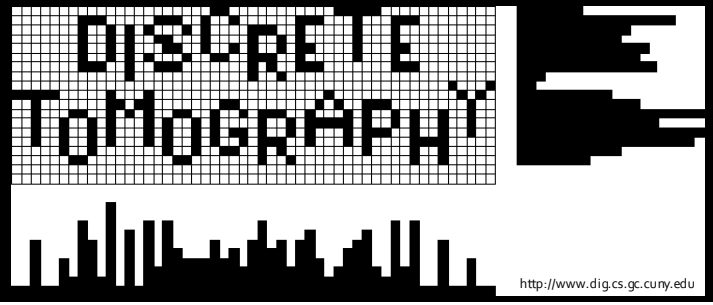
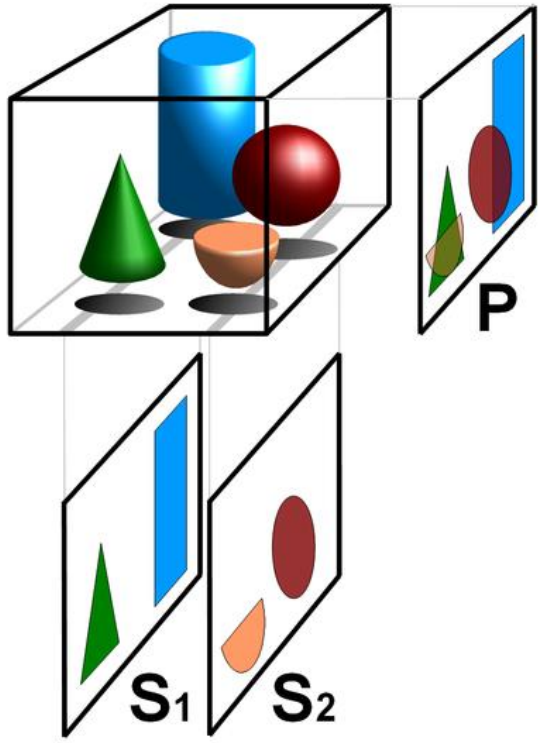
Pozitron Emissziós Tomográfia

PET

Tomográfia



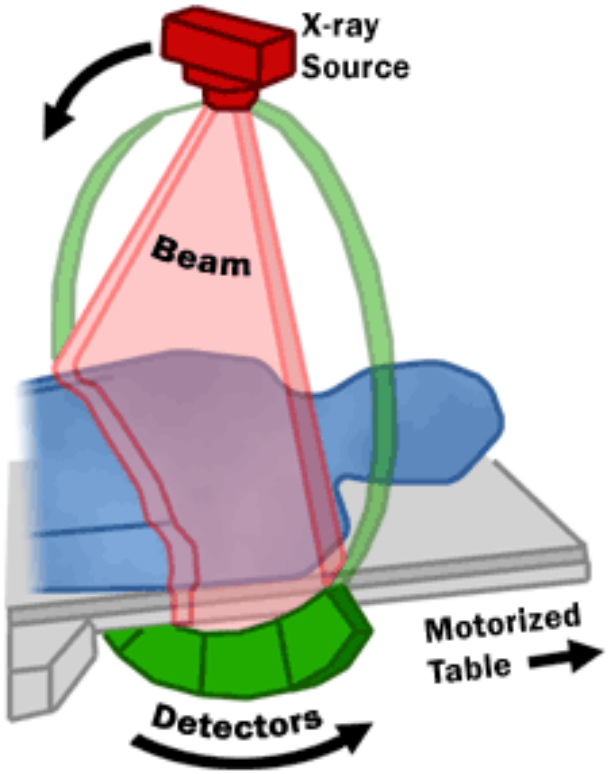
- Vetületi képekből állítsuk elő a (3D) objektumot vagy függvényt
- A vetületi képek előállítására különféle fizikai jelenségeket használunk, pl:



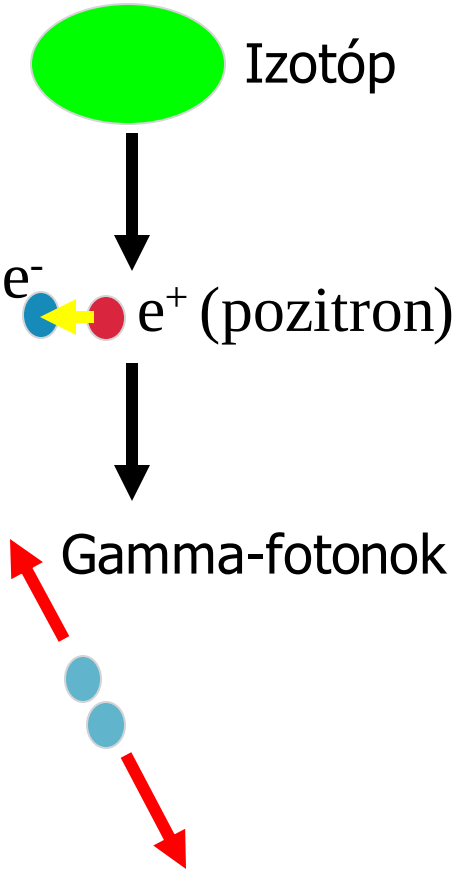
Fizikai jelenség	„Tomográfiás módszer”
Látható fény	Térlátás
Röntgen-sugár	CT
Gamma-sugár	SPECT
Rádió-hullám	MRI
Pozitron-elektron ütközés	PET

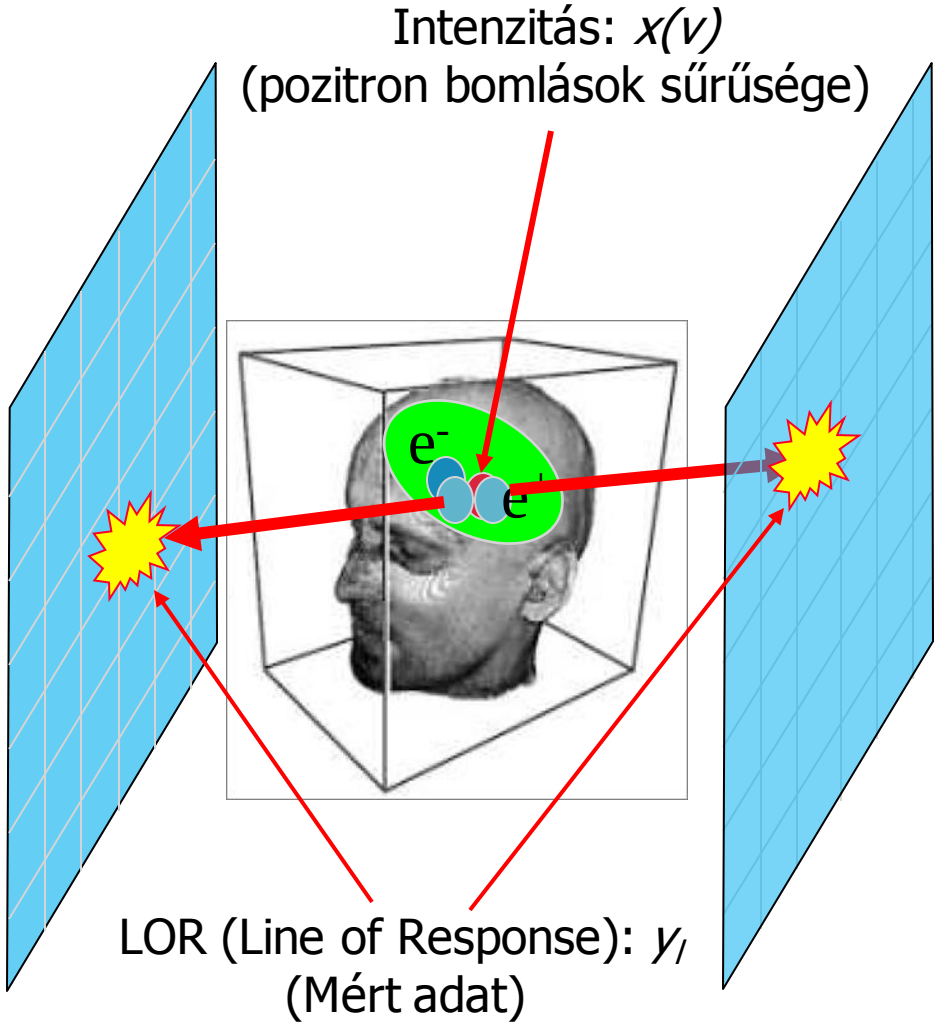
Komputertomográfia (CT)

- Mérési adat: a röntgen-sugár csillapodása egy adott út mentén
  - Az anyagsűrűség függvénye
- Kimenet: anyagsűrűség a tér pontjaiban



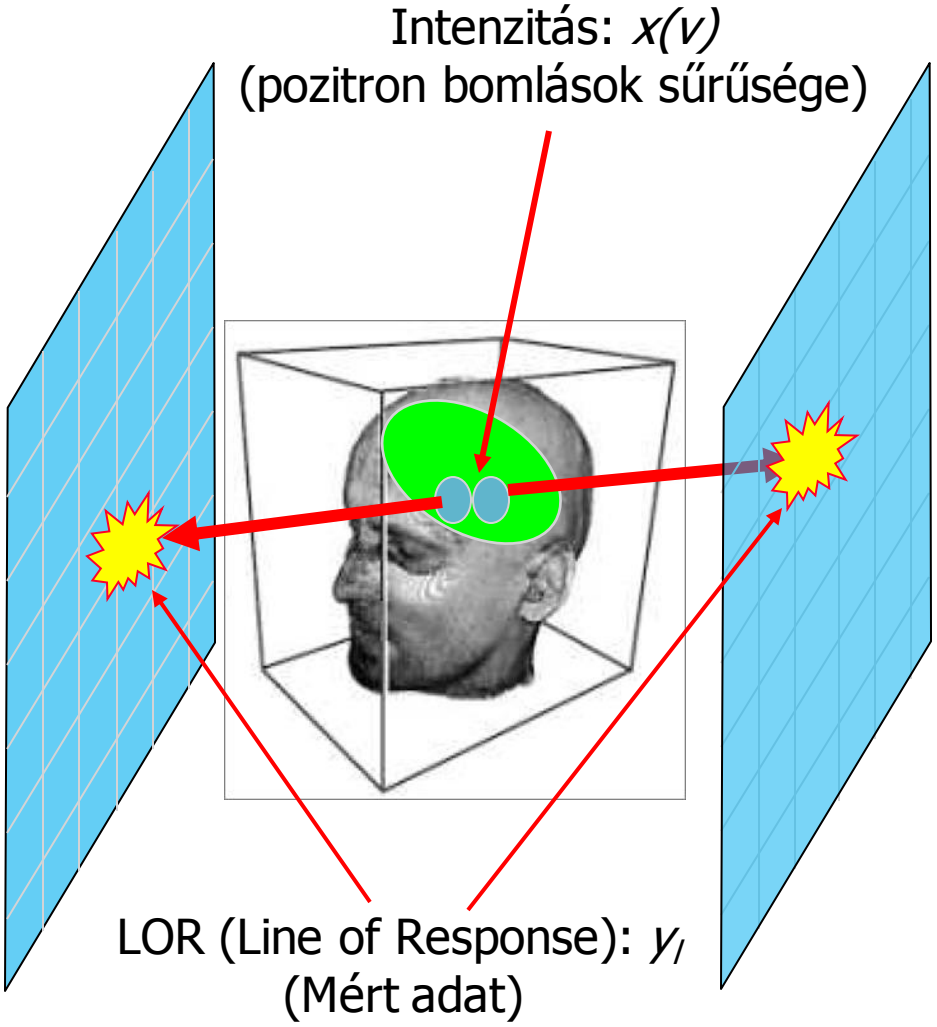
Pozitron Emissziós Tomográfia





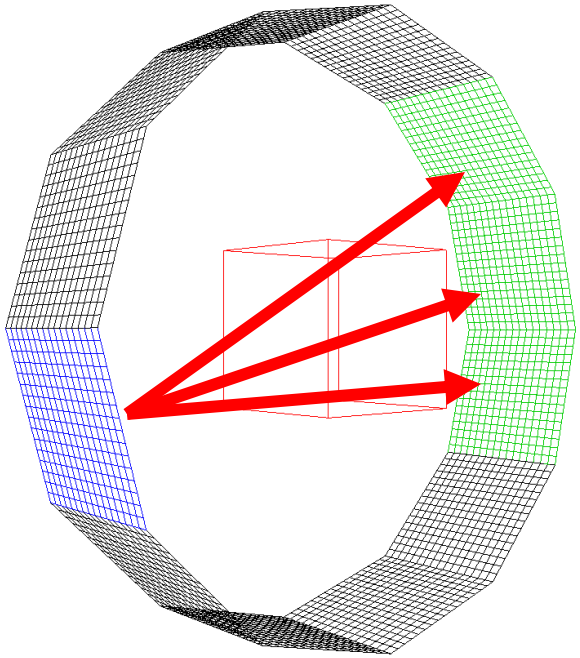
Pozitron Emissziós Tomográfia

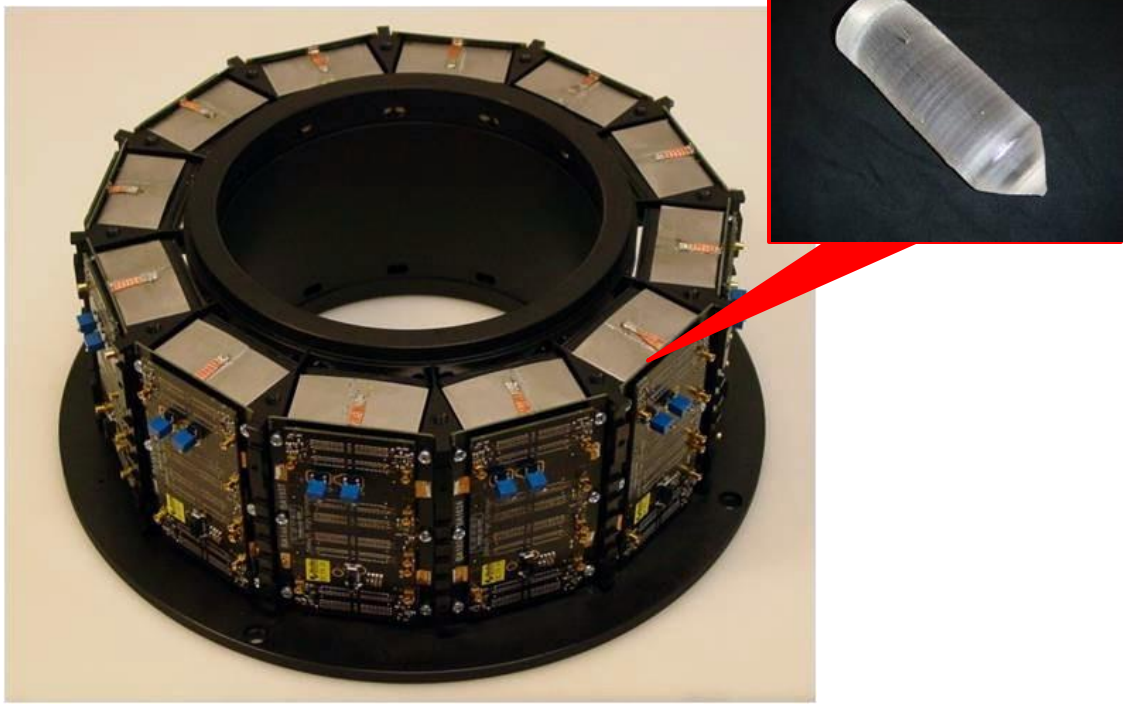
- A készülék az egyidejű fotonbecsapódásokat regisztrálja (számolja)



Pozitron Emissziós Tomográfia

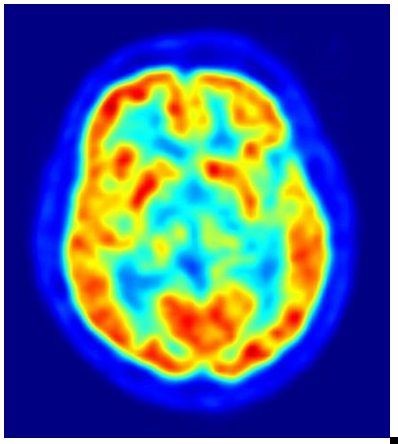
- A detektorlapokat (panel) gyűrű alakban helyezik el
- Minden panelt 2D rácson elhelyezett kristályok alkotnak
- A mérés 5D adat (párindex és modulonként 2-2 koordináta)
- Koincidencia

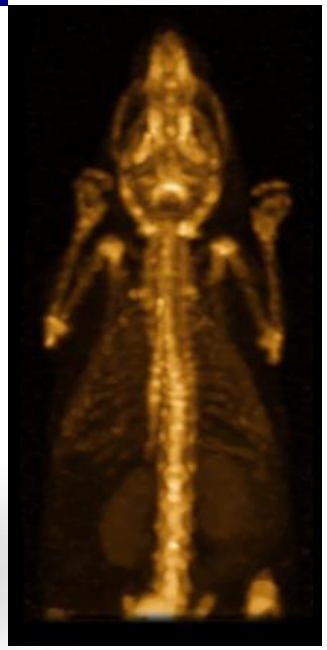


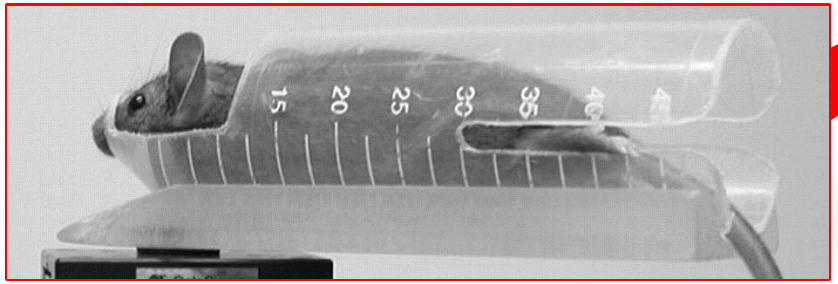


Mire jó?

