Inkrementális 3D képszintézis

Szirmay-Kalos László

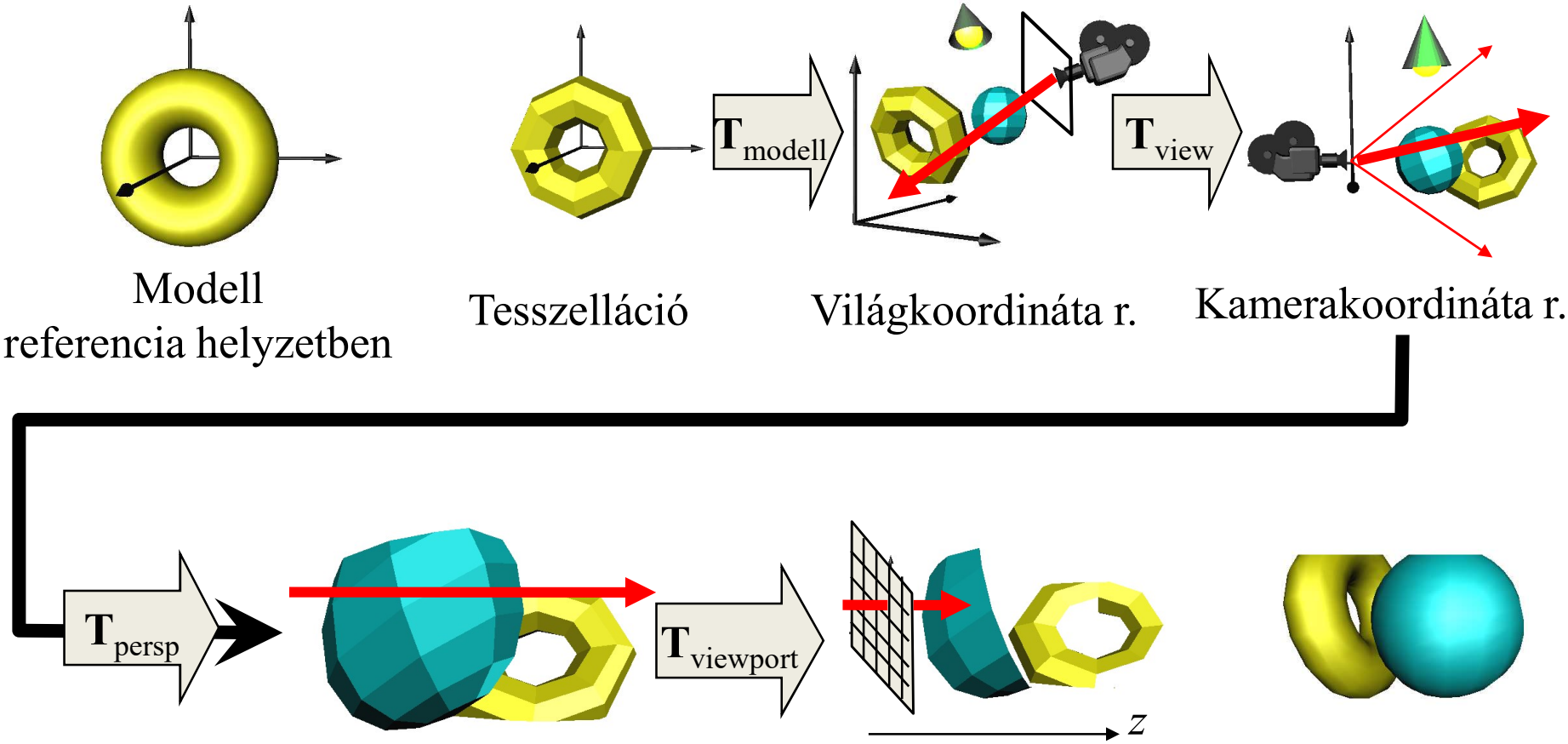
Inkrementális képszintézis

- Sugárkövetés számítási idő $\propto$

  Pixelszám × Objektumszám × (Fényforrás szám+1)

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

3D inkrementális képszintézis



Modell
referencia helyzetben

Tesszelláció

Világkoordináta r.

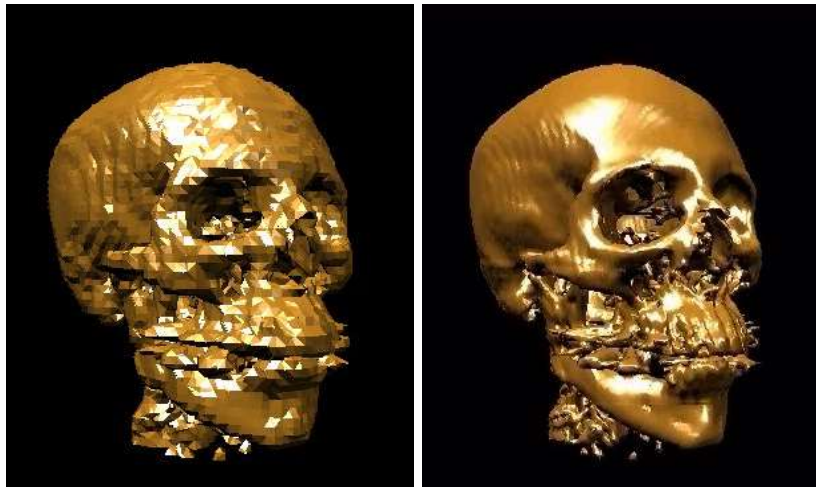
Kamerakoordináta r.

Normalizált
eszközkoordináta r.
Vágás

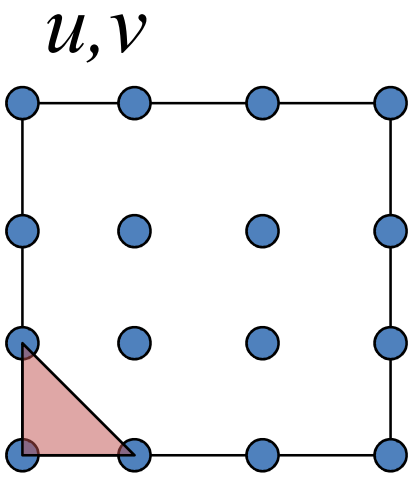
Képernyőkoordináta r.
Láthatóság + vetítés
Takarás és vetítés triviális!

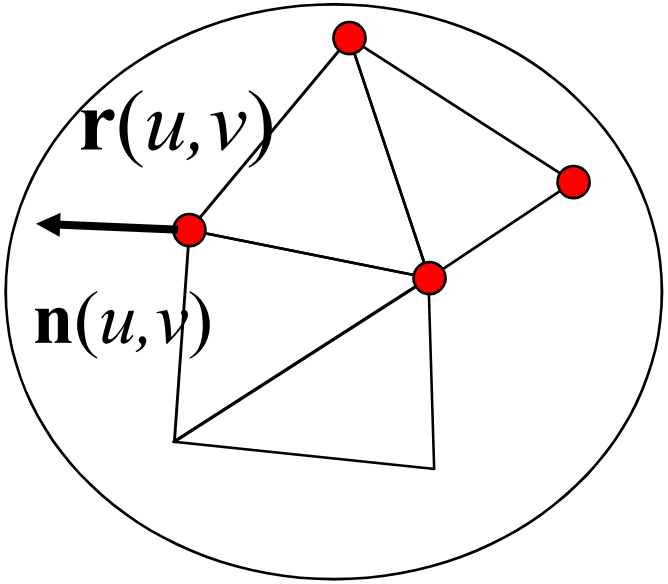
Árnyalt kép

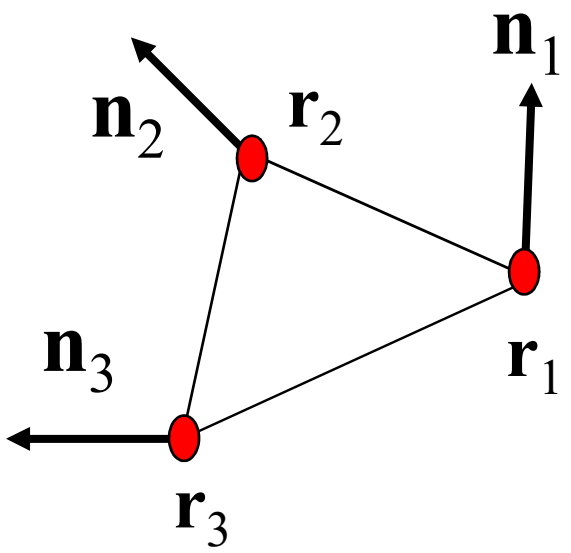
Tesszelláció



- Felületi pontok: $\mathbf{r}_{n,m} = \mathbf{r}(u_n, v_m)$
- Normálvektor: $\mathbf{n}(u_n, v_m) = \dfrac{\partial \mathbf{r}(u,v)}{\partial u} \times \dfrac{\partial \mathbf{r}(u,v)}{\partial v}$
- „Paramétertérben szomszédos” pontokból háromszögek

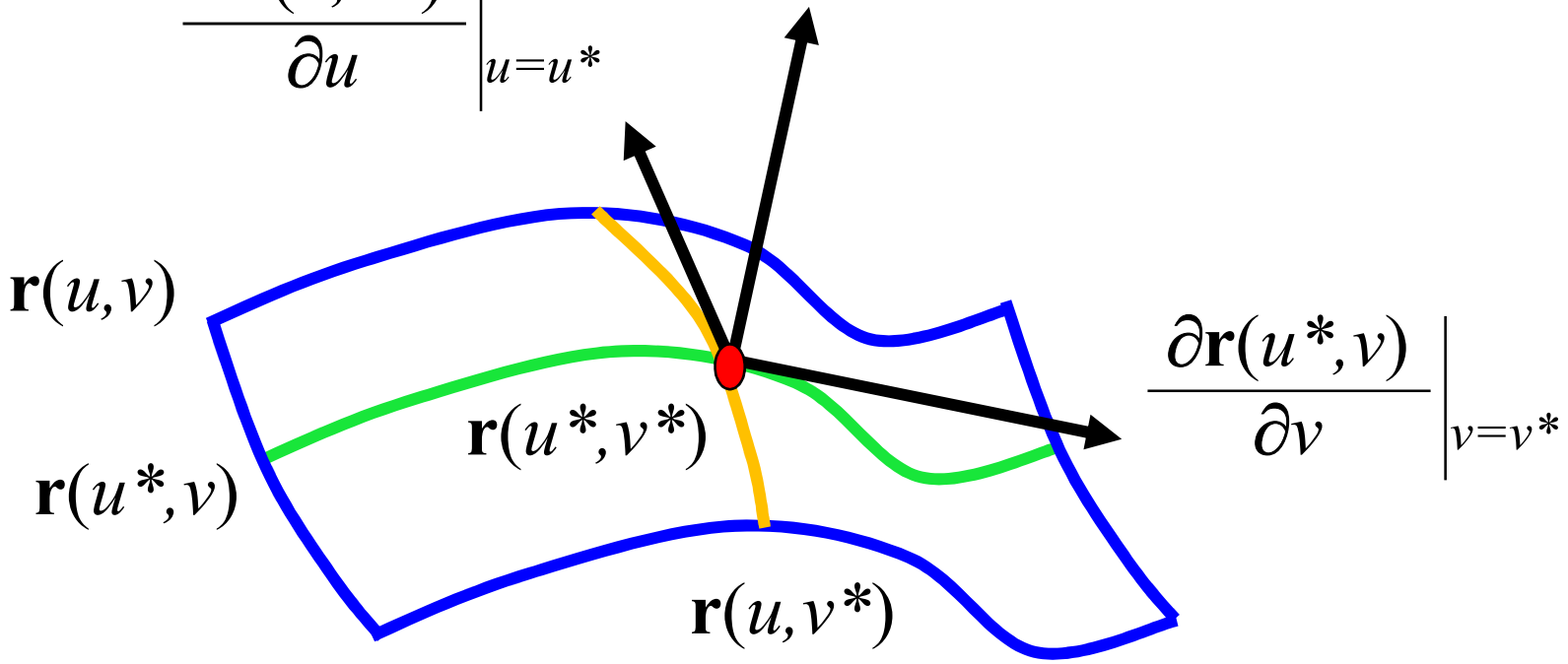






Parametrikus felületek normálvektora

$$\mathbf{n}(u,v) = \left. \frac{\partial \mathbf{r}(u,v)}{\partial u} \times \frac{\partial \mathbf{r}(u,v)}{\partial v} \right|_{\substack{u=u^* \\ v=v^*}}$$



Objektumok az GPU-nak

```
struct Geometry {
  unsigned int vao, nVtx;

  Geometry( ) { // Ilyen ne legyen globális változó!
     glGenVertexArrays(1, &vao);
     glBindVertexArray(vao);
  }
  void Draw() {
     glBindVertexArray(vao);
     glDrawArrays(GL_TRIANGLES, 0, nVtx);
  }
};

struct Vd { // VertexData, ArrayOfStruct: 3 + 3 + 2 float
  vec3 position, normal;
  float u, v;
};

struct ParamSurface : Geometry {
  virtual Vd r(float u, float v) = 0;
  void Create(int N, int M);
};
```

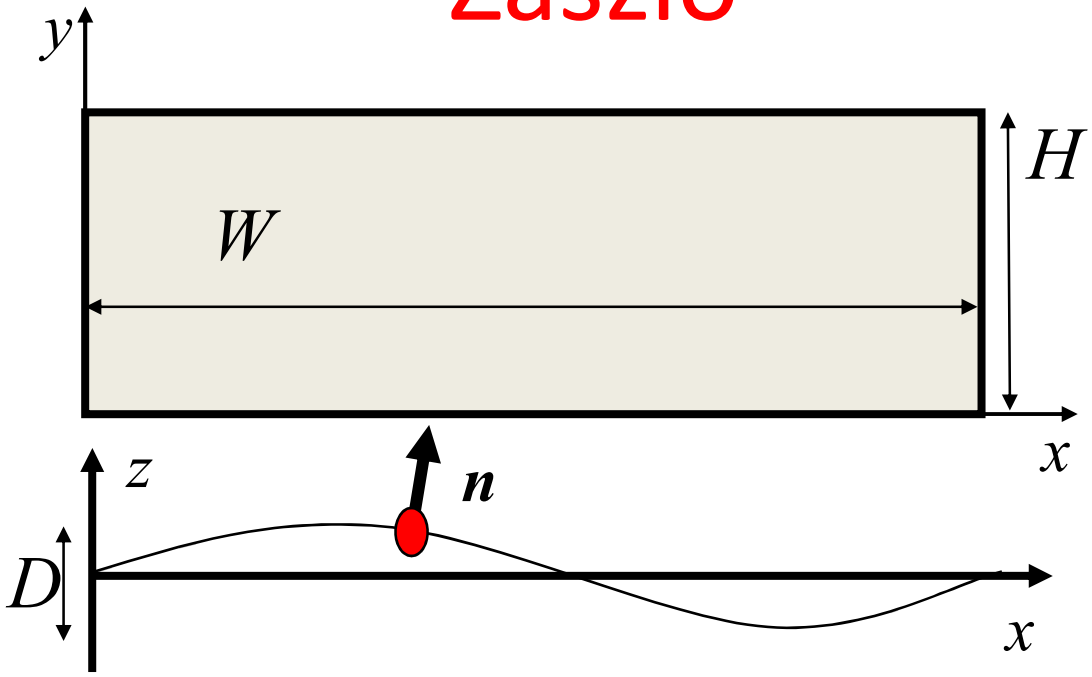
Parametrikus felület GPU-nak

```
void ParamSurface::Create(int N, int M) {
   nVtx = N * M * 6;
   unsigned int vbo;
   glGenBuffers(1, &vbo); glBindBuffer(GL_ARRAY_BUFFER, vbo);

   Vd *vtx = new Vd[nVtx], *pVtx = vtx;
   for (int i = 0; i < N; i++) for (int j = 0; j < M; j++) {
      *pVtx++ = r((float)i / N, (float)j / M);
      *pVtx++ = r((float)(i + 1) / N, (float)j / M);
      *pVtx++ = r((float)i / N, (float)(j + 1) / M);
      *pVtx++ = r((float)(i + 1) / N, (float)j / M);
      *pVtx++ = r((float)(i + 1) / N, (float)(j + 1) / M);
      *pVtx++ = r((float)i / N, (float)(j + 1) / M);
   }
   glBufferData(GL_ARRAY_BUFFER, nVtx * sizeof(Vd), vtx, GL_STATIC_DRAW);

   glEnableVertexAttribArray(0);  // AttribArray 0 = POSITION
   glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vd), (void*)0);
   glEnableVertexAttribArray(1);  // AttribArray 1 = NORMAL
   glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,
                         sizeof(Vd),(void*)offsetof(Vd, normal));
   glEnableVertexAttribArray(2);  // AttribArray 2 = UV
   glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE,
                         sizeof(Vd),(void*)offsetof(Vd, u));
}
```

