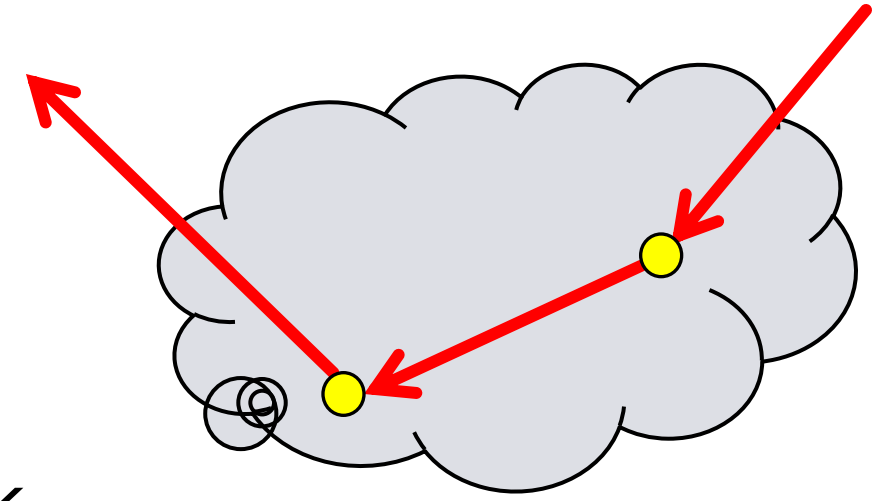
Monte Carlo szimuláció

Szóródás szimuláció

- Fény fotonok szimulációja
  - Nem változtatja meg a frekvenciát ütközéskor
- Homogén és inhomogén közegben

Szóródás szimuláció

- A megoldandó probléma
  - Szóródási pont meghatározása
    - Szabad úthossz
  - Energia változás
  - Új irány számítása

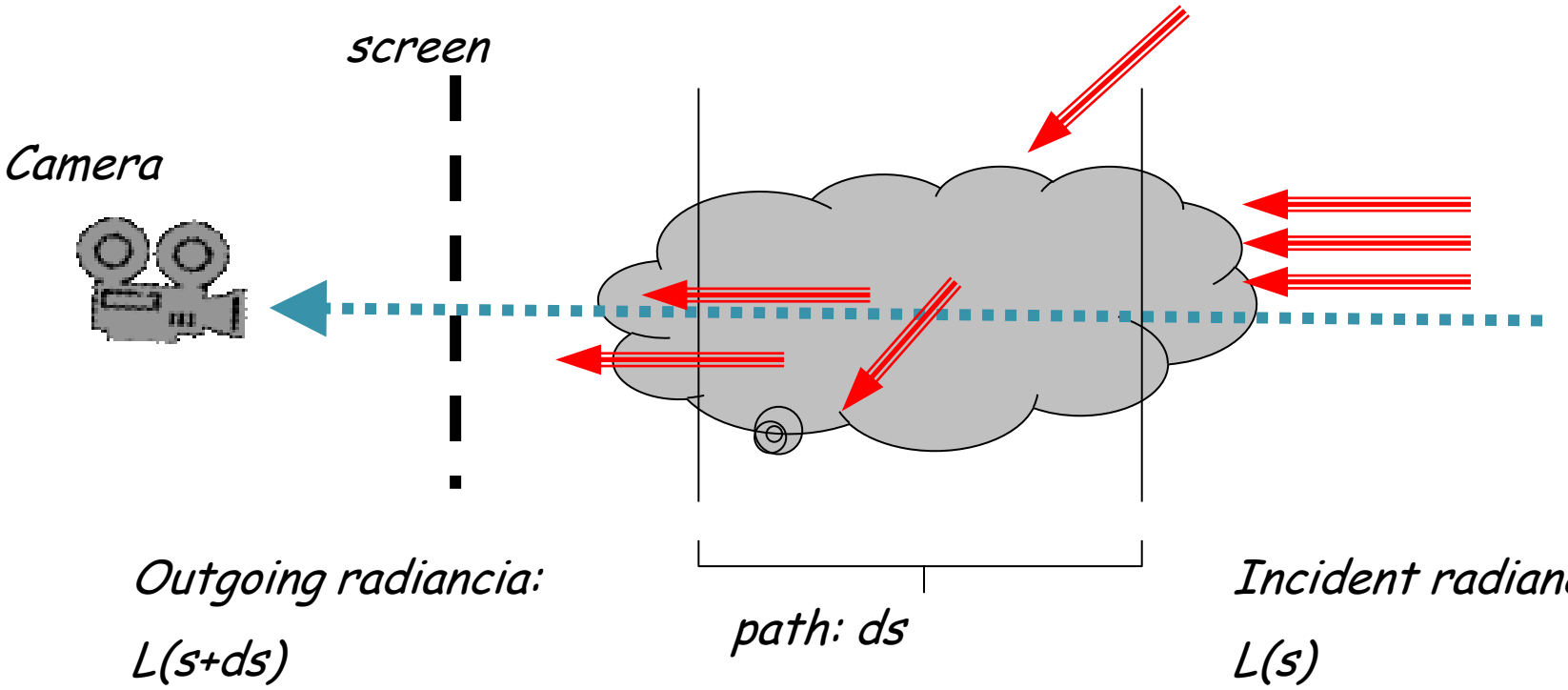


- Monte Carlo módszerek
  - Véletlen minták
  - Döntés lokális tulajdonságok alapján

Szóródás szimuláció

- Radiative transport egyenlet

$$\vec{\omega} \cdot \vec{\nabla} L = \left. \frac{dL(\vec{x} + \vec{\omega} s, \vec{\omega})}{ds} \right|_{s=0} = -\sigma_t(\vec{x}) L(\vec{x}, \vec{\omega}) + \sigma_s(x) \int_{\Omega} L(\vec{x}, \vec{\omega}') P(\vec{\omega}' \cdot \vec{\omega}) d\omega'$$



Szóródás szimuláció

- A szóródás valószínűsége
  - $\sigma_t(\vec{x}) = \sigma_s(\vec{x}) + \sigma_a(\vec{x})$
- A visszaverődés valószínűsége
  - albedo $a = \frac{\sigma_s}{\sigma_t}$

Szóródás szimuláció

- A visszaverődés iránya
  - fázis függvény $P(\cos\theta) \quad \cos\theta = \vec{\omega}'\cdot\vec{\omega}$
  - izotróp szóródás $P_{iso}(\vec{\omega}'\cdot\vec{\omega}) = \frac{1}{4\pi}$
  - anizotróp szóródás $g = \int_{\Omega} (\vec{\omega}'\cdot\vec{\omega}) P(\vec{\omega}'\cdot\vec{\omega}) d\vec{\omega}'$

$$P_{Rayleigh}(\vec{\omega}'\cdot\vec{\omega}) = \frac{3}{16\pi}\left(1+(\vec{\omega}'\cdot\vec{\omega})^2\right) = \frac{3}{16\pi}\left(1+\cos^2\theta\right)$$

Szóródás szimuláció

- Egyszeres szóródás közelítés
  - Radiancia $L(\vec{x},\vec{\omega}) = L_d(\vec{x},\vec{\omega}) + L_m(\vec{x},\vec{\omega})$
    - direkt tag $\frac{dL_d}{ds} = -\sigma_t(\vec{x})L_d(\vec{x},\vec{\omega})$
    - indirekt (media) tag

$$\frac{dL_m}{ds} = -\sigma_t(\vec{x})L_m(\vec{x},\vec{\omega}) + \sigma_s(\vec{x})\int_\Omega \left(L_d(\vec{x},\vec{\omega}) + L_m(\vec{x},\vec{\omega}')\right)P(\vec{\omega}'\cdot\vec{\omega})d\omega'$$

