

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

*Jim Blinn*

# Inkrementális 3D képszintézis

## 1. Bevezetés

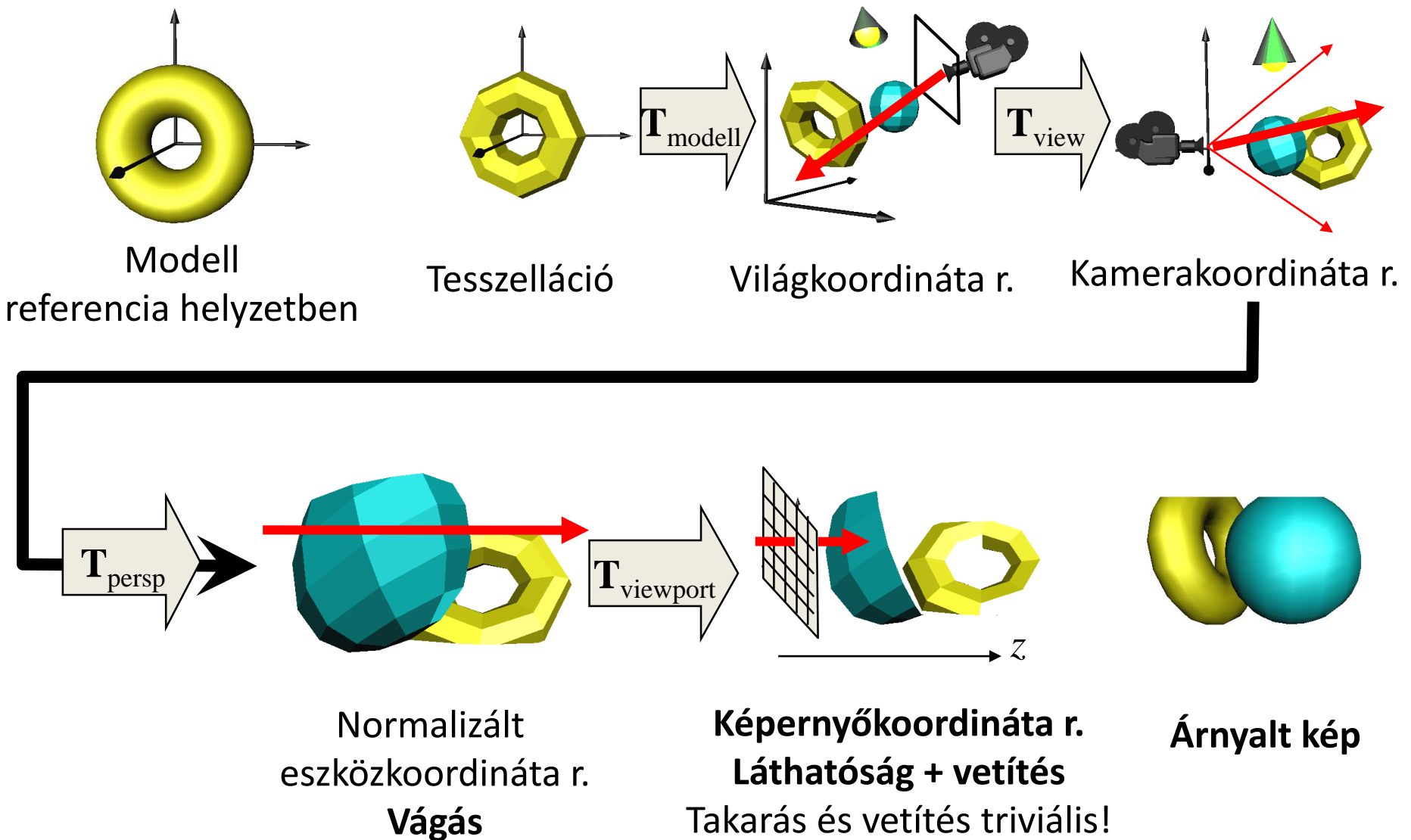
Szirmay-Kalos László



# Inkrementális képszintézis

- Objektum vezérelt megközelítés
  - Cél a sebesség és a hw támogathatóság
- 
- koherencia: oldjuk meg nagyobb egységekre
  - feleslegesen ne számoljunk: vágás
  - transzformációk: minden feladathoz megfelelő koordinátarendszert
    - vágni, transzformálni nem lehet akármit: tesszelláció

# 3D inkrementális képszintézis



*“Order is repetition of units.*

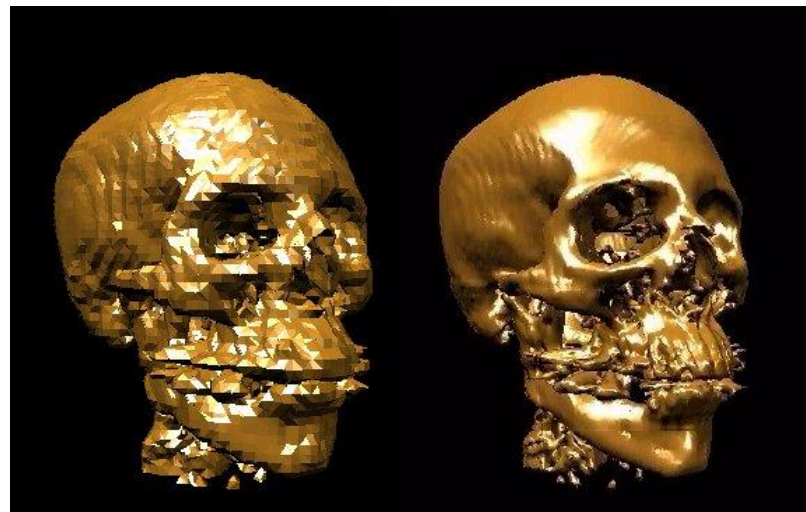
*Chaos is multiplicity without rhythm.”*

*Maurits Cornelis Escher*

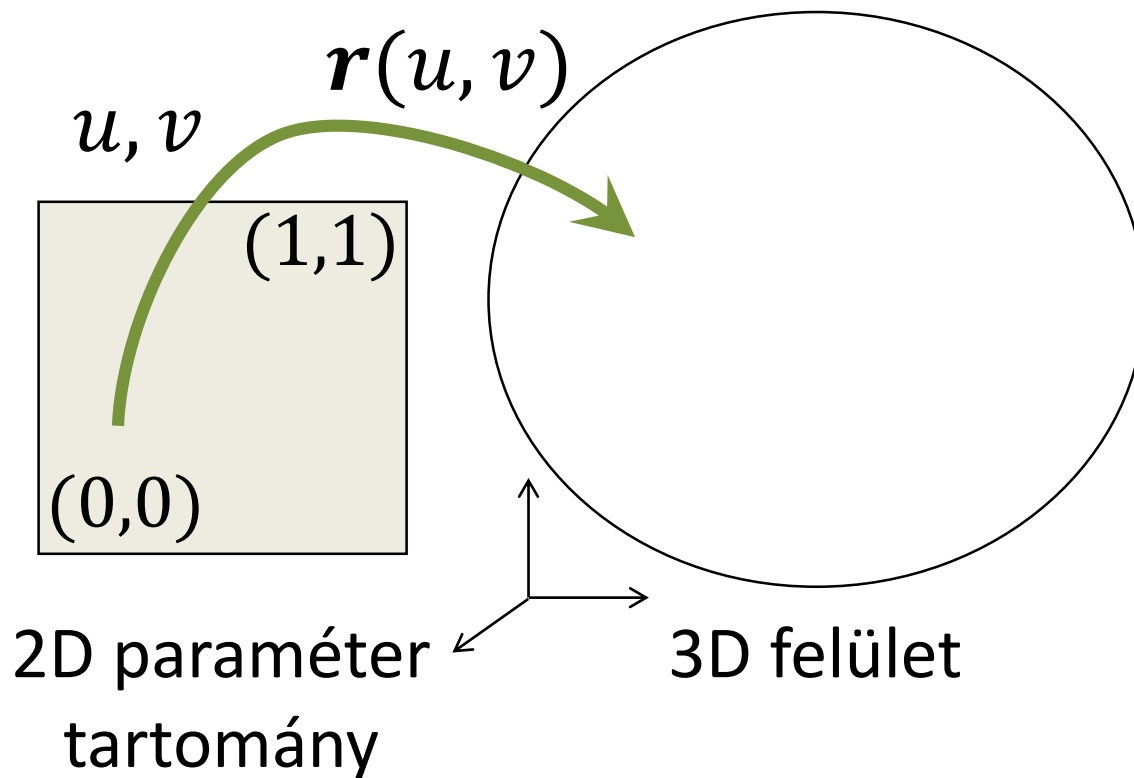
# **Inkrementális 3D képszintézis**

## **2. Geometria a GPU-nak**

Szirmay-Kalos László

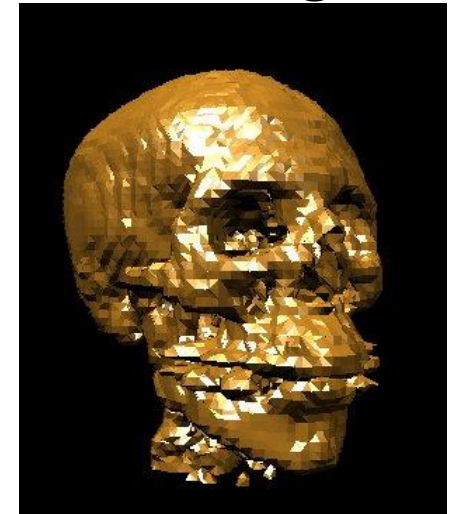
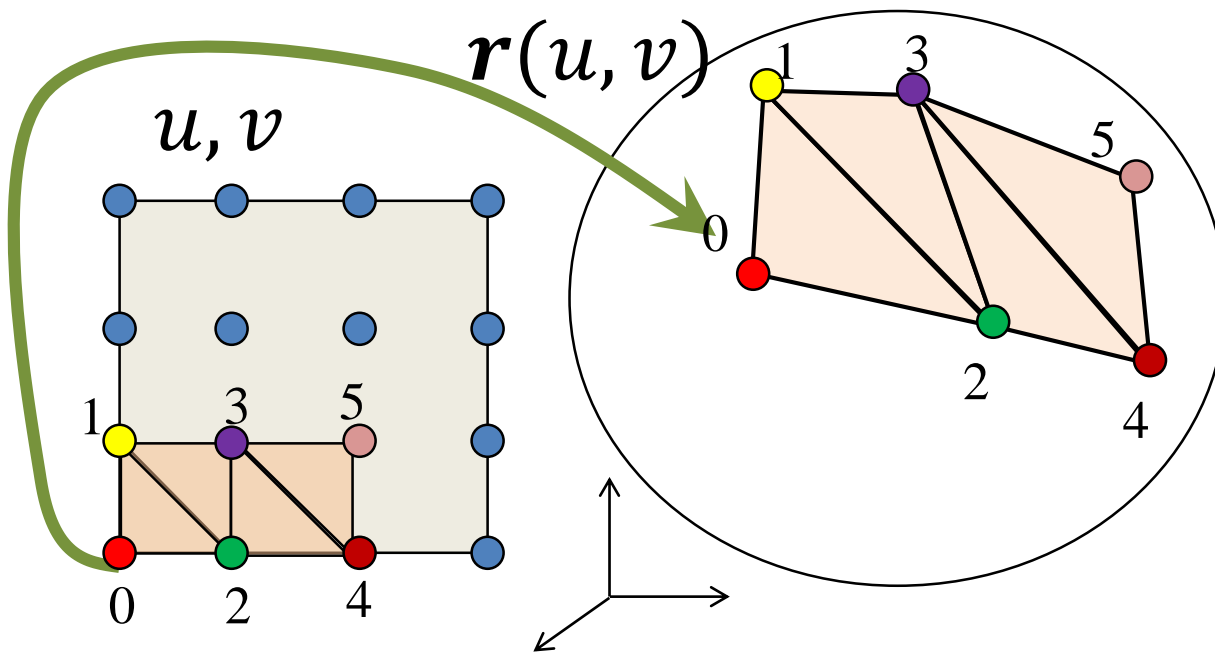


