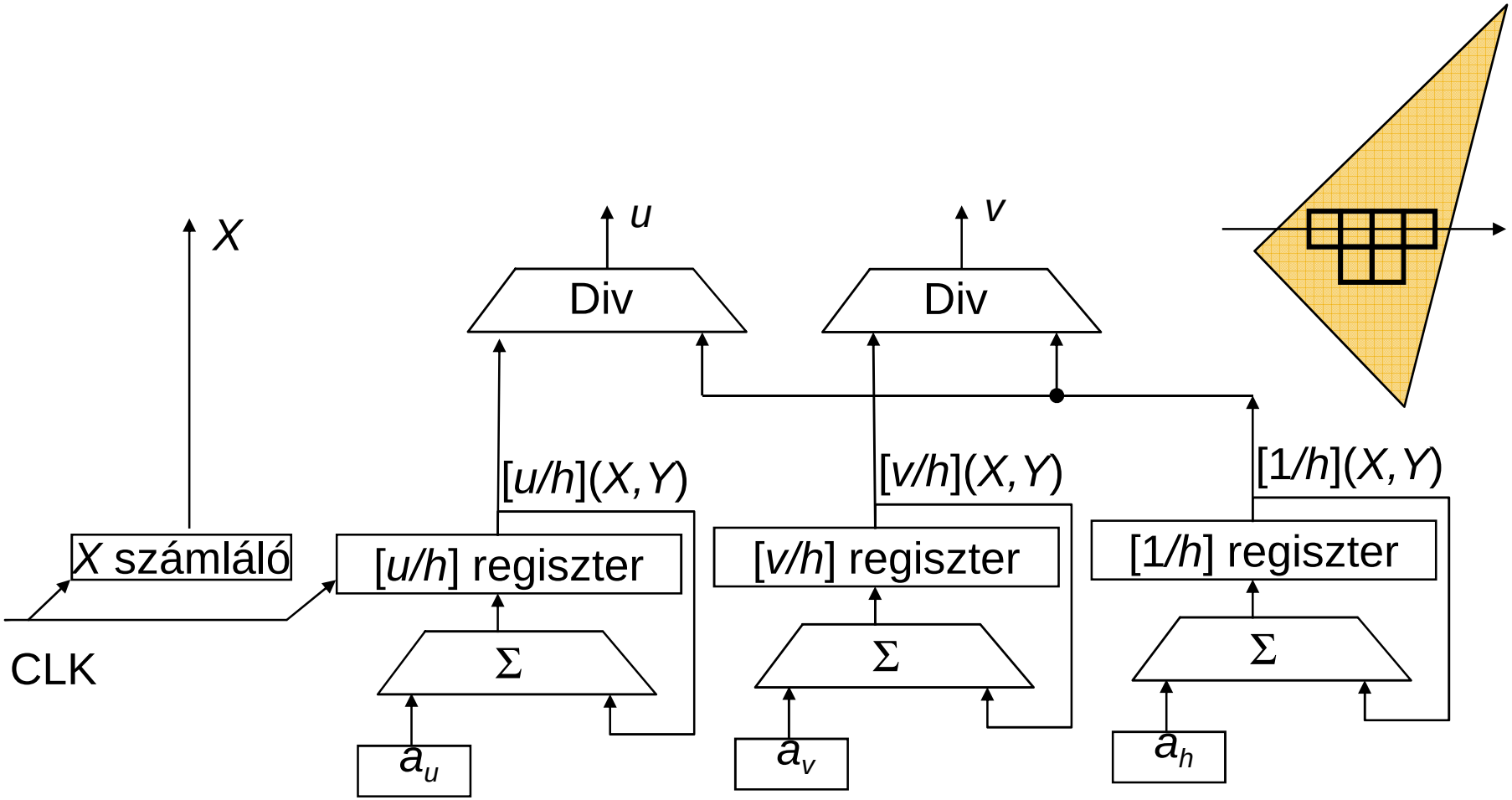
$$u/w = a_u X + b_u Y + c_u$$
$$v/w = a_v X + b_v Y + c_v$$
$$1/w = a_w X + b_w Y + c_w$$