      - térfogati megfogalmazás

$$\frac{dL_m}{ds} = -\sigma_t(\vec{x})L_m(\vec{x},\vec{\omega}) + \sigma_s(\vec{x})\int_\Omega \left(L_d(\vec{x},\vec{\omega}) + L_m(\vec{x},\vec{\omega}')\right) + \sigma_s(\vec{x})Q(\vec{x},\vec{\omega})$$

$$Q(\vec{x},\vec{\omega}) = \int_\Omega L_d(\vec{x},\vec{\omega})P(\vec{\omega}'\cdot\vec{\omega},\vec{x})d\omega'$$

Szóródás szimuláció

- Sztochasztikus megoldás
  - Egy pont radianciájához hozzájárul
    - Direkt foton kibocsátás
    - Egyszeresen szóródott fotonok
    - Kétszeresen szóródott fotonok
    - …
    - N-szeresen szóródott fotonok

Szóródás szimuláció

- Monte Carlo szimuláció
  - Véletlen fényutak generálása
    - Egy út sok fotont tartalmaz
    - A szóródási pontok és energiájuk gyűjtése
    - Szabad úthossz

$$P(s) = 1 - e^{-\int_0^s \sigma_t(\vec{x} + \vec{\omega} S)\, dS}$$

$$\sum_{i=0}^{n-1} \sigma_t(\vec{x} + \vec{\omega}_i \Delta s)\Delta s \leq -\log r_1 \leq \sum_{i=0}^{n} \sigma_s(\vec{x} + \vec{\omega}_i \Delta s)\Delta s$$

Szóródás szimuláció

- Monte Carlo szimuláció
  - Valószínűségi súlyozás
    - A szóródási események konverziója térfogati radianciává

$$L\left(\vec{x},\vec{\omega}_{eye}\right)=\frac{1}{\sigma_t\left(\vec{x}\right)}\frac{d^2\Phi}{d\omega dV}\approx\frac{1}{\sigma_t\left(\vec{x}\right)}\frac{\sum_{i=1}^{m}\Delta\Phi_i P\left(\vec{\omega}_i,\vec{\omega}_{eye}\right)}{\Delta V}$$

  - Gyűjtő séták
    - Nézőpont függő összegzés
    - Indirekt és direkt vizualizációs módszerek

Szóródás szimuláció

- Implementáció
  - Foton struktúra
    - Kezdőpont, irány, energia
  - Radiancia buffer
    - 3D tömb
  - Véletlen szám generátor
    - Kezdő állapotleírás

Szóródás szimuláció

```
__kernel
void simulation(const int iteration, __global uint4* seed,
                __global struct photon* photons,
                const int resolution, __global float* simulationBuffer,
                const float4 lightSourcePosition){
  int id = get_global_id(0);

  // random generator setup
  uint rng1 = seed[id].s0;
  uint rng2 = seed[id].s1;
  uint rng3 = seed[id].s2;
  uint rng4 = seed[id].s3;

  // scatter simulation
  if(0 == iteration || photons[id].energy < 0.2f){
    photons[id].origin = lightSourcePosition;
    photons[id].direction = getRandomDirection(&rng1, &rng2, &rng3, &rng4);
    photons[id].energy = 1.0f;
  } else {
    photons[id].direction = getRandomDirection(&rng1, &rng2, &rng3, &rng4);
  }

  tracePhotonRM(&photons[id], getRandom(&rng1, &rng2, &rng3, &rng4));
  storePhoton(&photons[id], resolution, simulationBuffer);

  // random state store
  seed[id].s0 = rng1;
  seed[id].s1 = rng2;
  seed[id].s2 = rng3;
  seed[id].s3 = rng4;
}
```

Szóródás szimuláció

```
float4 getRandomDirection(uint* rng1, uint* rng2, uint* rng3, uint* rng4){
  float x, y, z;
  bool inside = false;
  while(!inside){
    x = getRandom(rng1, rng2, rng3, rng4) * 2.0f - 1.0f;
    y = getRandom(rng1, rng2, rng3, rng4) * 2.0f - 1.0f;
    z = getRandom(rng1, rng2, rng3, rng4) * 2.0f - 1.0f;
    if( (x*x + y*y + z*z) <= 1.0f){
      inside = true;
    }
  }
  if( x*x + y*y + z*z == 0.0){
    x = 0.0f;
    y = 1.0f;
    z = 0.0f;
  }
  float vlen = sqrt(x*x + y*y + z*z);
  return (float4)(x/vlen, y/vlen, z/vlen, 0);
}
```

Szóródás szimuláció

```
void tracePhotonRM(__global struct photon* p, float rnd){
  // simple linear
  //p->origin = p->origin + p->direction * 0.1f;

  float s = -log(rnd) / densityScale;

  float t = 0.0f;
  float dt = 1.0f / 256.0f;
  float sum = 0.0f;
  float sigmat = 0.0f;

  while(sum < s){
    float4 samplePos = p->origin + t * p->direction;
    if(samplePos.x < 0.0f || samplePos.x > 1.0f ||
       samplePos.y < 0.0f || samplePos.y > 1.0f ||
       samplePos.z < 0.0f || samplePos.z > 1.0f){
      p->energy = 0.0f;
      break;
    } else {
      sigmat = getDensity(samplePos);
      sum += sigmat * dt;
      t += dt;
    }
  }

  p->origin = p->origin + p->direction * t;
  p->direction = p->direction;
  p->energy = p->energy * albedo;
}
```

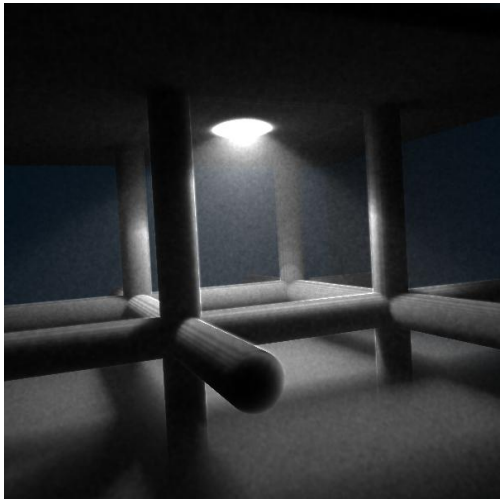
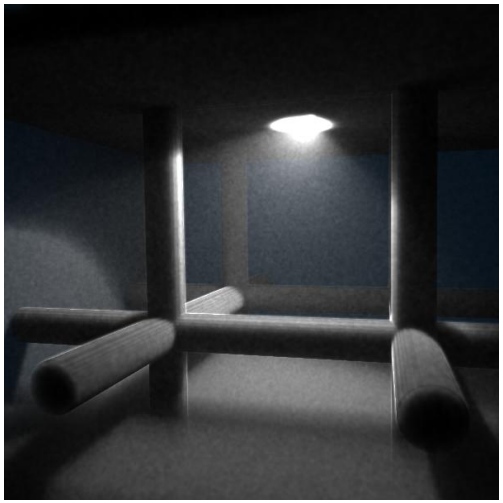
Szóródás szimuláció

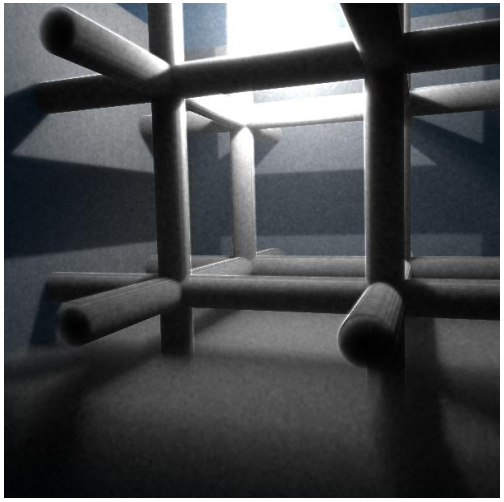
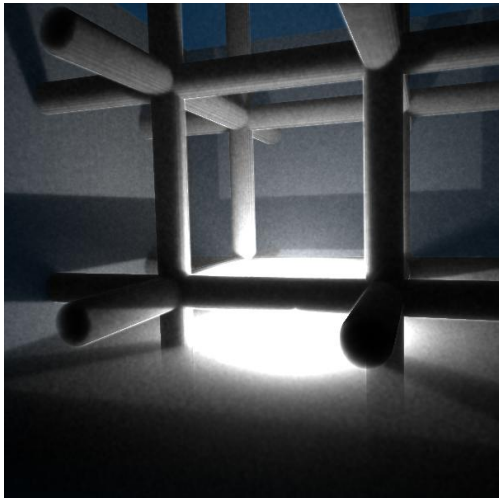
```
float getDensity(float4 p){
  // teto
  if(p.y > 0.78f && p.y < 0.83f &&
     ( (p.x - 0.5f) * (p.x - 0.5f) +
       (p.z - 0.5f) * (p.z - 0.5f) ) > 0.001f)
    return 100.0f;

  // falak
  if(p.x < 0.02f) return 100.0f;
  if(p.y < 0.02f) return 100.0f;
  if(p.z > 0.98f) return 100.0f;

  // alul besurusodik
  if(p.y < 0.2f) return (1.0f - p.y) * 5.0f;

  return 0.5f * densityScale;
}
```

