Direkt térfogat-vizualizáció

Csébfalvi Balázs
Irányítástechnika és Informatika Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

Vizualizációs algoritmusok csoportosítása

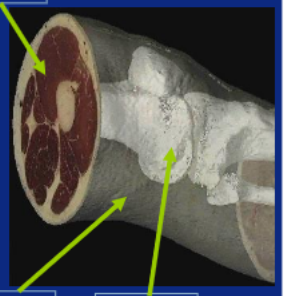
- Indirekt vizualizáció: Átmeneti reprezentációra van szükség
- Direkt vizualizáció: Közvetlenül a diszkrét adatot dolgozzuk fel
  - Képsorrendi (image order) megközelítés – a feldolgozást a pixelek sorrendjében végezzük
  - Objektumsorrendi (object order) megközelítés – a feldolgozást a voxelek sorrendjében végezzük
  - Hibrid megközelítés – ötvözi az objektumsorrendi és a képsorrendi módszerek előnyeit

Képsorrendi megközelítés - térfogati sugárkövetés

- Nincsenek analitikusan definiált objektumok
- A felületeket egy diszkrét implicit függvény reprezentálja
- Szükség van a folytonos implicit függvény rekonstrukciójára (approximáció/interpoláció)
- A pontos felületi normálisok a diszkretizálás során elvesznek
- Az árnyaláshoz becsülni kell a normálvektorokat

Belső struktúrák megjelenítése

- Szeletelés: a térfogat keresztmetszetét egy adott színkódolással megjelenítjük
- Szintfelület: nem átlátszó vagy áttetsző felületek on-line rekonstrukciója
- A különböző anyagok elnyelik a kibocsátott vagy visszavert fényt



Ray Tracing vs. Ray Casting

- A klasszikus sugárkövetés rekurzív (többszörös visszaverődések modellezése)
- Mért térfogati adatokban általában csak az elsődleges sugarakat (viewing ray – primary ray) követjük mivel a pontatlan normálvektorok akkumulált hibát eredményeznek
- Távolságtranszformációval konvertált geometriai modellek esetén a becsült gradiensek elég pontosak a rekurzív sugárkövetéshez

A sugárprofil definíciója

- Térfogati adat: skalármező a 3D térben

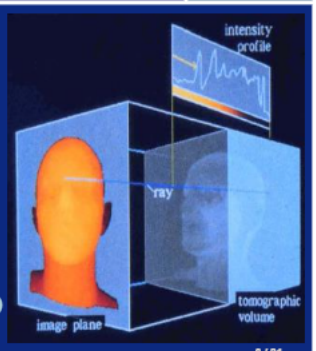
$$f(\mathbf{x}) \in \mathrm{R}^1, \mathbf{x} \in \mathrm{R}^3$$

- Sugár: félegyenes

$$r(t) \in \mathrm{R}^3, t \in \mathrm{R}^1 > 0$$

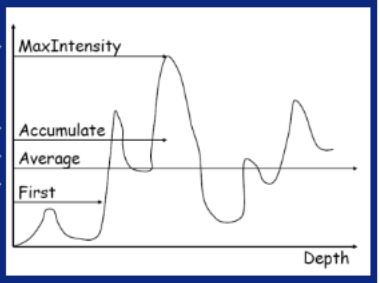
- Minták a sugár mentén: sugárprofil

$$f(r(t)) \in \mathrm{R}^1, t \in \mathrm{R}^1 > 0$$



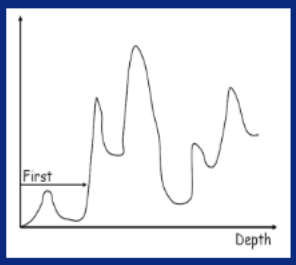
Sugárprofilok leképzései

- MIP ⇒
- Alpha-blending ⇒
- Röntgen ⇒
- Első ütközés ⇒



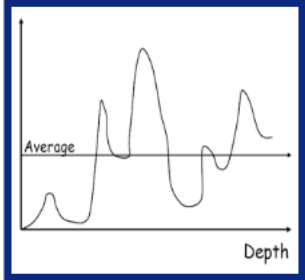
7 / 81

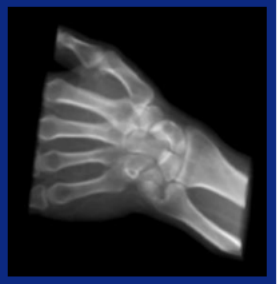
Első ütközés (first-hit ray casting)