w = -z

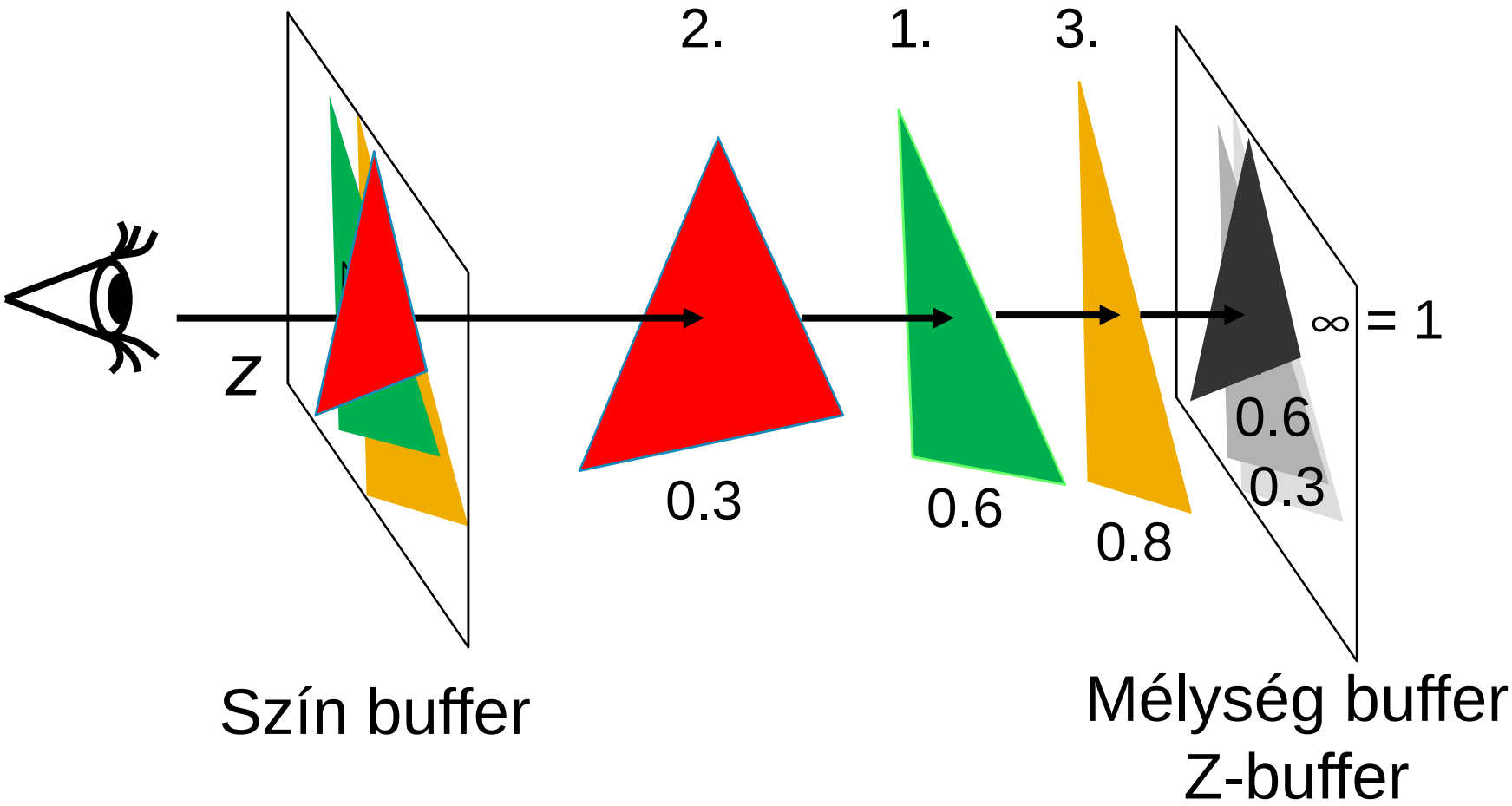
$$u = \frac{a_u X + b_u Y + c_u}{a_h X + b_h Y + c_h}$$

