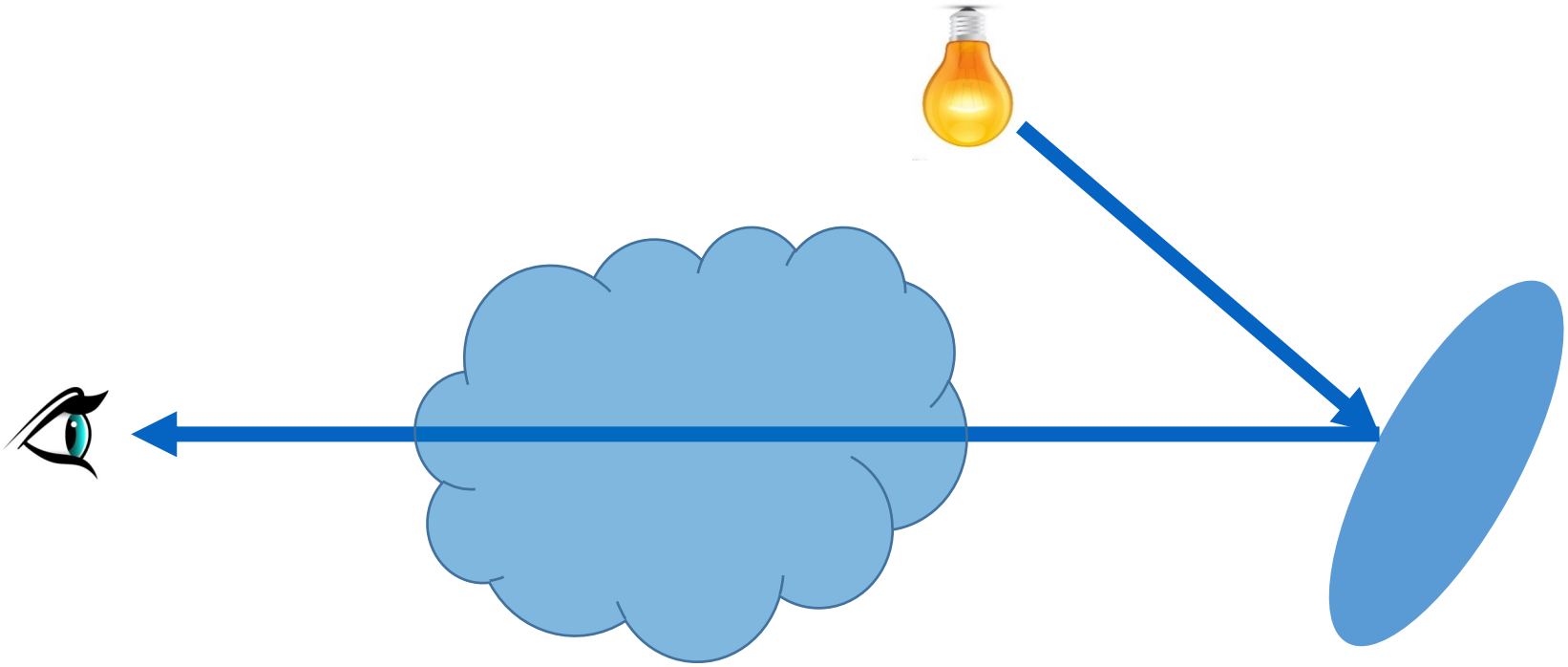
Térfogati illumináció

Szécsi László

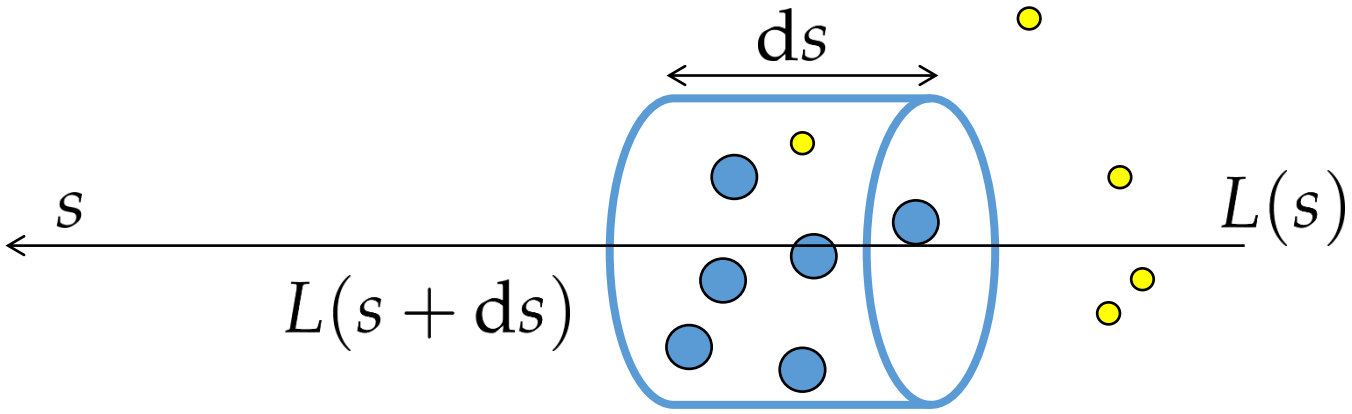
3D Grafikus Rendszerek

11. előadás

Fény terjedése közegben



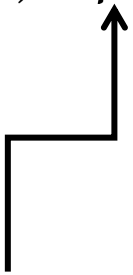
Radianca változása differenciális szakaszon



$$L(s + \mathrm{d}s) = L(s) + \quad ?$$

Elnyelődés - absorption

$$L(s + \mathrm{d}s) = L(s) - L(s) \cdot \mu_{\mathrm{a}}(s) \cdot \mathrm{d}s$$



elnyelődési tényező
absorption coefficient
[m^{-1}]

Kiszóródás - outscattering

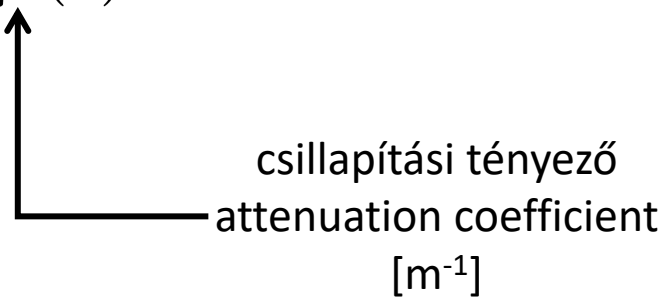
$$L(s + \mathrm{d}s) = L(s) - L(s) \cdot \mu_{\mathrm{a}}(s) \cdot \mathrm{d}s - L(s) \cdot \mu_{\mathrm{s}}(s) \cdot \mathrm{d}s$$

elnyelődési tényező
absorption coefficient
[m^{-1}]

szóródási tényező
scattering coefficient
[m^{-1}]

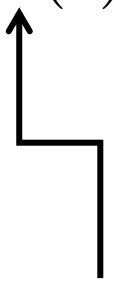
Elnyelődés + kiszóródás = csillaptás

$$L(s+\mathrm{d}s) = L(s) - L(s)\cdot\mu_{\mathrm{a}}(s)\cdot\mathrm{d}s - L(s)\cdot\mu_{\mathrm{s}}(s)\cdot\mathrm{d}s$$
$$L(s+\mathrm{d}s) = L(s) - L(s)\cdot\mathrm{d}s\cdot(\mu_{\mathrm{a}}(s)+\mu_{\mathrm{s}}(s))$$
$$L(s+\mathrm{d}s) = L(s) - L(s)\cdot\mathrm{d}s\cdot\mu(s)$$



Emisszió

$$L(s + \mathrm{d}s) = L(s) + \epsilon(s) \cdot \mathrm{d}s$$



emissziós tényező
emission coefficient
[W·(sr)$^{-1}$·m^{-3}]

Beszóródás - inscattering

$$L(s+\mathrm{d}s) = L(s) + \epsilon(s)\cdot \mathrm{d}s$$

$$+\mu_{\mathrm{s}}(s)\cdot \mathrm{d}s \cdot \int_{\Omega} L^{\mathrm{in}}(s,\omega')p(\omega',s,\omega)\mathrm{d}\omega'$$

fázisfüggvény
phase function
[$(sr)^{-1}$]

A szóródás valószínűségsűrűség-függvénye - fázisfüggvény

- az x felületi pontban az $\hat{l}$ irányból belépő egységnyi nem átengedett teljesítménysűrűség hatására a $\hat{v}$ irányba kilépő radianciára

$$p(\hat{l}, x, \hat{v}) = \frac{L(x, \hat{v})}{M(x, \hat{l})} \quad [\ (\mathrm{sr})^{-1}\ ]$$

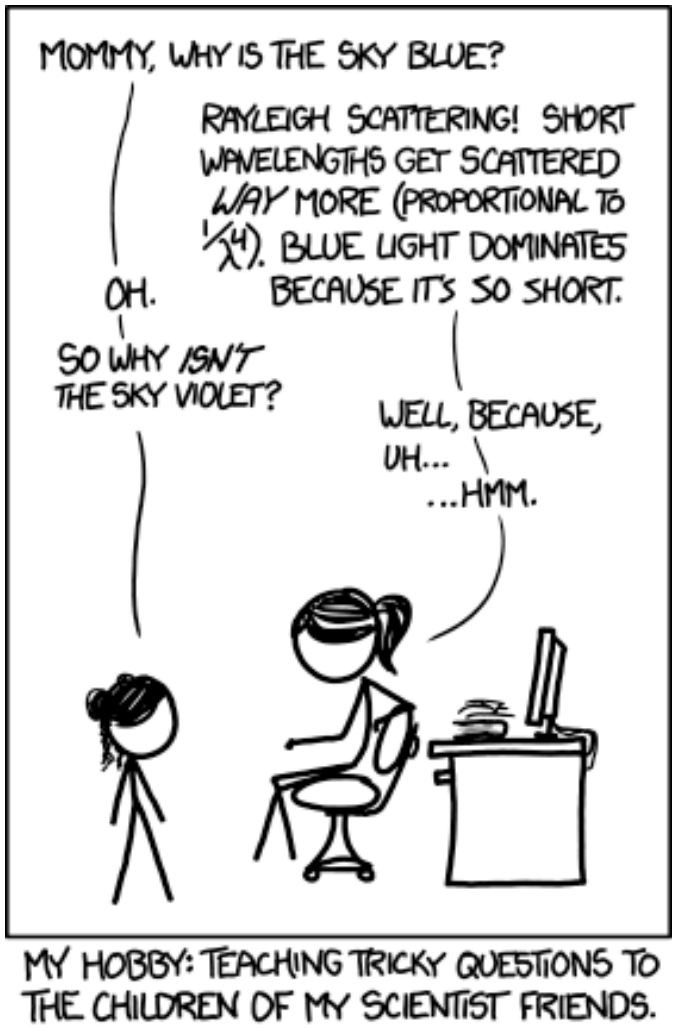
- ez a közeg optikai jellemzője

Izotropikus szóródás

$$p(\hat{\boldsymbol{l}}, \boldsymbol{x}, \hat{\boldsymbol{v}}) = \frac{1}{4\pi}$$

