
**Budapesti Műszaki és Gazdaságtudományi Egyetem**
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Varnyú Dóra

# ANTIALIASING ALKALMAZÁSA GPU-ALAPÚ PET REKONSTRUKCIÓBAN

Konzulens

Dr. Szirmay-Kalos László

BUDAPEST, 2018

# Tartalomjegyzék

# Összefoglaló

A pozitronemissziós tomográfia (PET) napjaink egyik legfontosabb nukleáris gyógyászati képalkotó technikája. Erőssége más típusú orvosi képalkotó technikákhoz, mint például a röntgenhez, CT-hez vagy MRI-hez képest abban áll, hogy a szervezet anatómiai felépítése helyett annak belső működését, úgy mint a véráramlást, oxigénfelvételt és az anyagcserét méri, ezáltal lehetővé téve az orvosok számára a különböző szervek és szövetek aktuális állapotának vizsgálatát.

A PET vizsgálat során a páciensbe kis mennyiségű radioaktív anyagot, úgynevezett nyomjelzőt fecskendeznek, amely szétterjed a testben és felszívódik a szövetekbe. A kibocsátott sugárzást a PET készülék észleli, és számítógépes analízis segítségével rekonstruálja a nyomjelző háromdimenziós térbeli eloszlását. A fokozott anyagcseréjű sejtek, mint például a rákos sejtek több nyomjelzőt szívnak fel, így fényes foltként jelennek meg a rekonstruált képen. A PET képalkotás segítségével az olyan súlyos betegségek, mint a tumor, koszorúér-betegség, Alzheimer-kór, epilepszia és Parkinson-kór még a tényleges anatómiai változások bekövetkezése előtt kimutathatóak és kezelhetőek.

A képrekonstrukció több százmillió egyenlet megoldásából áll a mérési folyamat közben. Ezt a számítási intenzív feladatot a grafikus feldolgozóegység (GPU) valósítja meg, kihasználva a masszívan párhuzamos architektúráját a számítások felgyorsítására.

A rekonstrukciós folyamat során egy iteratív séma kerül alkalmazásra, minden iterációban egyre pontosabb becslést kapva a nyomjelző eloszlásáról. A rekonstrukció $\gamma$-foton párok észlelésén alapul, melyek a radioaktív anyag és a test sejtjei közti kölcsönhatás során kerülnek kibocsátásra. A pár két fotonja a páciens körül elhelyezkedő detektorgyűrű két átellenes pontjába csapódik be. Minden átellenes detektor pontpár egy Line Of Response-t (LOR-t) alkot, és a $\gamma$-foton ütközések várható száma egy kettős integrál segítségével számolható ki a LOR-ok, valamint az annak végpontjait összekötő vonalszakasz pontjai felett. Az ütközések várt és ténylegesen mért számának összehasonlításával a radioaktív nyomjelző sűrűségeloszlásának becslése korrigálható.

Dolgozatom első részében azt vizsgálom, hogy a számítógépes grafikában használt antialiased vonalhúzó algoritmusok segítségével hogyan lehet hatékonyan közelíteni a vonalintegrál értékét egy LOR mentén, illetve hogy ez milyen hatással van a rekonstrukció sebességére. A későbbi fejezetekben bemutatom a dinamikus PET rekonstrukciót, ahol a nyomjelző eloszlása időben változik. E modell szerint a mért térfogat egy pontja egy előzetesen becsült pálya mentén halad. Ha ezt a pályát megfelelően kicsi, egyenes szakaszokkal közelítjük, az antialiased vonalhúzó algoritmusok alkalmazhatóak a pont hatékony nyomon követésére a rekonstrukció során.

# Abstract

Positron emission tomography (PET) is one of today's most important nuclear medicine imaging techniques. Its strength compared to other types of medical imaging techniques, such as X-Ray, CT and MRI lies in the fact that instead of the anatomical structure of the body, it measures its functioning, like blood flow, oxygen intake and metabolism, thus enabling doctors to examine the current state of different organs and tissues.

During a PET scan, a small amount of radioactive material called radiotracer is injected into the patient, which then spreads through the body and gets absorbed by tissues. The radiation emitted by the tracer is detected by the PET scan machine, and with the help of computer analysis a three-dimensional image of tracer concentration within the body is reconstructed. Cells that have a higher metabolic rate, like cancer cells absorb more radiotracer molecules, thus show up as bright spots on the reconstructed image. With the help of PET scans, diseases like tumors, coronary artery disease, Alzheimer's disease, epilepsy and Parkinson's disease can be detected and treated even before actual anatomical changes occur.

Image reconstruction consists of solving hundreds of millions of equations. This computationally intensive procedure is realized by the graphics processing unit (GPU), exploiting its massively parallel architecture to speed up calculations.

During the reconstruction procedure, an iterative scheme is applied, getting a better estimation of the radiotracer concentration in each iteration. The reconstruction is based on the detection of γ-photon pairs emitted during the interaction between the radioactive material and the body cells. The two photons of a pair hit two opposite points on the detector ring surrounding the patient. Each pair of opposite detector points forms a Line Of Response (LOR), and the expected number of γ-photon hits can be calculated with a multi-dimensional integral over the LORs and the line segments connecting their endpoints. By comparing the expected and the measured number of hits, the estimation of the radiotracer density distribution can be corrected.

In the first part of my work, I focus on how antialiased line drawing algorithms from computer graphics can be used to efficiently estimate the line integral along a LOR, and its effect on the speed of the reconstruction procedure. In the later chapters, I will introduce motion compensation in dynamic PET reconstruction. According to this model, a point in the measured volume moves along an a-priori estimated path over time. If this path is approximated with a sequence of sufficiently small line segments, antialiased line drawing algorithms can be applied to efficiently track the point during the reconstruction procedure.

# Köszönetnyilvánítás

A dolgozatban megfogalmazott eredmények a Budapesti Műszaki és Gazdaságtudományi Egyetem Villamosmérnöki és Informatikai Kar Balatonfüredi Hallgatói Kutatócsoport szakmai közössége keretében jöttek létre a régió gazdasági fejlődésének elősegítése érdekében. Az eredmények létrehozása során figyelembe vettük a balatonfüredi központú Rendszertudományi Innovációs Klaszter által megfogalmazott célkitűzéseket, valamint a párhuzamosan megvalósuló EFOP 4.2.1-16-2017-00021 pályázat támogatásával elnyert „BME Balatonfüredi Tudáscentrum" térségfejlesztési terveit.

A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg (EFOP-3.6.2-16-2017-00013).

Szeretnék köszönetet mondani konzulensemnek, Dr. Szirmay-Kalos Lászlónak is, akinek végtelen türelmére és segítőkész válaszaira mindig számíthatok. Az ő iránymutatása nélkül ez a kutatás nem valósulhatott volna meg.

# 1 Bevezetés

Napjainkban az egészségügy és az egészségügyi berendezések fejlődése által sok vizsgálat egyre gyorsabbá, olcsóbbá és megbízhatóbbá válik. Az orvosi diagnosztikát segítő technikákon belül kiemelt szerepe van a képalkotásnak, amely lehetővé teszi a páciens belső szerveinek a vizuális megjelenítését, így más módszerekkel nem megszerezhető információt tár a szakorvos elé. A *pozitronemissziós tomográfia* (PET) egyike ezen képalkotó módszereknek; segítségével háromdimenziós képet kaphatunk a páciens testének egy adott területéről. Működése során a sejtek funkcionális jellemzőire, például a véráramlásra, oxigénfelvételre és az anyagcserére támaszkodik, aminek köszönhetően képes az olyan súlyos betegségeket, mint a tumort, koszorúér-betegséget, Alzheimer-kórt, epilepsziát és Parkinson-kórt még a tényleges anatómiai elváltozásokat megelőzően kimutatni.

A PET úgynevezett nukleáris képalkotó technika, kis mennyiségű radioaktív anyag bejuttatásával térképezi fel a páciens testét. Az érzékelt sugárzás alapján számítógépes analízis segítségével rekonstruálja a radioaktív nyomjelző háromdimenziós térbeli eloszlását. Ez egy rendkívül számításigényes folyamat, így a mérési idő egy kritikus paraméter. A számítások felgyorsítását a *grafikus feldolgozóegység* (Graphics Processing Unit, GPU) végzi. Masszívan párhuzamos architektúrájuknak köszönhetően a GPU-k mára már messze felülmúlják a hagyományos CPU-kat számítási teljesítményt és sávszélességet tekintve [1][2].

A képrekonstrukció során több millió vonalintegrál kiszámítása szükséges. Munkám első körben arra irányult, hogy a számítógépes grafikában használt antialiased vonalhúzó algoritmusok segítségével hogyan lehet hatékonyan közelíteni a vonalintegrálok értékét a rekonstrukció úgynevezett előrevetítés szakaszában. Ezt követően az antialiasingnak a dinamikus PET rekonstrukció mozgáskompenzációjában való alkalmazásával foglalkoztam.

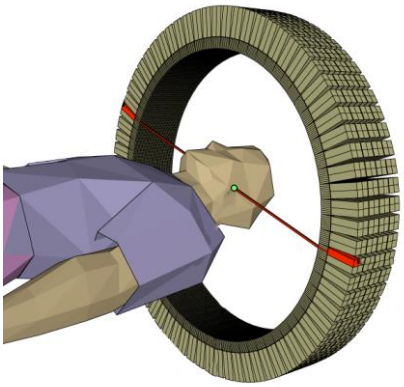
Dolgozatomban elsőként a PET rekonstrukció és az antialiasing általános ismertetését foglaltam össze. Ezután bemutatom az antialiased vonalhúzó algoritmusok használatát az előrevetítés, majd a mozgáskompenzáció során. Az eljárások hatékonyságát CPU-n és GPU-n futó tesztprogramokkal is mértem.

# 2 Pozitronemissziós Tomográfia

A PET gép belülről gyűrű alakban elhelyezkedő detektorkristályokból áll, melyek közé betolják a pácienst. A vizsgálat időtartama kb. 15-20 perc, ezalatt a páciens végig magánál van.

## 2.1 A fizikai folyamat

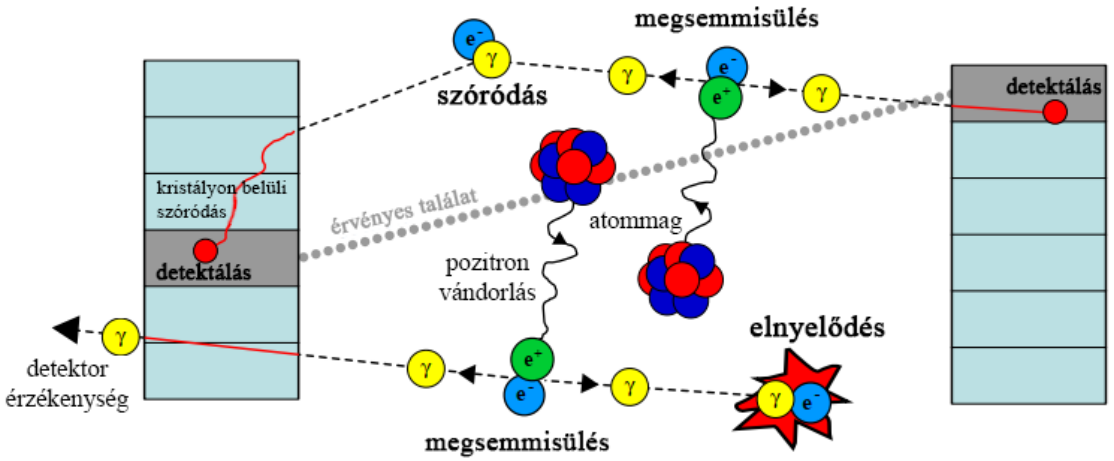
*1. ábra: Egy tipikus PET-CT gép. Kép forrása:* [59].



*2. ábra: A detektorgyűrűk elhelyezkedése. Kép forrása:* [58].

A mérés során a páciens vérkeringésébe *radioaktív nyomjelző anyagot* juttatnak. A nyomjelző rövid felezési idejű (2-110 perc [3]) izotópokból áll, melyet tipikusan az anyagcseréhez nélkülözhetetlen molekulákba, például oxigénbe vagy glükózba építenek be speciális kémiai eljárással. A kapott radioaktív molekulákat ezt követően a vér a magas biológiai aktivitású szövetekhez szállítja. A leggyakrabban alkalmazott nyomjelző anyag a *18F-fluor-dezoxi-glükóz* (röviden FDG; glükóz molekulába épített fluor) [4], mely a fokozott glükóz-felhasználású szövetekben, mint például az agyban, a szívizomzatban, vagy rosszindulatú tumorokban halmozódik fel, azok aktivitásával arányos mértékben. A képalkotás során a feladat ezen nyomjelző anyag térbeli eloszlásának rekonstruálása.

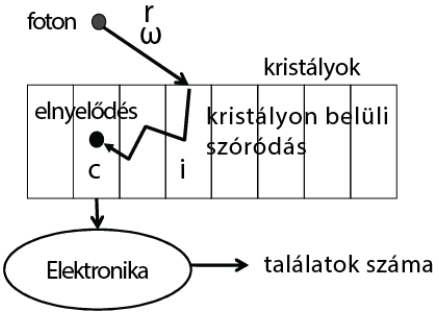
A radioaktív izotóp a bomlása során egy pozitront bocsát ki, mely a szövetben vándorolva rövid időn belül nekiütközik egy elektronnak. Az interakció során mind az elektron, mind a pozitron megsemmisül, kibocsátva egy ellentétes irányban haladó, egyenként 511 keV energiájú *γ-foton párt*. A fotonok jelentős része ezt követően további interakció nélkül, egyenes pályán haladva elhagyja a testet, és nekiütközik a páciens körül elhelyezkedő detektorgyűrűnek. Amennyiben két foton egy időben (legfeljebb pár nanoszekundum különbséggel) és ellentétes oldalon ütközik a detektorgyűrűnek, úgy *érvényes találatot regisztrálunk*. A nem párban érkező fotonokat figyelmen kívül hagyjuk. A találatok száma a fotonok által aktivált két detektorkristályt összekötő vonalban, más néven *válaszvonalban* (Line Of Response, LOR) végbemenő pozitron-elektron megsemmisülések számával arányos. Az így kapott találatszámot direkt komponensnek is nevezik.




*3. ábra: A fizikai folyamat modellje.*

A fotonok útjuk során a körülvevő közeg elektronjaival is kölcsönhatásba léphetnek, melynek következtében *elnyelődnek* vagy *szétszóródnak*. Az előbbi esetén a detektált fotonok száma csökken, melynek következtében fennáll a veszélye, hogy alulbecsüljük a szövetben felhalmozódott nyomjelző tényleges mennyiségét. Szóródás esetén a foton elveszíti energiájának egy részét, és irányt változtat, így pályája egy egyenes helyett akár tetszőleges számú és irányú szakaszokból is állhat. Ez azt eredményezi, hogy a detektált találatot kiváltó megsemmisülés a mért alanyon belül akárhol végbemehetett, akár a két detektorkristály által meghatározott LOR-on kívül is. Emberi páciens esetén a fotonok hozzávetőleg 30-50%-a szóródik legalább egyszer a detektálást megelőzően, a berendezés geometriájától és az alany méretétől függően [5]. Ebből is látszik, hogy az elnyelődési és a szóródási modell humán PET esetén kulcsfontosságú szerepet játszik. Kisállat PET esetén a foton-elektron kölcsönhatás valószínűsége jelentősen kisebb.

A PET gépek többsége *szcintillációs detektorrendszert* tartalmaz. A szcintillátor kristályok a beérkező ɣ-fotonok energiájából látható fényt (felvillanásokat) állítanak elő, amelyből *fotoelektron-sokszorozók* (photomultiplier tube, PMT) elektromos jelet generálnak. Ezekből a jelekből tudjuk azonosítani a találatokat. Azonban az aktivált szcintillációs kristály nem feltétlenül az, amelyikbe a foton eredetileg becsapódott, mivel a detektorkristályokon belül is megfigyelhető a szóródás (4. ábra). Az is előfordulhat, hogy a foton nem nyelődik el a detektorban, hanem észrevétlenül elhagyja a rendszert. Továbbá szem előtt kell tartani, hogy a szcintillátor kristályok egy becsapódás érzékelése után csak egy rövid holtidő elteltével képesek újabb becsapódást érzékelni. A rekonstrukció során mindezen szempontokat figyelembe kell venni, melynek pontos kivitelezése túlmutat a dolgozat keretein.




*4. ábra: A detektorrendszer működése.*

## 2.2 Alkalmazási területek

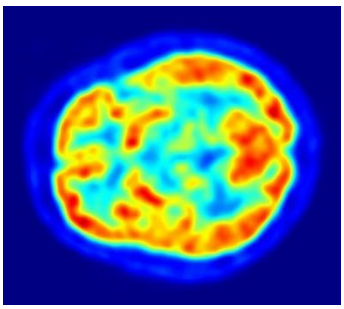
A továbbiakban a PET néhány legfontosabb felhasználási területét szeretném kiemelni.

### Onkológia

Napjainkban a tumorok és daganatos megbetegedések képalkotása során a legelterjedtebb eljárás az FDG PET/CT vizsgálat, mely 18F-fluor-dezoxi-glükóz (FDG) nyomjelzővel végrehajtott PET mérésből és ezzel egyidejű CT felvételből áll. A PET kép megjeleníti a magas glükózfelvételű tumorszövetek elhelyezkedését és aktivitását, míg a CT felvételről anatómiai információt nyerünk a kérdéses területről. Metasztázis (áttét) keresésére is rendszeresen alkalmazzák.

### Neurológia

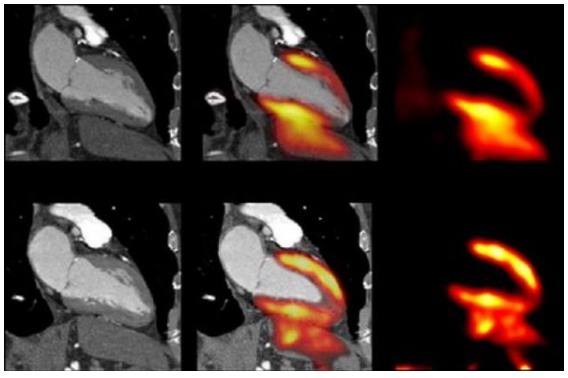
PET méréssel az agy egyes területeinek aktivitása is vizsgálható. Segítségével nyomon követhetjük az agy területeinek oxigénfelvételét és glükóz-anyagcseréjét, melynek köszönhetően többek között az Alzheimer-kór és a Parkinson-kór is már az egészen korai fázisuktól kezdve kimutatható. Emellett alkalmazzák epilepsziás betegek műtéte előtt az epilepsziás fókusz beazonosítására is.



*5. ábra: PET felvétel az agyról. Kép forrása:* [6].

### Kardiológia

A klinikai kardiológiában a szívizomszövet életképességét FDG PET/CT-vel vizsgálják. Segítségével megállapítható, hogy a hibernáló szívizomszövetet már maradandó károsodás érte-e, vagy még egy koszorúérműtéttel funkcionalitása helyreállítható lenne. Amennyiben a szövet felveszi az FDG-t, úgy a műtét sikerrel járhat.



*6. ábra: PET/CT felvételek a szívizomszövetről. Kép forrása:* [7].

**Farmakológia**

A preklinikai vizsgálatok során az új hatóanyagokat radioaktív jelöléssel látják el, és így adják be a kísérleti állatoknak. A gyógyszer felszívódásának, majd a szövetekben történő felhalmozódásának vizsgálata PET-tel sokkal gyorsabb, költséghatékonyabb és humánusabb megfigyelést tesz lehetővé, mint korábban a kísérleti állatok leölése és felboncolása.

Patkányok számára olyan miniatűr, fejre húzható PET berendezést is kifejlesztettek, melynek segítségével az állat a vizsgálat során szabadon mozoghat, még altatásra sincsen szükség [8]. A készüléket elsősorban tudományos és gyógyszerészeti célokra alkalmazzák, ritkább esetekben állategészségügyben.

## 2.3 Előnyök és hátrányok

**Előnyök:**

- A PET a biokémiai funkciók vizsgálata révén képes még a strukturális elváltozások előtt kimutatni a betegségeket, ellentétben a szervezet anatómiai felépítését vizsgáló képalkotó eljárásokkal, mint például a röntgennel, CT-vel vagy MRI-vel.
- Az anyagcsere-funkciók tanulmányozása által a PET képalkotás a biopszia és egyéb feltáró műtétek alternatívájaként használható különböző betegségek terjedésének megállapítására.
- Segítségével a jóindulatú (nem rákos) és rosszindulatú (rákos) daganatok nagyobb bizonyossággal megkülönböztethetőek, így csökkenthető a helytelen diagnózis okozta szükségtelen műtétek száma.
- Alkalmas különböző neurológiai megbetegedések, például epilepszia, Alzheimer-kór, Parkinson-kór és más demenciák diagnosztizálására már a korai stádiumukban.
- A PET vizsgálatnál nem áll fenn fertőzésveszély.
- A PET vizsgálat során használt radioaktív izotópok rövid felezési ideje miatt a pácienst más tomográf vizsgálatokhoz képest minimális sugárterhelés éri.

**Hátrányok:**

- A sugárterhelés alacsony volta ellenére sem alkalmazható terhesség esetén.
- A radioaktív izotópok rövid felezési ideje miatt azokat közvetlenül a vizsgálat előtt, a helyszínen vagy annak közelében kell előállítani, mely emeli a vizsgálat költségét.
- Ugyancsak a vizsgálat költségét növeli, hogy az izotópok szintetizálása a nyomjelző molekulákba speciális kémiai berendezést igényel, melyet közvetlenül az izotóp előállítása után kell alkalmazni, mielőtt még az lebomlana.
- A PET képek önmagukban nem hordoznak anatómiai információt, emiatt leggyakrabban más képalkotó eljárásokkal, például CT-vel vagy MRI-vel kiegészítve szokták alkalmazni.

# 3 PET rekonstrukció

A PET képalkotás során a sugárzás térbeli eloszlását rekonstruáljuk véges adathalmazon. A legegyszerűbb esetben a tértartományt $N_V$ darab homogén voxelre bontjuk fel. Ekkor a vizsgált tértartomány egy adott $\vec{v}$ pontjában az aktivitás leírható az egyes voxelekhez tartozó, $N_V$ darab bázisfüggvény segítségével. Formálisan az $x(\vec{v})$ pozitron kibocsátási sűrűségre vagyunk kíváncsiak véges függvénysorozat alakban:

$$x(\vec{v}) = \sum_{V=1}^{N_V} x_V \cdot b_V(\vec{v}),$$

*1. egyenlet*

ahol $\boldsymbol{x} = (x_1, x_2, \dots, x_{N_V})$ ismeretlen együtthatók és $b_V(\vec{v})$ $(V = 1, \dots, N_V)$ az egyes voxelekhez tartozó bázisfüggvények. Feltehető, hogy minden $x(\vec{v})$, $x_i$ és $b_i(\vec{v})$ nemnegatív. Dinamikus PET rekonstrukció esetén a sugárzási sűrűség az időnek is függvénye.

## 3.1 Kinetikus modellek

Feltételezésünk szerint a radioaktív nyomjelző koncentrációjának időfüggése leírható egy általános $C(\boldsymbol{p}_V, t)$ *kinetikus modellel*, melyben $\boldsymbol{p}_V$ *kinetikus paraméter* egy, a térfüggő tulajdonságokat jellemző, $N_P$ dimenziós vektor. ($N_P$, azaz a paraméterek száma általában kevesebb, mint tíz.) Az alkalmazott kinetikus modellre több alternatíva is fellelhető az irodalomban, úgy mint a spektrális módszer [9], a Patlak módszer [10], a relatív egyensúly diagram (Relative Equilibrium Plot) [11], valamint az egyszövetes és kétszövetes kompartment modell [12]. A rekonstrukciós folyamat elsődleges kimenete a paramétervektor minden egyes voxelhez, amelyekből felrajzolható az idő–pozitron kibocsátási sűrűség függvény.

## 3.2 Dinamikus PET rekonstrukciós folyamat

A PET gép a detektorkristály-párokon bekövetkező egyidejű fotonbecsapódások *eseményeit* gyűjti. Az esemény a detektorpár azonosítójából (Line Of Response, LOR), illetve az észlelés időpontjából áll. A rekonstrukció során a detektált események listájából indulunk ki. A mérés időtartamát $N_T$ véges tartományra bontjuk $t_1, t_2, \dots, t_{N_T}$ határokkal, és az egyes eseményeket a megfelelő $(t_T, t_{T+1})$ időkeretekbe soroljuk. Az egyes időkereteken belül az események bekövetkezésének pontos idejét, az események sorrendjét nem különböztetjük meg. Ennek az egyszerűsítésnek a következtében a rekonstrukció időbeli bonyolultsága csupán az időkeretek számával lesz arányos.

A $C(\boldsymbol{p}_V, t)$ sűrűségfüggvényt felhasználva a $V$ voxelben bekövetkező radioaktív bomlások várható száma a T időkeretben, azaz a $[t_T, t_{T+1}]$ időintervallumban:

$$\tilde{x}_T(\boldsymbol{p}_V) = \int_{t_T}^{t_{T+1}} C(\boldsymbol{p}_V, t)\, e^{-\lambda t} \mathrm{d}t,$$

ahol λ a radioaktív izotóp bomlásállandója.

A pozitron létrejövetele és a ɣ-fotonok detektálása közötti összefüggést az $\tau(\vec{v} \to L)$ *detektorérzékenység* fejezi ki, mely annak a valószínűségét adja meg, hogy a $\vec{v}$ pontban kibocsátott pozitront az *L* LOR-ban detektáljuk. Az *L* LOR-ban detektált fotonok várható száma a $T$ időkeretben ($\tilde{y}_{L,T}$) megegyezik a rekonstruálandó tértartomány (V) összes pontjának a hozzájárulásának az összegével:

$$\tilde{y}_{L,T} = \int_{\mathrm{V}} x_T(\vec{v})\, \tau(\vec{v} \to L) \mathrm{d}v.$$

$x_T(\vec{v})$-nek az 1. egyenletben szereplő véges függvénysorozat alakját kihasználva:

$$\tilde{y}_{L,T} = \sum_{V=1}^{N_V} \boldsymbol{A}_{L,V}\, x_{V,T},$$

ahol

$$\boldsymbol{A}_{L,V} = \int_{\mathrm{V}} b_V(\vec{v})\, \tau(\vec{v} \to L) \mathrm{d}v.$$

$\boldsymbol{A}$ az úgynevezett *rendszermátrix*, melynek tetszőleges $\boldsymbol{A}_{L,V}$ eleme annak a valószínűségét adja meg, hogy a $V$ voxelben létrejött pozitront az $L$ LOR-ban detektáljuk. Ezt előzetes méréssel [13], Monte Carlo becsléssel [14] vagy menet közbeni számolással [15] lehet meghatározni. A detektortalálatok várható értékének számítását *előrevetítésnek* nevezzük.

Az ismeretlen $\boldsymbol{p}_V$ paramétervektorokat úgy keressük, hogy a ténylegesen mért adatok valószínűsége a lehető legnagyobb legyen. Az *L* LOR-ban a $T$ időkeretben mért találatok száma $\tilde{y}_{L,T}$ várható értékű Poisson eloszlást követ. Mivel a különböző LOR-ok és időkeretek egymástól statisztikailag függetlenek, az összes LOR-t és az összes időkeretet figyelembe vevő kombinált valószínűség az elemi valószínűségek szorzataként áll elő. Az ismeretlen paraméterek az úgynevezett *maximum likelihood* statisztikai eljárással [16][17] kerülnek meghatározásra. A módszerrel a következő *log-likelihood függvényt* maximalizáljuk:

$$\log L = \sum_{L=1}^{N_L} \sum_{T=1}^{N_T} (y_{L,T} \log \tilde{y}_{L,T} - \tilde{y}_{L,T}).$$

*2. egyenlet*

A maximum likelihoodon alapuló dinamikus rekonstrukciót kétfázisú iterációs sémával valósítjuk meg. Minden iteráció előrevetítésekből és visszavetítésekből áll. Az előrevetítés során a detektortalálatok várható értékét számítjuk ki a radioaktív bomlások számának várható

értékéből, míg a visszavetítés során a bomlások eloszlásának egy új becslését határozzuk meg a detektortalálatok várt és tényleges értékének arányával korrigálva.