# Parametrikus felületek tesszellációja



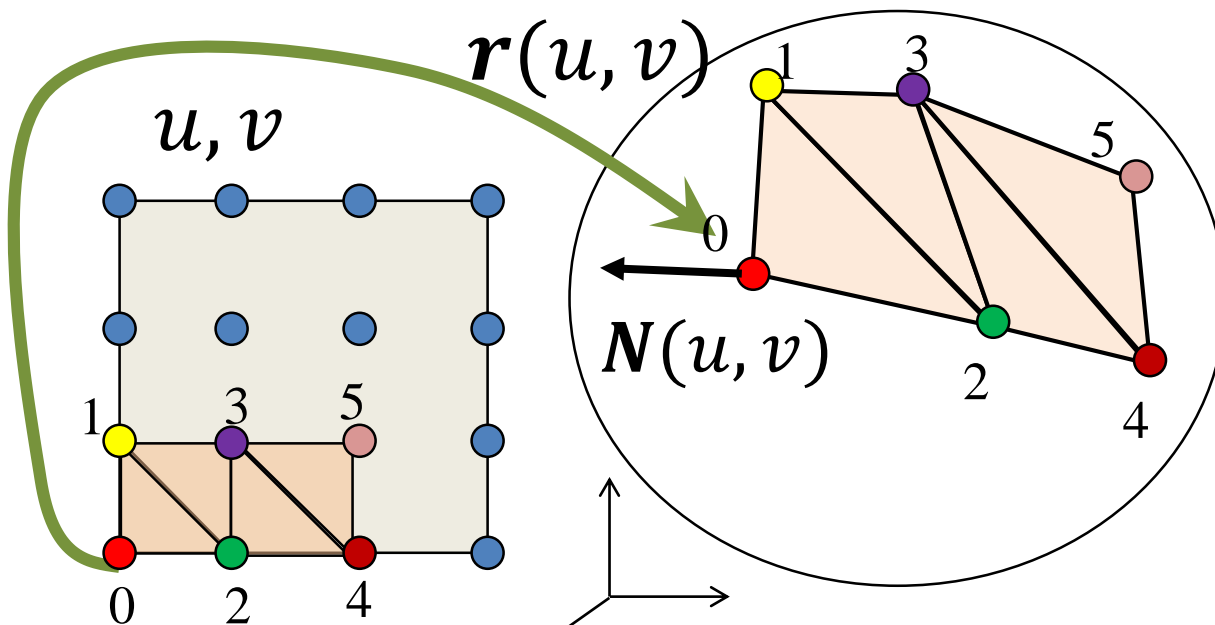
# Parametrikus felületek tesszellációja

„Paraméterterben szomszédos” pontokból háromszögek



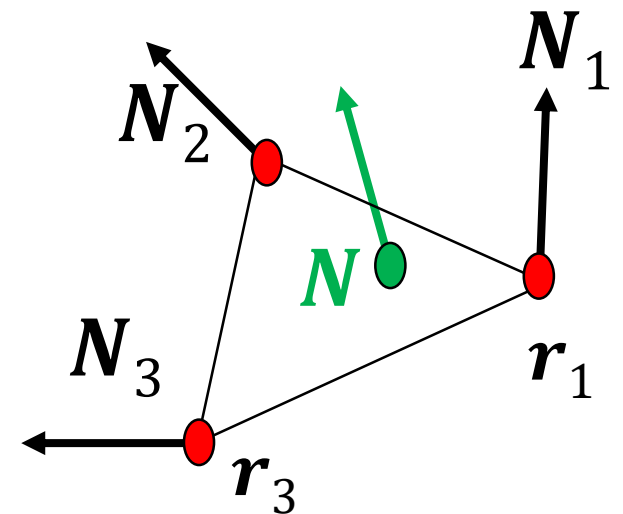
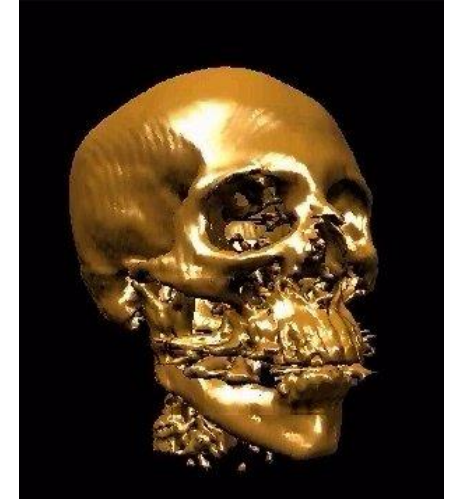
# Parametrikus felületek tesszellációja

„Paraméterterben szomszédos” pontokból háromszögek



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

Árnyaló normálok



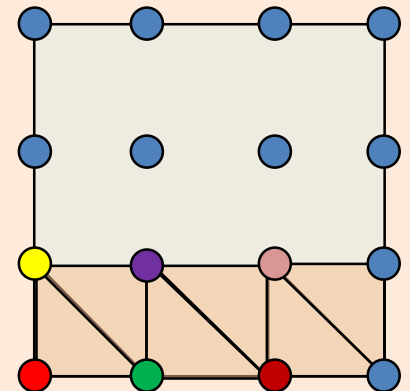
# Objektumok az GPU-nak

```
class Geometry {  
protected:  
    unsigned int vao, vbo;  
public:  
    Geometry() {  
        glGenVertexArrays(1, &vao);  
        glBindVertexArray(vao);  
        glGenBuffers(1, &vbo);  
        glBindBuffer(GL_ARRAY_BUFFER, vbo);  
    }  
    virtual void Draw() = 0;  
    ~Geometry() {  
        glDeleteBuffers(1, &vbo);  
        glDeleteVertexArrays(1, &vao);  
    }  
};
```

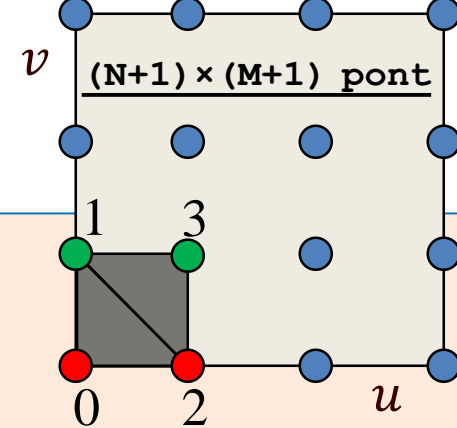


# Parametrikus felületek GPU-nak

```
class ParamSurface : public Geometry {  
    unsigned int nVtxStrip, nStrips;  
virtual void eval(float u, float v, vec3& pos, vec3& norm)=0;  
  
    struct VertexData {  
        vec3 pos, norm;  
        vec2 tex;  
    };  
  
    virtual VertexData GenVertexData(float u, float v) {  
        VertexData vd;  
        vd.tex = vec2(u, v); eval(u, v, vd.pos, vd.norm);  
        return vd;  
    }  
  
public:  
    void Create(int N, int M);  
    void Draw() {  
        glBindVertexArray(vao);  
        for (int i = 0; i < nStrips; i++)  
            glDrawArrays(GL_TRIANGLE_STRIP, i*nVtxStrip, nVtxStrip);  
    }  
};
```



# Parametrikus felület GPU-nak

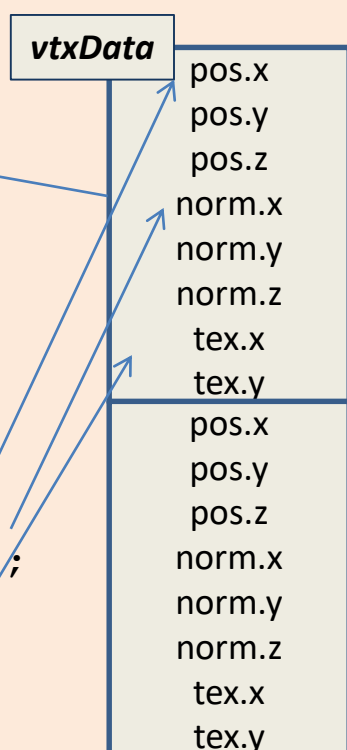


```
void ParamSurface::Create(int N, int M) {
    nVtxStrip = (M + 1) * 2;
    nStrips = N;
    vector<VertexData> vtxData; // CPU-n
    for (int i = 0; i < N; i++) for (int j = 0; j <= M; j++) {
        vtxData.push_back(GenVertexData((float)j / M, (float)i / N));
        vtxData.push_back(GenVertexData((float)j / M, (float)(i + 1) / N));
    }
    glBindVertexArray(vao);
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER,
                 vtxData.size() * sizeof(VertexData),
                 &vtxData[0], GL_STATIC_DRAW);

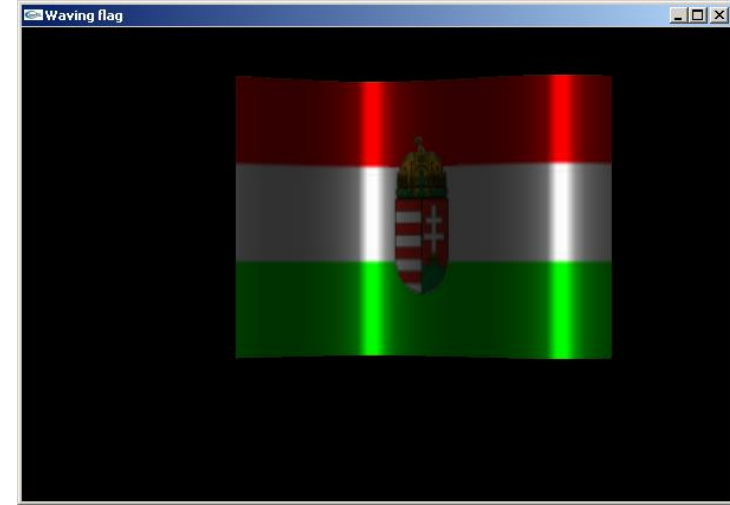
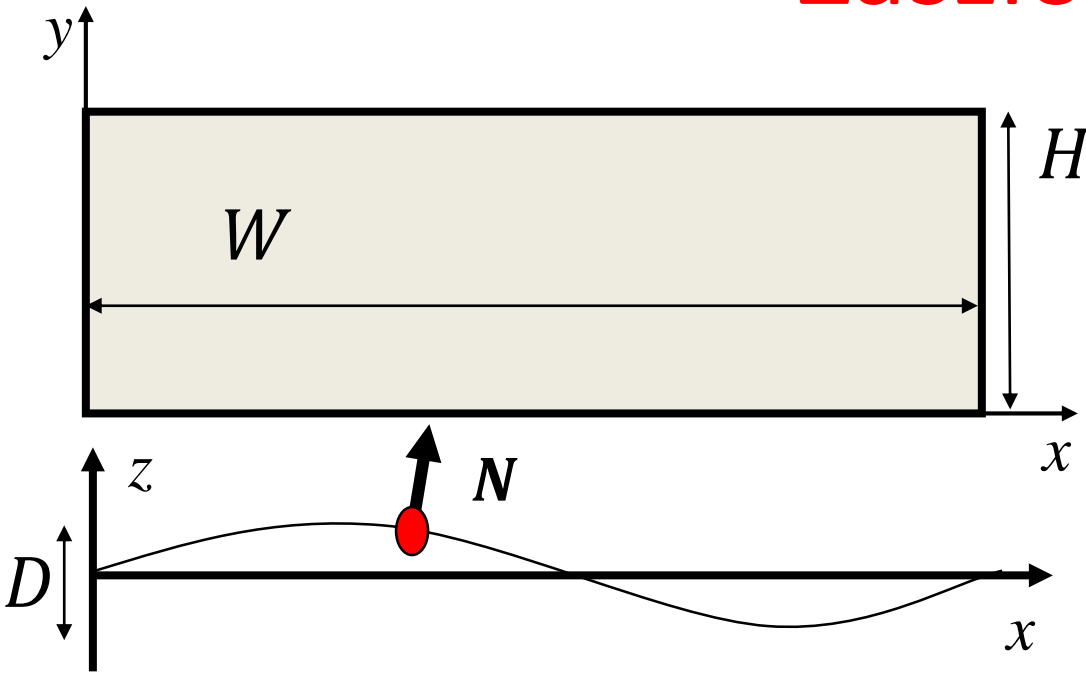
    glEnableVertexAttribArray(0); // AttArr 0 = POSITION
    glEnableVertexAttribArray(1); // AttArr 1 = NORMAL
    glEnableVertexAttribArray(2); // AttArr 2 = UV
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE,
                          sizeof(VertexData), (void*)offsetof(VertexData, pos));
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
                          sizeof(VertexData), (void*)offsetof(VertexData, norm));
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE,
                          sizeof(VertexData), (void*)offsetof(VertexData, tex));
}
```

$u$

$v$



# Zászló

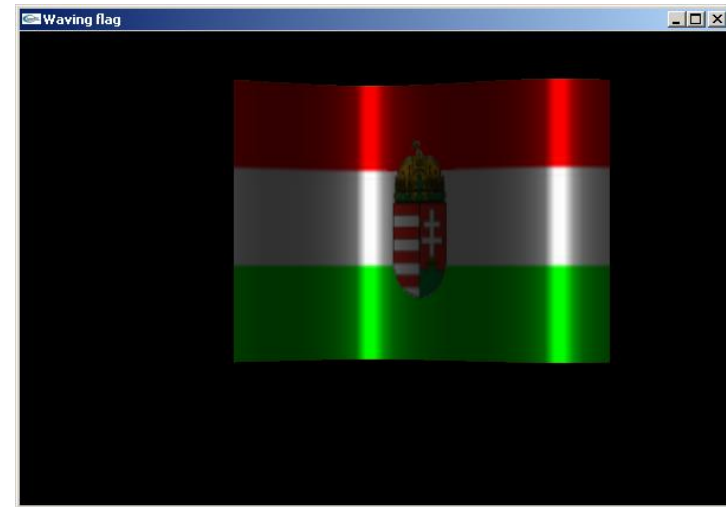


$$\mathbf{r}(u, v) = [uW, \quad vH, \quad D \sin(Ku + \alpha)]$$

$$\frac{\partial \mathbf{r}}{\partial u} = \begin{bmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ W, & 0, & K D \cos(Ku + \alpha) \\ 0, & H, & 0 \end{bmatrix}$$

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

# Zászló



```
class Flag : public ParamSurface {  
    float W, H, D, K, phase;  
public:  
  
    void eval(float u, float v, vec3& pos, vec3& norm) {  
        float angle = u * K + phase;  
        pos = vec3(u * W, v * H, D * sin(angle));  
        norm = vec3(-K * D * cos(angle), 0, W);  
    }  
};
```

# 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:

$$\begin{aligned} [\mathbf{r}, 1] \mathbf{T}_{Model} &= [\mathbf{r}_{world}, 1] \\ \mathbf{T}_{Model}^{-1} [\mathbf{N}, 0]^T &= [\mathbf{N}_{world}, d]^T \end{aligned}$$

Kamera transzformáció:

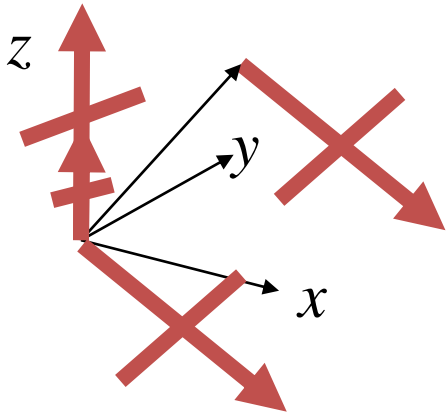
$$[\mathbf{r}_{world}, 1] \mathbf{T}_{View} = [\mathbf{r}_{camera}, 1]$$