Rayleigh szóródás

$$p(\hat{\boldsymbol{l}}, \boldsymbol{x}, \hat{\boldsymbol{v}}) = (1 + \cos^2 \theta') \frac{2}{16\pi}$$



Henyey-Greenstein

$$p(\hat{\boldsymbol{l}}, \boldsymbol{x}, \hat{\boldsymbol{v}}) = \frac{1}{4\pi} \frac{1 - g^2}{1 + g^2 - 2g \cos^{2/3} \theta'}$$

anizotrópia-faktor

Beszóródás + emisszió = forráshatás

$$L(s+\mathrm{d}s) = L(s) + \epsilon(s)\cdot\mathrm{d}s$$

$$+\mu_{\mathrm{s}}(s)\cdot\mathrm{d}s\cdot\int_{\Omega} L^{\mathrm{in}}(s,\omega')p(\omega',s,\omega)\mathrm{d}\omega' =$$

$$L(s) + q(s)\cdot\mathrm{d}s$$

forrástag
source term
[W·(sr)$^{-1}$·m^{-3}]

Térfogati árnyalási egyenlet

$$\frac{\mathrm{d}L}{\mathrm{d}s} = (-\mu_\mathrm{a} - \mu_\mathrm{s}) \cdot L + \epsilon + \mu_\mathrm{s} \int_\Omega L^\mathrm{in}(\omega') p(\omega', s, \omega) \mathrm{d}\omega'$$

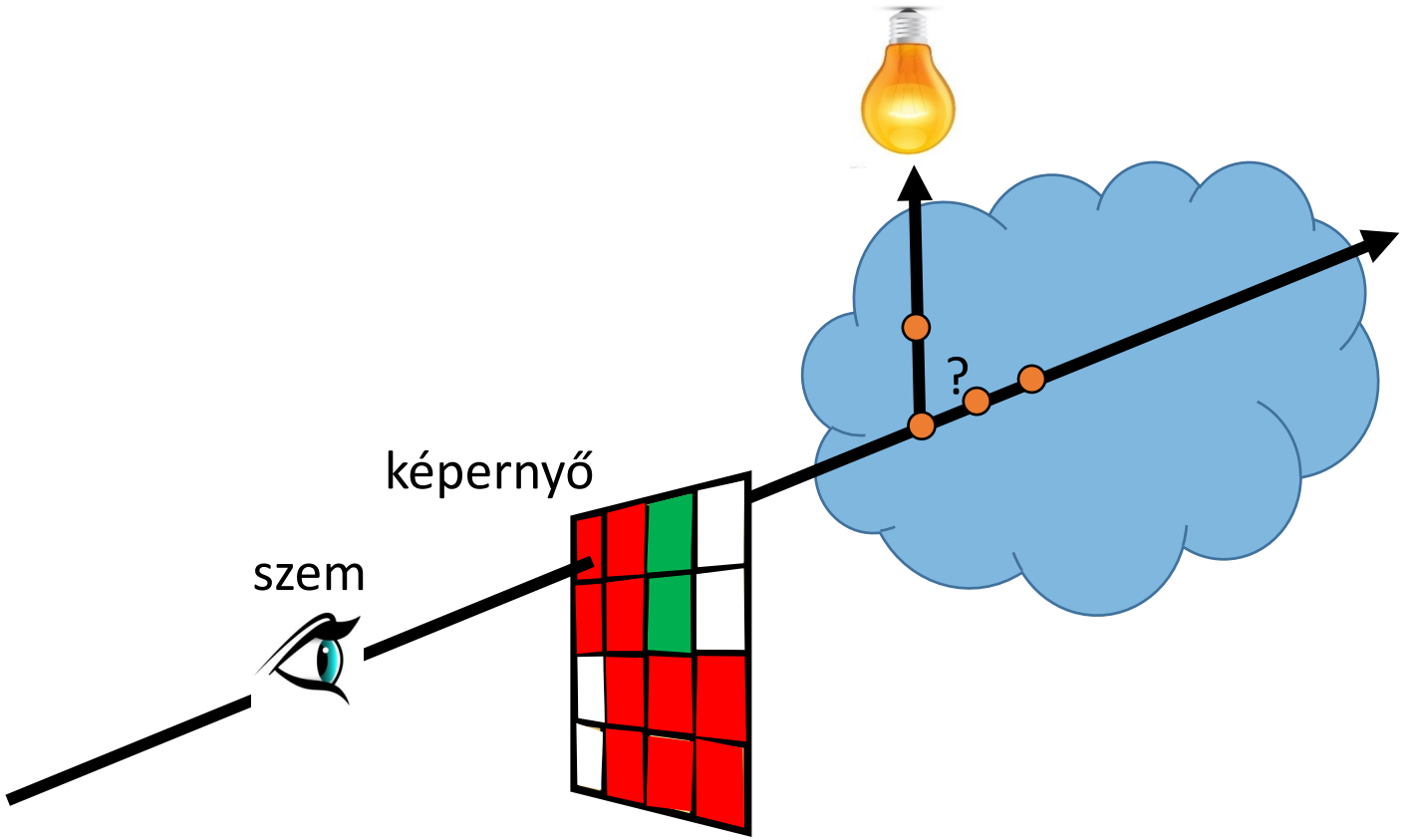
$$\frac{\mathrm{d}L}{\mathrm{d}s} = -\mu \cdot L + q$$

Megoldás konstans csillapítás és forráshatás esetén

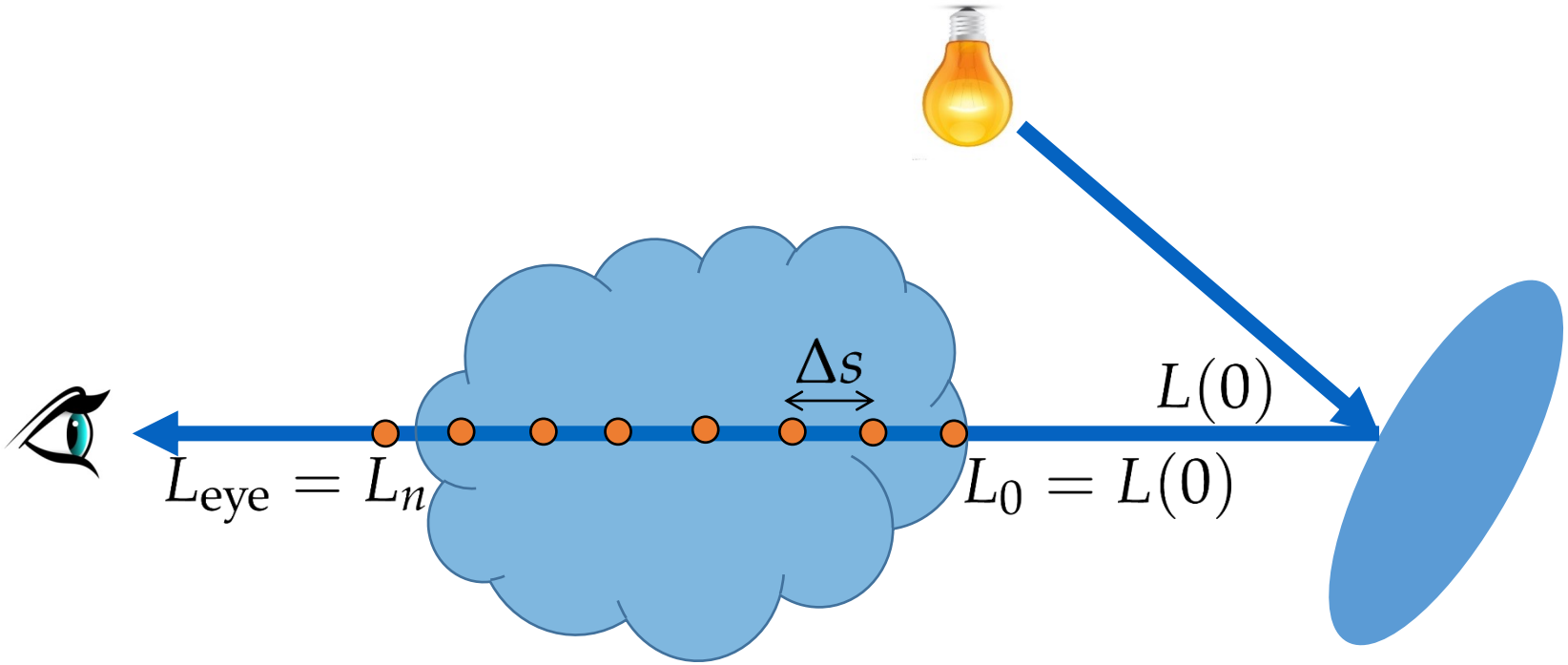
$$\frac{\mathrm{d}L}{\mathrm{d}s} = -\mu \cdot L + q$$

