“All problems in computer graphics can be solved with a matrix inversion.”

Jim Blinn

Inkrementális 3D képszintézis
1. Bevezetés

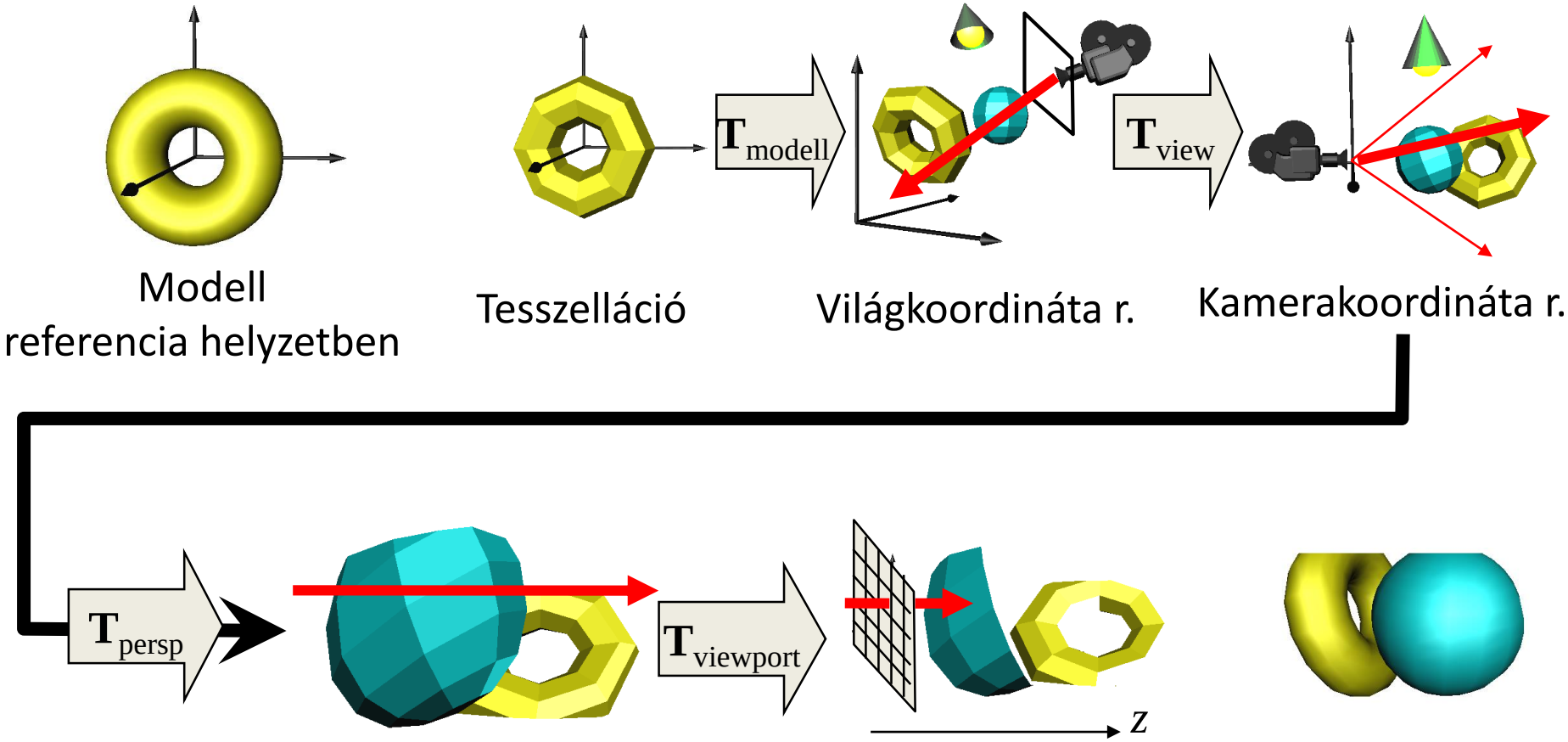
Szirmay-Kalos László

Inkrementális képszintézis

- Objektum vezérelt megközelítés
- Cél a sebesség és a hw támogathatóság

- koherencia: oldjuk meg nagyobb egységekre
- feleslegesen ne számoljunk: vágás
- transzformációk: mindenhez megfelelő koordinátarendszert
  - vágni, transzformálni nem lehet akármit: tesszelláció

3D inkrementális képszintézis



Modell
referencia helyzetben

Tesszelláció

Világkoordináta r.

Kamerakoordináta r.

Normalizált
eszközkoordináta r.
Vágás

Képernyőkoordináta r.
Láthatóság + vetítés
Takarás és vetítés triviális!

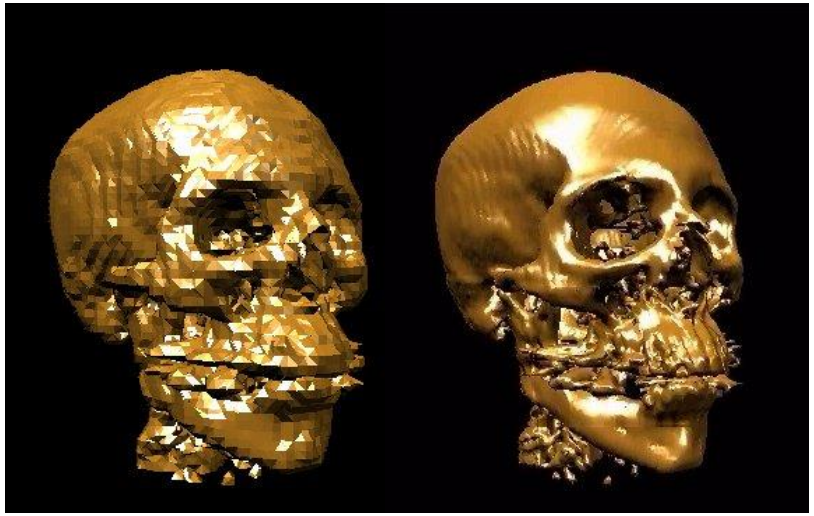
Árnyalt kép

“Order is repetition of units.
Chaos is multiplicity without rhythm.”

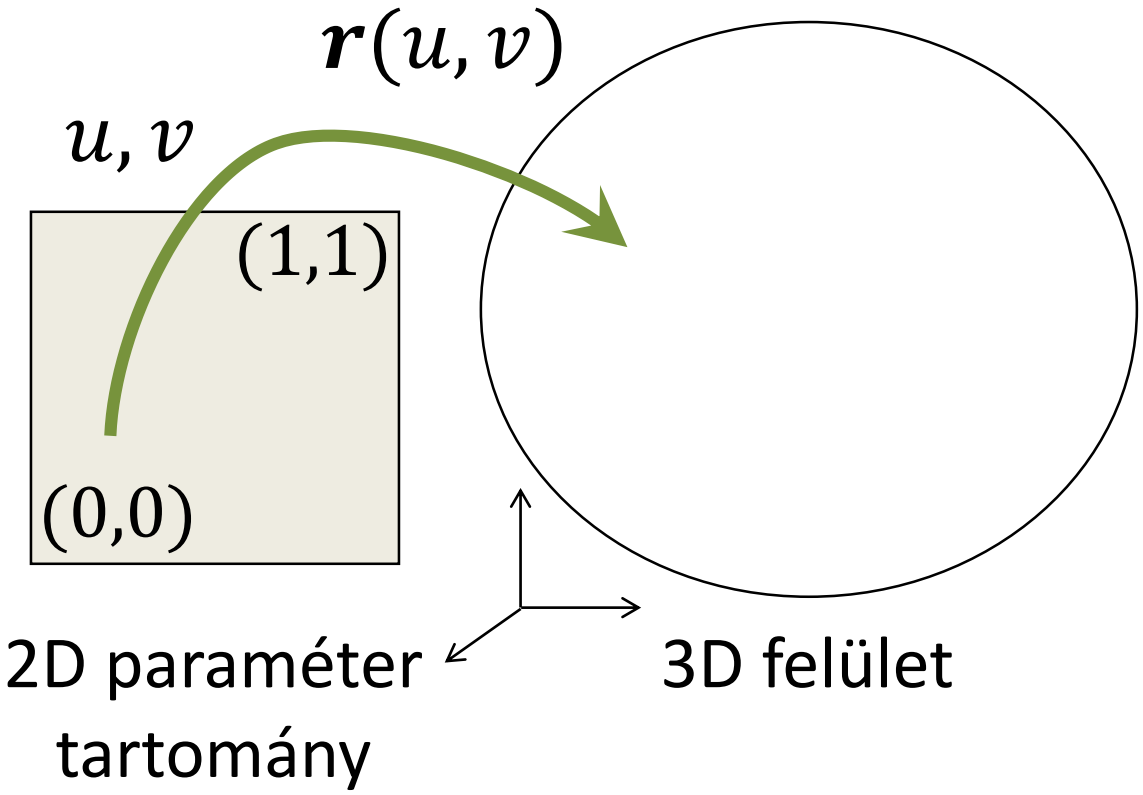
Maurits Cornelis Escher

Inkrementális 3D képszintézis
2. Geometria a GPU-nak

Szirmay-Kalos László

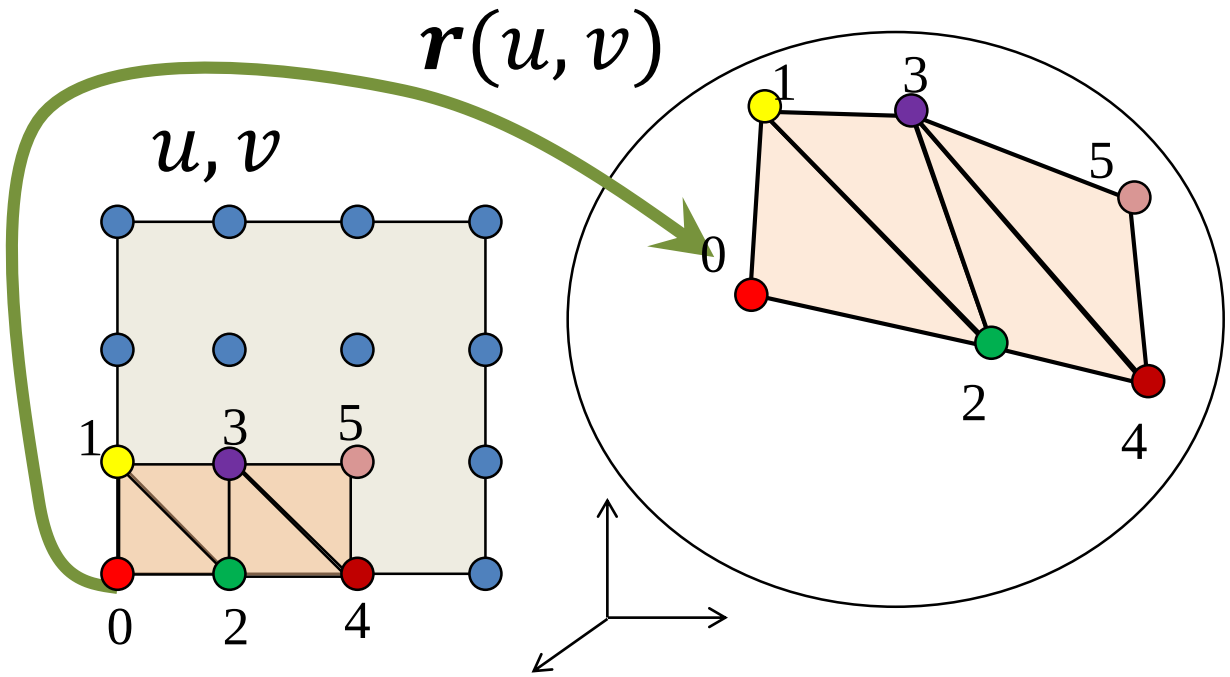


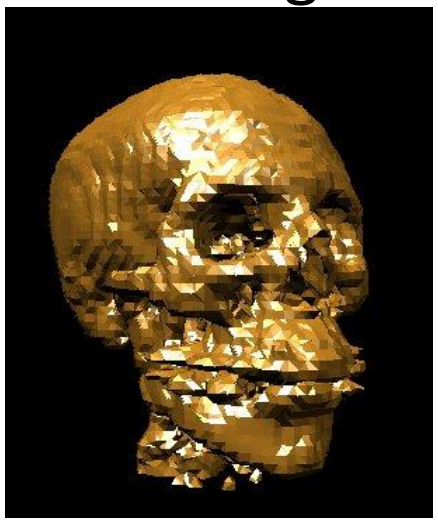
Parametrikus felületek tesszellációja



Parametrikus felületek tesszellációja

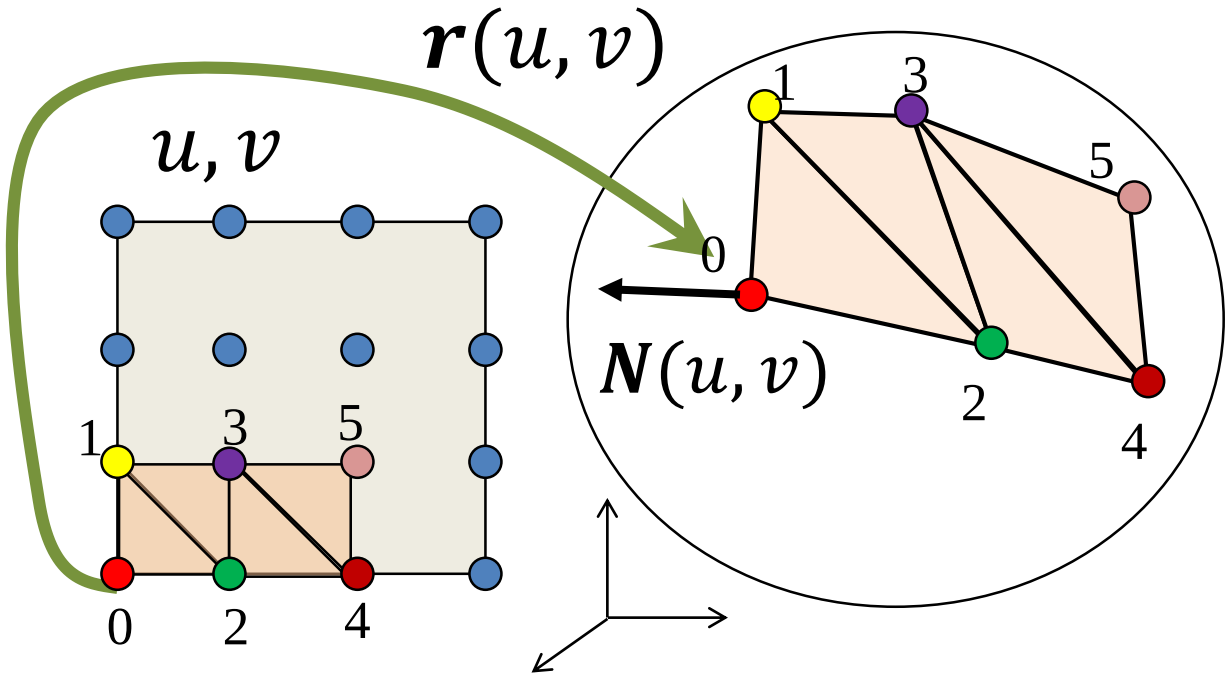
„Paramétertérben szomszédos” pontokból háromszögek