A rekonstrukciós folyamat a 2. egyenlet log-likelihood függvényének maximalizálását jelenti. A likelihood ott veszi fel a maximumát, ahol minden parciális deriváltja nulla. A parciális deriváltakat felírva a következő összefüggést kapjuk:

$$\sum_T \frac{\partial \tilde{x}_T(\boldsymbol{p}_V)}{\partial \boldsymbol{p}_{V,P}} \sum_L \left( \boldsymbol{A}_{L,V} \frac{y_{L,T}}{\tilde{y}_{L,T}} - \boldsymbol{A}_{L,V} \right) = 0,$$

*3. egyenlet*

ahol $V = 1,2,\dots,N_V$ és $P = 1,2,\dots,N_P$. Ebből következően $N_V \times N_P$ egyenletünk van, ezek mindegyike $N_T$ tagból áll, amelyek mindegyike az összes voxel ismeretlen paramétereitől függ. Kiszámításukhoz figyelembe kell vennünk az összes $L$ LOR-t, amelyben $\boldsymbol{A}_{L,V}$ nem nulla. A pontos rekonstrukcióhoz emellett szükség van a pozitron vándorlás és a szóródott részecskék útjának figyelembe vételére is, amelyek a rendszermátrixot sűrűvé teszik.

A log-likelihood deriváltjainak kiszámításához minden T időkeretben egy-egy statikus előrevetítés és visszavetítés végrehajtására van szükség. A visszavetítés az aktivitás eloszlás egy új becslését állítja elő:

$$x_{V,T} = \tilde{x}_T(\boldsymbol{p}_V) \cdot \frac{\sum_L \boldsymbol{A}_{L,V} \frac{y_{L,T}}{\tilde{y}_{L,T}}}{\sum_L \boldsymbol{A}_{L,V}}.$$

A fenti egyenletből kifejezhető, hogy

$$\sum_L \boldsymbol{A}_{L,V} \frac{y_{L,T}}{\tilde{y}_{L,T}} = \frac{x_{V,T}}{\tilde{x}_T(\boldsymbol{p}_V)} \sum_L \boldsymbol{A}_{L,V},$$

melyet a 3. egyenletbe behelyettesítve a következőt kapjuk:

$$\sum_T \frac{\partial \tilde{x}_T(\boldsymbol{p}_V)}{\partial \boldsymbol{p}_{V,P}} \left( \sum_L \boldsymbol{A}_{L,V} \right) \left( \frac{x_{V,T}}{\tilde{x}_T(\boldsymbol{p}_V)} - 1 \right) = 0.$$

Mindkét oldalt elosztva a voxelek érzékenységével − azaz $\sum_L \boldsymbol{A}_{L,V}$-vel − megkapjuk a likelihood maximumának szükséges feltételét:

$$\sum_T \frac{\partial \tilde{x}_T(\boldsymbol{p}_V)}{\partial \boldsymbol{p}_{V,P}} \left( \frac{x_{V,T}}{\tilde{x}_T(\boldsymbol{p}_V)} - 1 \right) = 0.$$

*4. egyenlet*

Ebben az egyenletben $\tilde{x}_T(\boldsymbol{p}_V)$ a $V$ voxel ismeretlen $\boldsymbol{p}_V$ paramétervektorától függ, míg az $x_{V,T}$ számításához egy előrevetítés és egy visszavetítés szükséges, és az összes voxel paraméterhalmazától függ. Ennek megfelelően amennyiben $x_{V,T}$ ismert, akkor a számítások

szétbonthatóak az egyes voxelekre úgy, hogy minden részszámításban egy $N_P$ ismeretlent tartalmazó egyenletrendszert kell megoldanunk. Ezzel a megközelítéssel az előrevetítés és a visszavetítés elválasztható az ismeretlen paraméterek számításától, így az algoritmus számítási bonyolultsága a két lépés bonyolultságának összege és nem a szorzata lesz. Rögzített $x_{V,T}$ esetén a nemlineáris egyenlet egy görbeillesztési problémaként is értelmezhető és megoldható.

A maximum likelihood módszer hátránya, hogy a célfüggvény maximalizációjának kényszerítése sokszor drasztikus oszcillációkat tartalmazó megoldást eredményez, ezért *regularizációra* van szükség. Az egyik lehetőség erre egy *regularizációs tag* hozzáadása az optimalizációs célfüggvényhez, ami bünteti a nem megfelelő megoldásokat, azaz például azokat az eseteket, ahol a szórás túl nagy [18]. Egy másik lehetséges megközelítés a sziták módszerének alkalmazása, mely az iterált becslést szűri minden lépést követően [19].

Az eddig ismertetett lépéseket összefűzve a rekonstrukció pszeudokódja:

**for** $n = 1$-től $n_{max}$-ig // fő iteráció
  **for** $T = 1$-től $N_T$-ig // időkeretek iterációja
    **for minden** $V$ **voxelre párhuzamosan** // kiértékelés: bomlások száma
      $\tilde{x}_T(\boldsymbol{p}_V) = \int_{t_T}^{t_{T+1}} C(\boldsymbol{p}_V, t)\, e^{-\lambda t} \mathrm{d}t$
    **for minden** $L$ **LOR-ra párhuzamosan** // előrevetítés: a LOR-okban találatszáma
      $\tilde{y}_L = \sum_{V'} \boldsymbol{A}_{L,V'} \tilde{x}_{V',T}$
    **for minden** $V$ **voxelre párhuzamosan** // visszavetítés: korrekció
      $x_{V,T} = \tilde{x}_T(\boldsymbol{p}_V) \cdot \frac{\sum_L \boldsymbol{A}_{L,V} \frac{y_{L,T}}{\tilde{y}_{L,T}}}{\sum_L \boldsymbol{A}_{L,V}}$
    **for minden** $V$ **voxelre párhuzamosan** // szűrés: sziták módszere
      $x_{V,T} = \text{Szűrő}(x_{V,T})$
  **endfor**
  **for minden** $V$ **voxelre párhuzamosan** // görbeillesztés → kinetikus paraméterek
    $\boldsymbol{p}_V =$ a 3.4 egyenlet megoldása
  **endfor**
**endfor**

Ennek az optimalizációs problémának a megoldása nagyon nagy számításigényű. A szabad változók száma, azaz a keresési tér dimenziója $N_P \times N_V$. A log-likelihood, mely az optimalizáció célfüggvénye, $N_L \times N_T$ tag összegeként áll elő. Az $N_V$ és $N_L$ nagysága tipikusan több százmillió, az $N_T$ százas nagyságrendű, végül az $N_P$ kevesebb, mint tíz, hiszen az időfüggvényt mindössze néhány paraméter segítségével szeretnénk leírni. Ezekből következően mind a keresési tér, mind az optimalizáció célfüggvényének tagjainak száma milliós nagyságrendű lehet. A rekonstrukció leginkább számításigényes lépései az előrevetítés és a visszavetítés, melyeket minden iterációban, minden egyes időkeretben végre kell hajtani. Jól látható, hogy a számítások elvégzése elképzelhetetlen nagyteljesítményű rendszerek és körültekintően kiválasztott algoritmusok használata nélkül.

## 3.3 A technológia jelen állása

A technológia jelenlegi állását és a dinamikus PET kinetikus paramétereinek becslésére vonatkozó korábbi munkákat a [11] és [20] cikkek tekintik át.

Ha a kinetikus modell nemnegatív paraméterek lineáris függvénye, mint ahogy a spektrális, a Patlak és a Logan modellek esetében van, akkor a klasszikus maximum likelihood számítási séma kiegészíthető az összes paraméter megtalálására [21]. A biológiailag helytálló kinetikai modellek, például a kompartment modellek, gyakran függenek nemlineárisan a paramétereiktől, melynek köszönhetően a klasszikus maximum likelihood megközelítés nem alkalmazható. A kétszövetes kompartment modell paramétereinek régiónkénti rekonstrukciójára látott egyik megoldás egy büntetett likelihood maximumát kereste iterált koordinátacsökkentéssel [22]. Az egyszövetes kompartment modellt a [23] cikkben vizsgálták. Wang és társai [24] az optimalizációs átvitel elvének alkalmazását javasolták a megfelelő kinetikai modell megtalálására, amely lokálisan becsüli az optimalizáció célfüggvényét.

A tomográf rekonstrukció számára a nagyteljesítményű rendszerek, mint például az FPGA-k [25], multi-CPU rendszerek [26], a CELL processzor [27] vagy a GPU-k [28] közül a masszívan párhuzamos GPU bizonyult a számítási költséget tekintve a leghatékonyabb platformnak [29]. A GPU-k két különböző modell alapján programozhatóak. Az OpenGL/GLSL-hez vagy a Direct3D/HLSL-hez hasonló shader API-k a GPU hardverét a grafikus csővezeték közvetlen megvalósításaként mutatják be [30], beleértve a programozható és a fix feldolgozási szakaszokat is. Ezzel ellentétben a CUDA [31], a StreamSDK [32] és az OpenCL [33] hozzáférést biztosít a GPU multiprocesszoraihoz, ahol minden egyes multiprocesszor úgynevezett warpokba rendezett skalárprocesszorokat tartalmaz, melyek SIMD (Single Instruction, Multiple Data) elven működnek.

Már korábbi munkákban is volt rá példa, hogy a geometriai projekció, illetve a szóródás mint Gauss-elmosás implementációját a GPU-n valósították meg. Ameddig csak a geometriai hatásokat vesszük figyelembe, a shader API-ok megfelelnek a célra [28][34], mivel a rajtuk keresztül elérhető raszterező és alfa összemosó egységek támogatják ezeket az egyszerű számításokat. A csővezeték korlátozottsága miatt viszont többek között a fizikailag helytálló szóródási korrekciót is egyszerű elmosással kell helyettesíteni [35]. Ha azonban komplexebb algoritmusokat szeretnénk megvalósítani, a shader processzorok feletti irányítás előnyei felülmúlják a csővezeték fix működési elemeinek, például a vágásnak, a raszterizációnak, a mélységtesztelésnek és az alfa összemosásnak a kihasználását. Ebből kifolyólag a GPU programomat a CUDA platformon implementáltam.

# 4 Antialiasing

Modellezés során a világot leíró folytonos jelet számítógépes feldolgozásra alkalmas, diszkrét jellé alakítjuk. Az analóg-digitális konverzió alatt a jel mintavételezésen és kvantáláson esik át. A Shannon–Nyquist-féle mintavételezési tétel alapján ha a mintavételezési frekvencia az eredeti jel frekvenciájának több, mint kétszerese, akkor a jel maradéktalanul rekonstruálható.

A való világban azonban az objektumok hirtelen, egységugrás függvényhez hasonlóan jelennek meg, így ilyenkor a frekvenciájuk végtelen nagynak tekinthető. Ennek következtében a mintavételezési tétel sosem teljesíthető be. *Aliasingról* akkor beszélünk, amikor a nem kielégítő mintavételezés következtében az alacsony frekvenciatartományban magas frekvenciájú komponensek jelennek meg. Ennek eredményei a számítógépes grafikában a jól ismert pixeles, lépcsőfokszerű vonalhatárok, melyeket a magas frekvenciájú komponensek nem megfelelő kiszűrése okozza. A helyzet a pixelek méretével összehasonlítható nagyságú objektumok, mint például textúrák számára még rosszabb, mivel azok kedvezőtlen esetben akár teljesen el is tűnhetnek.

Az aliasing hatásának csökkentése *előszűréssel* vagy *utószűréssel* lehetséges.

Előszűrés esetén a mintavételezés előtt aluláteresztő szűrőt alkalmazunk, így a mintavételezést már sávhatárolt, magas frekvenciájú komponenseket nem tartalmazó jelen végezzük. Ideális aluláteresztő szűrővel az aliasing kialakulása teljes mértékben elkerülhető. A gyakorlatban azonban a számítási hatékonyság figyelembe vétele miatt az ideális szűrőnek csak különböző közelítéseit alkalmazzák.

Utószűrés esetén a szűrés a mintavételezés után történik. Miután ekkor már megjelent az aliasing, ennek hatását teljesen eltüntetni nem, csak csökkenteni lehet. Az utószűrés nagyobb frekvenciájú mintavétellel, úgynevezett *supersamplinggal* társul. Ennek során a kép a megjelenítéshez szükségesnél nagyobb felbontással áll elő, melyet kifinomult digitális szűrők csökkentenek a kívánt felbontásra.

A PET rekonstrukció során a cél a minél tisztább, zajmentes kép előállítása. Az utószűrés, jóllehet csak utólagosan csökkenteni tudja az aliasing hatását, mégis alkalmazható a rekonstruált kép minőségének javítására [36]. Az aliasing megelőzésének lehetősége miatt azonban az előszűrés nagyobb jelentőséggel bír, így a munkám során elsődlegesen erre a területre koncentráltam. A fejezet további részében a képek előszűrésének matematikai hátterét és az általam használt szűrőtípusokat mutatok be.

## 4.1 Aluláteresztő szűrők

A szűrőt kezdetben az idő- vagy a frekvenciatartományban definiáljuk. Ezt követően alapvetően kétféleképpen lehet a szűrést megvalósítani: konvolúcióval az időtartományban vagy szorzással a frekvenciatartományban.

Frekvenciatartományban a szűrés a szűrő függvénnyel való szorzást jelenti:

$$I_f^*(\alpha,\beta) = I^*(\alpha,\beta) \cdot F(\alpha,\beta),$$

ahol $I^*(\alpha,\beta)$ az eredeti jel, $I_f^*(\alpha,\beta)$ a szűrt jel és $F(\alpha,\beta)$ a szűrő függvény.

Az időtartományban a szűrt jelet az eredeti jel és (az időtartományban megadott) szűrő függvény konvolúciójaként lehet előállítani:

$$I_f(x,y) = I(x,y) * f(x,y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(t,\tau)\cdot f(x-t,y-\tau)\,\mathrm{d}t\mathrm{d}\tau.$$

Amennyiben a szűrő függvény a frekvenciatartományban adott, úgy $f(x,y)$-t belőle inverz Fourier-transzformációval kapjuk meg ($f(x,y)$ az $F(\alpha,\beta)$ impulzusválasza).

Mivel az eredeti kép az időtartományban van megadva és a szűrt képet is abban várjuk, ezért ha a szűrést a frekvenciatartományban szeretnénk elvégezni, először át kell transzformálni a képet a frekvenciatartományba, majd a szűrés után visszatranszformálni azt az időtartományba. A két Fourier-transzformáció számítási terhe még az olyan hatékony algoritmusok, mint a gyors Fourier-transzformáció (Fast Fourier Transformation, FFT) [37] használata esetén is jelentős, így a frekvenciatartománybeli szűrést csak néhány speciális esetben alkalmazzák.

### 4.1.1 Ideális szűrő

A mintavételezési tétel alapján egy $\Delta x, \Delta y$ periódusidővel mintavételezett 2D jel Nyquist határfrekvenciája [38] $\frac{\pi}{\Delta x}$, illetve $\frac{\pi}{\Delta y}$. Az aliasing elkerülése érdekében olyan aluláteresztő szűrőre van szükség, mely kiszűr minden, a határfrekvencia feletti komponenst, de az az alattiakat érintetlenül hagyja. Az ideális szűrő a frekvenciatartományban a következő:

$$F(\alpha,\beta) = \begin{cases} 1, & ha\ |\alpha| < \dfrac{\pi}{\Delta x}\ \acute{e}s\ |\beta| < \dfrac{\pi}{\Delta y} \\ 0, & egyébként \end{cases}$$

Mivel a szűrést az időtartományban szeretnénk elvégezni, nem pedig a frekvenciatartományban, ezért a szűrőnek az impulzusválaszára vagyunk kíváncsiak. Ez a jól ismert $sinc$ függvényen alapul:

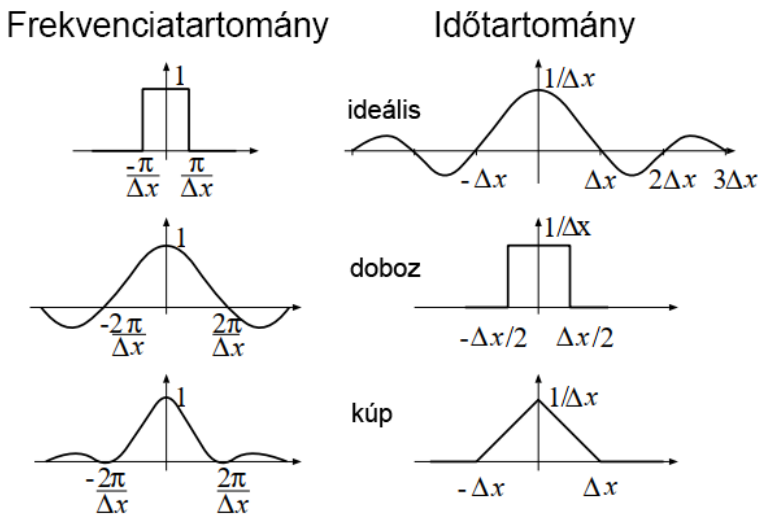
$$f(x,y) = \frac{\sin\left(x\cdot\frac{\pi}{\Delta x}\right)}{x\cdot\frac{\pi}{\Delta x}} \cdot \frac{\sin\left(y\cdot\frac{\pi}{\Delta y}\right)}{y\cdot\frac{\pi}{\Delta y}} = \mathrm{sinc}\left(x\cdot\frac{\pi}{\Delta x}\right)\cdot\mathrm{sinc}\left(y\cdot\frac{\pi}{\Delta y}\right).$$

A szűrés megvalósítása egy kétdimenziós $sinc$ függvénnyel való konvolválással komoly hátrányokkal jár. A $sinc$ függvény értéke lassan csökken, így egyetlen pontban a szűrt kép színére az eredeti kép nagy része hatással van, ezáltal a szűrés bonyolulttá válik. Ezenkívül a $sinc$ függvénynek negatív részei is vannak, ami negatív színeket eredményezhet. Az eredeti kép éles színátmenetei észrevehető oszcillációt, Gibbs-effektust okoznak a szűrt képen. Ezen problémák leküzdése érdekében a $sinc$ függvényt véges tartományra korlátozott (*Finite*

*Impulse Response, FIR*), pozitív, folytonos függvényekkel becsüljük. Ezekkel a szűrőkkel szemben elvárás, hogy az $\alpha = 0, \beta = 0$ frekvenciák esetén értékük 1 legyen, így az impulzusválaszuk integráljának is 1-nek kell lennie:

$$F(0,0) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y) = 1.$$

Munkám során a következő FIR szűrőkkel dolgoztam:




*7. ábra: Az ideális és az ezt közelítő FIR szűrők időbeli és frekvenciabeli viselkedése egydimenzióban.*

### 4.1.2 Dobozszűrő

Az időtartományban:

$$f(x,y) = \begin{cases} \dfrac{1}{\Delta x \cdot \Delta y}, & \text{ha } |x| < \dfrac{\Delta x}{2} \text{ és } |y| < \dfrac{\Delta y}{2} \\ 0, & \text{egyébként} \end{cases}$$

A frekvenciatartományban a dobozszűrő egy $sinc$ függvény, ami az ideális aluláteresztő szűrőnek egyáltalán nem pontos közelítése.

### 4.1.3 Kúpszűrő

Az időtartományban, a $(0,0)$ ponttól való normalizált távolságot $r$-rel jelölve, ahol $r(x,y) = \sqrt{\left(\frac{x}{\Delta x}\right)^2 + \left(\frac{y}{\Delta y}\right)^2}$:

$$f(x,y) = \begin{cases} (1-r)\cdot\frac{3}{\pi}, & \text{ha } r < 1 \\ 0, & \text{egyébként} \end{cases}$$

A $\frac{3}{\pi}$ együttható garantálja, hogy a kúp teljes térfogata, azaz az impulzusválasz integrálja 1 legyen. Az impulzusválasz Fourier-transzformáltja egy $sinc^2$ alakú függvény, amely a dobozszűrőnél jobban közelíti az ideális függvényt.

### 4.1.4 Hengerszűrő

Az időtartományban, $r(x,y) = \sqrt{\left(\frac{x}{\Delta x}\right)^2 + \left(\frac{y}{\Delta y}\right)^2}$ távolságparaméter bevezetésével:

$$f(x,y) = \begin{cases} 1\cdot\frac{4}{\pi}, & \text{ha } r < 1 \\ 0, & \text{egyébként} \end{cases}$$

A konstans $\frac{4}{\pi}$ szorzó miatt az integrál a teljes tartományon 1. Az impulzusválasz Fourier-transzformáltja a dobozszűrőhöz hasonlóan egy $sinc$ függvény, ami az ideális aluláteresztő szűrőnek egyáltalán nem pontos közelítése.

## 4.2 Képek előszűrése

Előszűrés esetén a képet a szűrés után mintavételezzük, a képernyő felbontásának megfelelő mintavételi rátával ($\Delta x = 1, \Delta y = 1$). A mintavételezés Dirac-fésűvel, másnéven Shah-függvénnyel [39] valósítható meg. A szűrést az időtartományban elvégezve a szűrt és mintavételezett jel a következő:

$$I_{sf}(x,y) = [I(x,y) * f(x,y)] \cdot \sum_i \sum_j \delta(x-i, y-j).$$

Egy tetszőleges, X, Y egész koordinátákkal megadott pixelre:

$$I_{sf}(X,Y) = [I(x,y) * f(x,y)] \cdot \delta(x-X, y-Y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} I(t,\tau)\cdot f(X-t, Y-\tau)\,\mathrm{d}t\mathrm{d}\tau.$$

FIR szűrők esetén a fenti végtelen integráltartományt véges intervallumra lehet cserélni.

Dobozszűrő esetén:

$$I_{s,box}(X,Y) = \int_{X-0.5}^{X+0.5} \int_{Y-0.5}^{Y+0.5} I(t,\tau)\,\mathrm{d}t\mathrm{d}\tau.$$

Tegyük fel, hogy $P$ számú konstans színű primitív metszi az $(X,Y)$ pixel $1 \times 1$-es téglalapját. Jelölje a $p$ primitív színét $C_p$, a metszet területét $I_p$. Ekkor az integrál az

$$I_{s,box}(X,Y) = \sum_{p=1}^{P} C_p \cdot I_p$$

alakra egyszerűsödik.

Kúpszűrő esetén, $(X,Y)$ középpontú $(r,\theta)$ polárkoordináta-rendszert feltételezve:

$$I_{s,cone}(X,Y) = \frac{3}{\pi} \iint_{x^2+y^2\le 1} I(x,y)\cdot(1-r)\,dxdy = \frac{3}{\pi} \int_{r=0}^{1} \int_{\theta=0}^{2\pi} I(r,\theta)\cdot(1-r)\cdot r\,\mathrm{d}\theta\mathrm{d}r.$$

A dobozszűrőhöz hasonlóan ha $P$ konstans színű primitív járul egy pixelhez, vagyis metszi az $(X,Y)$ középpontú, egységsugarú kört, akkor az integrál leegyszerűsödik az

$$I_{s,cone}(X,Y) = \frac{3}{\pi} \sum_{p=1}^{P} C_p \int_{I_p} (1-r)\mathrm{d}i$$

alakra, ahol $\int_{I_p} (1-r)\mathrm{d}i$ a kúpnak az $I_p$ metszet fölé eső térfogatrésze.

Hengerszűrő esetén, $(X,Y)$ középpontú $(r,\theta)$ polárkoordináta-rendszert feltételezve:

$$I_{s,cylinder}(X,Y) = \frac{4}{\pi} \iint_{x^2+y^2\le 1} I(x,y)\cdot 1\,dxdy = \frac{3}{\pi} \int_{r=0}^{1} \int_{\theta=0}^{2\pi} I(r,\theta)\cdot r\,\mathrm{d}\theta\mathrm{d}r.$$

Ha $P$ konstans színű primitív járul egy pixelhez, vagyis metszi az $(X,Y)$ középpontú, egységsugarú kört, akkor az integrál a következő alakra egyszerűsödik:

$$I_{s,cylinder}(X,Y) = \frac{4}{\pi} \sum_{p=1}^{P} C_p \int_{I_p} 1\mathrm{d}i = \frac{4}{\pi} \sum_{p=1}^{P} C_p \cdot I_p.$$

# 5 Antialiasing alkalmazása az előrevetítésben

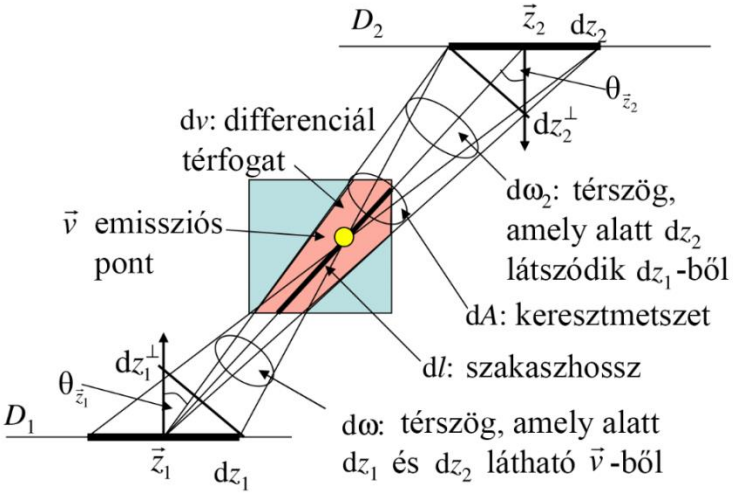
Az előrevetítés során a detektortalálatok várható számát határozzuk meg a radioaktív bomlások eloszlására állított becslésünk és a detektorérzékenység alapján. Az $L$ LOR-ban detektált események várható értéke megegyezik a rekonstruálandó térfogat összes pontjának a hozzájárulásának az összegével:

$$\tilde{y}_L = \int_V x(\vec{v})\,\tau(\vec{v} \to L)\mathrm{d}v.$$

## 5.1 Az előrevetítés rekonstrukciós folyamata

Kizárólag a geometriát tekintve tegyük fel, hogy egy $\vec{v}$ pontban keletkező ɣ-foton pár egy adott LOR-ban kerül detektálásra, ha a LOR-t alkotó két detektorkristály a pontból $\vec{\omega}$, illetve $-\vec{\omega}$ irányokból látszanak. Ez azt jelenti, hogy a $\vec{v}$ emissziós pont és a $\vec{\omega}$ irány egyértelműen azonosítja a $\vec{z}_1$ és $\vec{z}_2$ találati pontokat a detektorfelületeken, vagy fordítva, a $\vec{z}_1$ és $\vec{z}_2$ találati pontok meghatározzák azokat a $\vec{v}$ emissziós pontokat és $\vec{\omega}$ irányokat, amelyek hozzájárulhatnak a LOR találatszámához.

LOR-központú megközelítés esetén az emissziós pontok és az irányok helyett az algoritmust a találati pontok vezérlik. Keressük tehát a változó transzformációhoz kapcsolódó Jacobi-determinánst, melynek segítségével a találatszám az emissziós pontok és irányok helyett az érzékelőfelület pontjai fölötti integrálként fejezhető ki.