Szóródás szimuláció

```
void storePhoton(__global struct photon* p,
                 const int resolution,
                __global float* simulationBuffer){

  if(p->energy < 0.1f) return;

  int x = p->origin.x * resolution;
  int y = p->origin.y * resolution;
  int z = p->origin.z * resolution;

  if(x > resolution -1 || x < 0) return;
  if(y > resolution -1 || y < 0) return;
  if(z > resolution -1 || z < 0) return;

  simulationBuffer[x+y*resolution+z*resolution*resolution]
                                              += p->energy;
}
```

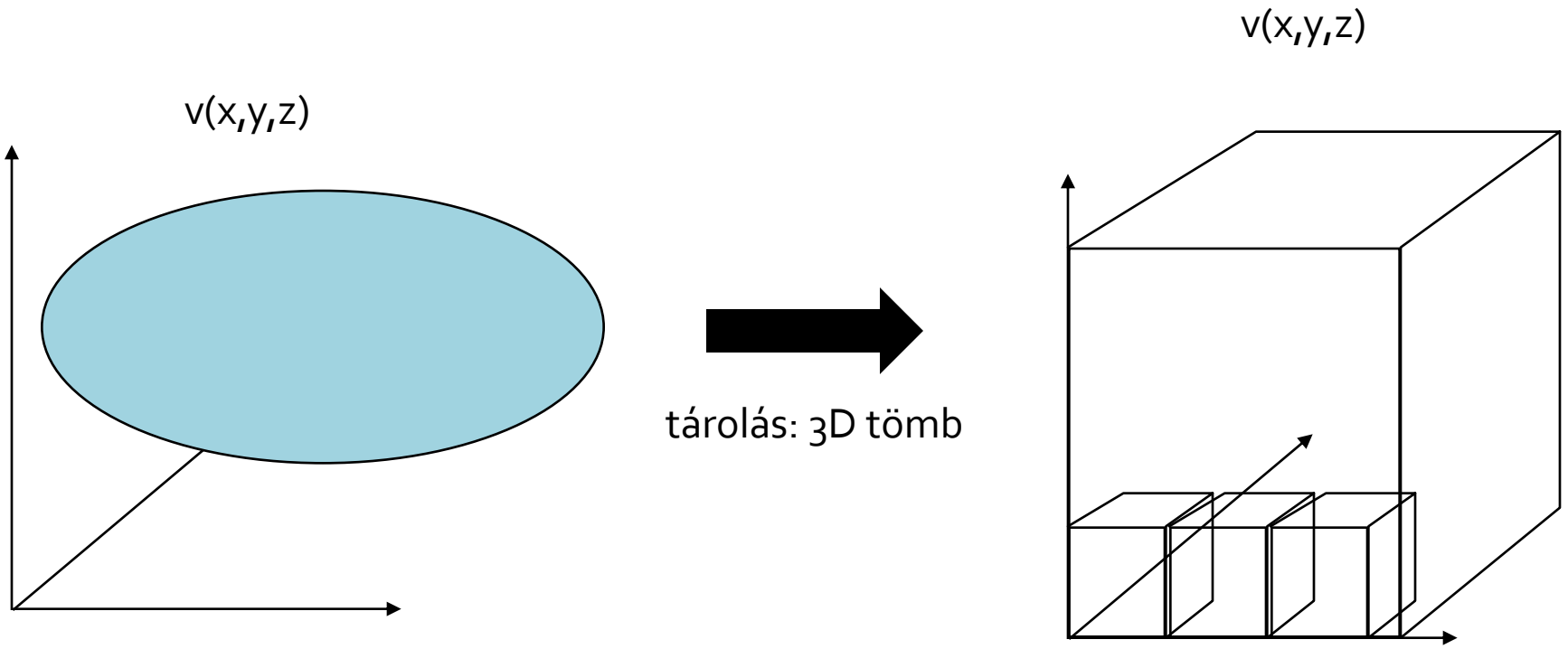
Szóródás szimuláció

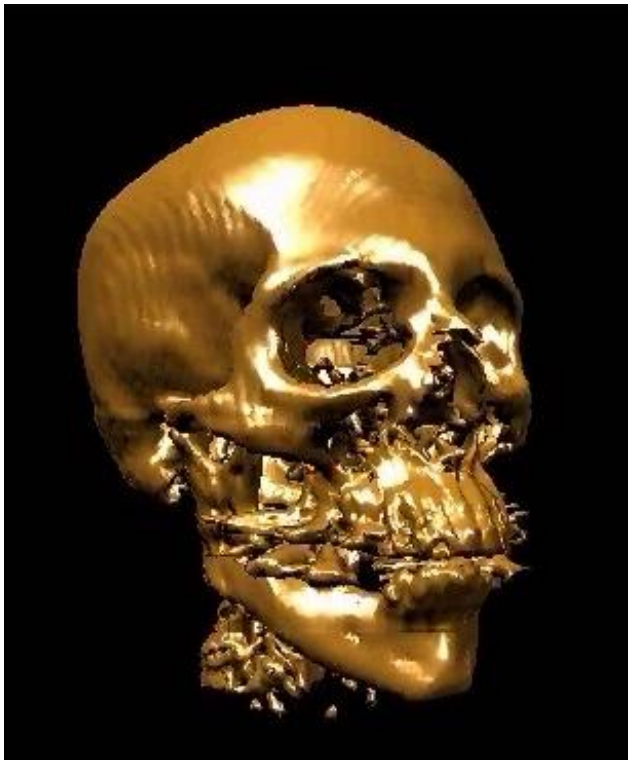
Térfogat vizualizáció

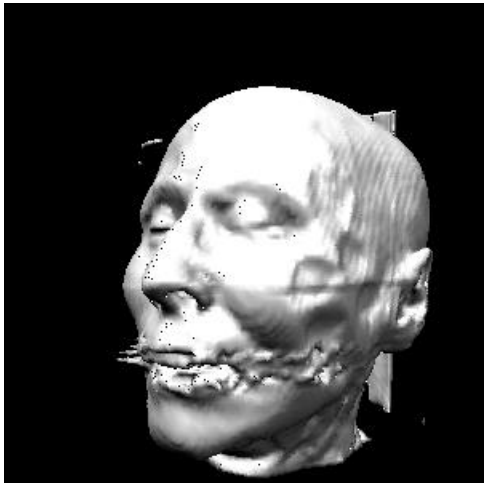
- Adott térfogati reprezentáció megjelenítése
  - hőmérséklet, légnyomás, sűrűség, stb