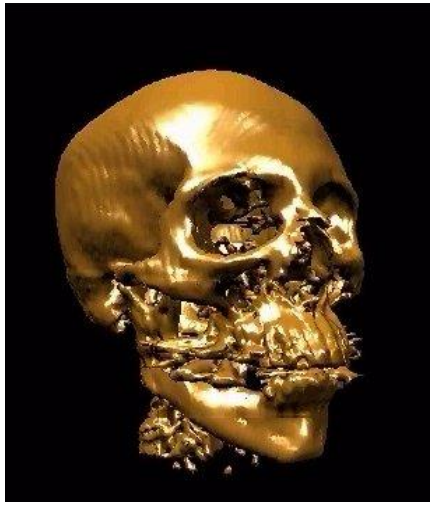




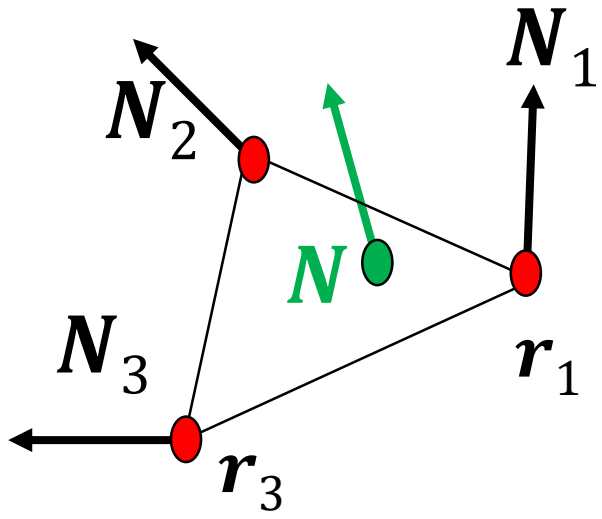
Parametrikus felületek tesszellációja

„Paramétertérben szomszédos” pontokból háromszögek





$$\boldsymbol{N} = \frac{\partial \boldsymbol{r}(u,v)}{\partial u} \times \frac{\partial \boldsymbol{r}(u,v)}{\partial v}$$



Árnyaló normálok

Geometry<T>, T=VtxData

```
struct VtxData {
   vec3 pos, normal;
   vec2 texcoord;
};
```

```
template<class T> class Geometry {
   unsigned int vao, vbo; // GPU
   vector<T> vtx;         // CPU
public:
   Geometry() {
      glGenVertexArrays(1, &vao); glBindVertexArray(vao);
      glGenBuffers(1, &vbo); glBindBuffer(GL_ARRAY_BUFFER, vbo); glEnableVertexAttribArray(0);
      int nf = sizeof(T)/sizeof(float);
      if (nf <= 4) glVertexAttribPointer(0, nf, GL_FLOAT, GL_FALSE, 0, NULL);
   }
   vector<T>& Vtx() { return vtx; }
   void updateGPU() {
      glBindBuffer(GL_ARRAY_BUFFER, vbo);
      glBufferData(GL_ARRAY_BUFFER, vtx.size() * sizeof(T), &vtx[0], GL_DYNAMIC_DRAW);
   }
   void Bind() { glBindVertexArray(vao); }
   virtual ~Geometry() { glDeleteBuffers(1, &vbo); glDeleteVertexArrays(1, &vao); }
};
```

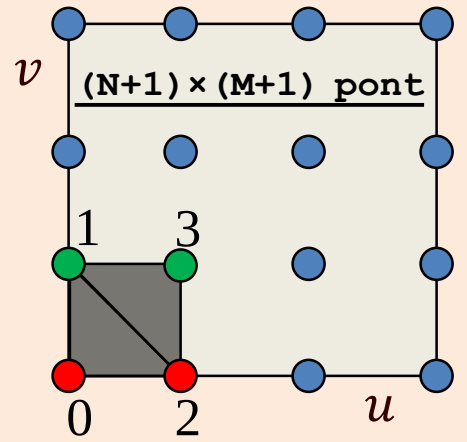
Parametrikus felületek

```
struct VtxData {
   vec3 pos, normal;
   vec2 texcoord;
};
```

pos.x
pos.y
pos.z
normal.x
normal.y
normal.z
texcoord.x
texcoord.y

pos.x
pos.y
pos.z
normal.x
normal.y
normal.z
texcoord.x
texcoord.y

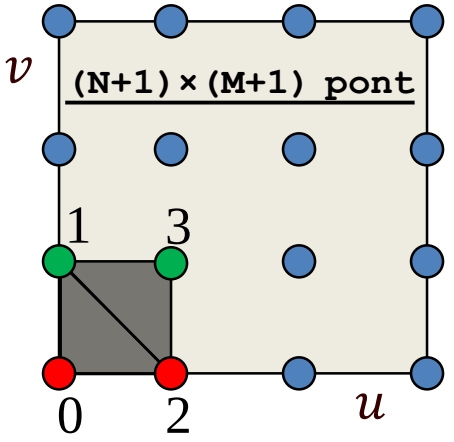
```
class ParamSurface : public  Geometry<VtxData> {
   int nVtxInStrip, nStrips;
public:
   ParamSurface() {
      glEnableVertexAttribArray(0);  // 0. regiszter = pozíció
      glEnableVertexAttribArray(1);  // 1. regiszter = normál vektor
      glEnableVertexAttribArray(2);  // 2. regiszter = textúra koordináta
      int nb = sizeof(VtxData);
      glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, nb, (void*)offsetof(VtxData, pos));
      glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, nb, (void*)offsetof(VtxData, normal));
      glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, nb, (void*)offsetof(VtxData, texcoord));
   }
   virtual VtxData GenVtxData(float u, float v) = 0;
   void create(int M, int N);
   void Draw() {
      Bind();
      for (unsigned int i = 0; i < nStrips; i++)
         glDrawArrays(GL_TRIANGLE_STRIP, i * nVtxInStrip, nVtxInStrip);
   }
};
```



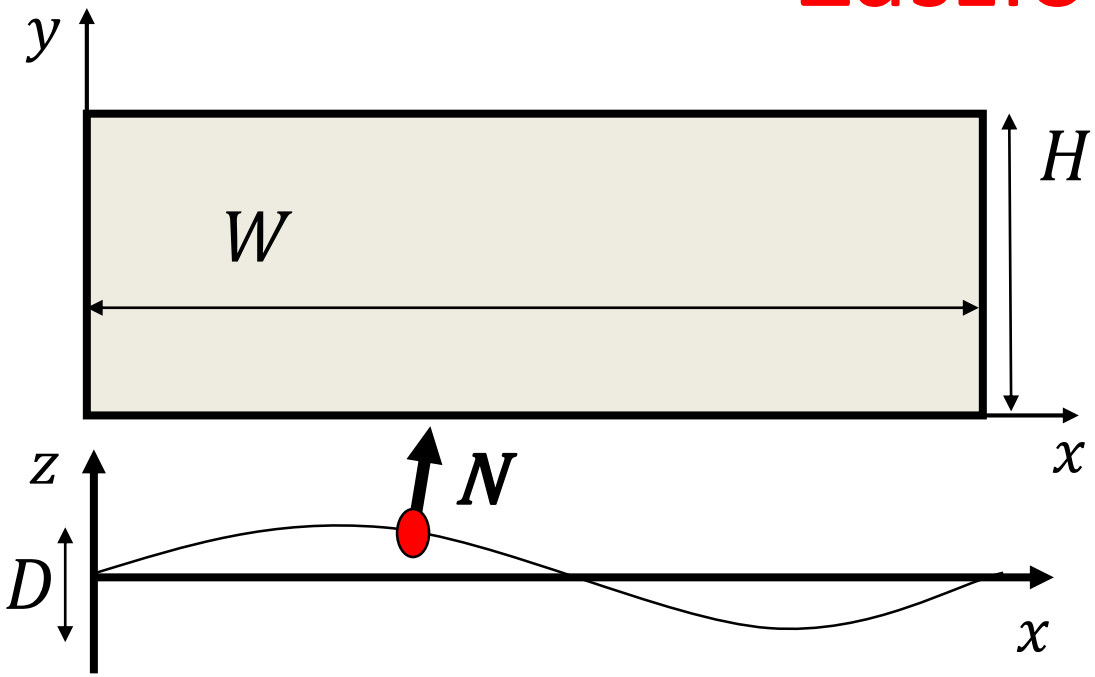
Parametrikus felület GPU-nak

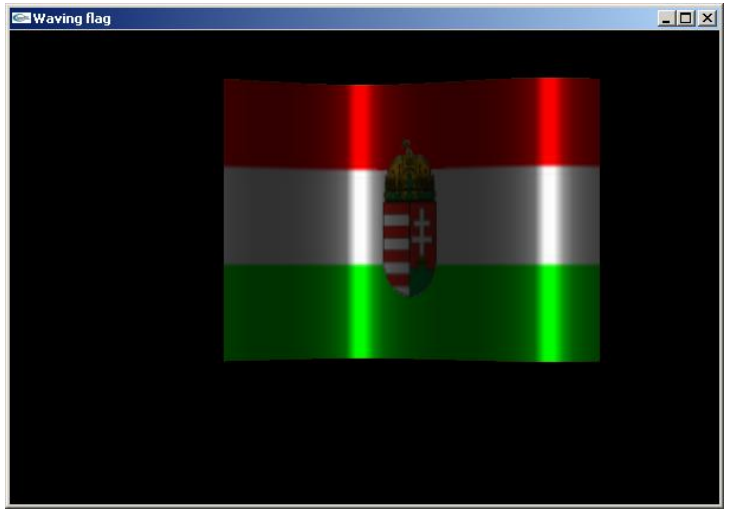
```
void ParamSurface :: create(int M, int N) {
    nVtxInStrip = (M + 1) * 2;
    nStrips = N;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j <= M; j++) {
            vtx.push_back(GenVtxData((float)j/M, (float)i/N));
            vtx.push_back(GenVtxData((float)j/M, (float)(i+1)/N));
        }
    }
    updateGPU();
}
```

u v



Zászló





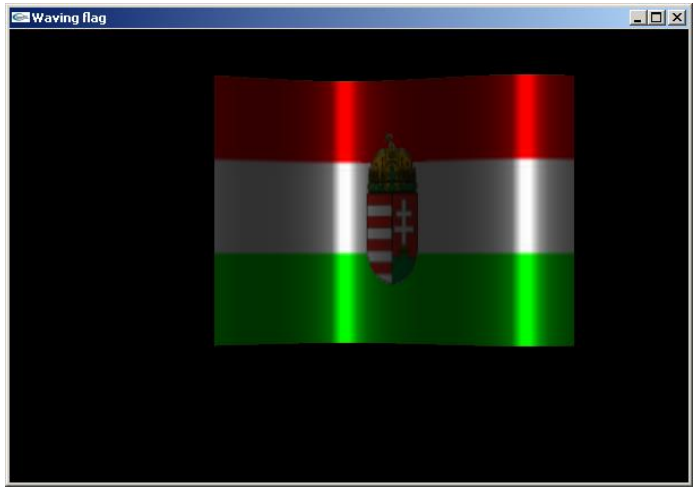
$$\boldsymbol{r}(u,v) = [uW, \quad vH, \quad D\sin(Ku+\alpha)]$$

$$\begin{matrix} & \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ W, & 0, & KD\cos(Ku+\alpha) \\ 0, & H, & 0 \end{vmatrix} \\ \partial\boldsymbol{r}/\partial u = & \\ \partial\boldsymbol{r}/\partial v = & \end{matrix}$$

$$\boldsymbol{N}(u,v) = \partial\boldsymbol{r}/\partial u \times \partial\boldsymbol{r}/\partial v = [-HKD\cos(Ku+\alpha), 0, WH]$$

Zászló

```
class Flag : public ParamSurface {
    float W, H, D, K, phase;
public:
    VtxData GenVtxData(float u, float v) override {
        VtxData vd;
        vd.texcoord = vec2(u, v);
        float angle = u * K + phase;
        vd.pos = vec3(u * W, v * H, D * sin(angle));
        vd.normal = vec3(-K * D * cos(angle), 0, W);
        return vd;
    }
};
```



Geometria a GPU-nak

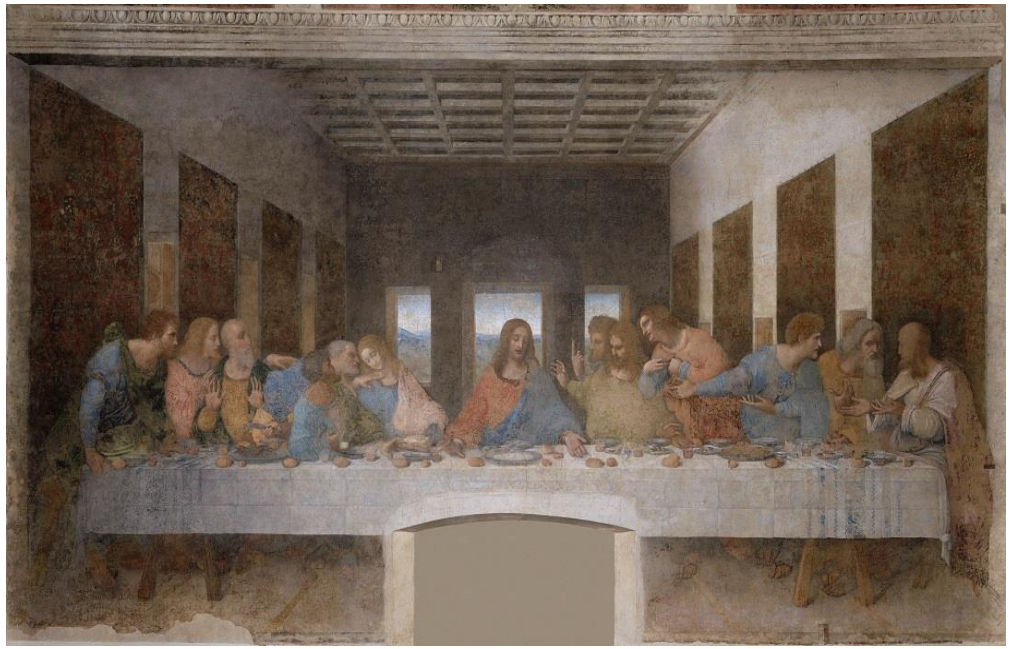
- A GPU egy VAO-ba szervezett VBO-kban háromszögeket vár, amit a CPU program legkönnyebben paraméteres felületek tesszellációjával készíthet el.
- Elsődlegesen a paraméterteret tesszelláljuk.
- A csúcspontokhoz árnyaló normálisok és textúra koordináták is tartozhatnak.
- A normálvektor a parciális deriváltak vektoriális szorzata (lásd: Felület modellezés és Automatikus deriválás fejezetek).
- A tesszellált háromszögháló kompakt tárolásához találták ki a GL_TRIANGLE_STRIP-et.