Projektív transzformáció:

$$[\mathbf{r}_{camera}, 1] \mathbf{T}_{Proj} = [\mathbf{r}_{ndc} w, w]$$

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

# Modellezési transzformáció



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

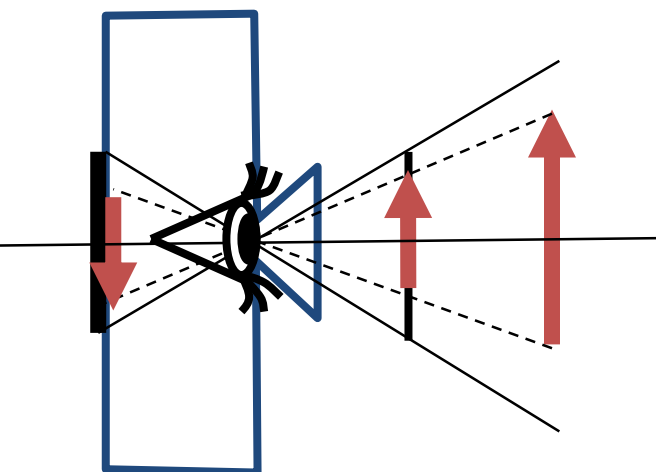
$$\mathbf{T}_{4 \times 4} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{i}'_x & \mathbf{i}'_y & \mathbf{i}'_z & 0 \\ \mathbf{j}'_x & \mathbf{j}'_y & \mathbf{j}'_z & 0 \\ \mathbf{k}'_x & \mathbf{k}'_y & \mathbf{k}'_z & 0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

$$\mathbf{r}' = \mathbf{r} \cos(\varphi) + \underbrace{\mathbf{d}(\mathbf{r} \cdot \mathbf{d})}_{\text{projection}} (1 - \cos(\varphi)) + \mathbf{d} \times \mathbf{r} \sin(\varphi)$$

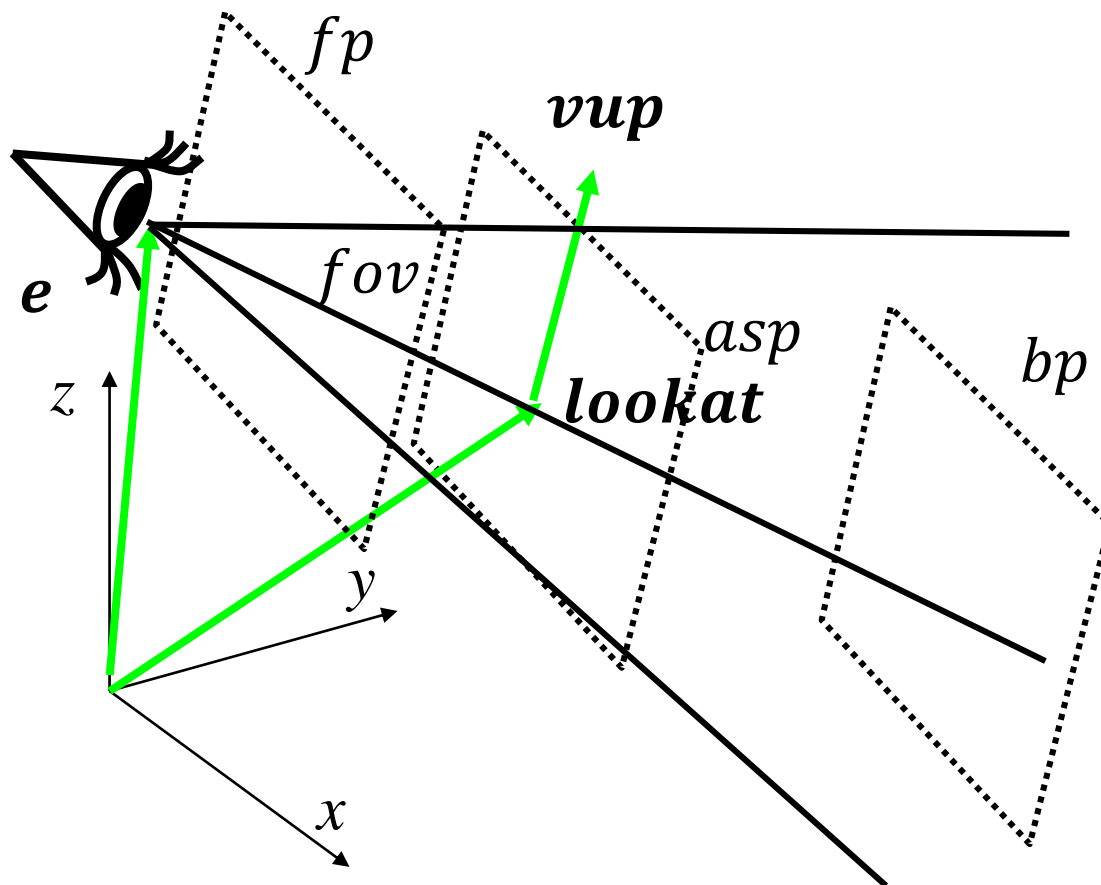




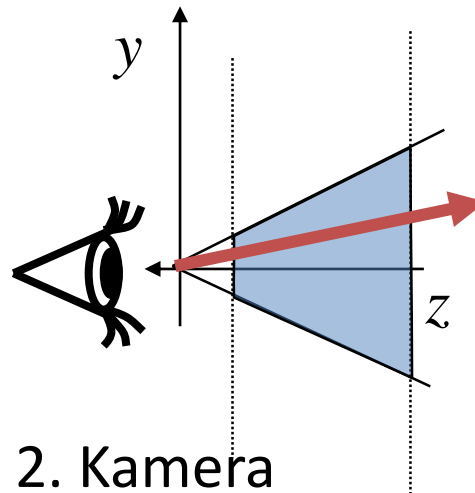
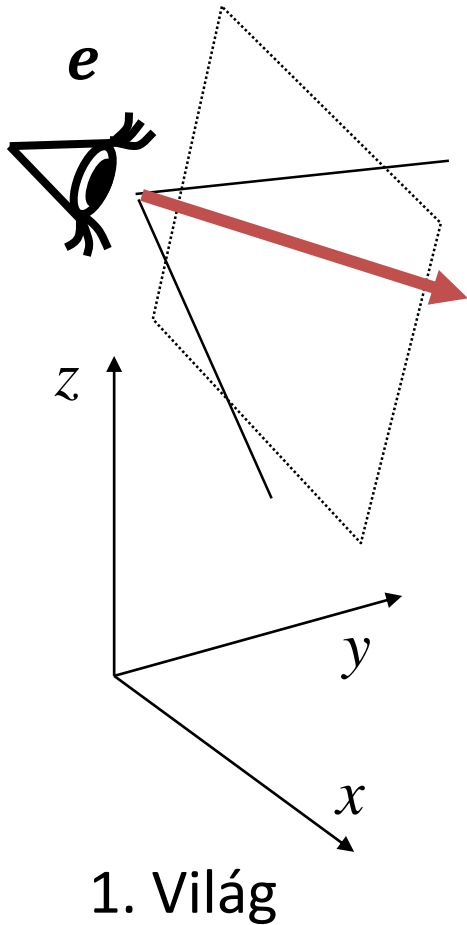
# Kamera modell



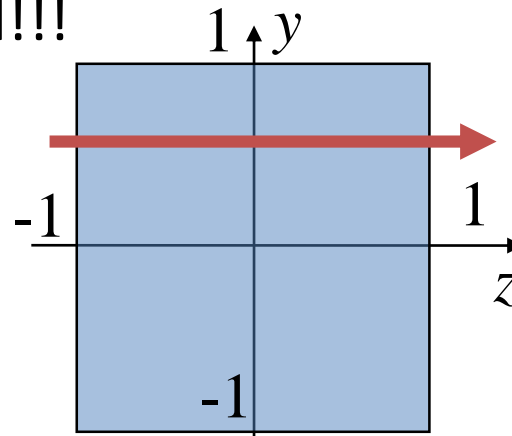
Mi: Camera obscura



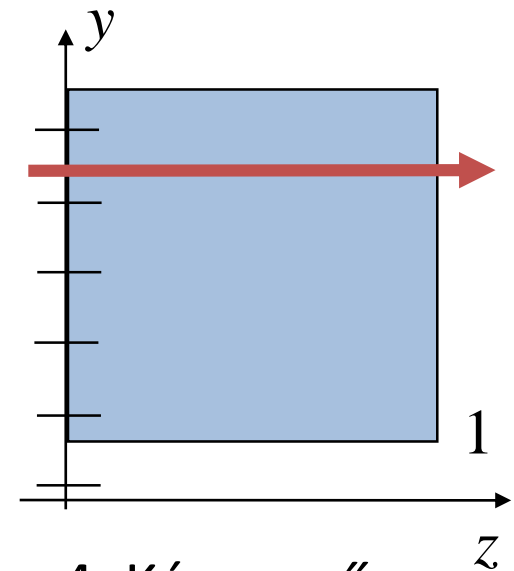
# Világból a képernyőre



Bal!!!

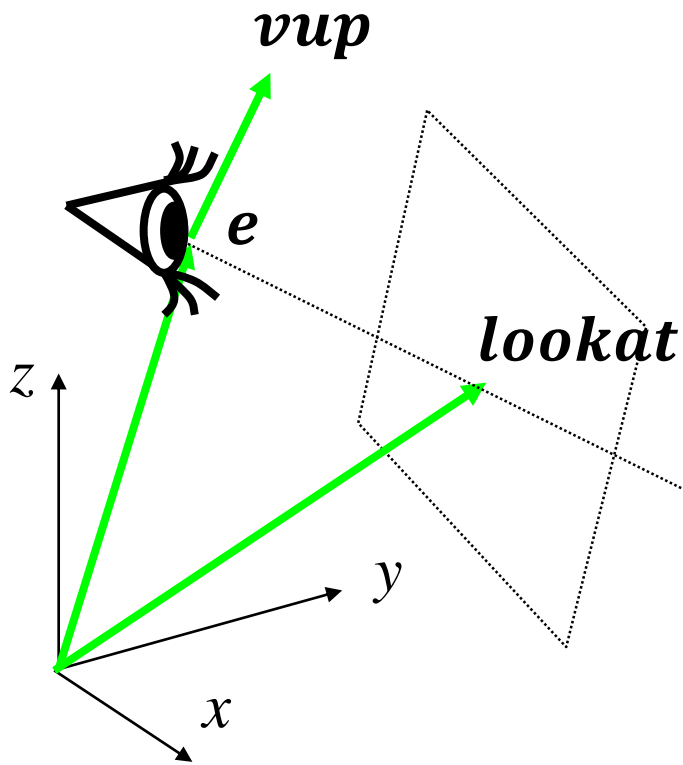


3. Normalizált képernyő



4. Képernyő

# View transzformáció

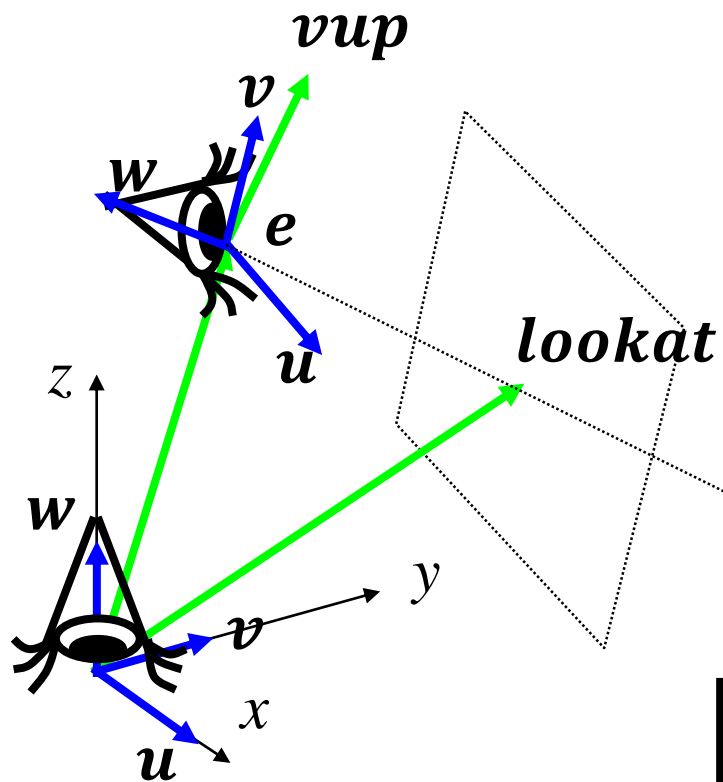


$$w = (e - lookat) / |e - lookat|$$

$$u = (vup \times w) / |w \times vup|$$

$$v = w \times u$$

# View transzformáció



$$\mathbf{w} = (\mathbf{e} - \text{lookat}) / |\mathbf{e} - \text{lookat}|$$

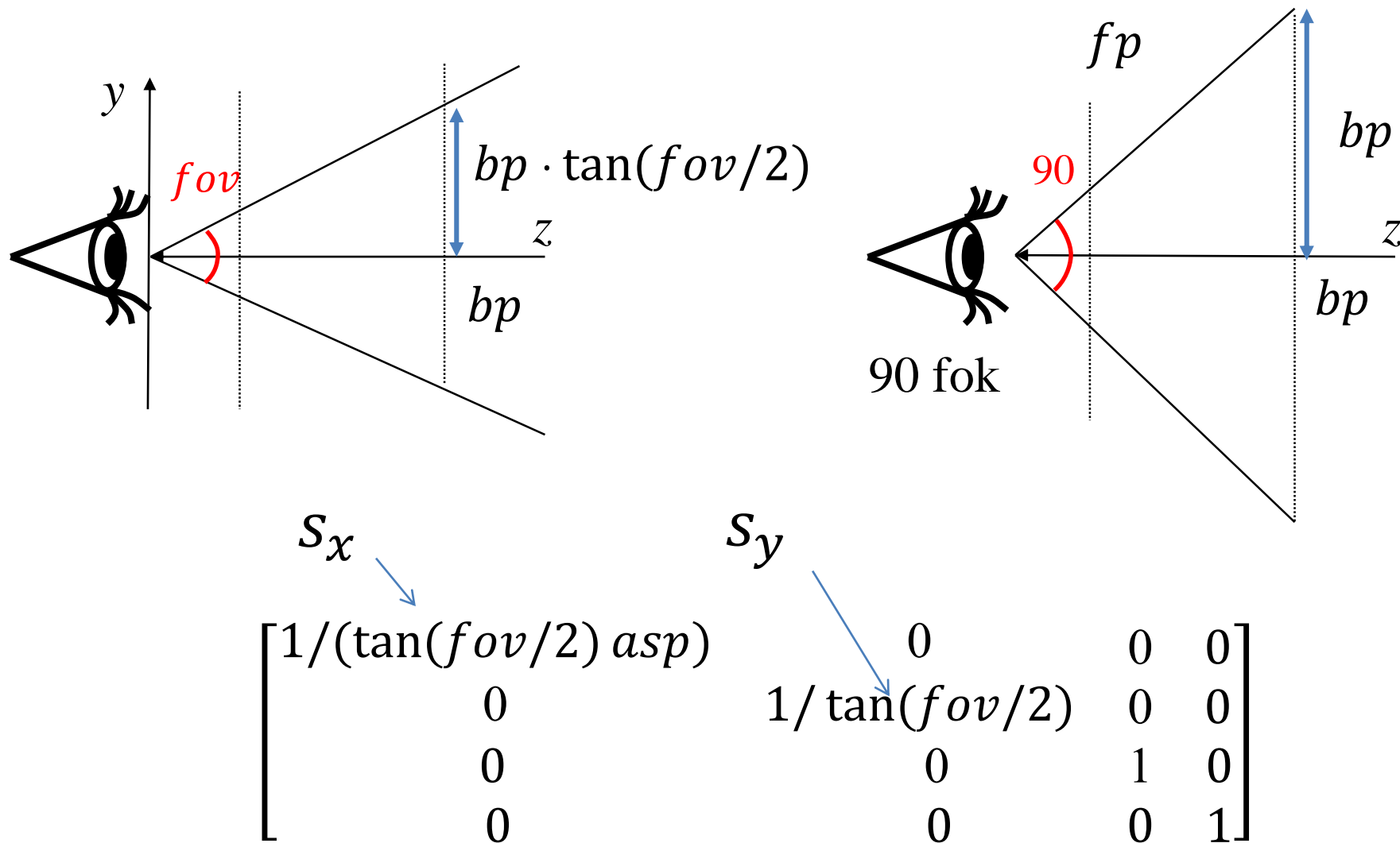
$$\mathbf{u} = (\mathbf{vup} \times \mathbf{w}) / |\mathbf{w} \times \mathbf{vup}|$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