- A vizsgálat során radioaktív kontrasztanyagot juttatnak a szervezetbe, amelyet a sejtek az anyagcseréjük során felvesznek (pl. valamilyen cukoroldat)
- A vizsgálat célja a kontrasztanyag térbeli sűrűségeloszlásának meghatározása
- Így nyomon követhetővé válik az anyagcsere gyorsasága
  - Diagnosztika: a daganatos sejtek anyagcseréje jóval intenzívebb
  - Szervműködés: agyműködés feladatok elvégzése közben
  - Kisállat PET: gyógyszerkísérletek







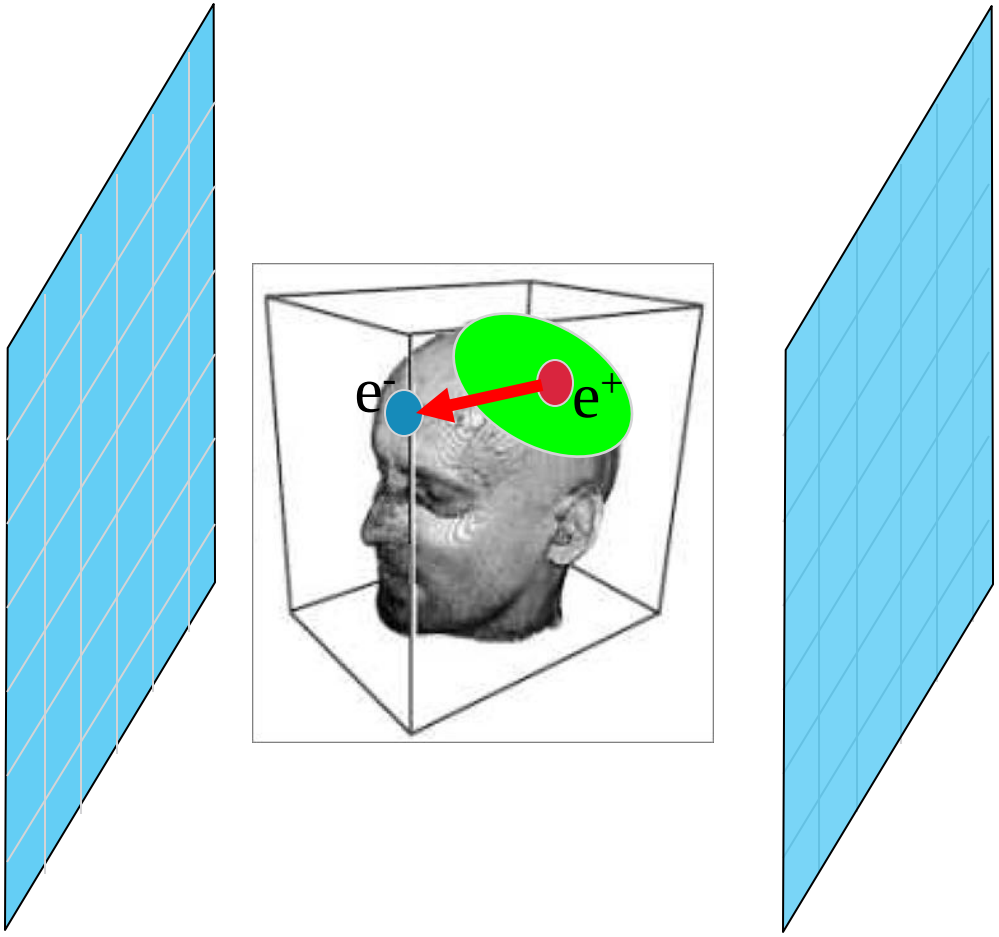
Fizikai hatások

- A pozitron születésétől a γ-foton detektálásáig több fizikai jelenség megy végbe, melyeket különböző mértékben modellezünk, közelítünk
- Pozitron vándorlás
- Direkt hatás
- γ-foton szóródás, elnyelődés
- …

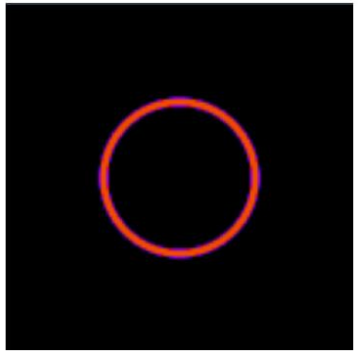
- A rekonstrukciós képminőségre gyakorolt hatásuk mértéke elsősorban a tomográf méretétől függ (kisállat, humán stb.)

Pozitron vándorlás

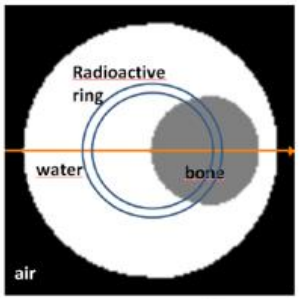
- A pozitron létrejöttét követően néhány cm-t is vándorolhat, mielőtt az elektronba ütközik
- A vándorlás várható úthossza az alkalmazott kontrasztanyagtól és az elektronsűrűségtől függ (páciensenként más, a CT alapján számítható)
- Általában a bomlásfüggvény helyfüggő elkenésével modellezik



Ha figyelmen kívül hagyjuk…

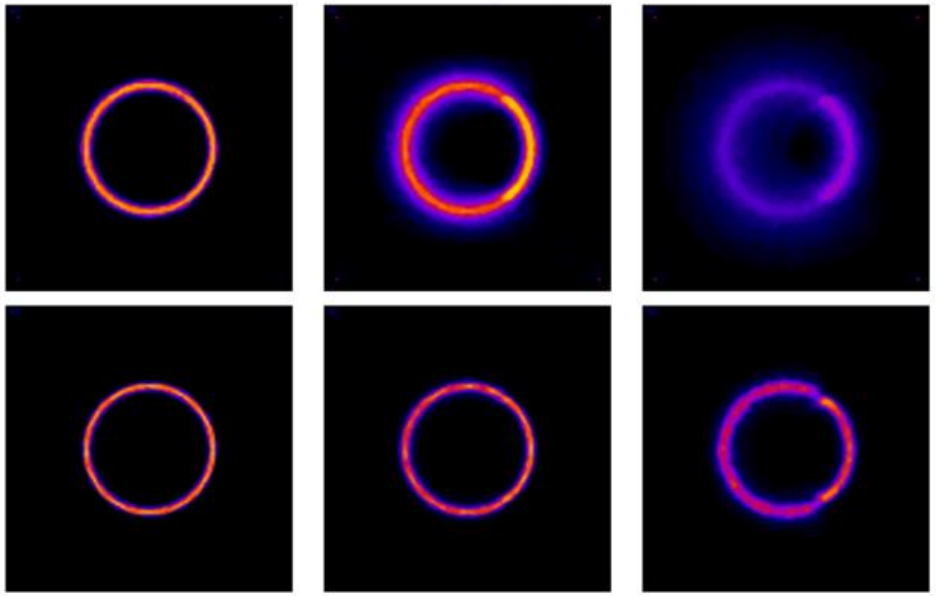


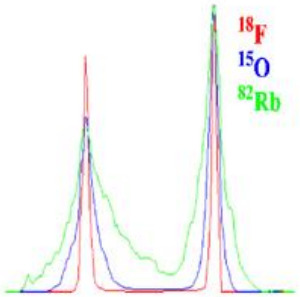
Fantom

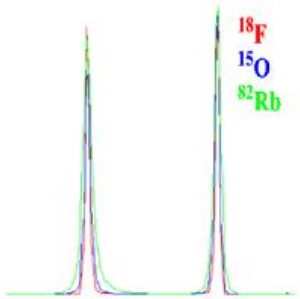


Pozitronvándorlás nélkül

Pozitronvándorlás szimulációval







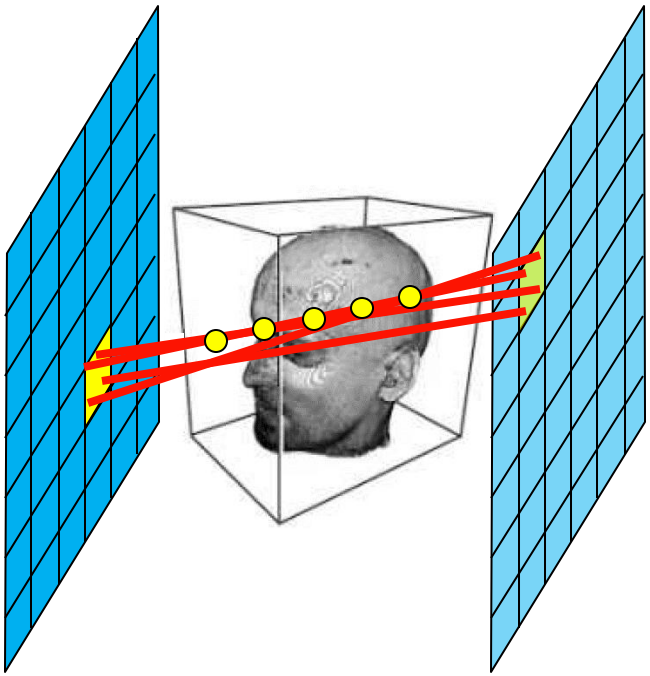
^{18}F ^{15}O ^{82}Rb line profiles

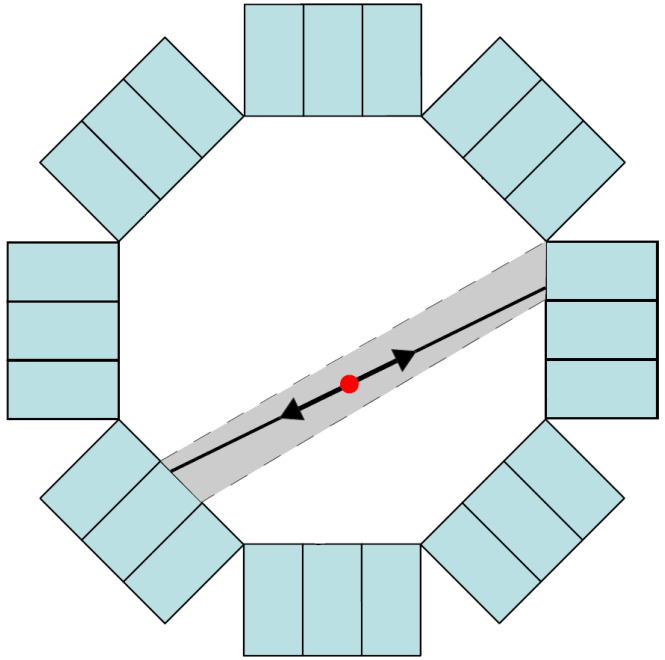
Max úthossz (víz, mm)	2.3	7.9	16.7

Vonalprofilok

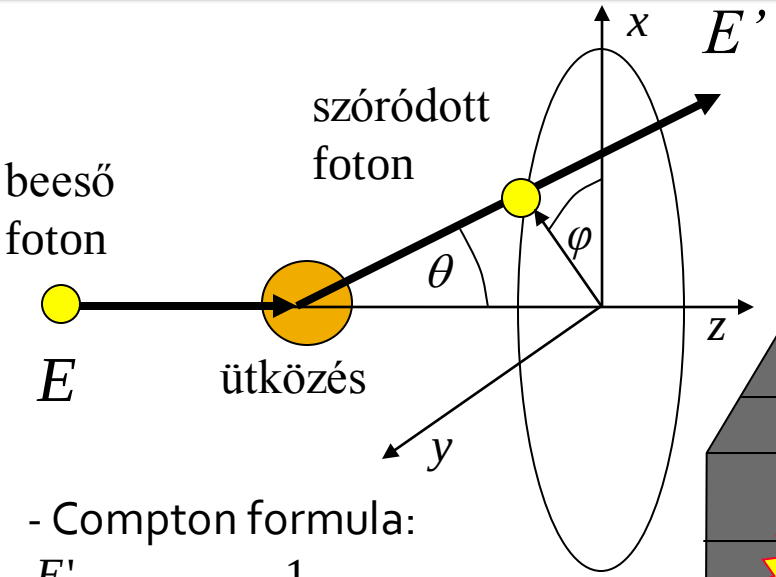
Direkt hatás

- A fotonok nagy része nem szóródik
  - A fotonpár útja egy egyenes
  - Egy adott detektorpárra hatással lévő fotonok keletkezési helye jól körülhatárolt: a detektált (nem szóródott) fotonpárok a detektorok felületét összekötő szakaszok mentén keletkeztek





Szóródás, elnyelődés



- Compton formula:

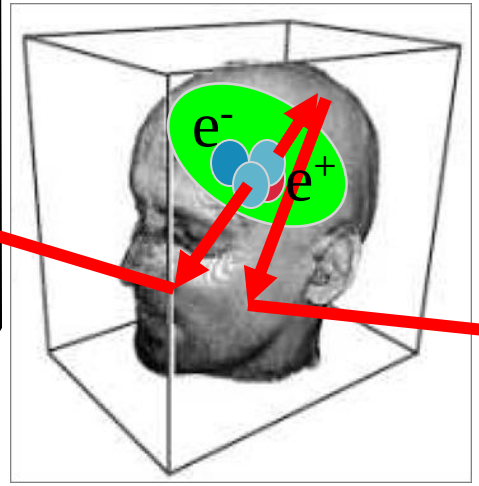
$$\frac{E'}{E} = \frac{1}{1 + \frac{E}{m_e c^2}(1 - \cos\theta)}$$

- Klein-Nishina:

$$P(\theta) = C\left(\frac{E'}{E} + \left(\frac{E'}{E}\right)^3 - \left(\frac{E'}{E}\right)^2 \sin^2\theta\right)$$

Elektronsűrűség
(anyagfüggő – CT mérésből)