“The Matrix is everywhere. It is all around us. A prison for your mind.”

Morpheus

Inkrementális 3D képszintézis
3. Transzformációk

Szirmay-Kalos László



Transzformációk

Modellezési transzformáció:

$$\boldsymbol{T}_{Model}\begin{bmatrix}\boldsymbol{r}\\1\end{bmatrix}=\begin{bmatrix}\boldsymbol{r}_{world}\\1\end{bmatrix}$$

$$[\boldsymbol{N},0]\boldsymbol{T}_{Model}^{-1}=[\boldsymbol{N}_{world},d]$$

Kamera transzformáció:

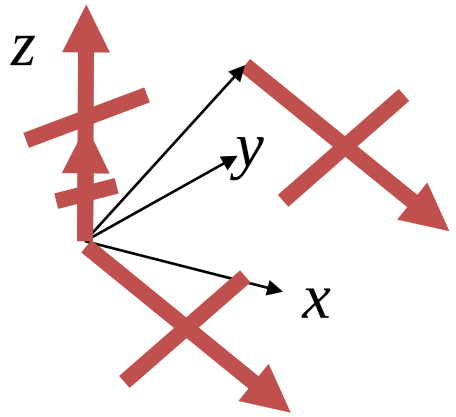
$$\boldsymbol{T}_{View}\begin{bmatrix}\boldsymbol{r}_{world}\\1\end{bmatrix}=\begin{bmatrix}\boldsymbol{r}_{camera}\\1\end{bmatrix}$$

Perspektív transzformáció:

$$\boldsymbol{T}_{Proj}\begin{bmatrix}\boldsymbol{r}_{camera}\\1\end{bmatrix}=\begin{bmatrix}\boldsymbol{r}_{ndc}w\\w\end{bmatrix}$$

MVP transzformáció: $\boldsymbol{T}_{Proj}\boldsymbol{T}_{View}\boldsymbol{T}_{Model}=\boldsymbol{T}_{MVP}$

Modellezési transzformáció



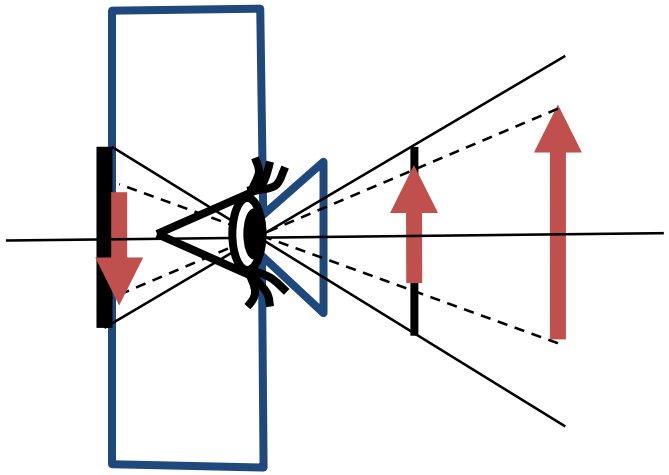
1. skálázás: s_x, s_y, s_z
2. orientáció: $\boldsymbol{d}, \varphi$
3. pozíció: $\boldsymbol{v}$

$$\boldsymbol{T}_{4\times4} = \begin{bmatrix} 1 & 0 & 0 & v_x \\ 0 & 1 & 0 & v_y \\ 0 & 0 & 1 & v_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{i'}_x & \boldsymbol{j'}_x & \boldsymbol{k'}_x & 0 \\ \boldsymbol{i'}_y & \boldsymbol{j'}_y & \boldsymbol{k'}_y & 0 \\ \boldsymbol{i'}_z & \boldsymbol{j'}_y & \boldsymbol{k'}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{r'} = \boldsymbol{r}\cos(\varphi) + \boldsymbol{d}(\boldsymbol{r}\cdot\boldsymbol{d})(1-\cos(\varphi)) + \boldsymbol{d}\times\boldsymbol{r}\sin(\varphi)$$

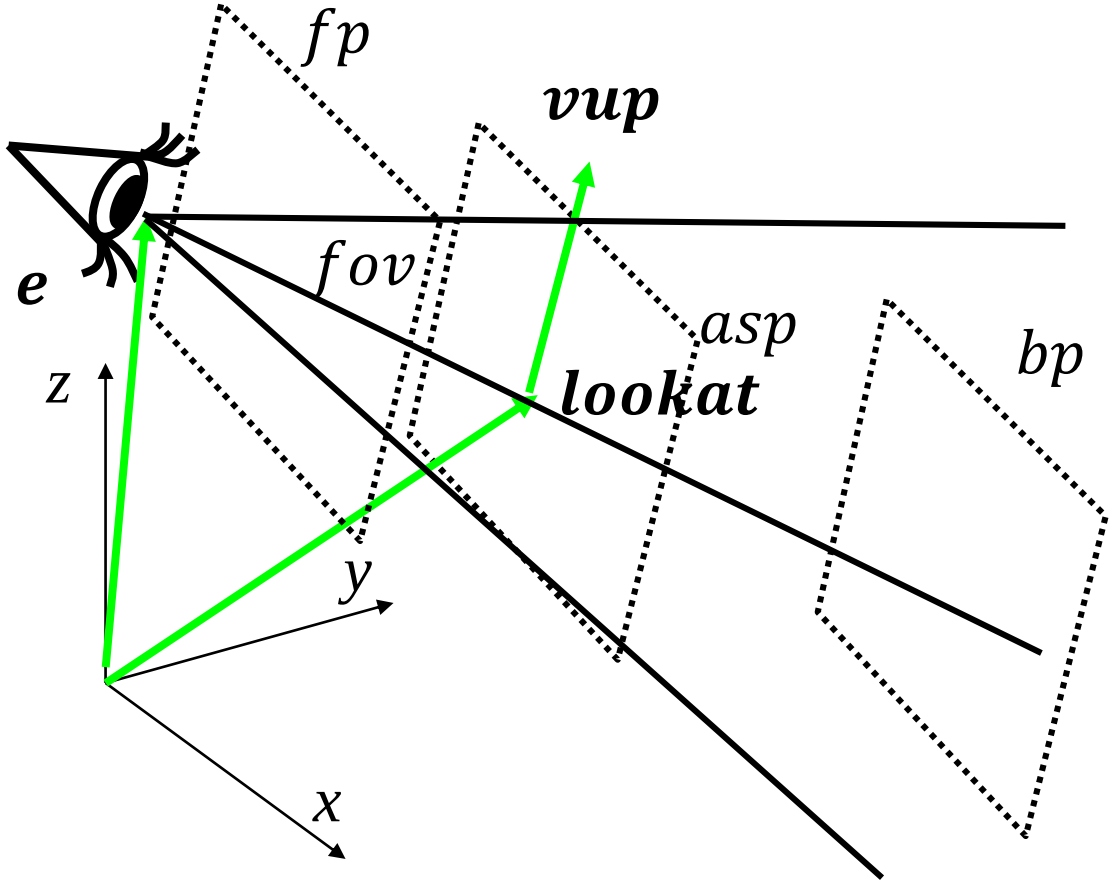
szem

Nézeti téglalap

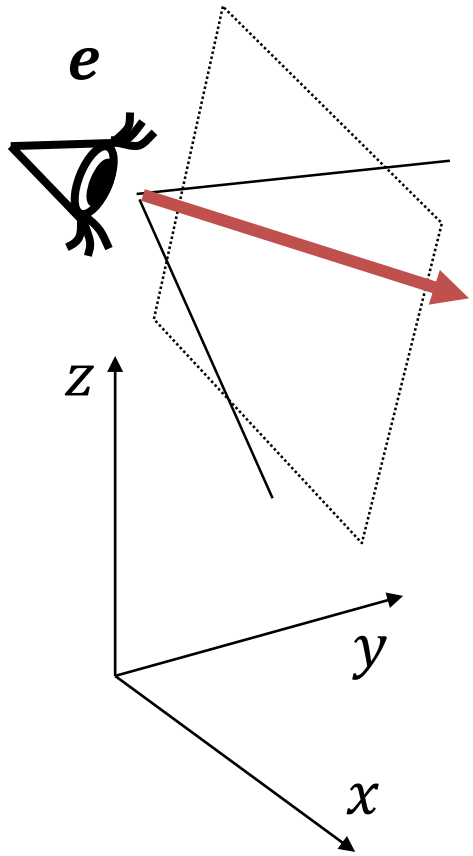


Mi: Camera obscura

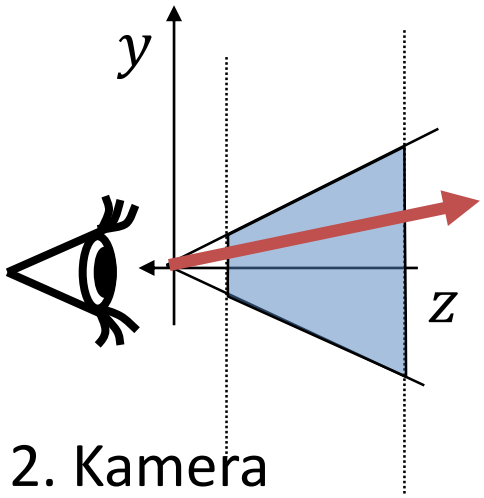
Kamera modell



Világból a képernyőre

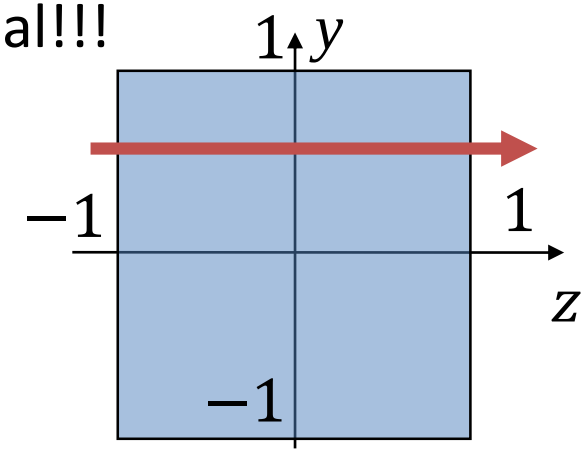


1. Világ

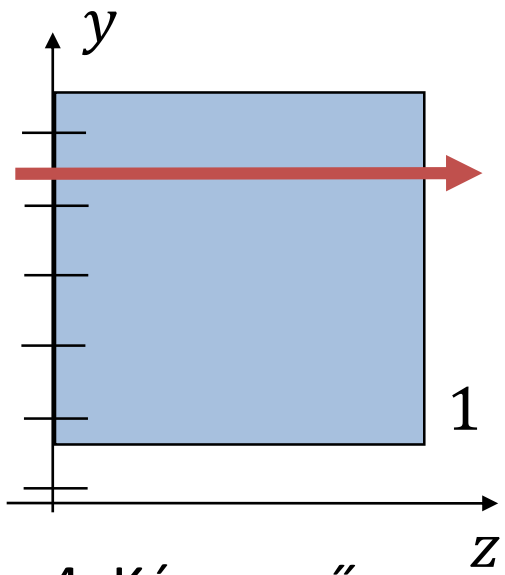


2. Kamera

Bal!!!

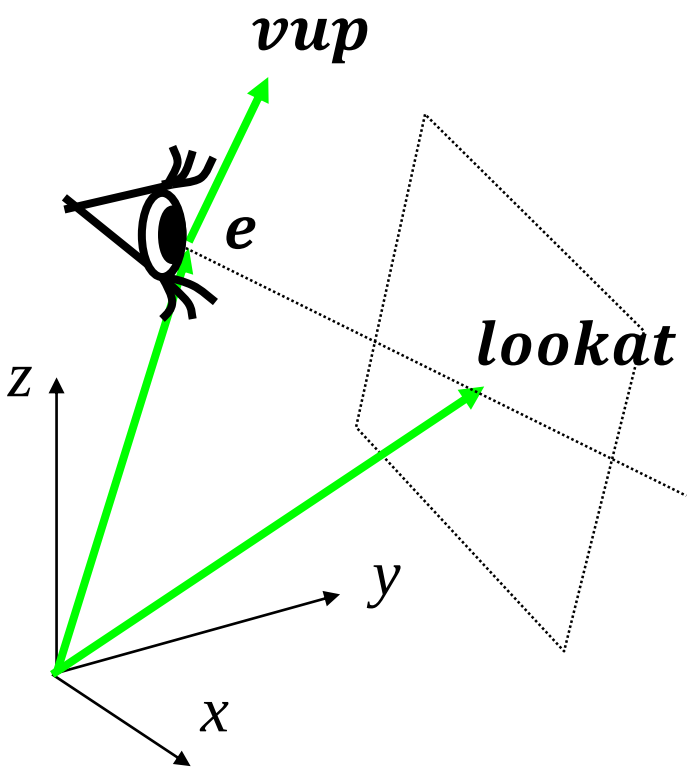


3. Normalizált képernyő



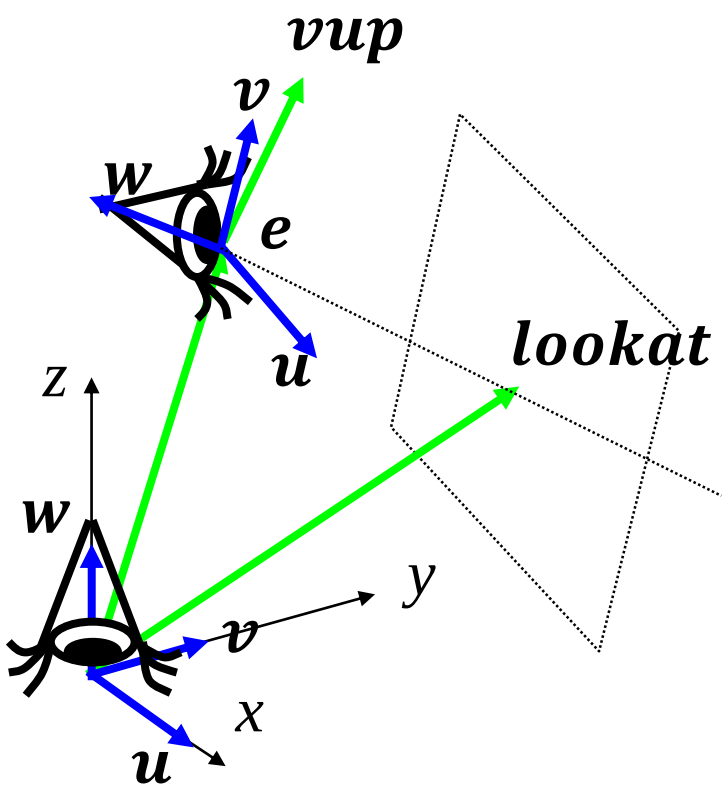
4. Képernyő

View transzformáció



$$\boldsymbol{w} = (\boldsymbol{e} - \boldsymbol{lookat})/|\boldsymbol{e} - \boldsymbol{lookat}|$$
$$\boldsymbol{u} = (\boldsymbol{vup} \times \boldsymbol{w})/|\boldsymbol{w} \times \boldsymbol{vup}|$$
$$\boldsymbol{v} = \boldsymbol{w} \times \boldsymbol{u}$$