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

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

# Látószög normalizálás



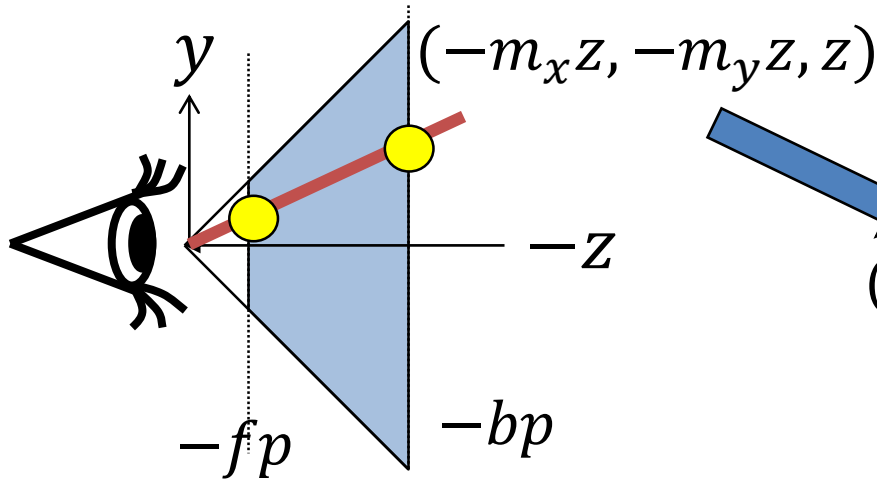
2D egyenes explicit egyenlete:

$$y = mx + b$$

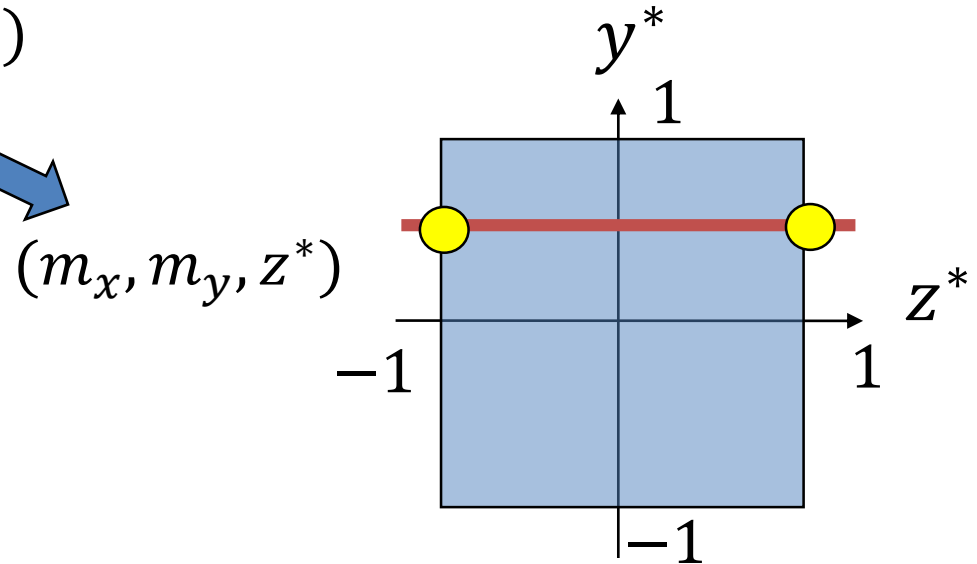
Origón átmenő:  $y = mx$

Vízszintes:  $y = b$

# Normalizálás utáni perspektív transzformáció



Normalizált kamera

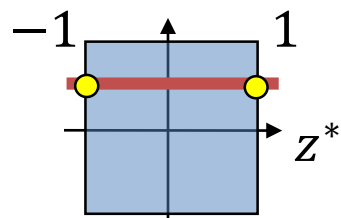
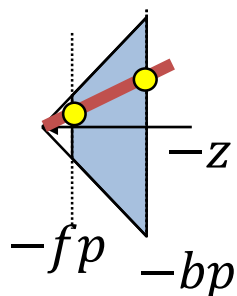


Normalizált képernyő: NDC

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

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

# Perspektív transzformáció



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

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

$$-z z^* = \alpha z + \beta \quad \rightarrow$$

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

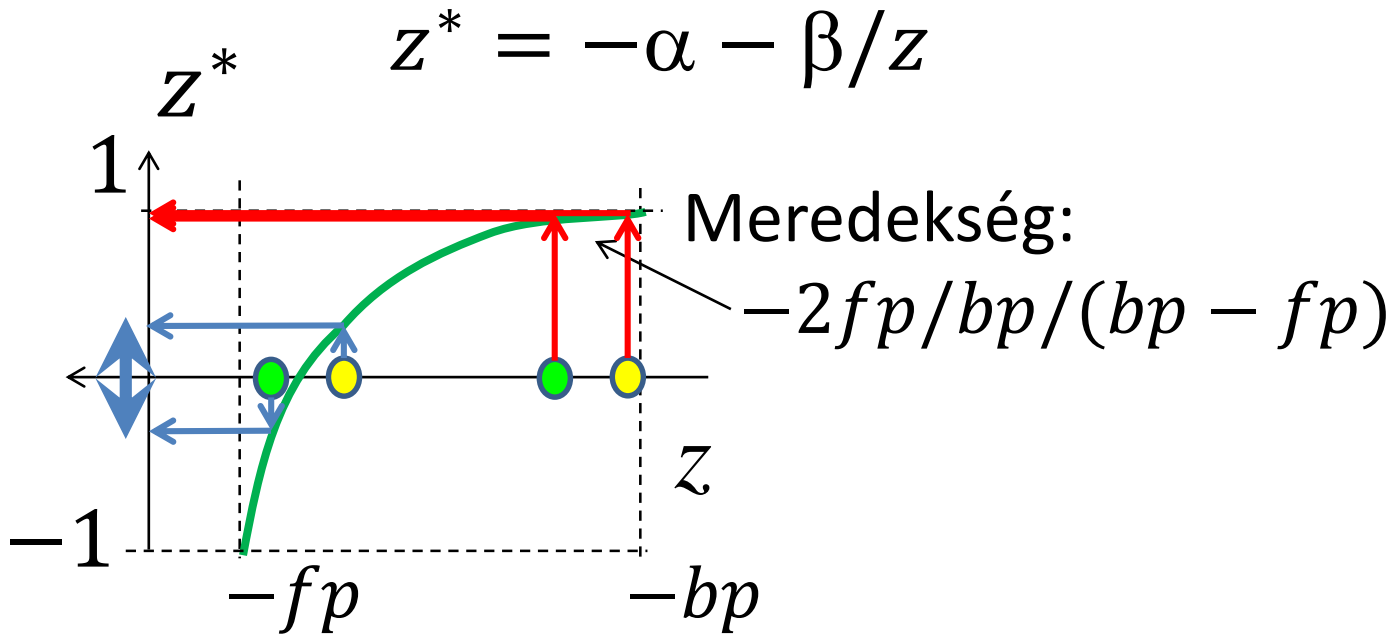
$$fp(-1) = \alpha(-fp) + \beta$$

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

$$\alpha = -(fp + bp)/(bp - fp)$$

$$\beta = -2fp \cdot bp/(bp - fp)$$

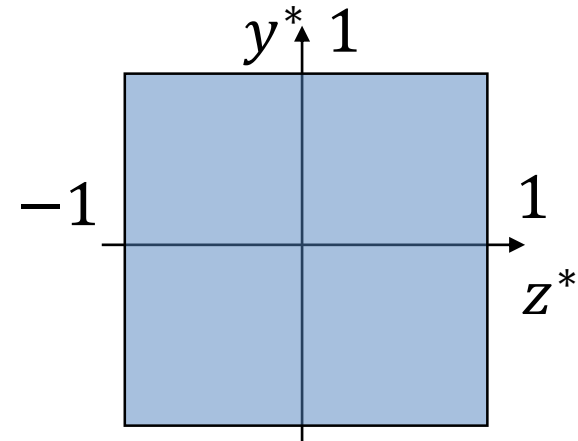
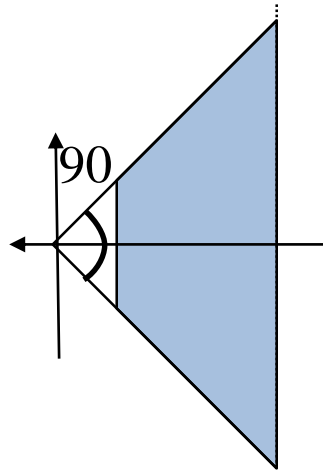
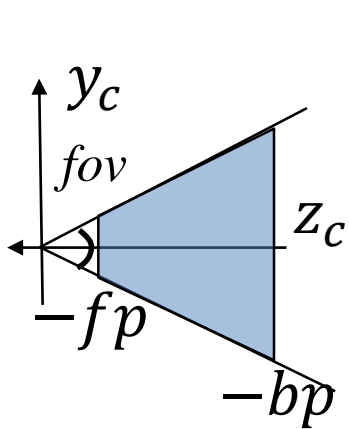
# Z-fighting



*fp/bp* nem lehet kicsi!



# Projekció (perspektív) transzformáció



$$\mathbf{T}_{Proj} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & -(fp + bp)/(bp - fp) & -1 \\ 0 & 0 & -2fp \cdot bp/(bp - fp) & 0 \end{bmatrix}$$

$$[x_c, y_c, z_c, 1] \cdot [X, Y, Z, w]$$

**Perspektív torzítás = homogén osztás**

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

$$w = -z_c$$

# Camera osztály

```
class Camera {  
    vec3  wEye, wLookat, wVup; // extrinsic parameters  
    float fov, asp, fp, bp;    // intrinsic parameters  
public:  
    mat4 V() { // view matrix  
        vec3 w = normalize(wEye - wLookat);  
        vec3 u = normalize(cross(wVup, w));  
        vec3 v = cross(w, u);  
        return TranslateMatrix(-wEye) * mat4( 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);  
    }  
    mat4 P() { // projection matrix  
        float sy = 1/tanf(fov/2);  
        return mat4(sy/asp, 0,    0,    0,  
                    0,    sy,    0,    0,  
                    0,    0,    -(fp+bp) / (bp-fp), -1,  
                    0,    0,    -2*fp*bp / (bp-fp), 0);  
    }  
};
```

# Transzformációk előkészítése a CPU-n

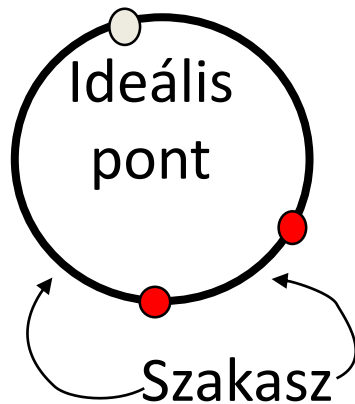
```
void Draw() {  
    mat4 M = ScaleMatrix(scale) *  
             RotationMatrix(rotAng, rotAxis) *  
             TranslateMatrix(pos);  
    mat4 Minv = TranslateMatrix(-pos) *  
                RotationMatrix(-rotAngle, rotAxis) *  
                ScaleMatrix(1/scale);  
  
    mat4 MVP = M * camera.V() * camera.P();  
  
    shader->setUniform(M, "M");  
    shader->setUniform(Minv, "Minv");  
    shader->setUniform(MVP, "MVP");  
  
    glBindVertexArray(vao);  
    glDrawArrays( . . . );  
}
```