$$v = \frac{a_v X + b_v Y + c_v}{a_h X + b_h Y + c_h}$$

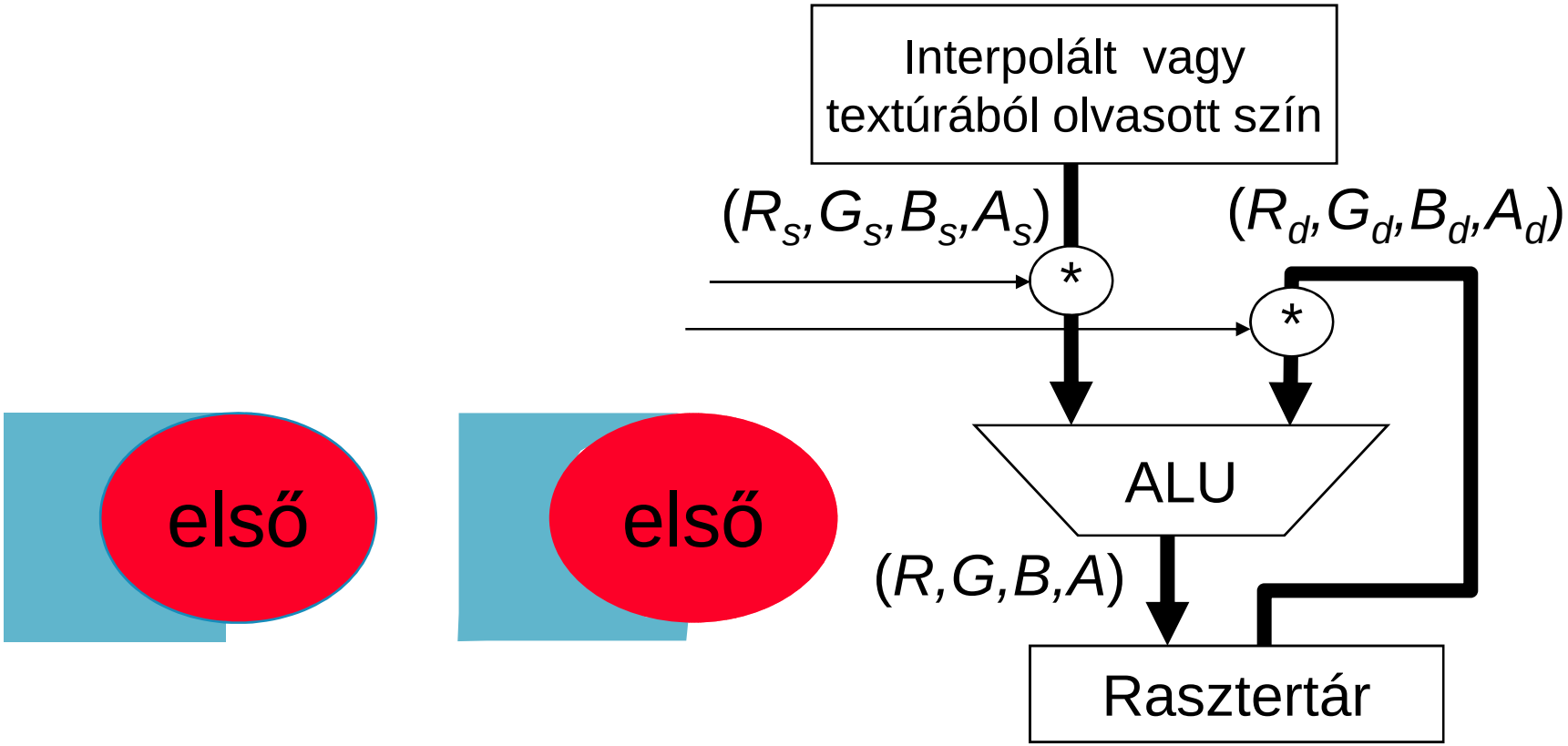
Homogén lineáris interpolációs hardver



Kompozitálás: Z-buffer algoritmus

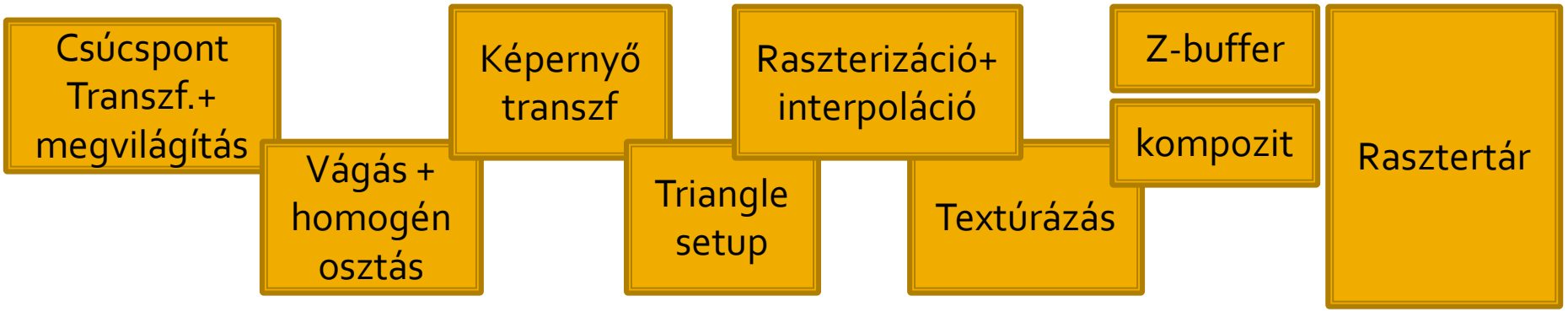


Kompozitálás: alfa blending



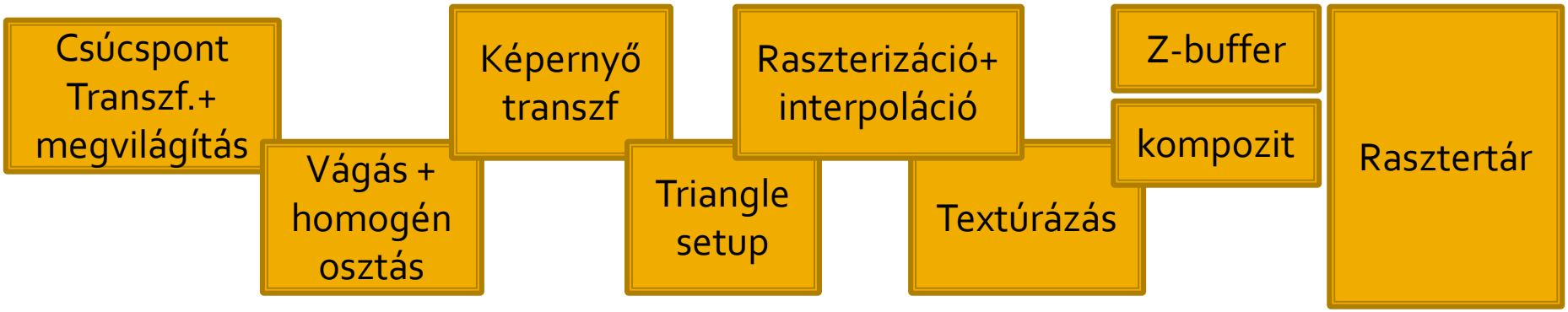
$$(R,G,B,A) = (R_sA_s+R_d(1-A_s),\ G_sA_s+G_d(1-A_s),\ B_sA_s+B_d(1-A_s),\ A_sA_s+A_d(1-A_s))$$

Képszintézis csővezeték



A PC-s GPU története

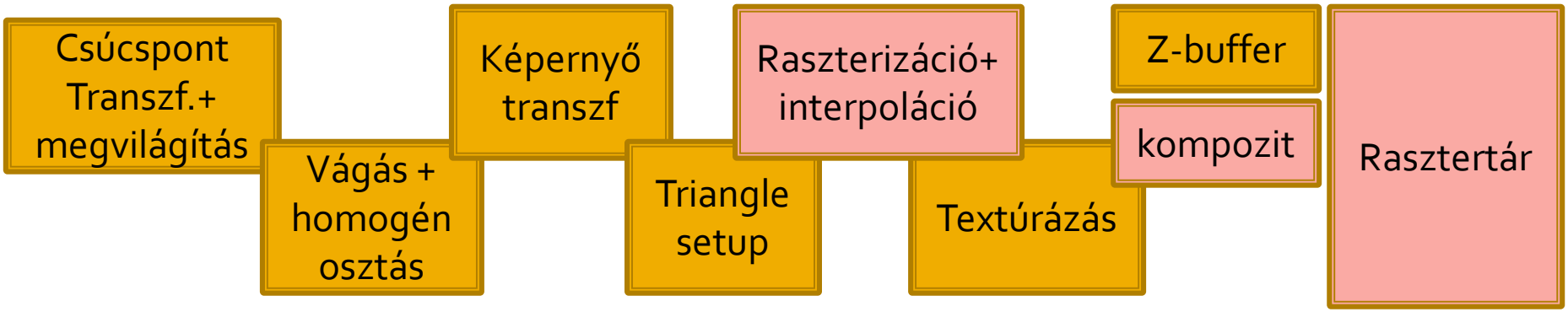
- Korai nem PC rendszerek
  - A teljes csővezeték megvalósítva
  - Workstation és Graphics Terminal
    - HP, Sun, Textronix