Gömb

```
class Sphere : public ParamSurface {
   vec3 center;
   float radius;
public:
   Sphere(vec3 c, float r) : center(c), radius(r) {
      Create(16, 8); // tessellation level
   }

   Vd r(float u, float v) {
      Vd vd;
      vd.normal = vec3(cos(u*2*M_PI) * sin(v*M_PI),
                       sin(u*2*M_PI) * sin(v*M_PI),
                       cos(v*M_PI));
      vd.position = vd.normal * radius + center;
      vd.u = u; vd.v = v;
      return vd;
   }
};
```

Zászló



$$\boldsymbol{r}(u,v) = [u \cdot W, \quad v \cdot H, \quad \sin(K \cdot u \cdot \mathrm{PI} + \mathrm{phase}) \cdot D \quad]$$

$$\partial \boldsymbol{r}/\partial u = \begin{bmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ W, & 0, & K \cdot \mathrm{PI} \cdot \cos(K \cdot u \cdot \mathrm{PI} + \mathrm{phase}) \cdot D \\ 0, & H, & 0 \end{bmatrix}$$

$$\partial \boldsymbol{r}/\partial v =$$

$$\boldsymbol{n}(u,v) = \partial \boldsymbol{r}/\partial u \times \partial \boldsymbol{r}/\partial v = [-K \cdot \mathrm{PI} \cdot H \cdot \cos(K \cdot u \cdot \mathrm{PI} + \mathrm{phase}) \cdot D, 0, W \cdot H]$$

Zászló

```
class Flag : public ParamSurface {
   float W, H, D, K, phase;
public:
   Flag(float w, float h, float d, float k, float p)
   : W(w), H(h), D(d), K(k), phase(p) {
      Create(60, 40); // tessellation level
   }

   Vd r(float u, float v) {
      Vd vd;
      float angle = u * K * M_PI + phase;
      vd.position = vec3(u * W, v * H, sin(angle)*D);
      vd.normal = vec3(-K * M_PI * cos(angle) * D, 0, W);
      vd.u = u; vd.v = v;
  }
};
```

Transzformációk

Modellezési transzformáció:

$$[\mathbf{r},1]\,\mathbf{T}_{\text{Model}} = [\mathbf{r}_{\text{world}},1]$$

$$[\mathbf{n},0]\,(\mathbf{T}_{\text{Model}}^{-1})^{\text{T}} = [\mathbf{n}_{\text{world}},d]$$

Kamera transzformáció:

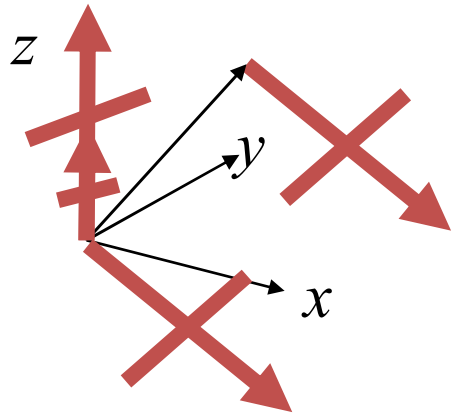
$$[\mathbf{r}_{\text{world}},1]\,\mathbf{T}_{\text{View}} = [\mathbf{r}_{\text{camera}},\ 1]$$

Perspektív transzformáció:

$$[\mathbf{r}_{\text{camera}},1]\,\mathbf{T}_{\text{Persp}} = [\mathbf{r}_{\text{screen}}h,\ h]$$

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

Modellezési transzformáció



1. skálázás: *sx, sy, sz*
2. orientáció: *wx, wy, wz,* α
3. pozíció: *px, py, pz*

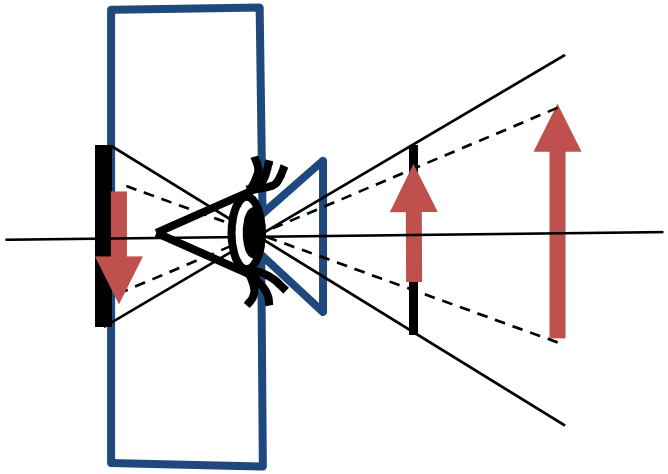
$$\mathbf{T}_M = \begin{bmatrix} sx & & & \\ & sy & & \\ & & sz & \\ & & & 1 \end{bmatrix} \begin{bmatrix} & & & \\ & \mathbf{R} & & \\ & & & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ px & py & pz & 1 \end{bmatrix}$$

$$\boldsymbol{r}' = \boldsymbol{r}\cos(\alpha) + \boldsymbol{w}^0(\boldsymbol{r}\cdot\boldsymbol{w}^0)(1-\cos(\alpha)) + \boldsymbol{w}^0\times\boldsymbol{r}\sin(\alpha)$$

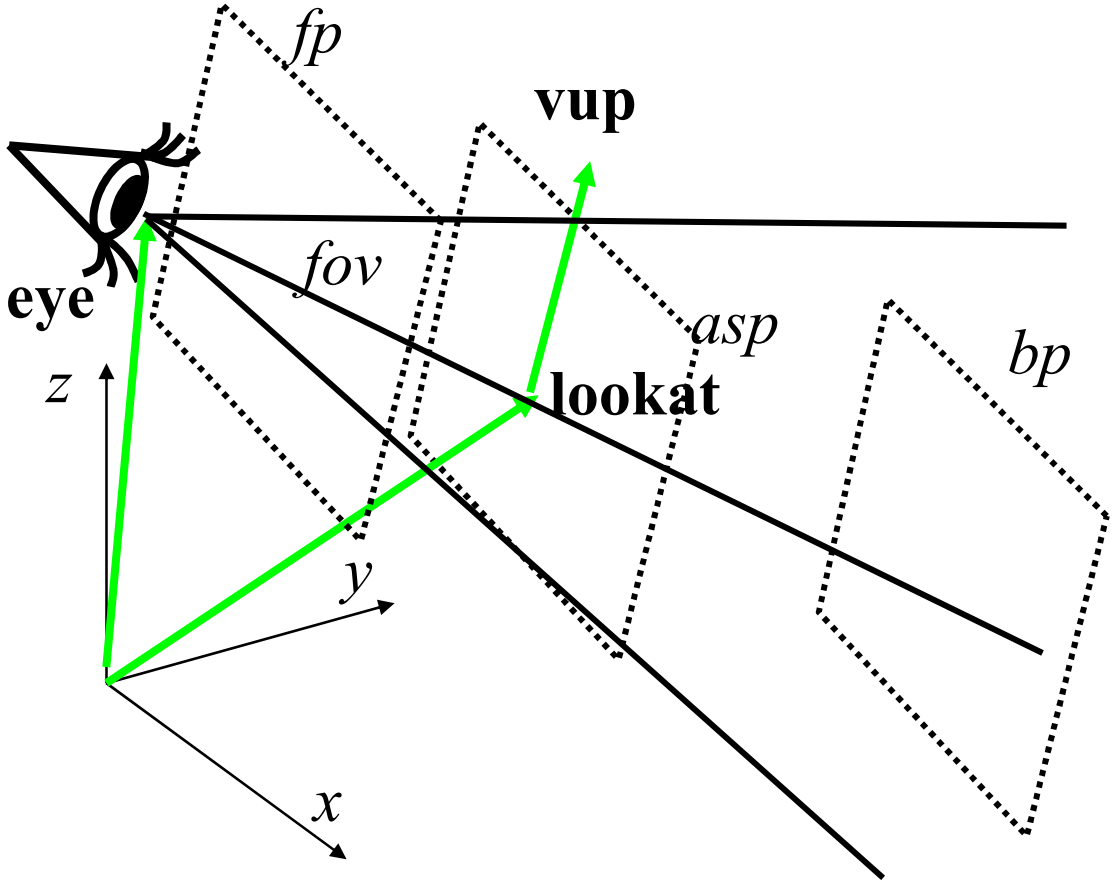
4x4-es mátrix a CPU-n

```
struct mat4 {
    float m[4][4];
    mat4(float m00,…, float m33) { … }
    mat4 operator*(const mat4& right);

    void SetUniform(unsigned shaderProg, char * name) {
       int loc = glGetUniformLocation(shaderProg, name);
       glUniformMatrix4fv(loc, 1, GL_TRUE, &m[0][0]);
    }
};

mat4 Translate(float tx, float ty, float tz) {
    return mat4(1,   0,   0,   0,
                0,   1,   0,   0,
                0,   0,   1,   0,
               tx,  ty,  tz,   1);
}
mat4 Rotate(float angle,float wx,float wy,float wz) {…}
mat4 Scale(sx, sy, sz) {…}
```

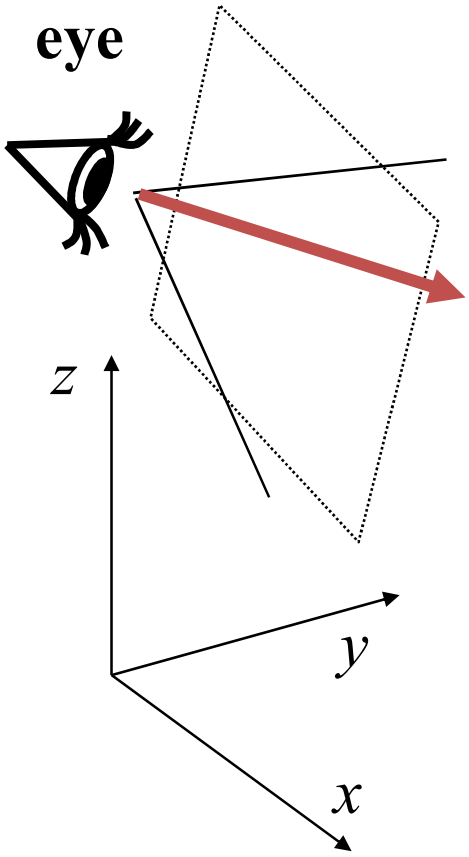
Kamera modell



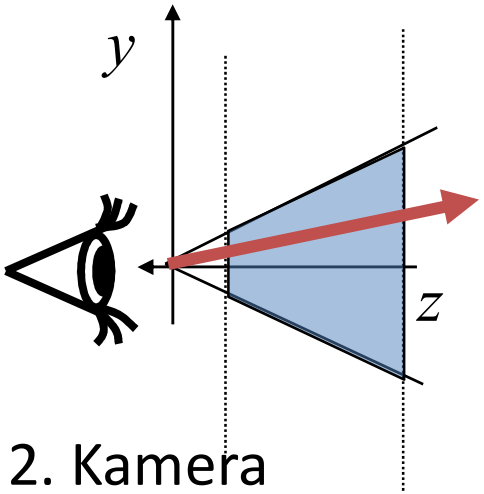
Mi: Camera obscura



Világból a képernyőre

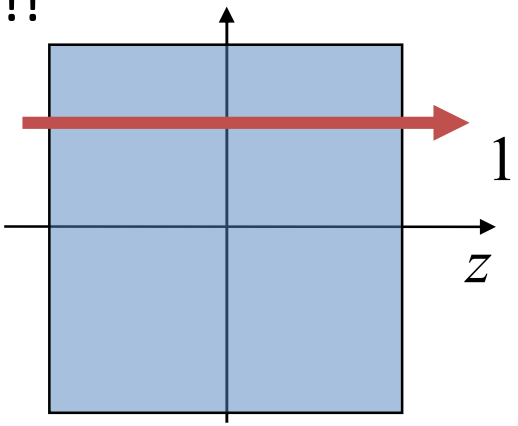


1. Világ

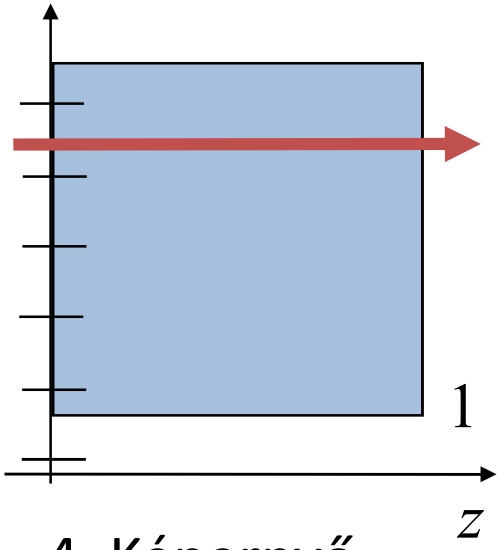


2. Kamera

Bal!!!

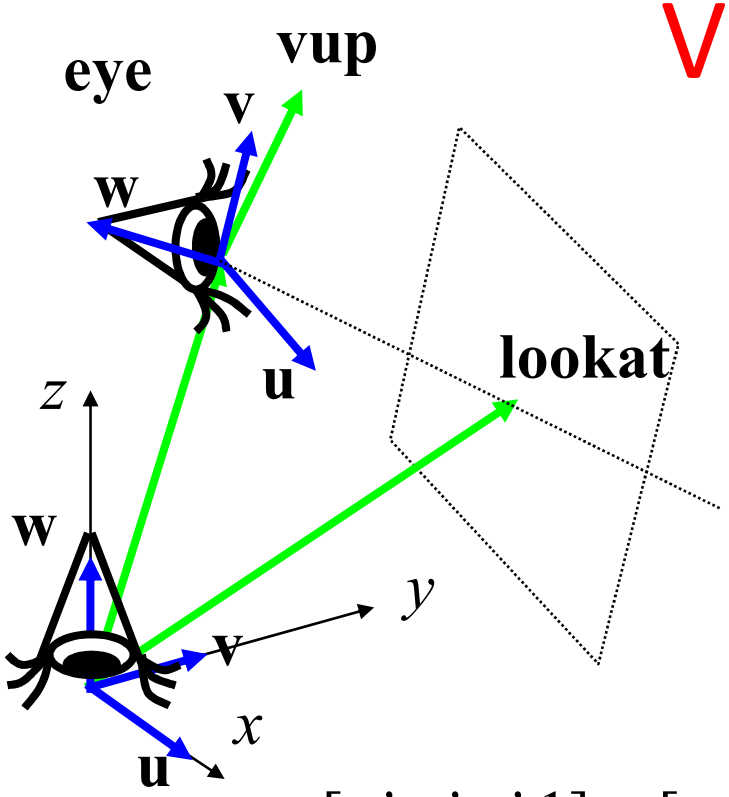


3. Normalizált képernyő



4. Képernyő

View transzformáció

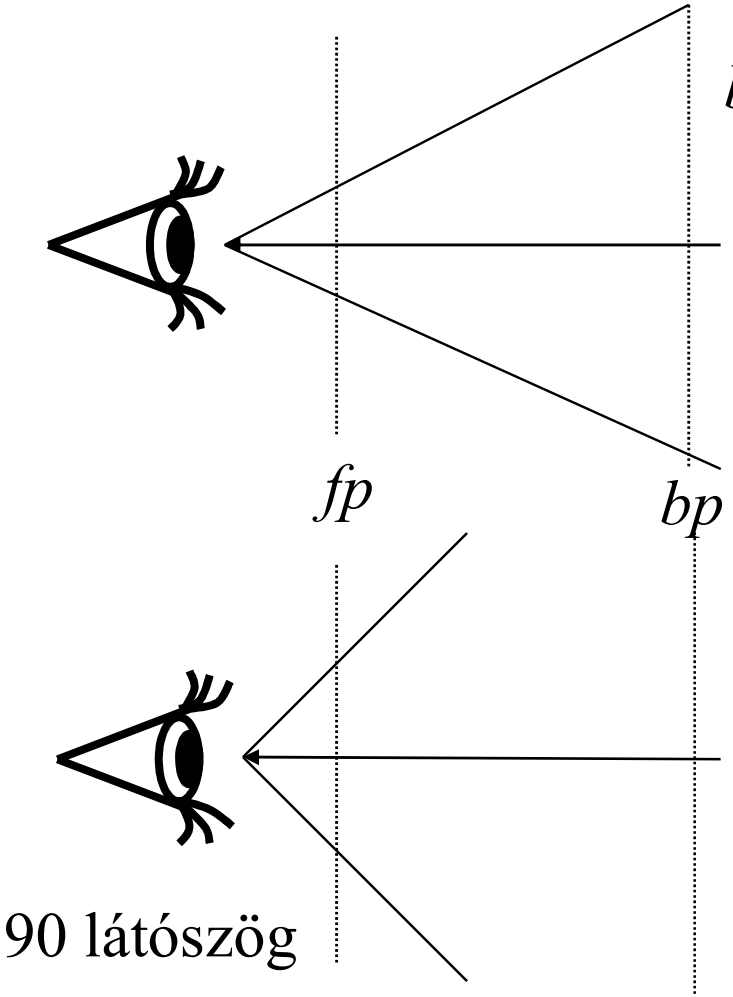


$$\mathbf{w} = (\mathbf{eye}-\mathbf{lookat})/|\mathbf{eye}-\mathbf{lookat}|$$
$$\mathbf{u} = (\mathbf{vup} \times \mathbf{w})/|\mathbf{w} \times \mathbf{vup}|$$
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

$$[x',y',z',1] = [x,y,z,1] \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\mathrm{eye}_x & -\mathrm{eye}_y & -\mathrm{eye}_z & 1 \end{pmatrix} \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}$$