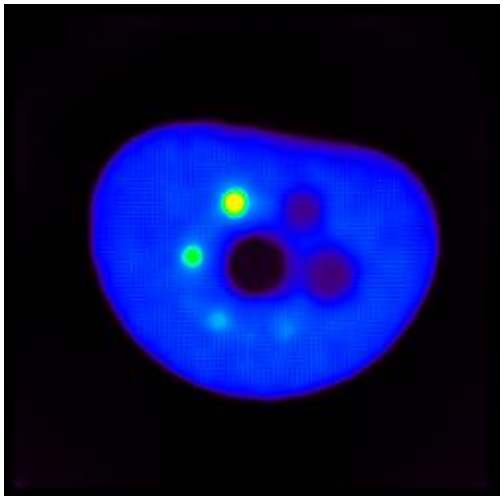
testen belüli szóródás,
elnyelődés



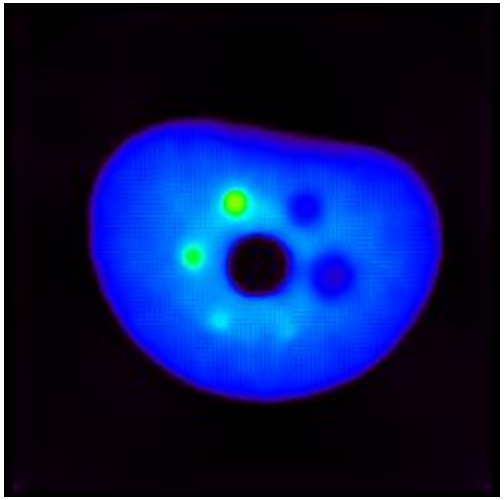
detektoron belüli szóródás,
elnyelődés

Ha figyelmen kívül hagyjuk…

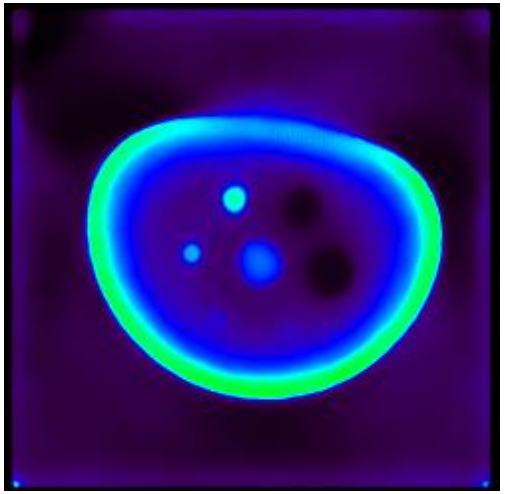
Szóródás+elnyelődés



Elnyelődés

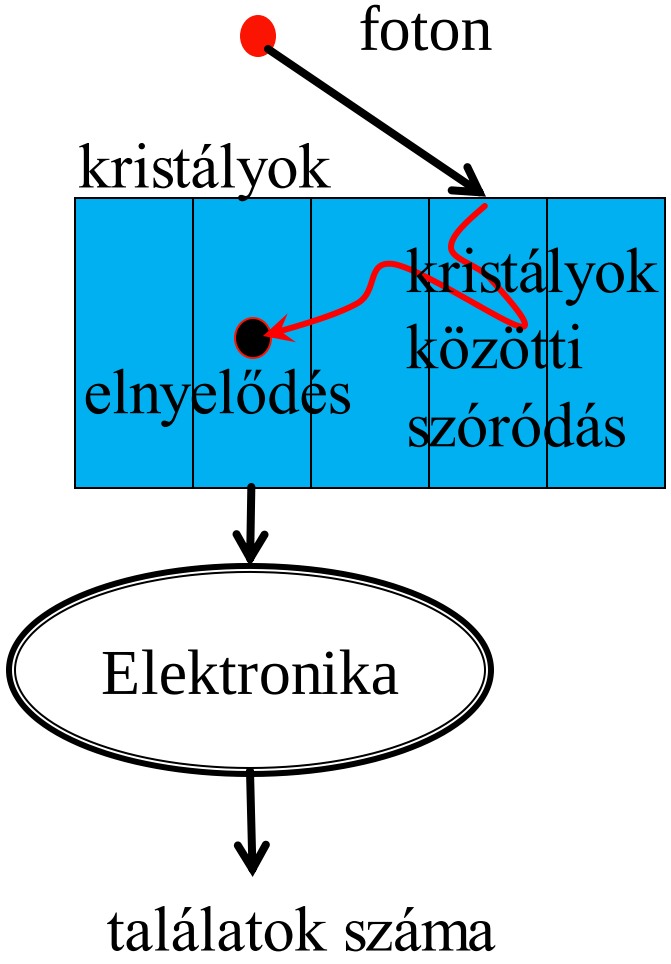


Csak geometria



Detektormodell

- Átszóródás: a foton a kristályok közt átszóródhat, így nem ott detektálódik, mint ahol beérkezett
- Kristályok és az elektronika tökéletlensége: egy fotonbecsapódást várható értékben nem pontosan egy találatként regisztrál (nem érzékeli, vagy több találatként regisztrálja)



Ha figyelmen kívül hagyjuk…

Rekonstrukciós módszer

- Az ismeretlen függvény modellje
  - Végeselem reprezentáció, interpolációs módok
- Rekonstrukciós algoritmus
  - ML-EM: iteratív, figyelembe veszi a mért adat Poisson eloszlását, tetszőleges fizikai jelenség modellje beépíthető
- Rendszermátrix
  - Direkt vs. Adjungált módszerek
  - Fizikai modellek

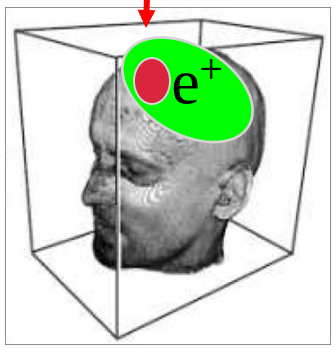
Végeselem módszer

- Az ismeretlen függvényt együtthatók és rögzített bázisfüggvények segítségével írjuk le:

$$x(\vec{v}) = \sum_{j=1}^{M} x_j b_j(\vec{v})$$

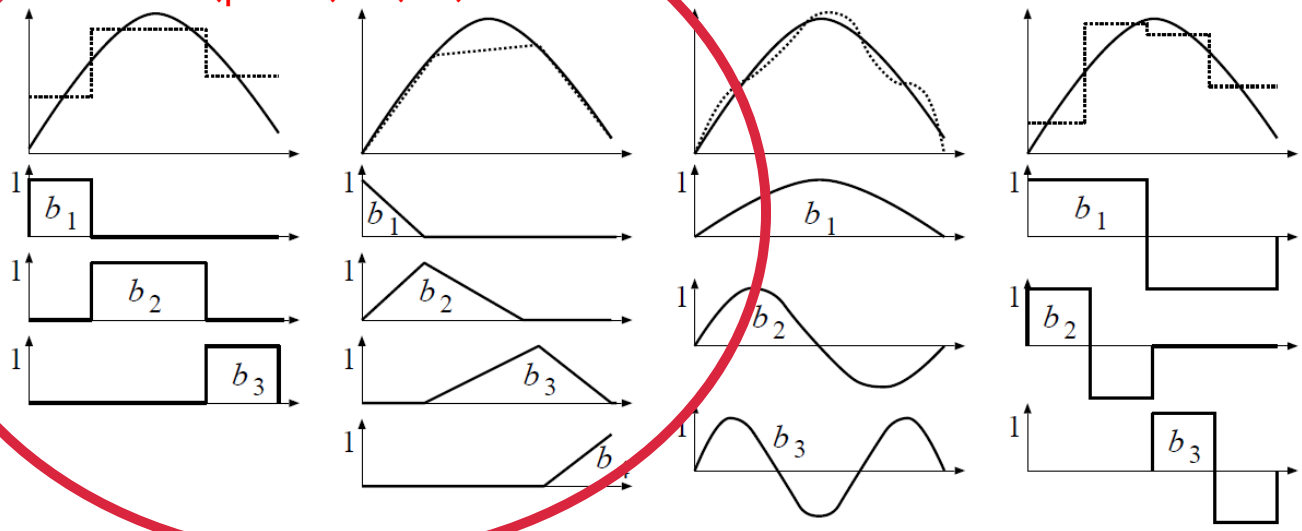
Voxelek, 3D rács

Intenzitás: *x(v)*
(pozitron bomlások sűrűsége)

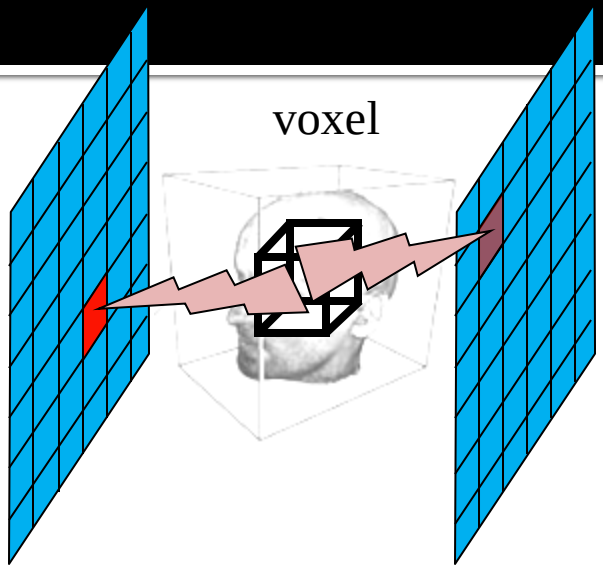


GPU

Nearest (point) (Tri)Linear



Modellezés: rendszermátrix



mért LOR válasz

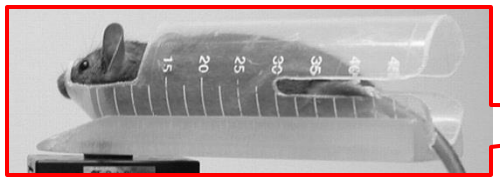
voxel intenzitás

$$y = Ax$$

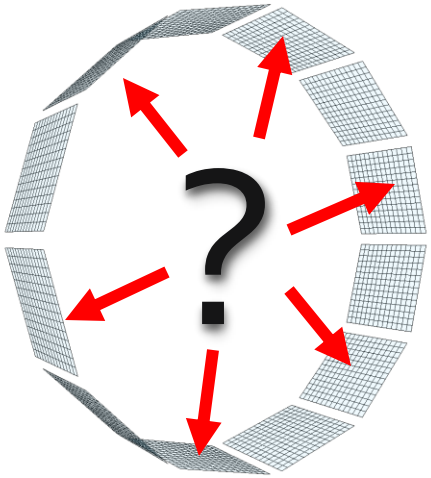
A_{ij}: annak valószínűsége, hogy az i-edik voxelben keletkezett pozitron a j-edik LoR-ban eredményez fotonpár becsapódást

- Problémák:
  - A gyakorlatban óriási rendszermátrix: $10^8 \times 10^7 = 10^{15}$ elem
  - Néhány TeraByte!
  - Minden mátrixelem végtelendimenziós integrál (szóródás)
  - A mátrixelemek függnek a mért objektumtól (szóródás, pozitron-vándorlás), ezért offline nem számolható pontosan
  - Ha eltekintünk a szóródástól, a mátrix ritka: gyorsítások
- A mátrixelemeket on-the-fly számoljuk!

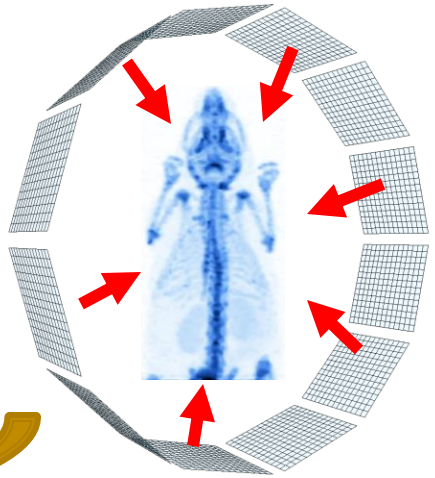
Az EM-algoritmus (Expectation Maximization)



Mért foton találatok



Becsült foton találatok



I. „Előrevetítés”:
A kontrasztanyag eloszlásának egy (tetszőleges) becsléséből kiindulva, egy **pontos fizikai modellt** alkalmazva kiszámoljuk, hogy a becslést feltételezve **mit kellene mérnünk**

III. Iteratív rekonstrukció:
Az egészet megismételjük, immáron az új, pontosabb becslésből kiindulva

II. „Visszavetítés”:
A becsült és a mért beütések aránya alapján **korrigáljuk** a kontrasztanyag eloszlására adott becslést

Az EM-algoritmus (Expectation Maximization)

- Maximum-likelihood becslés: találjuk meg azt az *x(v)* függvényt, amelyet feltételezve az adott mérés valószínűsége maximális
- Likelihood: $L(x(\vec{v})|y_1, \dots y_N) = P(y_1, \dots y_N|x(\vec{v})) = \prod_{i=1}^{N} P(y_i|x(\vec{v}))$
- Az EM algoritmus a fenti Likelihoodot maximalizálja feltételezve, hogy a LOR-beütések Poisson eloszlásúak

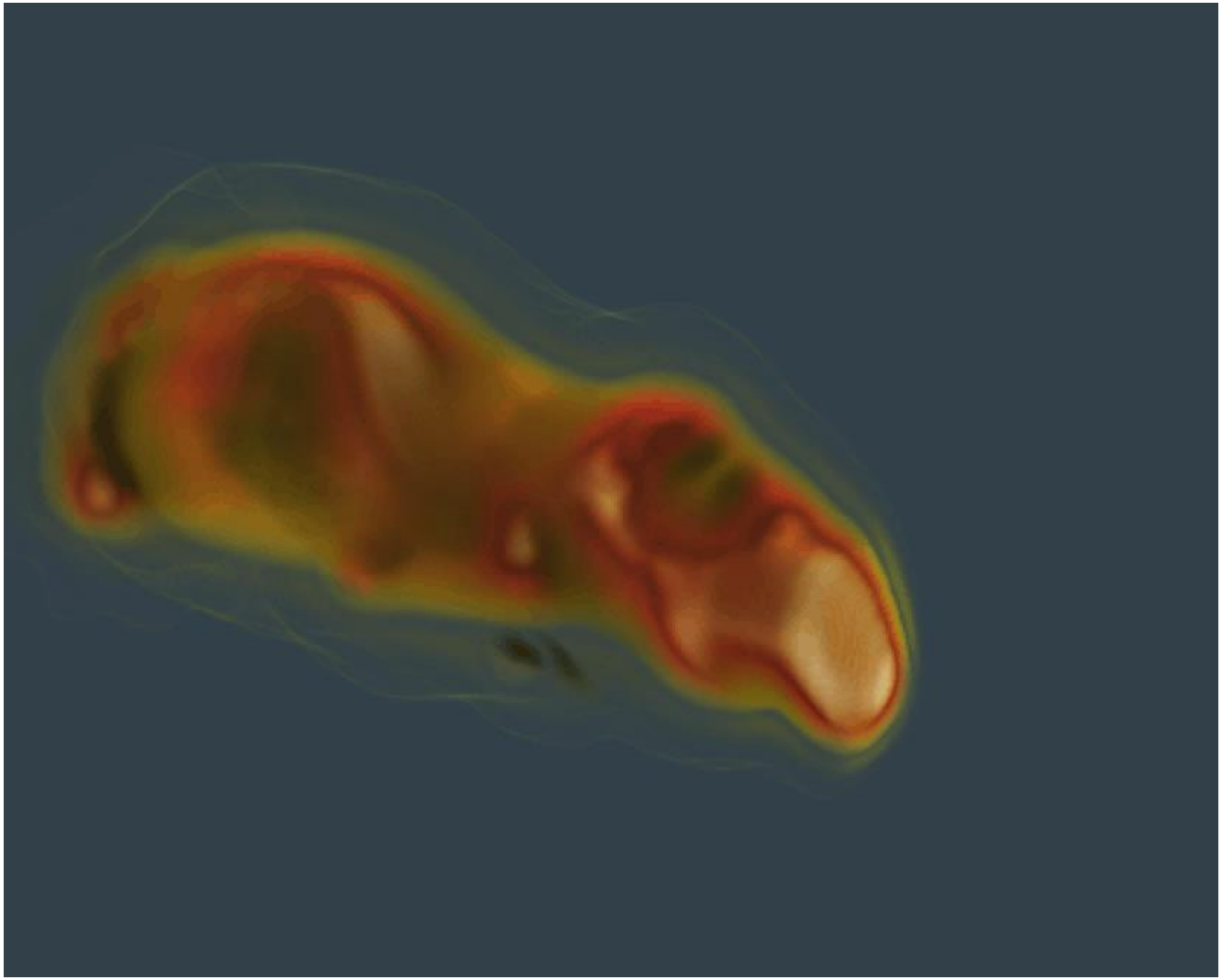
- EM-iterációs séma:
- $x_j^{(n+1)} = \dfrac{x_j^{(n)} \sum_{i=1}^{N} \boldsymbol{A}_{ij} \frac{y_i}{\tilde{y}_i}}{\sum_{i=1}^{N} \boldsymbol{A}_{ij}}$, $j = 1, \dots, M$ (minden voxelre)

  mérés

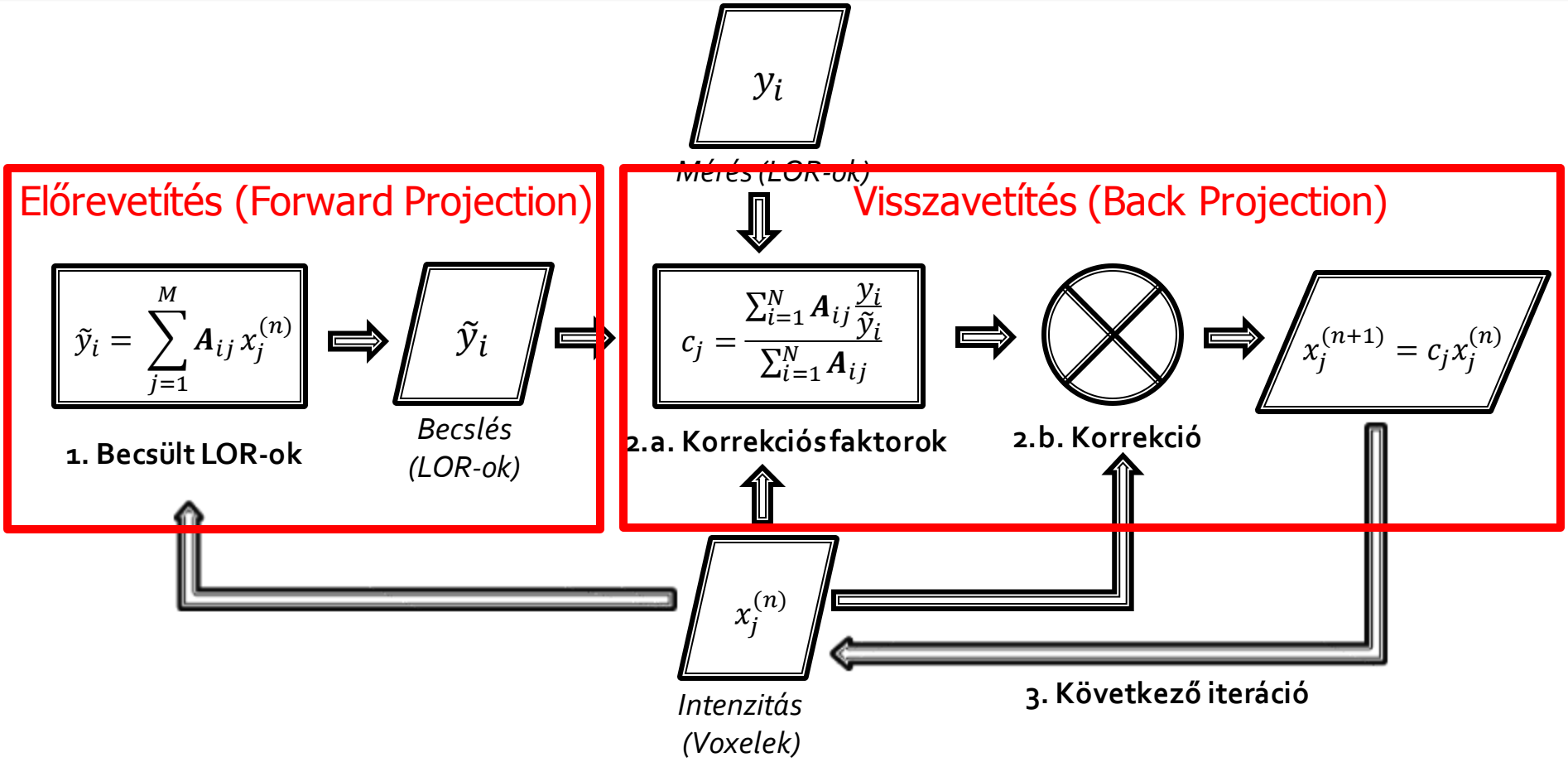
  becslés

  - Tetszőleges nemnulla $x_j^{(0)}$ jó
  - Minden iterációban a mért és becsült LOR-ok aránya alapján korrigáljuk a bomlássűrűség (*x(v)*) aktuális becslését
  - A becsült várható érték: $\tilde{y}_i = \sum_{j=1}^{M} \boldsymbol{A}_{ij} x_j$