$$L(s) = L(0) \cdot e^{-\mu s} + \frac{q \cdot (1 - e^{-\mu s})}{\mu}$$

Térfogati képalkotás



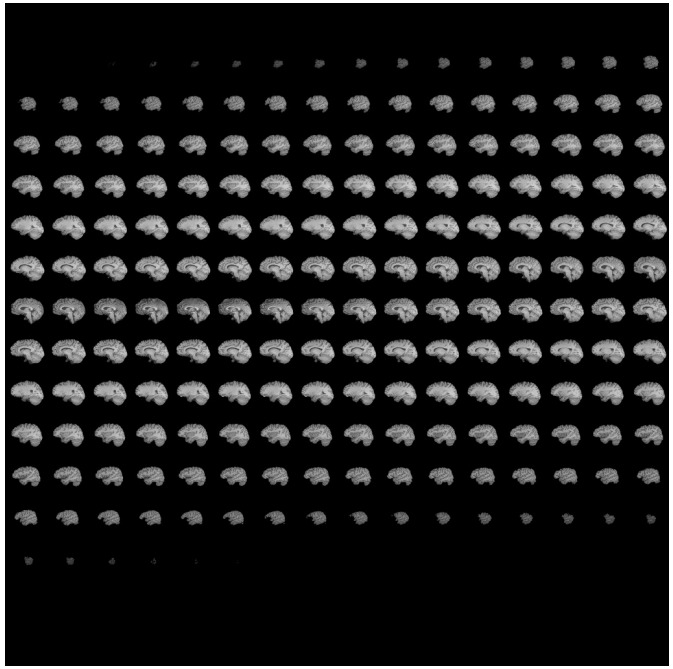
Sugármenetelés – ray marching



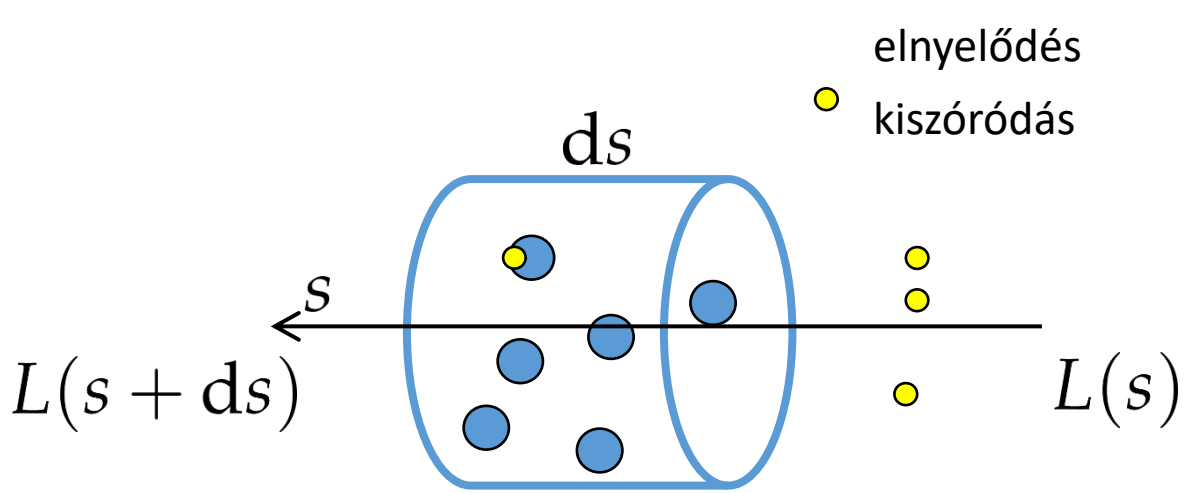
$$L_{i+1} = L_i + \Delta s \cdot (q(s) - L_i \cdot \mu(s))$$

Optikai paraméterek tárolása – térfogati adatmodellek

- implicit egyenlettel
- egyszerű implicit egyenletek összegeként
  - metaball
  - részecske-alapú térfogatreprezentáció
- térfogati rácson
  - voxeltömb
- részecskék + plakátok



Köd



radiancia változása

csillapítási tényező

forrástag

$$\frac{\mathrm{d}L}{\mathrm{d}s} = -\sigma \cdot L + q$$

egység hosszon a sugár mentén

radiancia épp itt

Diffegyenlet megoldása

elveszett százalék

$$L(s) = L(0) \cdot e^{-\sigma s} + \frac{q \cdot (1 - e^{-\sigma s})}{\sigma}$$

radiancia a sugár-felület metszéspontban

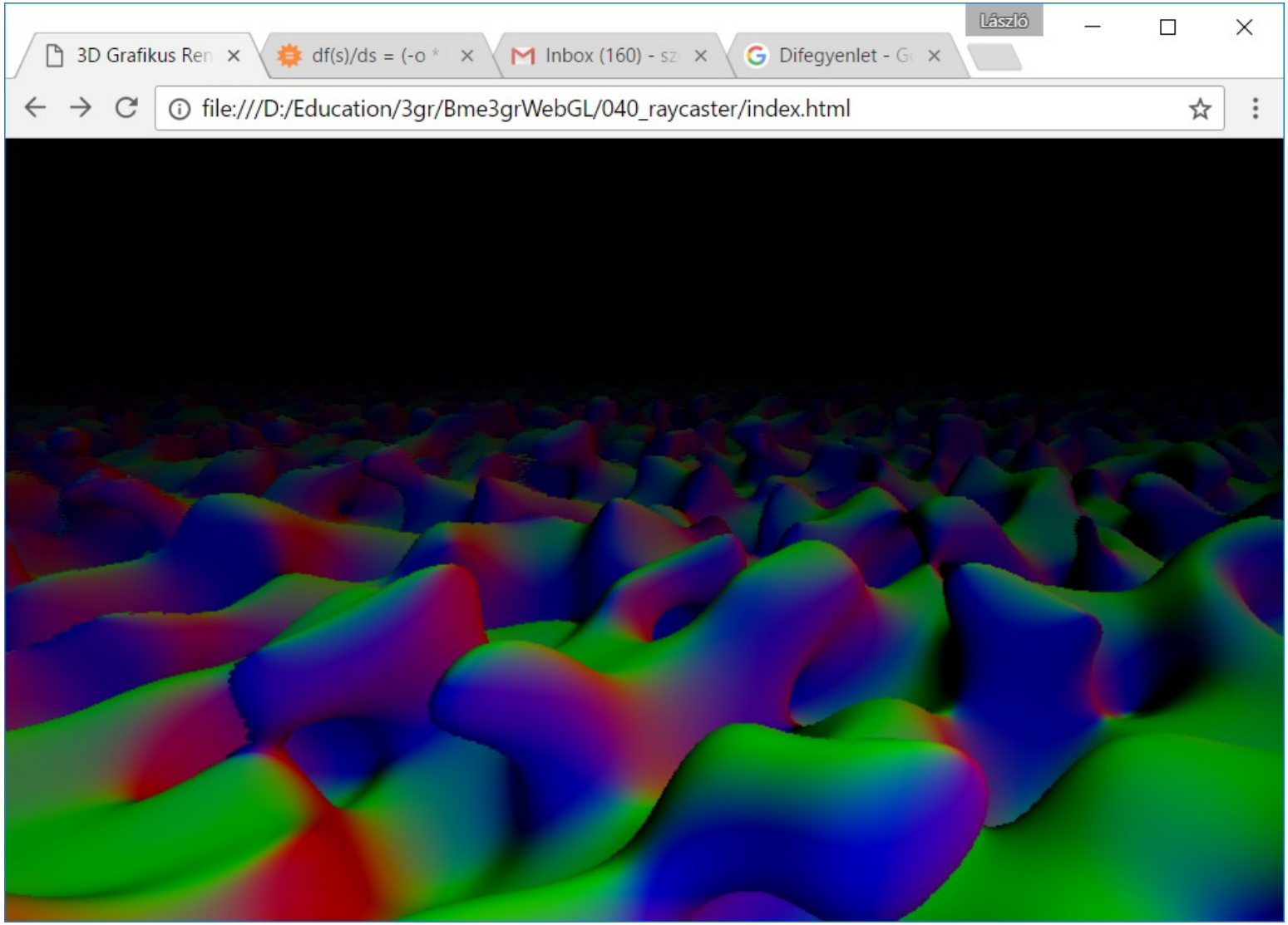
radiance a sugár kezdőpontjában, a metszésponttól s távolságra