*8. ábra: A változó transzformációhoz tartozó Jacobi-determináns kiszámítása. A térszög, amely alatt a $\vec{v}$ emissziós pontból a $dz_1$ és $dz_2$ detektorfelületek látszanak: $d\omega = \frac{dz_1 \cos\theta_{\vec{z}_1}}{|\vec{z}_1-\vec{v}|^2} = \frac{dz_2 \cos\theta_{\vec{z}_2}}{|\vec{z}_2-\vec{v}|^2}$. A térszög, amely alatt $dz_2$ látszik a $\vec{z}_1$ pontból: $d\omega_2 = \frac{dz_2 \cos\theta_{\vec{z}_2}}{|\vec{z}_2-\vec{z}_1|^2} = \frac{dA}{|\vec{z}_1-\vec{v}|^2}$. Végezetül, a differenciál térfogat: $dv = dl\,dA$, ahol $dl$ a $\vec{z}_1\vec{z}_2$ egyenesnek a voxelbe eső hossza és $dA$ a keresztmetszet.*

A fenti ábra alapján a változó transzformációhoz kapcsolódó Jacobi-determináns:

$$\mathrm{d}\omega\,\mathrm{d}v = \frac{\cos\theta_{\vec{z}_1}\cos\theta_{\vec{z}_2}}{|\vec{z}_1 - \vec{z}_2|^2}\,\mathrm{d}l\mathrm{d}z_1\mathrm{d}z_2,$$

ahol $\theta_{\vec{z}_1}$ és $\theta_{\vec{z}_2}$ a megfelelő felületi normálvektorok és a $\vec{z}_1$ és $\vec{z}_2$ pontokat összekötő egyenes közötti szögek. Ennek segítségével az integrál az adott LOR $D_1$ és $D_2$ detektorfelülete és a két felület $\vec{z}_1$, illetve $\vec{z}_2$ pontját összekötő szakasz fölötti hármas integrállal fejezhető ki:

$$\tilde{y}_L = \int_{D_1}\int_{D_2} \frac{\cos\theta_{\vec{z}_1}\cos\theta_{\vec{z}_2}}{2\pi|\vec{z}_1 - \vec{z}_2|^2}\left(\int_{\vec{z}_1}^{\vec{z}_2} x(\vec{l})\mathrm{d}l\right)\mathrm{d}z_1\mathrm{d}z_2.$$

*5. egyenlet*

A fenti integrál értéke diszkrét mintavételezéssel közelíthető. Jelöljünk ki a detektorfelületen $N_{samples}$ egyenletes eloszlású $(\vec{z}_1, \vec{z}_2)$ pontpárt, majd minden $\vec{z}_1\vec{z}_2$ szakaszt osszunk fel kis részekre $N_{march}$ darab $\vec{l}_{ij}$ pont kiválasztásával a szakaszon. Az integrálközelítő összeg:

$$\tilde{y}_L \approx \frac{D_1 D_2}{2\pi N_{samples}} \sum_{i=1}^{N_{samples}} \left(\frac{\cos\theta_{\vec{z}_1}\cos\theta_{\vec{z}_2}}{|\vec{z}_1 - \vec{z}_2|^2} \sum_{j=1}^{N_{march}} \left(x(\vec{l}_{ij})\Delta l_i\right)\right).$$

*6. egyenlet*

Munkám során az $\vec{l}_{ij}$ mintapontok választását különböző algoritmusokkal végeztem, melyeket az 5.3 fejezetben fejtek ki.

A továbbiakban az alábbi egyszerűsítő jelölést vezetem be:

$$g = \frac{D_1 \cdot D_2}{2\pi \cdot N_{samples}} \cdot \frac{\cos\theta_{\vec{z}_1}\cos\theta_{\vec{z}_2}}{|\vec{z}_1 - \vec{z}_2|^2},$$

melynek felhasználásával az integrálközelítő összeg a következő alakra egyszerűsödik:

$$\tilde{y}_L \approx \sum_{i=1}^{N_{samples}} \left(g \cdot \sum_{j=1}^{N_{march}} \left(x(\vec{l}_{ij})\Delta l_i\right)\right).$$

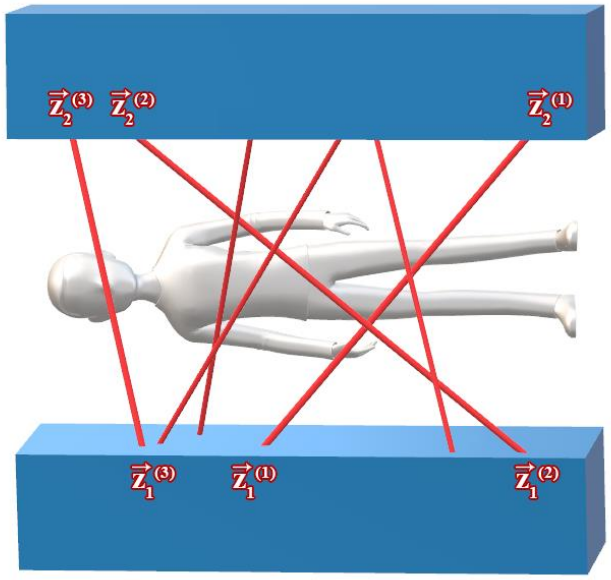
## 5.2 Tesztkörnyezet

Az előrevetítés szimulálására és a vonalintegrált közelítő eljárások összehasonlítására két tesztprogramot írtam C++ nyelven. Az első egy hagyományos, CPU-n futó program, mely implementálja az 5.3 fejezetben olvasható vonalhúzó algoritmusokat az előrevetítés végrehajtására. Ezt később átültettem CUDA platformra [31], amely a GPU-n futva kihasználja annak masszívan párhuzamos architektúráját a számítások gyorsítására.

### 5.2.1 CPU program

A pácienst magába foglaló mérési területet egy háromdimenziós voxeltömb reprezentálja. A tömb valós értékeket tartalmaz, elemei az adott voxel aktivitását ($x_V$) tárolják. A voxeltömbhöz hozzátartozik továbbá az $outside$ paraméter is, mely azt adja meg, hogy ha egy vonalhúzó algoritmus kilép a tömb határán, akkor milyen aktivitásértéket kapjon. Ennek segítségével vizsgálható, hogy az adott eljárásra mekkora hatással van a mérési terület közvetlen környezete.

A szimuláció során egyetlen LOR találatszámát számolom ki külön-külön a bemutatott algoritmusokkal. Ebben az egyszerűsített modellben a LOR-t alkotó két, szemközti detektorkristályon és a köztük húzódó mérési területen kívül más nem található. A LOR-t a két detektorfelületen $N_{samples}$ egyenletes eloszlású, véletlen $(\vec{z}_1, \vec{z}_2)$ pontpár kijelölésével mintavételezem.




*9. ábra: Egyszerűsített modell egyetlen LOR találatszámának kiszámításához.*

A vonalhúzó függvények paraméterei a következők:

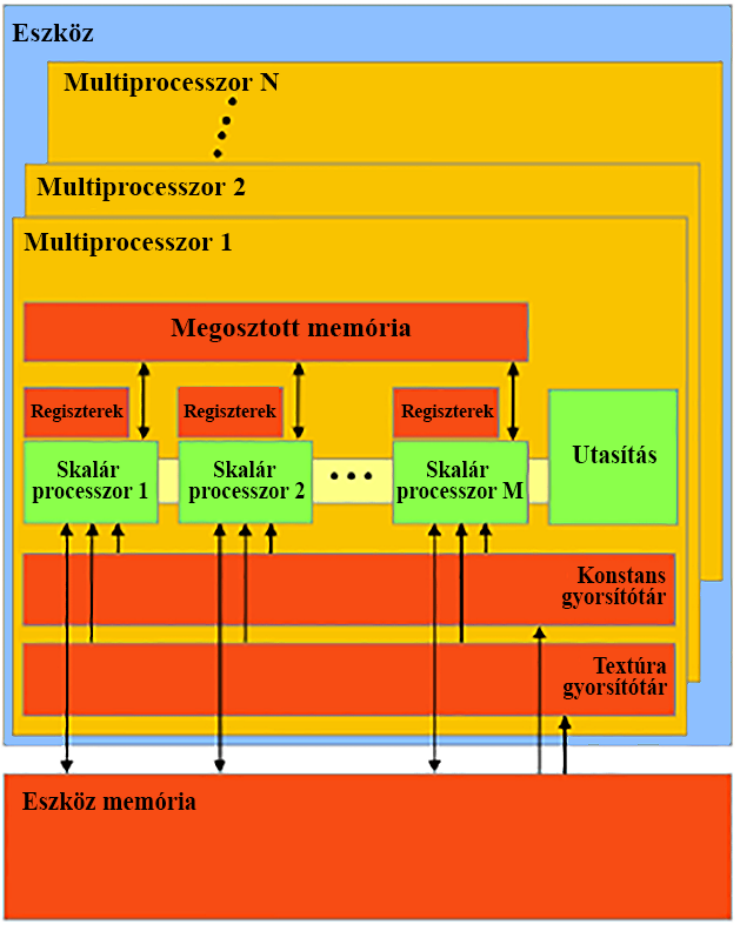
- *yLOR:* ebben a változóban kerül összegzésre a LOR találatszáma. A $\vec{z}_1\vec{z}_2$ szakasz mentén végighaladva a voxeltömbön a találatszámhoz hozzáadódik az érintett voxelek hozzájárulása.
- $g$: szorzótényező a vonalintegrál számításához. Pontos értéke (lásd 5.1 fejezet): $\frac{D_1 \cdot D_2}{2\pi \cdot N_{samples}} \cdot \frac{\cos\theta_{\vec{z}_1} \cos\theta_{\vec{z}_2}}{|\vec{z}_1 - \vec{z}_2|^2}$. Egy tetszőleges (x,y,z) voxel hozzájárulása a LOR találatszámához: $g \cdot \Delta l \cdot GetVoxelValue(x, y, z)$, ahol $\Delta l$ a $\vec{z}_1\vec{z}_2$ szakasznak az adott voxelbe eső részének hossza.
- $x_0, y_0, z_0$: a vonal $\vec{z}_1$ kezdőpontjának koordinátái. Modellem szerint a kezdőpont mindig az alsó detektorfelületen helyezkedik el, így $y_0 = 0$.
- $x_1$, $y_1$, $z_1$: a vonal $\vec{z}_2$ végpontjának koordinátái. Modellem szerint a végpont mindig a felső detektorfelületen helyezkedik el, így $y_1 = HEIGHT_{voxelArray}$.

### 5.2.2 CUDA program

A CUDA (Compute Unified Device Architecture) az Nvidia által létrehozott platform párhuzamos számítások elvégzésére. A programozók körében az OpenCL mellett az egyik legnépszerűbb választás GPGPU (General-Purpose Computing on Graphics Processing Units) számítások végrehajtására egyszerű használatának és hatékonyságának köszönhetően [40]. Népszerűségéhez az is hozzájárul, hogy olyan ismert programozási nyelvekkel dolgozik, mint például a C, C++ és Fortran, így nem igényel speciális nyelvismeretet.

## A CUDA programozási modellje

A CUDA a hagyományos grafikus csővezetéktől jelentősen eltérő programozási modellt nyújt. A GPU-t multiprocesszorok gyűjteményeként prezentálja a programozó felé, ahol minden multiprocesszor több, SIMD (Single Instruction Multiple Data) elven működő skalár processzort tartalmaz, amelyek minden pillanatban ugyanazt a gépi utasítást hajtják végre. A skalár processzoroknak saját regiszterkészletük vannak. Hozzáférnek a lassú globális memóriához, továbbá egy multiprocesszoron belül képesek gyors megosztott memórián keresztül kommunikálni egymással. Ezenkívül a CUDA eszköz (a GPU) beépített gyorsítótárat is tartalmaz textúrák és konstans adatok hatékony eléréséhez.

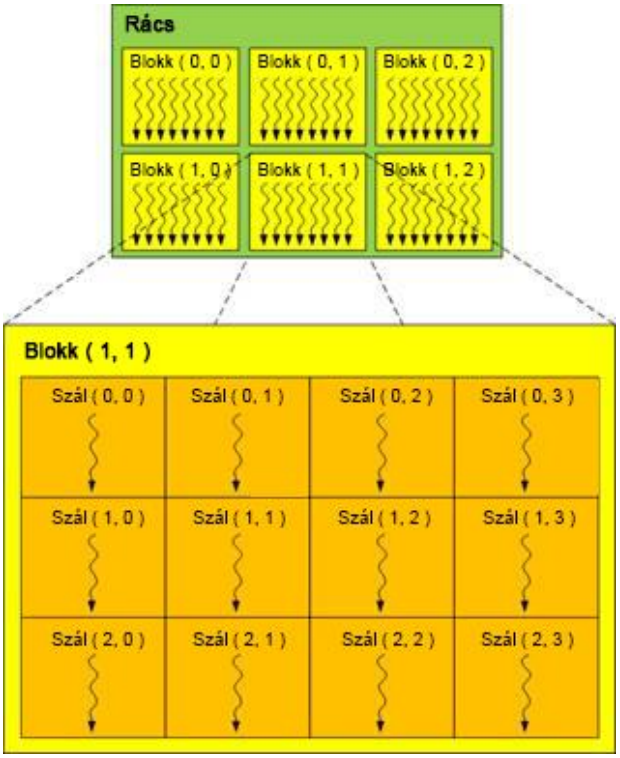



*10. ábra: CUDA programozási modell. Eredeti kép:* [41].

A modell programozásának legnagyobb kihívását az algoritmusok részfeladatokra, úgynevezett *kernelekre* bontása jelenti. A kernel lényegében egy szubrutin, amit egymással párhuzamosan futó *szálak* hajtanak végre az eszközön. A szálak között biztosítani kell, hogy minimális kommunikáció és szinkronizáció legyen szükséges, ugyanakkor a processzorok kapacitását minél jobban kihasználják. A szálak *blokkokba* vannak csoportosítva, amelyek egy-egy multiprocesszorhoz kerülnek hozzárendelésre. Azt, hogy egy blokk hány szálból áll, a programozó határozza meg. Egy blokkon belül a szálak hatékonyan tudnak kommunikálni egymással a megosztott memórián keresztül, így köztük a szinkronizáció egyszerűen megvalósítható. A blokkon belül a szálak a programozó döntésétől függően egy, két vagy három dimenzióban helyezkedhetnek el, és egyedi azonosítóval rendelkeznek, ahol a koordináták száma megegyezik az elrendezés dimenziószámával.

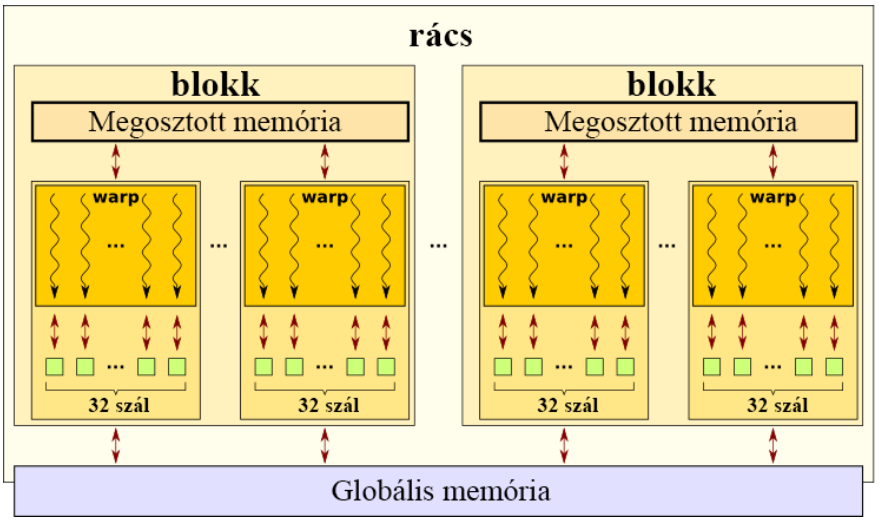
A blokkokat is tovább lehet csoportosítani *rácsokba*. A rács ugyanolyan dimenziójú blokkokból áll, ahol mindegyik blokk ugyanazt a kernelt hajtja végre. A rácsok hasznosak, ha sok szálat szeretnénk párhuzamosan indítani, mivel az architektúra egy blokkot legfeljebb 512 szálra korlátoz. Azonban gyors megosztott memória csak blokkszinten van, rácsszinten a szálak csak a lassú globális memórián keresztül tudnak kommunikálni. A rácson belül a blokkokat is lehet egy, két, illetve három dimenzióba rendezni, és az egyes blokkok egyedi azonosítóval rendelkeznek.

Az alábbi ábrán egy $2 \times 3$ blokkból álló rács látható. Minden blokk $3 \times 4$ szálból áll, ami összesen 72 szálat jelent. Minden szál ugyanazt a kernelkódot hajtja végre.




*11. ábra: Példa egy lehetséges szálhierarchiára.. Eredeti kép:*[42].

A multiprocesszorok a hozzájuk rendelt blokk szálait úgynevezett *warpokra* osztják, ahol minden warp 32 szálból áll. (Érdemes tehát a blokkonkénti szálak számát 32 többszörösének választani.) Egy warp szálait a processzor egymással párhuzamosan, SIMD módon dolgozza fel. Amikor a lassú memóriaelérés miatt a warp szálai várakozni kényszerülnek, a hardveres ütemező a következő warp végrehajtására vált. Emiatt a blokkok méretét úgy szokták megválasztani, hogy egynél több warpból álljon. Azonban a blokkonkénti szálak számát nem érdemes túlságosan nagynak választani, mivel ekkor kevés blokkot kapnánk, amelyek nem tudnák az összes rendelkezésre álló multiprocesszort kihasználni. Ez az egyik oka annak, hogy az architektúra legfeljebb 512 szálat engedélyez blokkonként. Mindezen szempontokat figyelembe véve általában 256 szálat szokás egy blokkhoz rendelni.




*12. ábra: A blokkok felbontása warpokra. Eredeti kép:*[43].

## A hatékony GPU implementáció szempontjai

A hatékony kód írása megköveteli a GPU jellemvonásainak figyelembe vételét már a problémameghatározás és az algoritmusfejlesztés első lépéseitől.

A következő szempontokat kell figyelembe venni:

Szál divergencia:

A SIMD végrehajtás miatt az algoritmus fejlesztése során minimalizálnunk kell a munka folyamának függését a bemeneti adatoktól.

Összefüggő memória hozzáférés és ütközésmentes írások:

Általánosságban elmondható, hogy a GPU-n a memória hozzáférés kifejezetten lassú a regiszter hozzáféréshez és a processzorok számítási teljesítményéhez képest, különösen, ha atomikus írásokra van szükség a szálütközések elkerülése érdekében. NVIDIA GPU-kon például az időkülönbség két nagyságrendet is jelenthet [44]. Az atomikus írások elkerülése mellett jelentős

gyorsulás érhető el az úgynevezett *egyesített memória hozzáféréssel* (coalesced memory access) is. Ha ugyanannak a skalár processzornak a szálai egymással szomszédos adatelemeket szeretnének elérni, akkor az átvitel egyetlen lépésben kerül végrehajtásra, drasztikusan lecsökkentve a hozzáféréshez szükséges időt. Ez azt jelenti, hogy az algoritmusokat úgy kell megtervezni, hogy a szomszédos szálak szomszédos adatelemekkel dolgozzanak.

Az iteratív maximum likelihood rekonstrukcióban felváltva hajtunk végre előrevetítést és visszavetítést. A két lépés egyenletei hasonlóak abban az értelemben, hogy mindkettő sok bemeneti értéket vár (voxel intenzitásokat, illetve LOR-okat) és sok kimeneti értéket számol ki (hasonlóan, LOR-okat és voxel intenzitásokat). Az ehhez hasonló „sok-sok" számítás kétféle struktúra szerint szervezhető. Egyrészt végighaladhatunk egyesével a bemeneti értékeken, minden alkalommal meghatározva az adott bemenet hozzájárulását minden egyes kimenethez. Ezt a sémát *bemenet vezérelt* vagy *szórás típusúnak* nevezzük. Másrészt ugyanez a gondolatmenet a kimenet szerint is végrehajtható: egyesével végigjárhatunk a kimeneti értékeken, kiszámolva a bemenetek hozzájárulását az adott kimeneti értékhez. Ezt a megközelítést *kimenet vezérelt* vagy *gyűjtés típusúnak* nevezzük. A gyűjtés típusú algoritmusok nagy előnye a szórás típusúakhoz képest, hogy náluk teljesen el lehet kerülni az írási ütközéseket, és bizonyos esetekben a memória hozzáférés összefüggőségét is növelik [45]. Az előrevetítés gyűjtés jellegű megközelítése, ha a szálakat a LOR-okhoz rendeljük és minden szál egyetlen $L$ LOR várható találatszámát számítja ki. Visszavetítésnél pedig a szálakat voxelekhez rendeljük úgy, hogy minden szál külön voxelben számítja ki $x_V$-t [45]. Hatékonyságán kívül e módszernek a tárigénye is alacsony, mivel csak a tömörített eseménylistára, egy LOR tömbre és egy voxeltömbre van szükség a végső eredményt tároló paraméterlistákon kívül.

Újrafelhasználás:

Különböző processzorokon futó, különálló szálak esetén nem tudjuk felhasználni a más processzorokon kapott részeredményeket, ami nyilvánvalóan lehetséges és célszerű lenne egy egyszálas implementációban. A részeredmények újrahasználata érdekében az algoritmust fázisokra kell bontani, például voxel feldolgozásra, voxel-LOR transzformációra és LOR feldolgozásra. Amikor az egyes fázisokat megvalósító szálak végeznek, az eredményeiket a globális memóriába írják. A következő fázis szálai felhasználhatják ezeket a köztes adatokat egymással párhuzamosan, külön kommunikáció nélkül.

Menet közbeni számítás:

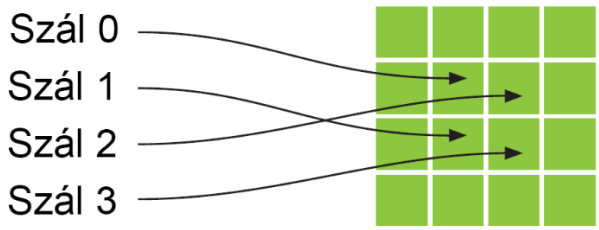
A jelenlegi GPU-k egy teraflopnál nagyobb számítási teljesítményt kínálnak, de a tárolókapacitásuk kicsi és a kommunikáció a CPU-val lassú. Ennélfogva a GPU implementáció során kompromisszumot kell kötni a tárolt előszámítás és a menet közbeni újraszámítás között. A PET rekonstrukcióban például a rendszermátrix a hatalmas mérete miatt teljes terjedelmében nem tárolható, annak elemeit célszerű minden használat előtt újra kiszámítani [15].

## Tesztprogramom felépítése

Az előrevetítést szimuláló CUDA tesztprogramot C++ nyelven írtam meg. A fordításhoz az Nvidia LLVM-alapú C/C++ fordítóját, az NVCC-t [46] használtam. A program struktúrájának felépítése során figyelembe vettem az előzőekben kifejtett szempontokat a hatékony GPU implementáció eléréséhez.

A CPU programhoz hasonlóan a LOR-t a detektorfelületeken véletlen $(\vec{z}_1, \vec{z}_2)$ pontpárok kijelölésével mintavételezem. Gyűjtés típusú megközelítést alkalmazva minden szál külön számolja a LOR találatszámát. (Mivel most csak egy LOR van, így minden szál ugyanazt a LOR-t számolja, és a kapott eredményeket átlagolom.) Mivel a szálak egymástól függetlenül számolnak, nem léphet fel írási ütközés. Blokkméretnek a 256-ot választottam, melyen belül a szálak egydimenzióban helyezkednek el. A rács $\left\lceil \frac{N_{samples}}{256} \right\rceil$ blokkból áll. (Amennyiben $N_{samples}$ nem osztható 256-tal, „felesleges” szálak is végrehajtásra kerülnek. Emiatt minden szál ellenőrzi, hogy az ő azonosítója kisebb-e $N_{samples}$-nél, és csak ebben az esetben generál mintapontpárt és végzi el az előrevetítést.)

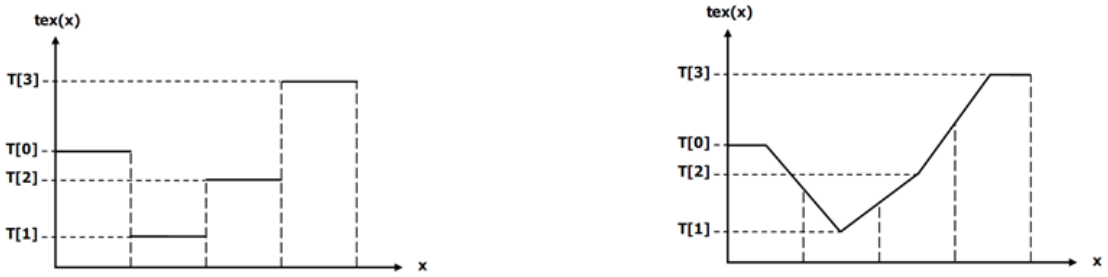
A voxeltömb gyors elérése kulcsfontosságú szereppel bír a program teljesítményét tekintve. A tömb egyszer, a program elején kerül feltöltésre értékekkel, ezt követően a szálak csak olvassák annak adatait, méghozzá nagyfokú térbeli lokalitást követve. A CUDA memória modelljében található egy memóriatípus kifejezetten ilyen jellegű feladatokhoz: az úgynevezett *textúra memória*. A textúra memória egy olyan, csak olvasható memória, amely bizonyos elérési mintákat követő olvasások esetén jelentősen lecsökkenti a szükséges memóriaforgalmat, ezáltal növelve a végrehajtás sebességét. Tervezése során eredeti célja a ritkán változó textúrák elérésének gyorsítása volt hagyományos grafikus alkalmazásokban, de bármilyen GPGPU program számára rendkívül hasznos, ahol az egymáshoz közeli szálak egymáshoz közeli memóriacímeket olvasnak. A textúra memória tulajdonképpen a globális memória része, de gyorsítótárazva van a grafikus kártyán a speciális térbeli lokalitású memóriaelérésekhez.




*13. ábra: Térbeli lokalitás a szálak között. Eredeti kép:* [47].

A fenti képen látható memóriaelérési szituációban hagyományos gyorsítótárak esetén nem lenne cache találat, mivel a négy memóriacím nem sorfolytonosan egymás után helyezkedik el. Azonban mivel a GPU textúra gyorsítótárat az ehhez hasonló elérési mintákhoz tervezték, jelentős sebességbeli növekedés figyelhető meg. A cache találatok szempontjából az is előnyös, ha egy szálon belül az egymást követő memóriaelérési kérések térben egymáshoz közel eső címekre szólnak. Mivel a tesztprogramban minden szál egy vonal mentén halad végig a voxeltömbön, a program nagymértékben kihasználja a gyorsítótárat.

A textúra memória másik előnye, hogy beépített hardveres lineáris interpolációt biztosít. Kérhetjük egy egydimenziós textúra 10.5-edik texelének értékét; ekkor a tizedik és a tizenegyedik texel átlagát kapjuk.




*14. ábra: Beépített lineáris interpoláció. Kép forrása:* [47].