A PC-s GPU története

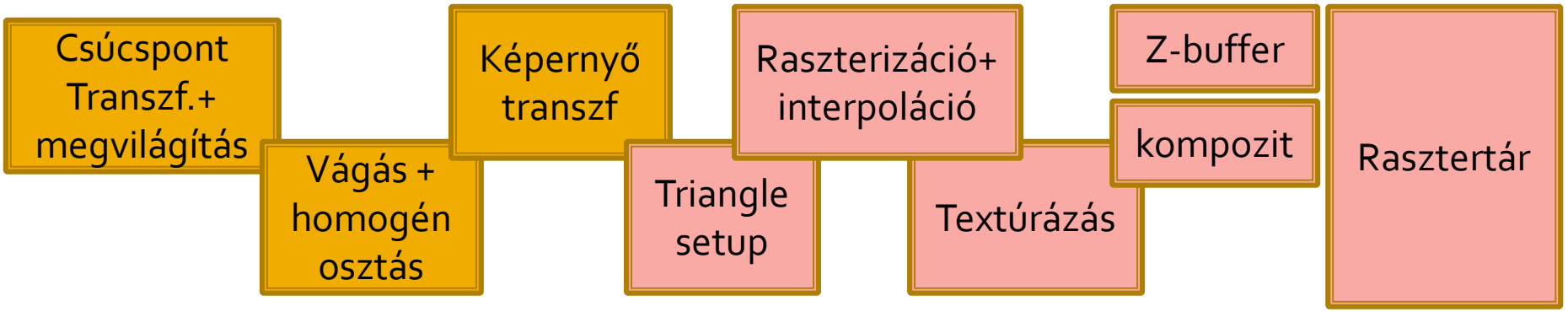
- 1970-1990
  - Tipikusan 2D
  - Spriteok, primitivek
    - Blitter, vonal, négyszög rajzolás
  - Ablakkezelő gyorsítás

  - Amiga, Atari, S3



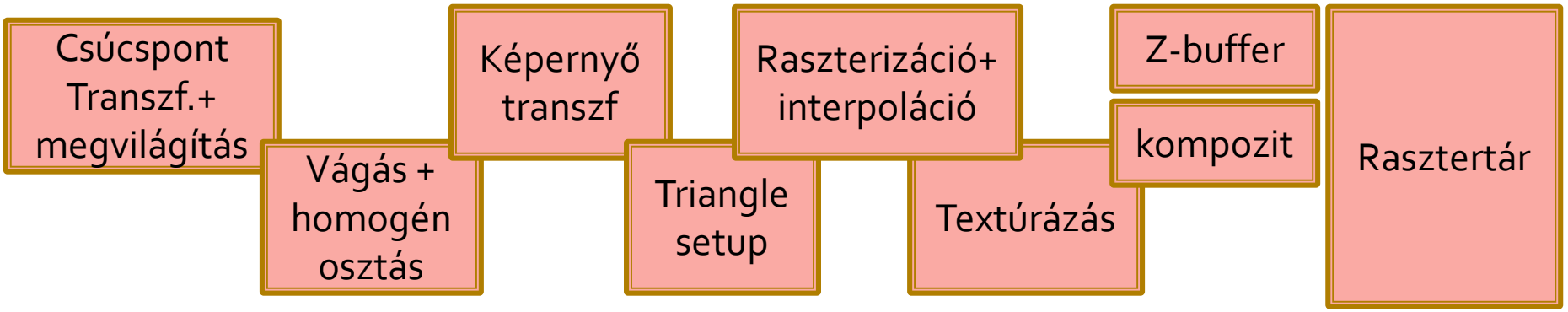
A GPU története

- 1990-1998
  - Vertex feldolgozás, textúra szűrés, Z-buffer
    - PlayStation, Nintendo64
    - S3 Virge, ATI Rage, Matrox Mystique
    - 3dfx Voodoo
    - 3dfx Glide
    - Microsoft DirectX 5.0



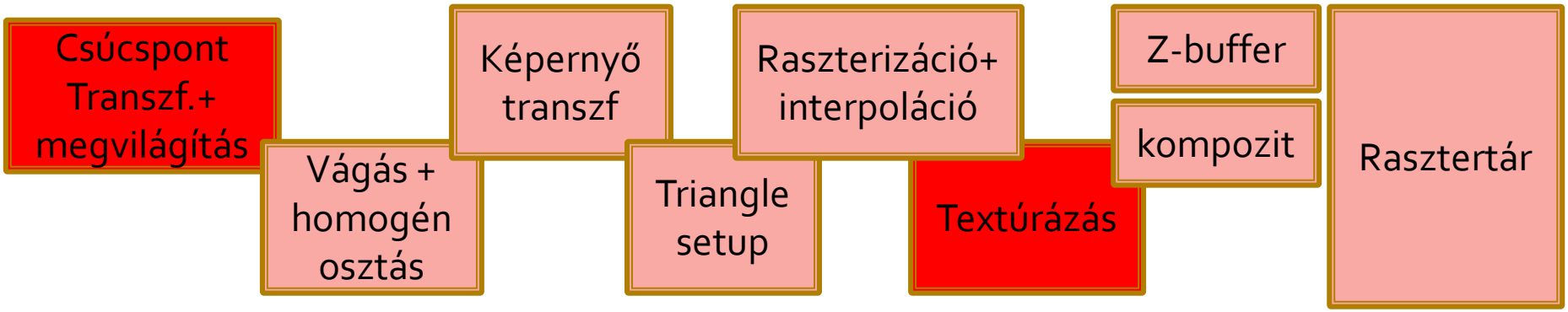
A GPU története

- 1990-2000
  - A csővezetés felbontása
    - Transform & Lighting
    - Microsoft DirectX 7.0
    - OpenGL
  - NVIDIA GeForce 256 (NV10)
    - az első nem professzionális 3D gyorsító