összeszedett többletradiancia

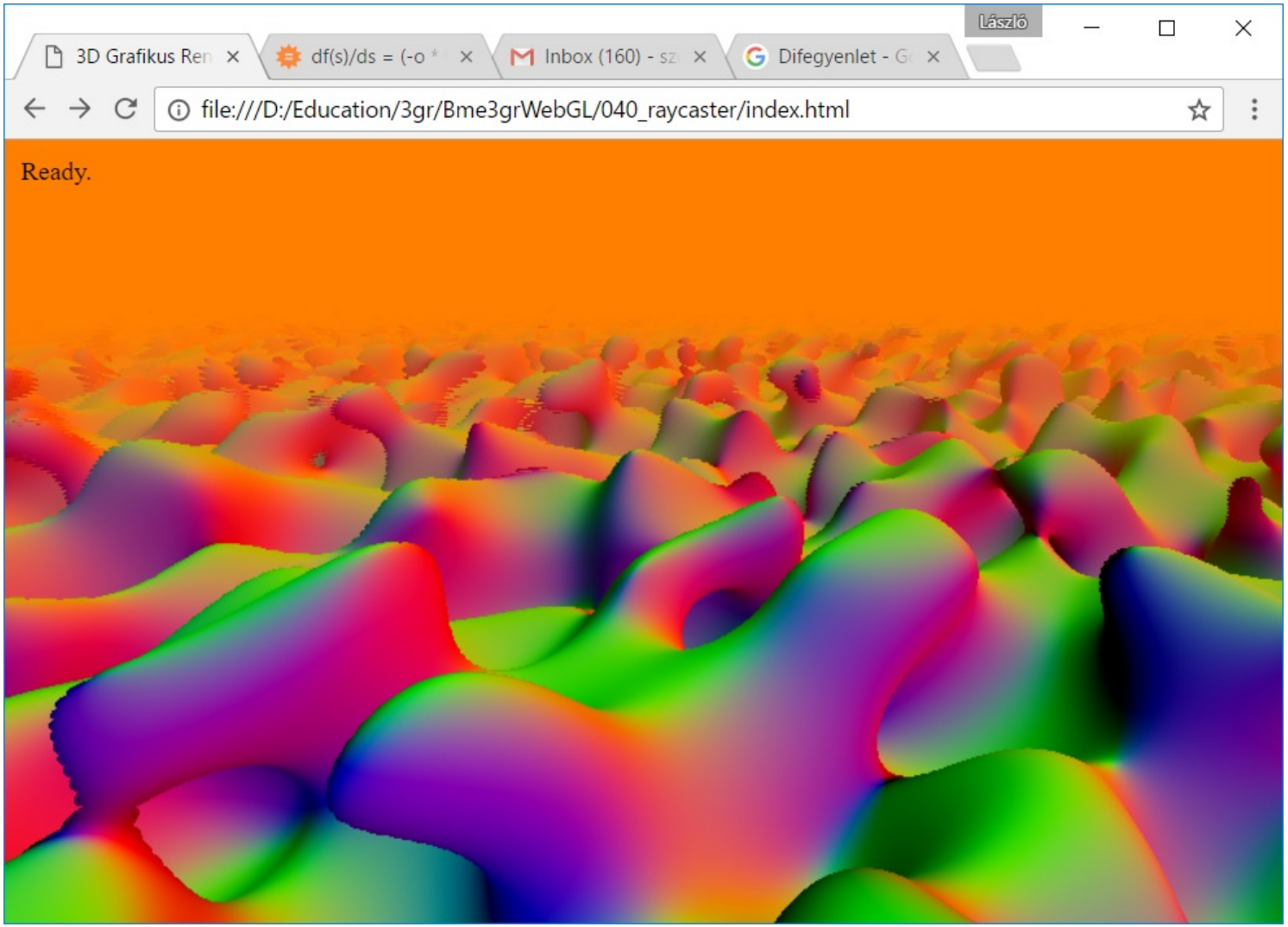
Köd shaderben

- válasszunk csillapítást és forrástagot, tudjuk ezeket könnyen változtatni
  - ergo érdemes ezekre változót gyártani
  - lehet skalár, vagy RGB, ha színes a köd
- a fenti képletet értékeljük ki a FS végefelé
- a háttér végtelen messze legyen

Nulla forráshatás, fekete köd



Színes köd, nem látszik a háttér (nyilván)



Nem-konstans köd

- lejjebb sűrűbb, feljebb ritkább, exponenciálisan

$$\sigma(y) = \sigma_0 e^{b \cdot y}$$

$$q(y) = q_0 e^{b \cdot y}$$

- y a sugáron

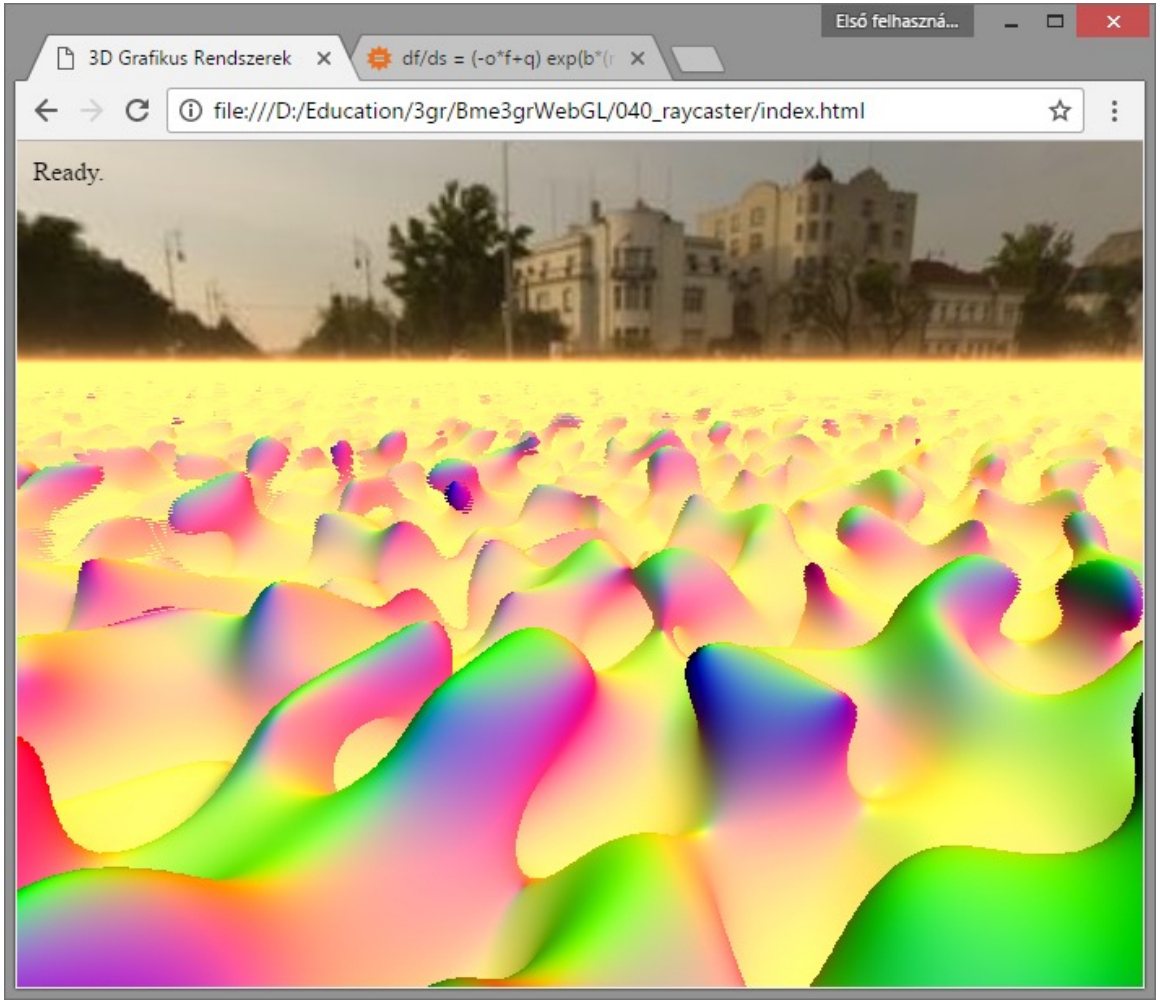
$$y(s) = y_{\text{ray\_origin}} + y_{\text{ray\_dir}} \cdot s$$

$$\frac{\mathrm{d}L}{\mathrm{d}s} = (-\sigma \cdot L + q) e^{b \cdot \left(y_{\text{ray\_origin}} + y_{\text{ray\_dir}} \cdot s\right)}$$

Diffegyenlet megoldás

$$L(s) = \frac{L(0)\cdot\sigma - q}{\sigma} e^{-\frac{\sigma}{b\cdot y_{\text{ray\_dir}}}} \left(e^{b\cdot\left(y_{\text{ray\_origin}} + s y_{\text{ray\_dir}}\right)} - e^{b\cdot y_{\text{ray\_origin}}} \right) + \frac{q}{\sigma}$$

Várt eredmény



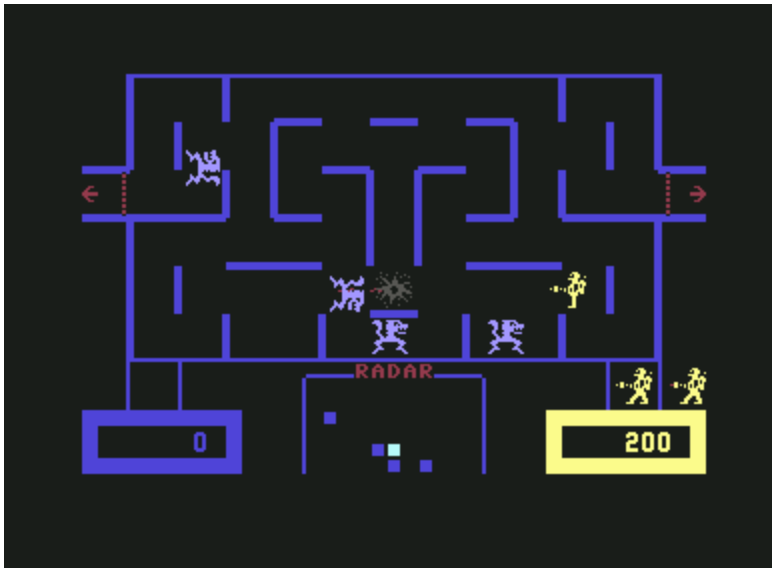
Plakátok, részecskerendszerek

Képalapú festés

- Montázs: képet képekből
- 2D grafika jellemző eszköze
  - modell: kép [sprite]
- 3D
  - 2D képével helyettesítsük a komplex geometriát
  - Image-based rendering

A sprite evolúciója

- C64
  - 2D grafika
  - hardver támogatás: 8db 24x21-es bitmap kirakása adott pixelkoordinátákra
  - nincs elforgatás, skálázás
  - blitter: bit block transfer
    - sprite mem → frame buffer



A sprite evolúciója

- modern grafikus pipelineban
  - pont primitívek rajzolása [point sprite]
  - pont mérete [p×p pixeles négyzet]:
    - D3DRS_POINTSIZE
  - pixel shaderben olvasható a kép mint textúra
  - Shader Model 4.0
    - point sprite már nincs, mivel háromszögekkel is megoldható (minek foglalja a tranzisztorokat)