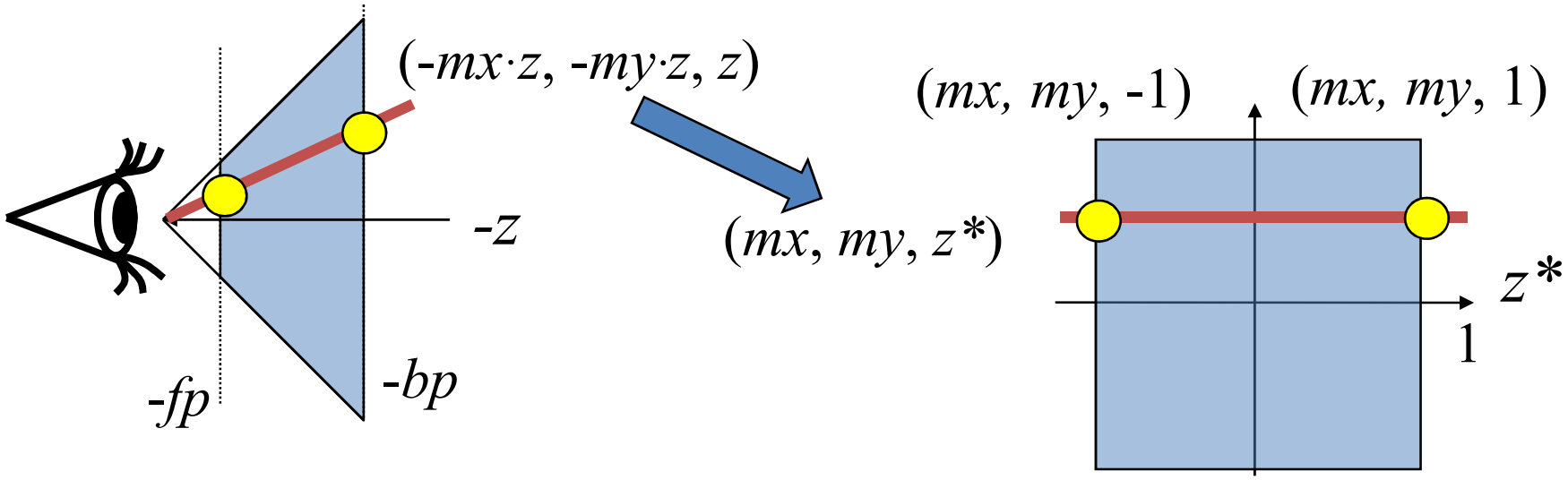
$$\begin{pmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Látószög normalizálás



$$\mathbf{T}_{\text{norm}} = \begin{pmatrix} 1/(\text{tg}(fov/2)\cdot asp) & 0 & 0 & 0 \\ 0 & 1/\text{tg}(fov/2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

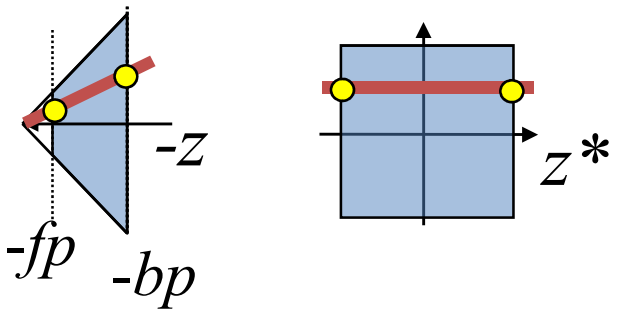
Normalizálás utáni perspektív transzformáció



$(-mx \cdot z, -my \cdot z, z) \Rightarrow (mx, my, z^*)$

$[-mx \cdot z, -my \cdot z, z, 1] \Rightarrow [mx, my, z^*, 1] \sim [-mx \cdot z, -my \cdot z, -z \cdot z^*, -z]$

Perspektív transzformáció



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

$$[-mx\cdot z,\ -my\cdot z,\ z,\ 1] \Rightarrow [-mx\cdot z,\ -my\cdot z,\ -z\cdot z^*,\ -z]$$

$$-z\cdot z^* = \alpha\cdot z + \beta \quad \Rightarrow \quad \boxed{z^* = -\alpha - \beta/z}$$

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

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

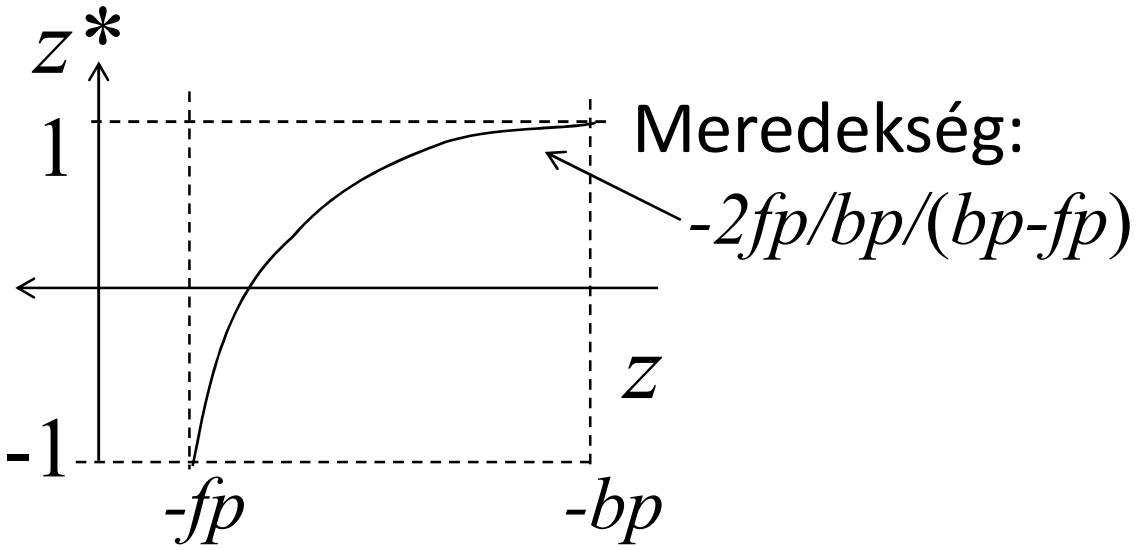
Z-fighting

fp/bp nem lehet kicsi!

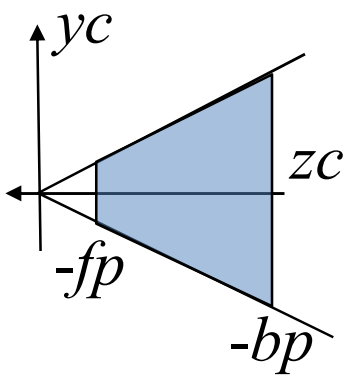
$z^* = -\alpha - \beta/z$

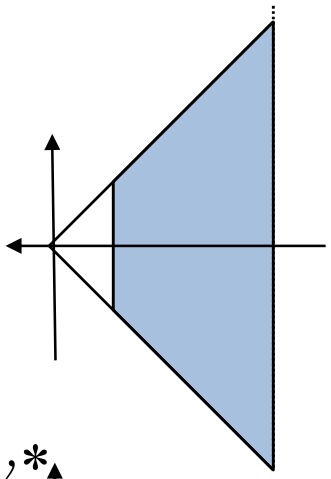
$\alpha = -(fp+bp)/(bp-fp)$

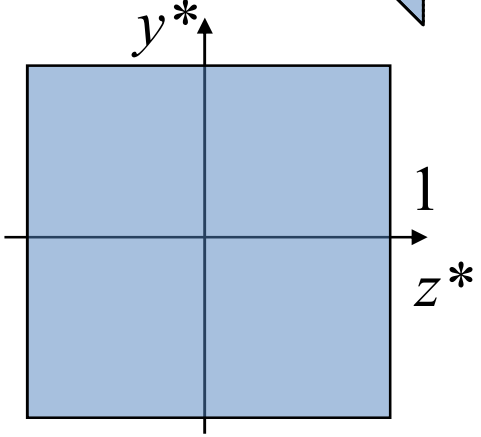
$\beta = -2fp \cdot bp/(bp-fp)$



Projekció (perspektív) transzformáció







$$\begin{pmatrix} 1/(\mathrm{tg}(fov/2)\cdot asp) & 0 & 0 & 0 \\ 0 & 1/\mathrm{tg}(fov/2) & 0 & 0 \\ 0 & 0 & -(fp+bp)/(bp-fp) & -1 \\ 0 & 0 & -2fp\cdot bp/(bp-fp) & 0 \end{pmatrix}$$

Kamera koord.

$h = -zc$

$$[X_h, Y_h, Z_h, h] = [xc, yc, zc, 1]\ \mathbf{T}_{\text{persp}}$$

$$[x^*, y^*, z^*, 1] = [X_h/h,\ Y_h/h,\ Z_h/h, 1]$$

Perspektív torzítás

Camera osztály

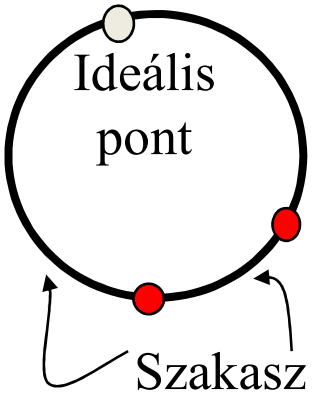
```
class Camera {
   vec3  wEye, wLookat, wVup; // extrinsic parameters
   float fov, asp, fp, bp;    // intrinsic parameters
public:
   mat4 V() { // view matrix
      vec3 w = (wEye - wLookat).normalize();
      vec3 u = cross(wVup, w).normalize();
      vec3 v = cross(w, u);
      return Translate(-wEye.x, -wEye.y, -wEye.z) *
             mat4( u.x,  v.x,  w.x,  0.0f,
                   u.y,  v.y,  w.y,  0.0f,
                   u.z,  v.z,  w.z,  0.0f,
                   0.0f, 0.0f, 0.0f, 1.0f );
   }
   mat4 P() { // projection matrix
      float sy = 1/tan(fov/2);
      return mat4(sy/asp, 0.0f,  0.0f,                0.0f,
                  0.0f,   sy,    0.0f,                0.0f,
                  0.0f,   0.0f, -(fp+bp)/(bp - fp), -1.0f,
                  0.0f,   0.0f, -2*fp*bp/(bp - fp),  0.0f);
   }
};
```

Transzformációk a GPU-n

```
const char *vertexSource = R"(
uniform mat4 M, Minv, MVP;
in vec3 vtxPos;  // Attrib array 0
in vec3 vtxNorm; // Attrib array 1
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);
})";
```

```
void Draw() {
   mat4 M = Scale(scale.x, scale.y, scale.z) *
            Rotate(rotAng, rotAxis.x, rotAxis.y, rotAxis.z) *
            Translate(pos.x, pos.y, pos.z);
   mat4 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);
   mat4 MVP = M * camera.V() * camera.P();

   M.SetUniform(shaderProg, "M");
   Minv.SetUniform(shaderProg, "Minv");
   MVP.SetUniform(shaderProg, "MVP");

   glBindVertexArray(vao);
   glDrawArrays(GL_TRIANGLES, 0, nVtx);
}
```

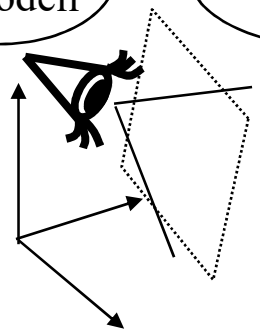
Képszintézis csővezeték

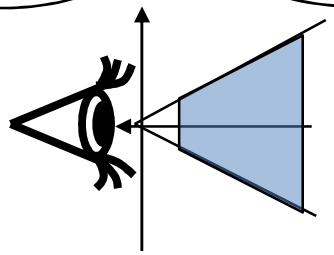


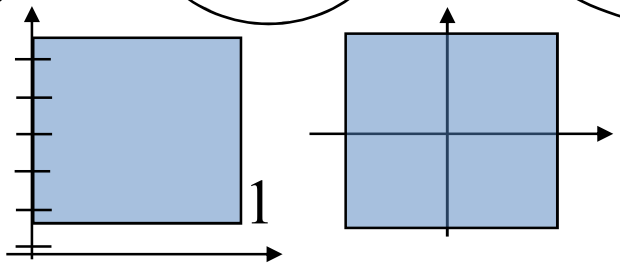
$$[X(t), Y(t), Z(t), h(t)] = [X_1, Y_1, Z_1, h_1] \cdot t + [X_2, Y_2, Z_2, h_2] \cdot (1\text{-}t)$$

csúcspontok **primitívek**

Modell: x,y,z → $\mathbf{T}_{modell}$ → $\mathbf{T}_{view}$ → $\mathbf{T}_{persp}$ → Vágás → Homogén osztás → $\mathbf{T}_{viewport}$ → Takarás → Vetítés → Rasztertár: X, Y







Vágás homogén koordinátákban

Cél: $-1 < X = X_h/h < 1$

$-1 < Y = Y_h/h < 1$

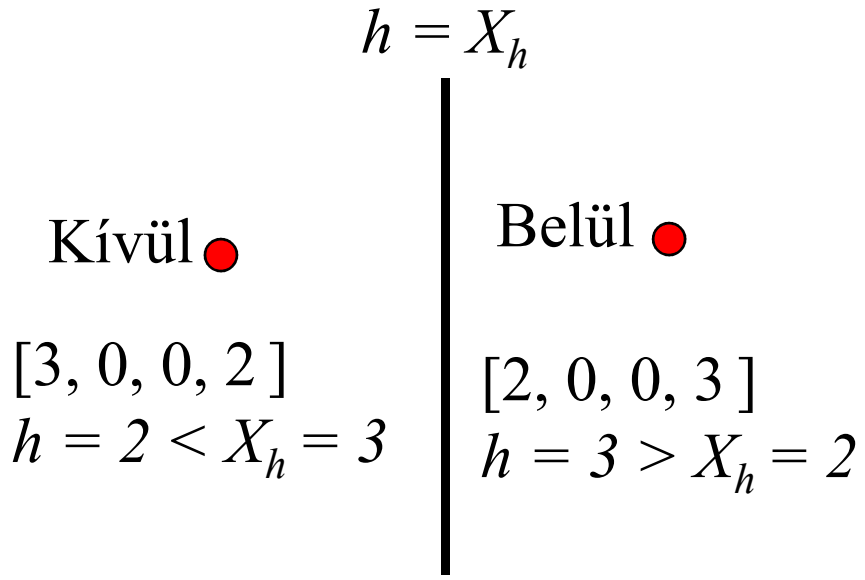
$-1 < Z = Z_h/h < 1$

Vegyük hozzá: $h > 0$ (mert $h = -zc$)