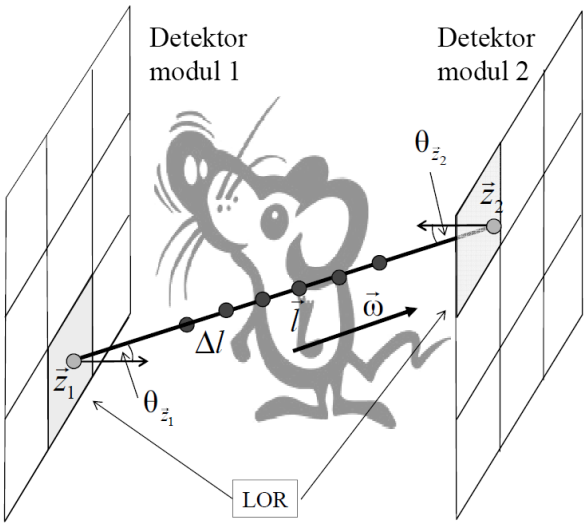
A tesztprogramban a lineáris interpolációt a memóriaelérési kérések számának csökkentésére használtam az Antialiased Bresenham és a Gupta-Sproull eljárásoknál. Az aktuális ciklus iterációban vizsgált voxelek között a súlyozásuk arányának megfelelően lineárisan interpolált értéket olvasok ki.

## 5.3 Vonalhúzó algoritmusok

A számítógépes grafikában számos algoritmus terjedt el egyenesek rajzolásához. A hagyományosnak tekinthető Bresenham [48] és Siddon [49] mellett a megjelenítő hardver fejlődésével igény keletkezett a finomabb vonalakat eredményező, antialiased vonalhúzó módszerekre is. Ezek túlnyomó része korábban már létező eljárásokat egészítenek ki az egyenes szűrésével. Ilyen az Antialiased Bresenham és a Gupta-Sproull [50] algoritmus, melyek Bresenham módszerét alapul véve hajtanak végre doboz-, illetve kúpszűrést a kirajzolandó szakaszon az aliasing csökkentése érdekében (15. ábra). Ezek az eljárások minimális módosítással elvonatkoztathatóak a grafikusok megjelenítéstől, és felhasználhatóak általános számítások végzésére. Ebben a fejezetben bemutatásra kerül, hogy az egyes algoritmusok segítségével hogyan lehet az előrevetítés során meghatározni a vonalintegrál értékét és becslést adni egy LOR találatszámára.

*15. ábra: Bresenham, Antialiased Bresenham és Gupta-Sproull algoritmussal rajzolt egyenesek.*

### 5.3.1 Ray marching




*16. ábra: Ray marching.*

A ray marching a $\vec{z}_1\vec{z}_2$ szakaszt $N_{march}$ egyenlő hosszúságú szakaszra osztja fel, vagyis az $\vec{l}_{ij}$ mintapontok azonos, $\Delta l$ távolságra helyezkednek el.

Az algoritmus pszeudokódja:

**Ray Marching**$(\mathbf{yLOR}, \mathbf{g}, \mathbf{x_0}, \mathbf{y_0}, \mathbf{z_0}, \mathbf{x_1}, \mathbf{y_1}, \mathbf{z_1})$:

$\Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0, \quad \Delta z = z_1 - z_0$

$\Delta l = \frac{\sqrt{(\Delta x)^2+(\Delta y)^2+(\Delta z)^2}}{N_{march}}$

**for** $i = 0$ amíg $i < N_{march}$

$x = x_0 + \frac{i+0.5}{N_{march}} \cdot \Delta x\,, \quad y = y_0 + \frac{i+0.5}{N_{march}} \cdot \Delta y\,, \quad z = z_0 + \frac{i+0.5}{N_{march}} \cdot \Delta z$

$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z)$

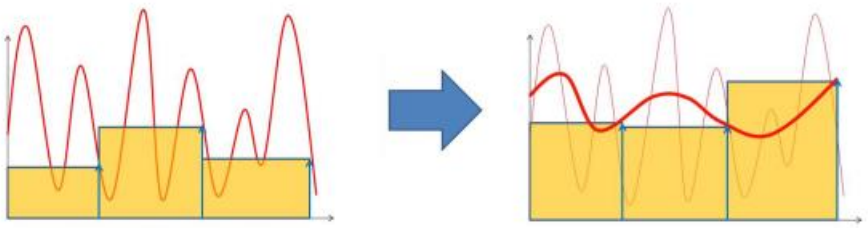
**endfor**

**eljárás vége**

### 5.3.2 Szűrt ray marching

#### Matematikai háttér

A 6. egyenlet a várható LOR találatok Monte Carlo becslése, mely diszkrét $\vec{l}_{ij}$ pontmintákat vesz a voxeltartományban. Annak érdekében, hogy pontos becslést kapjunk, az érintett LOR számára releváns bázisfüggvényeket kellően sűrűn kell mintavételezni, ami nagyon magas mintaszámhoz vezetne. Ha viszont a bázisfüggvényeket alulmintavételezzük, a becslés szórása nagy lesz, különösen akkor, ha az aktivitás szórása is nagy. A javasolt szűrés az aktivitás hirtelen ingadozásainak simítására használható a Monte Carlo mintavételezés előtt.



*17. ábra: A szűrt mintavételezés csökkenti az integrál kvadratúra közelítési hibáját az integrandus szórásának redukálásával.*

A szűrt mintavételezés [51] kicseréli az integrandust egy másik függvényre, melynek integrálja hasonló, de szórása kisebb, ezért diszkrét mintákkal pontosabban közelíthető (lásd 17. ábra). A szórás csökkentése egy aluláteresztő szűrő segítségével valósítható meg, mely kiszűri a magas frekvenciájú ingadozásokat. A szűrő feladata, hogy kiszűrje a mintavételi pontok sűrűségének megfelelő határon túli frekvenciákat, mindeközben az integrált csak minimális mértékben módosítsa.

A szűrt mintavételezést egy általános $f(x)$ függvény szűrésén keresztül mutatom be. Közelítsük az $f(x)$ függvény $I_f$ integrálját a $[0,1]$ tartományon $M$ darab független, egyenletes eloszlású véletlen minta felvételével ($x_1, \dots, x_M \in [0,1]$ ). Az $\hat{I}_f$ Monte Carlo becslő:

$$I_f = \int_0^1 f(x)\mathrm{d}x \approx \frac{1}{M}\sum_{i=1}^{M} f(x_i) = \hat{I}_f.$$

Az $\hat{I}_f$ véletlen becslő várható értéke megegyezik az eredeti integrandussal, azaz torzítatlan, és szórása a következő:

$$\boldsymbol{V}[\hat{I}_f] = \frac{1}{M}\left(\int_0^1 f^2(x)\mathrm{d}x - I_f^2\right).$$

Hajtsunk végre szűrést az integranduson a $b(x)$ kernellel! Az így kapott függvényt jelölje $F(x)$.

$$F(x) = \int_{-\infty}^{\infty} b(y)f(x-y)\mathrm{d}y = \int_{-\infty}^{\infty} f(y)b(x-y)\mathrm{d}y,$$

ahol a szűrő kernel nemnegatív, normalizált, és a $[-\Delta x, \Delta x]$ intervallumra korlátozott:

$$\int_{-\infty}^{\infty} b(y)\mathrm{d}y = \int_{-\Delta x}^{+\Delta x} b(y)\mathrm{d}y = 1.$$

Továbbá az is feltehető, hogy $b(y)$ szimmetrikus, tehát $b(y) = b(-y)$.

A nemnegatív, normalizált szűrő kernel valószínűségi sűrűségnek is tekinthető. A szűrés művelete ekkor várható érték számítással egyezik meg:

$$F(x) = \boldsymbol{E}_{b(y)}[f(x-y)].$$

A szűrt integrandus integrálja a következő:

$$I_F = \int_{x=0}^{1} F(x)\mathrm{d}x = \int_{x=0}^{1} \boldsymbol{E}_{b(y)}[f(x-y)]\mathrm{d}x = \boldsymbol{E}_{b(y)}\left[\int_{x=0}^{1} f(x-y)\mathrm{d}x\right]$$
$$= \boldsymbol{E}_{b(y)}\left[\int_{x_1=-y}^{1-y} f(x_1)\mathrm{d}x_1\right] = I_f + \boldsymbol{E}_{b(y)}\left[\int_{x=-y}^{0} f(x)\mathrm{d}x - \int_{x=1-y}^{1} f(x)\mathrm{d}x\right].$$

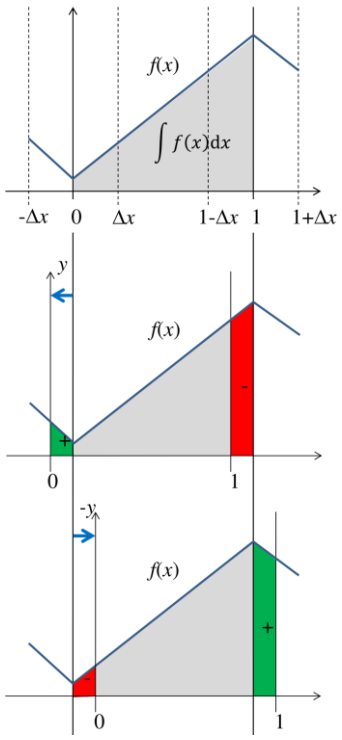
$F$ integrálja általában nem egyezik meg $f$-ével, mivel a szűrés átlépi az intervallum két határát. Következésképpen a Monte Carlo kvadratúra kiértékelése $F$-en $f$ integráljának torz becslését eredményezi. A határátlépés által okozott torzítás:

$$B = \boldsymbol{E}_{b(y)}\left[\int_{x=-y}^{0} f(x)\mathrm{d}x - \int_{x=1-y}^{1} f(x)\mathrm{d}x\right].$$

Ez a torzítás kiküszöbölhető, ha $f$-et az eredeti, $[0,1]$ integrálási tartományon kívül megfelelően definiáljuk.

Legyen $f(x) = f(-x)$, ha $-\Delta x < x < 0$ és $f(x) = f(2-x)$, ha $1 < x < 1 + \Delta x$, egyszóval tükrözzük az integrandust a két határra a szűrő kernel értelmezési tartományán belül. Ekkor $F$ integrálja a következőképpen számítható ki:

$$I_F = \boldsymbol{E}_{b(y)}\left[\int_{x_1=-y}^{1-y} f(x_1)\mathrm{d}x_1\right].$$




*18. ábra: Bizonyíték arra, hogy* $\boldsymbol{E}_{b(y)}\left[\int_{x=0}^{1} f(x-y)dx\right] = \boldsymbol{E}_{b(y)}\left[\int_{x=0}^{1} f(x)dx\right]$*, ha* $b(y)$ *szimmetrikus és* $f(x)$ *szimmetrikus a* $[-\Delta x, \Delta x]$ *és* $[1-\Delta x, 1+\Delta x]$ *határrégiókban.*

Az $y$ változó az $I_f$ integrál tartományának az eltolása. Amikor $y$ pozitív, a tartomány balra tolódik, hozzáadva az integrandushoz a $[-y, 0]$ intervallumba eső részt és kivonva az $[1-y, 1]$-be esőt. A két érintett régió a 18. ábrán pirossal, illetve zölddel van jelölve. Mivel az $f$ integrandus szimmetrikus a $[-\Delta x, \Delta x]$ és az $[1-\Delta x, 1+\Delta x]$ intervallumokon belül, a módosított terület teljes nagysága megegyezik, ha $y$-t $(-1)$-gyel szorozzuk, csupán a hozzáadásból kivonás lesz és fordítva. Annak következtében, hogy a szűrő kernel, vagyis a $b(y)$ valószínűségi sűrűség is szimmetrikus, a két eset kioltja egymást. Mivel minden $y$-hoz tartozik megfelelő $(-y)$, a határon túli szűrés által okozott teljes módosítás zérus.

Hasonlítsuk össze $\hat{I}_F$ és $\hat{I}_f$ szórását! A kettő várható értéke megegyezik, és ugyanannyi mintát veszünk belőlük, így $\boldsymbol{V}[\hat{I}_F]$ akkor és csak akkor kisebb $\boldsymbol{V}[\hat{I}_f]$-nél, ha

$$\int_0^1 F^2(x)\mathrm{d}x \leq \int_0^1 f^2(x)\mathrm{d}x.$$

$F(x)$ definícióját felhasználva:

$$\int_0^1 F^2(x)\mathrm{d}x = \int_{x=0}^1 \left(\boldsymbol{E}_{b(y)}[f(x-y)]\right)^2 \mathrm{d}x.$$

*7. egyenlet*

Az integrandus az $\boldsymbol{E}_{b(y)}[f(x-y)]$ várható érték négyzete. Mivel a négyzetre emelés konvex függvény, a Jensen-egyenlőtlenség [52] alapján felírható, hogy

$$\left(\boldsymbol{E}_{b(y)}[f(x-y)]\right)^2 \leq \boldsymbol{E}_{b(y)}[f^2(x-y)].$$

Visszahelyettesítve a 7. egyenletbe:

$$\int_0^1 F^2(x)\mathrm{d}x \leq \int_{x=0}^1 \boldsymbol{E}_{b(y)}[f^2(x-y)]\mathrm{d}x = \boldsymbol{E}_{b(y)}\left[\int_{x=0}^1 f^2(x-y)\mathrm{d}x\right] = \boldsymbol{E}_{b(y)}\left[\int_{x=0}^1 f^2(x)\mathrm{d}x\right]$$
$$= \int_0^1 f^2(x)\mathrm{d}x.$$

Az $\boldsymbol{E}_{b(y)}\left[\int_{x=0}^1 f^2(x-y)\mathrm{d}x\right] = \boldsymbol{E}_{b(y)}\left[\int_{x=0}^1 f^2(x)\mathrm{d}x\right]$ egyenlőség során azt használtuk ki, hogy $f^2$ örökli $f$ szimmetriáját, így rá is alkalmazható a 18. ábrán látható gondolatmenet.

Ezzel beláttuk, hogy megfelelő $b(y)$ kernellel való szűrés az integrál értékét nem változtatja meg, de szórása kisebb, ezért diszkrét mintákkal pontosabban közelíthető.

**Szűrési algoritmus**

Szűrt ray marching esetén a ray marching végrehajtása előtt a voxeltömbön szűrést alkalmazunk. A tesztprogramban egy háromdimenziós szűrő kernelt használtam, melynél a középső voxel súlya $centerWeight \in [0,1]$, a hat szomszédos voxel súlya pedig $(1-centerWeight)/6$.

A kernel egy kétdimenziós metszete:

$$\begin{pmatrix} 0 & (1-centerWeight)/6 & 0 \\ (1-centerWeight)/6 & centerWeight & (1-centerWeight)/6 \\ 0 & (1-centerWeight)/6 & 0 \end{pmatrix}$$

Belátható, hogy a kernel nemnegatív, normalizált (a voxelsúlyok összege 1), szimmetrikus, és az $x, y, z \in [-1,1]$ tartományra korlátozott. A szűrés algoritmusa:

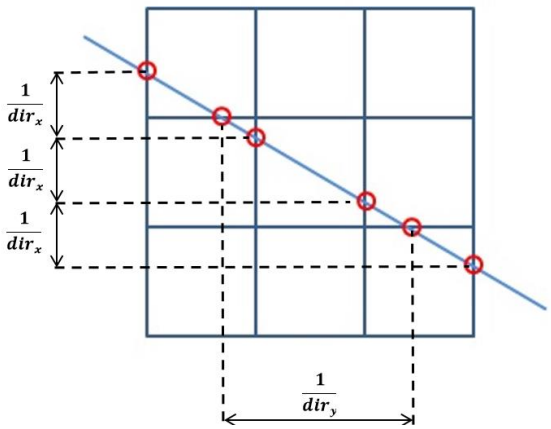
**Szűrés(centerWeight):**

tempArray$[HEIGHT_{voxelArray}][WIDTH_{voxelArray}][DEPTH_{voxelArray}]$
**for** $y = 0$ amíg $y < HEIGHT_{voxelArray}$
**for** $x = 0$ amíg $x < WIDTH_{voxelArray}$
**for** $z = 0$ amíg $z < DEPTH_{voxelArray}$
$\text{tempArray[y][x][z]} = \text{centerWeight} \cdot \text{GetVoxelValue(x, y, z)} + \frac{1-\text{centerWeight}}{6} \cdot (\text{GetVoxelValue(x, y + 1, z)} + \text{GetVoxelValue(x, y - 1, z)} + \text{GetVoxelValue(x + 1, y, z)} + \text{GetVoxelValue(x - 1, y, z)} + \text{GetVoxelValue(x, y, z + 1)} + \text{GetVoxelValue(x, y, z - 1)})$
**endfor**
**endfor**
**endfor**
SetVoxelArray(tempArray)
**eljárás vége**

Tesztprogramomban $centerWeight$ értékének 0.9-et választottam.

### 5.3.3 Siddon

A Siddon algoritmus arra az észrevételre épít, hogy a voxeleket határoló, $x$ tengelyre merőleges síkok és egy tetszőleges egyenes metszéspontjai egymástól mindig $\frac{1}{\overrightarrow{dir_x}}$ távolságra helyezkednek el, ahol $\overrightarrow{dir}$ az egyenes normalizált irányvektora. Hasonlóan az $y$, illetve a $z$ tengelyt tekintve a metszéspontok egymástól $\frac{1}{\overrightarrow{dir_y}}$, illetve $\frac{1}{\overrightarrow{dir_z}}$ távolságra vannak, tehát bármely tengely mentén a következő metszéspont egyetlen összeadással számítható.




*19. ábra: Voxelhatárt egy adott tengely mentén mindig rendre $\frac{1}{dir_x}$, $\frac{1}{dir_y}$, illetve $\frac{1}{dir_z}$ távolság után lépünk át, ahol dir az egyenes normalizált irányvektora. Az eredeti kép forrása:*[53].

Jelöljék a szakasz két végpontját az $(x_0, y_0, z_0)$ és $(x_1, y_1, z_1)$ valós koordináták, és tárolják az aktuális voxel koordinátáit az $X, Y$ és $Z$ egész értékű változók. Az $x$, $y$ és $z$ tengelyre merőleges voxelhatároló síkokkal keletkező metszéspontot rendre a $t_x$, $t_y$ és $t_z$ sugárparaméterek tartják nyilván. Adott pillanatban az $x$ tengely mentén a következő metszéspont az $x = x_0 + t_x \cdot dir_x$, az $y$ tengely mentén az $y = y_0 + t_y \cdot dir_y$, a $z$ tengely mentén pedig a $z = z_0 + t_z \cdot dir_z$ koordinátában lesz. A három sugárparaméter közül a legkisebb jelöli ki, hogy melyik tengely mentén fog az egyenes leghamarabb voxelhatárt átlépni. Minden ciklusiterációban a legkisebb sugárparaméterhez tartozó tengely mentén a következő voxelre lépünk ($X, Y$ vagy $Z$ iterálása), és az érintett sugárparaméter értékét egy előre kiszámolt növekménnyel növeljük ($txStep$, $tyStep$, illetve $tzStep$).

Kezdetben:

$$X = \text{Egészrész}(x_0), \quad Y = \text{Egészrész}(y_0), \quad Z = \text{Egészrész}(z_0)$$

**SugarparameterInicializal**$(\mathbf{t_x}, \mathbf{t_y}, \mathbf{t_z}, \mathbf{txStep}, \mathbf{tyStep}, \mathbf{tzStep})$:

ha $dir_x > 0$: $t_x = \frac{X+1-x_0}{dir_x}$, $txStep = \frac{1}{dir_x}$

egyébként ha $dir_x < 0$: $t_x = \frac{X-x_0}{dir_x}$, $txStep = \frac{-1}{dir_x}$

egyébként: $t_x = t_{max}$ // „$t_x = \infty$”

\+ ugyanez a hármas elágazás $dir_y$, illetve $dir_z$ alapján $t_y$ és $tyStep$, illetve $t_z$ és $tzStep$ meghatározására

**eljárás vége**

A következő voxel koordinátáit az alábbi módon kapjuk meg:

**for** amíg X, Y, Z nem lép túl az $(x_1, y_1, z_1)$ végponton

ha $t_x \leq t_y, t_z$: $X = X + sgn(dir_x)$, $t_x = t_x + txStep$

egyébként ha $t_y \leq t_x, t_z$: $Y = Y + sgn(dir_y)$, $t_y = t_y + tyStep$

egyébként: $Z = Z + sgn(dir_z)$, $t_z = t_z + tzStep$

**endfor**

A Siddon algoritmusnál a $\Delta l$, amely azt adja meg, hogy az vonal egy voxel belsejében mekkora utat tesz meg, nem számolható ki előre, ciklusiterációról ciklusiterációra változik. Jelölje $\Delta t \geq 0$ a mostani és az előző határátlépéshez tartozó sugárparaméterek különbségét.

ha $t_x \leq t_y, t_z$: $\Delta t = t_x -$ *az előző határátlépés sugárparamétere*

egyébként ha $t_y \leq t_x, t_z$: $\Delta t = t_y -$ *az előző határátlépés sugárparamétere*

egyébként: $\Delta t = t_z -$ *az előző határátlépés sugárparamétere*

Adott $\Delta t$ mellett az $x$ koordináta $\Delta t \cdot dir_x$-szel, az y koordináta $\Delta t \cdot dir_y$-nal, a z koordináta pedig $\Delta t \cdot dir_z$-vel növekedik, tehát

$$\Delta l = \sqrt{(\Delta t \cdot dir_x)^2 + \left(\Delta t \cdot dir_y\right)^2 + (\Delta t \cdot dir_z)^2} = \sqrt{\Delta t^2 \cdot \left(dir_x^2 + dir_y^2 + dir_z^2\right)} =$$

$\Delta t \cdot \sqrt{dir_x^2 + dir_y^2 + dir_z^2} = \Delta t \cdot 1 = \Delta t$, mivel az irányvektor normalizált.

A voxel hozzájárulása a LOR találatszámhoz:

$$yLOR = yLOR + g \cdot \Delta t \cdot GetVoxelValue(X, Y, Z).$$

A teljes algoritmus:

**Siddon**$(\mathbf{yLOR}, \mathbf{g}, \mathbf{x_0}, \mathbf{y_0}, \mathbf{z_0}, \mathbf{x_1}, \mathbf{y_1}, \mathbf{z_1})$:

$\Delta x = x_1 - x_0, \quad \Delta y = y_1 - y_0, \quad \Delta z = z_1 - z_0$

$lineLength = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$

$dir = (\frac{\Delta x}{lineLength}, \frac{\Delta y}{lineLength}, \frac{\Delta z}{lineLength})$

$X = \text{Egészrész}(x_0), \quad Y = \text{Egészrész}(y_0), \quad Z = \text{Egészrész}(z_0)$

$SugarparameterInicializal(t_x, t_y, t_z, txStep, tyStep, tzStep)$

$t_{last} = 0$

**for** amíg X, Y, Z nem lép túl az $(x_1, y_1, z_1)$ végponton

ha $t_x \leq t_y, t_z$:

$\Delta t = t_x - t_{last}, \qquad t_{last} = t_x$

$yLOR = yLOR + g \cdot \Delta t \cdot GetVoxelValue(X, Y, Z)$

$X = X + sgn(dir_x), \; t_x = t_x + txStep$

egyébként ha $t_y \leq t_x, t_z$:

$\Delta t = t_y - t_{last}, \qquad t_{last} = t_y$

$yLOR = yLOR + g \cdot \Delta t \cdot GetVoxelValue(X, Y, Z)$

$Y = Y + sgn(dir_y), \; t_y = t_y + tyStep$

egyébként:

$\Delta t = t_z - t_{last}, \qquad t_{last} = t_z$

$yLOR = yLOR + g \cdot \Delta t \cdot GetVoxelValue(X, Y, Z)$

$Z = Z + sgn(dir_z), \; t_z = t_z + tzStep$

**endfor**

eljárás vége

### 5.3.4 Bresenham

A könnyebb érthetőség kedvéért az algoritmust először kétdimenzióban, 0˚ és 45˚ közötti meredekségű egyenesekre ismertetem.

**2D, meredekség ∈ [0˚,45˚]**

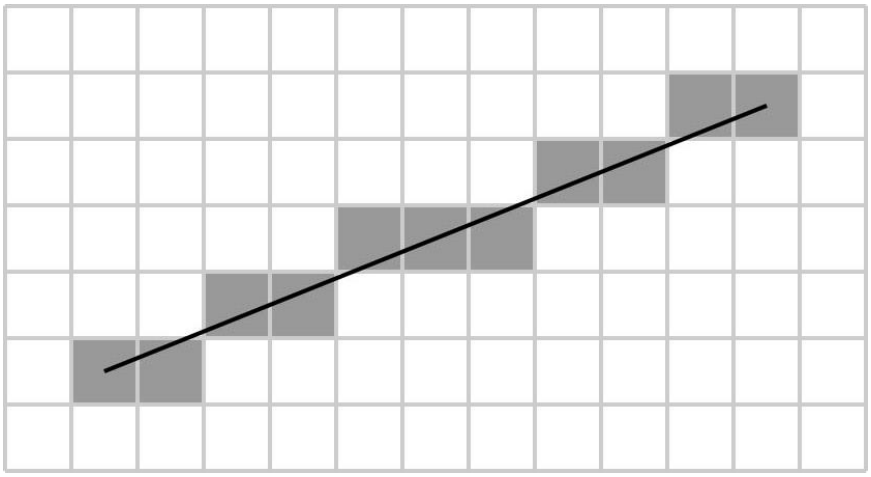
Legyen a szakasz kezdőpontja $(x_0, y_0)$ és végpontja $(x_1, y_1)$, ahol $x_0 < x_1$ és $y_0 < y_1$. Ha a meredekség 0˚ és 45˚ között van, akkor $\Delta x \geq \Delta y$, ezért a szakaszt az $x$ tengely mentén mintavételezzük.

A szakasz belső pontjainak a meghatározására egy kezdetleges, naív mód a következő:

$m = \Delta y / \Delta x$
**for** $x = x_0$ amíg $x < x_1$
    $y = \text{Kerekít}(m \cdot x + c)$
**endfor**

A fenti algoritmus működik, viszont nagyon lassú, mivel minden lépésben el kell végeznie az $m \cdot x + c$ lebegőpontos szorzást, majd a kapott valós számot egész koordinátává kell kerekítenie.

Bresenham inkrementális módon gondolkodott: ötlete az volt, hogy haladjunk végig az $x$ tengelyen egységnyi lépésközzel (pixelenként), és minden lépésben döntsük el, hogy $y$ értéke maradjon-e a jelenlegi vagy növekedjen eggyel. Más szóval tetszőleges $(x, y)$ koordináták esetén válasszuk ki, hogy a következő lépésben az $(x + 1, y)$ vagy az $(x + 1, y + 1)$ koordinátákra érkezzünk, attól függően, hogy melyik van közelebb az egyeneshez.



*20. ábra: 2D Bresenham algoritmus 0˚ és 45˚ közötti meredekségű egyenesekre*

Az algoritmus egy döntési paramétert vezet be, mely nyomon követi a korábbi $y$ választásoknak a meredekségben okozott hibáját. Ha a meredekségi hiba nagyobb lesz, mint 0.5, az azt jelenti, hogy az egyenes felfelé mozdult egy pixellel, emiatt növelni kell az $y$ koordinátát, és a különbözetet – azaz egyet – ki kell vonni a hibából.

$m = \Delta y / \Delta x$
$y = y_0, \quad slopeError = m - 0.5$
**for** $x = x_0$ amíg $x < x_1$
    $slopeError = slopeError + m$
    ha $slopeError \geq 0.5$: $y = y + 1,\ slopeError = slopeError - 1$
**endfor**

A fenti algoritmus továbbra is tartalmaz lebegőpontos aritmetikát. Ennek elkerülése érdekében módosítsuk az $m$ meredekséget, hogy az eredetinek a $\Delta x$-szorosa legyen (tehát $m = \Delta y$). Ennek megfelelően a meredekségi hibát is meg kell szorozni $\Delta x$-szel. Ekkor még mindig maradt egy 0.5-el való összehasonlítás, melynek kiküszöbölésére mind a meredekséget, mind a meredekségi hibát kettővel szorozzuk.