A GPU története

- 2000 –
  - Shaderek megjelenése
    - Vertex és fragmens shader (2001)
    - Geometria shader (2006)
    - Programozható tesszelátor (2010)



Grafikus API

- Általános felület a GPU-hoz
  - Csővezeték vezérlése
    - Shaderek
  - Geometria definíció
  - Textúrák
  - Framebuffer
    - Párhuzamos renderelés

Grafikus API

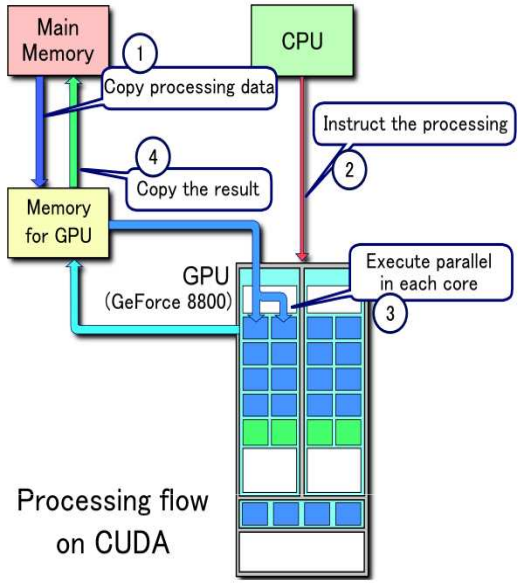
- Microsoft DirectX
  - Windows specifikus
  - DirectX7: transform & lighting
  - DirectX9: vertex és pixel shader
  - DirectX10: geometry shader
  - DirectX11: tesselation, DirectCompute

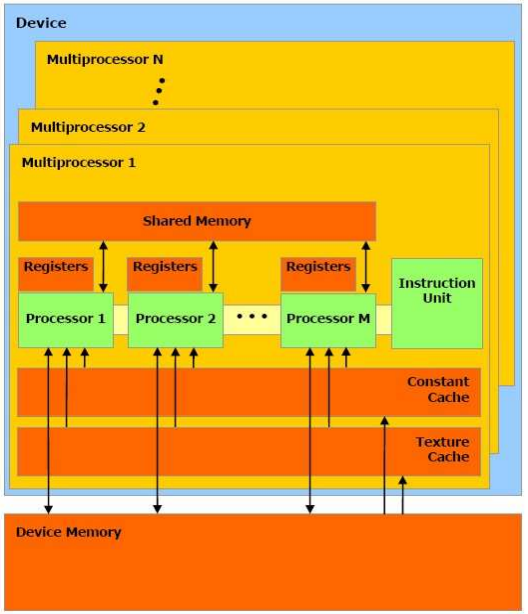
Grafikus API

- OpenGL
  - OpenGL 2.0: vertex és pixel shader
  - OpenGL 3.0: framebuffer objects
  - OpenGL 3.1: openCL interop
  - OpenGL 3.2: geometry shader
  - OpenGL 3.3/4.0: tesszeláció, GPGPU

GPGPU API

- CUDA
  - Compute Unified Device Architecture
  - 2007. NVIDIA
  - Magas szintű, C szerű nyelv
    - Atomi műveletek
    - Szavazás
    - Dupla pontosságú számítás
    - Szinkronizáció