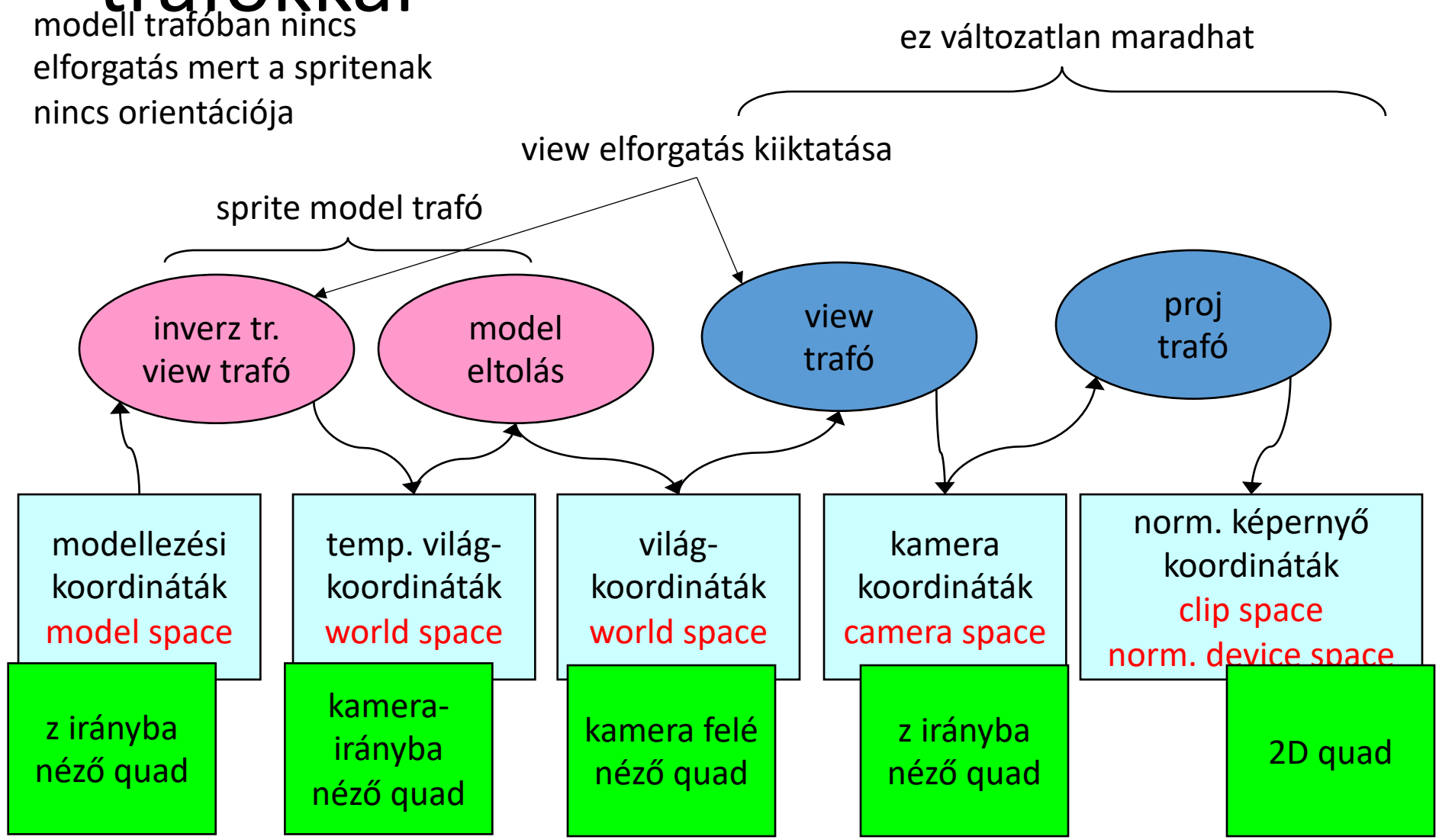
A sprite evolúciója

- 3D grafikában
  - a 2D sprite 3D geometriát helyettesít [impostor]
  - képernyőtérbeli quad, de
    - skálázható, mérete a távolság függvényében változhat
    - rendelhető hozzá mélység, z-teszt használható rá
  - világkoordinátákban adott pozíció, méret, de mindig a kamera felé fordul

Sprite rajzolása klasszikus trafókkal

- a sprite modellje egy z irányba néző quad
  - a világban úgy forgatjuk a quadot, hogy a kamera fele nézzen
  - a kamera trafó forgatás részét ki kell iktatni
  - kamera trafó inverz transzponáltjával kell a quad vertexeit transzformálni
  - utána jöhet a kamera trafó és projekció
- csak akkor van értelme ha a kamera trafó végrehajtásába nem tudunk belenyúlni