A teljes algoritmus:

**Bresenham**$(\mathbf{x_0}, \mathbf{y_0}, \mathbf{x_1}, \mathbf{y_1})$:
$\Delta x = x_1 - x_0;\ \ \Delta y = y_1 - y_0$
$m = 2 \cdot \Delta y$
$slopeError = 2 \cdot \Delta y - \Delta x$ *
$y = y_0$
**for** $x = x_0$ amíg $x < x_1$
$slopeError = slopeError + m$
ha $slopeError \geq 0$: **
$y = y + 1$
$slopeError = slopeError - 2 \cdot \Delta x$
**endfor**
**eljárás vége**

* A meredekségi hiba kezdőértékét [54] bizonyítja.
** Általában jobban kedvelt 1 helyett 0-val összehasonlítani, emiatt a meredekségi hibából ki szoktunk vonni 1-et. Így a kezdeti, valós értékű hibaváltozóból – jelölje most $slopeError_0$ – a végső, egész értékű hibaváltozó a következő módon kapható meg: $slopeError = 2 \cdot \Delta x \cdot (slopeError_0 - 1)$.

### 2D, tetszőleges meredekség

Bármilyen meredekségű egyenesnél vissza lehet vezetni a problémát a fent vázolt, egyszerűsített esetre.

1. $|\Delta x| \geq |\Delta y|$: A ciklus az $x$ tengely mentén halad.

Amennyiben $\Delta y < 0$, akkor $y$-t nem növelni, hanem csökkenteni kell. Ehhez vezessük be az $yStep = \begin{cases} -1, & ha\ \Delta y < 0 \\ 1, & ha\ \Delta y \geq 0 \end{cases}$ változót.

Ha $\Delta x < 0$, azaz $x_1 < x_0$, akkor visszavezethetjük a $\Delta x \geq 0$ esetre úgy, hogy kicseréljük a kezdő és a végpontot. Tehát kezdetben $x = x_1,\ y = y_1$ és a ciklus $x < x_0$-ig fut. Ekkor $\Delta x$ és $\Delta y$ is az ellentettjére változik, így $yStep$ is. A meredekség és a meredekségi hiba esetén $\Delta x$ és $\Delta y$ abszolútértékével kell számolni.

$m = 2 \cdot |\Delta y|$
$slopeError = 2 \cdot |\Delta y| - |\Delta x|$
$y = \begin{cases} y_0, & ha\ \Delta x \geq 0 \\ y_1, & ha\ \Delta x < 0 \end{cases}$, $yStep = \begin{cases} 1, & ha\ \Delta x \cdot \Delta y \geq 0 \\ -1, & ha\ \Delta x \cdot \Delta y < 0 \end{cases}$
**for** $x = \begin{cases} x_0 \to x_1, & ha\ \Delta x \geq 0 \\ x_1 \to x_0, & ha\ \Delta x < 0 \end{cases}$
$slopeError = slopeError + m$
ha $slopeError \geq 0$:
$y = y + yStep$
$slopeError = slopeError - 2 \cdot |\Delta x|$
**endfor**

2. $|\Delta x| < |\Delta y|$: A ciklus az $y$ tengely mentén halad.

   Hasonló az előző esethez, csak az $x$ és $y$ szerepek felcserélődnek.

$$m = 2 \cdot |\Delta x|$$
$$slopeError = 2 \cdot |\Delta x| - |\Delta y|$$
$$x = \begin{cases} x_0, & ha\ \Delta y \geq 0 \\ x_1, & ha\ \Delta y < 0 \end{cases}, \quad xStep = \begin{cases} 1, & ha\ \Delta x \cdot \Delta y \geq 0 \\ -1, & ha\ \Delta x \cdot \Delta y < 0 \end{cases}$$
$$\textbf{for } y = \begin{cases} y_0 \to y_1, & ha\ \Delta y \geq 0 \\ y_1 \to y_0, & ha\ \Delta y < 0 \end{cases}$$
$$slopeError = slopeError + m$$
ha $slopeError \geq 0$:
$$x = x + xStep$$
$$slopeError = slopeError - 2 \cdot |\Delta y|$$
**endfor**

**3D**

Háromdimenzióban a Bresenham algoritmus a kétdimenzióshoz hasonlóan működik azzal a különséggel, hogy két meredekséget és két, egymástól független meredekségi hibát tart számon. Továbbra is egy ciklus van, amely a három tengely közül a leggyorsabban változó mentén halad. Például amennyiben $|\Delta x| \geq |\Delta y|, |\Delta z|$, akkor a ciklus az $x$ koordináta mentén fut, és a $slopeError_Y$ az egyenesnek az $xy$-síkon, míg a $slopeError_Z$ az $xz$-síkon vett merőleges vetületének a meredekségi hibáját követi nyomon. Voltaképpen háromdimenzióban két független, kétdimenziós Bresenham algoritmus fut egyidejűleg.

A voxel hozzájárulását a ciklus minden iterációjában a megszokott módon számoljuk: $yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z)$, ahol $\Delta l = \frac{\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}}{\max(|\Delta x|, |\Delta y|, |\Delta z|)}$.

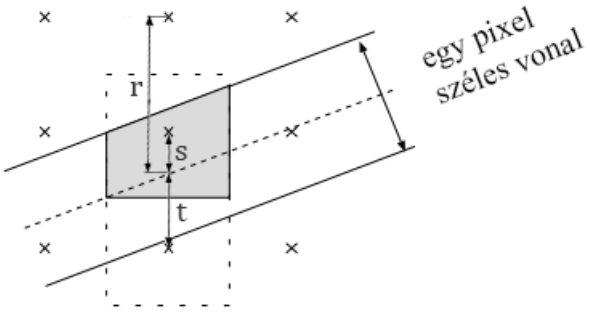
### 5.3.5 Antialiased Bresenham

Az Antialiased Bresenham a Bresenham algoritmus kiegészítése dobozszűréssel. A korábbihoz hasonlóan először ismét kétdimenzióban, 0° és 45° közötti meredekségű egyenesekre ismertetem az eljárást, ezt követően általánosítom tetszőleges meredekségre, majd háromdimenzióra.

**2D, meredekség ∈ (0°,45°)**

Kétdimenziós dobozszűrés esetén a konvolúciós integrálhoz az egy pixel széles vonal és az éppen vizsgált pixelt határoló téglalap metszetének területét kell kiszámolni. Az átfedés területe mint súly szorzótényezőként fog megjelenni a pixel LOR találatszámhoz való hozzájárulásának számításakor.

A 21. ábra alapján látható, hogy minden oszlopban legfeljebb három pixel tudja metszeni a vonalat, amennyiben annak meredeksége 0° és 45° közé esik.




*21. ábra: Vonal dobozszűrése kétdimenzióban.*

Jelölje a három legközelebbi pixel függőleges távolságát a vonal közepétől $r$, $s$ és $t$, és tegyük fel, hogy $s < t \leq r$. Geometriai okokból ekkor $s, t < 1$, $s + t = 1$ és $r \geq 1$ is igaz. Sajnálatos módon a metszet területe, $I_s, I_t$ és $I_r$ nemcsak $r$, $s$ és $t$ értékétől függ, hanem a vonal meredekségétől is. Ez azonban kiküszöbölhető a következő közelítéssel:

$$I_s \approx (1 - s), \qquad I_t \approx (1 - t) = s, \qquad I_r \approx 0$$

Ez csak abban az esetben helytálló, ha a vonal vízszintes, de 0° és 45° közötti meredekség esetén is elfogadható közelítést ad.

Ha az egyenes az $y = m \cdot x + b$ egyenlettel adott, akkor

$$s = m \cdot x + b - \text{Kerekít}(m \cdot x + b),$$

ami tulajdonképpen az egész koordinátával való közelítés hibáját adja meg.

Az Antialiased Bresenham algoritmus az $y$ koordinátát inkrementális elven, Bresenham módszerével számolja. A LOR hozzájárulások az adott oszlopban:

$$yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y) \cdot I_s$$

$$yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y + 1) \cdot I_t$$

Az $I_s$, $I_t$ súlyok is inkrementális elv alapján kerülnek kiszámításra. Amennyiben a legközelebbi pixel $y$ koordinátája nem változik $x$-ről $x + 1$-re lépve, akkor

$$I_s(x + 1) = I_s(x) - m, \qquad I_t(x + 1) = I_t(x) + m$$

Amennyiben $x$-ről $x + 1$-re lépve az $y$ koordinátát is növelni kell:

$$I_s(x + 1) = I_s(x) - m + 1, \qquad I_t(x + 1) = I_t(x) + m - 1.$$

Az algoritmus:

**AntialiasedBresenham**$(\boldsymbol{yLOR}, \boldsymbol{g}, \mathbf{x_0}, \mathbf{y_0}, \mathbf{x_1}, \mathbf{y_1})$:
$\Delta x = x_1 - x_0; \quad \Delta y = y_1 - y_0$
$\Delta l = \sqrt{\Delta x^2 + \Delta y^2}/\Delta x$
$m = 2 \cdot \Delta y$
$slopeError = 2 \cdot \Delta y - \Delta x$
$I_s = 1 + m; \quad I_t = -m$
$y = y_0$
**for** $x = x_0$ amíg $x < x_1$
$slopeError = slopeError + m$
$I_s = I_s - m; \quad I_t = I_t + m$
ha $slopeError \geq 0$:
$y = y + 1$
$slopeError = slopeError - 2 \cdot \Delta x$
$I_s = I_s + 1; \quad I_t = I_t - 1$
$yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y) \cdot I_s$
$yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y + 1) \cdot I_t$
**endfor**
eljárás vége

**2D, tetszőleges meredekség**

Tetszőleges meredekségű egyenesek kezelésénél a Bresenham eljárásnál összefoglalt esetszétválasztás érvényes az Antialiased Bresenhamhoz is. $I_s$, $I_t$ számolását nem befolyásolja $\Delta x$ és $\Delta y$ értéke, azonban a hozzájáruló pixelek koordinátáit igen.

1. $|\Delta x| \geq |\Delta y|$ (ciklus az $x$ tengely mentén):

$$yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y) \cdot I_s$$
$$yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y + yStep) \cdot I_t$$

2. $|\Delta x| < |\Delta y|$ (ciklus az $y$ tengely mentén):

$$yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y) \cdot I_s$$
$$yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x + xStep, y) \cdot I_t$$

**3D**

A háromdimenziós Antialiased Bresenham az eredeti Bresenham eljáráshoz hasonlóan két kétdimenziós algoritmust alkalmaz egyidejűleg. A ciklus a leggyorsabban változó tengely mentén halad, a másik két tengely mentén külön meredekséget, meredekségi hibát és $I_s$, $I_t$ súlyokat (a továbbiakban: az $x$ tengely mentén $IX_s, IX_t$, az $y$ tengely mentén $IY_s, IY_t$ és a $z$ tengely mentén $IZ_s, IZ_t$ súlyokat) tart nyilván. Az egyszerű kétdimenziós esethez képest azonban a voxelek hozzájárulásának a súlyozása eltérően alakul.

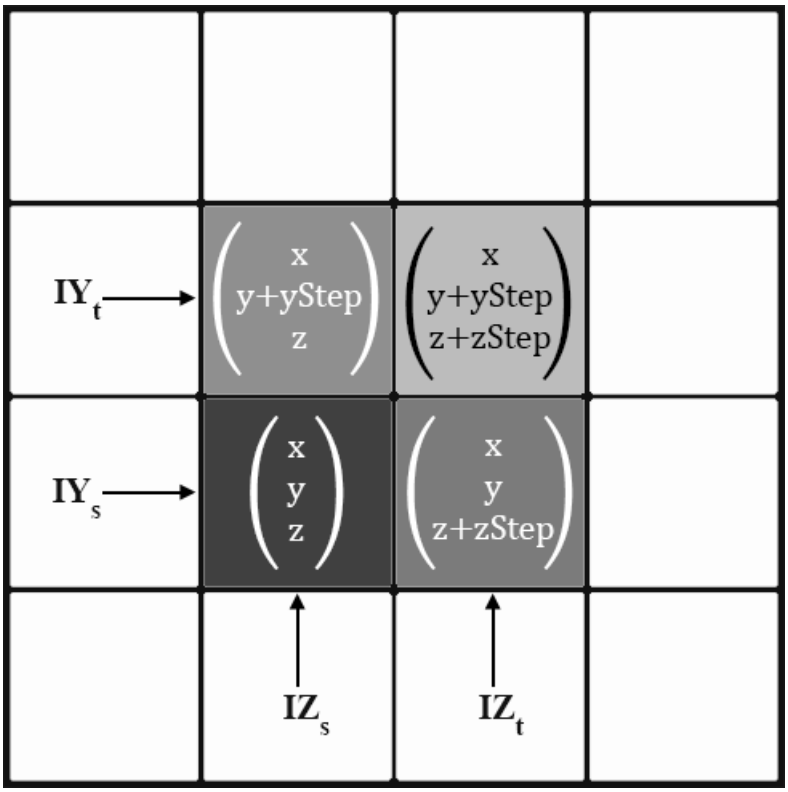
Kétdimenziós esetben oszloponként két pixel járult hozzá a LOR találatszámhoz. Háromdimenzióban két tengelyünk is van, amely mentén az oszlop fogalmát értelmezhetjük. Tegyük fel, hogy a ciklus az $x$ koordinátát lépteti. Ekkor ha egyszerűen az $xy$ és $xz$

kétdimenziós algoritmusokat egymás mellett alkalmaznánk, a következő voxel hozzájárulásokat kapnánk egy adott ciklus iterációban:

$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z) \cdot IY_s$$
$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y + yStep, z) \cdot IY_t$$

$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z) \cdot IZ_s$$
$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z + zStep) \cdot IZ_t$$

Ez három voxel aktivitását veszi figyelembe: az $(x, y, z)$, az $(x, y + yStep, z)$ és az $(x, y, z + zStep)$ koordinátákon lévőkét. Valódi háromdimenziós antialiasinghoz azonban szükség van egy negyedik voxel, az $(x, y + yStep, z + zStep)$ aktivitására is.




*22. ábra: Háromdimenziós Antialiased Bresenham súlyozása.*

A voxel hozzájárulások:

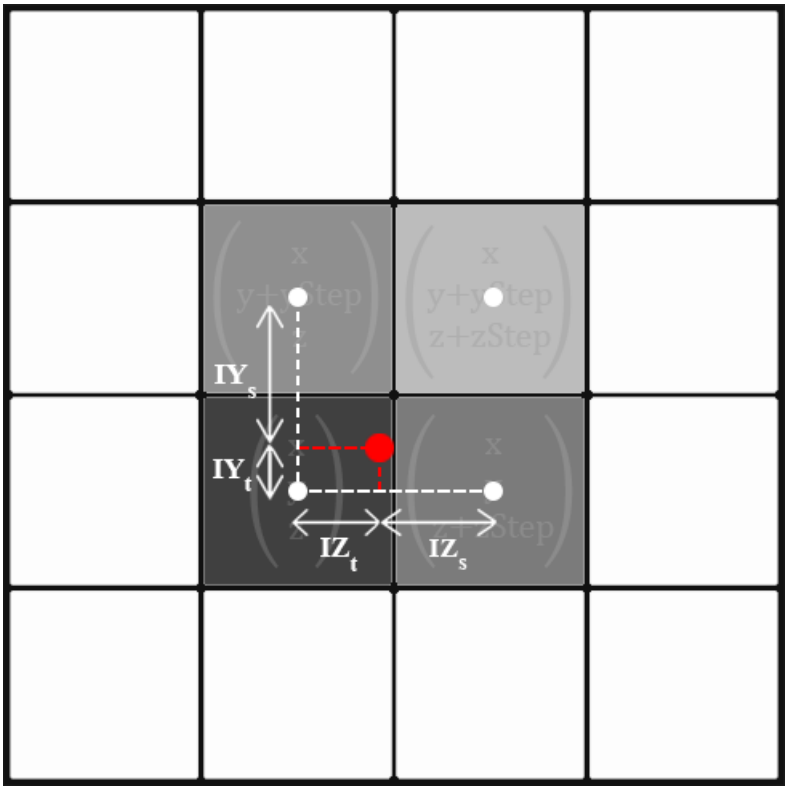
$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z) \cdot IY_s \cdot IZ_s$$
$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y + yStep, z) \cdot IY_t \cdot IZ_s$$
$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z + zStep) \cdot IY_s \cdot IZ_t$$
$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y + yStep, z + zStep) \cdot IY_t \cdot IZ_t$$

Belátható, hogy ez a súlyozás helyes. Az $IY_t = 1 - IY_s$ és az $IZ_t = 1 - IZ_s$ összefüggések miatt a súlyok összege egy:

$$IY_s \cdot IZ_s + IY_t \cdot IZ_s + IY_s \cdot IZ_t + IY_t \cdot IZ_t$$
$$= IY_s \cdot IZ_s + (1 - IY_s) \cdot IZ_s + IY_s \cdot (1 - IZ_s) + (1 - IY_s) \cdot (1 - IZ_s)$$
$$= IY_s \cdot IZ_s + IZ_s - IY_s \cdot IZ_s + IY_s - IY_s \cdot IZ_s + 1 - IY_s - IZ_s + IY_s \cdot IZ_s = 1.$$

Másrészt a legnagyobb súlyt, $IY_s \cdot IZ_s$-t a középső, az egyeneshez legközelebb eső voxel kapja ($IY_s \geq IY_t, IZ_s \geq IZ_t$), míg a legkisebbet az egyenestől legtávolabbi $(x, y + yStep, z + zStep)$ voxel.

A CUDA tesztprogramban a memóriaelérési kérések számát a textúra memória beépített hardveres lineáris interpolációjával csökkentem. Míg a CPU implementációban minden ciklus iterációban négy voxel középpontjából veszek mintát, addig a CUDA programban egyetlen mintát veszek, melyet a négy voxelközéppont kétdimenziós lineáris interpolációjával állítok elő. Ehhez mindkét tengely mentén az $I_s, I_t$ súlyokat használom fel az ábrán látható módon:




*23. ábra: Lineáris interpoláció az Antialiased Bresenham algoritmusnál.*

Voxelhozzájárulások számítása:

$$y_{interpolated} = y \cdot IY_s + (y + yStep) \cdot IY_t \quad \text{// valós szám!}$$

$$z_{interpolated} = z \cdot IZ_s + (z + zStep) \cdot IZ_t \quad \text{// valós szám!}$$

$$yLOR = yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y_{interpolated}, z_{interpolated})$$

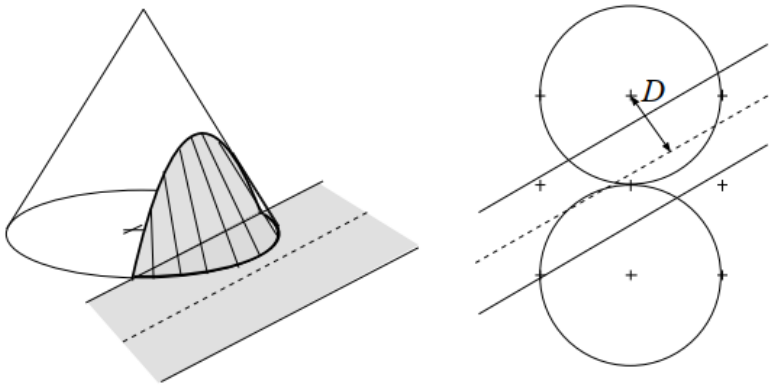
Látható, hogy a memóriaelérési kérések száma négyről egyre csökkent. Ez kombinálva a textúra memória térbeli lokalitásra optimalizált gyorsítótárával jelentős (~tízszeres) sebességbeli növekedést eredményez.

### 5.3.6 Gupta-Sproull kúpszűrés

Nézzük elsőként kétdimenzióban, amikor a meredekség 0˚ és 45˚ közé esik.

**2D, meredekség ∈ (0˚,45˚)**

Vegyünk egy kúpot, melynek alaplapja egy egységsugarú körlap, és magassága $\frac{3}{\pi}$. (A kúp magasságát úgy választottuk meg, hogy a teljes térfogata egy legyen.) Egy egységnyi széles vonal kúpszűrésénél a konvolúció úgy szemléltethető, hogy végigvesszük az egyes pixeleket, az éppen vizsgált pixelre ráhelyezzük a kúpot, és meghatározzuk a kúpnak a vonal fölé eső térfogatrészét. A térfogat értéke mint súly szorzótényezőként fog megjelenni a pixel LOR találatszámhoz való hozzájárulásának számításakor. A 24. ábra alapján látható, hogy minden oszlopban legfeljebb három pixelnek az egységsugarú körlapja tudja metszeni a vonalat, amennyiben a meredekség 0˚ és 45˚ közé esik.




*24. ábra: Vonal kúpszűrése.*

Jelölje $D$ a pixel középpontjának a távolságát a vonal közepétől. Akkor metszheti a kúp alaplapja a vonalat, ha $D$ a $[-1.5; 1.5]$ intervallumba esik. Tetszőleges $(X, Y)$ középpontú pixel esetén a konvolúciós integrál – azaz a kúpnak a vonal fölé eső térfogatrésze – csak $D$ értékétől függ, így előre kiszámolható különböző diszkrét $D$ értékekre már az algoritmus tervezési fázisában. Az eredményt egy $V[D]$ keresőtáblázatban tároljuk, amelyből adott $D$-re a konvolúciós integrál értéke a következő algoritmus alapján kapható meg:

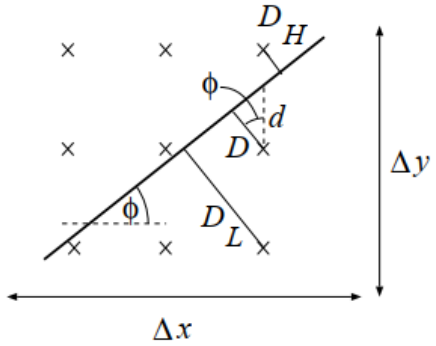
**GetV(D)**:
$index = \frac{|D|}{1.5} \cdot nTableEntries$ //nTableEntries: a táblázatbejegyzések száma
ha $index > nTableEntries$:
$index = nTableEntries$
visszatérési érték: $V[index]$
**eljárás vége**

$nTableEntries = 100$ esetén a $V[D]$ keresőtáblázat hat tizedesjegyre kerekített értéke:

```
V[100] = { 0.779611, 0.780068, 0.778471, 0.777074, 0.775480, 0.772634, 0.769064,
0.765338, 0.761834, 0.757069, 0.750989, 0.745603, 0.738822, 0.731373, 0.723630,
0.716200, 0.708056, 0.698284, 0.688795, 0.679074, 0.668878, 0.657882, 0.645989,
```

```
0.634359,  0.622820,  0.609960,  0.597614,  0.584235,  0.570894,  0.556573,  0.542793,
0.528724,  0.514229,  0.499696,  0.485269,  0.470819,  0.456624,  0.442645,  0.428078,
0.414039,  0.399691,  0.386726,  0.372428,  0.359391,  0.345845,  0.332673,  0.319609,
0.306868,  0.294111,  0.281776,  0.269968,  0.257704,  0.245980,  0.234642,  0.223675,
0.212678,  0.201874,  0.191307,  0.181217,  0.171512,  0.161696,  0.152167,  0.143471,
0.134558,  0.126016,  0.117941,  0.110082,  0.102352,  0.095254,  0.087961,  0.081290,
0.074923,  0.068843,  0.062911,  0.057296,  0.051900,  0.046869,  0.042072,  0.037666,
0.033540,  0.029608,  0.025947,  0.022621,  0.019522,  0.016688,  0.014089,  0.011796,
0.009660,  0.007827,  0.006205,  0.004794,  0.003574,  0.002565,  0.001753,  0.001113,
0.000645, 0.000313, 0.000114, 0.000020, 0.000000 };
```

A továbbiakban $D$ és a következő pixel koordinátáinak kiszámolásáról lesz szó. Gupta és Sproull [50] a Bresenham algoritmust javasolták a pixelkoordináták előállítására, és ennek kiegészítéseként egy iteratív módszert vezettek be $D$ kiszámítására.




*25. ábra: A D távolság inkrementális számolása.*

Jelölje az egyenes meredekségét $\phi$ ($0° \leq \phi \leq 45°$), a legközelebbi pixel és a vonal közepének függőleges távolságát pedig $d$. A 25. ábra alapján belátható, hogy a három függőlegesen elhelyezkedő pixel $D$ értékei:

$$D = d \cdot \cos\phi = \frac{d \cdot \Delta x}{\sqrt{(\Delta x)^2 + (\Delta y)^2}},$$

$$D_H = (1 - d) \cdot \cos\phi = -D + \frac{\Delta x}{\sqrt{(\Delta x)^2 + (\Delta y)^2}},$$

$$D_L = (1 + d) \cdot \cos\phi = D + \frac{\Delta x}{\sqrt{(\Delta x)^2 + (\Delta y)^2}}.$$

*8. egyenlet*

A $d$ távolság és a Bresenham algoritmus $slopeError$ hibaváltozója (lásd 5.3.4 fejezet) között közvetlen összefüggés fedezhető fel. Bresenham módszerének kezdeti, valós értékű hibaváltozója – jelölje $slopeError_0$ – a függőleges távolság plusz 0.5, hogy a kerekítés műveletét egy egyszerű levágással lehessen helyettesíteni. Ez alapján $d = slopeError_0 - 0.5$, ha $slopeError_0$ nem csordul túl (értsd: $slopeError_0$ nem nő 0.5 fölé, azaz $y$-t nem kell inkrementálni), ellenkező esetben $d = slopeError_0 - 1.5$.

$slopError_0$-ból a Bresenham algoritmus $slopeError$ egész értékű hibaváltozója a következő módon kapható meg: $slopeError = 2\Delta x \cdot (slopeError_0 - 1)$. Emiatt amennyiben a meredekségi hiba nem csordul túl:

$$slopeError = 2\Delta x \cdot (slopeError_0 - 1) = 2\Delta x \cdot (d - 0{,}5)$$

$$\rightarrow \qquad 2d \cdot \Delta x = slopeError + \Delta x.$$

Amennyiben túlcsordul:

$$slopeError = 2\Delta x \cdot (slopeError_0 - 1) = 2\Delta x \cdot (d + 0{,}5)$$

$$\rightarrow \qquad 2d \cdot \Delta x = slopeError - \Delta x.$$

Ez $2d \cdot \Delta x$ inkrementális számolását teszi lehetővé, így a 8. egyenletekben szereplő törteket bővíteni kell kettővel.