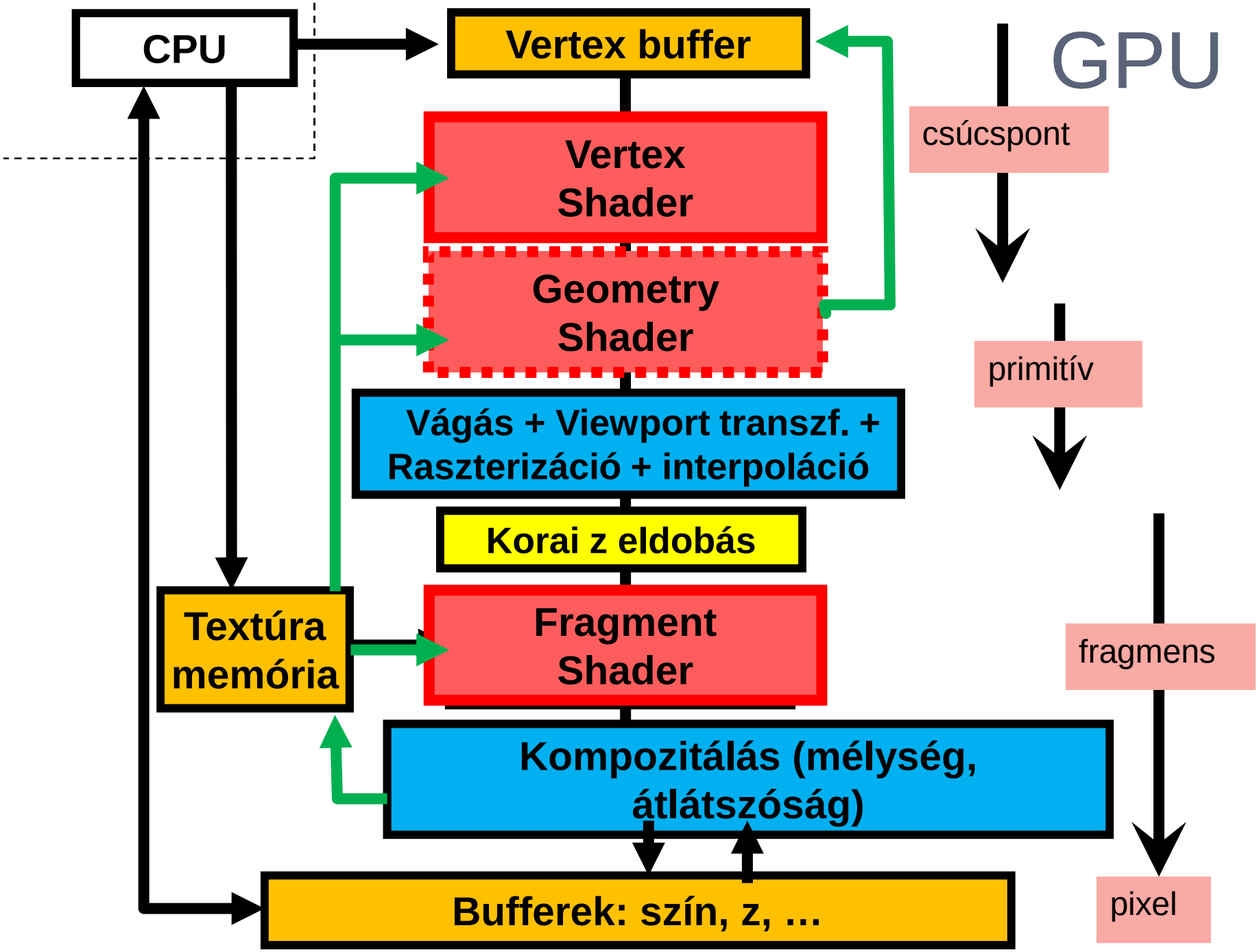




GPGPU API

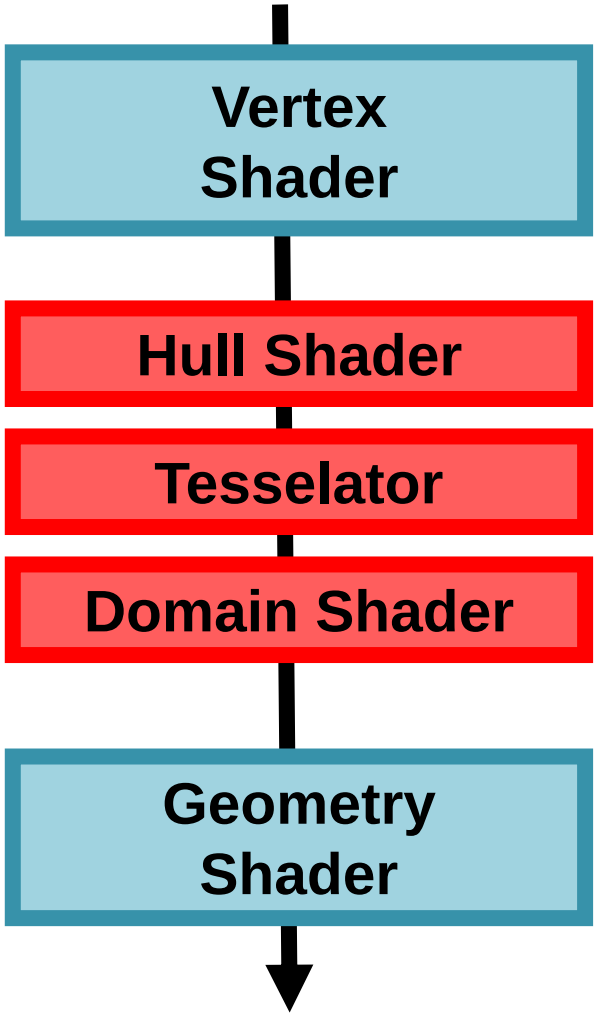
- OpenCL
  - 2008 november, Khronos Group
  - Gyártó független GPGPU API
  - CPU és GPU összemosás