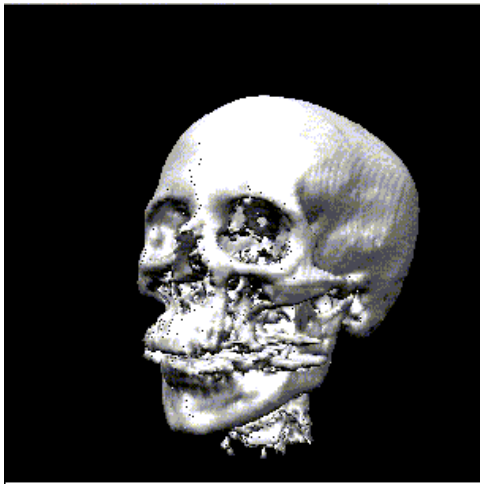


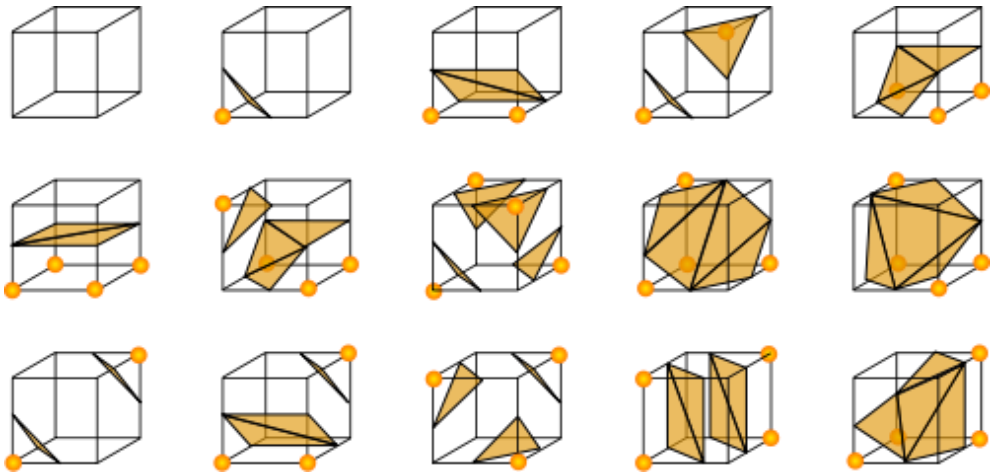
Térfogat vizualizáció

- Szintfelületek megjelenítése
  - Indirekt módszer
    - Marching cubes





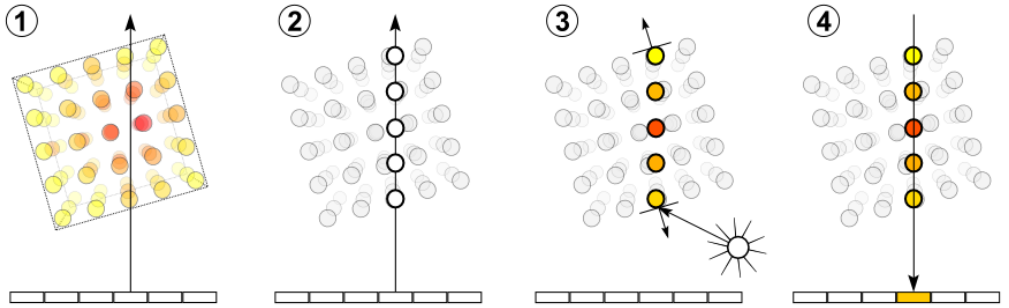


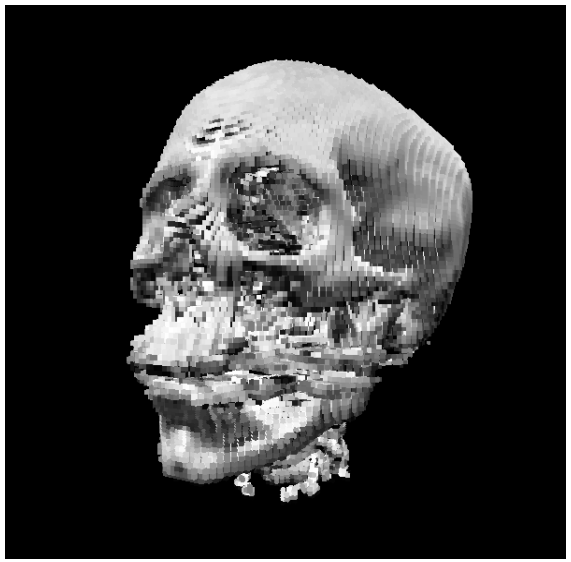


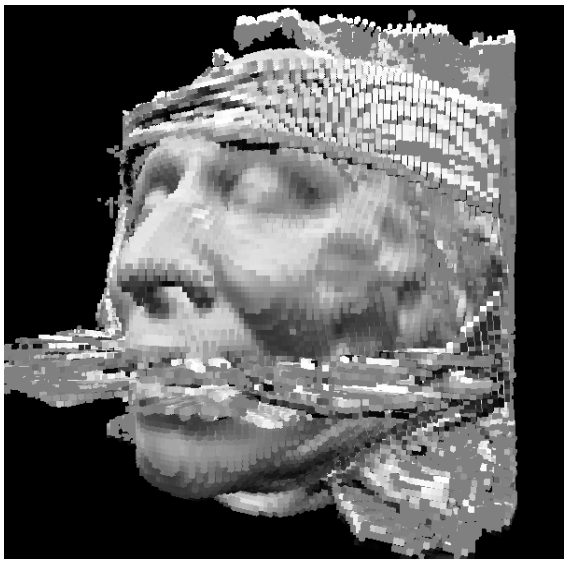
Térfogat vizualizáció

- Szintfelületek megjelenítése
  - Direkt módszer
    - Isosurface raycasting

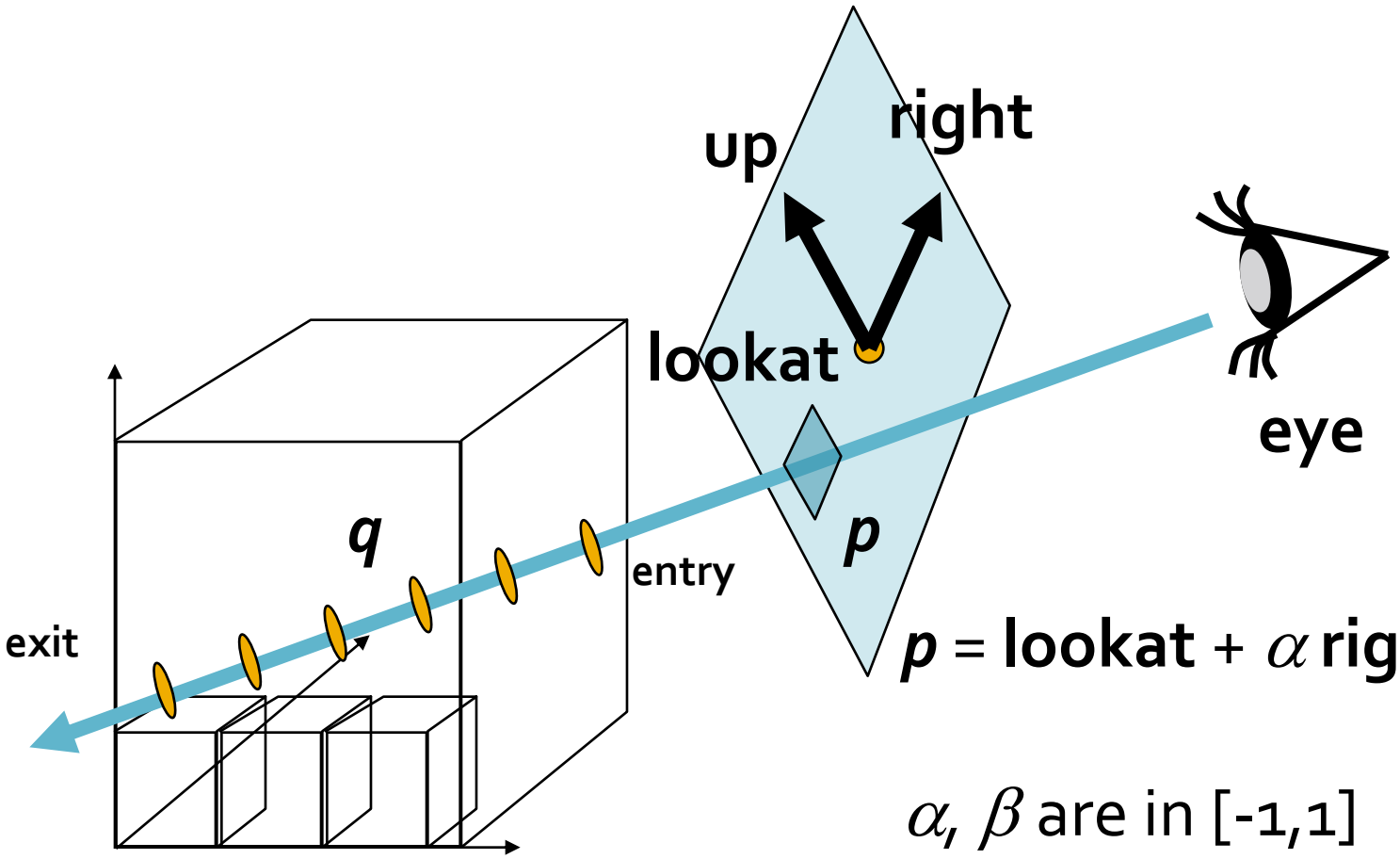
Raycasting







Isosurface raycasting



$p = \text{lookat} + \alpha\,\text{right} + \beta\,\text{up}$

α, β are in [-1,1]

Egység kocka 3D textúrával

Isosurface raycasting

```
__kernel
void isosurface(const int width, const int height, __global float4* visualizationBuffer,
                const int resolution, __global float* volumeData,
                const float isoValue, const float16 invViewMatrix){
  int2 id = (int2)(get_global_id(0), get_global_id(1));

  float2 uv = (float2)( (id.x / (float) width)*2.0f-1.0f, (id.y / (float) height)*2.0f-1.0f );

  float4 boxMin = (float4)(-1.0f, -1.0f, -1.0f,1.0f);
  float4 boxMax = (float4)(1.0f, 1.0f, 1.0f,1.0f);

  // calculate eye ray in world space
  struct ray eyeRay;

  eyeRay.origin = (float4)(invViewMatrix.sC, invViewMatrix.sD, invViewMatrix.sE,
                                                               invViewMatrix.sF);

  float4 temp = normalize(((float4)(uv.x, uv.y, -2.0f, 0.0f)));
  eyeRay.direction.x = dot(temp,
    ((float4)(invViewMatrix.s0,invViewMatrix.s1,invViewMatrix.s2,invViewMatrix.s3)));

  eyeRay.direction.y = dot(temp,
    ((float4)(invViewMatrix.s4,invViewMatrix.s5,invViewMatrix.s6,invViewMatrix.s7)));

  eyeRay.direction.z = dot(temp,
    ((float4)(invViewMatrix.s8,invViewMatrix.s9,invViewMatrix.sA,invViewMatrix.sB)));

  eyeRay.direction.w = 0.0f;

  // ...
```

Isosurface raycasting

```
  // ...
  float4 color = (float4)(0.0f);

  float tnear, tfar;
  int hit = intersectBox(eyeRay.origin, eyeRay.direction, boxMin, boxMax, &tnear, &tfar);
  if(hit){
    if(tnear < 0.0f) tnear = 0.0f;
    float maxStep = 256.0f;
    float step = (tfar - tnear) / maxStep;
    float t = tnear + 0.0001f;
    for(int i=0; i < maxStep; ++i){
      float4 pos = ((eyeRay.origin + t * eyeRay.direction) + 1.0f) / 2.0f;
      float density = getDensityFromVolume(pos, resolution, volumeData);

      // simple iso
      if( density > isoValue ){