Az osztáshoz és a négyzetgyökvonáshoz hasonló bonyolult műveleteket az egész vonal számára egyszer kell kiszámolni, ezáltal az eljárás pixel szinten csupán egyszerű utasításokat és egy szorzást tartalmaz, eltekintve a LOR hozzájárulás számításától. A komplex kifejezéseket a következő változókban tároljuk:

$$denom = \frac{1}{2\sqrt{(\Delta x)^2+(\Delta y)^2}}, \qquad \Delta D = \frac{2\Delta x}{2\sqrt{(\Delta x)^2+(\Delta y)^2}}.$$

Minden ciklusiterációban $nume = 2d \cdot \Delta x$ értékét az inkrementális képletek határozzák meg. A teljes algoritmus:

**GuptaSproull**$(\mathbf{x_0}, \mathbf{y_0}, \mathbf{x_1}, \mathbf{y_1})$:

$\Delta x = x_1 - x_0; \quad \Delta y = y_1 - y_0$
$\Delta l = \sqrt{\Delta x^2 + \Delta y^2}/\Delta x$
$m = 2 \cdot \Delta y$
$slopeError = 2 \cdot \Delta y - \Delta x$
$denom = \frac{1}{2\sqrt{(\Delta x)^2+(\Delta y)^2}}$
$\Delta D = 2 \cdot \Delta x \cdot denom$
$y = y_0$
**for** $x = x_0$ amíg $x < x_1$
    ha $slopeError < 0$:
        $nume = slopeError + \Delta x$
        $slopeError = slopeError + m$
    egyébként:
        $y = y + 1$
        $nume = slopeError - \Delta x$
        $slopeError = slopeError + m - 2 \cdot \Delta x$
    $D = nume \cdot denom$
    $D_H = -D + \Delta D; \quad D_L = D + \Delta D$
    $yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y) \quad \cdot GetV(D)$
    $yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y+1) \cdot GetV(D_H)$
    $yLOR = yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y-1) \cdot GetV(D_L)$
**endfor**
**eljárás vége**

**2D, tetszőleges meredekség**

Tetszőleges meredekségű egyenesek kezelésénél a Bresenham eljárásnál összefoglalt esetszétválasztás érvényes a Gupta-Sproull algoritmushoz is. A távolságok számítása során csupán $\Delta D$ értéke függ attól, hogy melyik tengely mentén mintavételezünk, illetve a LOR találatszámhoz hozzájáruló pixelek koordinátái is ennek függvényében határozhatók meg.

3. $|\Delta x| \geq |\Delta y|$ (ciklus az $x$ tengely mentén):

$$\Delta D = 2 \cdot |\Delta x| \cdot denom$$

$$\begin{aligned} yLOR &= yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y) \cdot GetV(D) \\ yLOR &= yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y + yStep) \cdot GetV(D_H) \\ yLOR &= yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y - yStep) \cdot GetV(D_L) \end{aligned}$$

4. $|\Delta x| < |\Delta y|$ (ciklus az $y$ tengely mentén):

$$\Delta D = 2 \cdot |\Delta y| \cdot denom$$

$$\begin{aligned} yLOR &= yLOR + g \cdot \Delta l \cdot GetPixelValue(x, y) \cdot GetV(D) \\ yLOR &= yLOR + g \cdot \Delta l \cdot GetPixelValue(x + xStep, y) \cdot GetV(D_H) \\ yLOR &= yLOR + g \cdot \Delta l \cdot GetPixelValue(x - xStep, y) \cdot GetV(D_L) \end{aligned}$$

**3D**

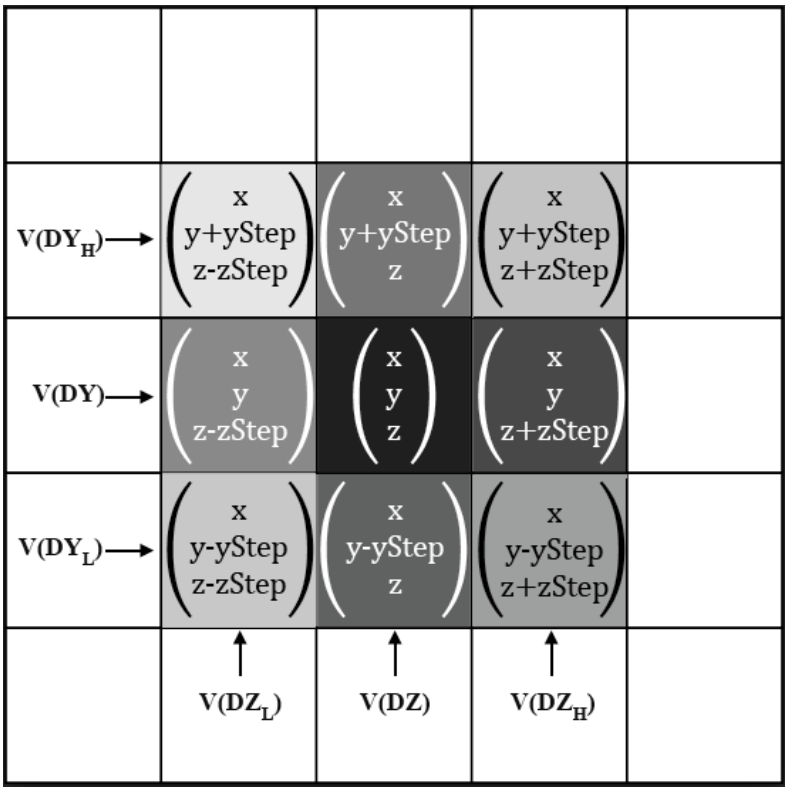
A háromdimenziós Gupta-Sproull eljárás a Bresenhamnál látott gondolatmenet alapján két kétdimenziós algoritmusból épül fel. A ciklus a leggyorsabban változó tengely mentén halad. A másik két tengely mentén külön meredekséget, meredekségi hibát és $D, D_H, D_L$ távolságokat tart nyilván.

Kétdimenziós esetben oszloponként három pixel járult hozzá a LOR találatszámhoz, azonban háromdimenzióban két tengelyünk is van, amely mentén az oszlop fogalmát értelmezhetjük. Tegyük fel, hogy a ciklus az $x$ irányban halad. Ekkor ha egyszerűen az $xy$ és $xz$ kétdimenziós algoritmusokat alkalmaznánk egymás mellett, a következő voxel hozzájárulásokat kapnánk egy adott ciklus iterációban:

$$\begin{aligned} yLOR &= yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z) \cdot GetV(DY) \\ yLOR &= yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y + yStep, z) \cdot GetV(DY_H) \\ yLOR &= yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y - yStep, z) \cdot GetV(DY_L) \end{aligned}$$

$$\begin{aligned} yLOR &= yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z) \cdot GetV(DZ) \\ yLOR &= yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z + zStep) \cdot GetV(DZ_H) \\ yLOR &= yLOR + g \cdot \Delta l \cdot GetVoxelValue(x, y, z - zStep) \cdot GetV(DZ_L) \end{aligned}$$

Ez összesen öt voxel aktivitását veszi figyelembe. Valódi háromdimenziós antialiasinghoz azonban szükség van a középső voxellel csak sarkosan érintkező voxelek aktivitására is.




*26. ábra: Háromdimenziós Gupta-Sproull súlyozása, ha az x tengely mentén mintavételezünk.*

A rekonstrukció pontossága megköveteli, hogy ha egy adott ciklusiterációban a mintavételezett kilenc voxel súlyait ($V(DY), V(DY_H)$, ...) összeadjuk, egyet kapjunk eredményül. Tehát valójában a középső, az egyenes egy pontját reprezentáló (ahhoz az adott iterációban legközelebb eső) voxelhez tartozó 1 súlyt szeretnénk szétosztani fontosság szerint annak közvetlen környezetében. Az Antialiased Bresenhamnál ez a kritérium mindig teljesült az $I_t = 1 - I_s$ összefüggés miatt. Kúpszűrésnél azonban nincs ilyen szimmetrikus összefüggés a $V(D), V(D_H)$ és $V(D_L)$ súlyok között (ezek összege nemcsak, hogy nagyobb, mint egy, de iterációról iterációra változik), így másképp kell biztosítani, hogy a mintavételezett kilenc voxel összsúlya egy legyen. (Megjegyzendő, hogy ez a probléma a számítógépes grafikában, vonalrajzoláskor nem jelentkezik, hiszen ott természetesen senki nem fogja azt vizsgálni, hogy egy oszlop mentén mennyi a pixelek színértékeinek összege. Vizuálisan ez úgy jelenik meg, hogy a kúpszűréssel kirajzolt egyenes vastagabb és erősebb hatást kelt, mint a dobozszűréssel rajzolt, lásd 15. ábra.)

Annak biztosítása érdekében, hogy az összsúly mindig egy legyen, bevezettem egy $n$ normalizációs paramétert, melynek pontos értéke a (normalizáció nélküli) összsúly reciproka.

$$n = 1/(GetV(DY) \cdot GetV(DZ) + GetV(DY_H) \cdot GetV(DZ) + GetV(DY_L) \cdot GetV(DZ) + GetV(DY) \cdot GetV(DZ_H) + GetV(DY) \cdot GetV(DZ_L) + GetV(DY_H) \cdot GetV(DZ_H) + GetV(DY_H) \cdot GetV(DZ_L) + GetV(DY_L) \cdot GetV(DZ_H) + GetV(DY_L) \cdot GetV(DZ_L))$$

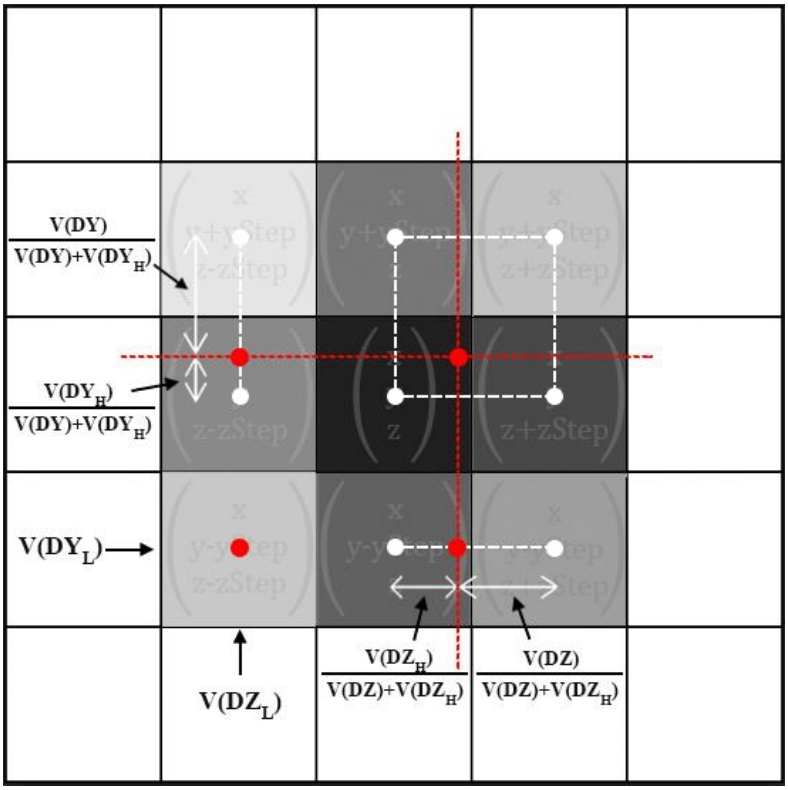
A súlyozás ennek megfelelően:

$$yLOR += g \cdot \Delta l \cdot GetVoxelValue(x, y, z) \cdot GetV(DY) \cdot GetV(DZ) \cdot n$$
$$yLOR += g \cdot \Delta l \cdot GetVoxelValue(x, y + yStep, z) \cdot GetV(DY_H) \cdot GetV(DZ) \cdot n$$
$$yLOR += g \cdot \Delta l \cdot GetVoxelValue(x, y - yStep, z) \cdot GetV(DY_L) \cdot GetV(DZ) \cdot n$$

$$yLOR+= g \cdot \Delta l \cdot GetVoxelValue(x, y, \quad z + zStep) \cdot GetV(DY) \cdot GetV(DZ_H) \cdot n$$
$$yLOR+= g \cdot \Delta l \cdot GetVoxelValue(x, y, \quad z - zStep) \cdot GetV(DY) \cdot GetV(DZ_L) \cdot n$$
$$yLOR+= g \cdot \Delta l \cdot GetVoxelValue(x, y + yStep, z + zStep) \cdot GetV(DY_H) \cdot GetV(DZ_H) \cdot n$$
$$yLOR+= g \cdot \Delta l \cdot GetVoxelValue(x, y + yStep, z - zStep) \cdot GetV(DY_H) \cdot GetV(DZ_L) \cdot n$$
$$yLOR+= g \cdot \Delta l \cdot GetVoxelValue(x, y - yStep, z + zStep) \cdot GetV(DY_L) \cdot GetV(DZ_H) \cdot n$$
$$yLOR+= g \cdot \Delta l \cdot GetVoxelValue(x, y - yStep, z - zStep) \cdot GetV(DY_L) \cdot GetV(DZ_L) \cdot n$$

Mivel $V(DY) > V(DY_H), V(DY_Z)$ és $V(DZ) > V(DZ_H), V(DZ_L)$, ezért az továbbra is igaz, hogy a legnagyobb súlyt, $V(DY) \cdot V(DZ)$-t az egyeneshez legközelebb eső, középső voxel kapja, míg a legkisebbeket az egyenestől legtávolabbi, a középső voxellel csak sarkosan érintkező voxelek.

A CUDA tesztprogramban a beépített lineáris iterpolációt kihasználva csökkentem a memóriaelérési kérések számát. A voxeleket a következőképpen csoportosítottam:




*27. ábra: Voxelek csoportosítása a Gupta-Sproull algoritmusnál.*

A jobb felső négy voxelt az AntialiasedBresenhamnál is alkalmazott kétdimenziós interpolációval egy memóriaeléréssé vontam össze. A maradék öt voxelből két-két párt alakítottam ki, melyek között egydimenziós interpolációt hajtok végre, és az utolsó (bal alsó) voxel önmagában maradt (itt egyszerűen a voxelközéppont értékét olvasom ki).

Voxelhozzájárulások számítása:

$$weightSumY = V(DY) + V(DY_H)$$
$$weightSumZ = V(DZ) + V(DZ_H)$$

$$y_{interp.} = y \cdot \frac{V(DY_H)}{weightSumY} + (y + yStep) \cdot \frac{V(DY)}{weightSumY}$$

$$z_{interp.} = z \cdot \frac{V(DZ_H)}{weightSumZ} + (z + zStep) \cdot \frac{V(DZ)}{weightSumZ}$$

$$yLOR += g \cdot \Delta l \cdot GetVoxelValue\left(x, y_{interp.}, z_{interp.}\right) \cdot weightSumY \cdot weightSumZ \cdot n$$

$$yLOR += g \cdot \Delta l \cdot GetVoxelValue(x, y - yStep, z - yStep) \cdot V(DY_L) \cdot V(DZ_L) \cdot n$$

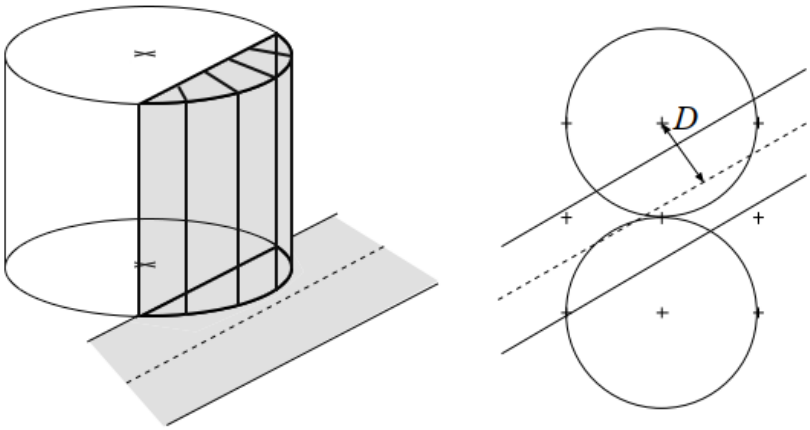
$$yLOR += g \cdot \Delta l \cdot GetVoxelValue\left(x, y_{interp.}, z + zStep\right) \cdot V(DZ_L) \cdot weightSumY \cdot n$$

$$yLOR += g \cdot \Delta l \cdot GetVoxelValue\left(x, y, z_{interp.}\right) \cdot V(DY_L) \cdot weightSumZ \cdot n$$

A memóriaelérési kérések száma kilencről négyre, hozzávetőleg a felére csökkent.

### 5.3.7 Hengeres Gupta-Sproull

A Gupta-Sproull algoritmus egy variációja lehet, ha kúp helyett hengert helyezünk a pixelekre, és a hengernek számoljuk ki az átfedő terület fölé eső térfogatrészét. Az algoritmus menete nem változik, csupán a $V[D]$ keresőtáblázat értékei módosulnak.




*28. ábra: Vonal hengerszűrése*

A keresőtáblázat hat tizedesjegyre kerekített értéke:

```
V[100] = { 0.608981, 0.609148, 0.608577, 0.608096, 0.607739, 0.606863, 0.605569,
0.604580,  0.603754,  0.602461,  0.600130,  0.598860,  0.596557,  0.594081,  0.591904,
0.589337,  0.587080,  0.583774,  0.580292,  0.577269,  0.573791,  0.570020,  0.565649,
0.561308,  0.556939,  0.552125,  0.547197,  0.541675,  0.535916,  0.529702,  0.522970,
0.515914,  0.508406,  0.499786,  0.490115,  0.480450,  0.471318,  0.461469,  0.451876,
0.442072,  0.432333,  0.423452,  0.413286,  0.404130,  0.394452,  0.384808,  0.375633,
0.366176,  0.356579,  0.347120,  0.338128,  0.328482,  0.319167,  0.310118,  0.301002,
0.291949,  0.282531,  0.273498,  0.264645,  0.255830,  0.246694,  0.237853,  0.229623,
0.220861,  0.212326,  0.203783,  0.195452,  0.187001,  0.179225,  0.170712,  0.162787,
0.154931,  0.147267,  0.139379,  0.131766,  0.124216,  0.116763,  0.109575,  0.102466,
0.095551,  0.088617,  0.081716,  0.075364,  0.068943,  0.062753,  0.056646,  0.051010,
0.045192,  0.039860,  0.034684,  0.029669,  0.024882,  0.020418,  0.016175,  0.012385,
0.008900, 0.005756, 0.003151, 0.001118, 0.000000 };
```

## 5.4 Eredmények

A mérések során a voxeltömb mérete a gyakorlatban alkalmazott arányokhoz lett igazítva. A detektorkristályok 8×8 voxeles felülettel rendelkeznek. A LOR-t alkotó két kristály távolságának pedig 128, 256, illetve 512 voxelt választottam, és minden mérettel két mérést végeztem. A voxeltömbön belül előre meghatározott számban egyenletes eloszlással véletlen voxelek kerültek kijelölésre, melyek 1000 aktivitásértéket kaptak. (A többi voxel aktivitása zérus, bennük nem keletkeztek pozitronok.) A vizsgált paraméter ezzel kapcsolatban az, hogy a voxeltömb teljes méretének hány százalékát töltik ki az aktív voxelek. Az alkalmazott tömbméret és az aktív voxel kitöltöttségi százalék alapján hat mérést végeztem el (1. táblázat).

| **Mérés sorszáma** | **Voxeltömbméret [db voxel] (Szélesség×Magasság×Mélység)** | **Voxelek hány százaléka aktív** |
|---|---|---|
| 1 | 8×128×8 | 25% |
| 2 | 8×128×8 | 50% |
| 3 | 8×256×8 | 12.5% |
| 4 | 8×256×8 | 75% |
| 5 | 8×512×8 | 50% |
| 6 | 8×512×8 | 0% |

*1. táblázat: CPU program tesztparaméterek.*

### 5.4.1 Pontosság

Első, CPU-ra írt programomban az ismertetett algoritmusok pontosságát vizsgáltam. Minden mérés alapjául az $yLOR$ találatszámnak előre kiszámolt referenciaértéke szolgált, melyet ray marching segítségével, magas precizitással ($10^7$ különböző $(\vec{z}_1, \vec{z}_2)$ pontpárral mintavételezve) határoztam meg. A mérések során azt vizsgálom, hogy az egyes algoritmusok által, 128 $(\vec{z}_1, \vec{z}_2)$ minta után kapott $yLOR$ találatszámérték referenciától való eltérése a referenciaértéknek hány százaléka ($L_1$ *hiba*). A véletlen mintavételezés következtében jelentkező zaj hatásának minimalizálására pedig minden eljárást egymás után 20 000-szer hajtok végre, és a kapott $yLOR$ értékeknek az átlagát veszem.

Az eredményeket a gnuplot [55] függvényrajzoló program segítségével ábrázoltam. A grafikonokon a hét vizsgált algoritmus referenciaértékhez viszonyított relatív hibája látható a mintaszám függvényében. Külön szeretném felhívni rá a figyelmet, hogy a grafikonok tengelyei logaritmikusan skálázottak, mivel a relatív hiba a mintaszám logaritmikus függvénye. Ez a fajta skálázás lehetővé teszi az egyes algoritmusok hibagörbéinek könnyebb megkülönböztetését.

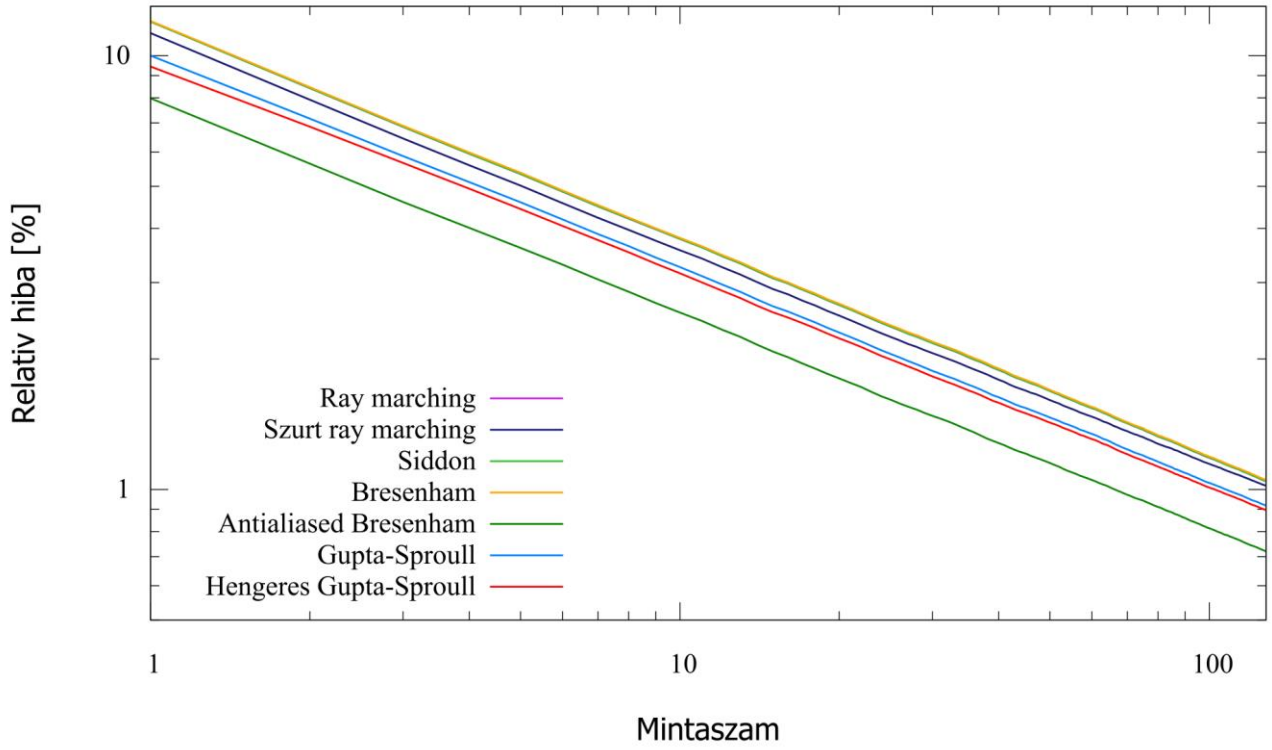
A grafikonokon kívül a 128 minta után kapott hibaszázalékokat táblázatos formában is összefoglaltam, melynek alapján minden mérés után pontossági sorrendet állítottam fel az eljárások között.

**1. mérés**

Voxeltömbméret: 8×128×8 voxel (összesen 8 192 voxel)

Aktív voxel kitöltöttség: 25% (összesen 2 048 voxel)

$outside = 0$



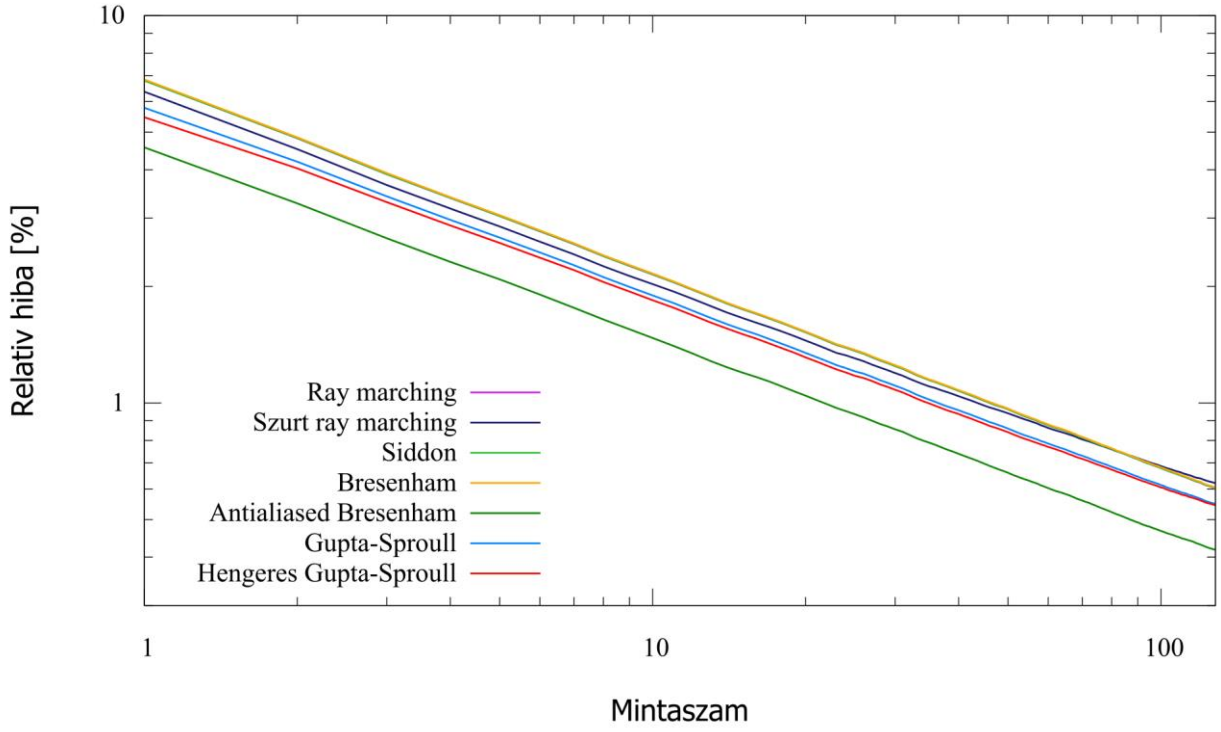

| **Rangsor** | **Algoritmus** | **Relatív hiba 128 minta után** |
|---|---|---|
| 1. | Antialiased Bresenham | 0.7201% |
| 2. | Hengeres Gupta-Sproull | 0.8962% |
| 3. | Gupta-Sproull | 0.9177% |
| 4. | Szűrt ray marching | 1.0190% |
| 5. | Siddon | 1.0437% |
| 6. | Ray marching | 1.0439% |
| 7. | Bresenham | 1.0513% |

**2. mérés**

Voxeltömbméret: 8×128×8 voxel (összesen 8 192 voxel)

Aktív voxel kitöltöttség: 50% (összesen 4 096 voxel)