View transzformáció

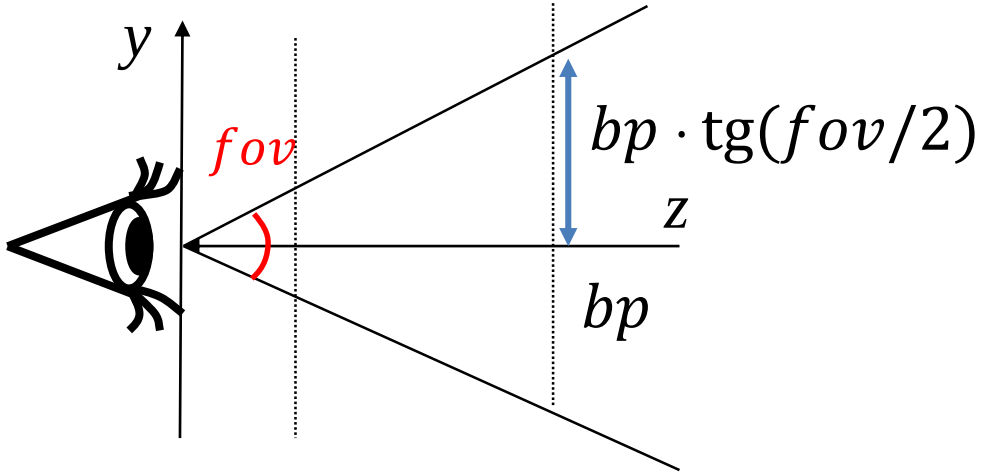


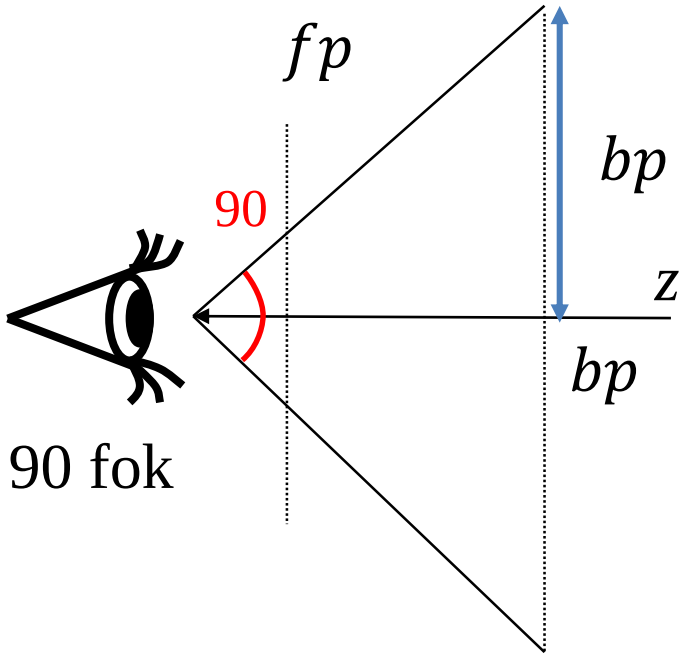
$$\boldsymbol{w} = (\boldsymbol{e} - \boldsymbol{lookat})/|\boldsymbol{e} - \boldsymbol{lookat}|$$
$$\boldsymbol{u} = (\boldsymbol{vup} \times \boldsymbol{w})/|\boldsymbol{w} \times \boldsymbol{vup}|$$
$$\boldsymbol{v} = \boldsymbol{w} \times \boldsymbol{u}$$

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Látószög normalizálás





$$\begin{bmatrix} 1/(\mathrm{tg}(fov/2)\,asp) & 0 & 0 & 0 \\ 0 & 1/\mathrm{tg}(fov/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

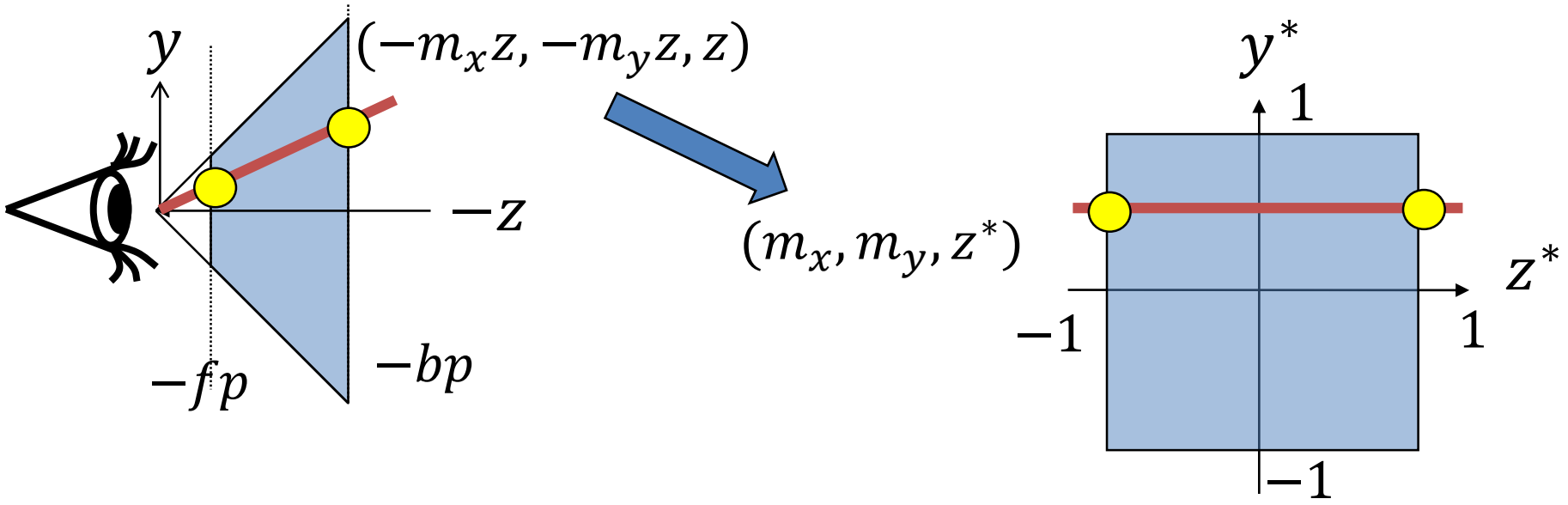
2D egyenes explicit egyenlete:

$$y = mx + b$$

Origón átmenő: $y = mx$

Vízszintes: $y = b$

Normalizálás utáni perspektív transzformáció



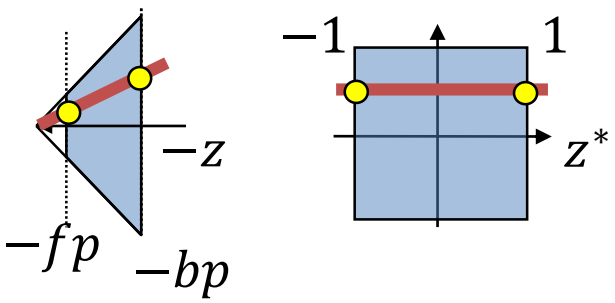
Normalizált kamera

Normalizált képernyő: NDC

$$(-m_x z, -m_y z, z) \rightarrow (m_x, m_y, z^*)$$

$$[-m_x z, -m_y z, z, 1] \rightarrow [m_x, m_y, z^*, 1] \sim [-m_x z, -m_y z, -z z^*, -z]$$

Perspektív transzformáció



$$\begin{bmatrix} -m_x z \\ -m_y z \\ -zz^* \\ -z \end{bmatrix} = \boldsymbol{T}_{Persp} \begin{bmatrix} -m_x z \\ -m_y z \\ z \\ 1 \end{bmatrix}$$

$$\boldsymbol{T}_{Persp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

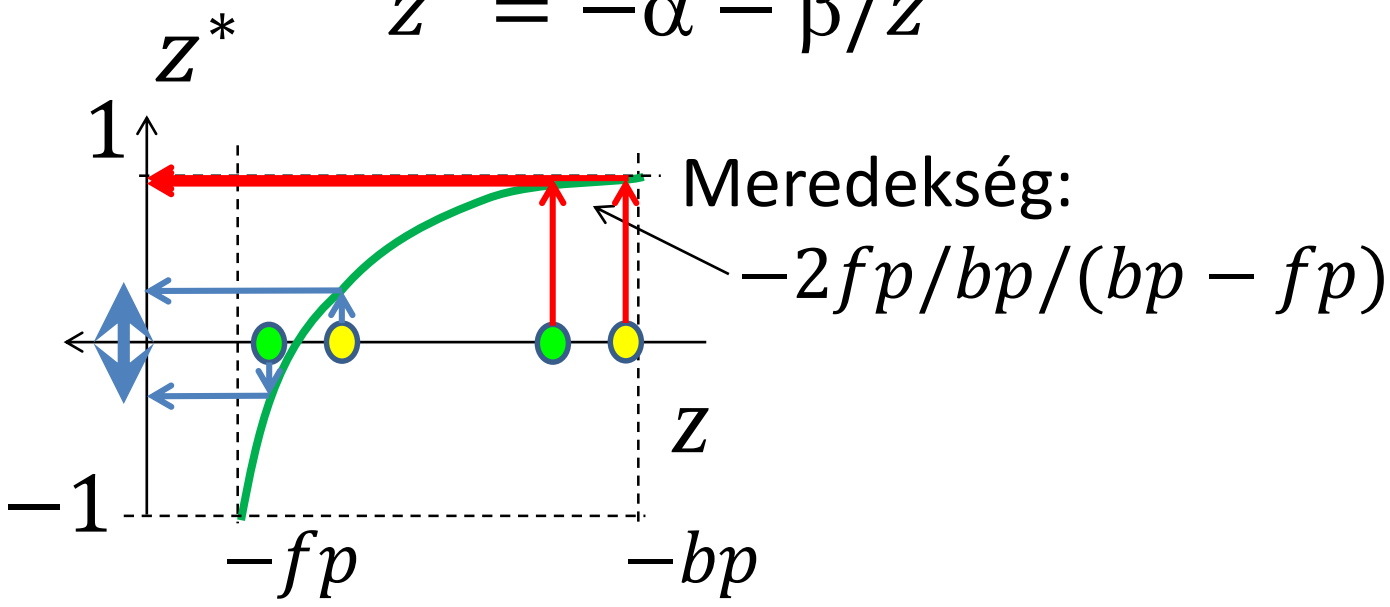
$$-zz^* = \alpha z + \beta \quad \rightarrow \quad \boxed{z^* = -\alpha - \beta/z}$$

$$fp(-1) = \alpha(-fp) + \beta$$
$$bp(1) = \alpha(-bp) + \beta$$

$$\boxed{\begin{aligned} \alpha &= -(fp + bp)/(bp - fp) \\ \beta &= -2fp \cdot bp/(bp - fp) \end{aligned}}$$

Z-fighting

$$z^* = -\alpha - \beta/z$$

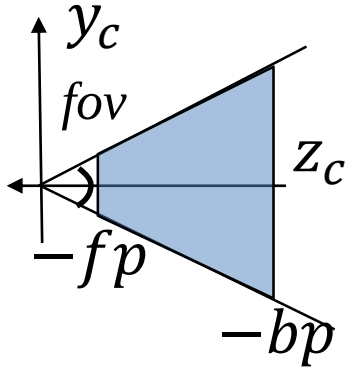


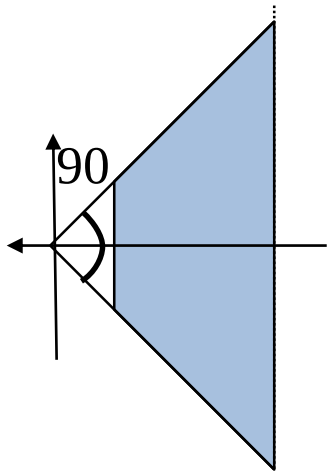
Meredekség:

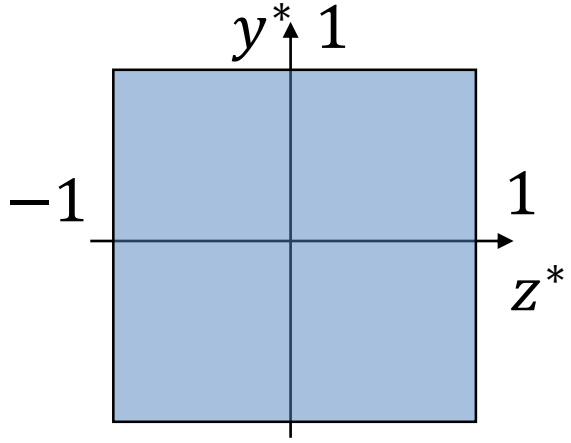
$$-2fp/bp/(bp - fp)$$

fp/bp nem lehet kicsi!