Az EM-algoritmus (Expectation Maximization)



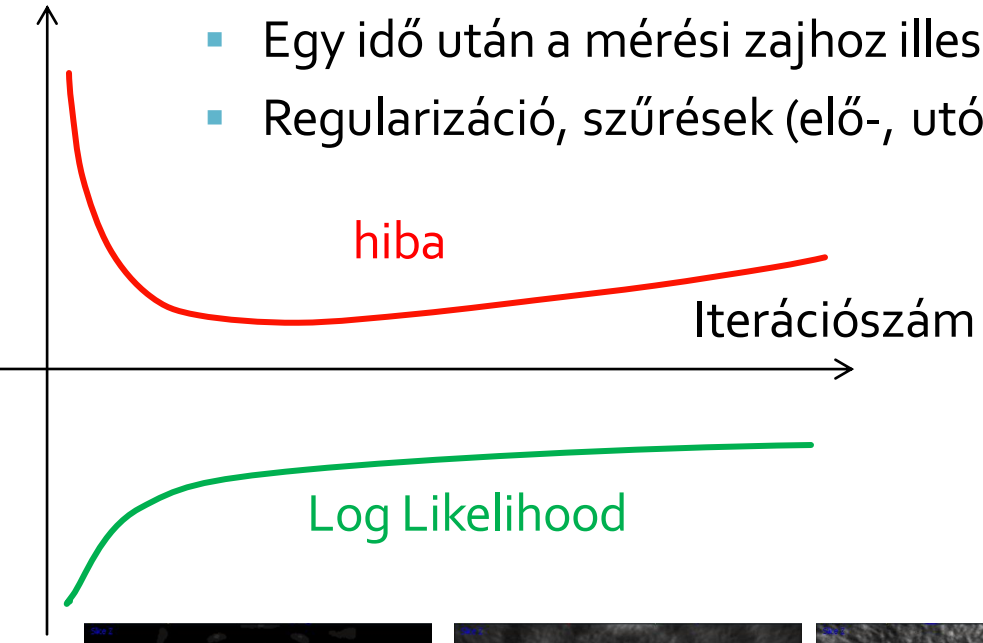
Az EM-algoritmus (Expectation Maximization)

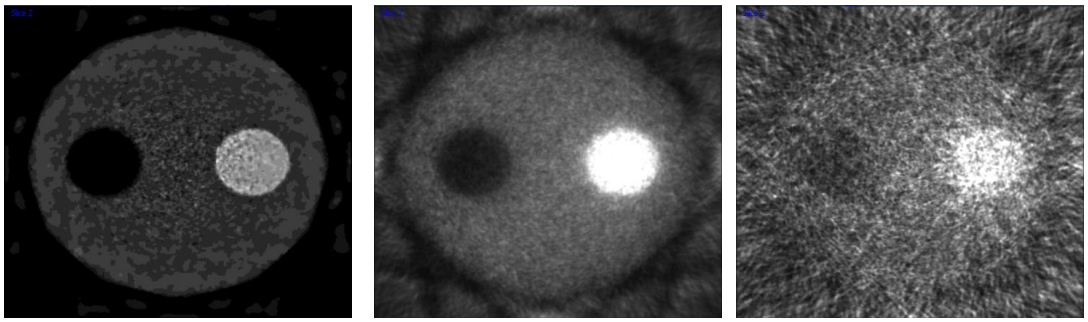


- Mindkét operátor egymástól függetlenül dolgozza fel az adatokat (párhuzamosítás!)

Hátrányok

- Rosszul kondicionált, inverz feladat: az ML-EM likelihoodban konvergál (LOR-tér), nem pl. voxeltérbeli L_2 hibában
  - Egy idő után a mérési zajhoz illeszti a rekonstruált térfogatot
  - Regularizáció, szűrések (elő-, utó-), megállási feltételek stb.



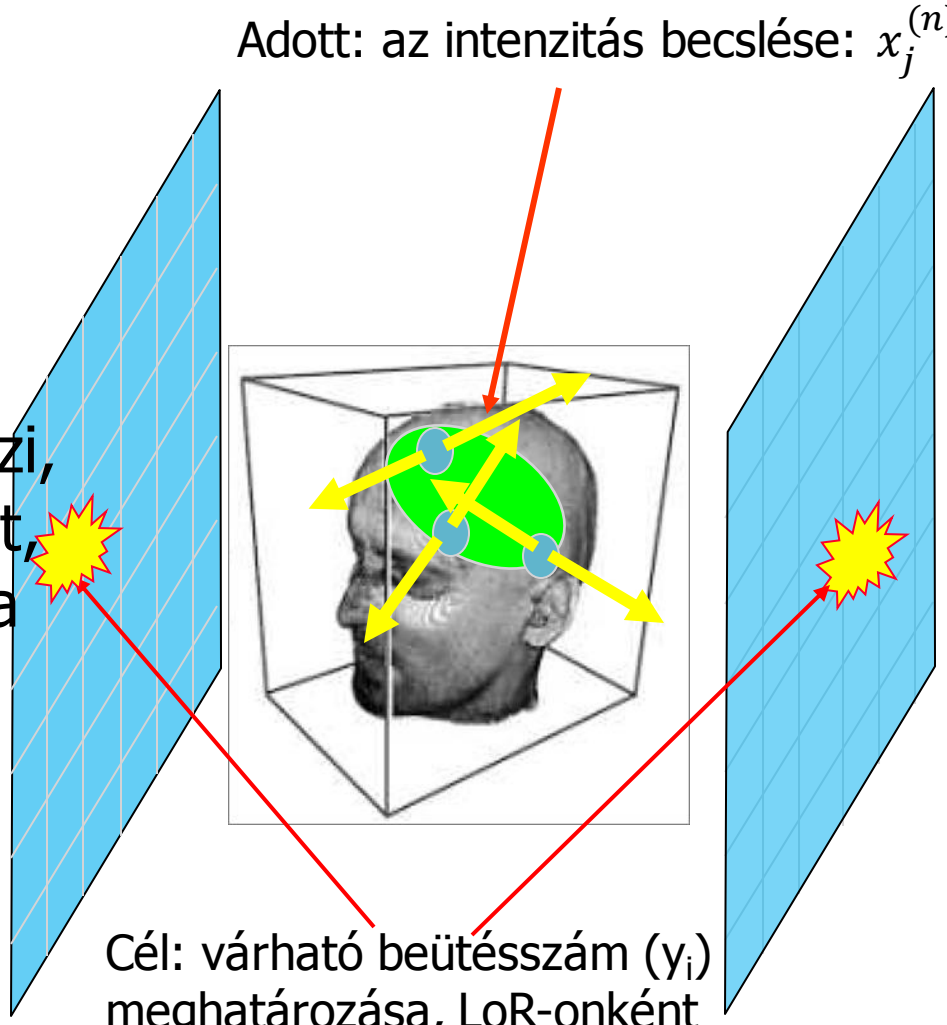


Előrevetítés: Direkt MC

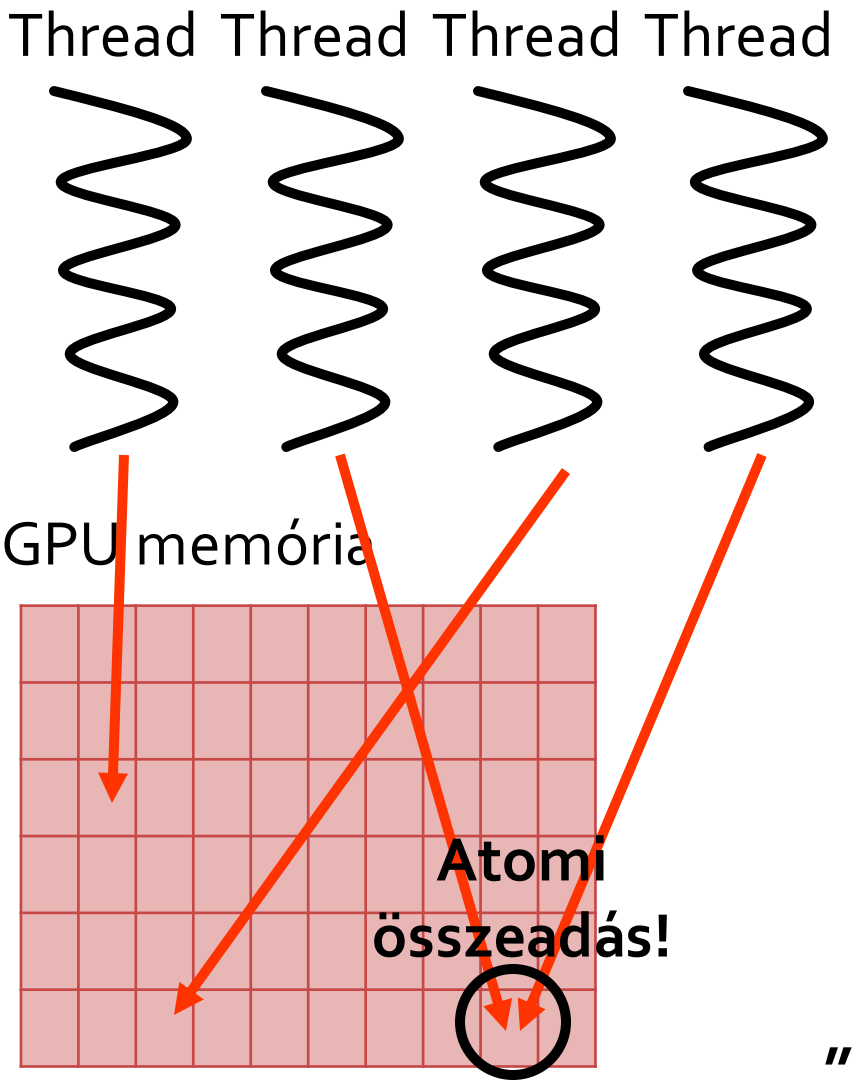
$$\tilde{y}_i = \int_V x(\vec{v}) P(\vec{v} \to i) dv$$

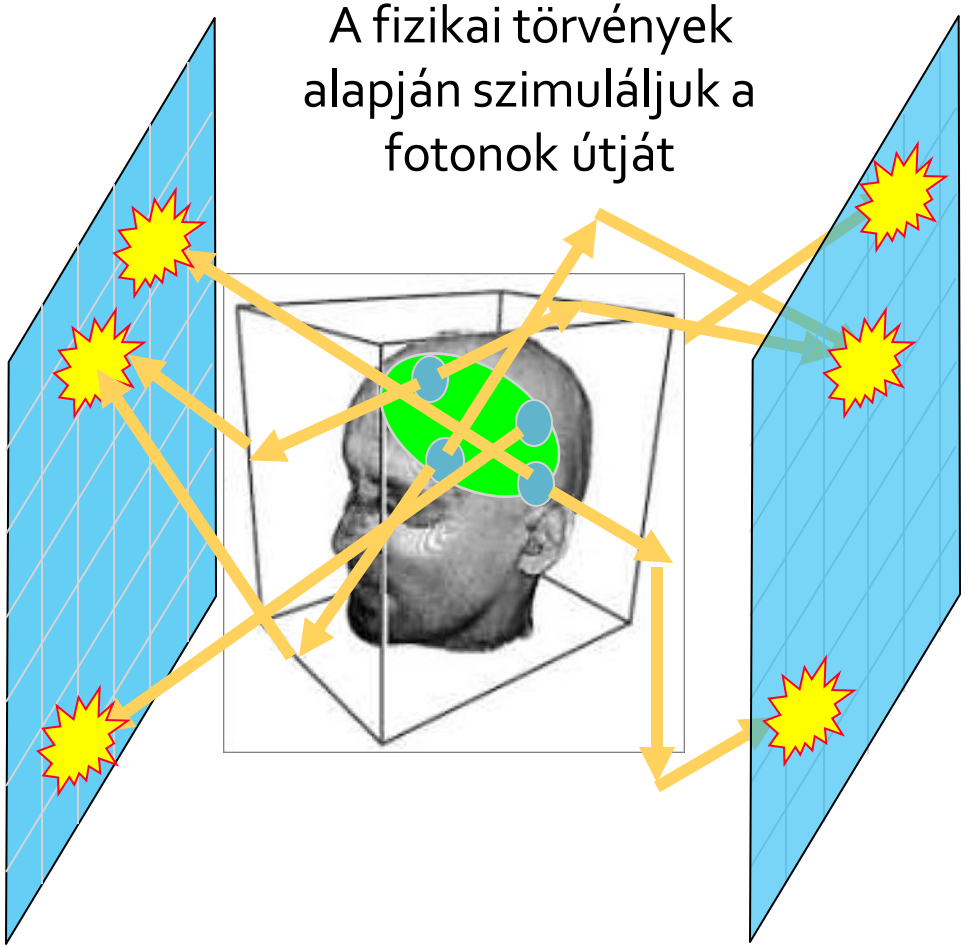
Direkt Monte-Carlo:
A térfogatot mintavételezi, szimulálja a fotonok útját, az út végén regisztrálja a becsapódás helyét

„Lövés” típus:
GPU-n általában nem hatékony!