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

# 3D vágás homogén koordinátákban (GPU)

## Homogén koordinátákban kell vágni



$$[X(t), Y(t), Z(t), w(t)] =$$

$$[X_1, Y_1, Z_1, w_1](1 - t) + [X_2, Y_2, Z_2, w_2]t$$

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

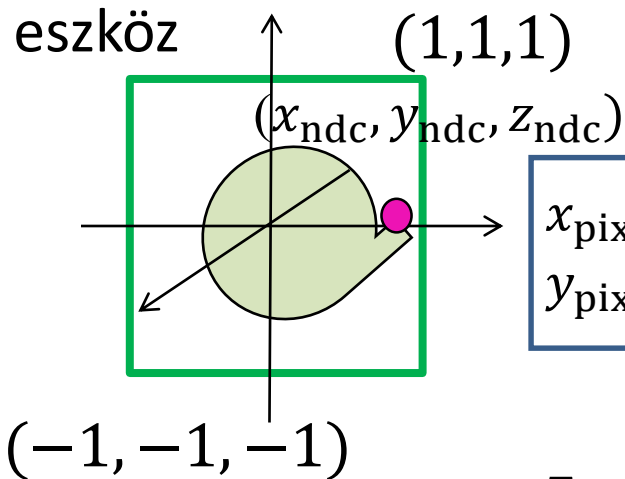
$$w > 0$$

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

Mert  $w = -z_c > 0$  a szem előtt

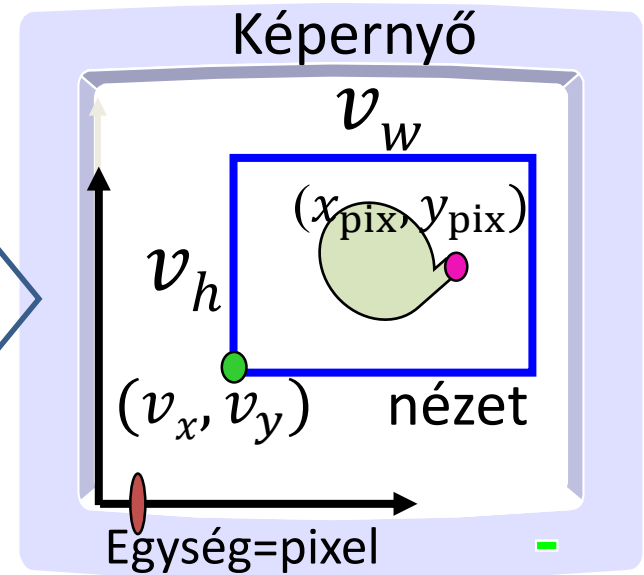
# Viewport transzformáció: Normalizáltból képernyő koordinátákba (GPU)

Normalizált



$$x_{\text{pix}} = v_w(x_{\text{ndc}} + 1)/2 + v_x$$
$$y_{\text{pix}} = v_h(y_{\text{ndc}} + 1)/2 + v_y$$

$$z_{\text{pix}} = (z_{\text{ndc}} + 1)/2$$



# 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

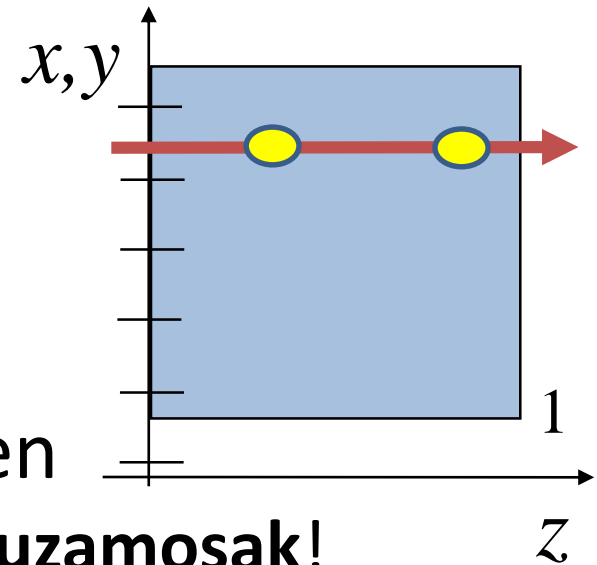
## 4. Láthatóság és árnyalás

Szirmay-Kalos László

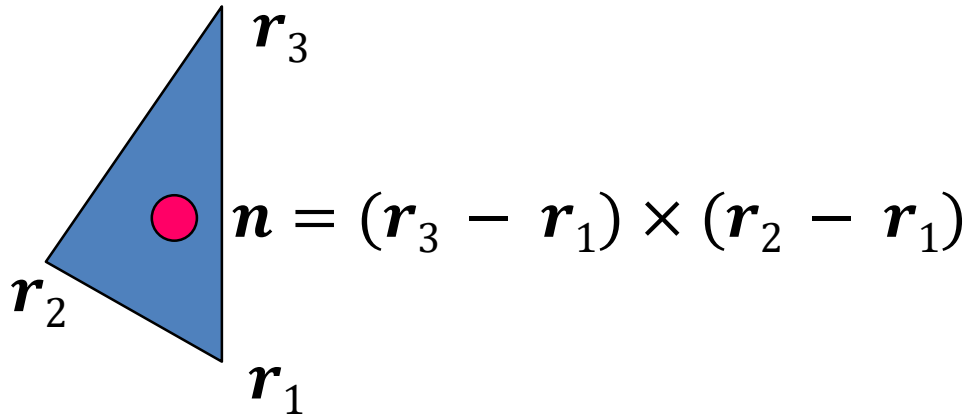




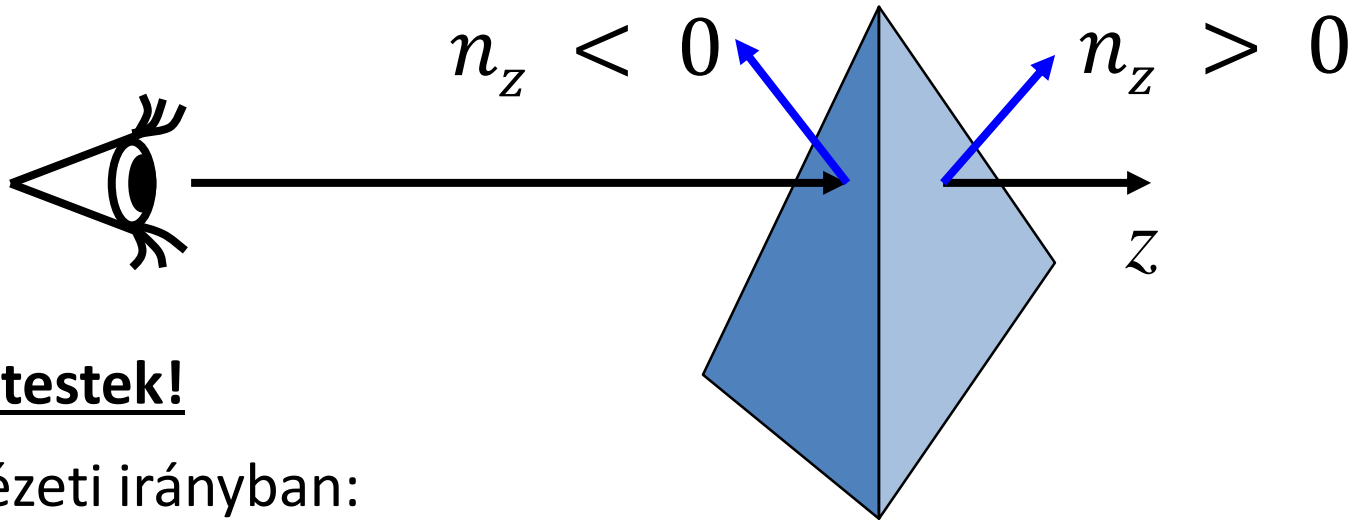
# Takarás



- Képernyő koordinátarendszerben
  - vetítő sugarak a  $z$  tengellyel párhuzamosak!
  - 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



### Valódi 3D testek!

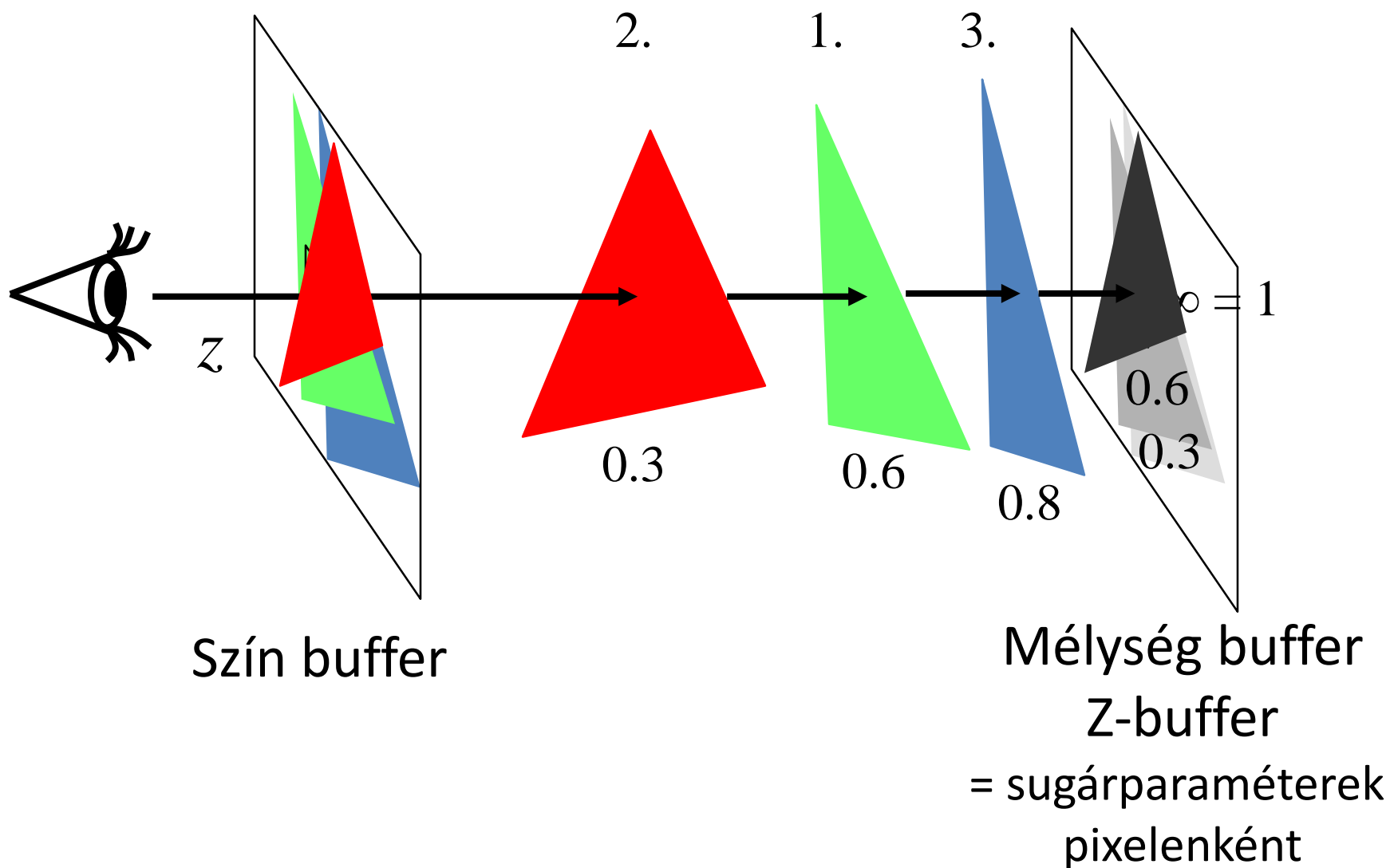
Lapok a nézeti irányban:

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

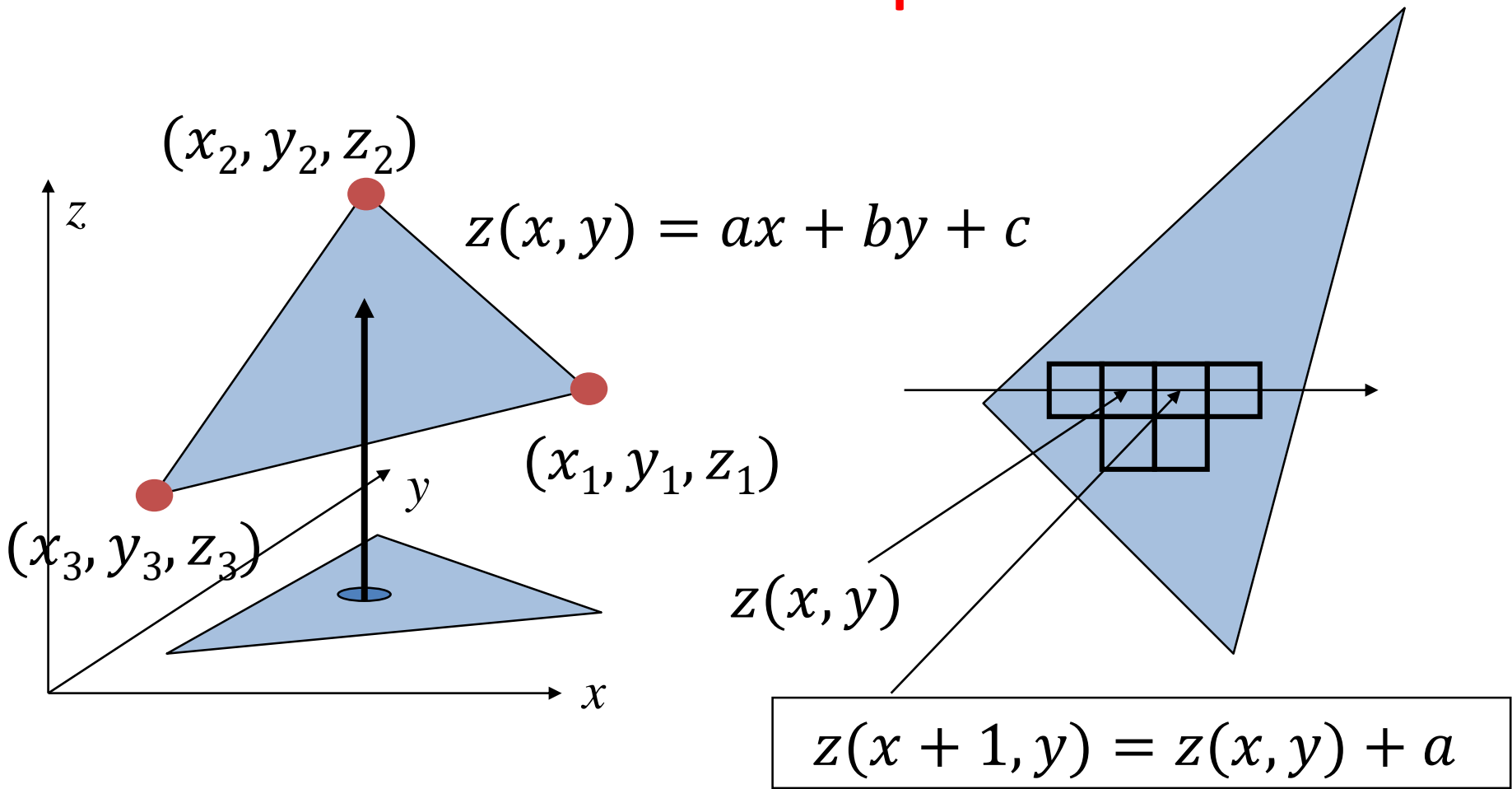
Feltételezés:

**Ha kívülről, akkor csúcsok óramutatóval megegyező körüljárásúak**

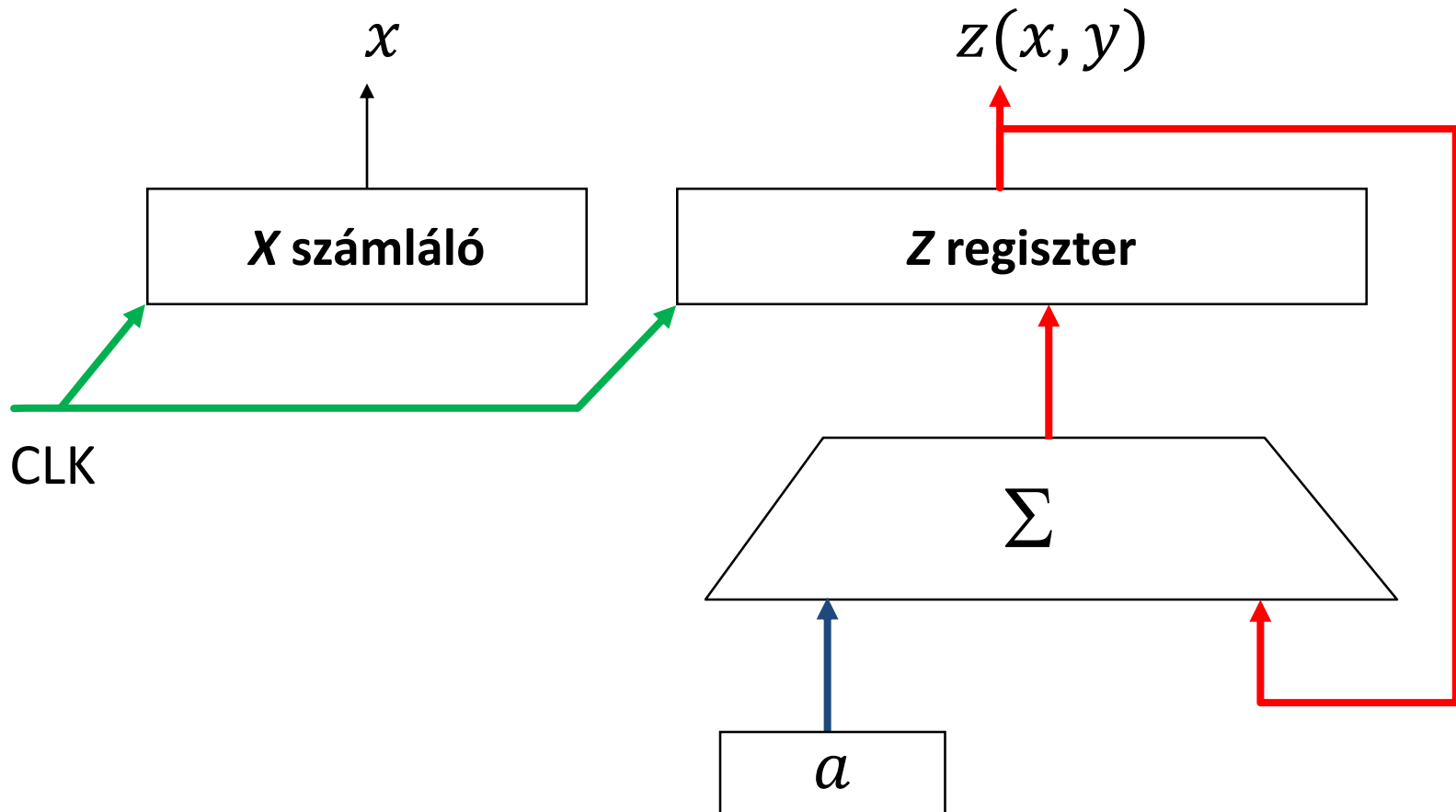
# Z-buffer algoritmus



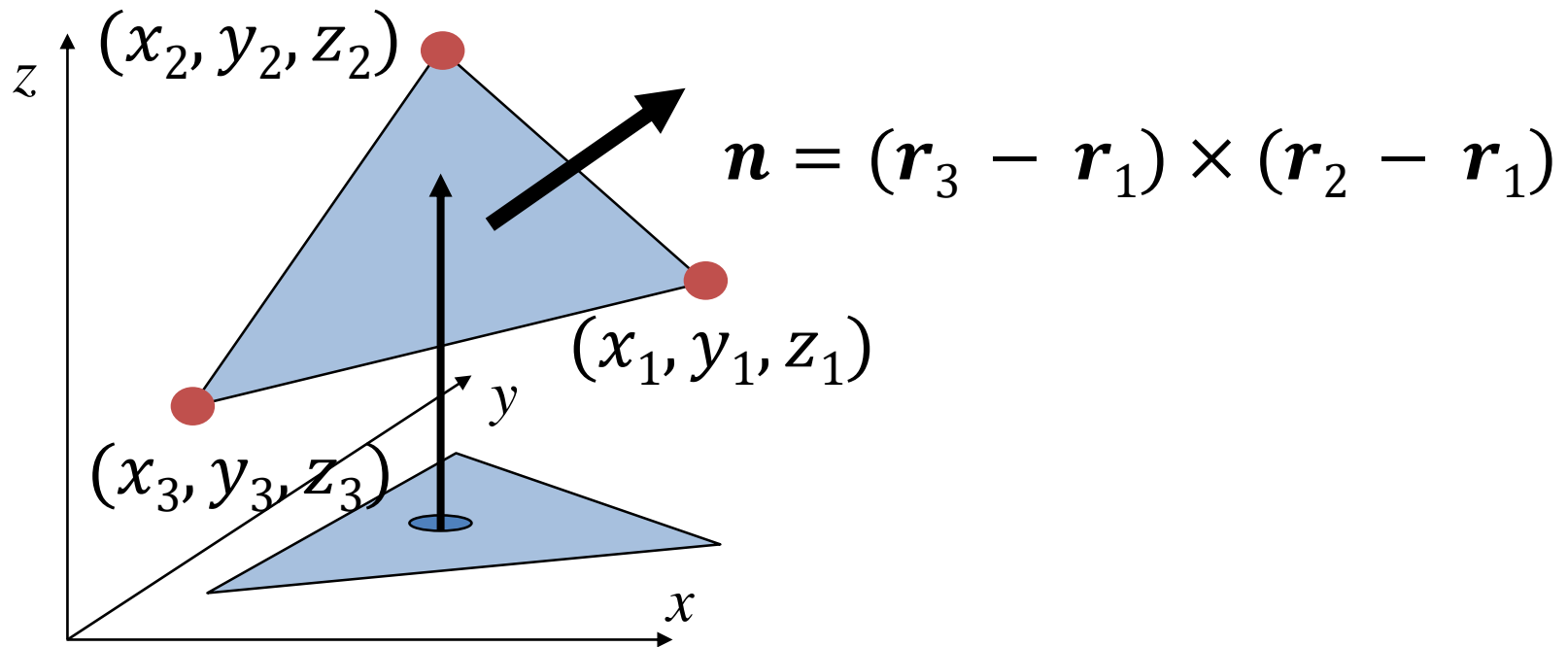
# Z: lineáris interpoláció



# Z-interpolációs hardver



# Triangle setup



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



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

# Takarás OpenGL-ben

```
int main(int argc, char * argv[]) {  
    ...  
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE |  
                        GLUT_DEPTH);  
    glEnable(GL_DEPTH_TEST); // z-buffer is on  
    glDisable(GL_CULL_FACE); // backface culling is off  
    ...  
}  
  
void onDisplay() {  
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
  
    rajzolás...  
  
    glutSwapBuffers(); // exchange the two buffers  
}
```

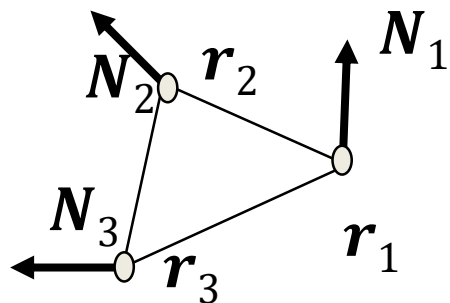
# Árnyalás: Lokális illumináció árnyék nélkül

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

- Koherencia: ne mindent pixelenként
- **Csúcspontonként:** belül az  $L$  „szín” interpolációja:  
Gouraud árnyalás (per-vertex shading)
- **Pixelenként:** belül a Normál ( $\mathbf{V}$ iew,  $\mathbf{L}$ ight) vektort interpoláljuk:  
Phong árnyalás (per-pixel shading)



# Per-vertex (Gouraud) árnyalás



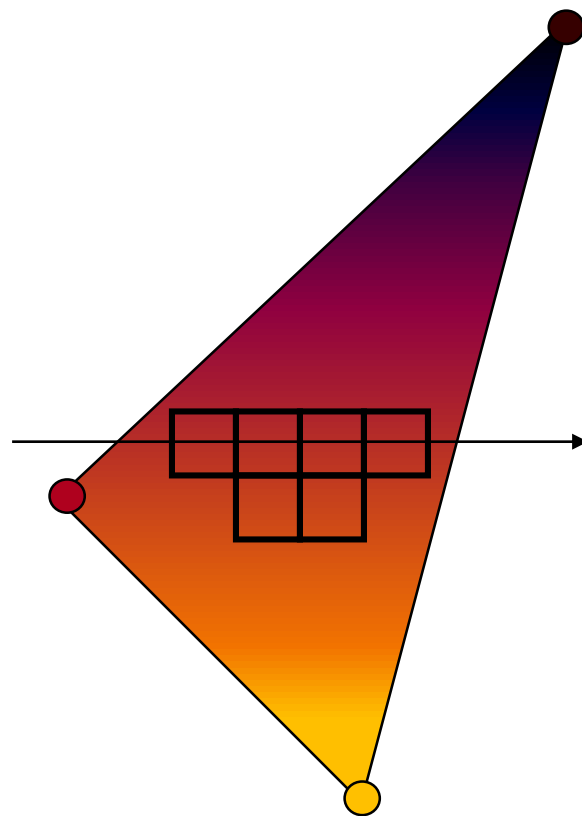
M: Modell

$$L(V) \approx \sum_l L_l^{in} * f_r(L_l, N, V) \cdot \cos^+ \theta_l^{in}$$



(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 = vec4(vtxPos, 1) * MVP; // to NDC

    vec4 wPos = vec4(vtxPos, 1) * M;
    vec3 L = normalize(wLiPos.xyz/wLiPos.w - wPos.xyz/wPos.w);
    vec3 V = normalize(wEye - wPos.xyz/wPos.w);
    vec4 wNormal = Minv * vec4(vtxNorm, 0);
    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

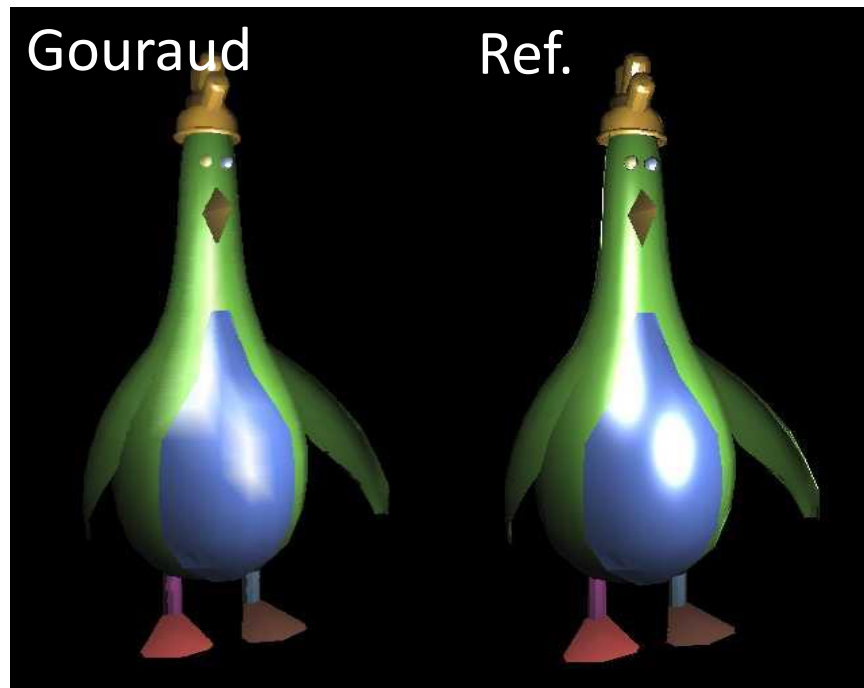
Gouraud



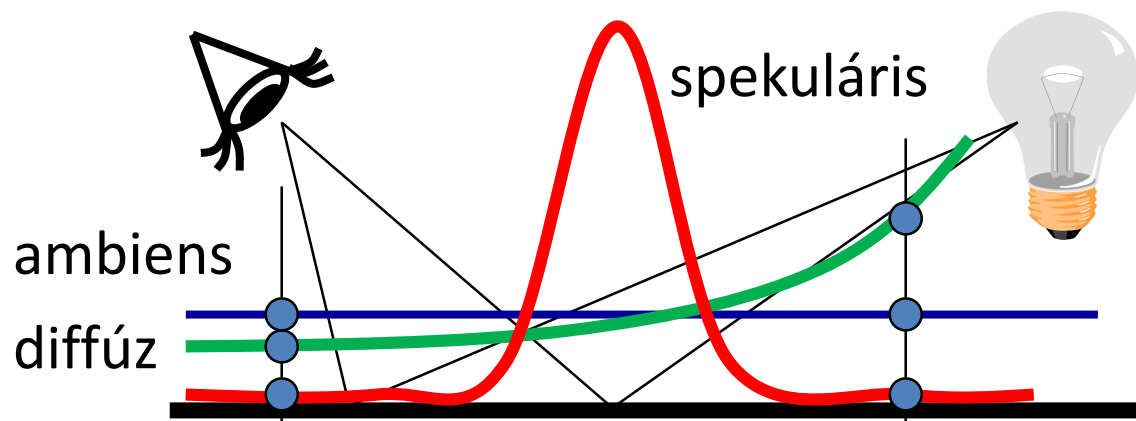
Ref.