Perspektív transzformáció







$$\boldsymbol{T}_{Proj}$$

$$\begin{bmatrix} 1/(\mathrm{tg}(fov/2)\,asp) & 0 & 0 & 0 \\ 0 & 1/\mathrm{tg}(fov/2) & 0 & 0 \\ 0 & 0 & -(fp+bp)/(bp-fp) & -2fp\cdot bp/(bp-fp) \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}$$

Perspektív torzítás = homogén osztás

$$(x^*, y^*, z^*) = (X/w, Y/w, Z/w)$$

$$w = -z_c$$

Camera osztály

```
struct Camera { // 3D kamera
   vec3 wEye, wLookat, wVup;   // külső paraméterek
   float fov, asp, fp, bp;     // belső paraméterek
public:
   Camera() {
      asp = (float)winWidth / winHeight;  // aspektus arány
      fov = 40.0f * (float)M_PI / 180.0f; // függőleges látószög
      fp = 1; bp = 20; // első és hátsó vágósík távolság
   }
   // Nézeti transzformáció
   mat4 V() { return lookAt(wEye, wLookat, wVup); }
   // Perspektív transzformáció
   mat4 P() { return perspective(fov, asp, fp, bp); }
};
```

Transzformációk előkészítése a CPU-n

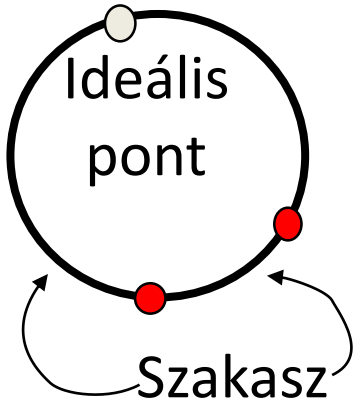
```
void Draw(GPUProgram* shader) {
   mat4 M = translate(translation) *
            rotate(rotAngle, rotAxis) *
            scale(scaling);
   shader->setUniform(M, "M");
   mat4 Minv = scale(1.0f / scaling) *
               rotate(-rotAngle, rotAxis) *
               translate(-translation);
   shader->setUniform(M, "Minv");
   shader->setUniform(camera.P() * camera.V() * M, "MVP");

   Rajzolás
}
```

Transzformációk végrehajtása a GPU vertex árnyalójában

```
uniform mat4 M, Minv, MVP;
layout(location = 0) in vec3 vtxPos;
layout(location = 1) in vec3 vtxNorm;

out vec4 color;

void main() {
   gl_Position = MVP * vec4(vtxPos, 1);

   vec4 wPos = M * vec4(vtxPos, 1);
   vec4 wNormal = vec4(vtxNorm, 0) * Minv;
   color = Illumination(wPos, wNormal);
}
```

3D vágás homogén koordinátákban (GPU)

Homogén koordinátákban kell vágni



$$[X(t), Y(t), Z(t), w(t)] = [X_1, Y_1, Z_1, w_1](1-t) + [X_2, Y_2, Z_2, w_2]t$$

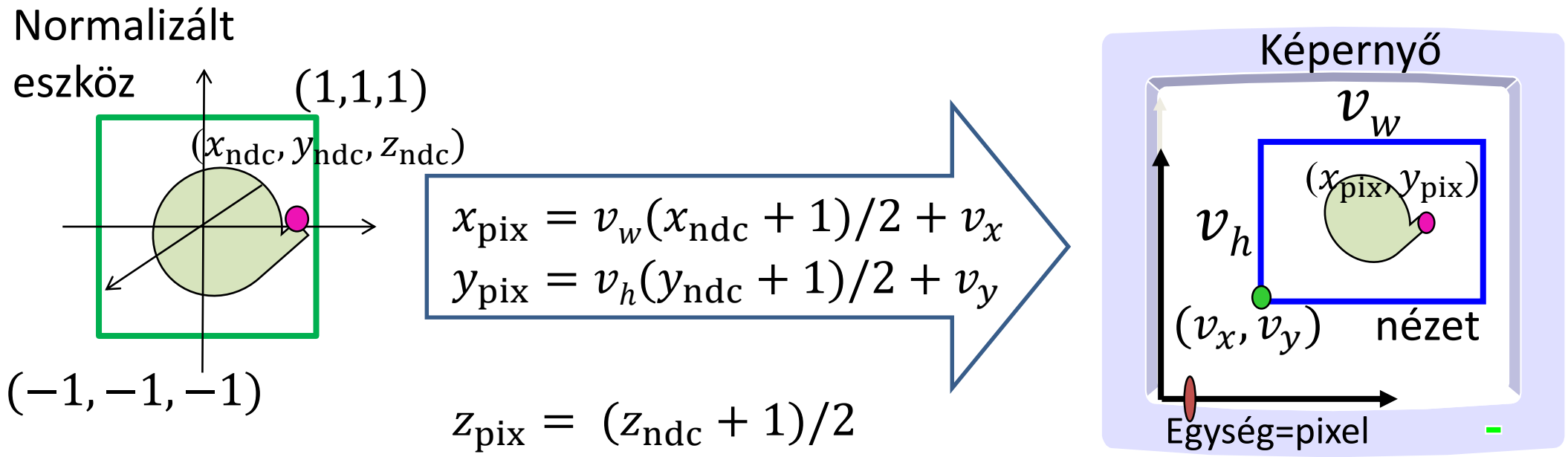
$$\begin{aligned} -1 < x = X/w < 1 \\ -1 < y = Y/w < 1 \\ -1 < z = Z/w < 1 \end{aligned}$$

$w > 0$

$$\begin{aligned} -w < X < w \\ -w < Y < w \\ -w < Z < w \end{aligned}$$

Mert $w = -z_c > 0$ a szem előtt

Viewport transzformáció: Normalizáltból képernyő koordinátákba (GPU)




```
glViewport(vx, vy, vw, vh)
```

Transzformációs csővezeték

- Transzformációk 4x4-es mátrixok szorzata:
  - Modell, Modell-inverz, MVP
- A CPU-n az egyes transzformációkat külön-külön számítjuk ki, majd a szorzatot adjuk át a csúcspont árnyalónak.
- A transzformációk homogén koordinátákat transzformálnak, a vágást is homogén koordinátákban kell megcsinálni.
- A vágás után visszatérhetünk Descartes koordinátákhoz.

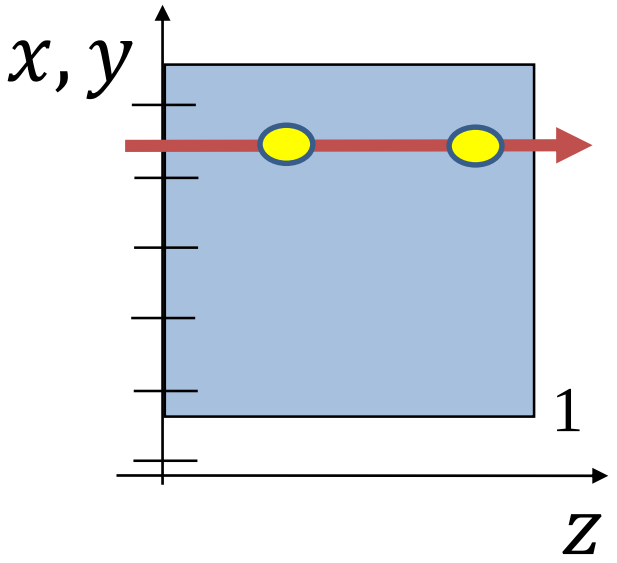
“If you can't make it good, at least make it look good.”

Bill Gates

Inkrementális 3D képszintézis
4. Láthatóság és árnyalás

Szirmay-Kalos László

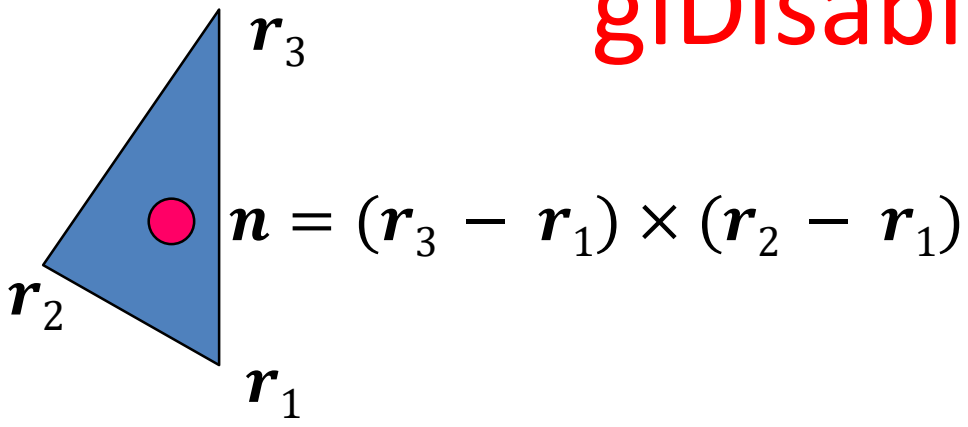
Takarás

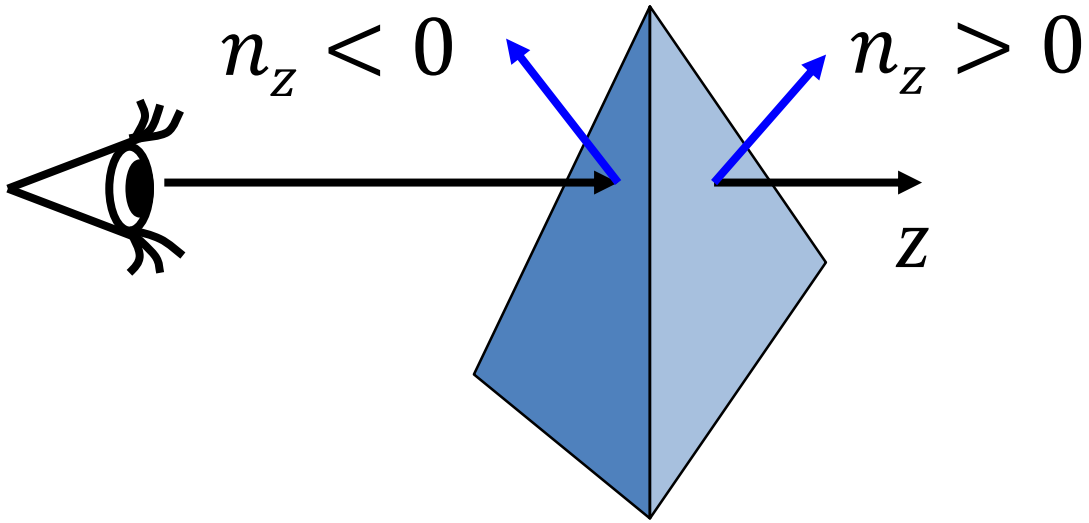


- Képernyő koordinátarendszerben
  - **<u>vetítősugarak a z tengellyel párhuzamosak</u>**!
  - Sugárparaméter = z koordináta
  - (x, y, z) pont az (x, y) pixelben látszik

- Objektumtér algoritmusok (folytonos):
  - láthatóság számítás nem függ a felbontástól
- Képtér algoritmusok (diszkrét):
  - Mi látszik egy pixelben?
  - Sugárkövetés ilyen volt!

Hátsólab eldobás: back-face culling
glDisable(GL_CULL_FACE)





Valódi 3D testek!

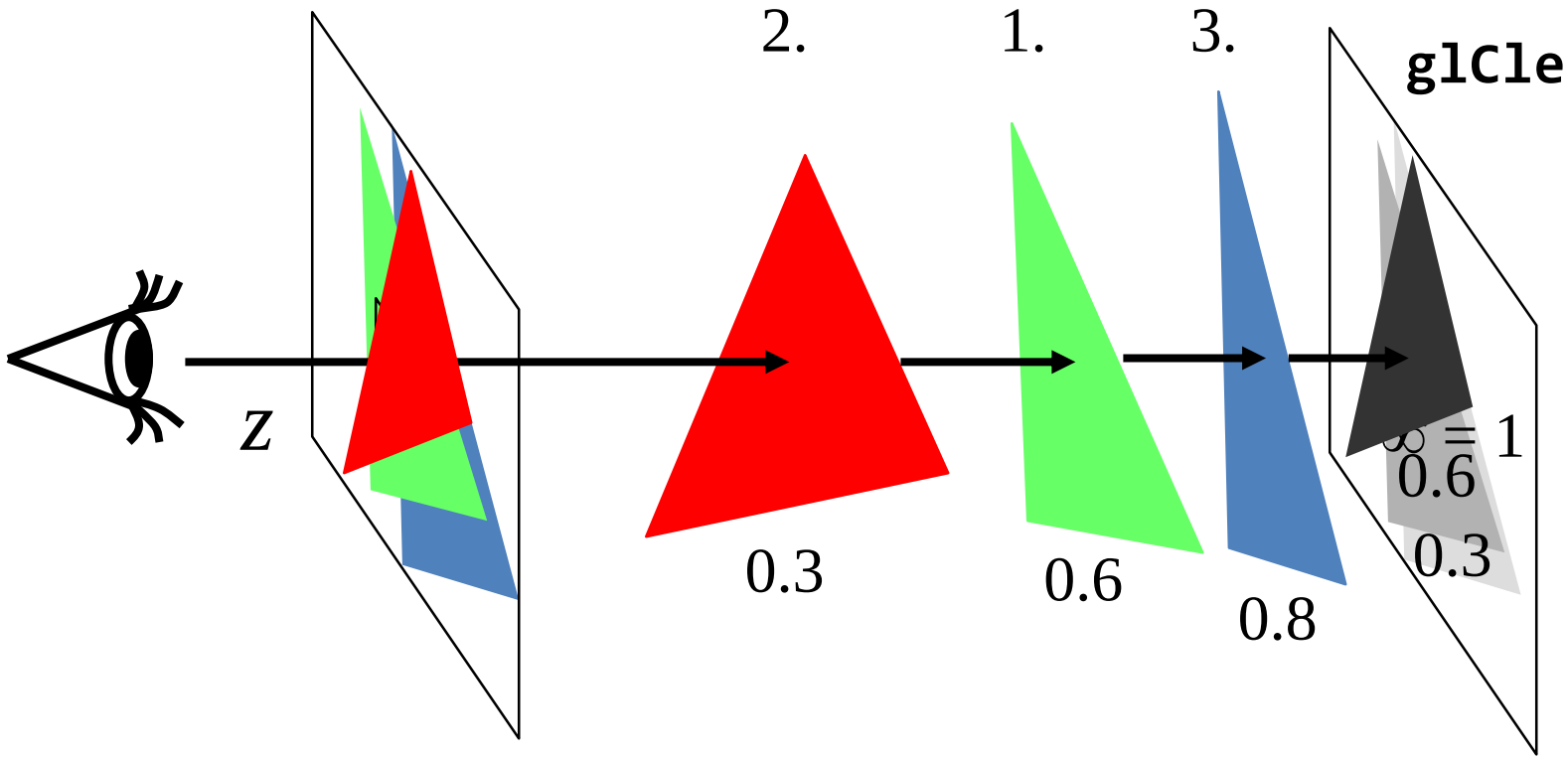
Lapok a nézeti irányban:

- Kívülről: lap, objektum: elülső oldal
- Belülről: objektum, lap: hátsó oldal

Feltételezés:

Ha kívülről, akkor csúcsok óramutatóval megegyező körüljárásúak

Z-buffer algoritmus: glEnable(GL_DEPTH_TEST)



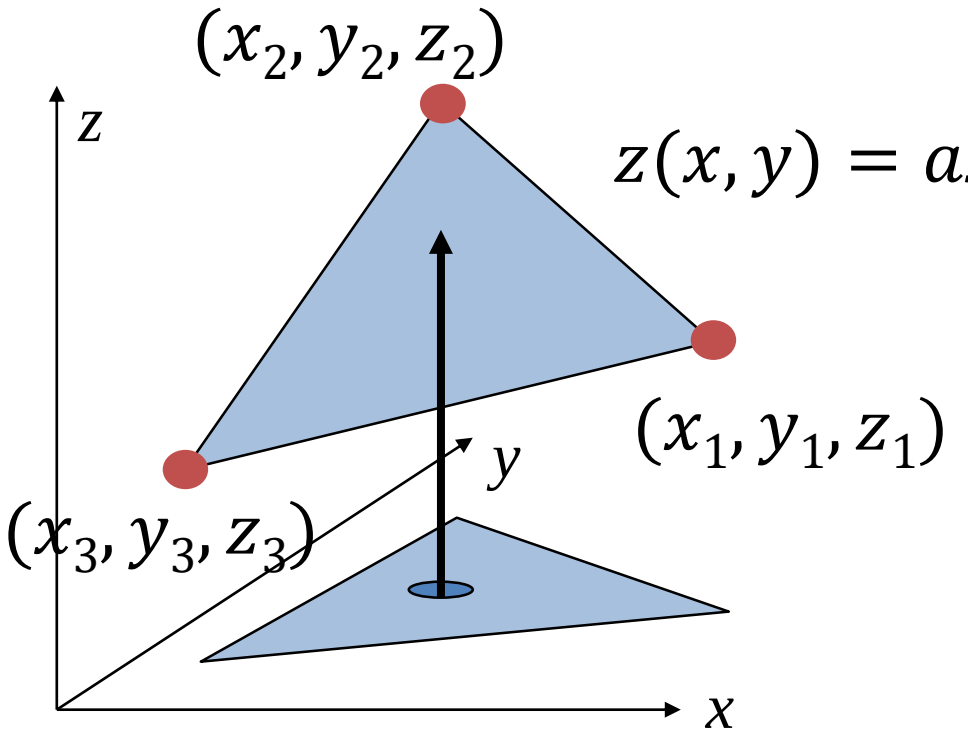

```
glClear(GL_COLOR_BUFFER_BIT |
        GL_DEPTH_BUFFER_BIT);
```

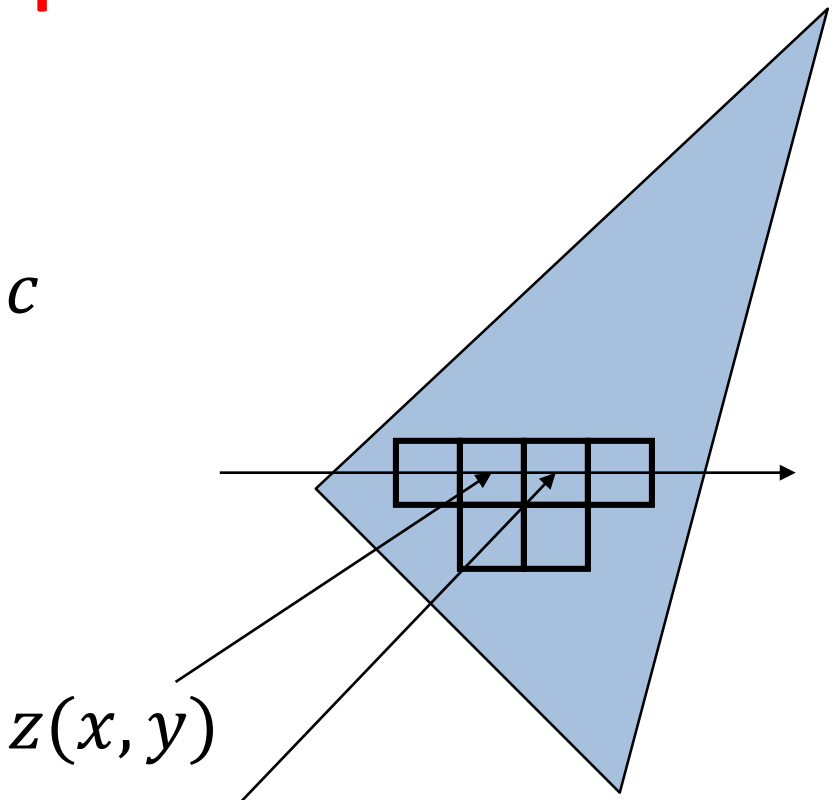
Szín buffer

Mélység buffer (Z-buffer)

= sugárparaméterek pixelenként

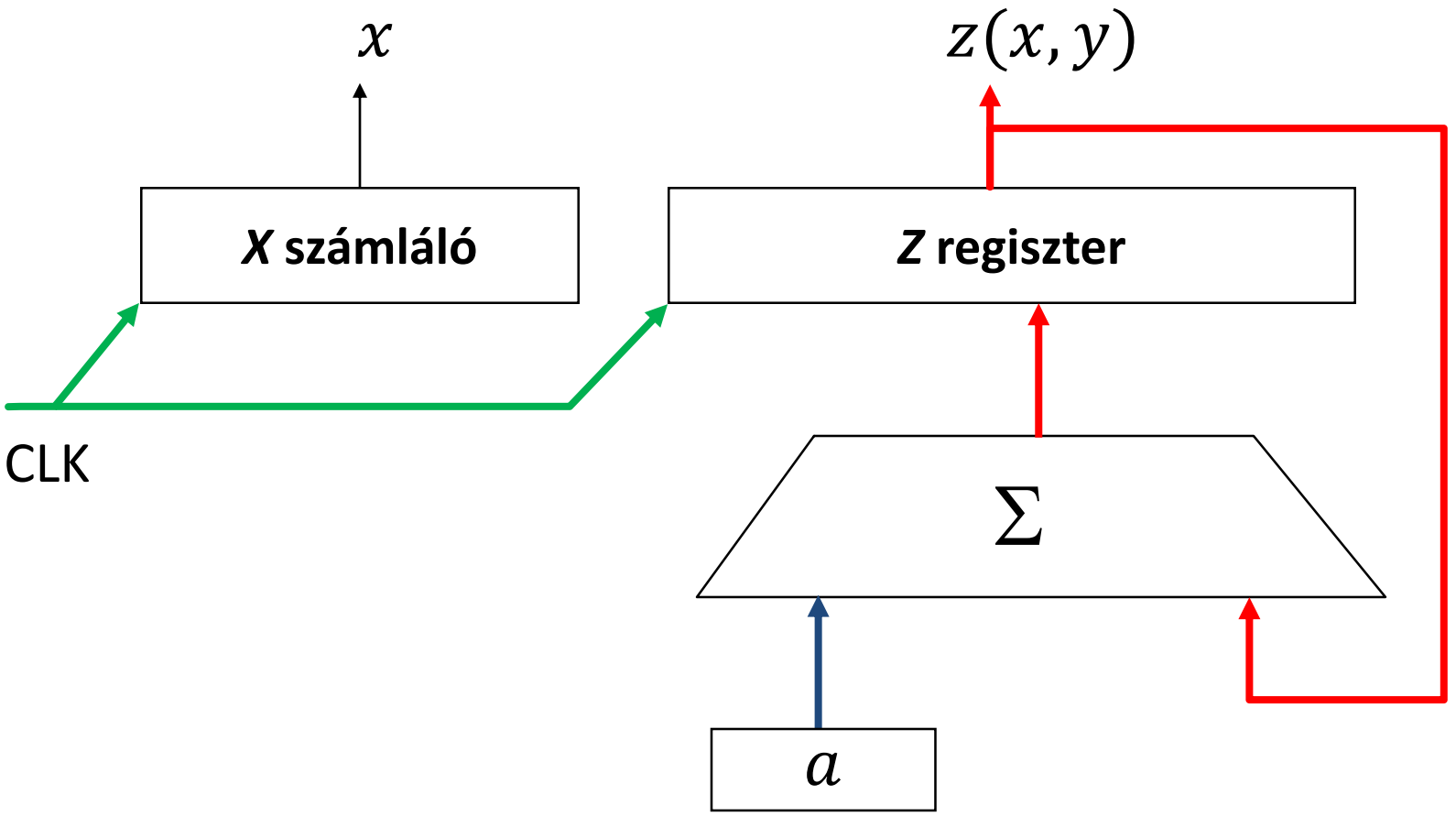
Z: lineáris interpoláció



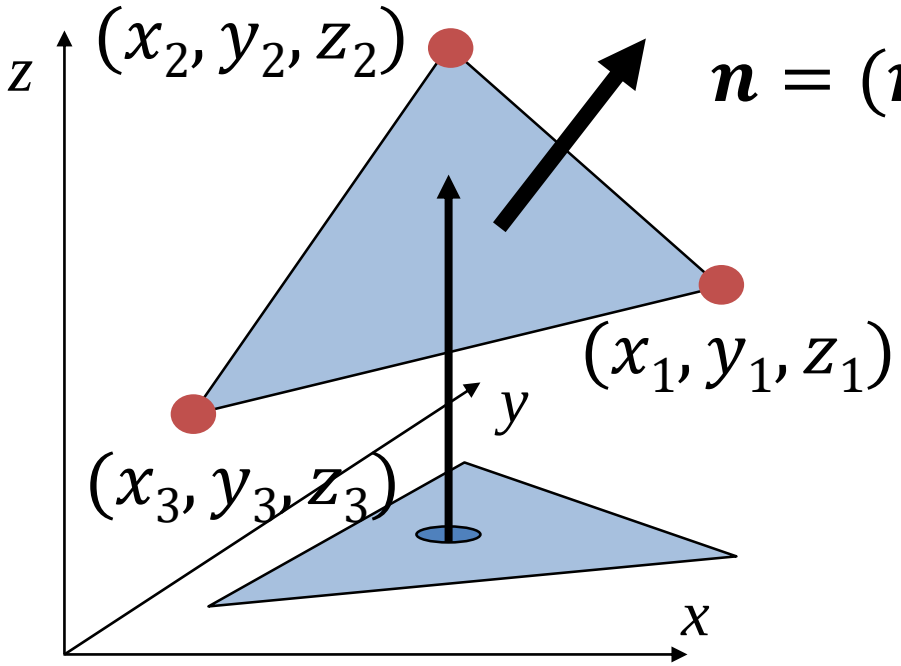


$$z(x + 1, y) = z(x, y) + a$$

Z-interpolációs hardver



Triangle setup



$$\boldsymbol{n} = (\boldsymbol{r}_3 - \boldsymbol{r}_1) \times (\boldsymbol{r}_2 - \boldsymbol{r}_1)$$

$$z(x, y) = ax + by + c$$
$$n_x x + n_y y + n_z z + d = 0$$

$$a = \frac{-n_x}{n_z}$$

Árnyalás: Lokális illumináció árnyék nélkül

$$L(\boldsymbol{V}) \approx \sum_l L^{in}{}_l \ * f_r(\boldsymbol{L}_l, \boldsymbol{N}, \boldsymbol{V}) \cdot \cos^{+}\theta^{in}{}_l$$

- Koherencia: ne mindent pixelenként

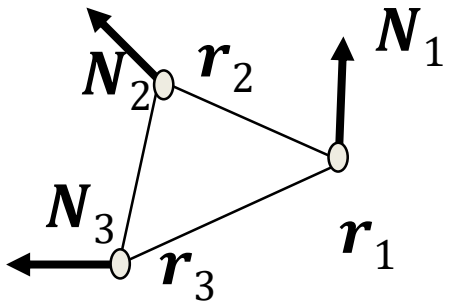
- **Csúcspontonként**: belül az *L* „szín” interpolációja:

  Gouraud árnyalás (per-vertex shading)

- **Pixelenként:** belül a **N**ormál (**V**iew, **L**ight) vektort interpoláljuk:

  Phong árnyalás (per-pixel shading)

Per-vertex (Gouraud) árnyalás

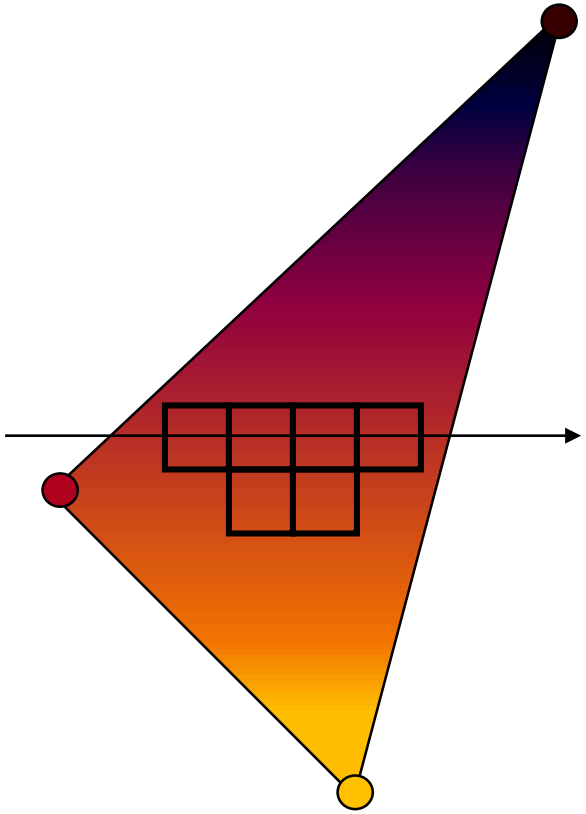


M: Modell

$$L(\boldsymbol{V}) \approx \sum_l {L^{in}}_l * f_r(\boldsymbol{L}_l, \boldsymbol{N}, \boldsymbol{V}) \cdot \cos^+ \theta^{in}{}_l$$