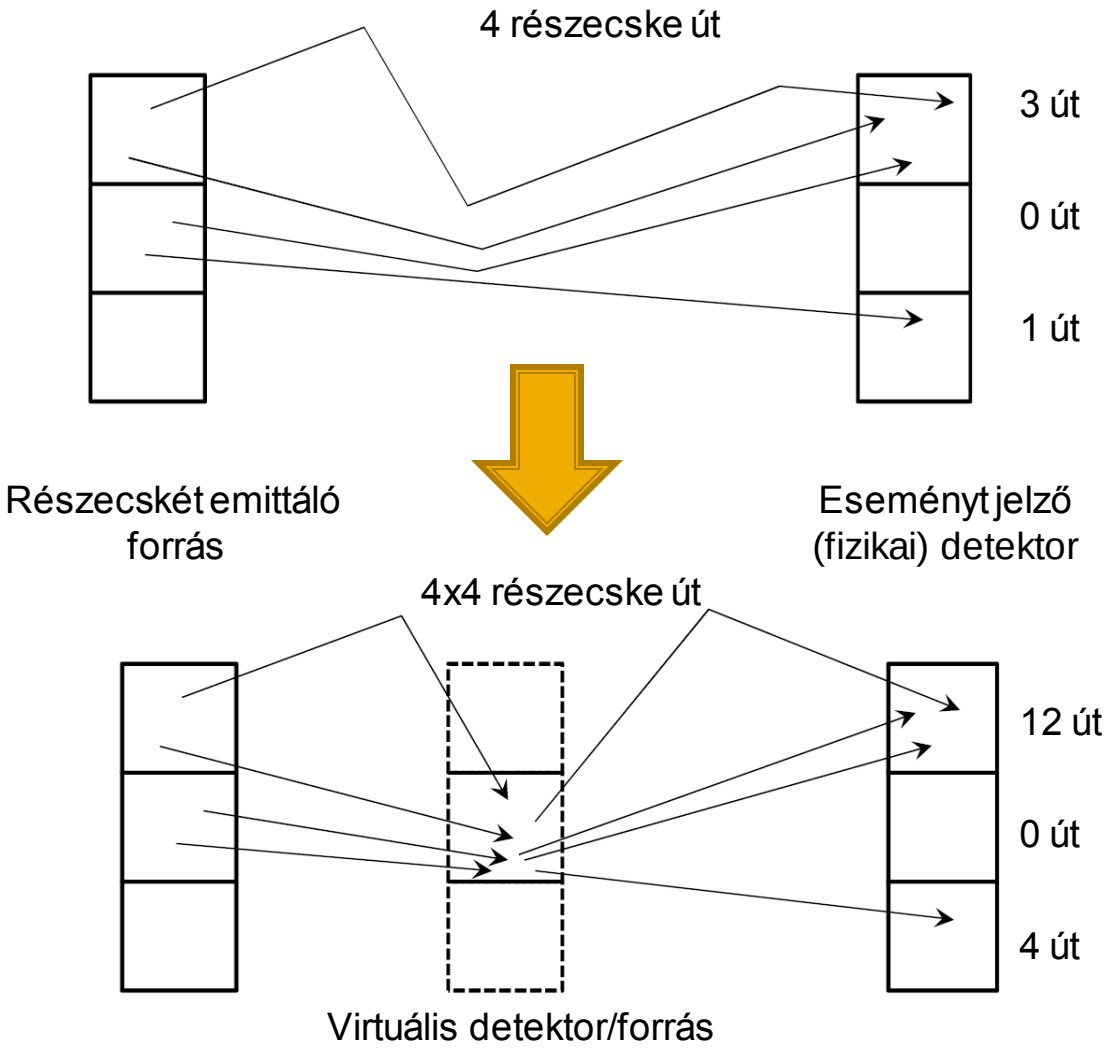
Direkt MC: szóródás





„Lövés”: GPU-n nem hatékony!

Párhuzamosítás: faktorizáció



$$\mathbf{Ax} = \mathbf{L} \cdot (\mathbf{D} + \mathbf{S}) \cdot \mathbf{P} \cdot \mathbf{x}$$

fizikai hatások

(**P**ozitronvándorlás,
Direkt hatás,
Szóródás,
Detektormodel**L**)

Előrevetítés

- Voxel-központú direkt MC: a voxelekből kiindulva szimuláljuk a fotonok útját, az út végén találatot regisztrálunk a detektorokban
  - Intuitív, „könnyű” lekódolni a fizikai jelenségeket
  - Nem illeszkedik a GPU architektúrájához
- LOR-központú „adjungált MC”: a detektorokból kiindulva megnézzük, hogy mely voxelek hatnak az adott detektorpárra
  - Gyűjtés-típusú módszer!
  - A probléma **matematikai modelljének átfogalmazása**!
  - Pl. geometriai vetítés:

$$\tilde{y}_i = \sum_{j=1}^{N_{voxel}} \mathbf{A}_{ij} x_j = \int_{\nu} \int_{2\pi} x(\vec{v}) P(\vec{v}, \vec{\omega} \to l) \frac{\mathrm{d}\omega}{2\pi} \mathrm{d}v = \int_{D_1} \int_{D_2} \frac{\cos\theta_1 \cos\theta_2}{|\vec{z}_1 - \vec{z}_2|^2} X(\vec{z}_1, \vec{z}_2) \mathrm{d}z_1 \mathrm{d}z_2$$

Jacobi det. teljes aktivitás

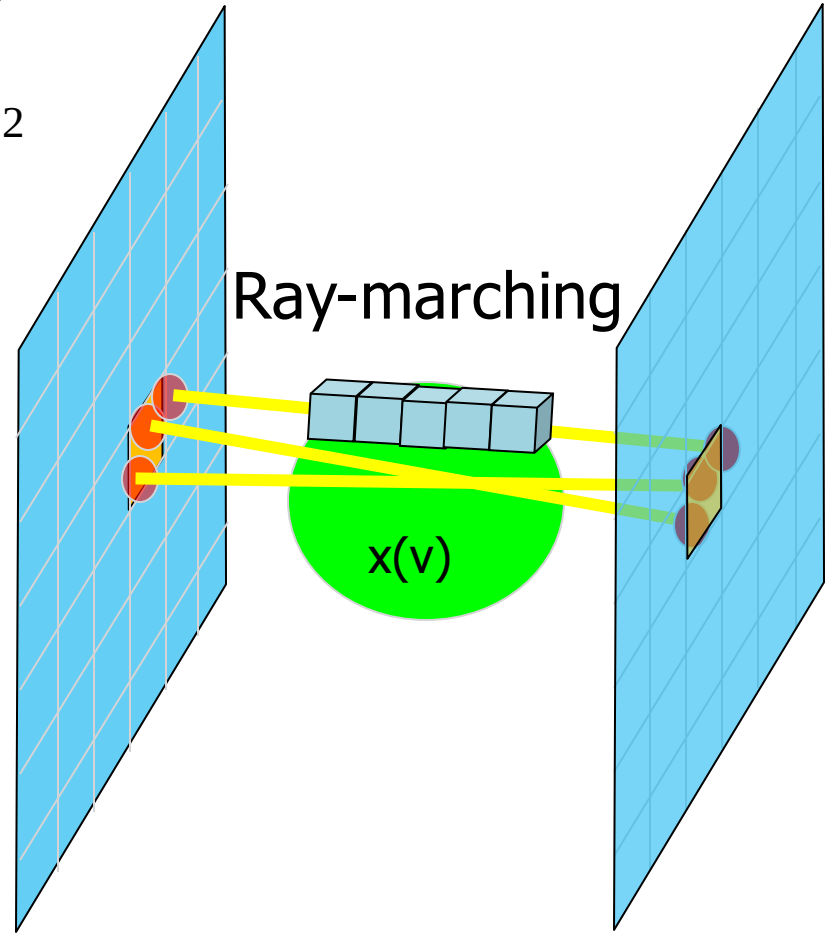
Geometriai előrevetítés: Adjungált MC

$$\int_{z_1}^{z_2} x(l)\,dl$$

$$\tilde{y}_i = \int_V x(\vec{v})P(\vec{v} \to i)dv$$

$$\tilde{y}_i = \int_{D_1}\int_{D_2} G(z_1, z_2)X(z_1, z_2)dz_1 dz_2$$

Adjungált Monte-Carlo:
A detektorfelületet mintavételezzük, összegyűjtjük a mintapontokat összekötő utak hozzájárulását



10 millió út/s (Direkt MC)
vs.
2.000 millió minta/s (adj. MC)

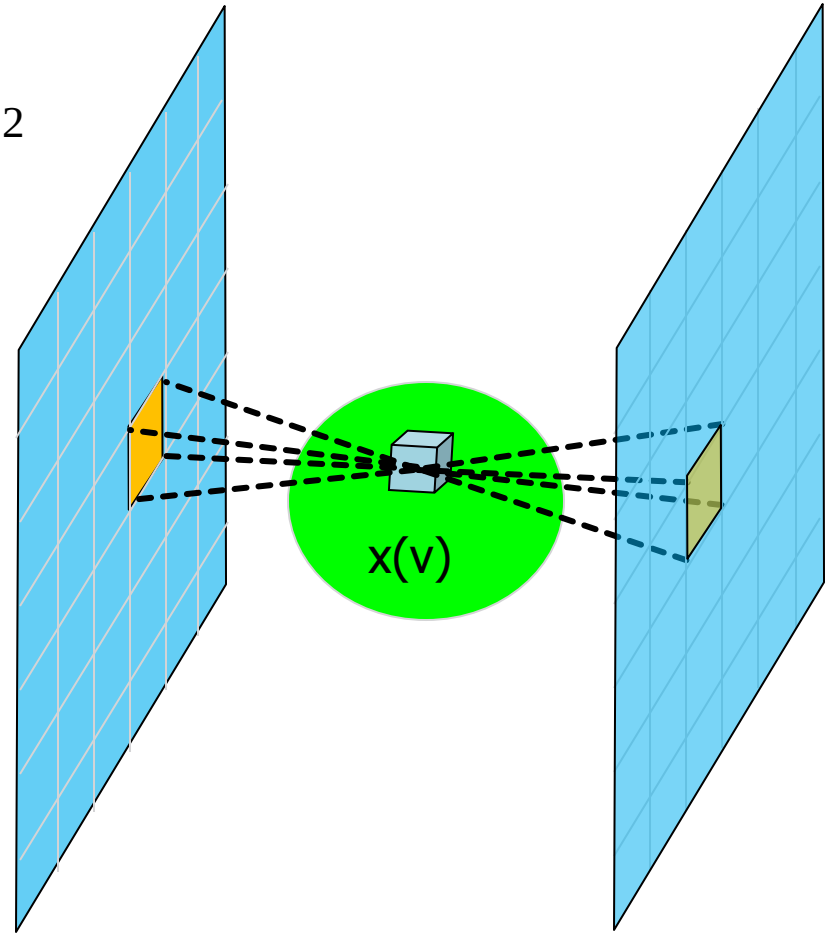
Előrevetítés

$$\frac{\cos\theta_{z_1}\cos\theta_{z_2}}{2\pi|z_1 - z_2|^2}$$

$$\tilde{y}_i = \int_{D_1 D_2} \int G(z_1, z_2) X(z_1, z_2) dz_1 dz_2$$

Valószínűség helyett a térszög jelenik meg, amelyben a detektorok felülete látszik

$$\tilde{y}_i \approx \frac{D_1 D_2}{2\pi N_{line}} \sum_{k=1}^{N_{line}} \frac{\sum_{t=1}^{N_{march}} x(\vec{l}_{kt}) \Delta l_k}{\left| \vec{z}_1^{(k)} - \vec{z}_2^{(k)} \right|^2}$$



Előrevetítés

$$\frac{\cos\theta_{z_1}\cos\theta_{z_2}}{2\pi|z_1 - z_2|^2} \qquad \int_{z_1}^{z_2} x(l)\,dl$$

$$\tilde{y}_i = \int_{D_1}\int_{D_2} G(z_1, z_2)\, X(z_1, z_2)\, dz_1 dz_2$$

Diszkrét mintákkal közelítve:

$$\tilde{y}_i \approx \frac{D_1 D_2}{2\pi N_{line}} \sum_{k=1}^{N_{line}} \frac{\sum_{t=1}^{N_{march}} x(\vec{l}_{kt})\,\Delta l_k}{\left|\vec{z}_1^{(k)} - \vec{z}_2^{(k)}\right|^2}$$

detektorfelület

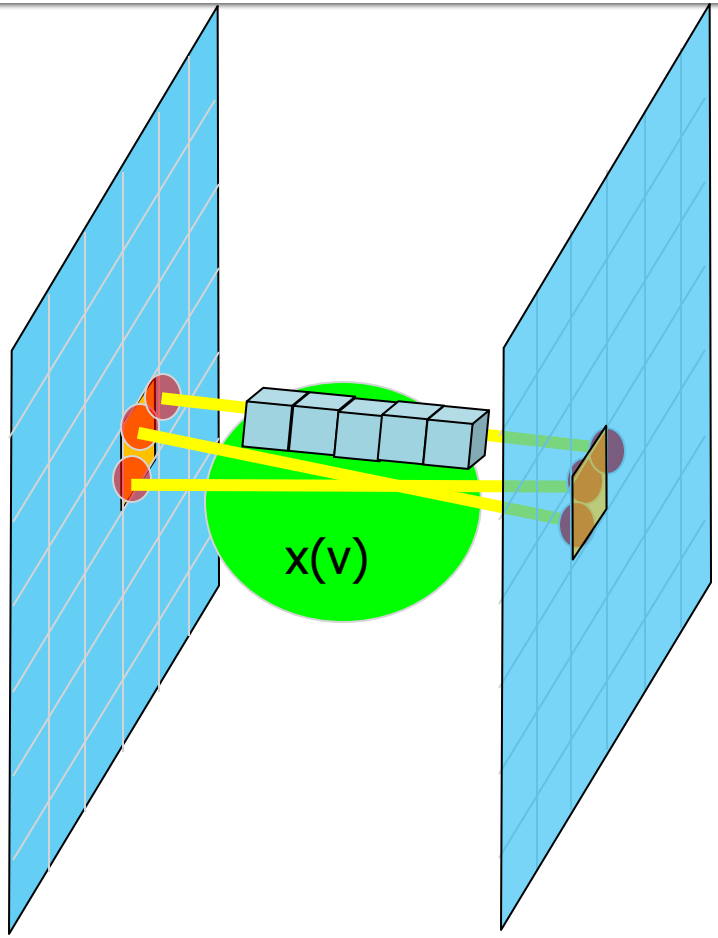
ray-marching lépésszám

x(v)-minta

ray-marching lépéshossz

vonalminták

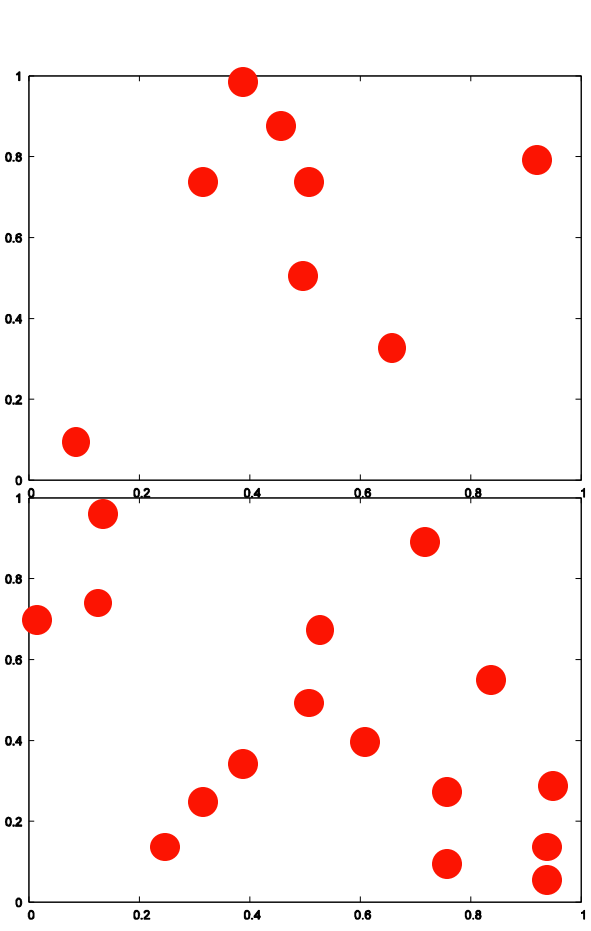
detektorminták (=a vonalminták végpontjai) távolsága