  - ATI, Intel, Apple, IBM és NVIDIA

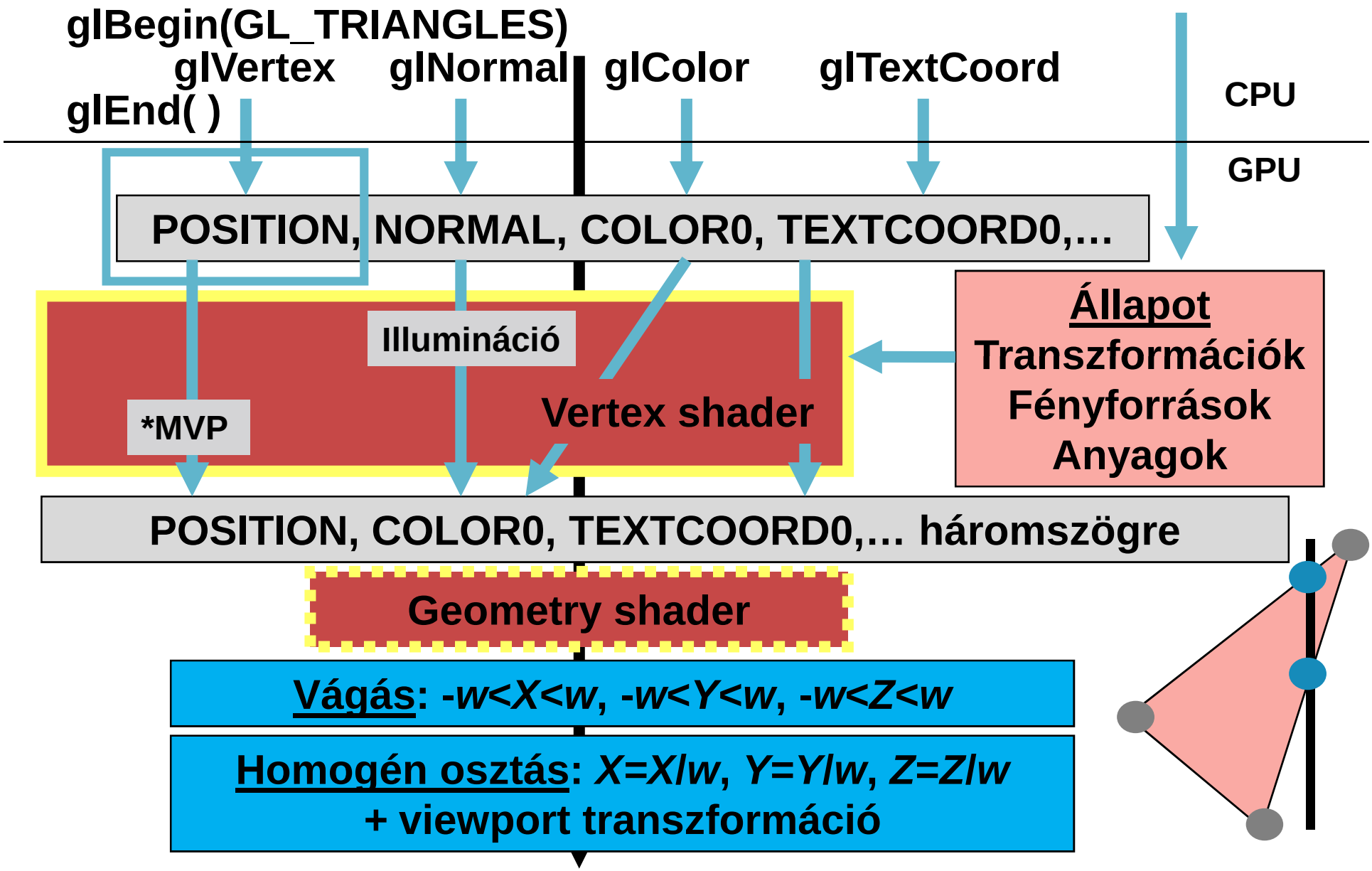

CPU
Vertex buffer
GPU
csúcspont
Vertex Shader
Geometry Shader
primitív
Vágás + Viewport transzf. + Raszterizáció + interpoláció
Korai z eldobás
Textúra memória
Fragment Shader
fragmens
Kompozitálás (mélység, átlátszóság)
Bufferek: szín, z, ...
pixel

Tesszelátor

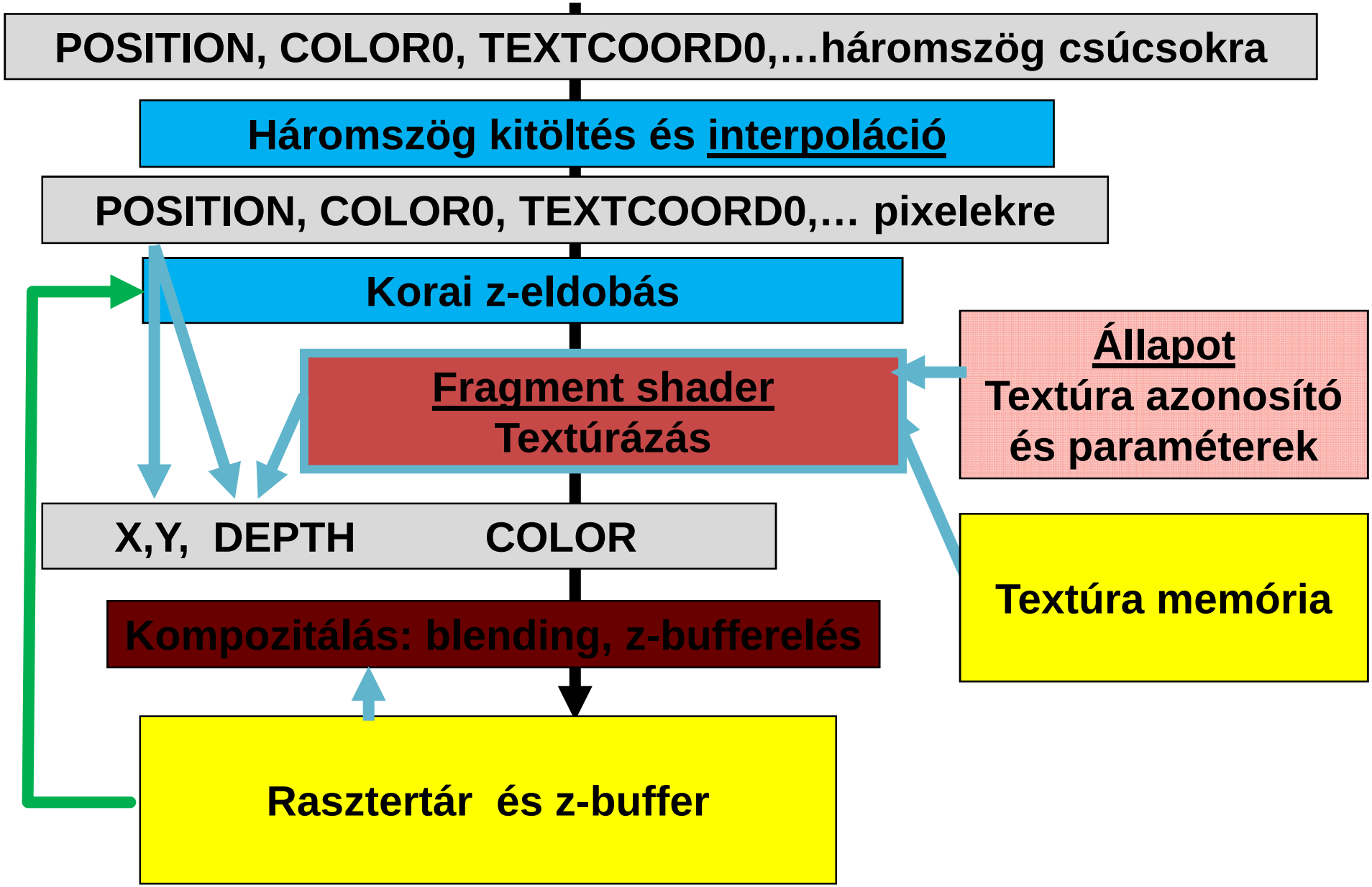
- Felszín árnyaló (Hull Shader)
  - Felbontás vezérlése
- Tesszelátor (Tesselator)
  - Fix funkciójú egység
- Tartomány árnyaló (Domain Shader)
  - Felbontás ellenőrzése
  - Vertexek létrehozása