$outside = 0$



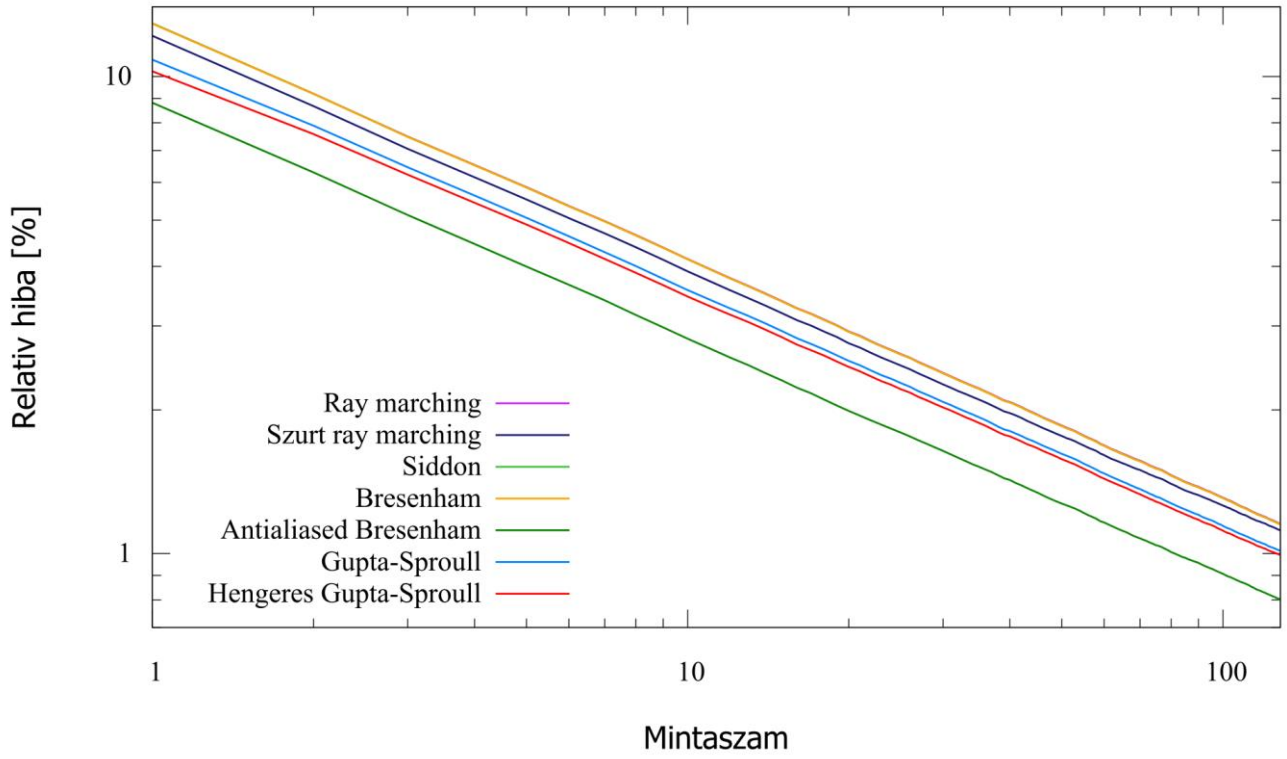

| **Rangsor** | **Algoritmus** | **Relatív hiba 128 minta után [%]** |
|---|---|---|
| 1. | Antialiased Bresenham | 0.4178% |
| 2. | Hengeres Gupta-Sproull | 0.5444% |
| 3. | Gupta-Sproull | 0.5490% |
| 4. | Ray marching | 0.6022% |
| 5. | Siddon | 0.6028% |
| 6. | Bresenham | 0.6053% |
| 7. | Szűrt ray marching | 0.6208% |

### 3. mérés

Voxeltömbméret: 8×256×8 voxel (összesen 16 384 voxel)

Aktív voxel kitöltöttség: 12.5% (összesen 2 048 voxel)

$outside = 0$



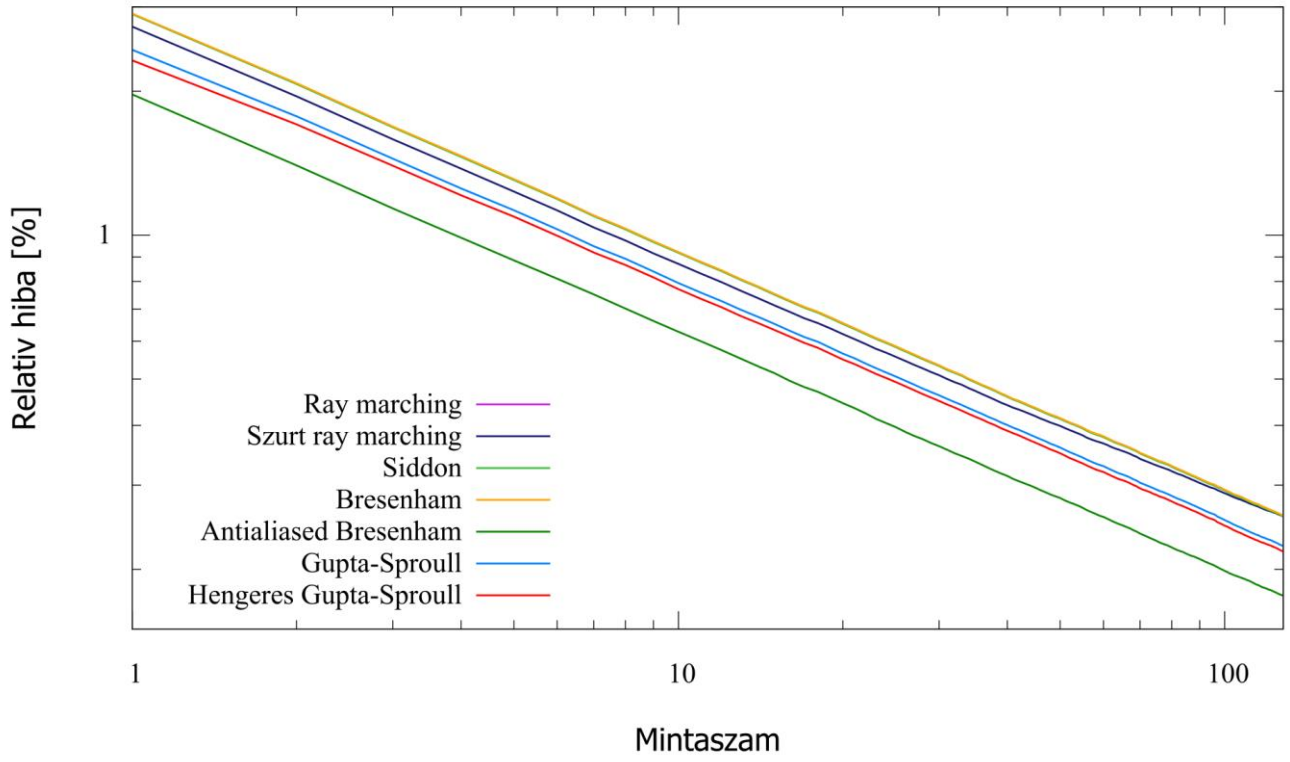

| **Rangsor** | **Algoritmus** | **Relatív hiba 128 minta után** |
|---|---|---|
| 1. | Antialiased Bresenham | 0.8004% |
| 2. | Hengeres Gupta-Sproull | 0.9927% |
| 3. | Gupta-Sproull | 1.0122% |
| 4. | Szűrt ray marching | 1.1180% |
| 5. | Siddon | 1.1507% |
| 6. | Bresenham | 1.1508% |
| 7. | Ray marching | 1.1534% |

**4. mérés**

Voxeltömbméret: 8×256×8 voxel (összesen 16384 voxel)

Aktív voxel kitöltöttség: 75% (összesen 12288 voxel)

$outside = 0$



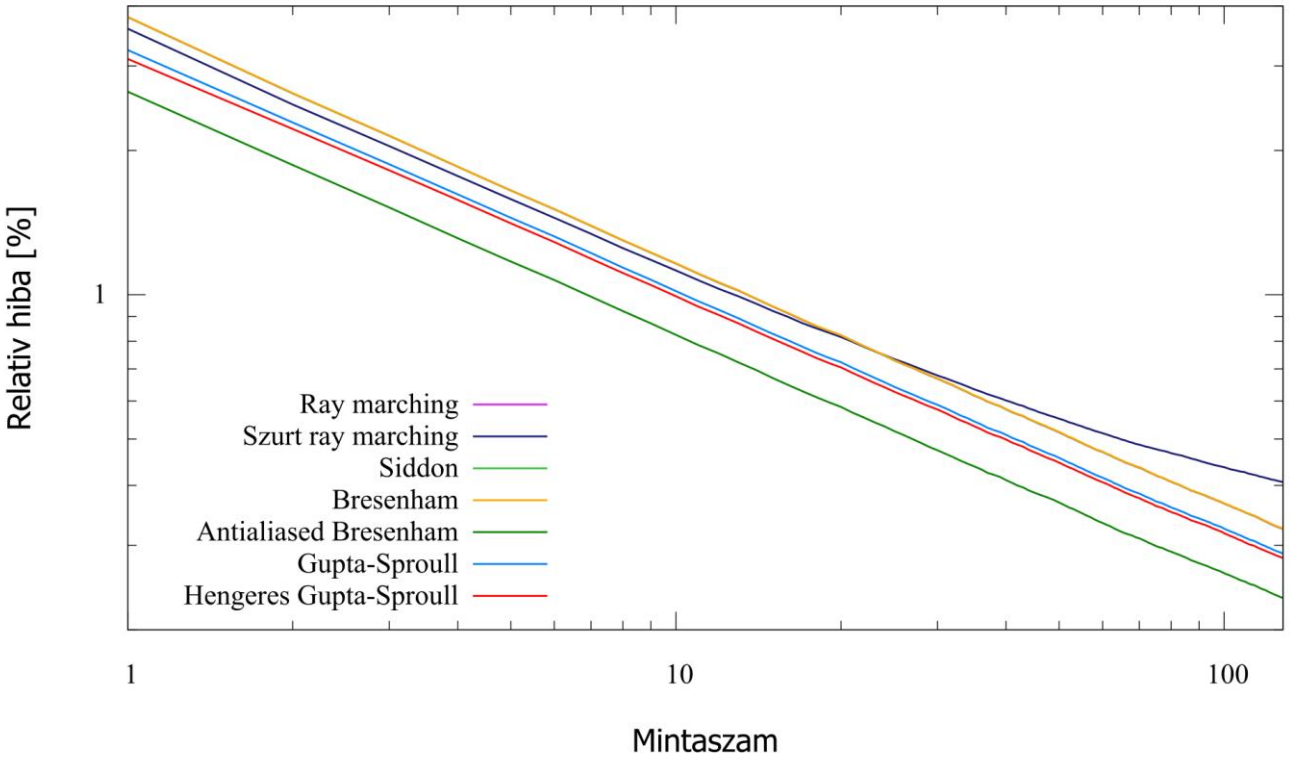

| **Rangsor** | **Algoritmus** | **Relatív hiba 128 minta után** |
|---|---|---|
| 1. | Antialiased Bresenham | 0.1762% |
| 2. | Hengeres Gupta-Sproull | 0.2181% |
| 3. | Gupta-Sproull | 0.2238% |
| 4. | Siddon | 0.2583% |
| 5. | Szűrt ray marching | 0.2583% |
| 6. | Ray marching | 0.2589% |
| 7. | Bresenham | 0.2594% |

### 5. mérés

Voxeltömbméret: 8×512×8 voxel (összesen 32 768 voxel)

Aktív voxel kitöltöttség: 50% (összesen 16 384 voxel)

$outside = 0$



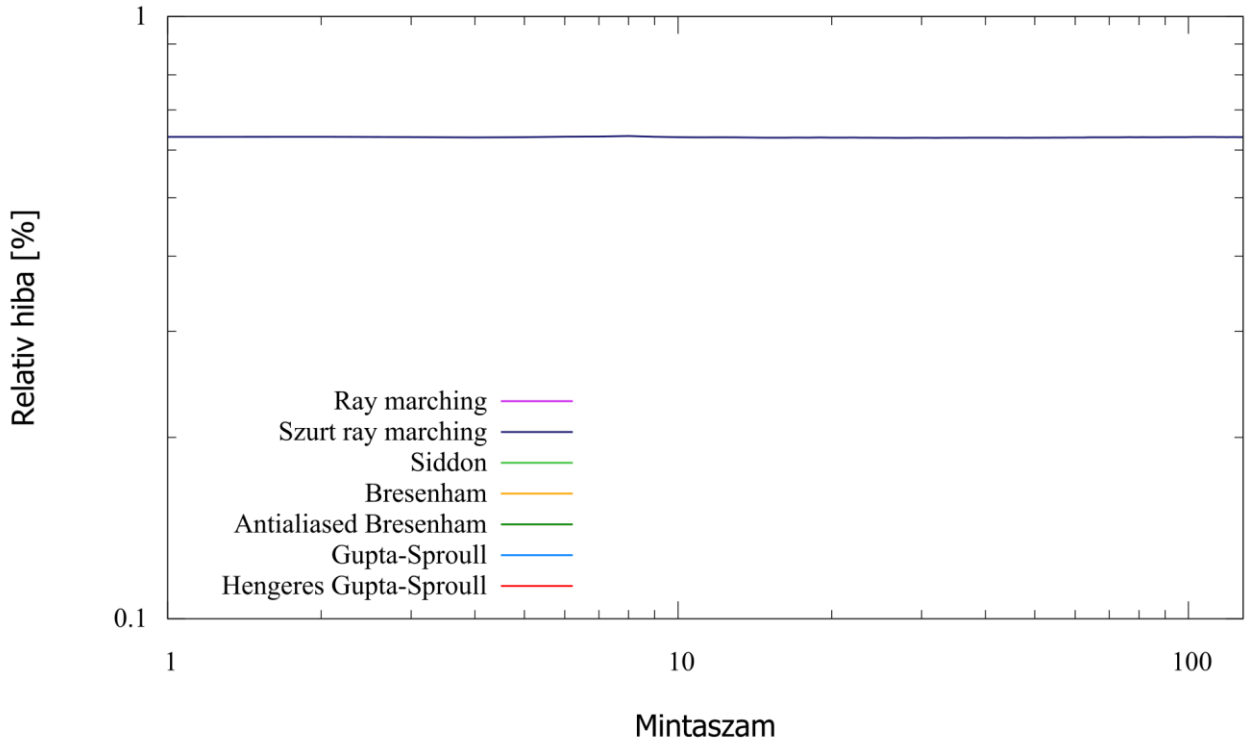

| Rangsor | Algoritmus | Relatív hiba 128 minta után |
|---|---|---|
| 1. | Antialiased Bresenham | 0.2329% |
| 2. | Hengeres Gupta-Sproull | 0.2822% |
| 3. | Gupta-Sproull | 0.2883% |
| 4. | Ray marching | 0.3242% |
| 5. | Siddon | 0.3246% |
| 6. | Bresenham | 0.3250% |
| 7. | Szűrt ray marching | 0.4063% |

### 6. mérés

Voxeltömbméret: 8×512×8 voxel (összesen 32 768 voxel)

Aktív voxel kitöltöttség: 0% (összesen 0 voxel)

$outside = 1\,000$




| Rangsor | Algoritmus | Relatív hiba 128 minta után |
|---|---|---|
| 1-6. | Antialiased Bresenham | 0.0000% |
| 1-6. | Bresenham | 0.0000% |
| 1-6. | Gupta-Sproull | 0.0000% |
| 1-6. | Hengeres Gupta-Sproull | 0.0000% |
| 1-6. | Ray marching | 0.0000% |
| 1-6. | Siddon | 0.0000% |
| 7. | Szűrt ray marching | 0.6305% |

**Összefoglalás**

Az utolsó mérés eredményének (és általában véve a szűrt ray marching vártnál rosszabb teljesítésének) magyarázata, hogy a szűrt ray marching a voxeltömb határain kilép a tömbből, és a közvetlenül a határvonal melletti, külső voxelek aktivitását is mintavételezi rendre $\frac{1-centerWeight}{6}$ súllyal. Amennyiben a voxeltömb erre nincs felkészítve (lásd 5.3.2, a szűrt ray marching matematikai hátterét leíró fejezet), mint ahogy ez az említett mérésnél is igaz volt, a határok mentén a szűrés hibás eredményhez vezet. Ez esetünkben rendkívüli módon kihangsúlyozódott, amikor is egyetlen voxel sem volt aktív, viszont a voxeltömb külsejét reprezentáló $outside$ paraméter értéke 1 000, ami jelentős aktivitást képvisel. A szűrés a határok mentén hozzáadott $1\,000 \cdot \frac{1-centerWeight}{6}$ aktivitást a valójában zérus aktivitású voxelekhez. Védekezni ez ellen az 5.3.2 fejezetben is olvasható gondolatmenet alapján úgy lehet, hogy a voxeltömbből kilépve nem egy konstans értéket adunk vissza, hanem a határvonalra tükrözzük a voxeltömb belsejét.

A hagyományos ray marching, a Bresenham és a Siddon, révén hogy nem hajtanak végre antialiasingot, érezhetően rosszabbul teljesítettek az antialiased vonalhúzó algoritmusokhoz képest. Felmerülhet a kérdés, hogy ha a referenciaérték ray marchinggal lett kiszámolva, később miért nem tudta reprodukálni annak eredményét? Ennek oka a mintapontok számában rejlik: míg a referenciaérték számításához $10^7$ $(\vec{z}_1, \vec{z}_2)$ pontpár került kijelölésre a detektorfelületeken, a mérés során mindössze 128. Mivel a valós idejű rekonstrukciós folyamat közben nincs idő magas mintaszámmal dolgozni (a gyakorlatban egy LOR-ról csupán 1-16 mintát szoktak venni), valóban látszik az antialiased eljárások jelentősége az eredmény pontosságának megőrzése szempontjából. A Gupta-Sproull algoritmus két változata elég szép eredményeket ér el. A kúp-, illetve hengerszűrést alkalmazó Gupta-Sproull változatok közül átlagosan a hengeres teljesít jobban. Ám ahogy az a hibagörbéken is megfigyelhető, a vizsgált algoritmusok közül a legjobbnak az Antialiased Bresenham bizonyul: ez az eljárás majdnem minden mérésben messze felülmúlta a többit pontosság tekintetében.

Az alábbi táblázat tartalmazza az összesített eredményt, amely a 128 minta után kapott relatív hiba átlaga alapján lett kiértékelve, az utolsó mérést nem beleszámítva.

| **Rangsor** | **Algoritmus** | **Átlagos hiba 128 minta után** |
|---|---|---|
| 1. | Antialiased Bresenham | 0.4695% |
| 2. | Hengeres Gupta-Sproull | 0.5867% |
| 3. | Gupta-Sproull | 0.5982% |
| 4. | Siddon | 0.6760% |
| 5. | Ray marching | 0.6765% |
| 6. | Bresenham | 0.6784% |
| 7. | Szűrt ray marching | 0.6845% |

Az utolsó mérés eredményét is beszámítva következőképpen alakul a rangsor:

| Rangsor | Algoritmus | Átlagos hiba 128 minta után |
|---|---|---|
| 1. | Antialiased Bresenham | 0.3912% |
| 2. | Hengeres Gupta-Sproull | 0.4889% |
| 3. | Gupta-Sproull | 0.4985% |
| 4. | Siddon | 0.5634% |
| 5. | Ray marching | 0.5638% |
| 6. | Bresenham | 0.5653% |
| 7. | Szűrt ray marching | 0.6755% |

Levonható a következtetés, hogy az Antialiased Bresenham algoritmus adja a legpontosabb eredményt, utána pedig a hengeres Gupta-Sproull nyújt ígéretes alternatívát. A szűrt ray marching alkalmazása csak akkor javasolt, ha a voxeltömböt felkészítjük a határmenti szűrés okozta pontatlanság kiküszöbölésére.

### 5.4.2 Futásidő

Az algoritmusok futásidejének vizsgálatát minden voxeltömbméret mellett egyszer végeztem el, mivel a futásidőre nincs hatással a voxelek aktivitásértéke. Reprezentatívan az 1., 3. és 5. méréseket választottam ki.

A mérések célja egyrészt az egyes eljárások, másrészt a CPU és a GPU implementáció sebességének összevetése volt. GPU implementáción belül vizsgáltam a textúra memória beépített lineáris interpolációjára épülő megoldásom teljesítményét is.

A LOR találatszámot 128 db $(\vec{z}_1, \vec{z}_2)$ minta alapján határozom meg. Mindezt 20 000-szer hajtom végre, és a kapott eredményeket átlagolom. Ez a CUDA program esetén azt jelenti, hogy 20 000 szálat indítok, és minden szál külön-külön kiszámolja a LOR találatszámot a saját 128 db $(\vec{z}_1, \vec{z}_2)$ mintája alapján. A CPU implementációban ugyanez egy 20 000 iterációból álló for ciklussal valósul meg. A futásidőt mind a 20 000*128 db minta számolásának befejezéséig mértem, és milliszekundum pontosságra kerekítettem.

**1. mérés**

| Algoritmus | CPU futásidő | GPU futásidő interpoláció nélkül | GPU futásidő interpolációval |
|---|---|---|---|
| Ray marching | 2 831 ms | 49 ms | 63 ms |
| Szűrt ray marching | 2 837 ms | 49 ms | 63 ms |
| Siddon | 2 241 ms | 113 ms | 116 ms |
| Bresenham | 1 890 ms | 35 ms | 29 ms |
| Antialiased Bresenham | 3 694 ms | 314 ms | 30 ms |
| Gupta-Sproull | 10 699 ms | 791 ms | 500 ms |
| Hengeres Gupta-Sproull | 10 667 ms | 791 ms | 501 ms |

**3. mérés**

| Algoritmus | CPU futásidő | GPU futásidő interpoláció nélkül | GPU futásidő interpolációval |
|---|---|---|---|
| Ray marching | 5 278 ms | 98 ms | 125 ms |
| Szűrt ray marching | 5 280 ms | 98 ms | 125 ms |
| Siddon | 3 760 ms | 185 ms | 177 ms |
| Bresenham | 3 292 ms | 64 ms | 58 ms |
| Antialiased Bresenham | 6 982 ms | 602 ms | 59 ms |
| Gupta-Sproull | 20 911 ms | 1 628 ms | 998 ms |
| Hengeres Gupta-Sproull | 20 944 ms | 1 628 ms | 998 ms |

**5. mérés**

| Algoritmus | CPU futásidő | GPU futásidő interpoláció nélkül | GPU futásidő interpolációval |
|---|---|---|---|
| Ray marching | 10 214 ms | 177 ms | 228 ms |
| Szűrt ray marching | 10 222 ms | 177 ms | 228 ms |
| Siddon | 6 671 ms | 304 ms | 317 ms |
| Bresenham | 6 167 ms | 128 ms | 114 ms |
| Antialiased Bresenham | 13 344 ms | 1 204 ms | 115 ms |
| Gupta-Sproull | 41 184 ms | 3 161 ms | 1 993 ms |
| Hengeres Gupta-Sproull | 41 334 ms | 3 257 ms | 1 994 ms |

**Összefoglalás**

A kapott értékekből látható, hogy míg a CPU implementáció futásideje a voxeltömb méretével lineárisan növekedik, a GPU-ra írt program sokkal jobb skálázódást tanúsít.

Az algoritmusok közül a Bresenham a leggyorsabb. A GPU textúra memóriájának és beépített lineáris interpolációjának kihasználása által sikerült elérni, hogy az Antialiased Bresenham ettől csak egy milliszekundummal legyen lassabb. Ez az interpoláció nélküli megoldáshoz képest tízszeres sebességnövekedést jelent, és ennek következtében az Antialiased Bresenham vált a leggyorsabb antialiased algoritmussá.

A Bresenhamtól nem sokkal marad le a ray marching két változata. Itt jön elő a szűrt ray marching előnye, hogy a szűrést mindössze egyszer, az eljárás legelején kell végrehajtani, és a későbbiekben nem lassítja az algoritmust. A szűrést követően semmilyen különbség nincs a hagyományos ray marchinghoz képest, így a futásideje rendkívül kedvező.

Futásidőt tekintve a Siddon eltérő eredményt ér el a CPU és a GPU implementációban. A CPU-n a Bresenhammal vetekedően az egyik leggyorsabb algoritmusnak számít. GPU-n sajnos a voxelek gyakori váltása és az ezzel járó intenzív memóriahasználat jelentős lassulást okoz, így mintegy kétszer lassabbá válik a Bresenhamnál.

A rangsort végül Gupta-Sproull variációi zárják. A táblázatok alapján megállapítható, hogy ez a két eljárás a többinél egy nagyságrenddel nagyobb futásidővel rendelkezik. Ennek oka egyrészt a komplex műveletek jelenléte (osztások, normalizálás), illetve az, hogy ciklusiterációnként kilenc voxelt is vizsgál, ami a számítási többleten felül a GPU-n a memóriaelérési kérések magas számát is magával vonja.

### 5.4.3 Hatékonyság

A pontosság és a futásidő alapján meghatároztam az algoritmusok hatékonyságát. Ehhez bevezettem egy hatékonysági tényezőt: $\text{hatékonyság} = \frac{1}{\text{hiba}^2 \cdot \text{futásidő}}$.

Az egyes algoritmusok hatékonysága az első öt mérés során (egészre kerekítve):

| Algoritmus | 1. mérés | 2. mérés | 3. mérés | 4. mérés | 5. mérés |
|---|---|---|---|---|---|
| Ray marching | 145 | 437 | 60 | 1 191 | 417 |
| Szűrt ray marching | 152 | 411 | 64 | 1 197 | 266 |
| Siddon | 79 | 238 | 43 | 847 | 299 |
| Bresenham | 308 | 931 | 131 | 2 576 | 828 |
| Antialiased Bresenham | 632 | 1 879 | 265 | 5 472 | 1 599 |
| Gupta-Sproull | 24 | 66 | 10 | 200 | 60 |
| Hengeres Gupta-Sproull | 25 | 67 | 10 | 211 | 63 |

A kapott eredményekből arra a következtetésre jutottam, hogy az Antialiased Bresenham bizonyul a leghatékonyabb algoritmusnak.

# 6 Antialiasing alkalmazása mozgáskompenzációban

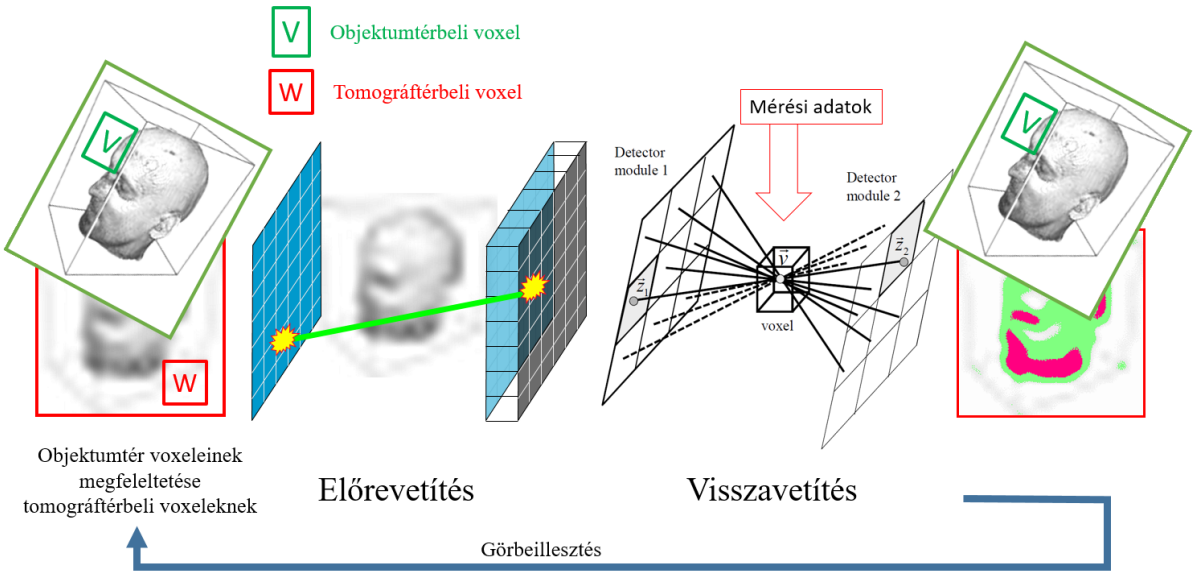
Neurológiai rendellenességek, például epilepszia tanulmányozásakor a páciens mozgása a PET vizsgálat alatt sokszor nem kerülhető el. A pontos rekonstrukció érdekében a mozgást mérni kell, és a kapott információt be kell építeni az alkalmazott fizikai modellbe (pozitron vándorlás, szóródás, elnyelődés, stb.) [56].