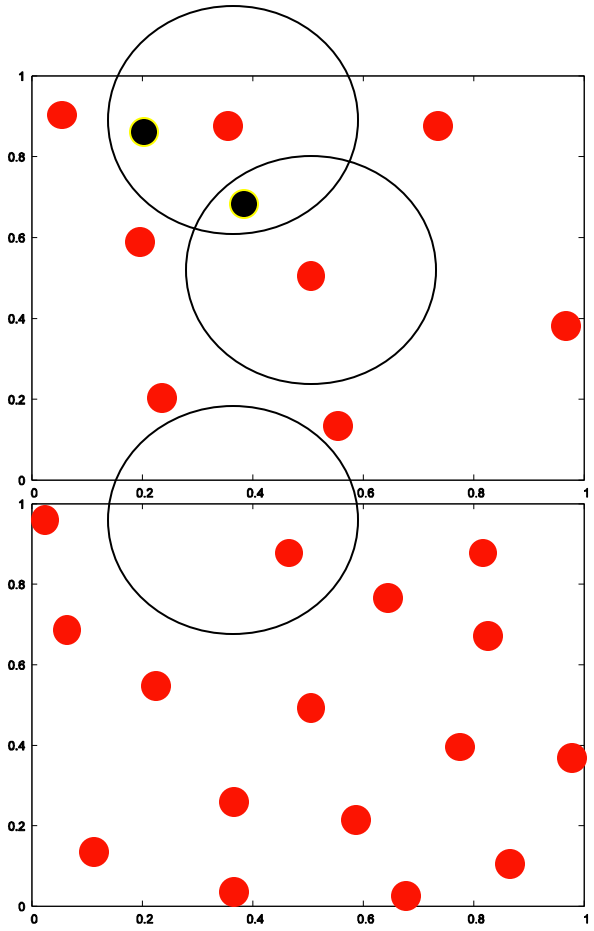


Vonalminták végpontjai

Rnd

Poisson Disk



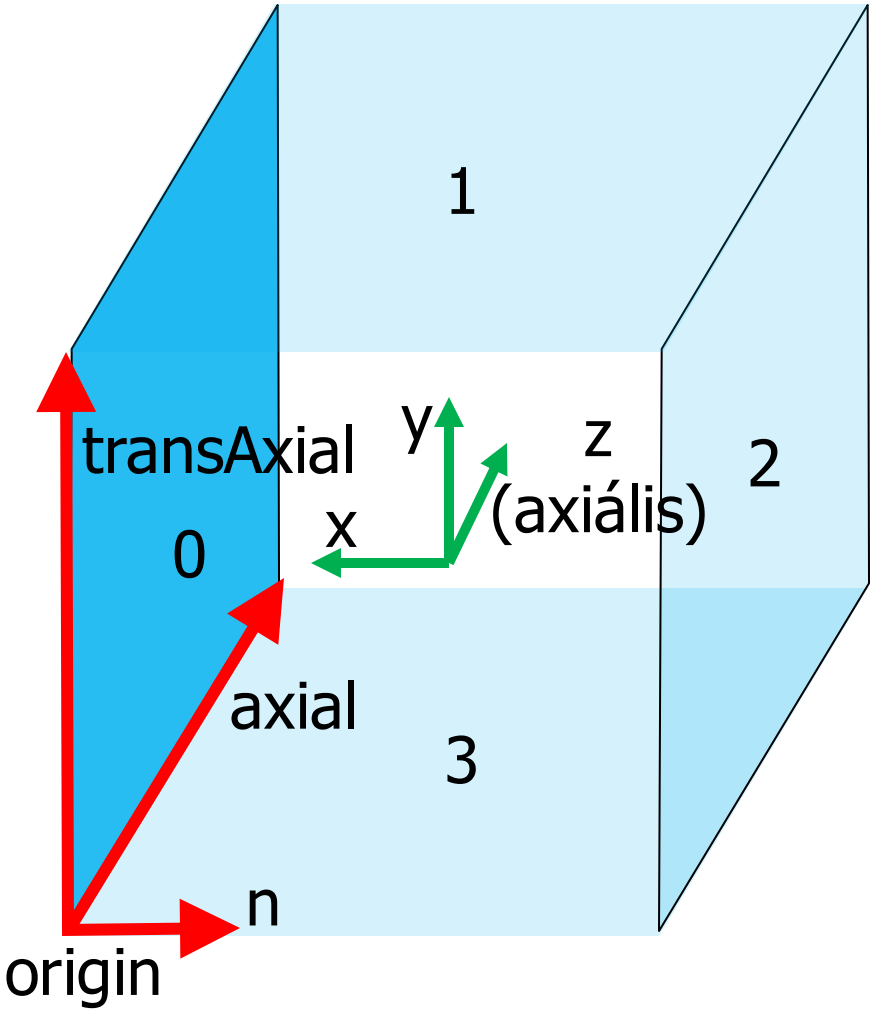


Implementáció

- PET-geometria
  - Detektorok geometriája
  - Volume leírás
- Előrevetítő operátor
- Visszavetítő operátor

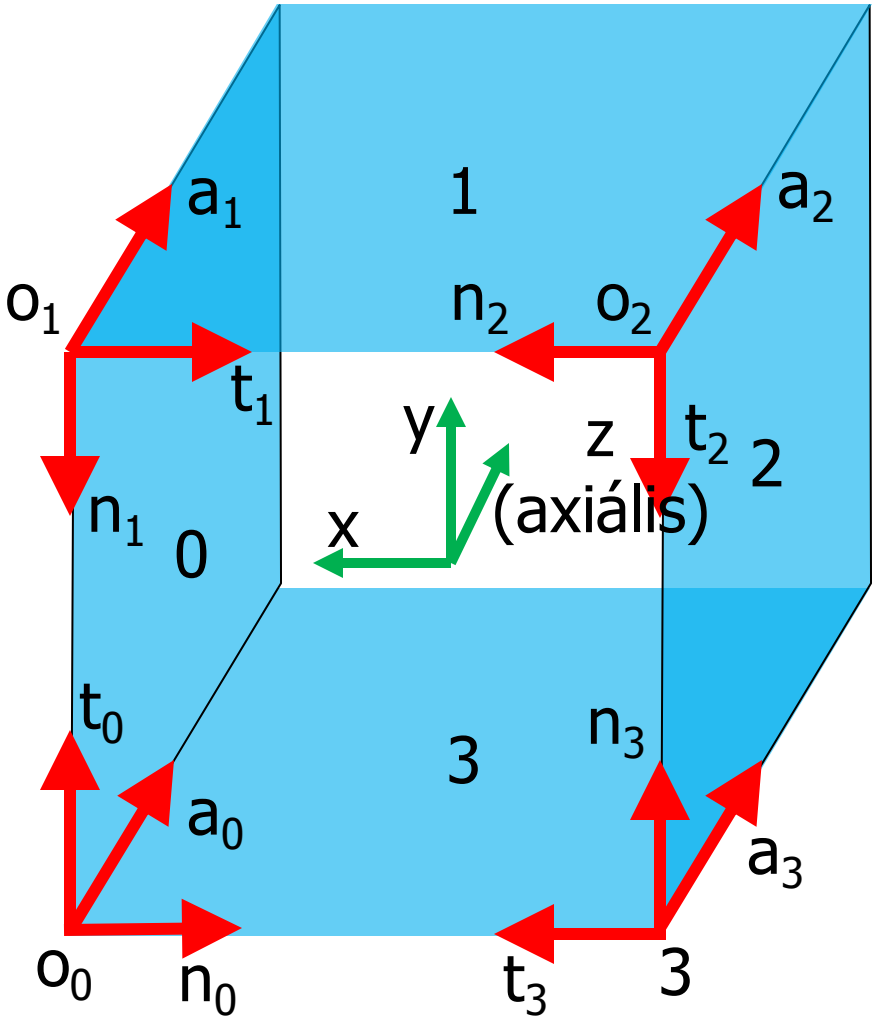
Modulok

```
struct module
{
   cl_float4 origin;
   cl_float4 axial;
   cl_float4 transAxial;
   cl_float4 n;
};

cl_float4 origin;
cl_float4 axial;
cl_float4 transAxial;
cl_float4 n;
origin.s[3] = axial.s[3] = transAxial.s[3] = n.s[3] =
0.0f;
origin.s[0] = origin.s[1] = origin.s[2] = -1.0f;
axial.s[2] = 2.0f; axial.s[0] = axial.s[1] = 0.0f;
transAxial.s[0] = 2.0f;
transAxial.s[1] = transAxial.s[2] = 0.0f;
n.s[1] = 1.0f; n.s[0] = n.s[2] = 0.0f;
```



Modulok

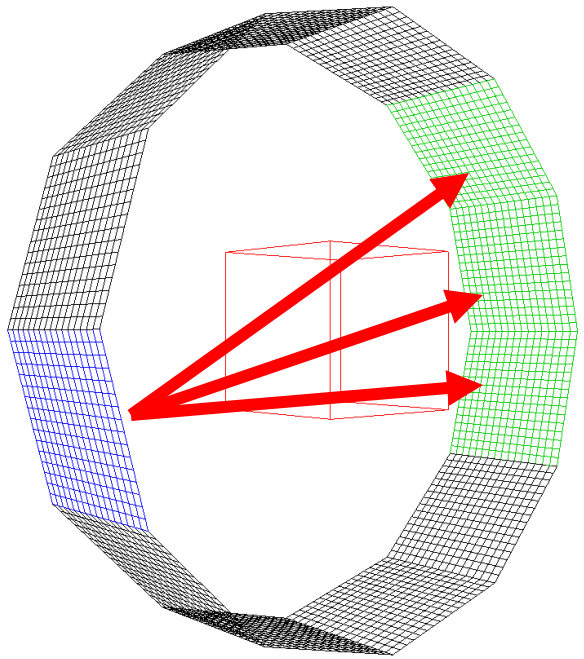
```
moduleBufferGPU = clCreateBuffer(context,
CL_MEM_READ_WRITE,
sizeof(module)*nModules, NULL, NULL);
for ( int i = 0; i < nModules; ++i )
{
   moduleBufferCPU[i].origin = origin;
   moduleBufferCPU[i].axial = axial;
   moduleBufferCPU[i].transAxial = transAxial;
   moduleBufferCPU[i].n = n;

   rotategoxy(origin);
   rotategoxy(transAxial);
   rotategoxy(n);
}
clEnqueueWriteBuffer(commands,
                moduleBufferGPU, CL_TRUE, 0,
                sizeof(module)*nModules,
                moduleBufferCPU, 0, NULL, NULL);
```



Koincidencia

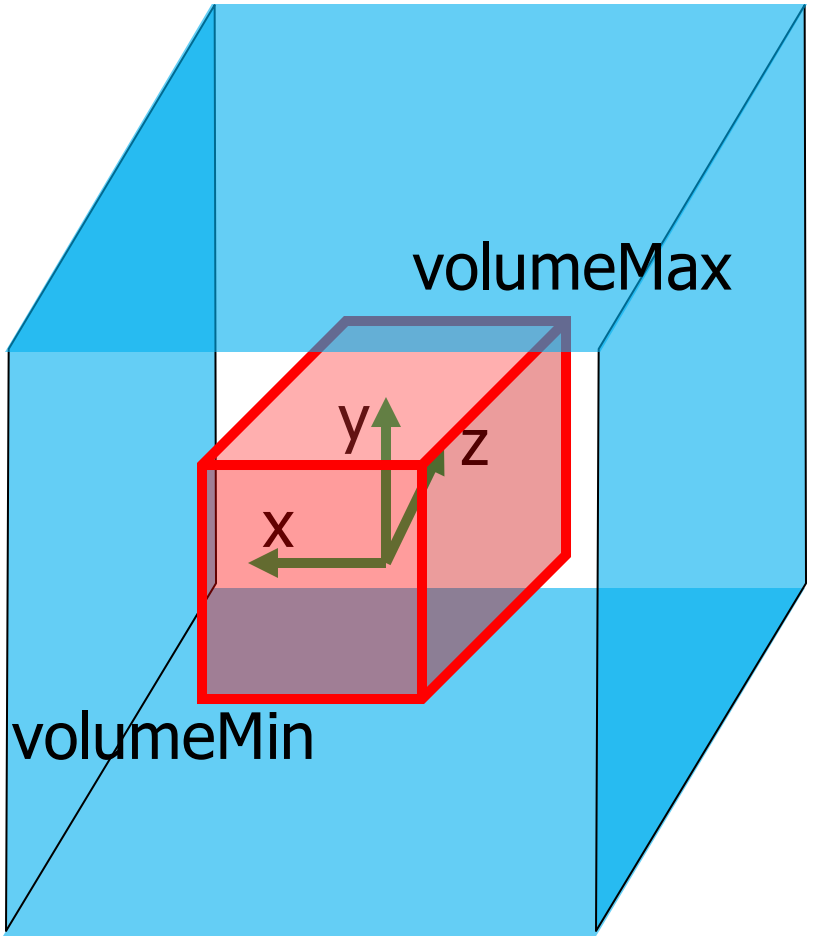
- Int ⟶ Int × Int leképzés
- Melyik panel melyik másikkal alkothat LOR-okat

```
// csúnya, de egyszerű ☺

int2 getModules(int pair)
{
        switch ( pair )
        {
                case 0: return (int2)(0,2);
                case 1: return (int2)(1,3);
        };
        return (int2)(0,0);
}
```



Volume fizikai méret

```
volumeMin.s[0] = volumeMin.s[1] =
        volumeMin.s[2] = -0.5f;
volumeMax.s[0] = volumeMax.s[1] =
        volumeMax.s[2] = 0.5f;
volumeMin.s[3] = volumeMax.s[3] = 0.0f;
```



Előrevetítő kernel

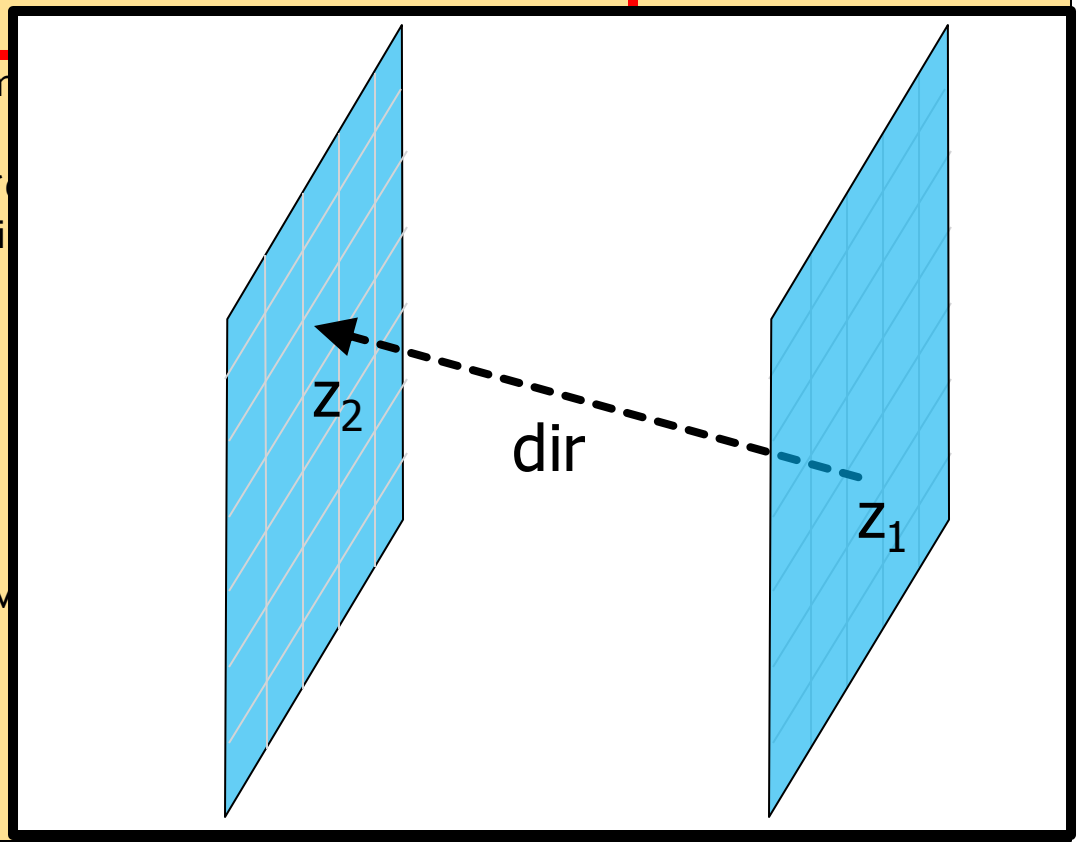
```
__kernel
void forwardProjection(
        const int nAxial,                   // modulok axiális felbontása
        const int nTransAxial,              // modulok tranzaxiális felbontása
        const int resolution,               // volume felbontás (most köbös)
        const float detectorArea,           // detektor felszín (térszöghöz)
        float4 volumeMin,                   // AABB
        float4 volumeMax,                   // AABB
        __global struct module* modules,    // modulleírók
        __global float* measuredLors,       // mért LOR beütésszámok
        __global float* estimatedLors,      // becsült LOR-ok
        __global float* volumeBuffer        // bomlássűrűség
)
{
 // ...
}
```

Problématér felosztása

```
forwardProjectionSize[0] = nPairs;
forwardProjectionSize[1] = forwardProjectionSize[2] = nTransAxial*nAxial;
CL_SAFE_CALL( clEnqueueNDRangeKernel(commands, forwardProjectionKernel,
                                     3, NULL, forwardProjectionSize, NULL, 0, NULL, NULL) );
__kernel void forwardProjection(
// ...
{
  int iPair = get_global_id(0);
  int iAxial1 = get_global_id(1) % nAxial;
  int iTransAxial1 = get_global_id(1) / nAxial;
  int iAxial2 = get_global_id(2) % nAxial;
  int iTransAxial2 = get_global_id(2) / nAxial;
  int lorIndex = getLorIndex( iPair, iAxial1, iAxial2, iTransAxial1, iTransAxial2, nAxial, nTransAxial );
}
int getLorIndex( int pair, int axial1, int axial2, int transAxial1, int transAxial2, int nAxial, int
nTransAxial ) {
        int lorNumber = nAxial*nAxial*nTransAxial*nTransAxial;
        int u = axial1 * nTransAxial + transAxial1;
        int v = axial2 * nTransAxial + transAxial2;
        return lorNumber*pair + u*nAxial*nTransAxial + v;
}
```