$-h < X_h < h$

$-h < Y_h < h$

$-h < Z_h < h$

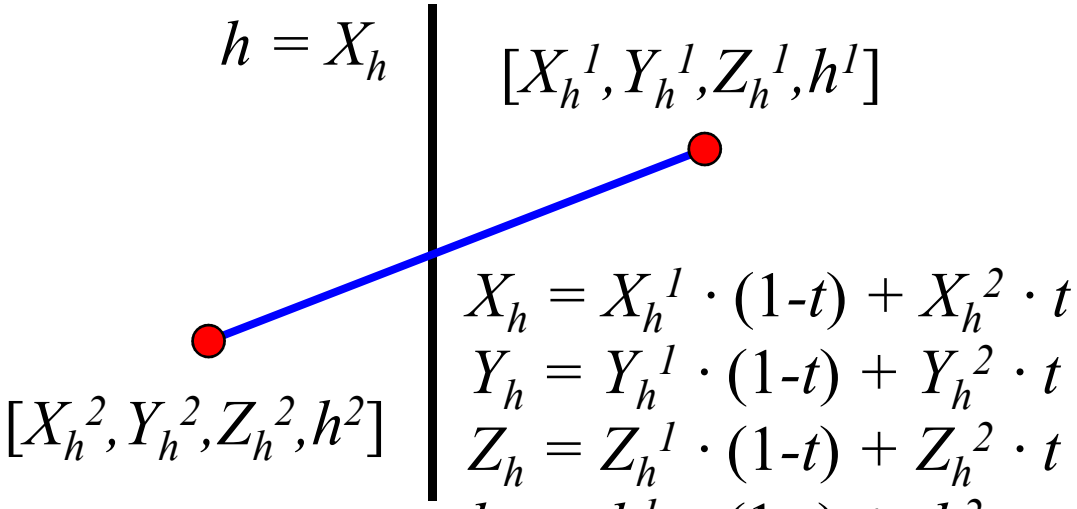


Szakasz/poligon vágás

$$-h < X_h < h$$
$$-h < Y_h < h$$
$$-h < Z_h < h$$

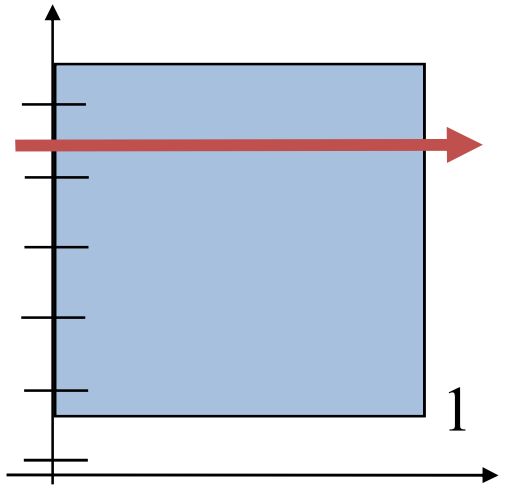
$$h = h^1 \cdot (1\text{-}t) + h^2 \cdot t =$$
$$= X_h = X_h^1 \cdot (1\text{-}t) + X_h^2 \cdot t$$

$$t = \ldots$$



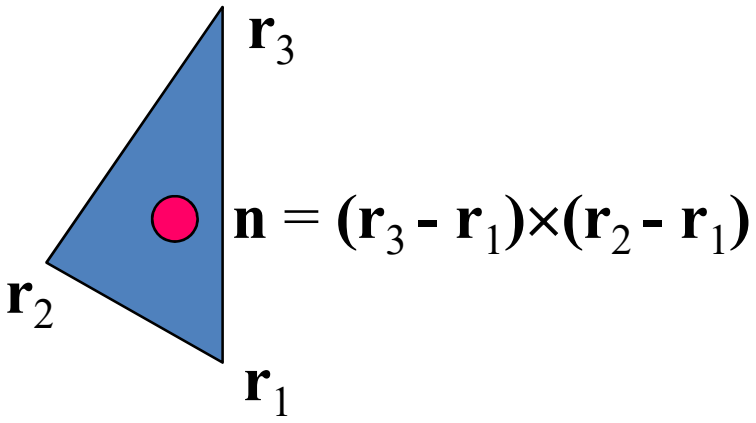
$$X_h = X_h^1 \cdot (1\text{-}t) + X_h^2 \cdot t$$
$$Y_h = Y_h^1 \cdot (1\text{-}t) + Y_h^2 \cdot t$$
$$Z_h = Z_h^1 \cdot (1\text{-}t) + Z_h^2 \cdot t$$
$$h = h^1 \cdot (1\text{-}t) + h^2 \cdot t$$

Takarás

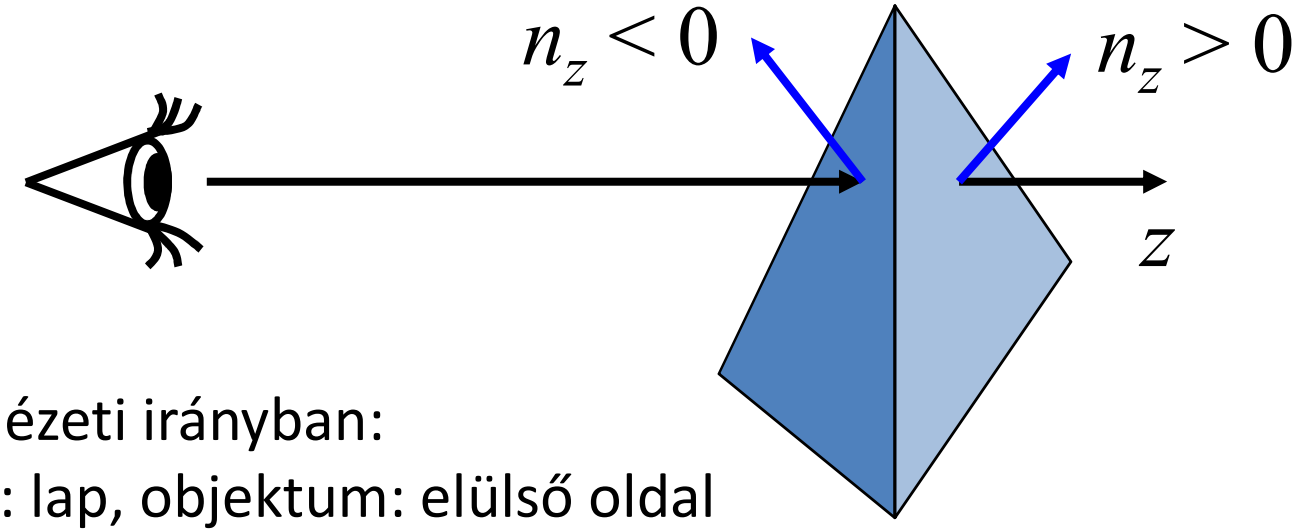


- Képernyő koordinátarendszerben
  - **vetítősugarak a *z* tengellyel párhuzamosak**!

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



Hátsólab eldobás: back-face culling



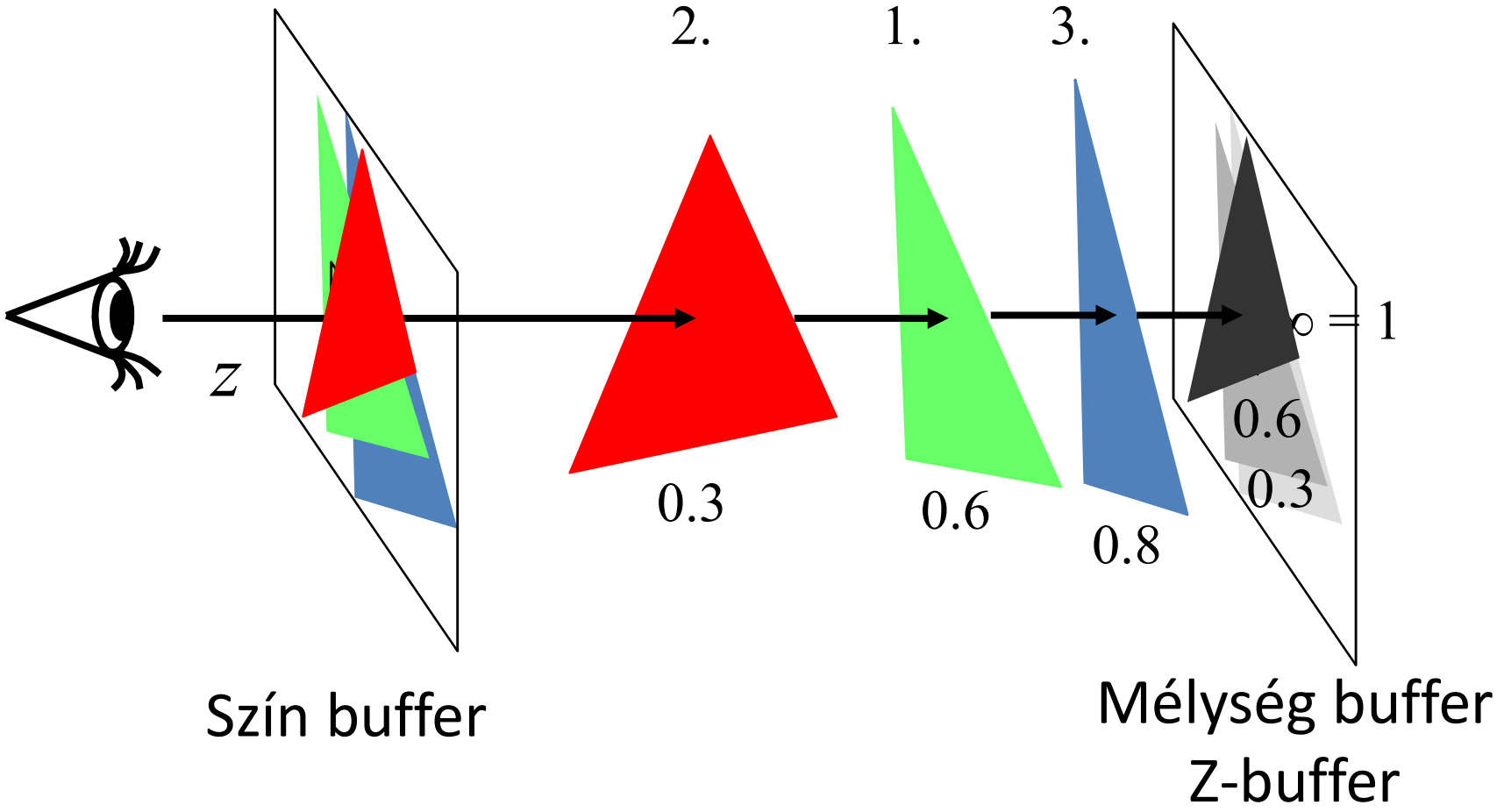
Lapok a nézeti irányban:

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

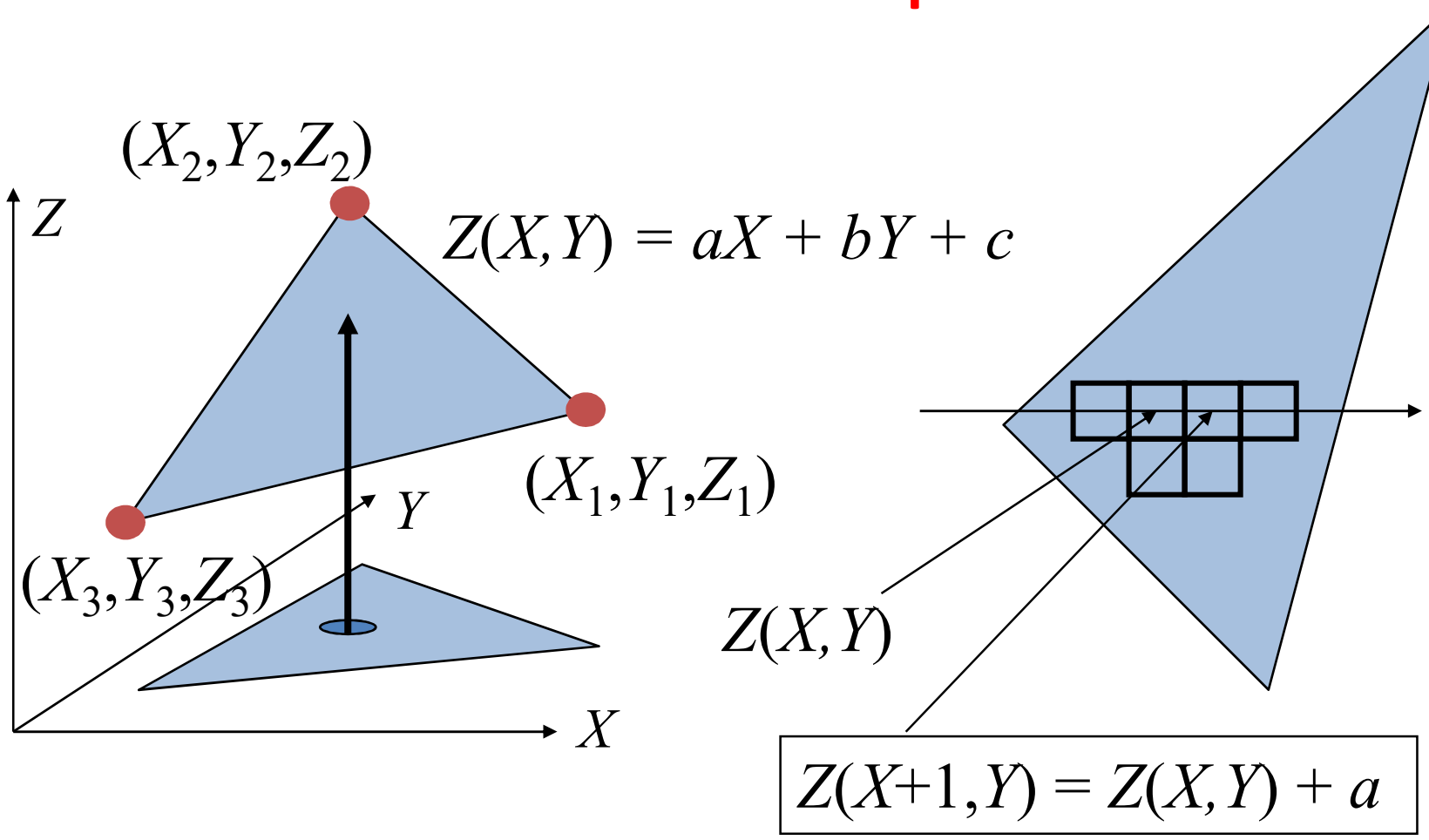
Feltételezés:

Ha kívülről, akkor csúcsok óramutatóval megegyező körüljárásúak

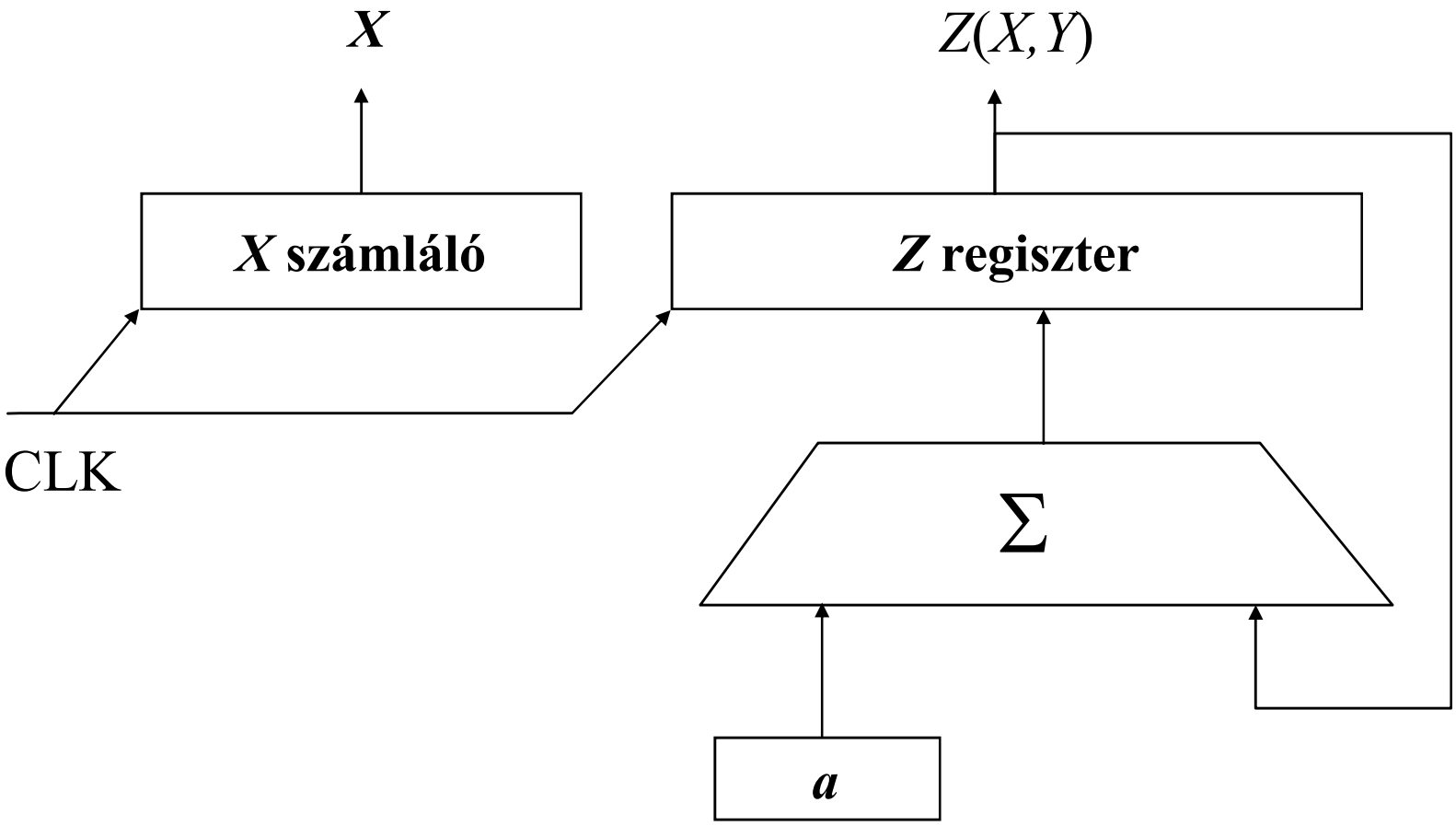
Z-buffer algoritmus



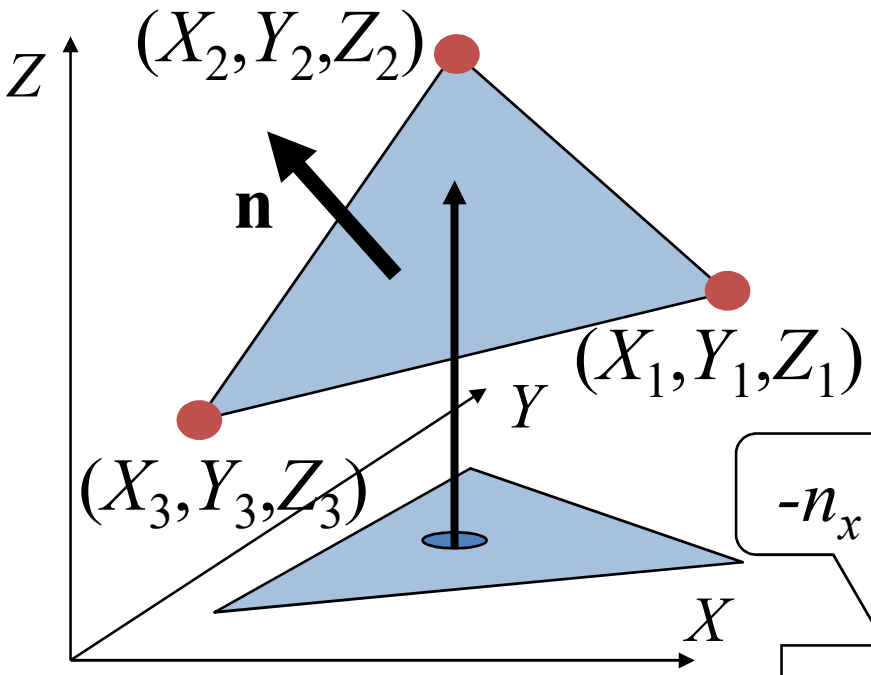
Z: lineáris interpoláció



Z-interpolációs hardver



Triangle setup



$$Z_1 = aX_1 + bY_1 + c$$
$$Z_2 = aX_2 + bY_2 + c$$
$$Z_3 = aX_3 + bY_3 + c$$

$$Z_3 - Z_1 = a(X_3 - X_1) + b(Y_3 - Y_1)$$
$$Z_2 - Z_1 = a(X_2 - X_1) + b(Y_2 - Y_1)$$