## 6.1 Mozgáskompenzáció

Az előre- és visszavetítést minden időkeretben és minden iterációban el kell végezni. Ezeket függetleníteni lehet a páciens mozgásától, ha feltételezzük, hogy egy időkereten belül a mozgás elhanyagolható, és a páciens pozíciója az adott időkeretben egyetlen geometriai transzformációval leírható. Sajnos azonban ez általában nem tehető fel a neurológiai rendellenességek vizsgálata során, mivel a mozgás sokkal gyorsabb dinamikájú a radioaktív nyomjelző terjedésénél. A gyors mozgás leírásához nagyon sok időkeretre lenne szükségünk, ami a rekonstrukció idejét elfogadhatatlanul hosszúra nyújtaná. Egy megoldás erre az úgynevezett kapuzott megközelítés [57], mely azonban a mért adatok jelentős részét figyelmen kívül hagyja, ezáltal kevésbé megbízható.

Az általam alkalmazott megoldás geometriai transzformációk helyett elmosást használ (*motion blur*). Az objektumtérbeli voxelek pályáját töröttvonalallal írja le, melyen antialiased vonalhúzó algoritmus számítja ki az elmosás eredményét. A folytonos motion blurt az előrevetítés előtt, illetve a visszavetítés után kell elvégezni. Ezt leszámítva minden más lépésben megegyezik a mozgáskompenzáció nélküli dinamikus PET rekonstrukcióval.

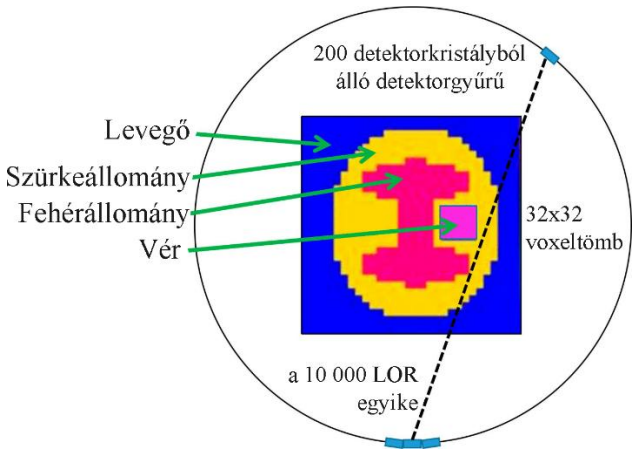
Ha a páciens mozog, akkor a pozitronok keletkezésének és detektálásának helyei közti kapcsolatot felállító $\boldsymbol{A}$ rendszermátrix időfüggő lesz, mivel a mérést objektum pontjai más-más időpillanatban más-más helyen lesznek. Emiatt két voxeltömbön dolgozok: az *objektumtérbeli* tömb követi a páciens mozgását, míg a *tomográftérbeli* tömb a tomográfhoz rögzített. (Az egyértelmű megkülönböztetés érdekében az objektumtérbeli voxeleket V-vel, míg a tomográftérbeli voxeleket W-vel jelölöm.) A rendszermátrix a tomográftérbeli voxelek kapcsolatát írja le a LOR-okkal, ezáltal nem időfüggő, viszont a tomográftérbeli voxelek aktivitása dinamikus egyrészt a radioaktív nyomjelző terjedése, másrészt az objektumtér és a tomográftér közötti transzformáció időfüggése miatt.




*29. ábra: Rekonstrukció mozgáskompenzációval.*

## 6.2 Modell

A mozgáskompenzáció gyakorlati vizsgálata során a 30. ábrán látható kétdimenziós agymodellt rekonstruáltam. Az egyszerűsített modell négy régióból áll: a koponyán belül szürkeállomány, fehérállomány és vér található, az egészet pedig kívülről levegő veszi körül. A mérési terület $32 \times 32$ voxelre lett felosztva. A detektorgyűrű 200 detektorkristályból áll, ami azt jelenti, hogy összesen $\frac{200 \cdot 100}{2} = 10\,000$ LOR van (minden kristály a szemközti 100 kristállyal alkothat LOR-t).




*30. ábra: Egyszerűsített kétdimenziós agymodell.*

## 6.3 Eredmények

A rekonstrukció 50 iterációból állt. Összesen 13 millió detektortalálat lett regisztrálva, melyek 10 időkeretbe lettek osztva. Kinetikus modellnek kétszövetes kompartment modellt használtam, regularizáció nélkül. Az alkalmazott antialiased vonalhúzó algoritmus az 5.4.3 fejezetben levont konklúzió alapján az Antialiased Bresenham eljárás lett.

A mérések során az alábbi paramétereket vizsgáltam:

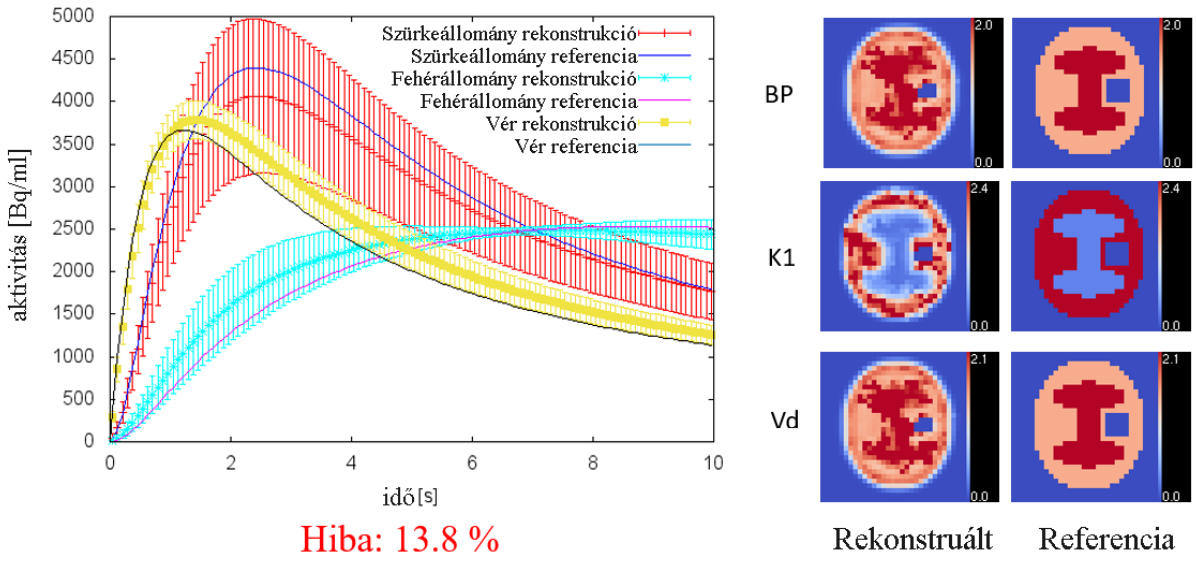
- a szürkeállomány, a fehérállomány és a vér rekonstruált **aktivitásértéke** az idő függvényében ([Bq/ml] mértékegységben)
- *kötési potenciál (Bindig Potential, **BP**):* az anyagcsere sebességének mutatója, vagyis hogy az adott szövetben a radioaktív nyomjelzővel ellátott anyag mekkora mértékben tud felhalmozódni
- ***K1***: a kétszövetes kompartment modellben a nyomjelző anyagnak a vérből az első szövet kompartmentbe való felszívódásának sebessége (ez egy konstans érték, mértékegysége [ml/cm$^2$/min])
- *eloszlási térfogat (Volume of dimension, **Vd**):* az egyensúlyi helyzet elérése után a vérben és az első szövet kompartmentben felhalmozódott nyomjelző anyag koncentrációjának aránya (például Vd=2 esetén egyensúlyi állapotban a nyomjelző koncentrációja kétszer akkora a szövetben, mint a vérben)
- **hiba:** a rekonstruált aktivitás L1 hibája (voxelenként és időlépésenként a számított és a referencia aktivitás különbségének abszolútértékének összege osztva a referencia voxelenként és időlépésenként vett összegével)

### 6.3.1 Álló objektum

Az első méréscsoportban azt mutatom be, hogy álló objektum esetén hogyan alakulnak a rekonstrukció paraméterei. Ez összehasonlítási alapot ad a mozgás során bekövetkező hiba vizsgálatához.

#### Mozgáskompenzáció nélkül

Ha a mért objektum nem mozog, akkor a mozgáskompenzáció nélküli dinamikus PET rekonstrukció (bármiféle regularizáció nélkül, kétszövetes kompartment modellt feltételezve) 50 iteráció után a következő eredményt adja:




*31. ábra: Rekonstrukció eredménye álló objektum esetén.*

**Mozgáskompenzációval**

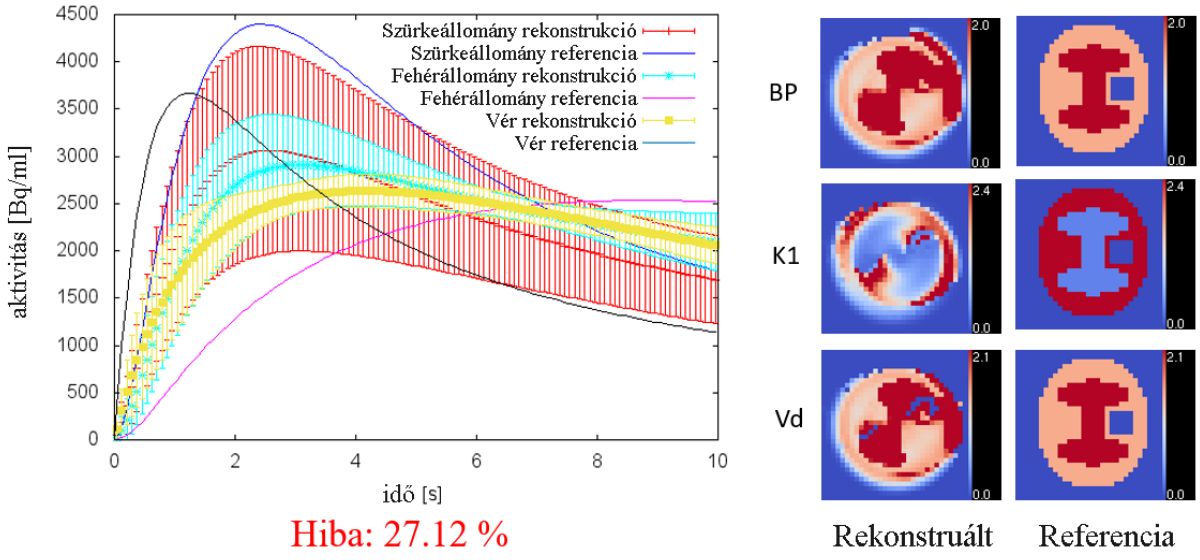
A mérést a 6.1 fejezetben ismertetett mozgáskompenzációs eljárással, az Antialiased Bresenham algoritmussal elvégezve pontosan ugyanazt kaptam, mint mozgáskompenzáció nélkül. (Ez nem meglepő, miután álló objektum esetén az objektumtérbeli voxelek egy az egyben a tomográftérbeli voxeleknek felelnek meg.)

### 6.3.2 Mozgó objektum

A második méréscsoportban a mért objektum forgó mozgást végzett, a 10 időkeret alatt egy teljes 360˚-os fordulatot tett meg a saját középpontja körül.

**Mozgáskompenzáció nélkül**

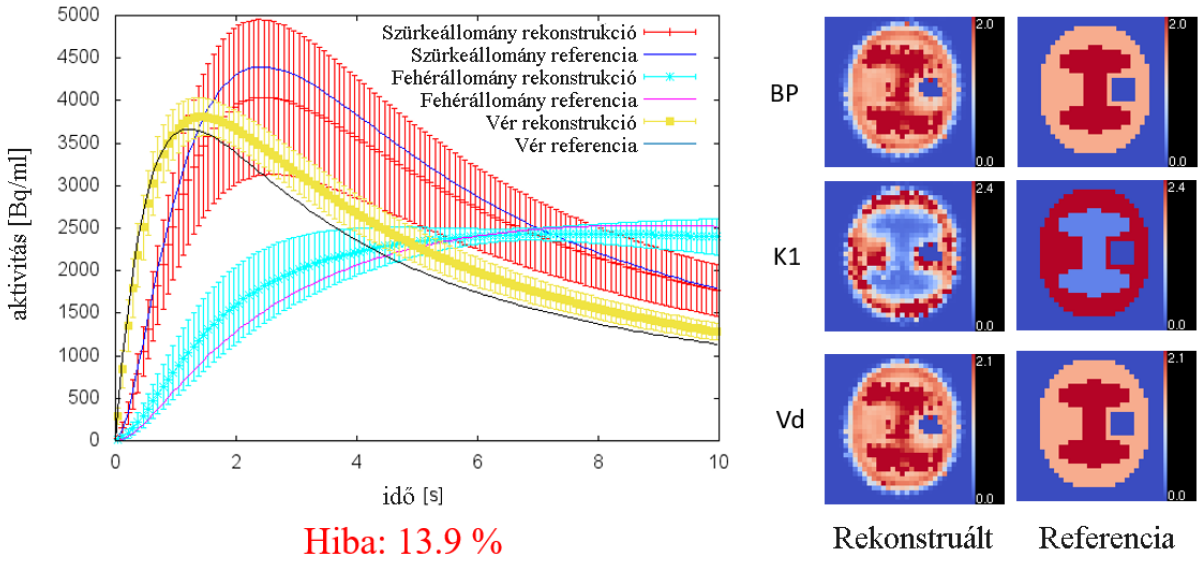
A mérés során azt vizsgáltam, hogy milyen eredménnyel jár, ha az objektum mozgását nem vesszük figyelembe. A 32. ábrán látható a rekonstrukció eredménye. Legjobban a K1 paraméteren figyelhető meg, hogy a referenciaképhez képest a rekonstruált kép a forgás következtében „elkenődött”.




*32. ábra: Rekonstrukció eredménye mozgó objektum esetén, mozgáskompenzáció nélkül.*

**Mozgáskompenzációval**

Mozgáskompenzációt alkalmazva a következő eredmény állt elő:




Hiba: 13.9 %

*33. ábra: Rekonstrukció eredménye mozgó objektum esetén, mozgáskompenzációval.*

Látható, hogy a rekonstruált képek elkenődése megszűnt. A rekonstrukció majdnem tökéletesen azt adta vissza, mint amit álló objektum esetén kaptunk, a hiba mindössze 0.1%-ot nőtt. Kijelenthető tehát, hogy a mozgáskompenzáció az Antialiased Bresenham algoritmus segítségével hatékonyan megvalósítható.

# Összefoglalás

Dolgozatomban az antialiased vonalhúzó algoritmusok alkalmazását vizsgáltam a PET rekonstrukció folyamán. E célból hét különböző algoritmust implementáltam és mértem.

Az előrevetítést két tesztprogrammal szimuláltam; az egyik a CPU-n futott, a másik a CUDA platformon keresztül a GPU-n. A tesztek során az egyes vonalhúzó algoritmusok pontosságát és futásidejét vizsgáltam. A kapott eredményekből arra a következtetésre jutottam, hogy mind a pontosságot, mind a futásidőt tekintve az Antialiased Bresenham bizonyul a leghatékonyabb algoritmusnak.

A mozgáskompenzációt motion blurrel valósítottam meg. A voxelek pályáját töröttvonalakkal közelítettem, melyek mentén az elmosást az Antialiased Bresenham algoritmus segítségével hajtottam végre. A módszer hatékonynak bizonyult, mozgó objektum esetén is majdnem tökéletesen (0.1% hibával) sikerült reprodukálni azt az aktivitás eloszlást, amit akkor kaptunk volna, ha az objektum végig állt volna.

# Irodalomjegyzék


[1] M. Authors, C. Cullinan Christopher Wyant Timothy Frattesi Advisor, and X. Huang, “Computing Performance Benchmarks among CPU, GPU, and FPGA.”

[2] W. Thomas and R. D. Daruwala, “Performance comparison of CPU and GPU on a discrete heterogeneous architecture,” *2014 Int. Conf. Circuits, Syst. Commun. Inf. Technol. Appl. CSCITA 2014*, pp. 271–276, 2014.

[3] D. J. Schlyer, “PET tracers and radiochemistry,” *Ann. Acad. Med. Singapore*, vol. 33, no. 2, pp. 146–154, 2004.

[4] “Fludeoxyglucose F 18 | C6H11FO5 - PubChem.” [Online]. Available: https://pubchem.ncbi.nlm.nih.gov/compound/450503#section=Top. [Accessed: 03-Sep-2018].

[5] H. Zaidi and M.-L. Montandon, “Scatter Compensation Techniques in PET,” *PET Clin.*, vol. 2, no. 2, pp. 219–234, Apr. 2007.

[6] “File:PET-image.jpg - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:PET-image.jpg. [Accessed: 01-Oct-2018].

[7] T. H. Schindler, H. R. Schelbert, A. Quercioli, and V. Dilsizian, “Cardiac PET imaging for the detection and monitoring of coronary artery disease and microvascular health,” *JACC Cardiovasc. Imaging*, vol. 3, no. 6, pp. 623–640, 2010.

[8] “RatCap - Synchropet.” [Online]. Available: https://synchropet.com/rat-cap/. [Accessed: 03-Sep-2018].

[9] M. Veronese, G. Rizzo, A. Bertoldo, and F. E. Turkheimer, “Spectral Analysis of Dynamic PET Studies: A Review of 20 Years of Method Developments and Applications,” *Comput. Math. Methods Med.*, vol. 2016, 2016.

[10] C. S. Patlak, R. G. Blasberg, and J. D. Fenstermacher, “Graphical Evaluation of Blood-to-Brain Transfer Constants from Multiple-Time Uptake Data,” *J. Cereb. Blood Flow Metab.*, vol. 3, no. 1, pp. 1–7, Mar. 1983.

[11] G. Wang and J. Qi, “Direct estimation of kinetic parametric images for dynamic PET,” *Theranostics*, vol. 3, no. 10, pp. 802–815, 2013.

[12] H. Watabe, Y. Ikoma, Y. Kimura, M. Naganawa, and M. Shidahara, “PET kinetic analysis--compartmental model.,” *Ann. Nucl. Med.*, vol. 20, no. 9, pp. 583–8, 2006.

[13] V. Y. Panin, F. Kehren, C. Michel, and M. Casey, “Fully 3-D PET reconstruction with system matrix derived from point source measurements,” *IEEE Trans. Med. Imaging*, vol. 25, no. 7, pp. 907–921, Jul. 2006.

[14] M. Rafecas *et al.*, “Use of a Monte Carlo-based probability matrix for 3-D iterative reconstruction of MADPET-II data,” *IEEE Trans. Nucl. Sci.*, vol. 51, no. 5, pp. 2597–2605, Oct. 2004.

[15] M. Mil, “GPU-BASED PARTICLE TRANSPORT FOR PET RECONSTRUCTION,” 2014.

[16] E. W. Weisstein, “Maximum Likelihood.”

[17] C.-T. Chen, C. E. Metz, and X. Hu, “Maximum Likelihood Reconstruction in PET and TOFPET,” in *Mathematics and Computer Science in Medical Imaging*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1988, pp. 319–329.

[18] M. Persson, D. Bone, and H. Elmqvist, “Total variation norm for three-dimensional iterative reconstruction in limited view angle tomography,” *Phys. Med. Biol.*, vol. 46, no. 3, pp. 853–866, Mar. 2001.

[19] D. L. Snyder and M. I. Miller, “The Use of Sieves to Stabilize Images Produced with the EM

Algorithm for Emission Tomography," *IEEE Trans. Nucl. Sci.*, vol. 32, no. 5, pp. 3864–3872, Oct. 1985.

[20] A. J. Reader and J. Verhaeghe, "4D image reconstruction for emission tomography," *Phys. Med. Biol.*, vol. 59, no. 22, pp. R371–R418, 2014.

[21] J. Matthews, D. Bailey, P. Price, and V. Cunningham, "The direct calculation of parametric images from dynamic PET data using maximum-likelihood iterative reconstruction.," *Phys. Med. Biol.*, vol. 42, no. 6, pp. 1155–73, Jun. 1997.

[22] Y. Rakvongthai, J. Ouyang, B. Guerin, Q. Li, N. M. Alpert, and G. El Fakhri, "Direct reconstruction of cardiac PET kinetic parametric images using a preconditioned conjugate gradient approach," 2013.

[23] J. Yan, B. Planeta-Wilson, and R. E. Carson, "Direct 4-D PET list mode parametric reconstruction with a novel em algorithm," *IEEE Trans. Med. Imaging*, vol. 31, no. 12, pp. 2213–2223, 2012.

[24] G. Wang and J. Qi, "An optimization transfer algorithm for nonlinear parametric image reconstruction from dynamic PET data," *Med. Imaging, IEEE Trans.*, vol. 31, no. 10, pp. 1977–1988, 2012.

[25] S. Coric, M. Leeser, E. Miller, and M. Trepanier, "Parallel-beam backprojection," in *Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays - FPGA '02*, 2002, p. 217.

[26] D. W. Shattuck, J. Rapela, E. Asma, A. Chatzioannou, J. Qi, and R. M. Leahy, "Internet2-based 3D PET image reconstruction using a PC cluster," *Phys. Med. Biol.*, vol. 47, no. 15, p. 317, Aug. 2002.

[27] M. Kachelrieß, M. Knaup, and O. Bockenbach, "Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware," *Med. Phys.*, vol. 34, no. 4, pp. 1474–1486, Mar. 2007.

[28] F. Xu and K. Mueller, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Phys. Med. Biol.*, vol. 52, no. 12, pp. 3405–3419, Jul. 2007.

[29] N. Gac, S. Mancini, M. Desvignes, and D. Houzet, "High speed 3D tomography on CPU, GPU, and FPGA," *Eurasip J. Embed. Syst.*, vol. 2008, no. 1, 2008.

[30] L. Szirmay-Kalos, L. Szécsi, and M. Sbert, "GPU-Based Techniques for Global Illumination Effects," *Synth. Lect. Comput. Graph. Animat.*, vol. 2, no. 1, pp. 1–275, Jan. 2008.

[31] "CUDA Zone | NVIDIA Developer." [Online]. Available: https://developer.nvidia.com/cuda-zone. [Accessed: 07-Sep-2018].

[32] "StreamSDK | Midnight Status." [Online]. Available: http://midnightstatus.com/?page_id=176. [Accessed: 01-Oct-2018].

[33] "OpenCL Overview - The Khronos Group Inc." [Online]. Available: https://www.khronos.org/opencl/. [Accessed: 29-Sep-2018].

[34] G. Pratx, G. Chinn, P. D. Olcott, and C. S. Levin, "Fast, Accurate and Shift-Varying Line Projections for Iterative Reconstruction Using the GPU," *IEEE Trans. Med. Imaging*, vol. 28, no. 3, pp. 435–445, Mar. 2009.

[35] F. Xu and K. Mueller, "GPU-Acceleration of Attenuation and Scattering Compensation in Emission Computed Tomography," *9th Fully Three-Dimensional Image Reconstr. Radiol. Nucl. Med. (High Perform. Image Reconstr. Work.*, pp. 5–8, 2007.

[36] Chung Chan, S. Meikle, R. Fulton, Guang-Jian Tian, Weidong Cai, and D. D. Feng, "A non-local post-filtering algorithm for PET incorporating anatomical knowledge," in *2009 IEEE Nuclear Science Symposium Conference Record (NSS/MIC)*, 2009, pp. 2728–2732.

[37] E. W. Weisstein, "Fast Fourier Transform."

[38] E. W. Weisstein, "Nyquist Frequency."

[39] E. W. Weisstein, “Shah Function.”

[40] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, “A performance study of general-purpose applications on graphics processors using CUDA,” *J. Parallel Distrib. Comput.*, vol. 68, no. 10, pp. 1370–1380, Oct. 2008.

[41] R. J. Barrientos, J. I. Gomez, C. Tenllado, and M. Prieto, “Heap Based k-Nearest Neighbor Search on GPUs,” *XXI Jornadas Paralelismo*, no. June 2014, pp. 559–566, 2010.

[42] “CUDA: synchronization of units — IT daily blog, news, magazine, technologies.” [Online]. Available: http://developers-club.com/posts/151897/. [Accessed: 29-Sep-2018].

[43] “CUDA Overview.” [Online]. Available: http://cuda.ce.rit.edu/cuda_overview/cuda_overview.htm. [Accessed: 29-Sep-2018].

[44] “CUDA C Programming Guide.” [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. [Accessed: 02-Sep-2018].

[45] G. Pratx and L. Xing, “GPU computing in medical physics: A review,” *Med. Phys.*, vol. 38, no. 5, pp. 2685–2697, May 2011.

[46] “CUDA LLVM Compiler | NVIDIA Developer.” [Online]. Available: https://developer.nvidia.com/cuda-llvm-compiler. [Accessed: 29-Sep-2018].

[47] “CUDA Programming: Texture Memory in CUDA | What is Texture Memory in CUDA programming.” [Online]. Available: http://cuda-programming.blogspot.com/2013/02/texture-memory-in-cuda-what-is-texture.html. [Accessed: 29-Sep-2018].

[48] J. E. Bresenham, “Algorithm for computer control of a digital plotter,” *IBM Syst. J.*, vol. 4, no. 1, pp. 25–30, 1965.

[49] R. L. Siddon, “Fast calculation of the exact radiological path for a three-dimensional CT array,” *Med. Phys.*, vol. 12, no. 2, pp. 252–255, Mar. 1985.

[50] S. Gupta, R. F. Sproull, S. Gupta, and R. F. Sproull, “Filtering edges for gray-scale displays,” in *Proceedings of the 8th annual conference on Computer graphics and interactive techniques - SIGGRAPH ’81*, 1981, vol. 15, no. 3, pp. 1–5.

[51] J. Křivánek and M. Colbert, “Real-time Shading with Filtered Importance Sampling,” *Comput. Graph. Forum*, vol. 27, no. 4, pp. 1147–1154, Jun. 2008.

[52] E. W. Weisstein, “Jensen’s Inequality.”

[53] H. Gao, “Fast parallel algorithms for the x-ray transform and its adjoint,” *Med. Phys.*, vol. 39, no. 11, pp. 7110–7120, Nov. 2012.

[54] T. Carter, “The Bresenham Line Algorithm,” 2014.

[55] “gnuplot homepage.” [Online]. Available: http://www.gnuplot.info/. [Accessed: 23-Sep-2018].

[56] J. Jiao *et al.*, “Direct Parametric Reconstruction with Joint Motion Estimation/Correction for Dynamic Brain PET Data,” *IEEE Trans. Med. Imaging*, vol. 36, no. 1, pp. 203–213, 2017.

[57] M. Toussaint, J.-P. Dussault, and R. Lecomte, “Revisiting motion compensation models in PET image reconstruction,” in *2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, 2016, pp. 90–94.

[58] “File:PET-schema.png - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:PET-schema.png. [Accessed: 01-Oct-2018].

[59] “File:16slicePETCT.jpg - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:16slicePETCT.jpg. [Accessed: 01-Oct-2018].