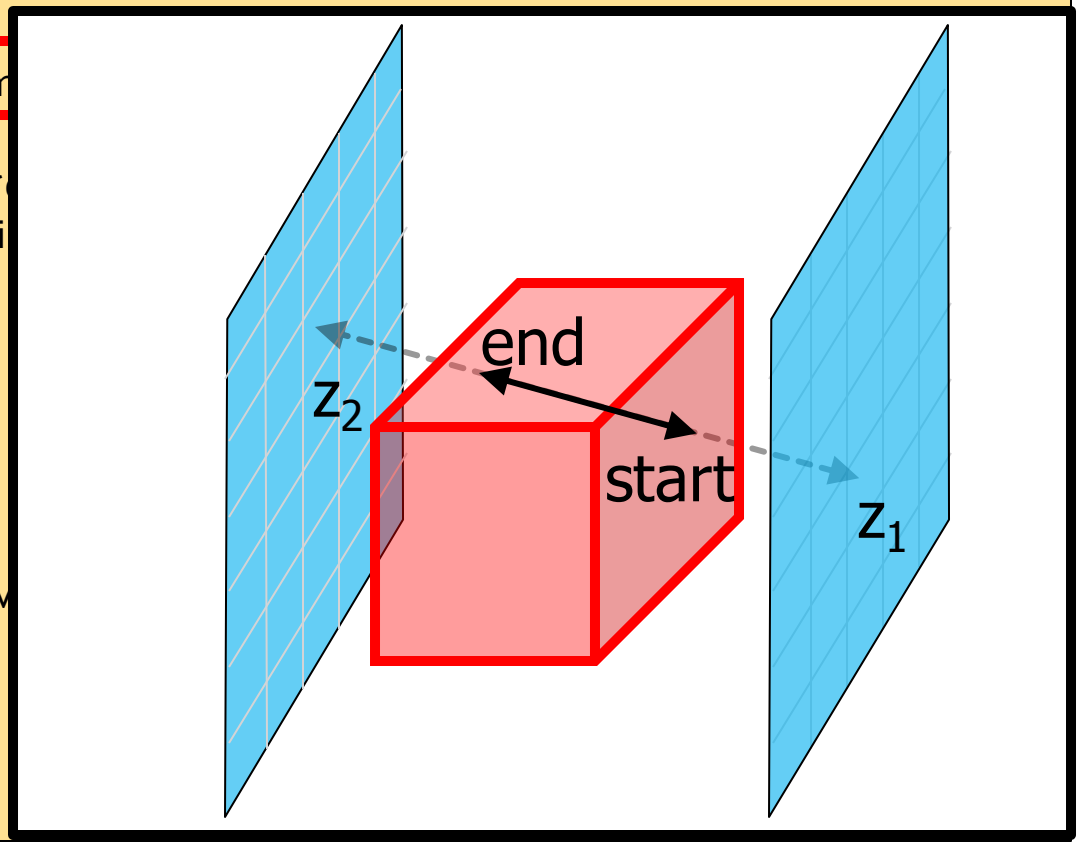
Előrevetítő kernel

```
float4 z1 =  // 1 vonalminta, a detektor közepén
             modules[iModule.x].origin  +
             modules[iModule.x].transAxial*(iTransAxial1+0.5f)/nTransAxial  +
             modules[iModule.x].axial*(iAxial1+0.5f)/nAxial;
float4 z2 = ...
float4 dir = z2-z1;
if ( intersectBox(  z1, dir, volumeMin,  volum
   float G = -detectorArea*detectorArea  *
             dot(modules[iModule.x].n,dir)*(
             / (2.of*M_PI*dot(dir,dir)*dot(di
   float4 start = z1+tnear*dir;
   float4 end = z1+tfar*dir;
   float4 step = (end - start) / resolution;
   float dl = length(step);
   float4 voxel = start;
   for ( int i = 0; i < resolution;  ++i ) {
      float x = getIntensity(  (voxel - volumeM
      xSum += G*x*dl;
      voxel += step;
   }
}
```



Előrevetítő kernel

```
float4 z1 =  // 1 vonalminta, a detektor közepén
             modules[iModule.x].origin  +
             modules[iModule.x].transAxial*(iTransAxial1+0.5f)/nTransAxial  +
             modules[iModule.x].axial*(iAxial1+0.5f)/nAxial;
float4 z2 = ...
float4 dir = z2-z1;
if ( intersectBox(  z1, dir, volumeMin,  volum
   float G = -detectorArea*detectorArea  *
             dot(modules[iModule.x].n,dir)*
             / (2.0f*M_PI*dot(dir,dir)*dot(di
   float4 start = z1+tnear*dir;
   float4 end = z1+tfar*dir;
   float4 step = (end - start) / resolution;
   float dl = length(step);
   float4 voxel = start;
   for ( int i = 0; i < resolution;  ++i ) {
      float x = getIntensity(  (voxel - volumeM
      xSum += G*x*dl;
      voxel += step;
   }
}
```



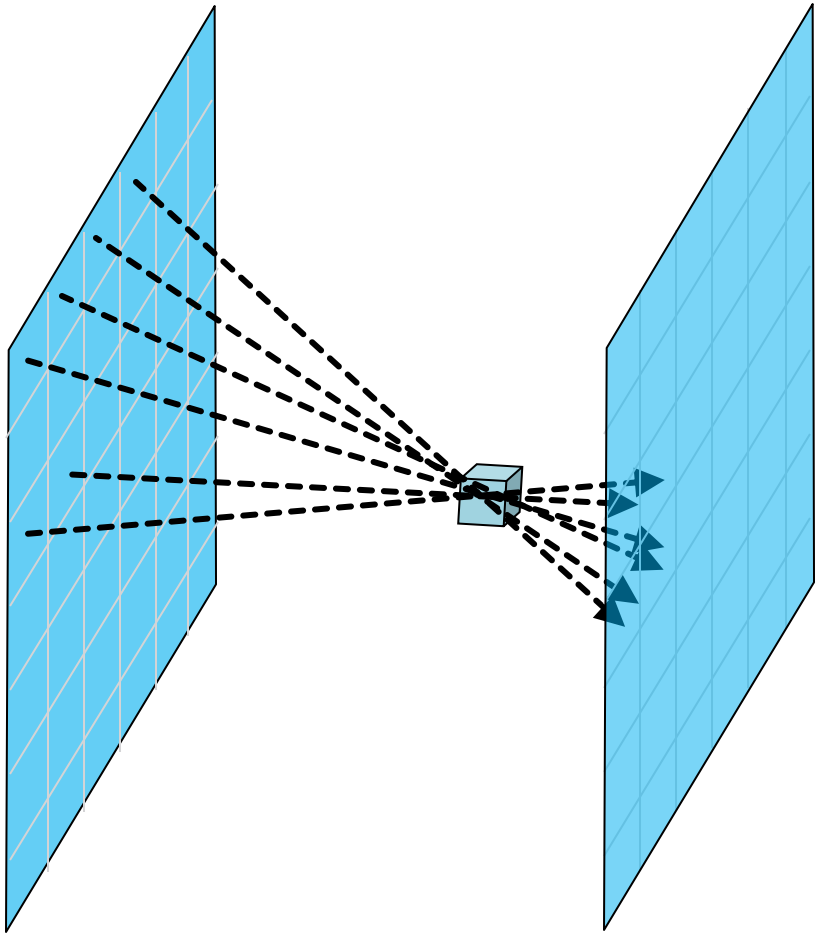
Előrevetítő kernel

```
float4 z1 =  // 1 vonalminta, a detektor közepén
             modules[iModule.x].origin  +
             modules[iModule.x].transAxial*(iTransAxial1+0.5f)/nTransAxial  +
             modules[iModule.x].axial*(iAxial1+0.5f)/nAxial;
float4 z2 = ...
float4 dir = z2-z1;
if ( intersectBox(  z1, dir, volumeMin,  volumeMax, &tnear, &tfar ) ) {
  float G = -detectorArea*detectorArea  *
             dot(modules[iModule.x].n,dir)*dot(modules[iModule.y].n,dir)
             / (2.0f*M_PI*dot(dir,dir)*dot(dir,dir));
  float4 start = z1+tnear*dir;
  float4 end = z1+tfar*dir;
  float4 step = (end - start) / resolution;
  float dl = length(step);
  float4 voxel = start;
  for ( int i = 0; i < resolution;  ++i ) {
    float x = getIntensity(  (voxel - volumeMin) / (volumeMax  - volumeMin),  resolution,  volumeBuffer );
    xSum += G*x*dl;
    voxel += step;
  }
}
```

Ray marching

Visszavetítés

- $x_j^{(n+1)} = \frac{x_j^{(n)} \sum_{i=1}^{N} \boldsymbol{A}_{ij} \frac{y_i}{\tilde{y}_i}}{\sum_{i=1}^{N} \boldsymbol{A}_{ij}}$
- Minden olyan LOR-t meg kell látogatnunk, amely metszi az adott voxelt!
- Minden panelpárra végiggyalogolunk a távolabb lévő panelen és a detektorokat a másik panelre vetítjük

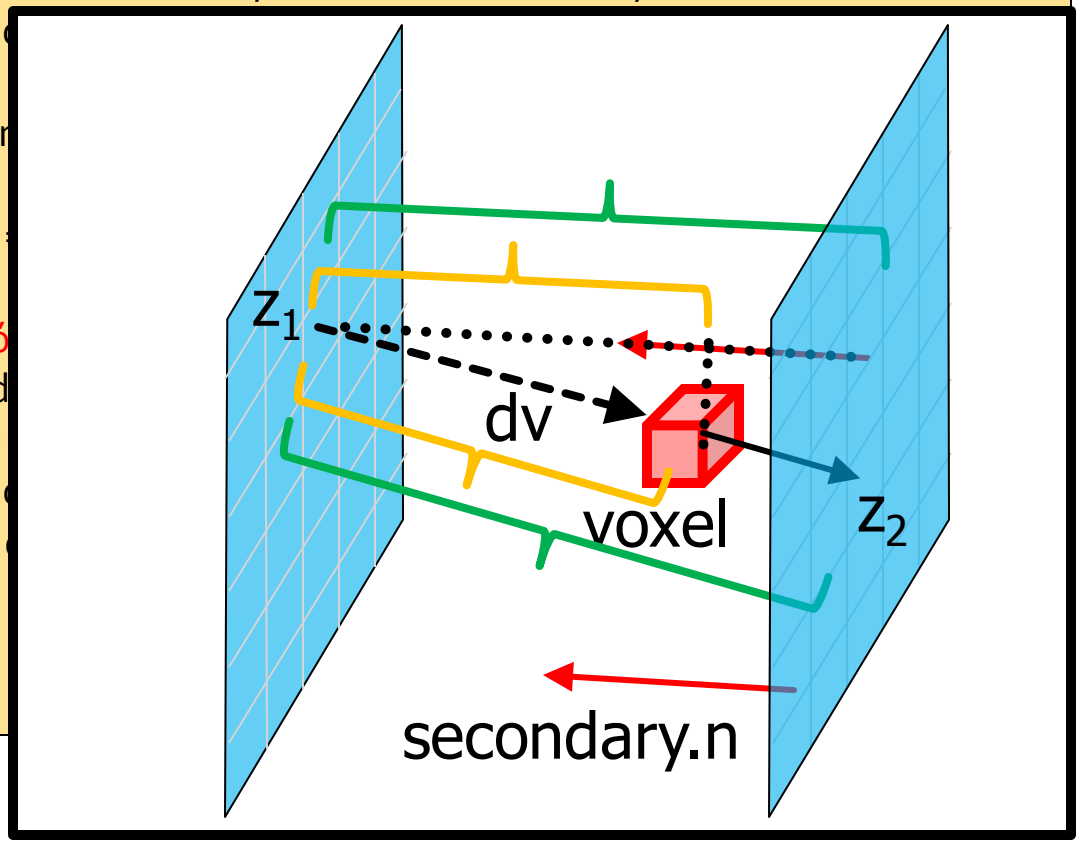


Visszavetítő kernel

```
__kernel
void backProjection(
        const int nPairs,                       // párok száma
        const int nAxial,                       // modul axiális felbontása
        const int nTransAxial,                  // modul tranzaxiális felbontása
        const int resolution,                   // volume felbontás  (most köbös)
        float4 volumeMin,                       // AABB
        float4 volumeMax,                       // AABB
        __global struct module* modules,        // modulleírók
        __global float* estimatedLors,          // mérés és becsült LOR-ok aránya
        __global float* volumeBuffer            // bomlássűrűség
){
        int4 iVoxel = (int4)(get_global_id(0),get_global_id(1),get_global_id(2),0);
        int linearIndex  = iVoxel.x + iVoxel.y*resolution  + iVoxel.z*resolution*resolution;
        float4 voxel = (float4)(iVoxel.x,iVoxel.y,iVoxel.z,0);
        voxel /= resolution;                                    // [0,1]
        voxel = (volumeMax-volumeMin)*voxel  + volumeMin;  // [volumeMin,VolumeMax]
        …
}
```

Visszavetítő kernel

```
for ( int iPair = 0; iPair < nPairs; ++iPair ) {                    // végigmegyünk minden páron
        int2 iModule = getModules(iPair);
        struct module primary = // távolabbi
        struct module secondary = // közelebbi
        float sTransAxial = nTransAxial / dot(secondary.transAxial,secondary.transAxial);
        float sAxial = nAxial / dot(secon
        for ( int p = 0; p < nAxial; ++p )
                for ( int q = 0; q < nTr
                {
                        float4 z1 =
                        float4 dv
                        // hasonló
                        float t = d
                        float4 z2
                        float fr = (
                        float fs = (
                        // ...
                }
}
```

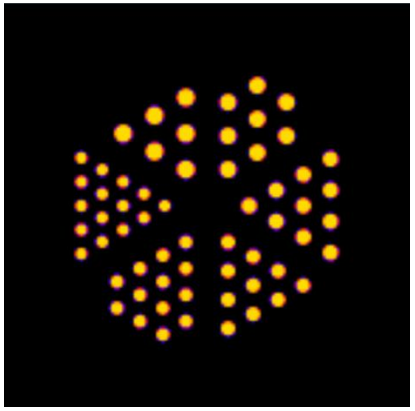


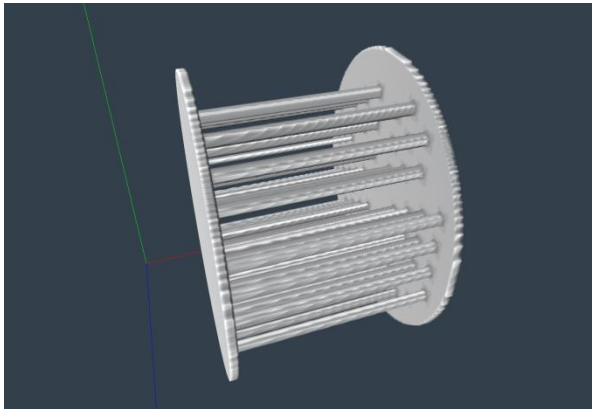
Visszavetítő kernel

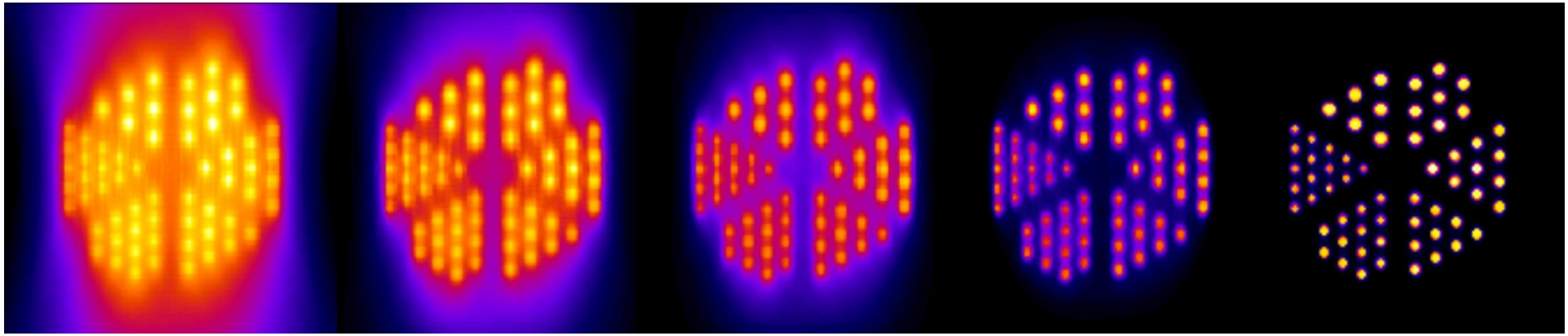
```
if ( 0 <= fr && fr < nAxial && 0 <= fs && fs < nTransAxial )
{
        int r = (int)fr;
        int s = (int)fs;
        int lorIndex = getLorIndex( iPair, p, q, r, s, nAxial, nTransAxial ); // sorrend! (ld getModules)
        float y_div_yw = estimatedLors[lorIndex];
        float ddv = length(dv);
        float Alv = dot(primary.n,dv) / (ddv*ddv*ddv);  // itt csak az egyik térszöggel közelítjük G-t

        numerator += y_div_yw * Alv;
        denominator += Alv;
}

if ( denominator > 0.0f )          // = van olyan LOR, ami átmegy a voxelen
{
        volumeBuffer[linearIndex] *= numerator / denominator;
}
```