$-n_x$

$$Z(X,Y) = aX + bY + c$$
$$n_xX + n_yY + n_zZ + d = 0$$

$$a = \frac{(Z_3 - Z_1)(Y_2 - Y_1) - (Y_3 - Y_1)(Z_2 - Z_1)}{(X_3 - X_1)(Y_2 - Y_1) - (Y_3 - Y_1)(X_2 - X_1)}$$

n_z

$$\mathbf{n} = (\mathbf{r}_3 - \mathbf{r}_1) \times (\mathbf{r}_2 - \mathbf{r}_1) = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ X_3 - X_1 & Y_3 - Y_1 & Z_3 - Z_1 \\ X_2 - X_1 & Y_2 - Y_1 & Z_2 - Z_1 \end{vmatrix}$$

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

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

- Koherencia: ne mindent pixelenként
- Csúcspontonként:

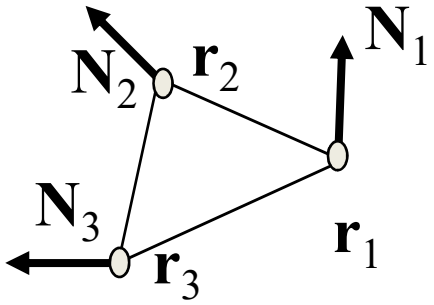
  belül az *L* „szín” interpolációja:

  Gouraud árnyalás (per-vertex shading)
- Pixelenként:

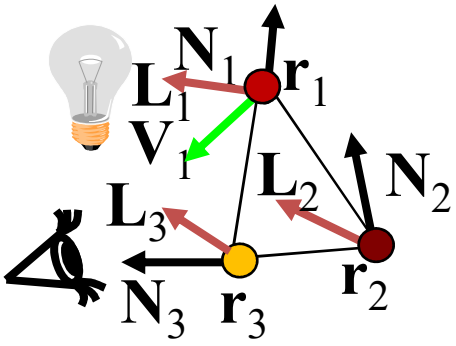
  belül a **N**ormál (**V**iew, **L**ight) vektort interpoláljuk:

  Phong árnyalás (per-pixel shading)

Per-vertex (Gouraud) árnyalás



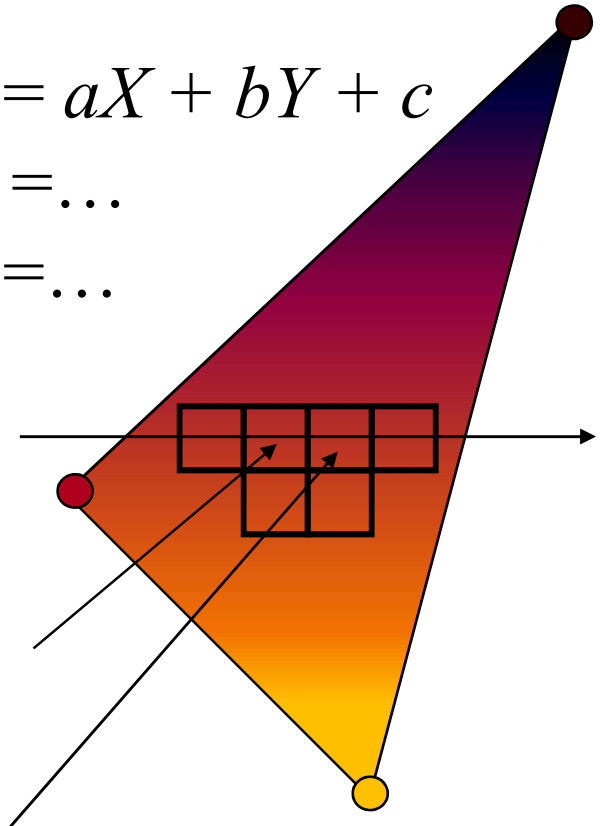
Modell
[+kamera]



(per-vertex shading)

Perspektív
+ nézeti

$$R(X,Y) = aX + bY + c$$
$$G(X,Y) = \ldots$$
$$B(X,Y) = \ldots$$



$R(X,Y)$

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

Képernyő

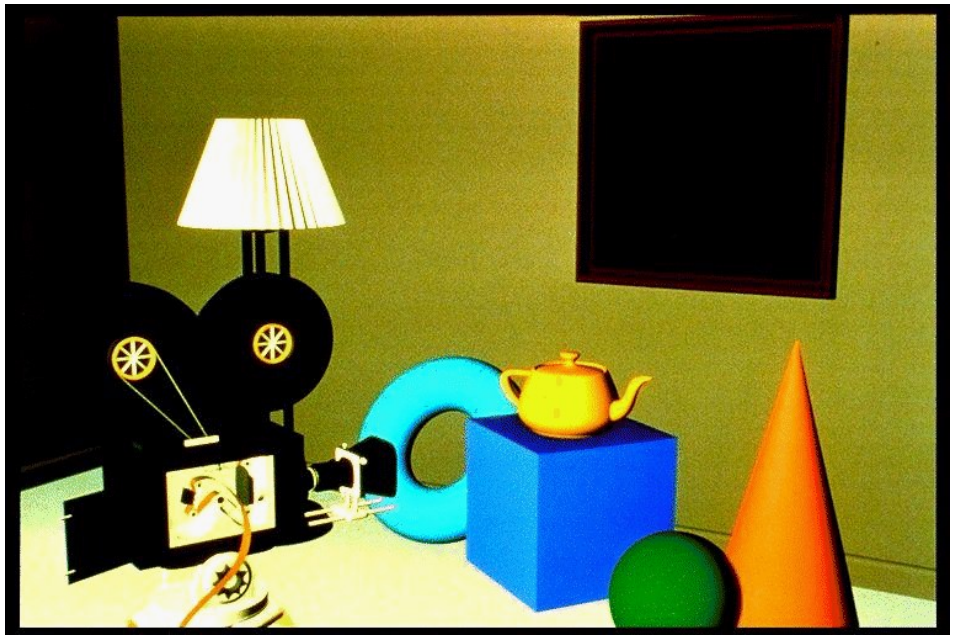
Per-vertex shading: Vertex shader

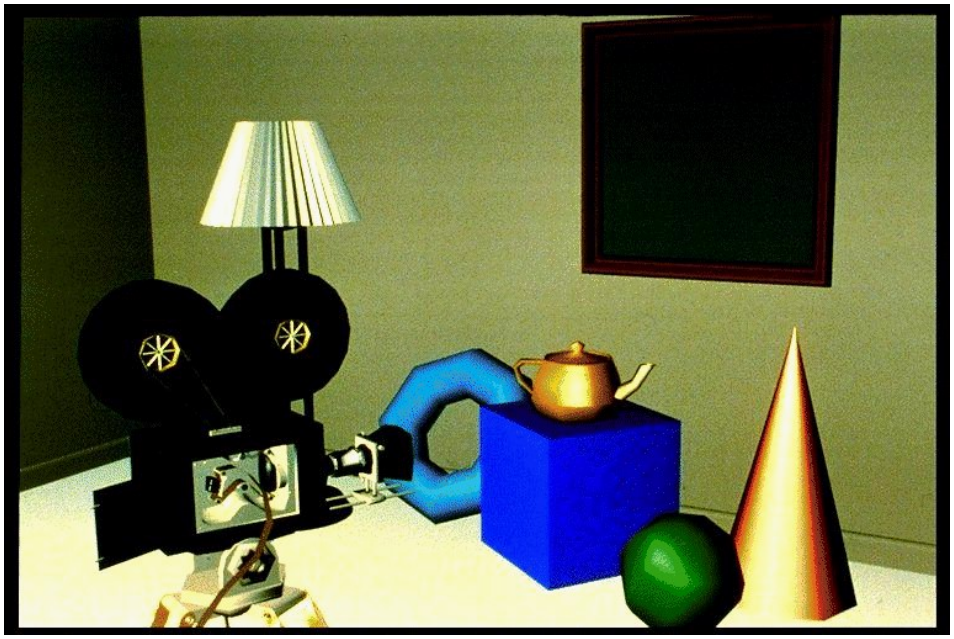
```
uniform mat4  MVP, M, Minv; // MVP, Model, Model-inverse
uniform vec4  kd, ks, ka;   // diffuse, specular, ambient ref
uniform float shine;        // shininess for specular ref
uniform vec4  La, Le;       // ambient and point sources
uniform vec4  wLiPos;       // pos of light source in world
uniform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

in vec3 vtxPos;             // pos in modeling space
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.w - wPos.xyz);
   vec3 N = normalize( (Minv * vec4(vtxNorm, 0)).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;
}
```

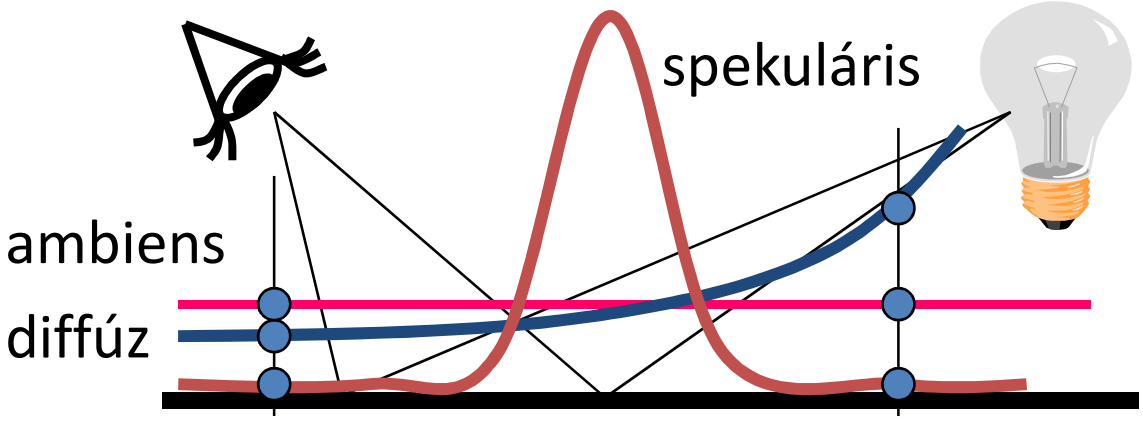
Per-vertex shading: Pixel shader

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

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

Gouraud árnyalás hasfájásai





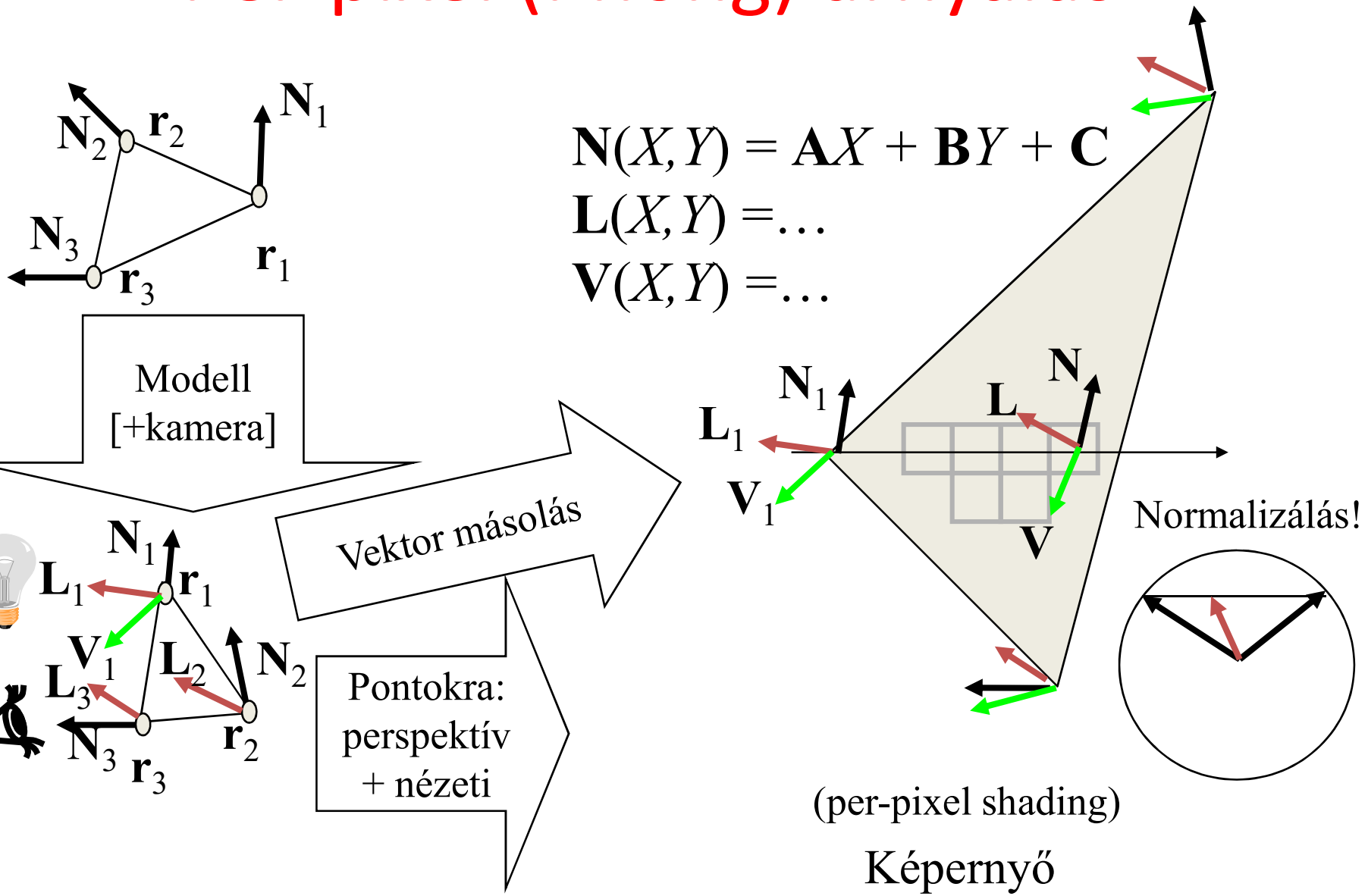


<u>További bajok:</u>

- 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

in  vec3 vtxPos;            // pos in modeling space
in  vec3 vtxNorm;           // normal in modeling space