(per-vertex shading)

VP +
viewport



Képernyő

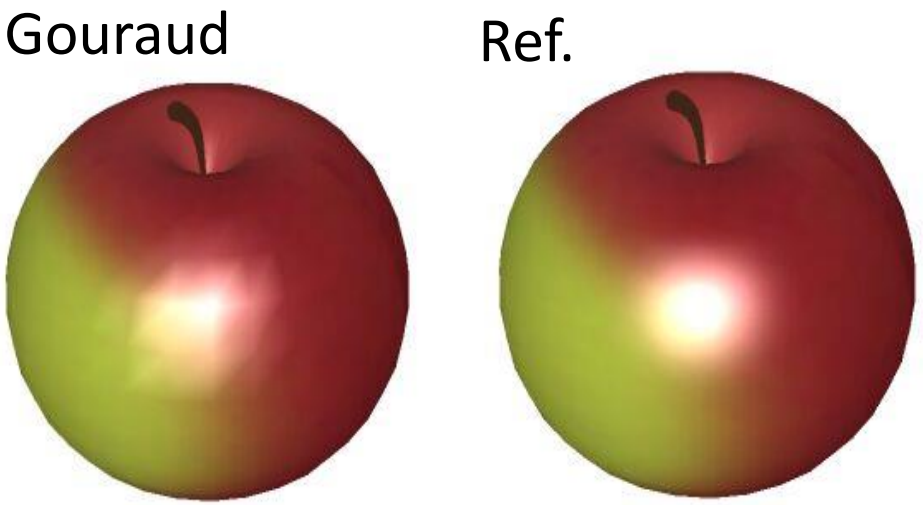
Per-vertex shading: Vertex shader

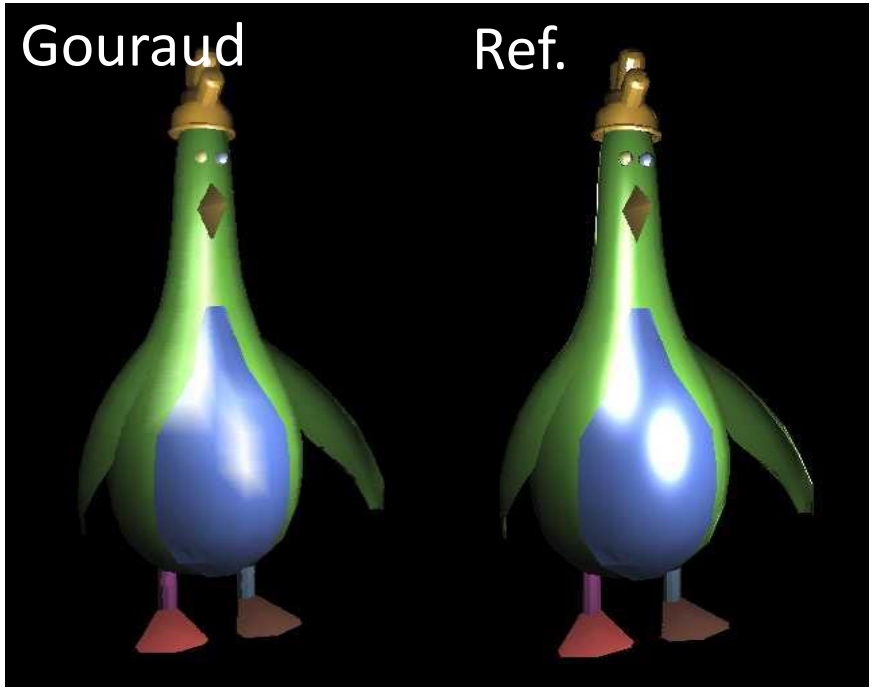
```
uniform mat4  MVP, M, Minv; // MVP, Model, Model-inverse
uniform vec4  kd, ks, ka;   // diffuse, specular, ambient ref
uniform float shine;        // shininess for specular ref
uniform vec4  La, Le;       // ambient and point sources
uniform vec4  wLiPos;       // pos of light source in world
uniform vec3  wEye;         // pos of eye in world

layout(location = 0) in vec3 vtxPos;  // pos in modeling space
layout(location = 1) in vec3 vtxNorm; // normal in modeling space
out vec4 color;               // computed vertex color

void main() {
   gl_Position = MVP * vec4(vtxPos, 1); // to NDC

   vec4 wPos = M * vec4(vtxPos, 1);
   vec3 L = normalize( wLiPos.xyz/wLiPos.w - wPos.xyz/wPos.w);
   vec3 V = normalize(wEye - wPos.xyz/wPos.w);
   vec4 wNormal = vec4(vtxNorm, 0) * Minv;
   vec3 N = normalize(wNormal.xyz);
   vec3 H = normalize(L + V);
   float cost = max(dot(N, L), 0), cosd = max(dot(N, H), 0);
   color = ka * La + (kd * cost + ks * pow(cosd, shine)) * Le;
}
```

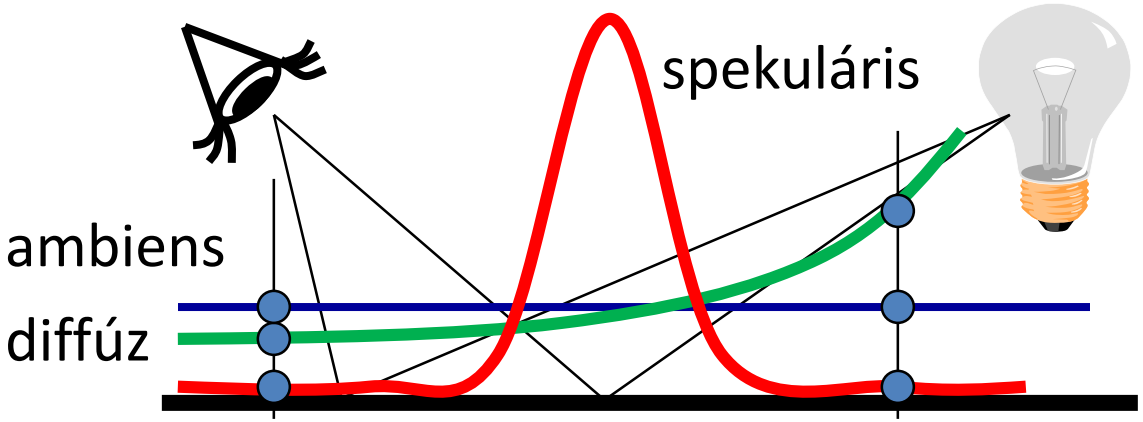
Per-vertex shading: Pixel shader

```
in vec4 color;           // interpolated color of vertex shader
out vec4 fragmentColor; // output goes to frame buffer

void main() {
   fragmentColor = color;
}
```

(Henri) Gouraud árnyalás problémái





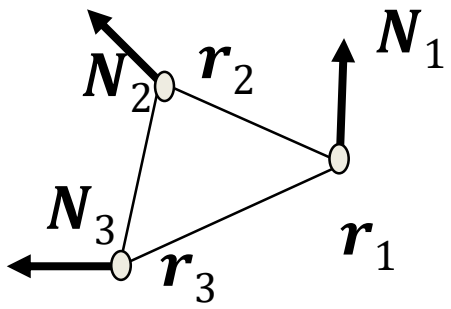


<u>További bajok:</u>

- anyagtulajdonság konstans
- árnyék nincs

különben a színt nem lehet interpolálni

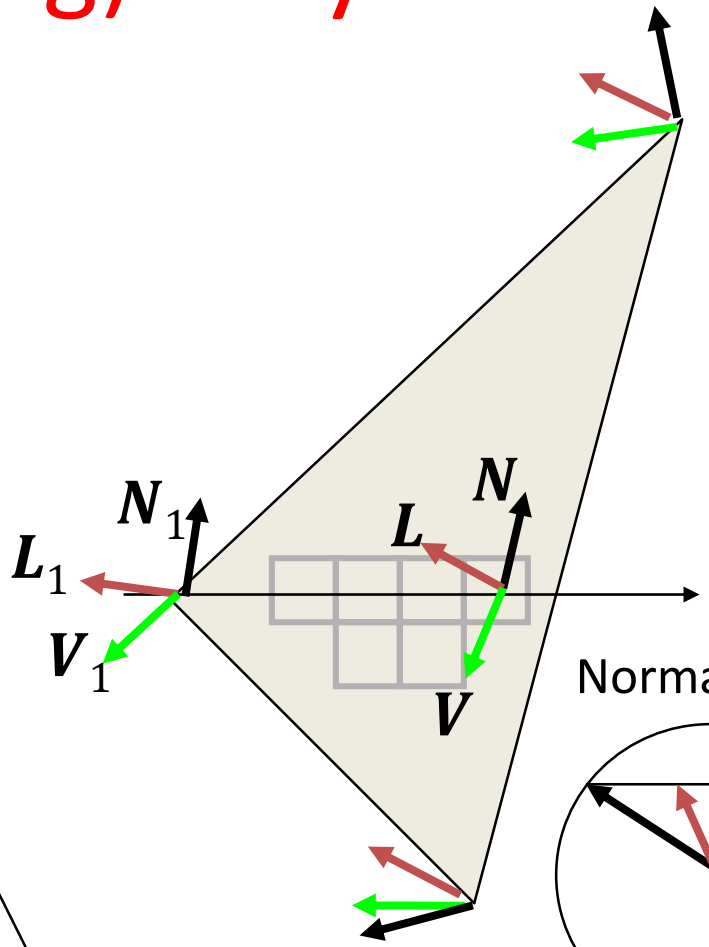
Per-pixel (Phong) árnyalás



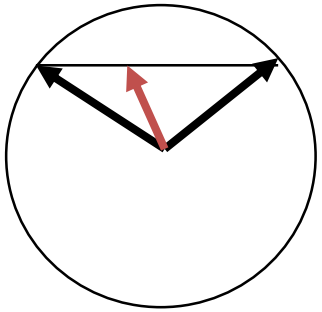
M: Modell

Vektor másolás

VP +
viewport



Normalizálás!



(per-pixel shading)

Képernyő

$$L(\boldsymbol{V}) \approx \sum_l L^{in}{}_l \; * f_r(\boldsymbol{L}_l, \boldsymbol{N}, \boldsymbol{V}) \cdot \cos^{+}\theta^{in}{}_l$$

Per-pixel shading: Vertex shader

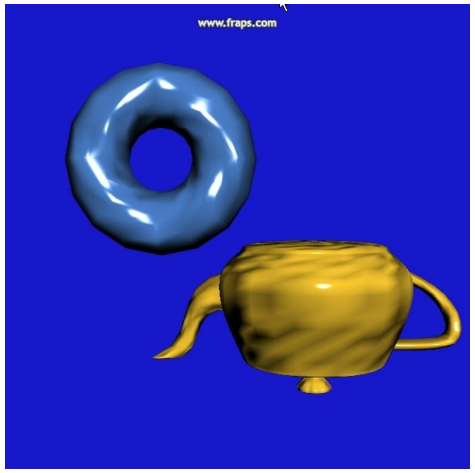
```
uniform mat4  MVP, M, Minv; // MVP, Model, Model-inverse
uniform vec4  wLiPos;       // pos of light source
uniform vec3  wEye;         // pos of eye

layout(location = 0) in vec3 vtxPos; // pos in model sp
layout(location = 1) in vec3 vtxNorm;// normal in model sp

out vec3 wNormal;           // normal in world space
out vec3 wView;             // view in world space
out vec3 wLight;            // light dir in world space

void main() {
    gl_Position = MVP * vec4(vtxPos, 1); // to NDC

    vec4 wPos = M * vec4(vtxPos, 1);
    wLight  = wLiPos.xyz/wLiPos.w - wPos.xyz/wPos.w;
    wView   = wEye - wPos.xyz/wPos.w;
    wNormal = (vec4(vtxNorm, 0) * Minv).xyz;
}
```

Per-pixel shading: Pixel shader

```
uniform vec3 kd, ks, ka;// diffuse, specular, ambient ref
uniform float shine;    // shininess for specular ref
uniform vec3 La, Le;    // ambient and dir/point source rad
in  vec3 wNormal;       // interpolated world sp normal
in  vec3 wView;         // interpolated world sp view
in  vec3 wLight;        // interpolated world sp illum dir
out vec4 fragmentColor; // output goes to frame buffer

void main() {
   vec3 N = normalize(wNormal);
   vec3 V = normalize(wView);
   vec3 L = normalize(wLight);
   vec3 H = normalize(L + V);
   float cost = max(dot(N,L), 0), cosd = max(dot(N,H), 0);
   vec3 color = ka * La + (kd * cost + ks * pow(cosd,shine)) * Le;
   fragmentColor = vec4(color, 1);
}
```

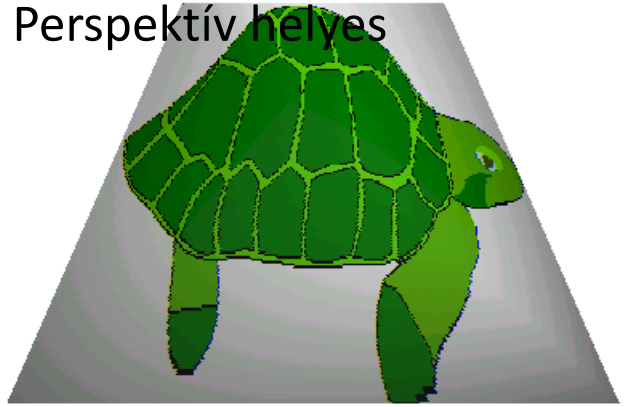
NPR: Non-Photorealistic Rendering

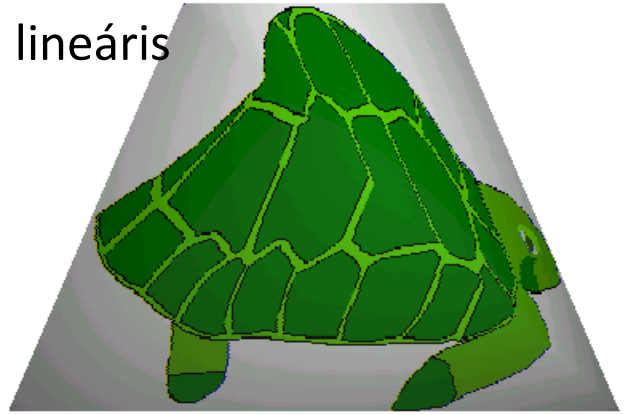
```
uniform vec3 kd;                        // diffuse ref
in  vec3 wNormal, wView, wLight; // interpolated
out vec4 fragColor;                     // output to frame buffer

void main() {
    vec3 N = normalize(wNormal);
    vec3 V = normalize(wView);
    vec3 L = normalize(wLight);
    float y = (dot(N, L) > 0.5) ? 1 : 0.5;
    fragColor = (abs(dot(N, V)) < 0.2) ? vec4(0, 0, 0, 1) : vec4(y * kd, 1);
}
```