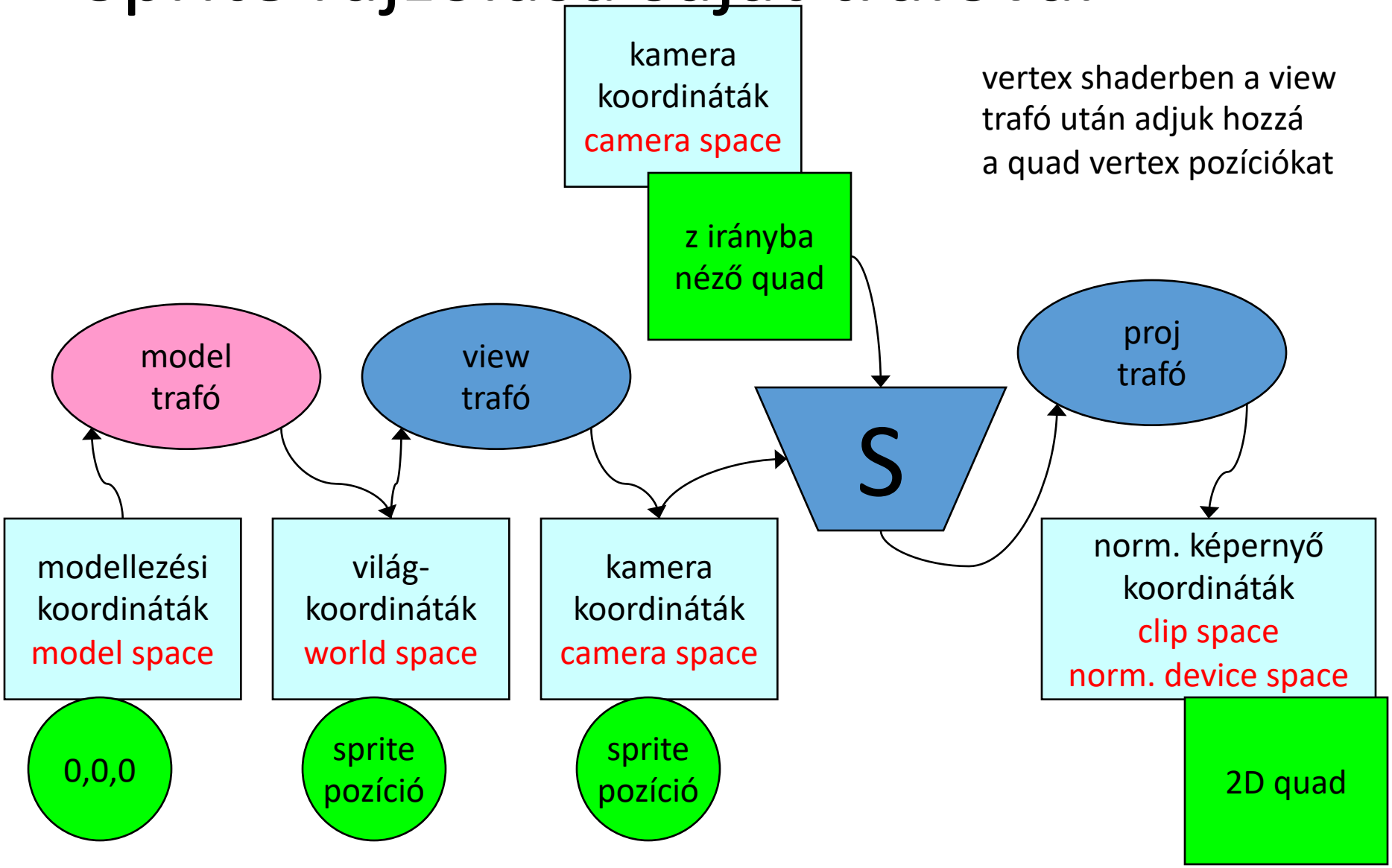
Sprite rajzolása klasszikus trafókkal



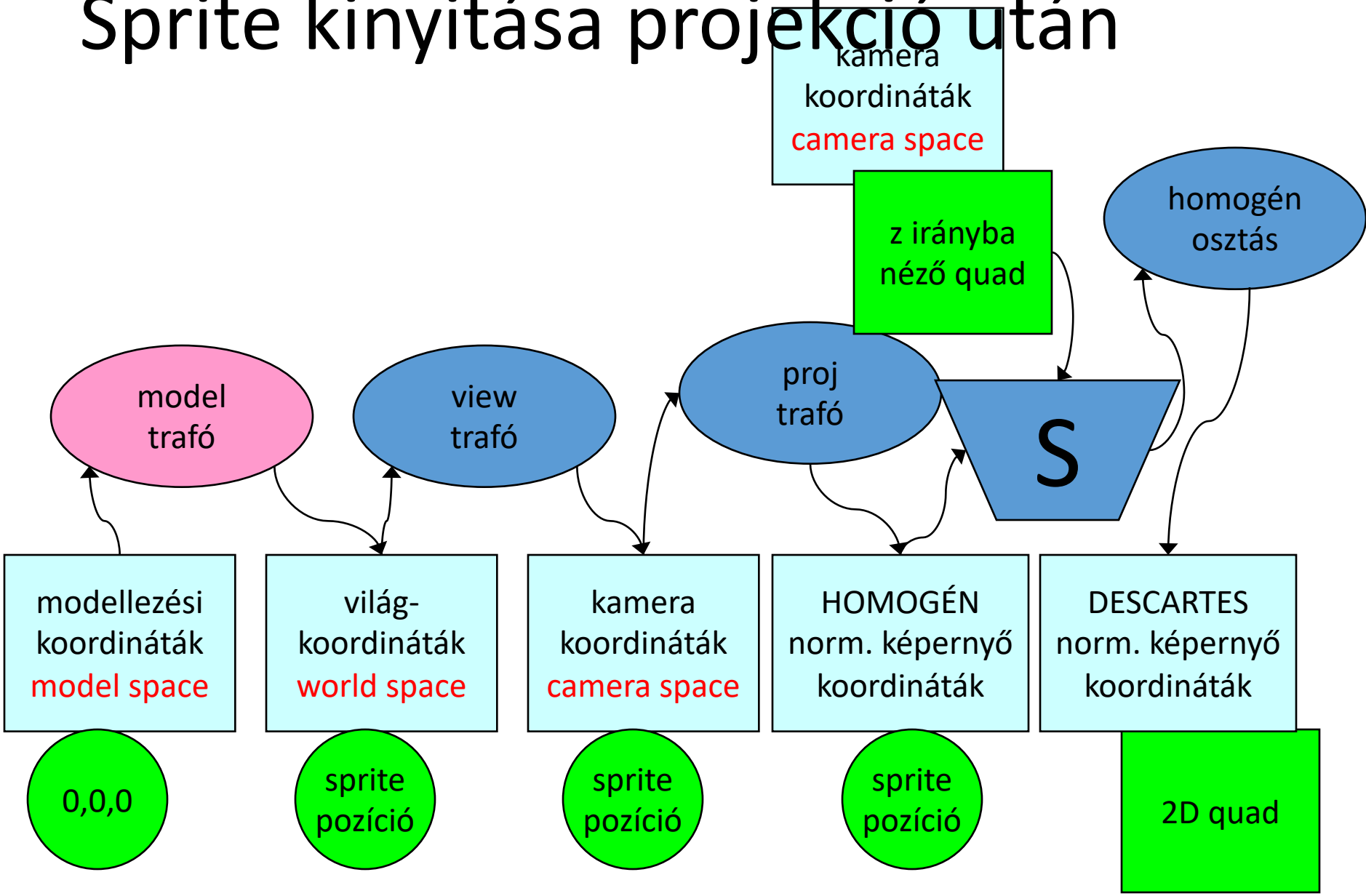
Sprite rajzolása saját trafóval

- a modellezési origót [sprite középpontjának a pozícióját] transzformáljuk a kamera koodináta-rendszerbe
  - kamera koordinátákban adjuk hozzá a csúcsok (spriteon belüli) pozícióit
  - utána jöhet a proj trafó

Sprite rajzolása saját trafóval



Sprite kinyitása projekció után



Sprite rajzolása Geometry Shaderrel - VS

```
// pontprimitívek
struct IaosBillboard
{
    float3 pos : POSITION;
    float lifespan : LIFESPAN;
    float age : AGE;
};

typedef IaosBillboard VsosBillboard;

VsosBillboard vsBillboard(IaosBillboard input)
{
    return input;
}
```

Sprite rajzolása Geometry Shaderrel – GS

```
struct GsosBillboard
{
    float4 pos : SV_Position;
    float2 tex : TEXCOORD;
};

cbuffer perSwapChain {
float4 billboardSize;

};
```

```
float4 worldBillboardSize(50.0, 50.0, 0, 0);
float4 screenBillboardSize =
       worldBillboardSize * firstPersonCam->getProjMatrix();
```

Sprite rajzolása Geometry Shaderrel

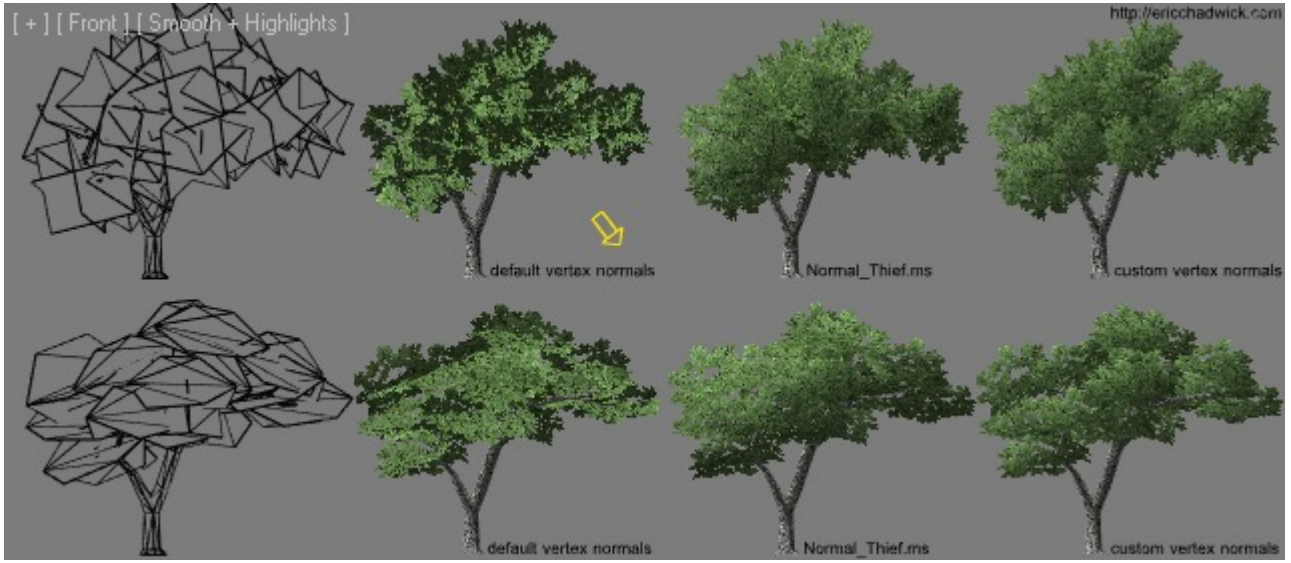
```
[maxvertexcount(4)]
void gsBillboard(point VsosBillboard input[1],
                 inout TriangleStream<GsosBillboard> stream) {
float4 hndcPos= mul(float4(input[0].pos, 1), modelViewProjMatrix);
GsosBillboard output;
output.pos = hndcPos;
output.pos.x += billboardWidth;
output.pos.y += billboardHeight;
output.tex = float2(1, 0);
stream.Append(output);
output.pos = hndcPos;
output.pos.x += billboardWidth;
output.pos.y -= billboardHeight;
output.tex = float2(1, 1);
stream.Append(output);
output.pos = hndcPos;
output.pos.x -= billboardWidth;
output.pos.y += billboardHeight;
output.tex = float2(0, 0);
stream.Append(output);
output.pos = hndcPos;
output.pos.x -= billboardWidth;
output.pos.y -= billboardHeight;
output.tex = float2(0, 1);
stream.Append(output);
}
```

Plakát [billboard]

- sprite mindig teljesen a kamera fele néz
- a plakát is egy textúrázott quad
  - lehet statikus
  - foroghat 1 tengely körül
  - illeszkedhet felületre stb.
- billboard általánosabb, mint a sprite

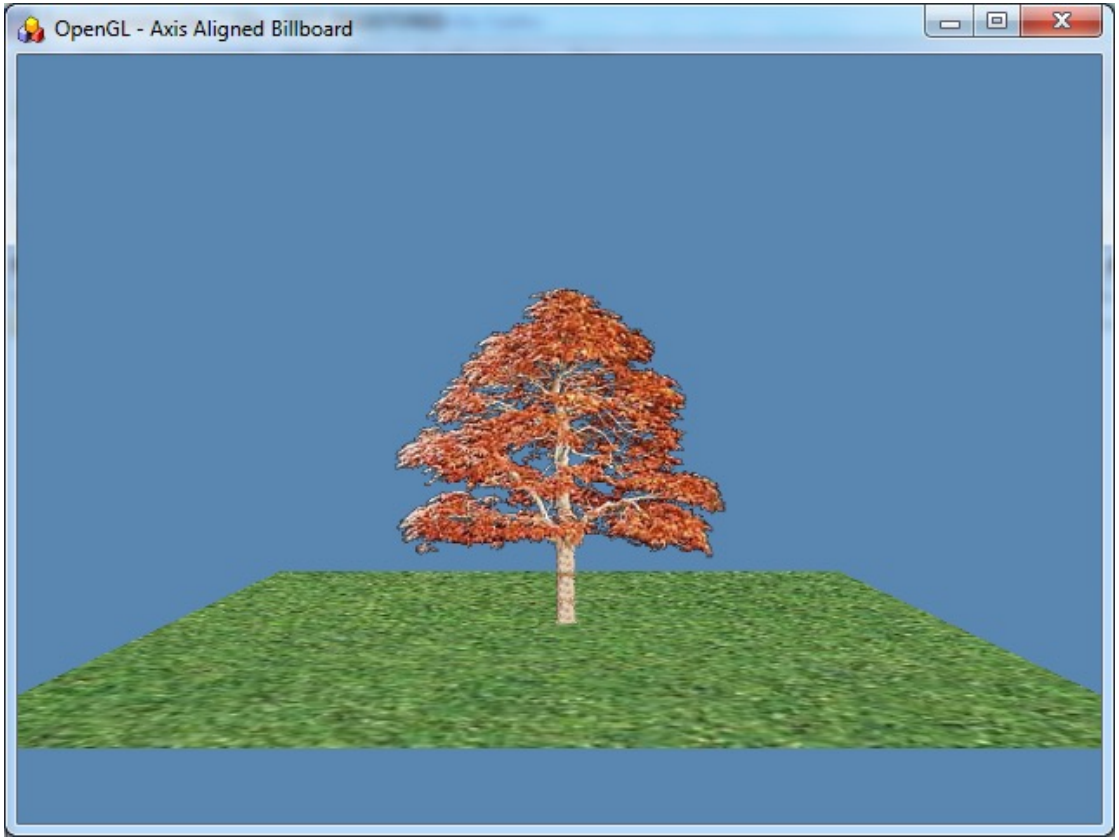
Statikus plakát

- 3D objektumok megjelenítésére önmagában gyenge, ha rossz irányból nézünk rá
- sok felületből már meggyőző modell lehet
- plakátfelhő [billboard cloud]