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

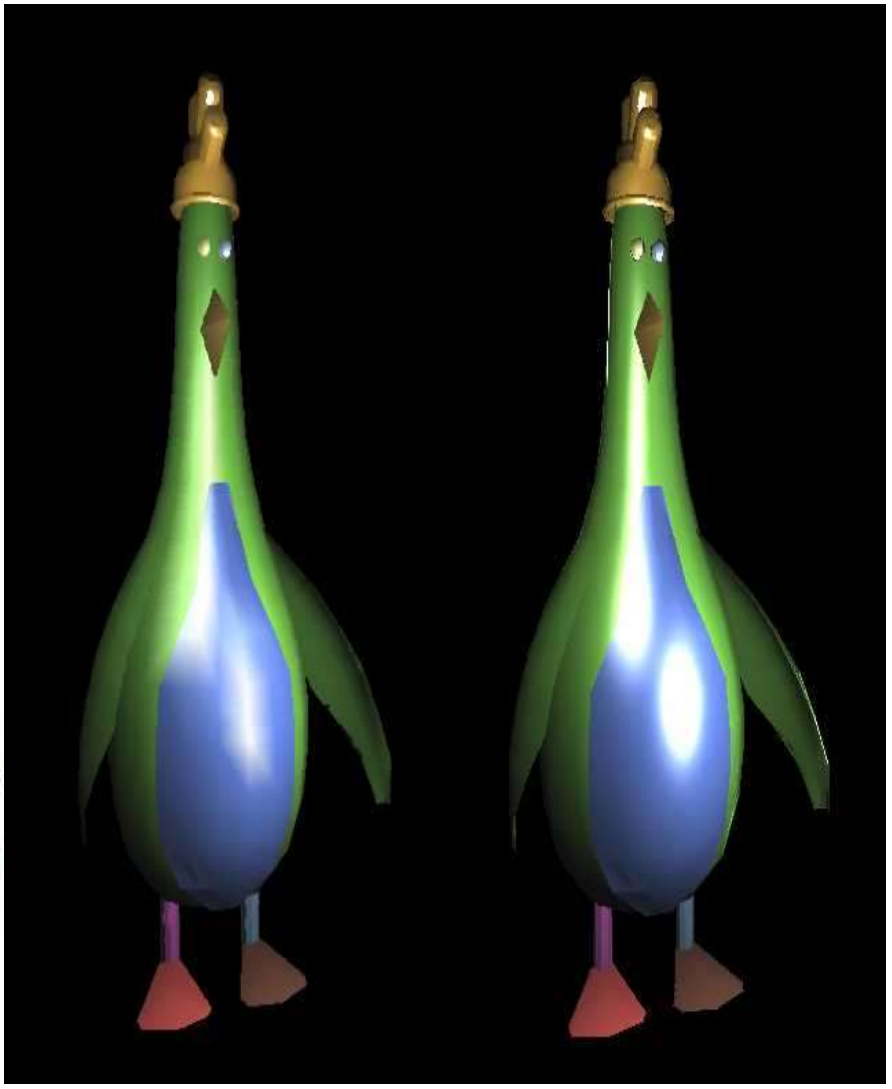
Per-pixel shading: Pixel shader

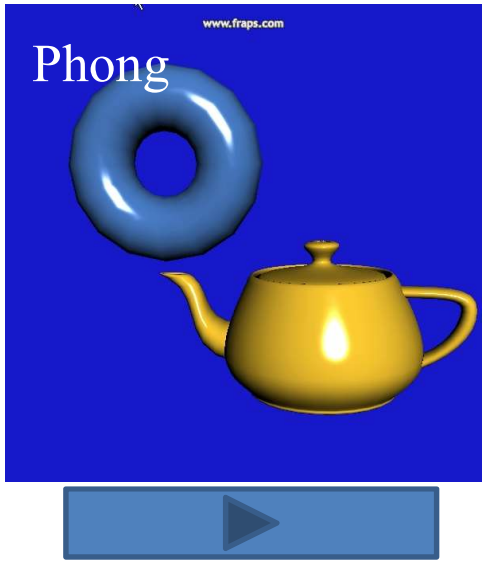
```
uniform vec3 kd, ks, ka;// diffuse, specular, ambient ref
uniform vec3 La, Le;    // ambient and point source rad
uniform float shine;    // shininess for specular ref
in  vec3 wNormal;       // interpolated world sp normal
in  vec3 wView;         // interpolated world sp view
in  vec3 wLight;        // 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);
}
```

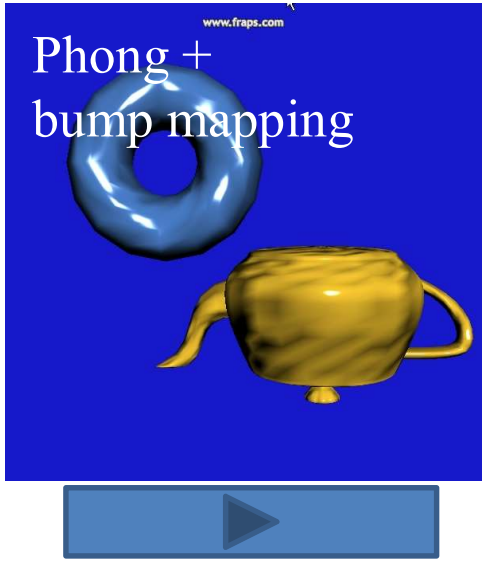
Gouraud versus Phong

Gouraud

Phong

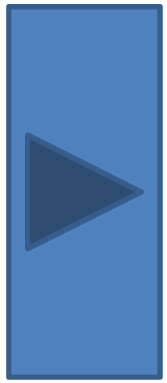


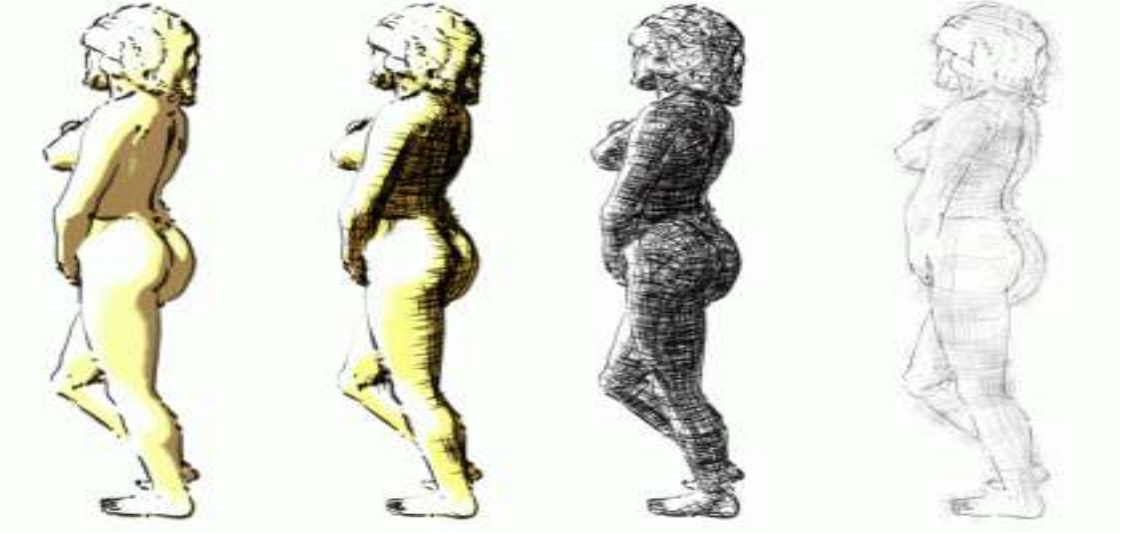




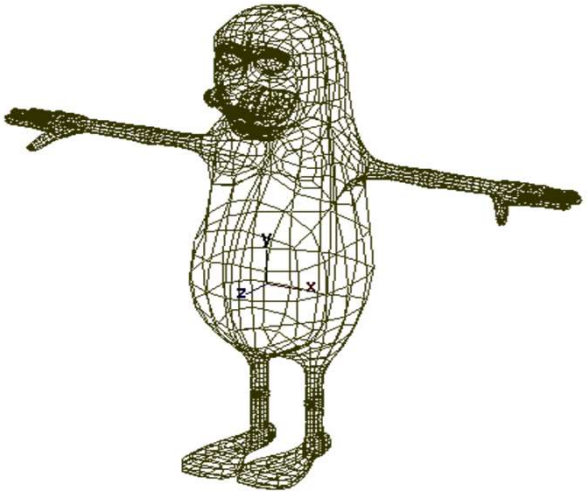
NPR: Non-Photorealistic Rendering

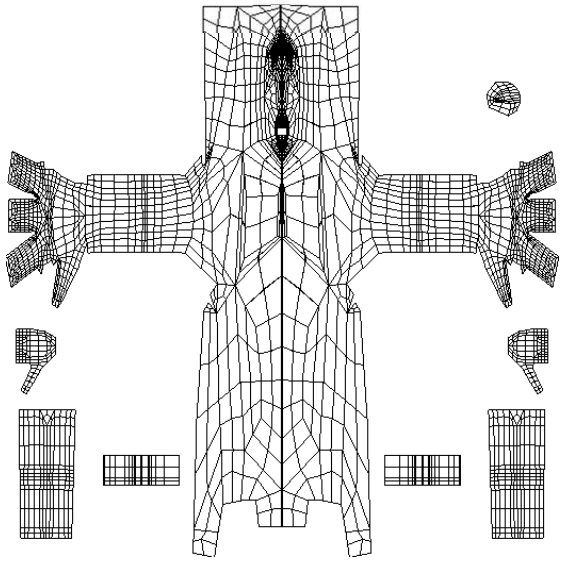
```
uniform vec3 kd;                        // diffuse ref
in  vec3 wNormal, wView, wLight;        // interpolated
out vec4 fragmentColor;                 // output goes to frame buffer

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);
}
```



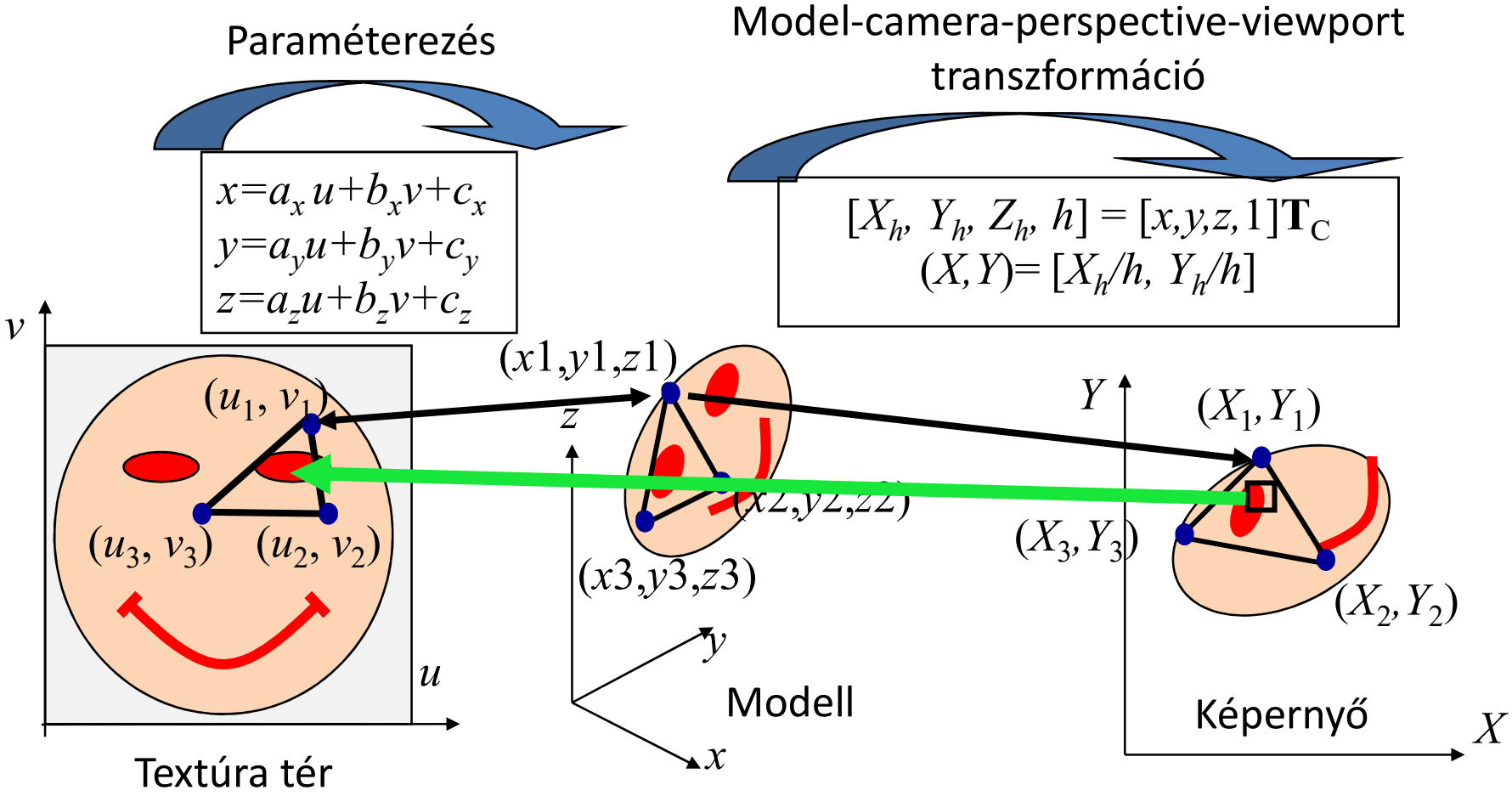


Textúra leképzés: anyagjellemzők változnak a felületen





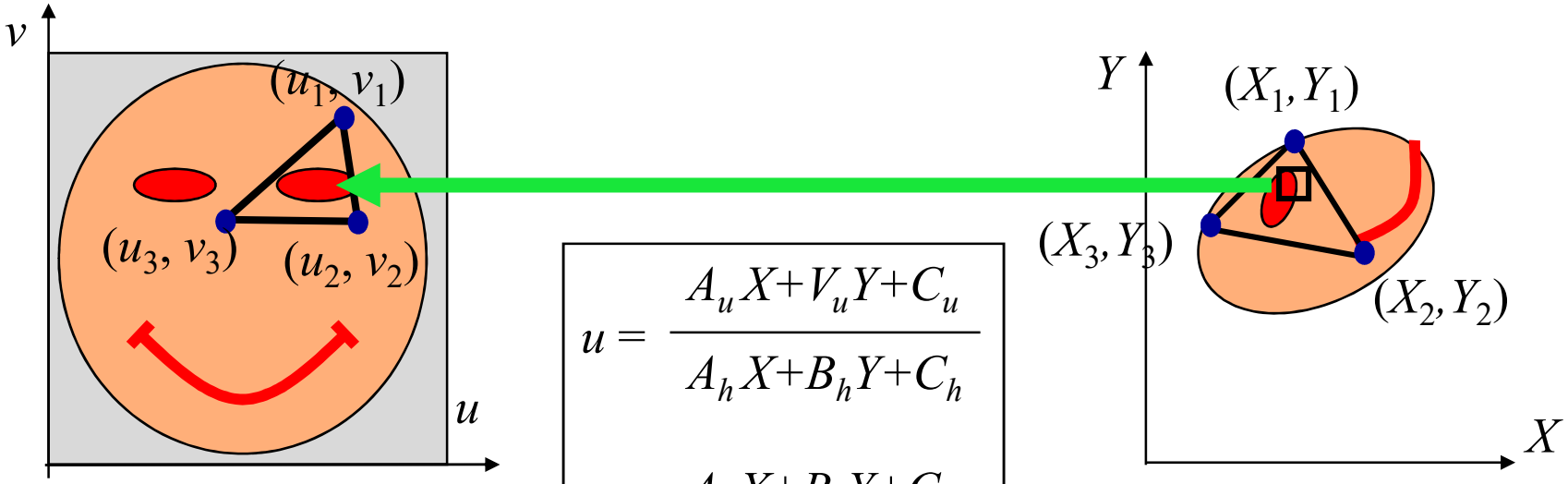
Textúrázás



$[X \cdot h,\ Y \cdot h,\ h] = [u,v,1] \cdot \mathbf{T}$, ahol $h = -zc$

$[u/h,\ v/h,\ 1/h] = [X,\ Y,\ 1] \cdot \mathbf{T}^{-1}$

Perspektíva helyes textúrázás



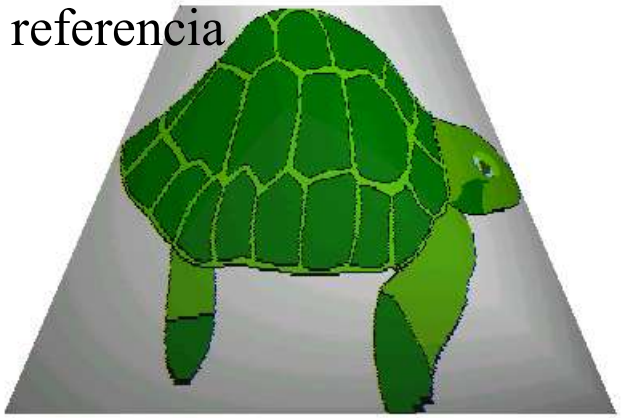
$$u = \frac{A_u X + V_u Y + C_u}{A_h X + B_h Y + C_h}$$

$$v = \frac{A_v X + B_v Y + C_v}{A_h X + B_h Y + C_h}$$

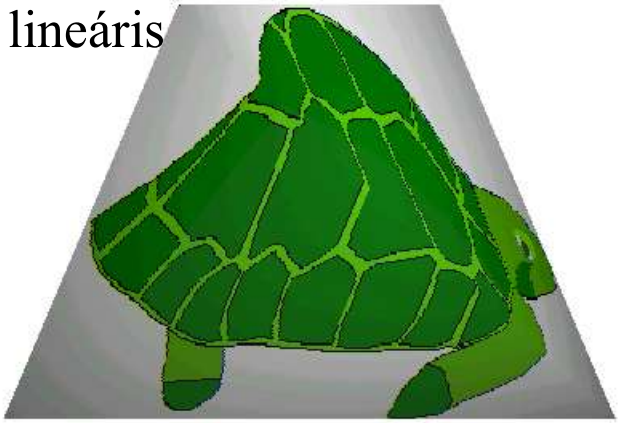
$$u/h = A_u X + B_u Y + C_u$$
$$v/h = A_v X + B_v Y + C_v$$
$$1/h = A_h X + B_h Y + C_h$$