Gouraud



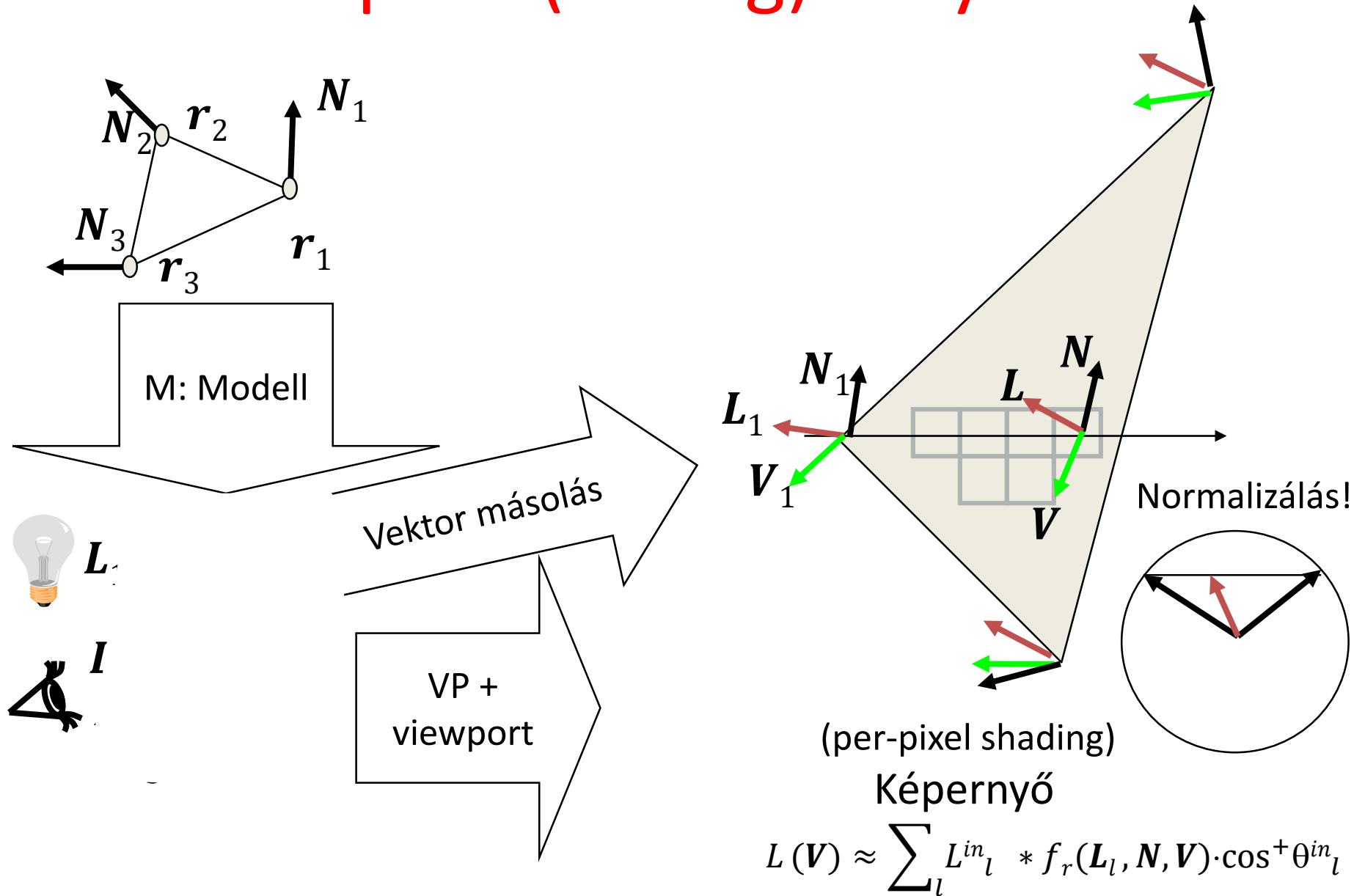
Ref.



## További bajok:

- anyagtulajdonság konstans
  - árnyék nincs
- különben a színt nem lehet interpolálni

# Per-pixel (Phong) árnyalás



# 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 mod 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 = vec4(vtxPos, 1) * MVP; // to NDC

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

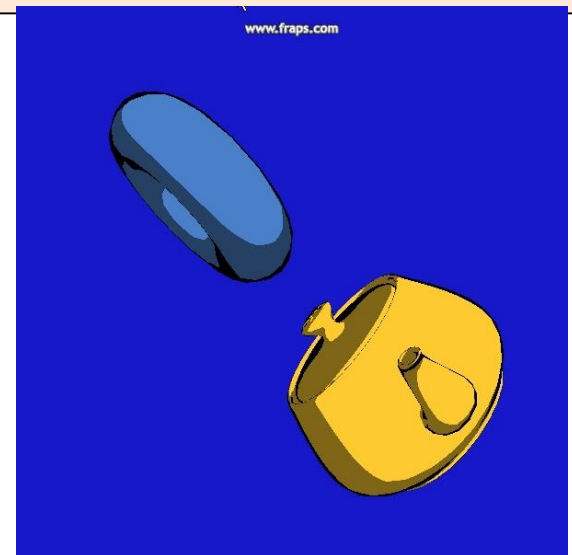
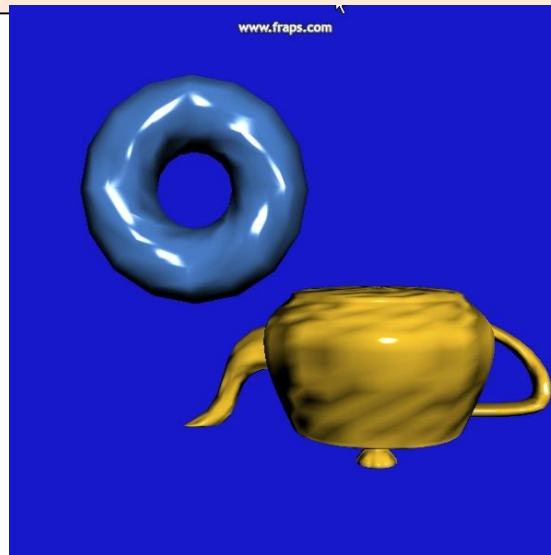
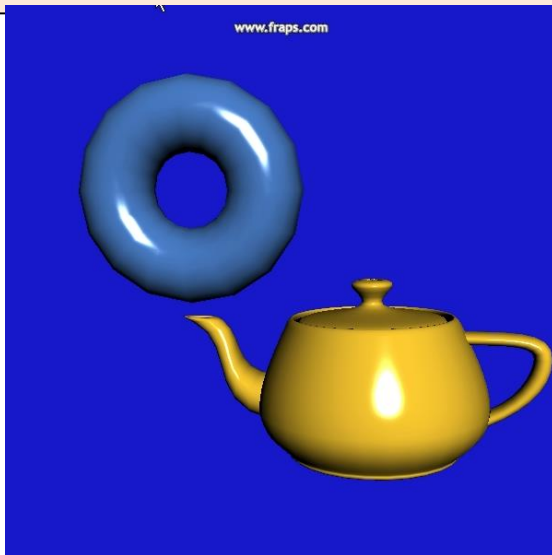
in  vec3 wNormal;        // interpolated world sp normal
in  vec3 wView;          // interpolated world sp view
in  vec3 wLight;         // interpolated world sp illum dir
out vec4 fragmentColor; // output goes to frame buffer

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 fragmentColor; // output goes to frame buffer

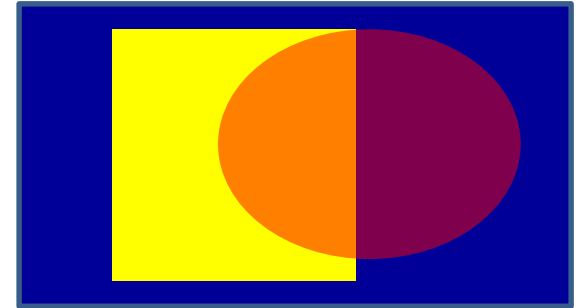
void main() {
    vec3 N = normalize(wNormal);
    vec3 V = normalize(wView);
    vec3 L = normalize(wLight);
    float y = (dot(N, L) > 0.5) ? 1 : 0.5;
    if (abs(dot(N, V)) < 0.2) fragmentColor = vec4(0, 0, 0, 1);
    else
        fragmentColor = vec4(y * kd, 1);
}
```





# Kompozitálás és átlátszóság

## Sorrend számít!



```
glEnable(GL_BLEND);
```

```
glBlendFunc(
```

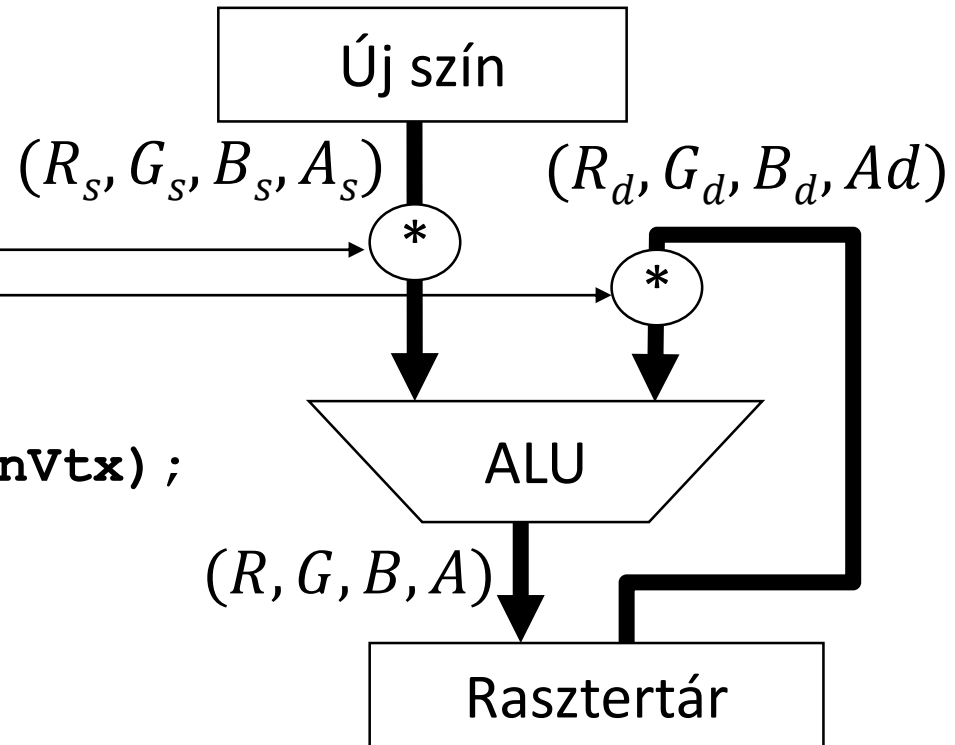
```
    GL_SRC_ALPHA,
```

```
    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

### 3. házi: Luxo Grandpa

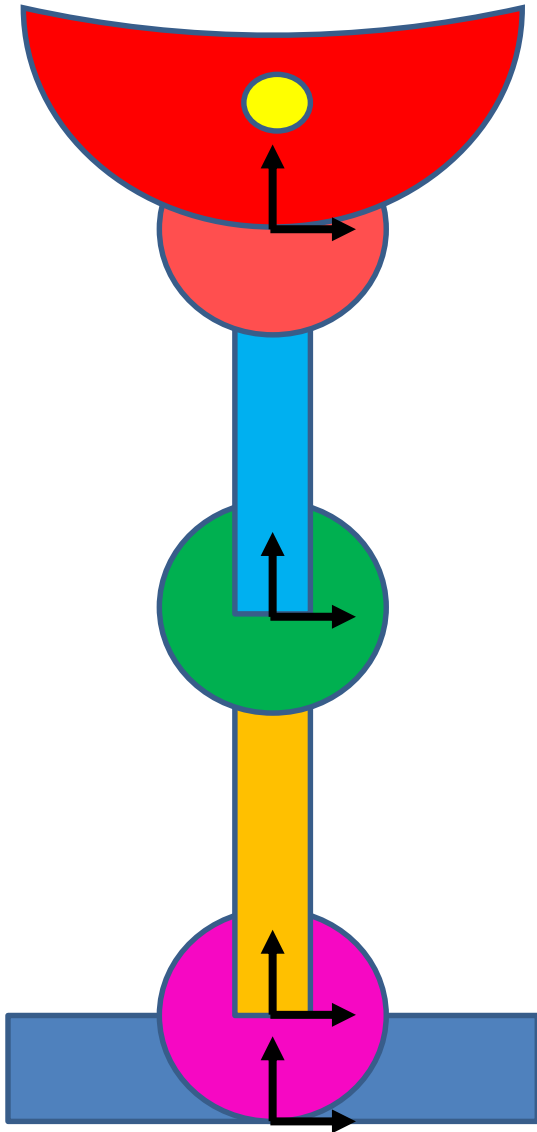
Luxo Junior-ból *Luxo Grandpa (LG)* lett, ezért új **inkrementális képszintézis programot** szentelünk neki.

LG egy síkon áll, a szerkezete (alulról felfelé): henger talp, gömbcsukló1, henger rúd1, gömbcsukló2, henger rúd2, gömbcsukló3, paraboloid, amelynek fókuszpontjában egy pontfényforrás ül.

A gömbcsuklók a koordinátatengelyektől eltérő tengelyek mentén folyamatosan elfordulnak. A szereplőket még egy pontfényforrás és ambiens fény világítja meg. A kamera forog LG körül. **Árnyéket csak a paraboloid vet a fókuszpontjában lévő pontforrásra.**

A szereplők diffúz-spekuláris típusú rücskös anyagúak.