Vertex shader és környezete



Pixel shader és környezete



OpenGL/Cg (Shader Model 3)

- **.cpp CPU program:**
  - GLUT, OpenGL inicializálás
  - Shader environment létrehozás
  - GPU képesség beállítás (profile)
  - Vertex/fragment program betöltés és fordítás: CREATE
  - Vertex/fragment program átadás a GPU-nak: LOAD
  - Vertex/fragment program kiválasztás: BIND
  - Uniform vertex/fragment input változó létrehozás

Inicializálás

Display

  - Uniform vertex/fragment változó értékadás
  - Változó input változó értékadás (glVertex, glColor, glTexCoord)

- **.cg vertex program**
  - Fragment program változó input változó + homogén pozíció

- **.cg fragment program**
  - Szín kimenet

CPU program

- OpenGL/GLUT inicializálás, textúra feltöltés

```
#include <GL/glut.h>

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA|GLUT_DEPTH|GLUT_DOUBLE);
        glutInitWindowSize(600, 600);
        glutCreateWindow("GLUT Window");

        InitCg();

        glutDisplayFunc(onDisplay);
        glutKeyboardFunc(onKeyboard);
        glutIdleFunc(onIdle);

        glutMainLoop();
        return(0);
}
```

CPU program

- Árnyaló inicializálás

```
…
#include <Cg/cgGL.h>                          // Cg függvények

CGparameter LightDir, TextureShift;   // uniform paraméterek a CPU-n

int InitCg( ) {
   CGcontext shaderContext = cgCreateContext(); // árnyaló kontextus
```

Vertex shader betöltés

```
   CGprofile vertexProf = cgGLGetLatestProfile(CG_GL_VERTEX);
   cgGLEnableProfile(vertexProf);

   CGprogram vertexProgram = cgCreateProgramFromFile(
                                    shaderContext,
                                    CG_SOURCE, “vertex.cg",
                                    vertexProf, NULL, NULL);

   cgGLLoadProgram(vertexProgram);        // GPU-ra töltés
   cgGLBindProgram(vertexProgram);        // ez legyen a futó program

    // vertex program uniform paraméterek. CPU-n LightPos; GPU-n lightpos
   LightPos = cgGetNamedParameter(VertexProgram, “lightpos");
```

Fragment program betöltés

```
CGprofile fragmentProf =
cgGLGetLatestProfile(CG_GL_FRAGMENT);
cgGLEnableProfile(fragmentProf);

CGprogram fragmentProgram = cgCreateProgramFromFile(
                                    shaderContext,
                                    CG_SOURCE, “fragment.cg",
                                    fragmentProf,
                                          NULL, NULL);

cgGLLoadProgram(fragmentProgram); // GPU-ra töltés
cgGLBindProgram(fragmentProgram); // ez a program fusson

                                  // fragmens program uniform paraméterek
TextureShift = cgGetNamedParameter(fragmentProgram, “shift");
```

CPU: Display eseménykezelő

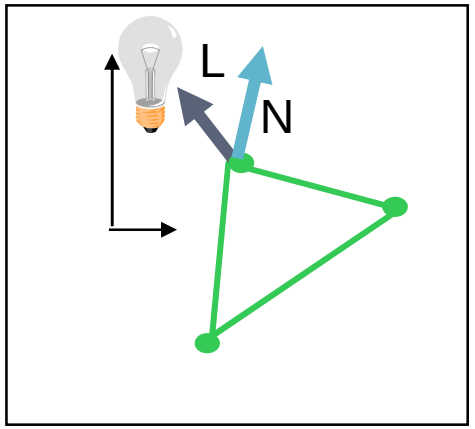
```
void onDisplay( ) {
                    // állapot (implicit uniform) paraméterek beállítása
   glLoadIdentity();
   gluLookAt(0, 0, -10, 0, 0, 0, 0, 1, 0);
   glRotatef(angle, 0, 1, 0);

                    // explicit uniform paraméterek beállítása
   cgGLSetParameter3f(LightPos, 10, 20, 30); // lightpos
   cgGLSetParameter1f(Shine, 40);            // shininess
   cgGLSetParameter3f(Kd, 1, 0.8, 0.2);      // kd
   cgGLSetParameter3f(Ks, 2, 2, 2);          // ks

                    // változó paraméterek, a PASS
   glBegin( GL_TRIANGLES );
   for( … ) {
      glTexCoord2f(u, v);      // TEXCOORD0 regiszter
      glVertex3f(x, y, z);     // POSITION regiszter + trigger
   }
   glEnd();
}
```

Vertex árnyaló

```
void main(
        in float4 position     : POSITION,
        in float4 texcoord     : TEXCOORD0,
        in float3 normal       : NORMAL,
        in float3 color        : COLOR0,
        uniform float4x4 MVP   : state.matrix.mvp,
        uniform float3   lightpos, // fényforrás világ k-ban
        out float4 hposition   : POSITION,
        out float3 otexcoord   : TEXCOORD0,
        out float3 ocolor      : COLOR0
)
{
        hposition = mul(MVP, position);
        ocolor = color * dot(normal, lightpos);
        otexcoord = texcoord;
}
```



Fragmens árnyaló

```
void main( in float2 texcoord    : TEXCOORD0,
           in float3 color       : COLOR0,
           uniform sampler2D texture_map,
           uniform float2    shift,
           out float4 ocolor     : COLOR )
{
      texcoord += shift;
      float3 texcol = tex2D(texture_map, texcoord);
      ocolor = texcol * color;
}
```