$$h = -zc$$

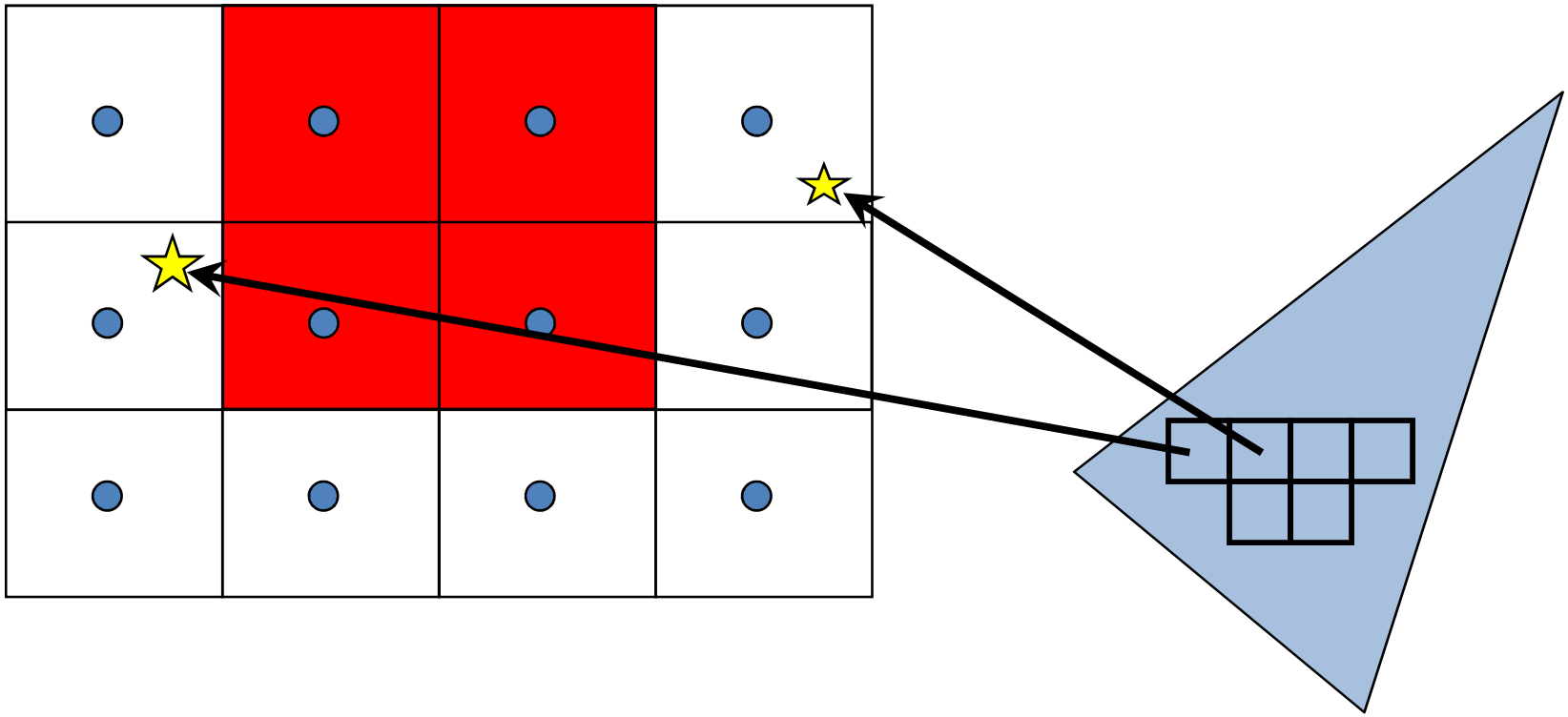
referencia



lineáris



Textúra szűrés

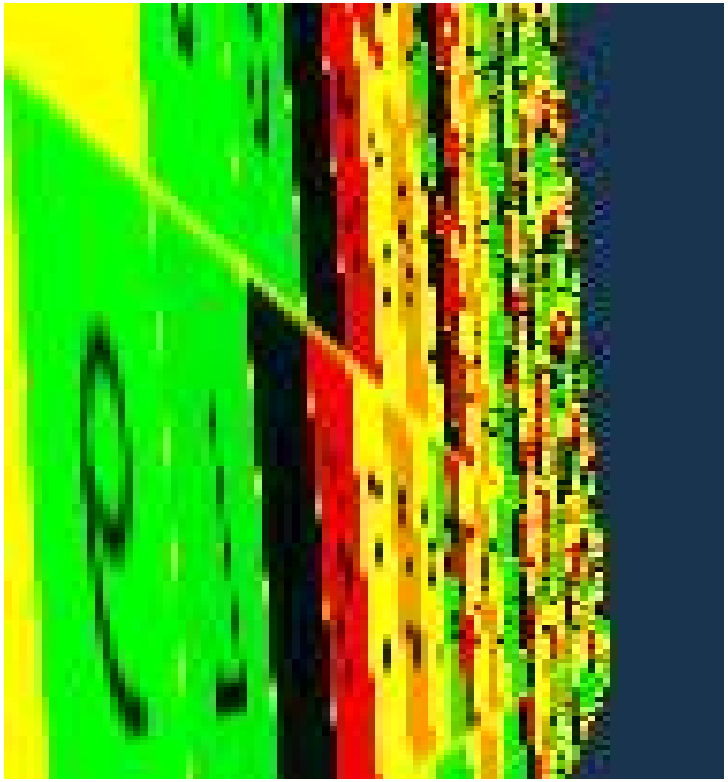


Textúra tér

Képtér

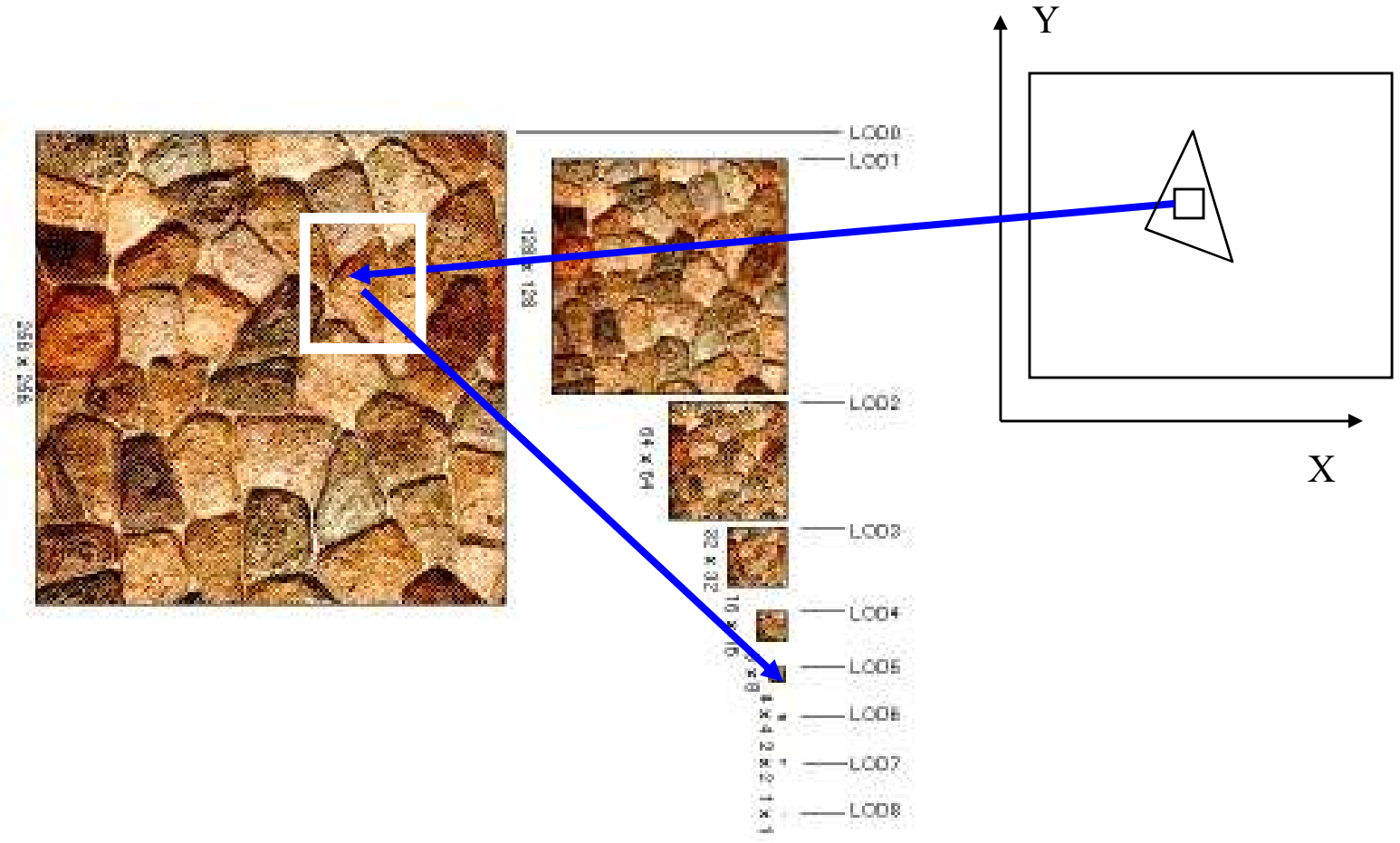
Textúratér és képtér kapcsolata

Magnification



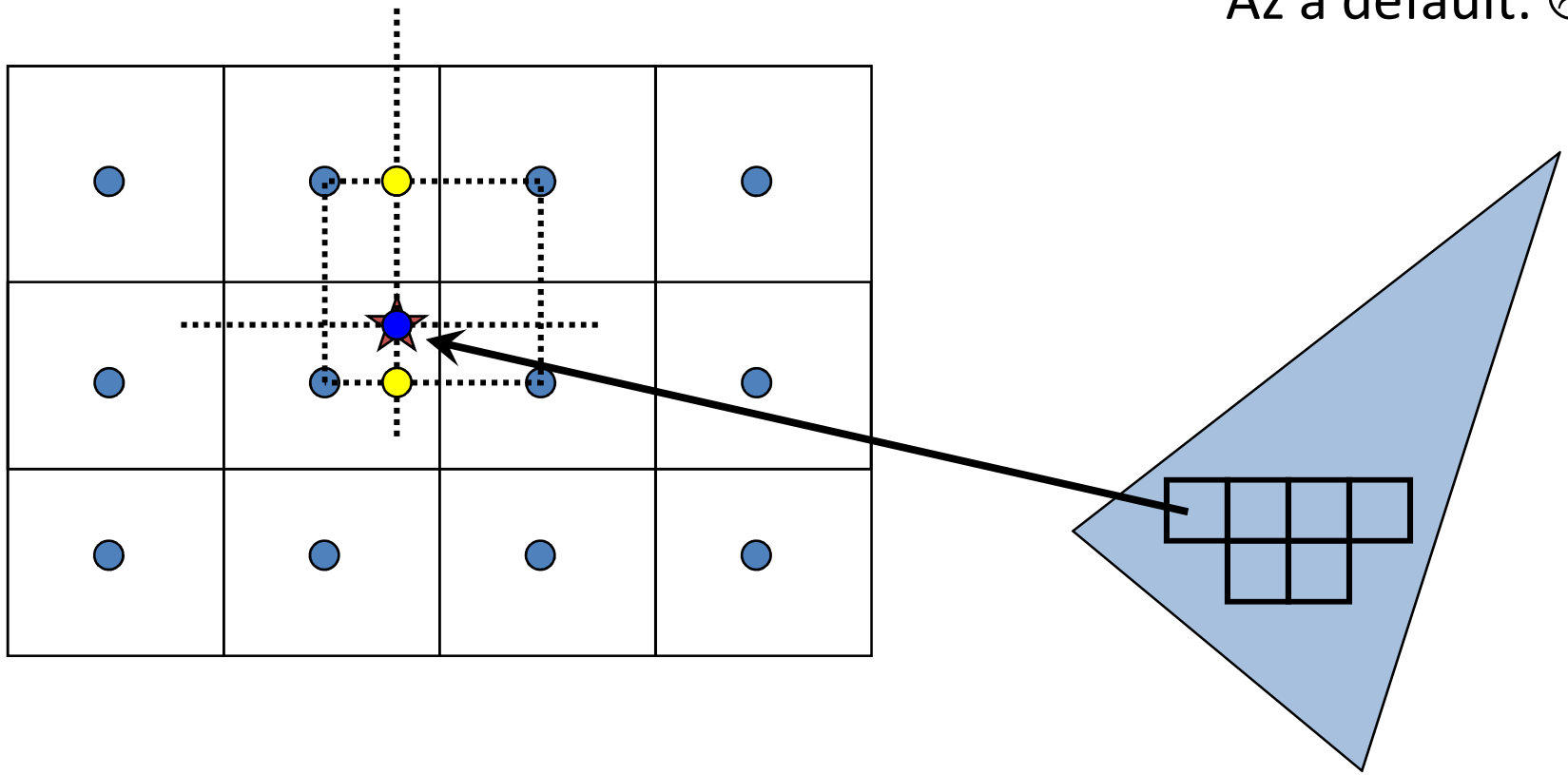
Minification

Mip-map (multum in parvo)



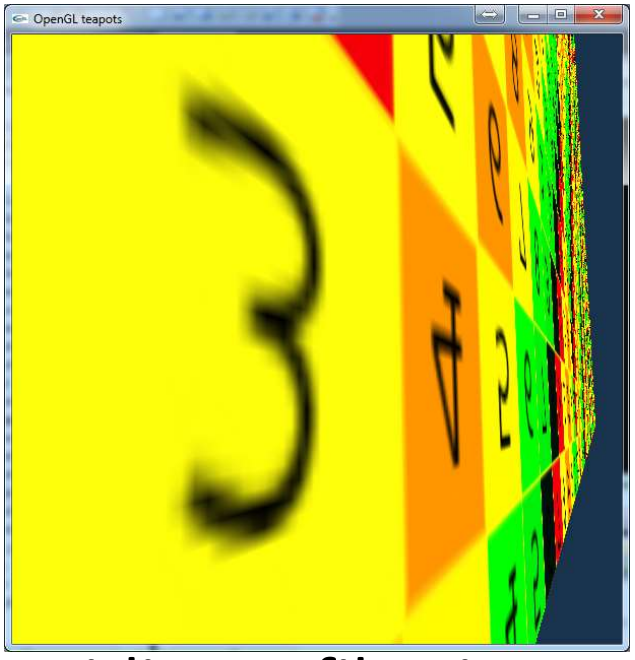
Bi-linear textúra szűrés

Mip-map is van: ☺
Az a default: ☹

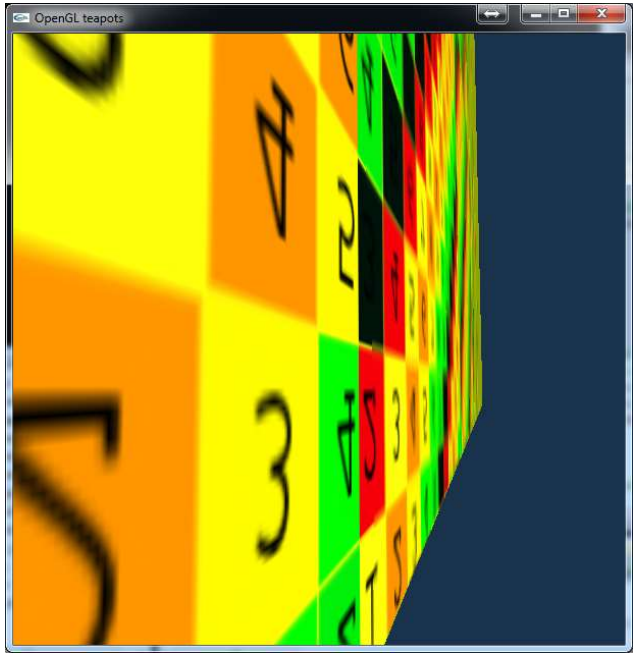


```
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D,
                GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

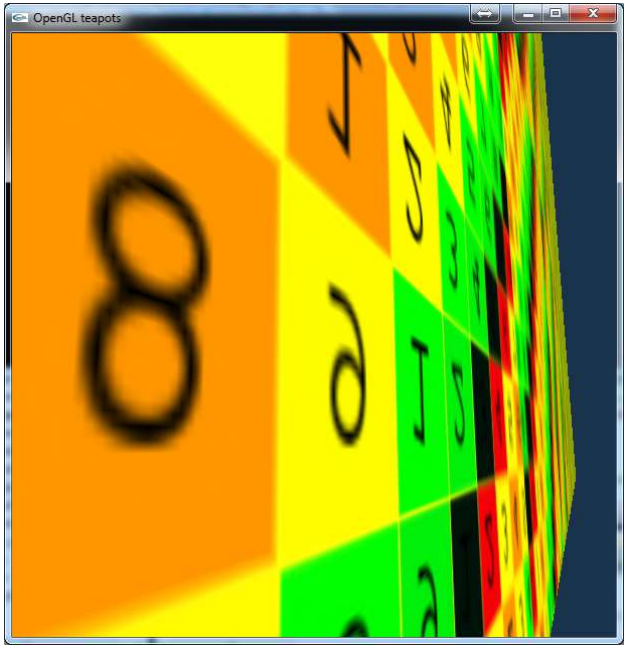
Textúra szűrési módok



Bi-linear filtering

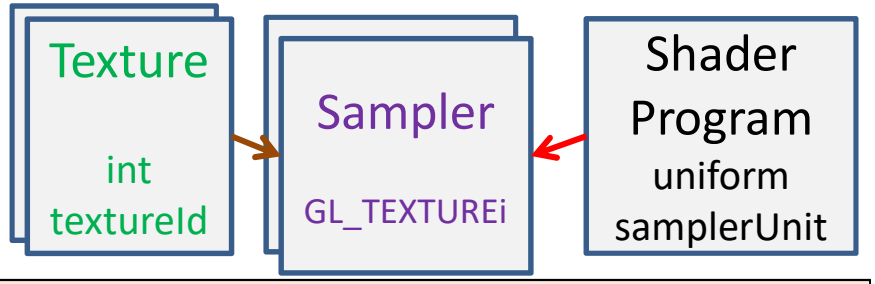


Mip-mapping



Linear mip-map szint

Textúrázás