          color = 0.5f + 0.5f * dot(normalize((float4)(0.3f, -2.0f, 0.0f, 0.0f)),
                                    getNormalFromVolume(pos, resolution, volumeData));
          break;
      }

      t += step;
      if(t>tfar) break;
    }
  }

  if(id.x < width && id.y < height){
    visualizationBuffer[id.x + id.y * width] = color;
  }
}
```

Isosurface raycasting

```
int intersectBox(float4 r_o, float4 r_d,
                 float4 boxmin, float4 boxmax,
                 float *tnear, float *tfar){

  // compute intersection of ray with all six bbox planes
  float4 invR = (float4)(1.0f,1.0f,1.0f,1.0f) / r_d;
  float4 tbot = invR * (boxmin - r_o);
  float4 ttop = invR * (boxmax - r_o);

  // re-order intersections to find smallest and largest on each axis
  float4 tmin = min(ttop, tbot);
  float4 tmax = max(ttop, tbot);

  // find the largest tmin and the smallest tmax
  float largest_tmin = max(max(tmin.x, tmin.y), max(tmin.x, tmin.z));
  float smallest_tmax = min(min(tmax.x, tmax.y), min(tmax.x, tmax.z));

  *tnear = largest_tmin;
  *tfar = smallest_tmax;

  return smallest_tmax > largest_tmin;
}
```

Isosurface raycasting

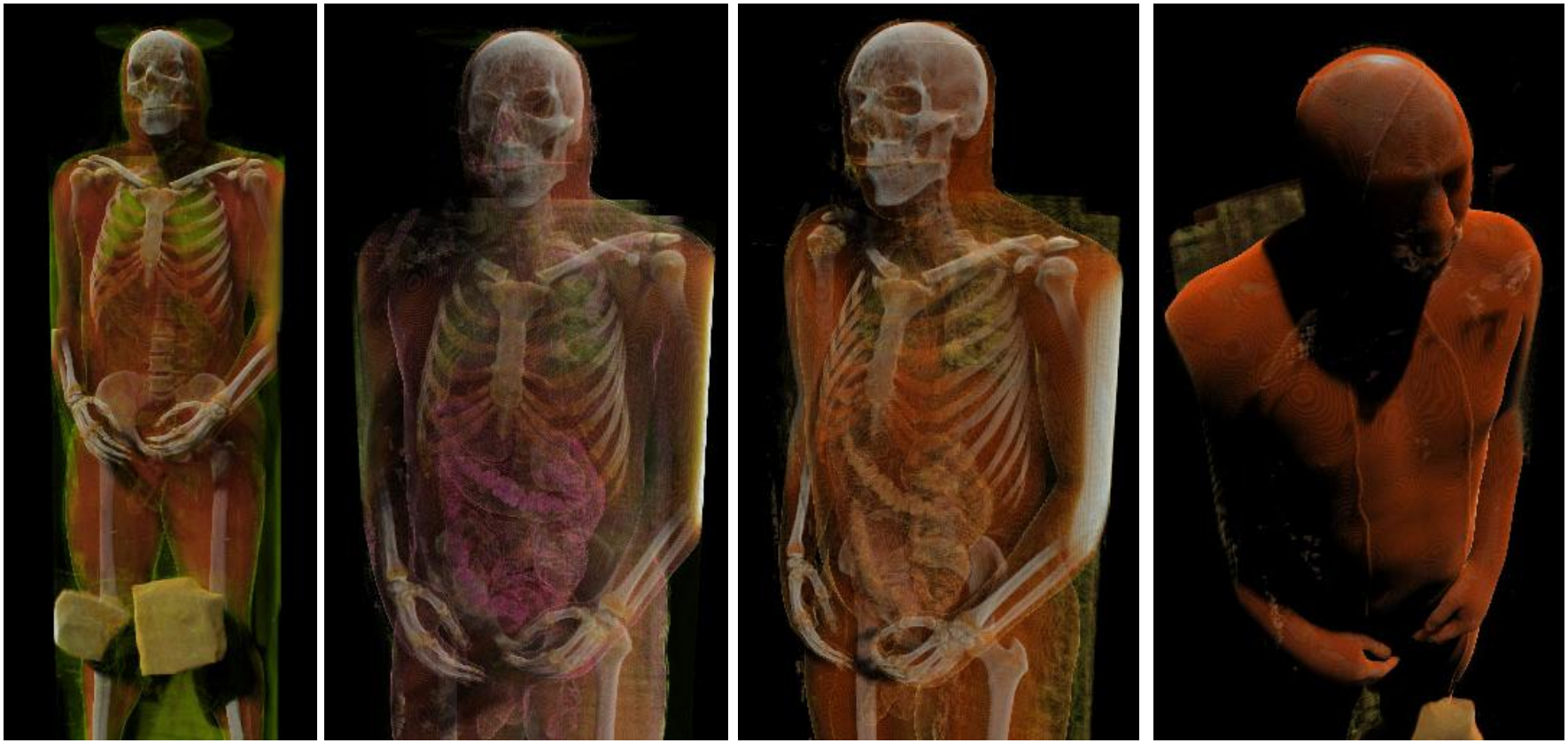
```
float4 getNormalFromVolume(const float4 p,
                           const int resolution, __global float* volumeData){
  float4 normal;

  normal.x = getDensityFromVolume((float4)(p.x + 2.0f/resolution, p.y, p.z,
                                           0.0f), resolution, volumeData) -
             getDensityFromVolume((float4)(p.x - 2.0f/resolution, p.y, p.z,
                                           0.0f), resolution, volumeData);
  normal.y = getDensityFromVolume((float4)(p.x, p.y + 2.0f/resolution, p.z,
                                           0.0f), resolution, volumeData) -
             getDensityFromVolume((float4)(p.x, p.y - 2.0f/resolution, p.z,
                                           0.0f), resolution, volumeData);
  normal.z = getDensityFromVolume((float4)(p.x, p.y, p.z + 2.0f/resolution,
                                           0.0f), resolution, volumeData) -
             getDensityFromVolume((float4)(p.x, p.y, p.z - 2.0f/resolution,
                                           0.0f), resolution, volumeData);
  normal.w = 0.0f;

  if(dot(normal, normal) < 0.001f){
    normal = (float4)(0.0f, 0.0f, 1.0f, 0.0f);
  }

  return normalize(normal);
}
```