Tengely körül forgó plakát

- tipikus alkalmazás: fák



Részecske-rendszer

- térfogati jelenségek
  - pl. tűz, füst, köd, por, sugárhajtómű
  - bonyolult fizika, áramlás-szimuláció, Navier-Stokes difegyenlet helyett
  - szimuláljuk néhány részecske útját az áramlásban egyszerű szabályok alapján
  - jelenítsük meg őket mint felhőcskéket
  - áttetsző, textúrázott plakátokkal
    - billboard + blending + instancing [sok ugyanolyan geometria]

Részecskék életciklusa

- születés
  - emitter: pontszerű, vonalas, sík, térfogat
- jellemzők
  - pozíció, sebesség, életkor, méret, szín, átlátszóság
- hatások
  - effector: szél, felhajtóerő, turbulencia, gravitáció
- halál

Részecskerendszer és pontprimitívek

```
struct Particle
{
    Egg::Math::float3 position;
    Egg::Math::float3 velocity;
    float lifespan;
    float age;
    Particle() { reborn(); }
    void reborn() { /*init, emitter logika*/ }
    void move(float dt) {
        position += velocity * dt;
        age += dt;
        if(age > lifespan)   reborn();
    }
};
std::vector<Particle> particles;

Egg::Mesh::ShadedP fireBillboardSet;
```

Input element desc

```
D3D11_INPUT_ELEMENT_DESC particleElements[3];
particleElements[0].AlignedByteOffset =
offsetof(Particle, position);
particleElements[0].Format = DXGI_FORMAT_R32G32B32_FLOAT;
particleElements[0].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
particleElements[0].SemanticName = "POSITION";
particleElements[0].SemanticIndex = 0;
particleElements[0].InputSlot = 0;
particleElements[0].InstanceDataStepRate = 0;
```

Input element desc

```
particleElements[1].AlignedByteOffset =
offsetof(Particle, lifespan);
particleElements[1].Format = DXGI_FORMAT_R32_FLOAT;
particleElements[1].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
particleElements[1].SemanticName = "LIFESPAN";
particleElements[1].SemanticIndex = 0;
particleElements[1].InputSlot = 0;
particleElements[1].InstanceDataStepRate = 0;
particleElements[2].AlignedByteOffset =
offsetof(Particle, age);
particleElements[2].Format = DXGI_FORMAT_R32_FLOAT;
particleElements[2].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
particleElements[2].SemanticName = "AGE";
particleElements[2].SemanticIndex = 0;
particleElements[2].InputSlot = 0;
particleElements[2].InstanceDataStepRate = 0;
```

Vertex buffer (Egg::Mesh libbel)

```
Egg::Mesh::VertexStreamDesc particleBufferDesc;

particleBufferDesc.elements = particleElements;
particleBufferDesc.nElements = 3;
particleBufferDesc.nVertices = particles.size();
particleBufferDesc.vertexStride = sizeof(Particle);
particleBufferDesc.vertexData = &particles.at(0);
particleBufferDesc.cpuAccessFlags = D3D11_CPU_ACCESS_WRITE;
particleBufferDesc.usage = D3D11_USAGE_DYNAMIC;

Egg::Mesh::VertexStreamP particleVertexStream =
  Egg::Mesh::VertexStream::create(device,
                                  particleBufferDesc);
```

billboard.hlsli

```
struct IaosBillboard {
    float3 pos : POSITION;
    float lifespan : LIFESPAN;
    float age : AGE;
};
typedef IaosBillboard VsosBillboard;
struct GsosBillboard {
    float4 pos : SV_Position;
    float2 tex : TEXCOORD;
};
cbuffer perSwapChain{
  float4 billboardSize;
}
```

vsBillboard.hlsl

```
VsosBillboard vsBillboard(IaosBillboard input)
{
    return input;
}
```

gsBillboard.hlsl

```
[maxvertexcount(4)]
void gsBillboard(
    point VsosBillboard input[1],
    inout TriangleStream<GsosBillboard> stream) {
    float4 hndcPos = mul(float4(input[0].pos, 1),
                         modelViewProjMatrix);
    GsosBillboard output;
    output.pos = hndcPos;
    output.pos.x += billboardSize.x;
    output.pos.y += billboardSize.y;
    output.tex = float2(1, 0);
    stream.Append(output);
output.pos = hndcPos;
output.pos.x += billboardSize.x;
output.pos.y -= billboardSize.y;
output.tex = float2(1, 1);
stream.Append(output);
output.pos = hndcPos;
output.pos.x -= billboardSize.x;
output.pos.y += billboardSize.y;
output.tex = float2(0, 0);
stream.Append(output);
output.pos = hndcPos;
output.pos.x -= billboardSize.x;
output.pos.y -= billboardSize.y;
output.tex = float2(0, 1);
stream.Append(output); }
```

psFire.hlsl

```
Texture2D billboardTexture;
SamplerState ss;

float4 psFire(GsosBillboard input) : SV_Target
{
    return billboardTexture.Sample(
        ss,
        input.tex.xy);
}
```

Mesh::Shaded

```
Egg::Mesh::MaterialP fireMaterial
  = Egg::Mesh::Material::create();
// load shaders here
fireMaterial->setShader(Egg::Mesh::ShaderStageFlag::Vertex,
   billboardVertexShader);
fireMaterial->setShader(Egg::Mesh::ShaderStageFlag::Geometry,
   billboardGeometryShader);
fireMaterial->setShader(Egg::Mesh::ShaderStageFlag::Pixel,
   firePixelShader);

fireMaterial->setShaderResource("billboardTexture",
       bbSrv);
fireMaterial->setConstantBuffer("perSwapChain",
       perSwapChainConstantBuffer);
fireMaterial->setConstantBuffer("perObject",
       perObjectConstantBuffer);
```

Mesh::Shaded

```
ComPtr<ID3D11InputLayout> billboardInputLayout =
inputBinder->getCompatibleInputLayout(
    billboardVertexShaderByteCode,
    particleVertexStream);
fireBillboardSet = Egg::Mesh::Shaded::create(
   particleVertexStream, fireMaterial,
   billboardInputLayout);
```

Adatok feltöltése (animate)

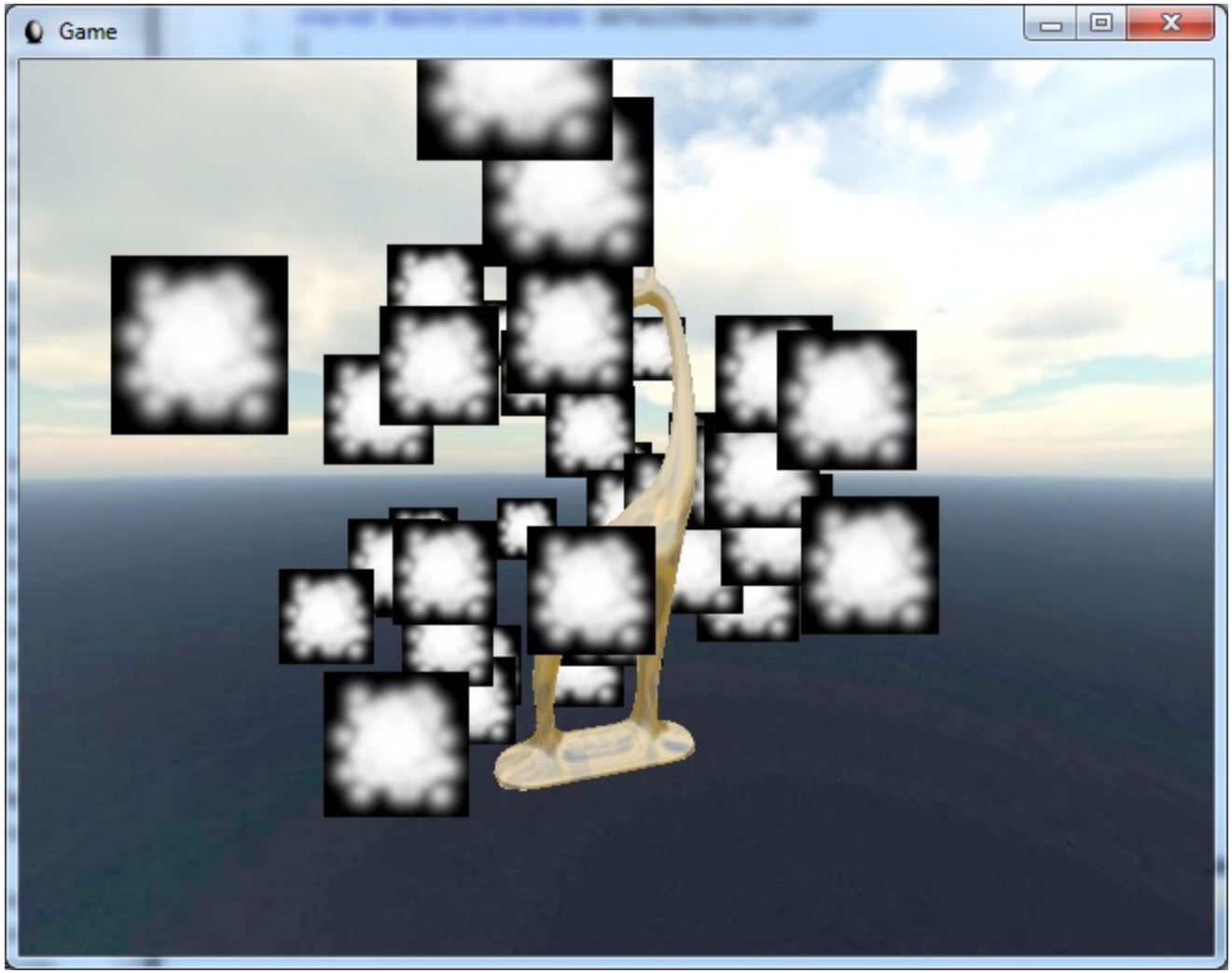
```
using namespace Microsoft::WRL;

ComPtr<ID3D11Buffer> vertexBuffer = fireBillboardSet
        ->getGeometry()
        ->getPrimaryBuffer();
ComPtr<ID3D11DeviceContext> context;
device->GetImmediateContext(context.GetAddressOf());
D3D11_MAPPED_SUBRESOURCE mappedVertexData;
context->Map(vertexBuffer.Get(), 0, D3D11_MAP_WRITE_DISCARD,
  0, &mappedVertexData);

memcpy(mappedVertexData.pData, &particles.at(0),
  sizeof(Particle) * particles.size());

context->Unmap(vertexBuffer.Get(), 0);
```