# Luxo Grandpa



## Feladatok:

Téglalap, paraboloid, henger, gömb

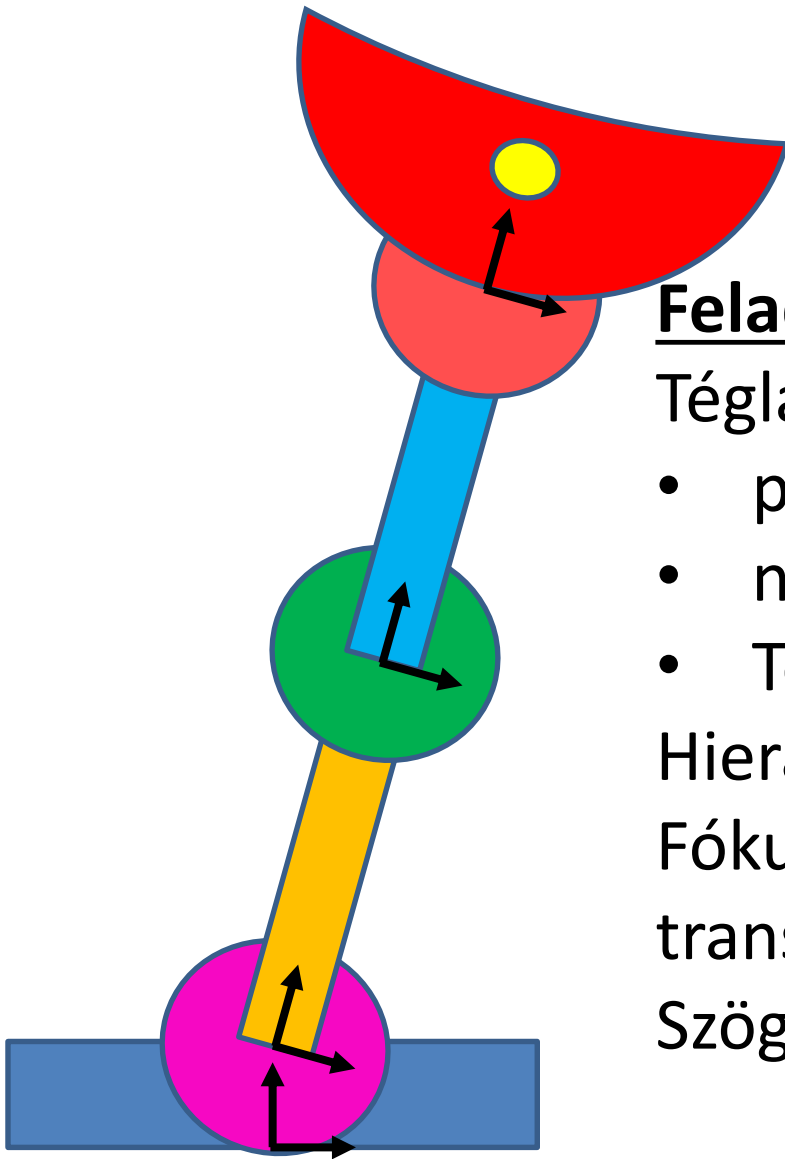
- parametrikus egyenlete
- normálvektora
- Tesszellációja

Hierarchikus transzformálás.

Fókuszpont és tengely hierarchikus transzformálása

Szög vizsgálat a lámpa hatásánál

# Luxo Grandpa



## Feladatok:

Téglalap, paraboloid, henger, gömb

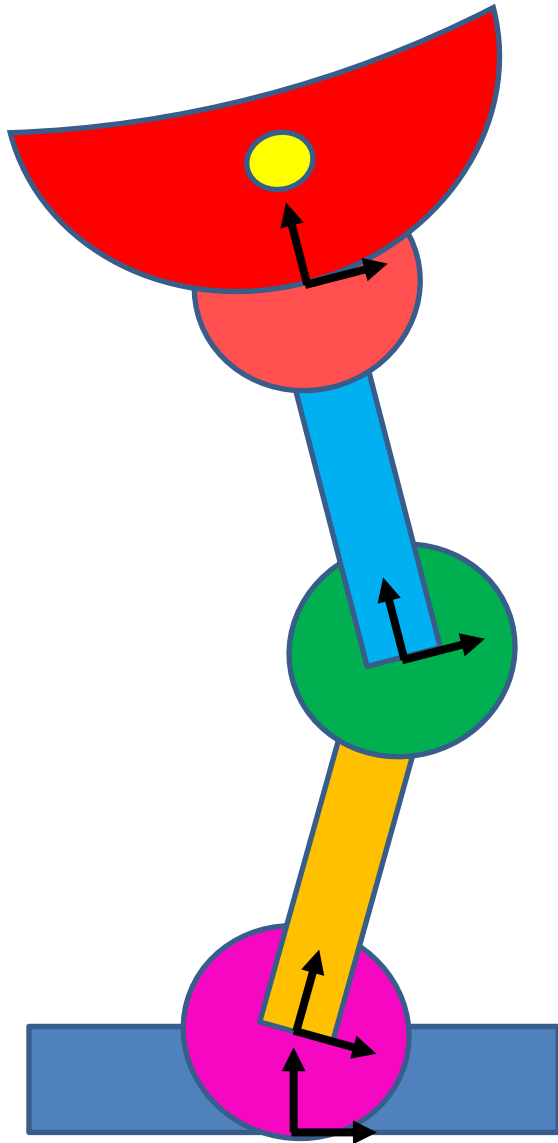
- parametrikus egyenlete
- normálvektora
- Tesszellációja

Hierarchikus transzformálás.

Fókuszpont és tengely hierarchikus transzformálása

Szög vizsgálat a lámpa hatásánál

# Luxo Grandpa



## Feladatok:

Téglalap, paraboloid, henger, gömb

- parametrikus egyenlete
- normálvektora
- Tesszellációja

Hierarchikus transzformálás.

Fókuszpont és tengely hierarchikus transzformálása

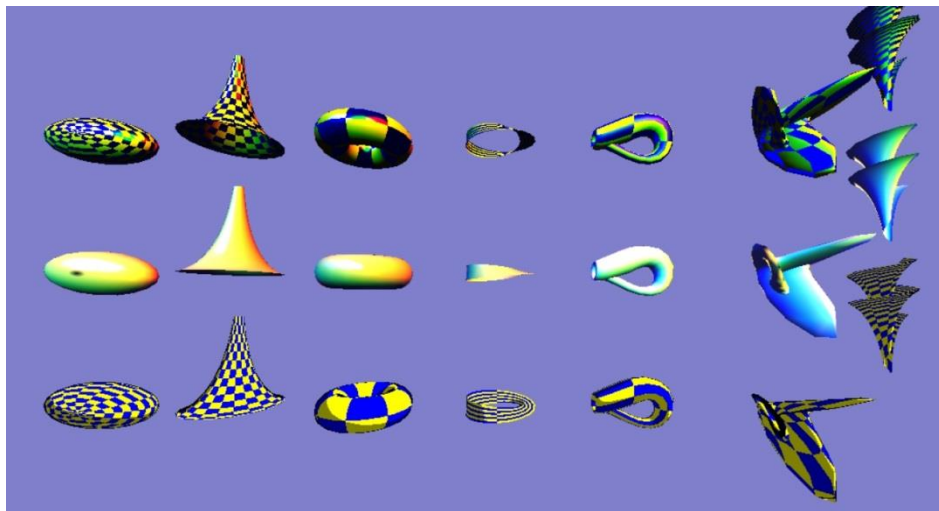
Szög vizsgálat a lámpa hatásánál

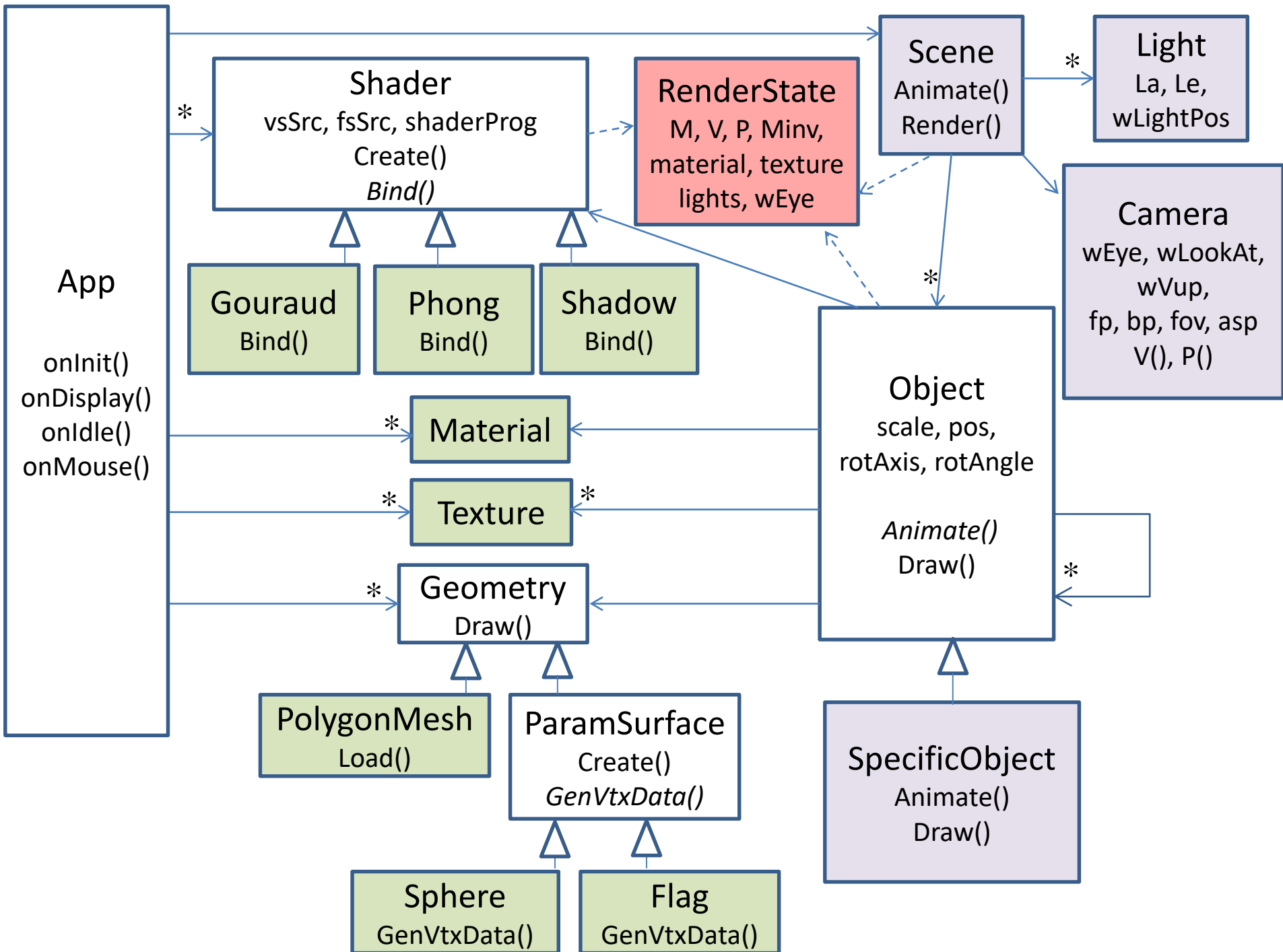


# Inkrementális 3D képszintézis

## 5. Program: 3D motorka

Szirmay-Kalos László







# Scene

```
class Scene {
    Camera camera;
    vector<Object *> objects;
    vector<Light *> lights;
public:
    void Render() {
        RenderState state;
        state.M = state.Minv = UnitMatrix();
        state.wEye = camera.wEye;
        state.V = camera.V();
        state.P = camera.P();
        state.lights = lights;
        for (Object * obj : objects) obj->Draw(state);
    }
    void Animate(float ts, float te) {
        for (Object * obj : objects) obj->Animate(ts, te);
    }
};
```

# Object

```
class Object {
    Shader *    shader;
    Material * material;
    Texture *   texture;
    Geometry *  geometry;
    vec3 scale, pos, rotAxis;
    float rotAngle;
    vector<Object *> children;

public:
    virtual void Draw(RenderState state) {
        state.M = Scale(scale.x, scale.y, scale.z) *
            Rotate(rotAngle, rotAxis.x, rotAxis.y, rotAxis.z) *
            Translate(pos.x, pos.y, pos.z) * state.M;
        state.Minv = state.Minv *
            Translate(-pos.x, -pos.y, -pos.z) *
            Rotate(-rotAngle, rotAxis.x, rotAxis.y, rotAxis.z) *
            Scale(1/scale.x, 1/scale.y, 1/scale.z);
        state.material = material; state.texture = texture;
        shader->Bind(state); // uniform változók beállítása
        geometry->Draw(); // háromszögek végigmennek a pipeline-on
        for (Object * child : children) child->Draw(state);
    }
    virtual void Animate(float ts, float te) {}
};
```

Érték szerinti paraméterátadás:  
Objektumok nem zavarják egymást

# Shader

```
struct Shader {
    unsigned int shaderProg;

    void Create(const char * vsSrc,
               const char * fsSrc, const char * fsOuput) {
        unsigned int vs = glCreateShader(GL_VERTEX_SHADER);
        glShaderSource(vs,1,&vsSrc,NULL); glCompileShader(vs);
        unsigned int fs = glCreateShader(GL_FRAGMENT_SHADER);
        glShaderSource(fs,1,&fsSrc,NULL); glCompileShader(fs);
        shaderProg = glCreateProgram();
        glAttachShader(shaderProg, vs);
        glAttachShader(shaderProg, fs);

        glBindFragDataLocation(shaderProg, 0, fsOuput);
        glLinkProgram(shaderProg);
    }

    virtual void Bind(RenderState& state) {
        glUseProgram(shaderProg);
    }
};
```

# ShadowShader



```
class ShadowShader : public Shader {
    const char * vsSrc = R"(
        uniform mat4 MVP;
        layout(location = 0) in vec3 vtxPos;
        void main() { gl_Position = vec4(vtxPos, 1) * MVP; }
    )";
    const char * fsSrc = R"(
        out vec4 fragmentColor;
        void main() { fragmentColor = vec4(0, 0, 0, 1); }
    )";
public:
    ShadowShader() {
        Create(vsSrc, fsSrc, "fragmentColor");
    }
    void Bind(RenderState& state) {
        glUseProgram(shaderProg);
        mat4 MVP = state.M * state.V * state.P;
        MVP.SetUniform(shaderProg, "MVP");
    }
};
```