Isosurface raycasting

```
float getDensityFromVolume(const float4 p,
                           const int resolution,
                           __global float* volumeData){
  int x = p.x * resolution;
  int y = p.y * resolution;
  int z = p.z * resolution;

  if(x > resolution -1 || x < 0) return(0.0f);
  if(y > resolution -1 || y < 0) return(0.0f);
  if(z > resolution -1 || z < 0) return(0.0f);

  return volumeData[x+y*resolution+z*resolution*resolution];
}
```

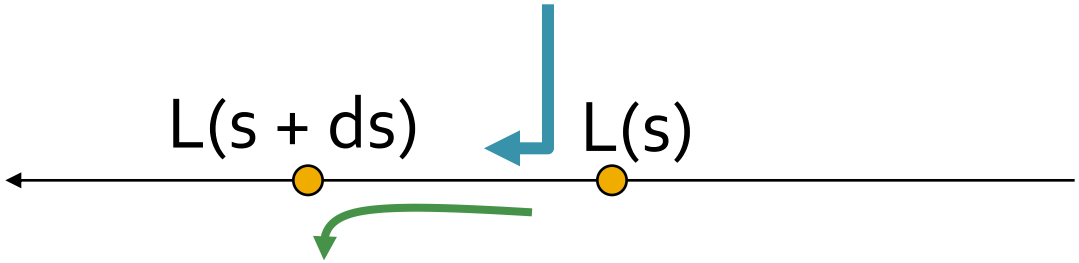
Térfogat vizualizáció

- Bele akarunk látni?

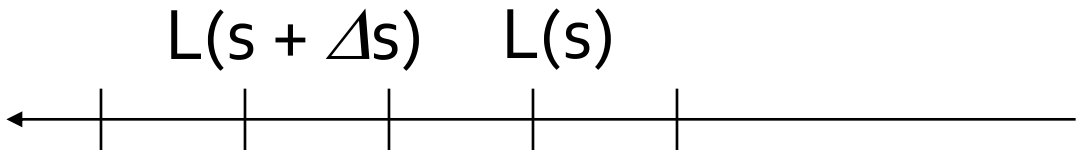


Térfogat vizualizáció

- Átlátszó anyagok



$$dL(s)/ds = -kt \cdot L(s) + ka \cdot L^e + \int f(\omega',\omega)\, L^i(\omega')\, d\omega'$$



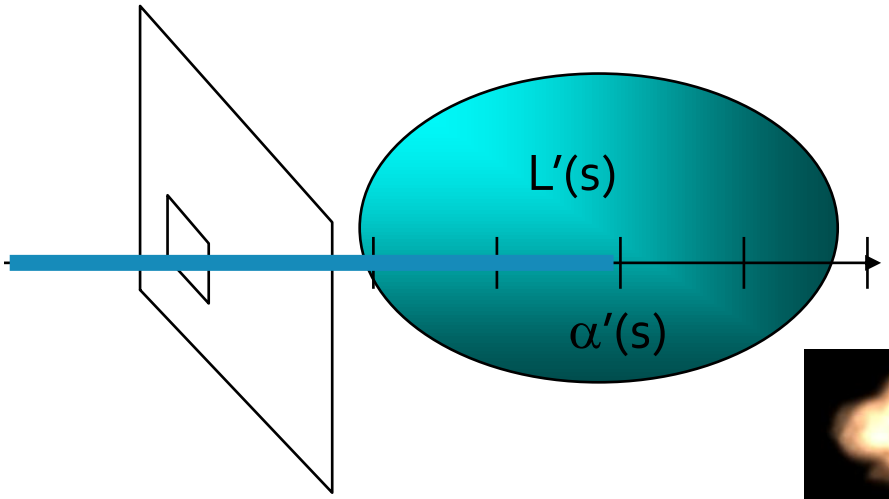
$$L(s + \Delta s) = L(s) - \underbrace{kt\,\Delta s}_{\alpha(s)} \cdot L(s) + \underbrace{Li(s)\,\Delta s}_{C(s)}$$