Iteratív rekonstrukciós algoritmus

- Derenzo fantom





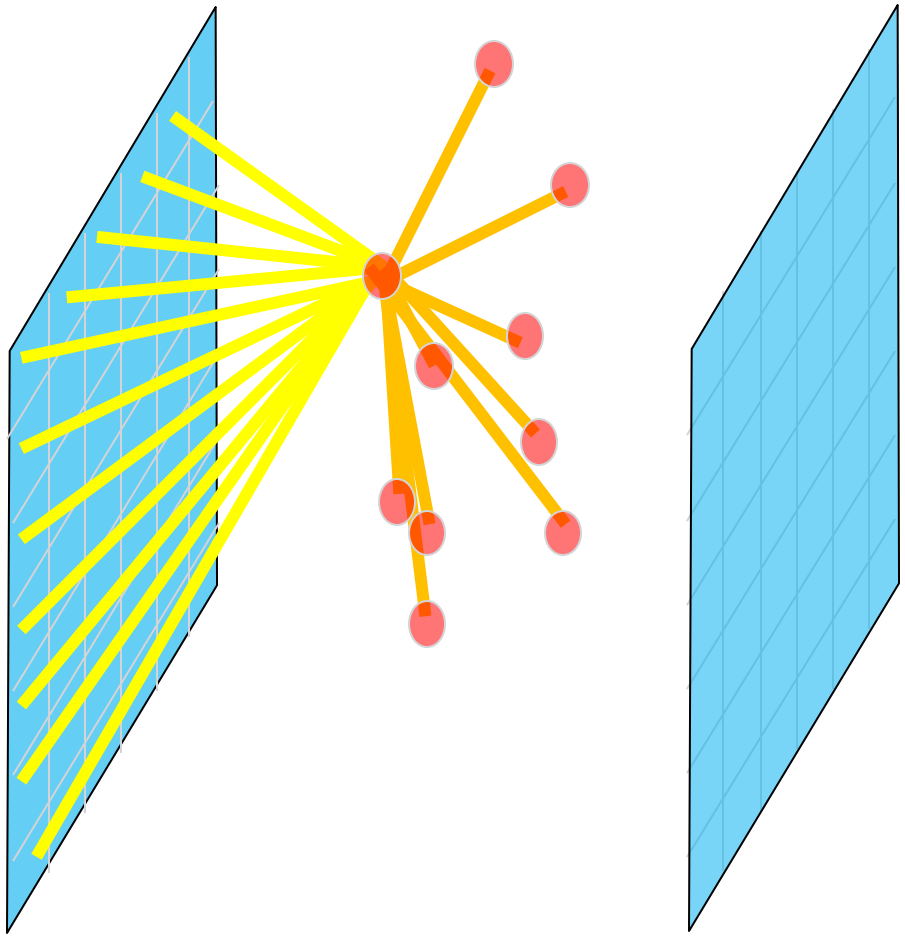


Egyéb hatások

- Testen belüli szóródás
- Detektoron belüli szóródás
- Alapötlet szintjén

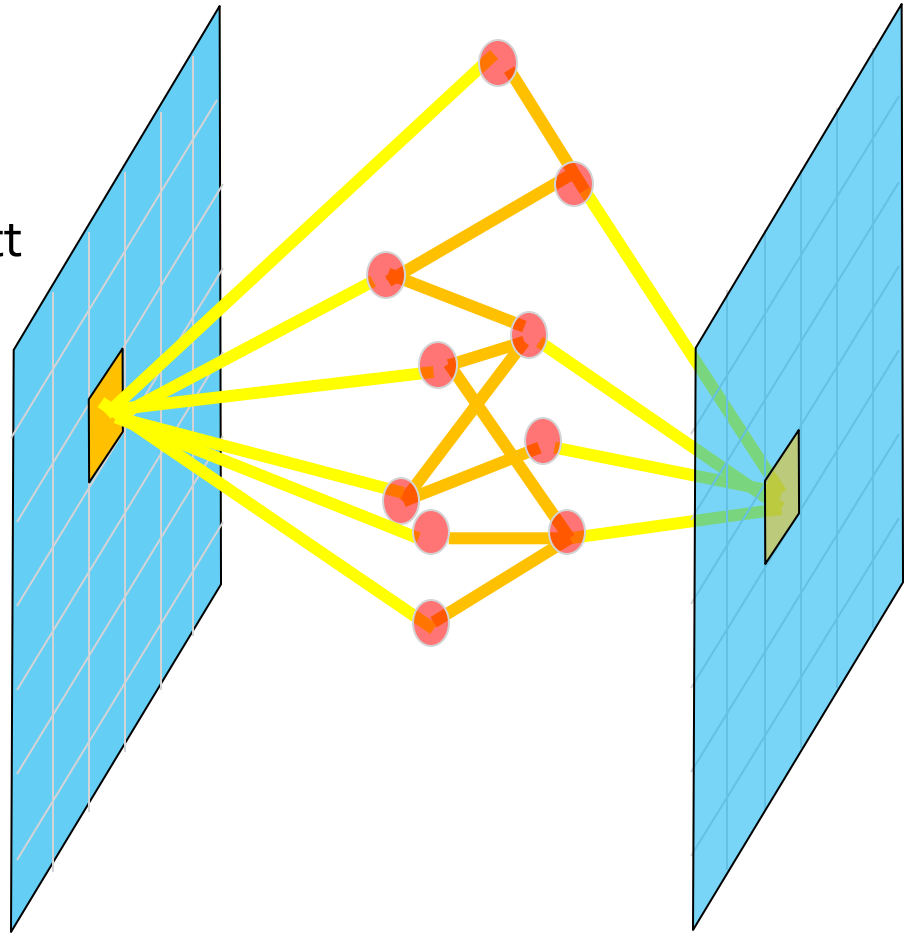
Szóródásmodell (kétszeres, pl.)

- Alapötlet: utak újrafelhasználása
- 1: szóródási pontok generálása
  - Fontosság szerint
  - Néhány 100!
- 2: összekötés a detektorokkal
  - Utak hozzájárulásának kiszámítása, elnyelődés figyelembevételével
  - Mintha a szóródási pontok is detektorok lennének
  - Néhány 10.000 detektor
- 3: Szóródási pontok összekötése, utak kiértékelése



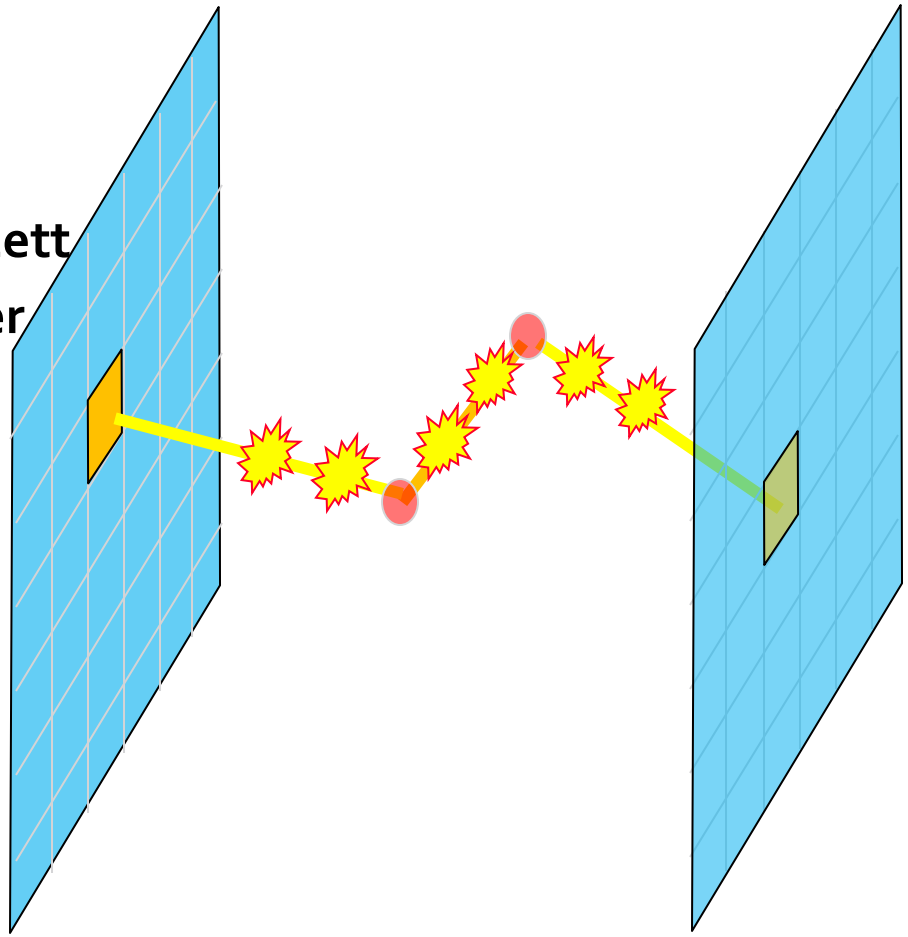
Szóródásmodell (kétszeres)

- Alapötlet: utak újrafelhasználása
- 4: Utak kombinálása, LOR-onként
  - Eredmény: 3-hosszú törtvonalak
  - Minden, a törtvonalon keletkezett fotonpár tagjai összesen kétszer szóródtak
  - A szóródási pontokat összekötő szakaszok **minden LOR-ra azonosak**!



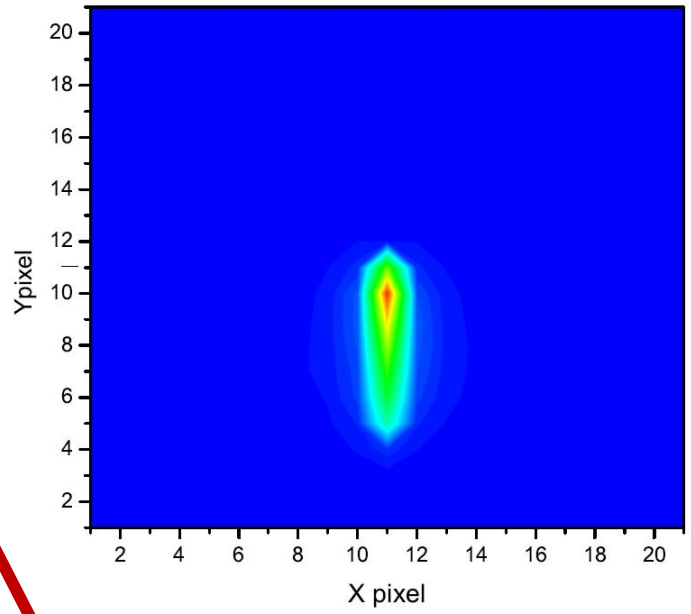
Szóródásmodell (kétszeres)

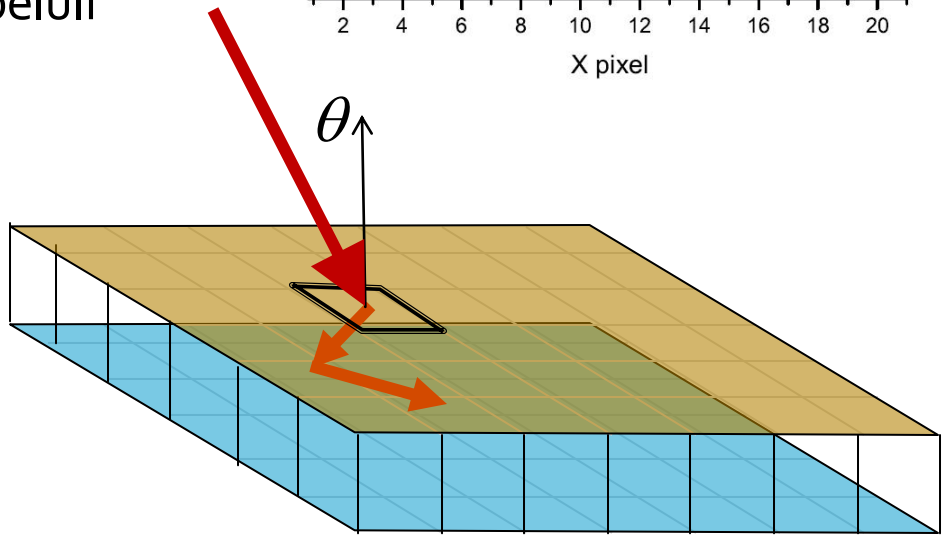
- Alapötlet: utak újrafelhasználása
- 4: Utak kombinálása, LOR-onként
  - Eredmény: 3-hosszú törtvonalak
  - **Minden, a törtvonalon keletkezett fotonpár tagjai összesen kétszer szóródtak**
  - A szóródási pontokat összekötő szakaszok **minden LOR-ra azonosak**!



Detektormodell

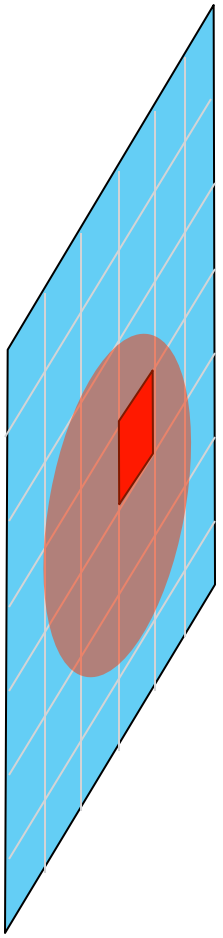
- A detektoron belüli szóródás független a méréstől (nem cseréljük ki a kristályokat)
- Annak a valószínűsége, hogy a beérkező foton a detektoron belül hogyan szóródik, csak a beesési iránytól függ
  - És a foton energiájától, de ettől most eltekintünk
- Előre leszimulálható a detektoron belüli szóródás eloszlása: „csóvák”
- A két foton szóródása független
  - Elegendő fotononként szimulálni



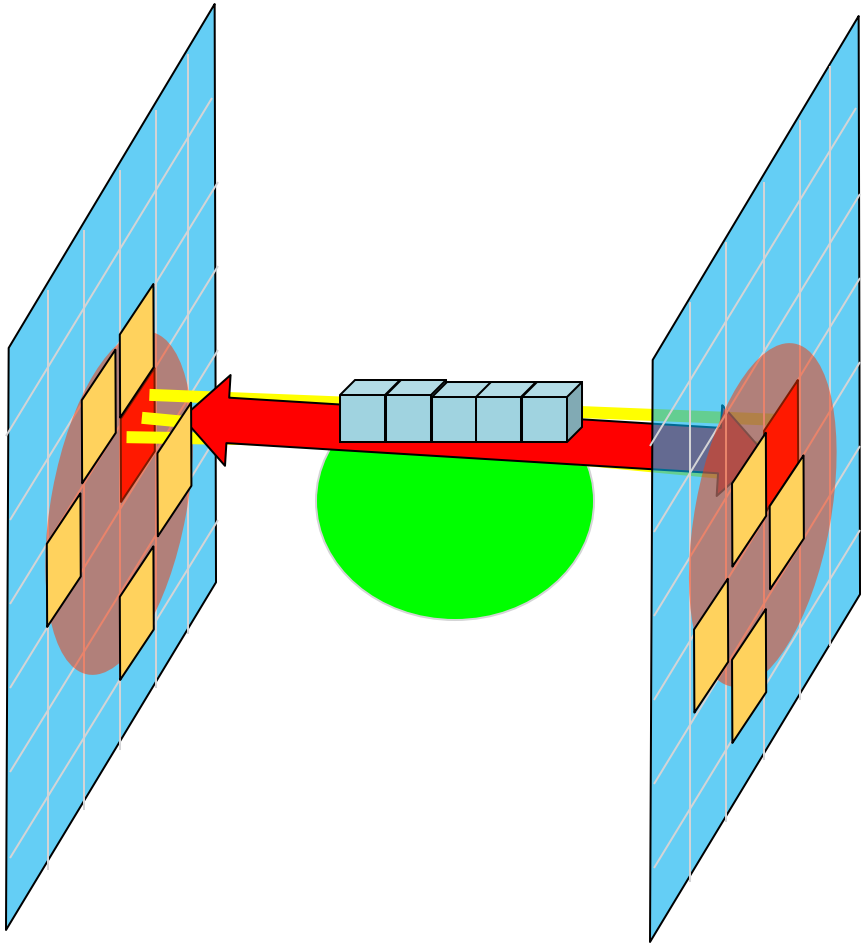


Detektormodell

- Ha egy adott detektort vizsgálunk, csak a csóván belülről, a csóva által meghatározott valószínűséggel érkezhettek fotonok



Detektormodell



1. Detektormodell nélküli előrevetítés

2. Offline (fontosság szerint) generált *index-offset* minták alapján a már kiszámolt értékek *kiolvasása, konvolúció (összeadás)*

Számítási költség elhanyagolható a ray-marchinghoz képest, így jóval több mintát is vehetünk