```
struct Texture {
    unsigned int textureId;
    Texture(char * fname) {
        glGenTextures(1, &textureId);
        glBindTexture(GL_TEXTURE_2D, textureId);     // binding
        int width, height, level = 0, border = 0;
        float *image = LoadImage(fname,width,height); // megírni!
        glTexImage2D(GL_TEXTURE_2D, level, GL_RGB, width, height, border,
                     GL_RGB, GL_FLOAT, image); //Texture -> OpenGL
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    }
};
```

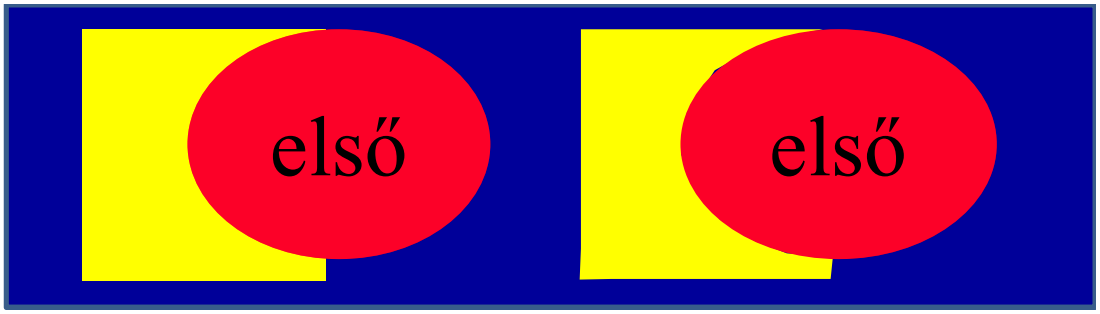
```
void Geometry::Draw( ) {
  int sampler = GL_TEXTURE0; // GL_TEXTURE1, …
  int loc = glGetUniformLocation(shaderProg, "samplerUnit");
  glUniform1i(loc, sampler); // Shader samplerUnit to sampler GL_TEXTURE0
  glActiveTexture(sampler);
  glBindTexture(GL_TEXTURE_2D,texture.textureId); // sampler to textureId
  glBindVertexArray(vao); glDrawArrays(GL_TRIANGLES, 0, nVtx);
}
```

Vertex és Pixel Shader

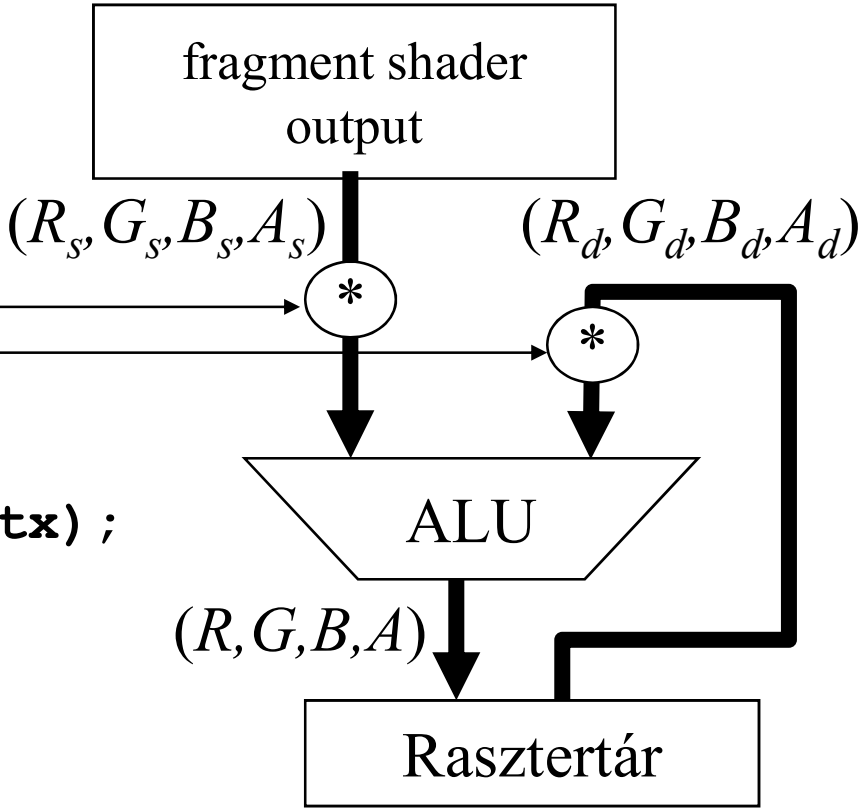
```
in vec3 vtxPos;
in vec3 vtxNorm;
in vec2 vtxUV;
out vec2 texcoord;

void main() {
    gl_Position = vec4(vtxPos, 1) * MVP;
    texcoord = vtxUV;
    …
}
```

```
uniform sampler2D samplerUnit;
in vec2 texcoord;
out vec4 fragmentColor;

void main() {
   fragmentColor = texture(samplerUnit, texcoord);
}
```

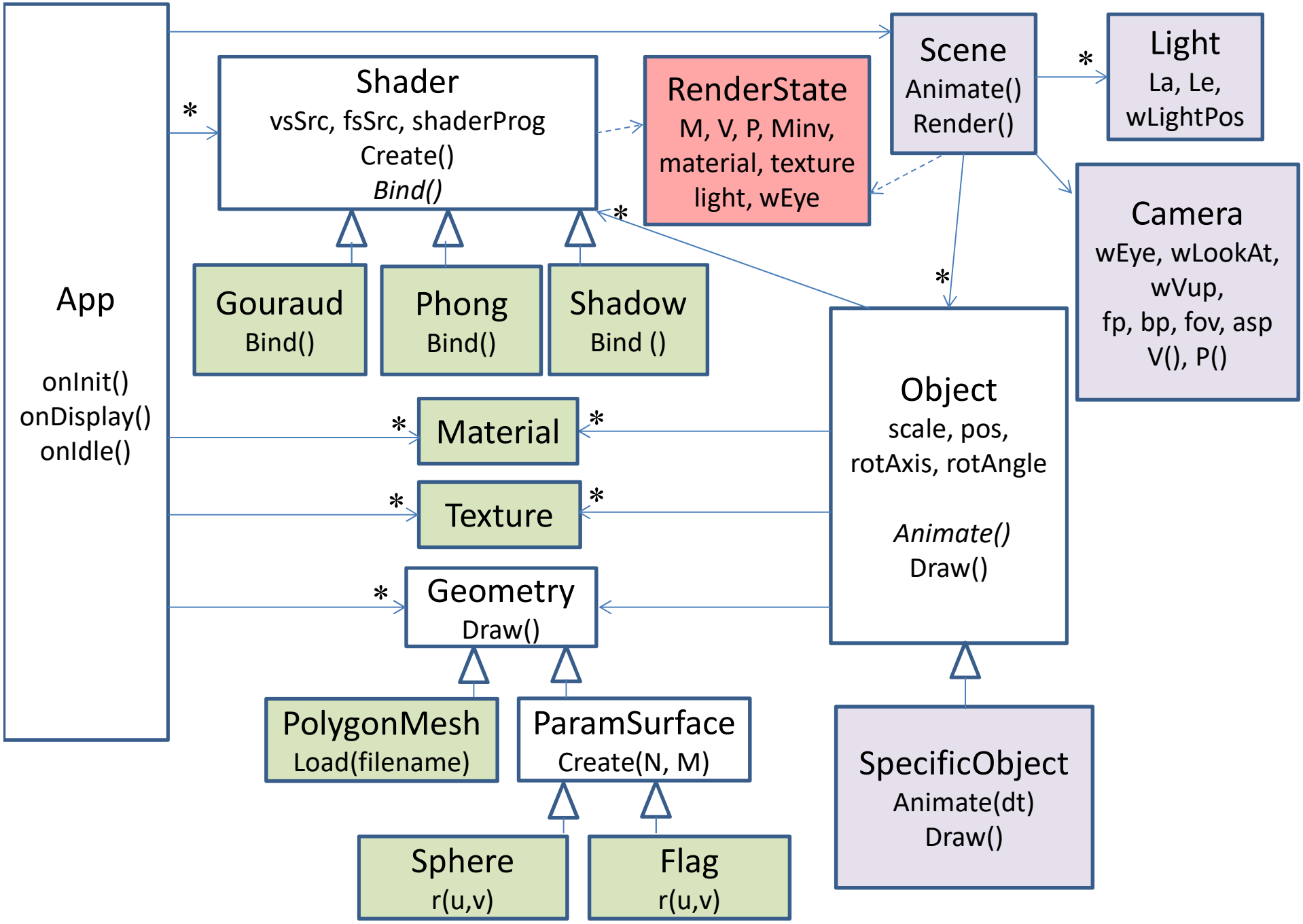
Átlátszóság: Sorrend számít!




```
glEnable(GL_BLEND);

glBlendFunc(
    GL_SRC_ALPHA,
    GL_ONE_MINUS_SCR_ALPHA
);

glDrawArrays(GL_TRIANGLES,0,nVtx);

glDisable(GL_BLEND);
```



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


App
onInit()
onDisplay()
onIdle()
Shader
vsSrc, fsSrc, shaderProg
Create()
Bind()
RenderState
M, V, P, Minv,
material, texture
light, wEye
Scene
Animate()
Render()
Light
La, Le,
wLightPos
Camera
wEye, wLookAt,
wVup,
fp, bp, fov, asp
V(), P()
Gouraud
Bind()
Phong
Bind()
Shadow
Bind ()
Material
Texture
Geometry
Draw()
Object
scale, pos,
rotAxis, rotAngle
Animate()
Draw()
PolygonMesh
Load(filename)
ParamSurface
Create(N, M)
Sphere
r(u,v)
Flag
r(u,v)
SpecificObject
Animate(dt)
Draw()
*

Scene

```
class Scene {
   Camera camera;
   vector<Object *> objects;
   Light light;
   RenderState state;
public:
   void Render() {
      state.wEye = camera.wEye;
      state.V = camera.V;
      state.P = camera.P;
      state.light = light;
      for (Object * obj : objects) obj->Draw(state);
   }
   void Animate(float dt) {
      for (Object * obj : objects) obj->Animate(dt);
   }
};
```

Object

```
class Object {
    Shader *   shader;
    Material * material;
    Texture *  texture;
    Geometry * geometry;
    vec3 scale, pos, rotAxis;
    float rotAngle;
public:
    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.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);
        geometry->Draw();
    }
    virtual void Animate(float dt) {}
};
```

Érték szerinti paraméterátadás:
Objektumok nem zavarják egymást

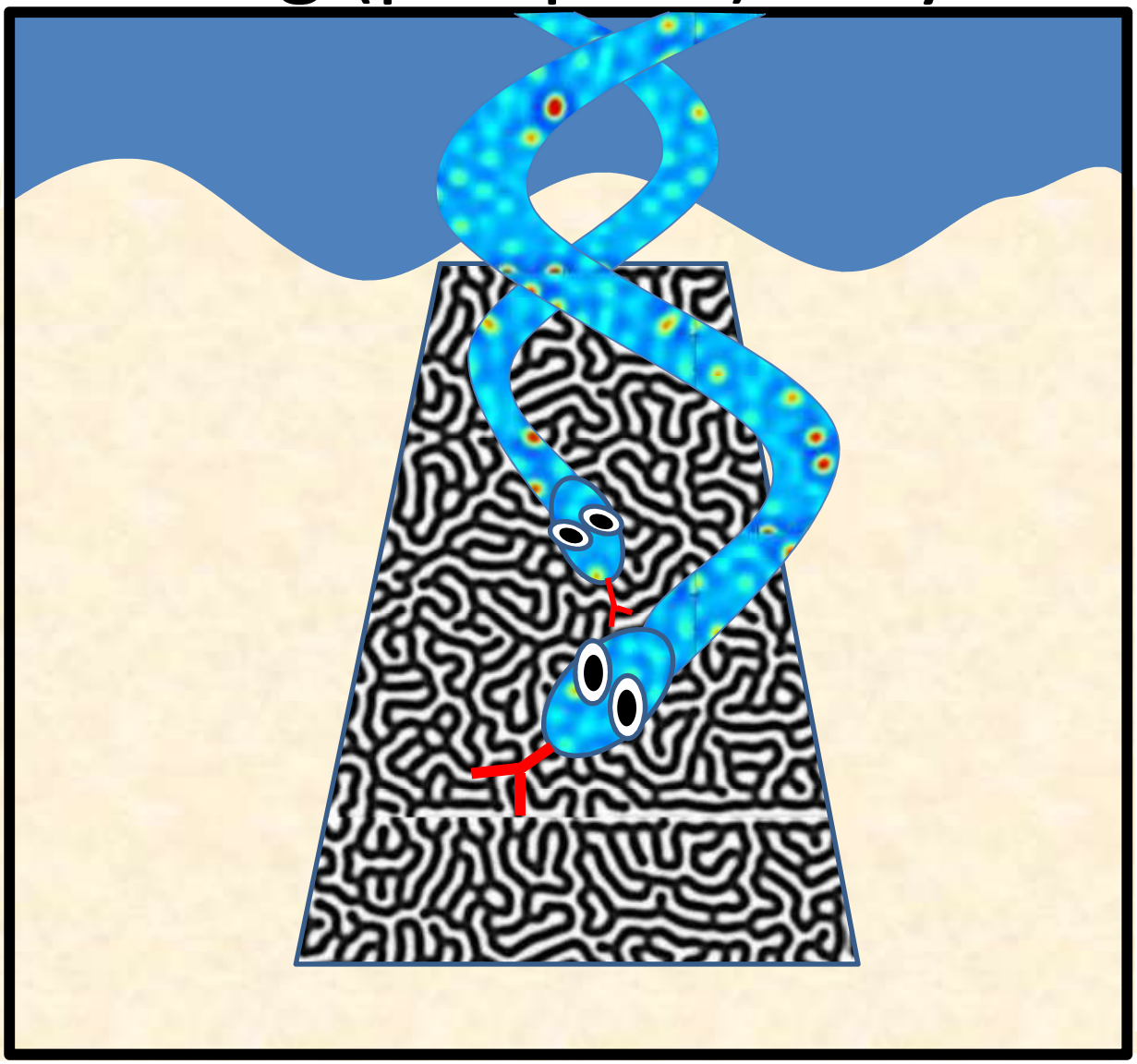
Shader

```
struct Shader {
   unsigned int shaderProg;

   void Create(const char * vsSrc, const char * vsAttrNames[],
               const char * fsSrc, const char * fsOuputName) {
      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);
      shaderProgram = glCreateProgram();
      glAttachShader(shaderProg, vs);
      glAttachShader(shaderProg, fs);

      for (int i = 0; i < sizeof(vsAttrNames)/sizeof(char*); i++)
          glBindAttribLocation(shaderProg, i, vsAttrNames[i]);
      glBindFragDataLocation(shaderProg, 0, fsOuputName);
      glLinkProgram(shaderProg);
   }
   virtual
      void Bind(RenderState& state) { glUseProgram(shaderProg); }
};
```

ShadowShader

```
class ShadowShader : public Shader {
    const char * vsSrc = R"(
        uniform mat4 MVP;
        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() {
        static const char * vsAttrNames[] = { "vtxPos" };
        Create(vsSrc, vsAttrNames, fsSrc, "fragmentColor");
    }
    void Bind(RenderState& state) {
        glUseProgram(shaderProg);
        mat4 MVP = state.M * state.V * state.P;
        MVP.SetUniform(shaderProg, "MVP");
    }
};
```

3. házi: Vergődő kígyók
Phong (per-pixel) árnyalás



Kígyó geometria:
Test: Extruded surface,
Bézier gerinc,
ellipszis profil
Fej/szem: ellipszoid

Anyag:
Textúrázott Diffúz +
Fehér Phong-Blinn

Animáció:
Kontrolpontok
zárt paraméteres görbén,
más periódussal

Homok:
procedurális magasságmező

Procedurális textúrák (CPU):
Reaction-diffusion process
Pl.
http://www.karlsims.com/rd.html