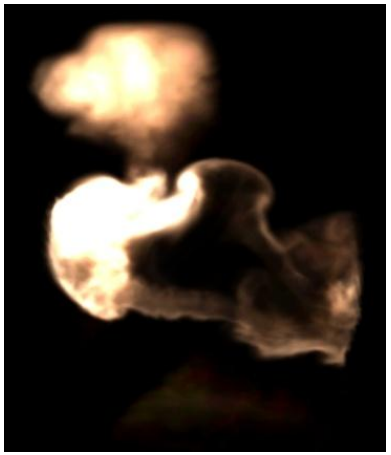
Térfogat vizualizáció

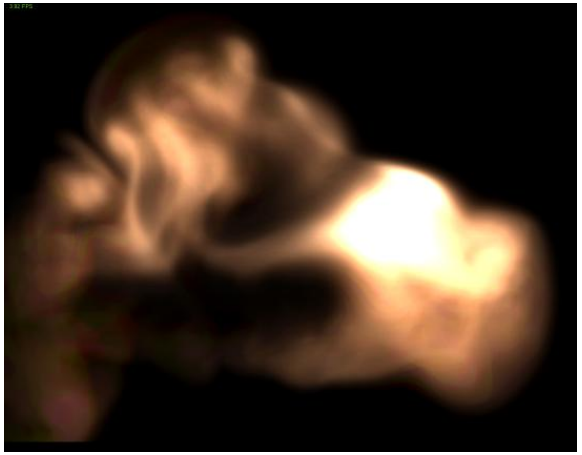
- Láthatóság sugárkövetés

$$L'(s-\Delta s) = L'(s) + (1-\alpha'(s)) \cdot C(s)$$

$$1-\alpha'(s-\Delta s) = (1-\alpha'(s)) \cdot (1-\alpha(s))$$



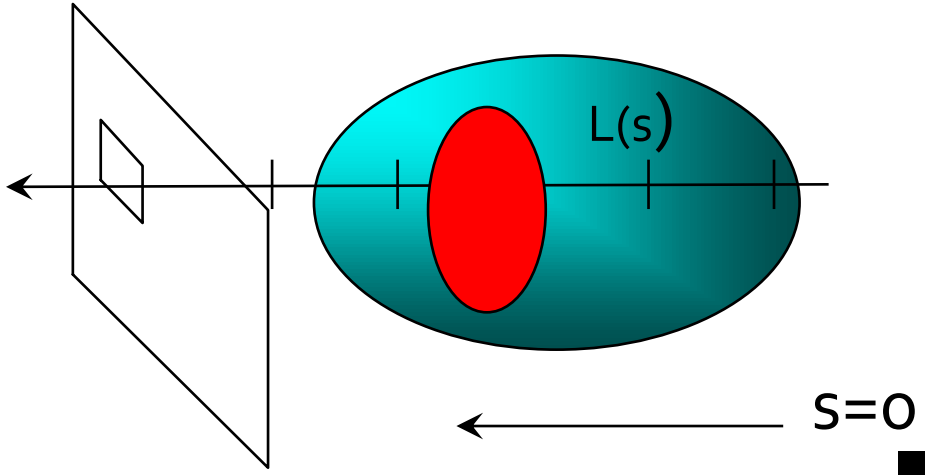


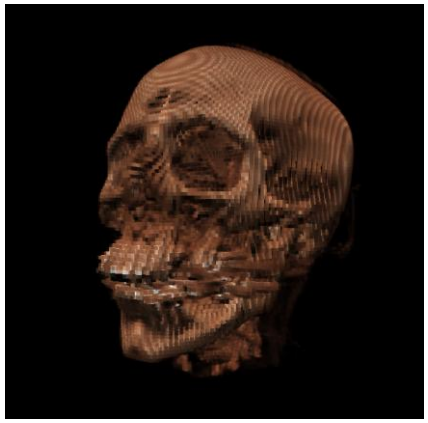


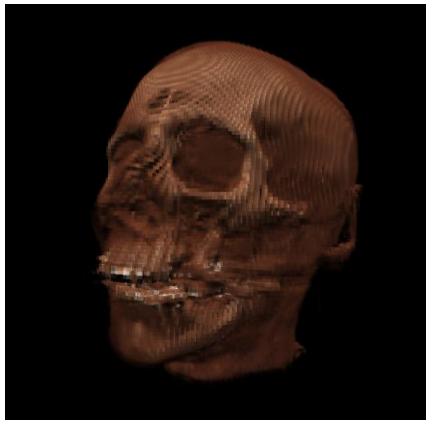
Térfogat vizualizáció

- Fénysugár követés

$$L(s+\Delta s) = (1-\alpha(s))\cdot L(s) + C(s)$$





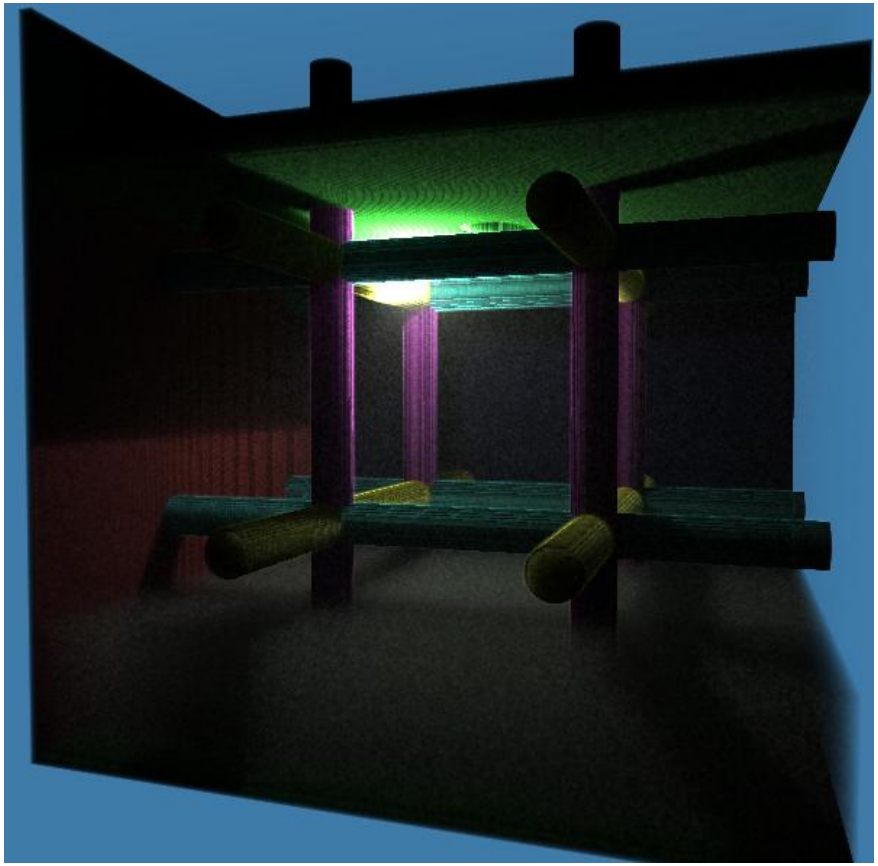


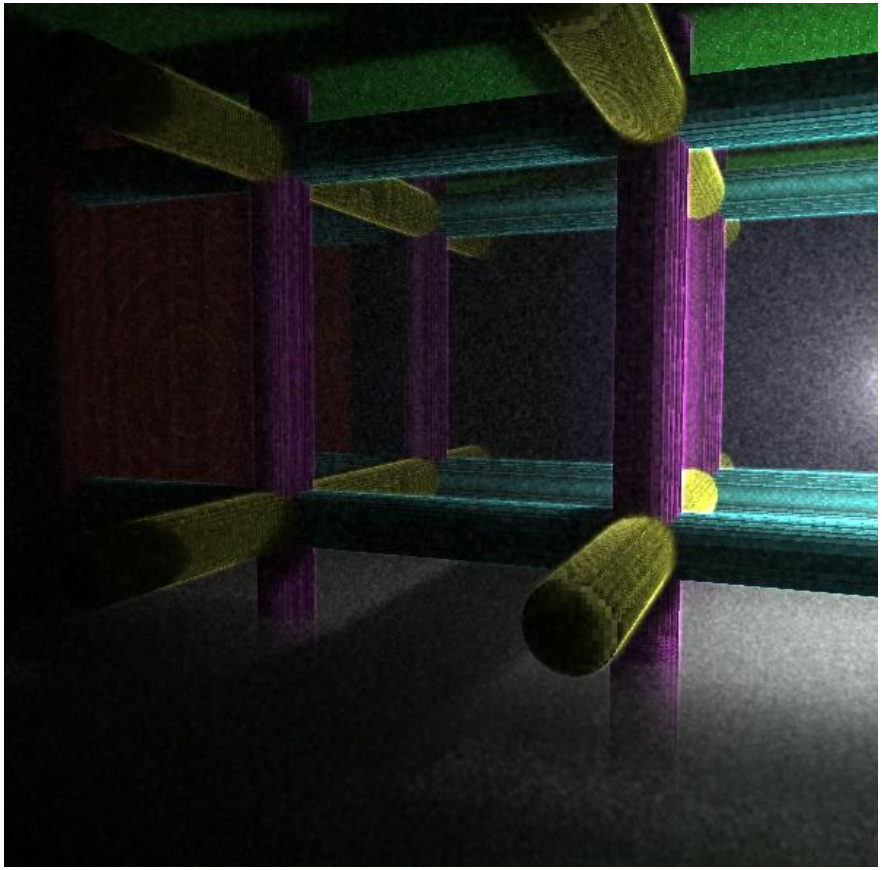
Fénysugár követés

```
__kernel
void alphaBlended(const int width, const int height, __global float4* visualizationBuffer,
           const int resolution, __global float* volumeData,
           const float alphaExponent, const float alphaCenter,
           const float16 invViewMatrix){
  int2 id = (int2)(get_global_id(0), get_global_id(1));

  float2 uv = (float2)( (id.x / (float) width)*2.0f-1.0f, (id.y / (float) height)*2.0f-1.0f );

  float4 boxMin = (float4)(-1.0f, -1.0f, -1.0f,1.0f);
  float4 boxMax = (float4)(1.0f, 1.0f, 1.0f,1.0f);

  // calculate eye ray in world space
  struct ray eyeRay;

  eyeRay.origin = (float4)(invViewMatrix.sC, invViewMatrix.sD, invViewMatrix.sE,
    invViewMatrix.sF);

  float4 temp = normalize(((float4)(uv.x, uv.y, -2.0f, 0.0f)));
  eyeRay.direction.x = dot(temp,
    ((float4)(invViewMatrix.s0,invViewMatrix.s1,invViewMatrix.s2,invViewMatrix.s3)));
  eyeRay.direction.y = dot(temp,
    ((float4)(invViewMatrix.s4,invViewMatrix.s5,invViewMatrix.s6,invViewMatrix.s7)));
  eyeRay.direction.z = dot(temp,
    ((float4)(invViewMatrix.s8,invViewMatrix.s9,invViewMatrix.sA,invViewMatrix.sB)));
  eyeRay.direction.w = 0.0f;

  // ...
```