Plakátok



Blending

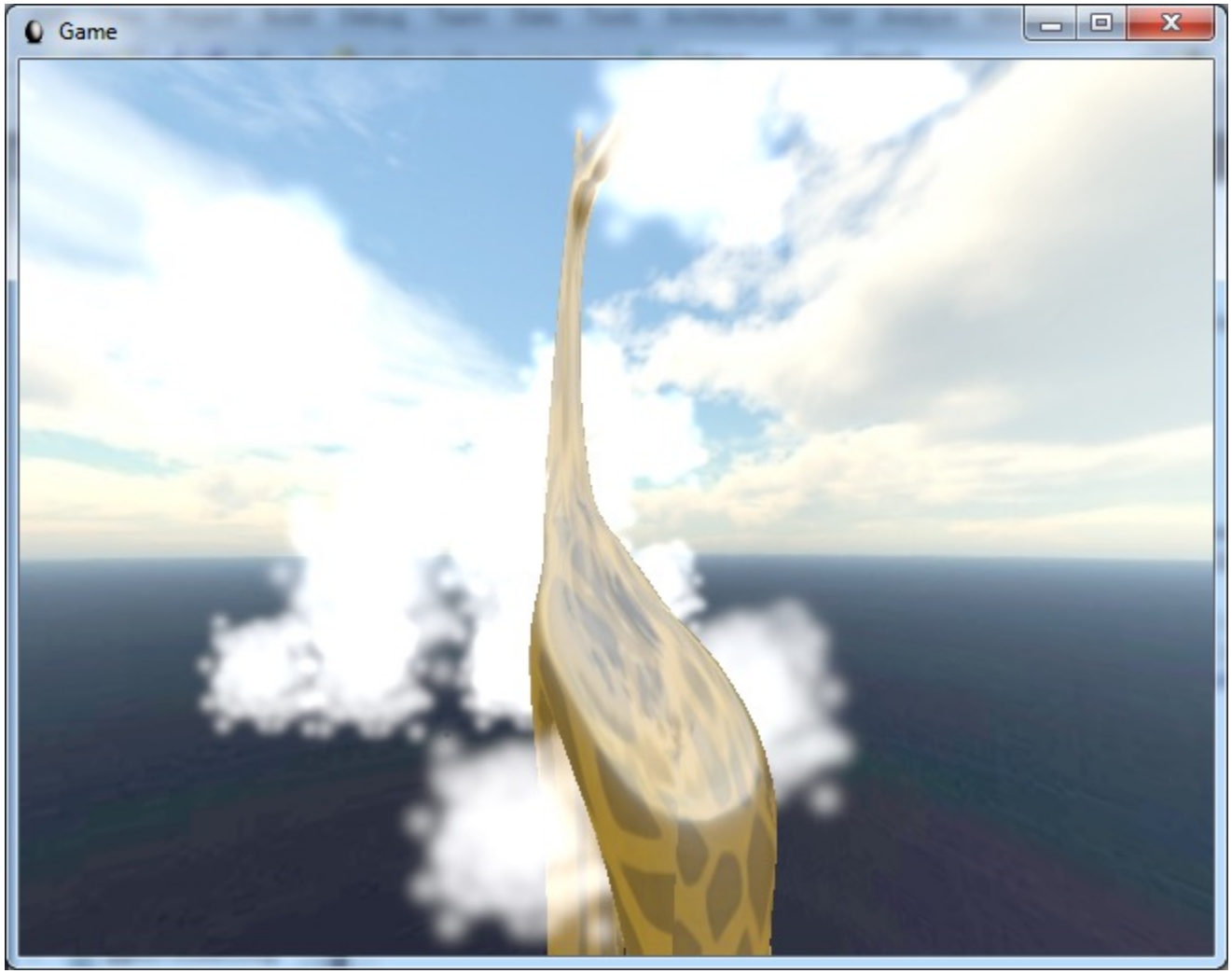
```
Microsoft::WRL::ComPtr<ID3D11BlendState> additiveBlender;
CD3D11_BLEND_DESC additiveBlendDesc;
additiveBlendDesc.IndependentBlendEnable = FALSE;
additiveBlendDesc.RenderTarget[0] = {
TRUE,
D3D11_BLEND_SRC_ALPHA, D3D11_BLEND_ONE, D3D11_BLEND_OP_ADD,
D3D11_BLEND_SRC_ALPHA, D3D11_BLEND_ONE, D3D11_BLEND_OP_ADD,
D3D11_COLOR_WRITE_ENABLE_ALL,
};
device->CreateBlendState(&additiveBlendDesc,
       additiveBlender.GetAddressOf());


fireMaterial->setBlendState(additiveBlender);
```

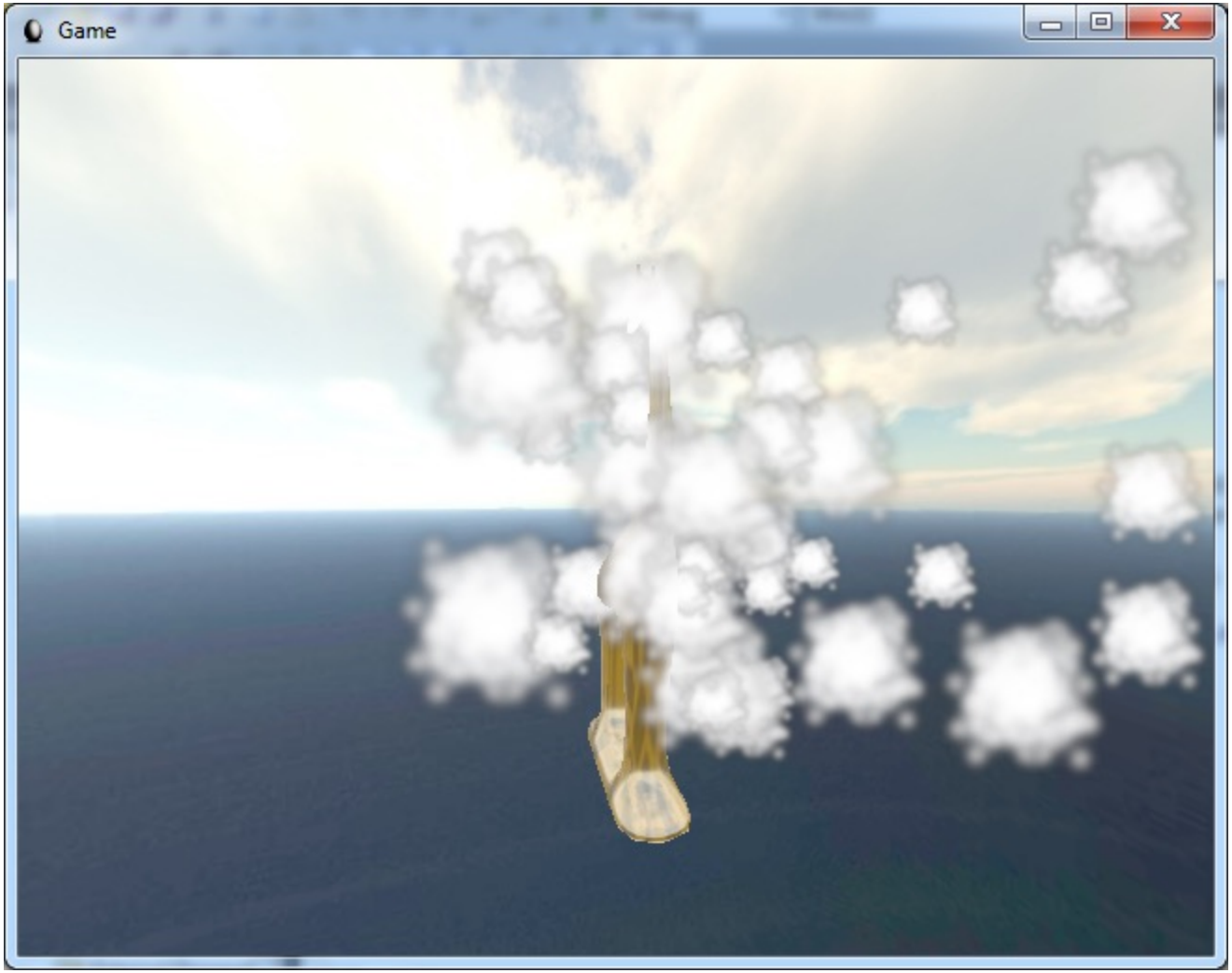
Blending van… de z-teszt is

Nincs depth test – zsiráf sem takar

Z-teszt marad, Z-írás kikapcs



Rossz a sorrend… hátsók takarnak előbbiekre



Rendezés

```
using namespace Egg::Math;
struct CameraDepthComparator {
  float3 ahead;
  bool operator() (const Particle& a, const Particle& b) {
    return ((a.position.dot(ahead)) >
                  (b.position.dot(ahead) + 0.01));
  }
} comp = { firstPersonCam->getAhead() };
std::sort(particles.begin(), particles.end(), comp);
```