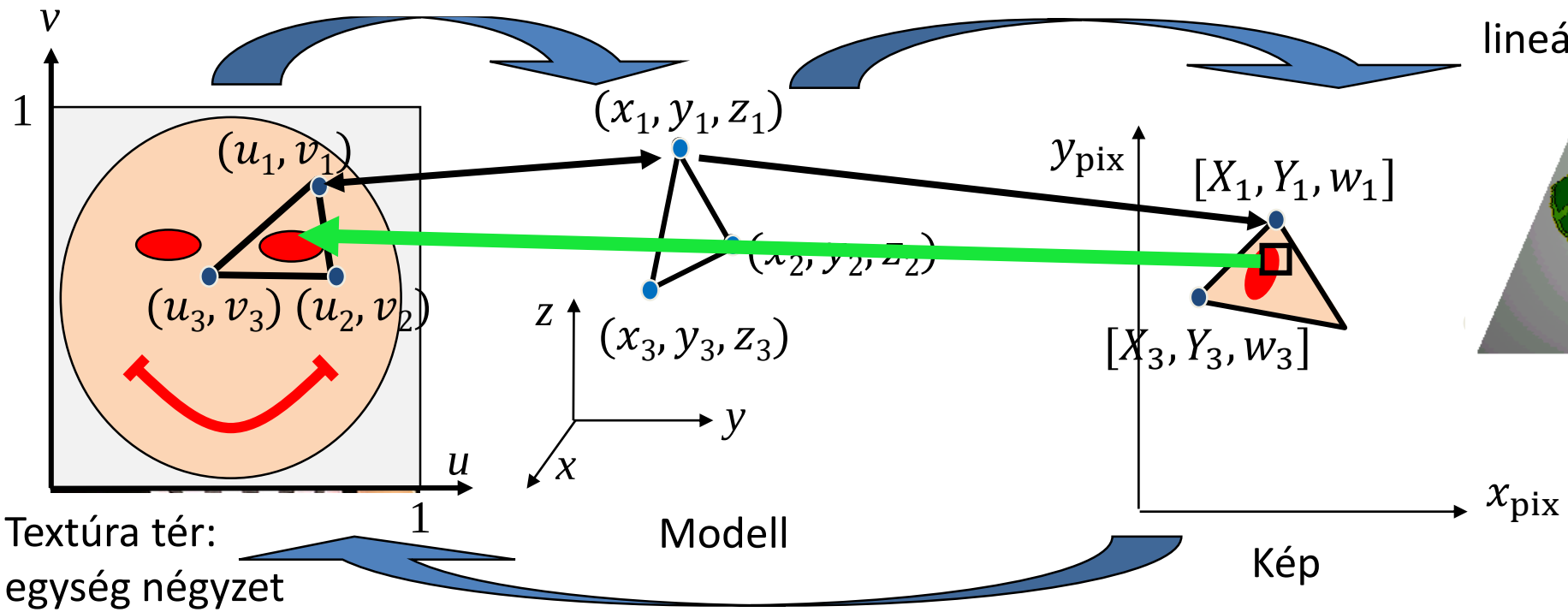


2D Textúrázás

$$[X, Y, w] = [u, v, 1] \cdot \boldsymbol{P}$$
$$(x_{\text{pix}}, y_{\text{pix}}) = [X/w, Y/w]$$

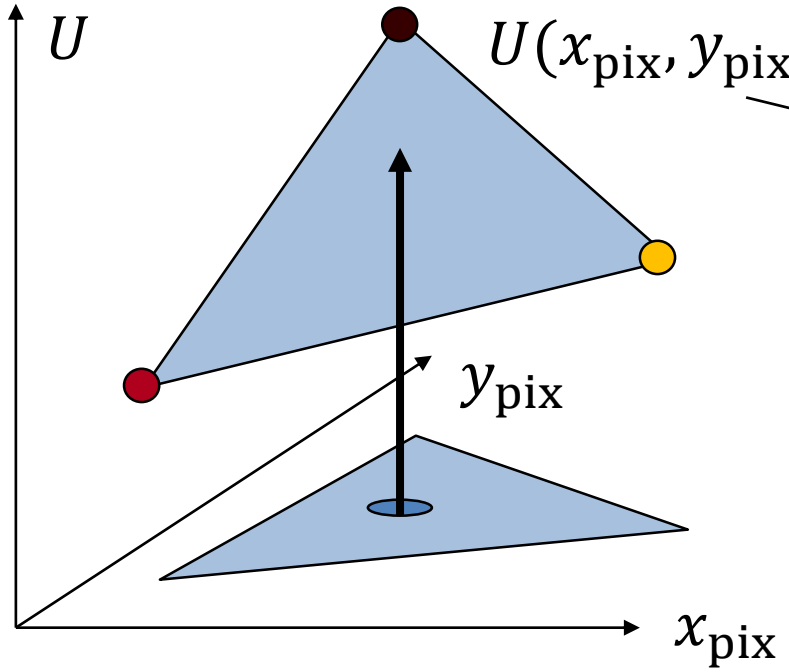






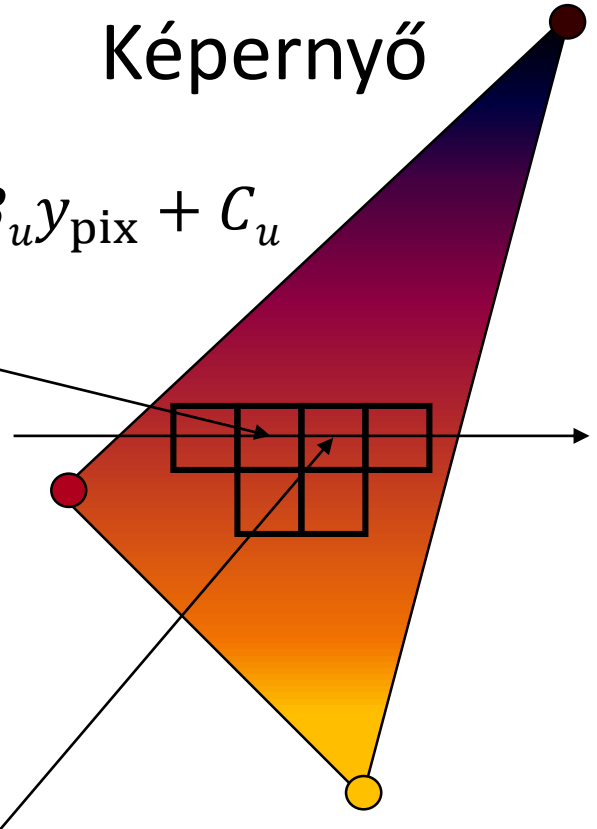
$$[u/w, v/w, 1/w] = [x_{\text{pix}}, y_{\text{pix}}, 1] \cdot \boldsymbol{P}^{-1}$$
$$[U, \quad V, \quad W] \rightarrow u = U/W, \; v = V/W$$

Lineáris interpoláció



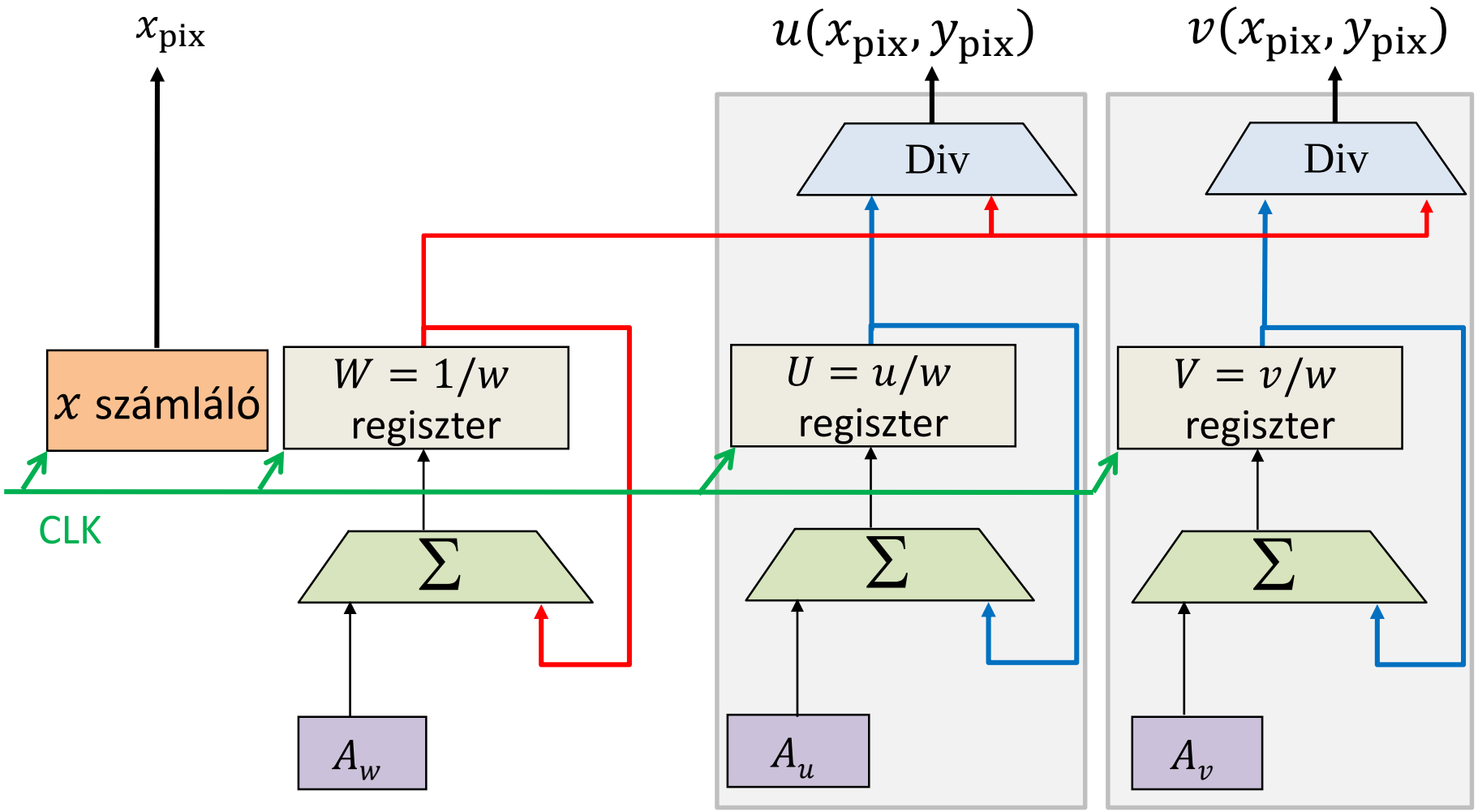
$$U(x_{\text{pix}}, y_{\text{pix}}) = A_u x_{\text{pix}} + B_u y_{\text{pix}} + C_u$$

Képernyő

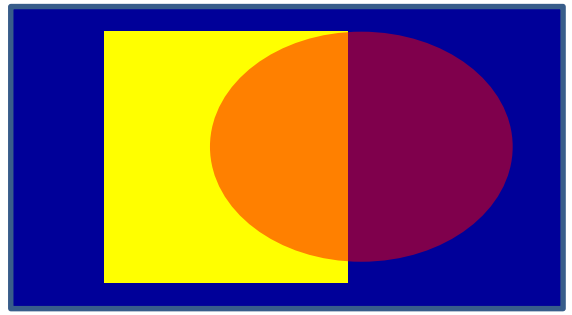


$$U(x_{\text{pix}} + 1, y_{\text{pix}}) = U(x_{\text{pix}}, y_{\text{pix}}) + A_u$$

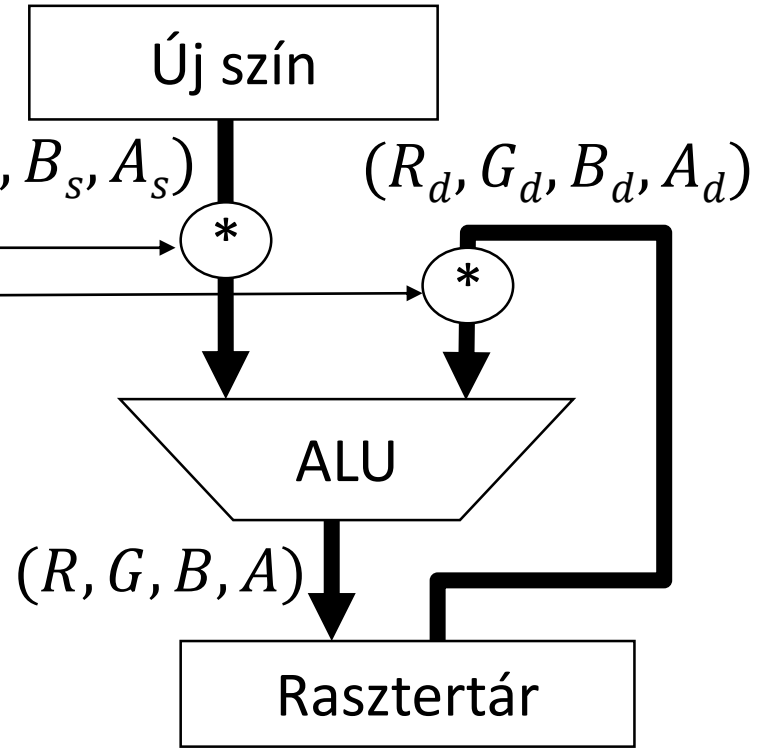
Interpolációs hardver



Kompozitálás és átlátszóság
Sorrend számít!



```
glEnable(GL_BLEND);

glBlendFunc(
    GL_SRC_ALPHA,
    GL_ONE_MINUS_SRC_ALPHA
);

glDrawArrays(GL_TRIANGLES, 0, nVtx);

glDisable(GL_BLEND);
```



Inkrementális képszintézis csővezeték

- Azért transzformáltunk, hogy a láthatósági feladatot és a vetítést képernyő koordináta-rendszerben oldhassuk meg
  - Triviális hátsó lap eldobás
  - Z-buffer algoritmus
  - Vetítés = z eldobása, 3D háromszög = 2D háromszög
- Per-pixel árnyalás:
  - Vektorokat csúcspontonként számítjuk és pixelekre interpoláljuk
  - Illuminációs képlet pixelenként