Fénysugár követés

```
  // ...
  float4 sum = (float4)(0.0f);

  float tnear, tfar;
  int hit = intersectBox(eyeRay.origin, eyeRay.direction, boxMin, boxMax, &tnear, &tfar);
  if(hit){
    if(tnear < 0.0f) tnear = 0.0f;
    float maxStep = 256.0f;
    float step = (tfar - tnear) / maxStep;
    float t = tfar - 0.0001f;
    for(int i=0; i < maxStep; ++i){
      float4 pos = ((eyeRay.origin + t * eyeRay.direction) + 1.0f) / 2.0f;
      float density = getDensityFromVolume(pos, resolution, volumeData);
     float alpha = clamp(alphaExponent * (density - alphaCenter) + 0.5f, 0.0f, 1.0f);
      float4 color = (float4)(0.5f + 0.5f * dot(normalize((float4)(0.3f, -2.0f, 0.0f, 0.0f)),
    getNormalFromVolume(pos, resolution, volumeData)));

      color *= (float4)(density, density * density, density * density * density, 1.0f) + 0.1f;

      sum = (1.0f - alpha) * sum + alpha * color;

      t -= step;
      if(t<tnear) break;
    }
  }

  if(id.x < width && id.y < height){
    visualizationBuffer[id.x + id.y * width] = (float4)(sum);
  }
}
```





OpenCL textúra támogatás

- Image object
  - 1D / 2D / 3D textúrák
  - 4 elemű vektorok
  - Lineáris interpoláció
  - Címzési módok

- Textúra támogatás

```
cl_int clGetDeviceInfo(...)
```

  - CL_DEVICE_IMAGE_SUPPORT

OpenCL textúra támogatás

- Image object létrehozása

```
cl_mem clCreateImage2D(...)
```

```
cl_mem clCreateImage3D(...)
```

  - image formátum
  - pitch: egy sor tárolásához szükséges byte méret
    - `0` ha a host_pointer NULL
    - `>= szélesség * elem` méret byteban ha a host_pointer adott
    - csak a betöltéshez szükséges, a belső formátum más lehet!

OpenCL textúra támogatás

- Textúra formátum leírás

```
typedef struct _cl_image_format {
        cl_channel_order image_channel_order;
        cl_channel_type  image_channel_data_type;
} cl_image_format;
```

- Csatorna sorrend
  - CL_R, CL_A
  - CL_INTENSITY
  - CL_LUMINANCE
  - CL_RG, CL_RA
  - CL_RGB
  - CL_RGBA, CL_ARGB, CL_BGRA

OpenCL textúra támogatás

- Textúra adat formátum
  - CL_SNORM_INT8 / 16
  - CL_UNORM_INT8 / 16
  - CL_UNORM_SHORT_565 / 555
  - CL_UNORM_INT_101010
  - CL_SIGNED_INT8 / 16 / 32
  - CL_UNSIGNED_INT8 / 16 / 32
  - CL_HALF_FLOAT
  - CL_FLOAT

OpenCL textúra támogatás

- Támogatott formátumok lekérdezése

```
cl_int clGetSupportedImageFormats(cl_context context,
                                  cl_mem_flags flags,
                                  cl_mem_object_type image_type,
                                  cl_uint num_entries,
                                  cl_image_format* image_formats,
                                  cl_uint* num_image_formats)
```

  - image_type: 2D/3D image object
  - image_formats: a támogatott formátumok listája

OpenCL textúra támogatás

- Olvasás Image objektumból

```
cl_int clEnqueueReadImage(...)
```

- Irás Image objektumba

```
cl_int clEnqueueWriteImage(...)
```

  - origin[3]: kezdő koordináták
  - region[3]: másolandó méret
  - row_pitch / slice_pitch: reprezentációs méret

OpenCL textúra támogatás

- Másolás Image objektumok között

```
cl_int clEnqueueCopyImage(...)
```

  - src_origin[3]: forrás koordináták
  - dst_origin[3]: cél koordináták
  - region[3]: másolandó terület mérete
- Másolás Image és Buffer objektum között

```
cl_int clEnqueueCopyImageToBuffer(...)
```

```
cl_int clEnqueueCopyBufferToImage(...)
```

OpenCL textúra támogatás

- Sampler objektum

```
cl_sampler clCreateSampler(cl_context context,
                           cl_bool normalized_coords,
                           cl_addressing_mode addressing_mode,
                           cl_filter_mode filter_mode,
                           cl_int* errcode_ret)
```

  - normalized_coords: címzési mód
  - addressing_mode: túlcímzés kezelése
    - REPEAT, CLAMP_TO_EDGE, CLAMP, NONE
  - filter_mode: textúra szűrés
    - NEAREST, LINEAR

OpenCL textúra támogatás

- Referencia számláló növelése / csökkentése

```
cl_int clRetainSampler(cl_sampler sampler)
```

```
cl_int clReleaseSampler(cl_sampler sampler)
```

- Sampler információk lekérdezése

```
cl_int clGetSamplerInfo(cl_sampler sampler,
                        cl_sampler_info param_name,
                        size_t param_value_size,
                        void* param_value,
                        size_t *param_value_size_ret)
```

  - CPU oldalon létrehozott sampler tulajdonságai

OpenCL textúra támogatás

- Image object
  - Csak olvasható: __read_only
  - Csak írható: __write_only
  - Olvasás és írás nem támogatott egyszerre!

- Sampler object
  - Host oldalon létrehozott sampler
  - Globális konstans sampler

```
const sampler_t samplerName = NORMALIZED_COORDS |
                              ADDRESS_MODE      |
                              FILTER_MODE;
```

OpenCL textúra támogatás

- Olvasás az Image objektumból

```
<o_típus> read_image{f,i,ui}(image2d_t image,
                             sampler_t sampler,
                             <c_típus> coord)
```

  - 4 elemű vektor az Image típusának megfelelően
  - {f, i, ui}: float, int, unsigned int
  - coord: a sampler címzésének megfelelő koordináták

OpenCL textúra támogatás

- Írás Image objektumba

```
void write_image{f,i,ui}(image2d_t image,
                         <coord_típus> coord,
                         <color_típus> color)
```

  - {f, i, ui}: float, int, unsigned int
  - coord: cél koordináták
  - color: a beírandó színvektor

OpenCL textúra támogatás

- Image objektum információk
  - Image dimenziók

```
int get_image_width(image{2,3}d_t image)
```

```
int get_image_height(image{2,3}d_t image)
```

```
int get_image_depth(image3d_t image)
```

```
int{2,4} get_image_dim(image{2,3}d_t image)
```

  - Image formátum

```
int get_image_channel_data_type(image{2,3}d_t image)
```

```
int get_image_channel_order(image{2,3}d_t image)
```