8 / 81

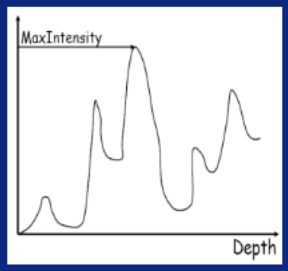
Röntgen-szimuláció

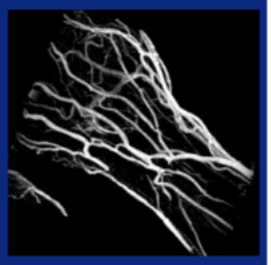




9 / 81

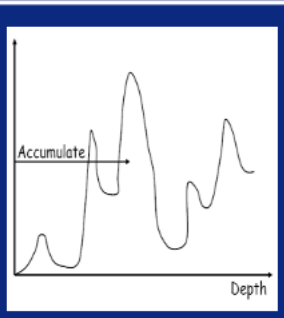
MIP - Maximum Intensity Projection





10 / 81

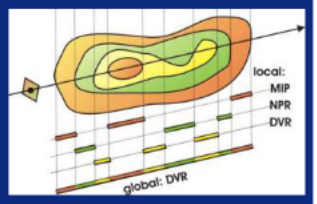
Akkumuláció - Alpha-blending



11 / 81

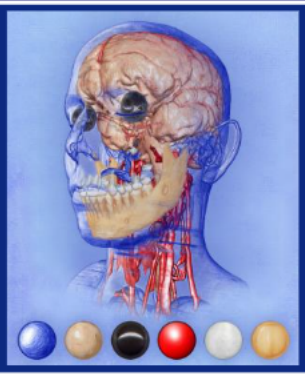
Többszintű térfogat-vizualizáció

- Szegmentálás
- Az egyes tartományokhoz különböző kompozitáló operátorokat használunk



12 / 81

Stílus átviteli függvények



- Illusztratív megközelítés
- Az egyes tartományokhoz megjelenítési stílust rendelünk

Térfogat-vizualizációs integrál

- A sugár parametrikus egyenlete:

$$\mathbf{r}(t) = \mathbf{o} + \mathbf{d} \cdot t$$

- Térfogat-vizualizációs integrál:

$$I = \int_0^{t_{max}} I(t) e^{-\int_0^t \mu(s)ds} dt$$

- Diszkrét közelítés:

$$I \approx \sum_{i=0}^{N} c(t_i) \cdot \alpha(t_i) \prod_{j=0}^{i-1} (1 - \alpha(t_j))$$

Rekurzív kiértékelés

- Back to front:

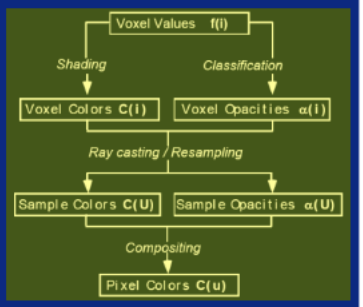
$$C_{out} = c(t_i) \cdot \alpha(t_i) + C_{in} \cdot (1 - \alpha(t_i))$$

- Front to back:
  - akkumulált opacitás pufferre van szükség
  - abbahagyjuk, ha az akkumulálódó opacitás már elért egy küszöbértéket (early ray termination)

Klasszikus Ray Casting

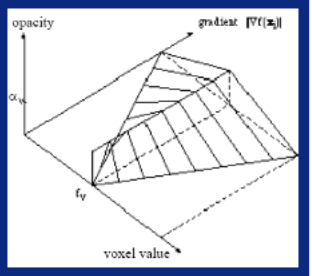
- Marc Levoy 1988

1. C(i) és α(i) értékeit az átviteli függvény (Transfer Function) határozza meg
2. Ray casting, interpoláció
3. Kompozitálás

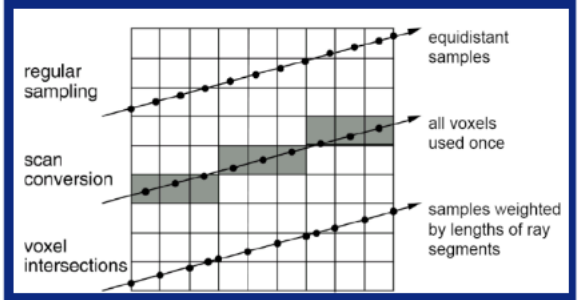


1. lépés: osztályozás, árnyalás

- Árnyalás: f(i) → C(i)
  - átviteli függvény
  - Phong modell
  - normálvektor: becsült gradiens
- Osztályozás: f(i) → α(i)
  - Levoy 88: moduláció a gradiens nagyságával
  - a jól definiált átmeneteket hangsúlyozza



2. lépés: Sugarak bejárása

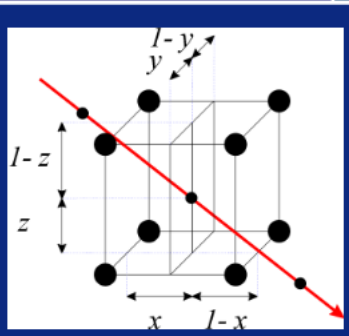


Sugarak bejárása, interpoláció

- Voxel-alapú vs. cella-alapú bejárás
- Trilineáris (interpoláció a cellán belül) vs. bilineáris (interpoláció a cellák lapjain)
- Trilineáris interpoláció:
  1. 4 új minta x irányban (interpolált négyzet)
  2. 2 új minta y irányban (interpolált vonal)
  3. 1 új minta z irányban (interpolált pont)
- Egységnyi vs. változó lépésköz: A kompozitálás különböző eredményt ad ⇒ opacitás-korrekció

19 / 81

Interpoláció - Újramintavételezés



20 / 81

3. lépés: kompozitálás

- Back-to-Front (B2F):
  - $Out_i = In_i (1 - \alpha_i) + C_i\alpha_i$, $In_{i+1} = Out_i$
  - példa:
    - Voxel i: C_i = piros, α_i =30%, In_i = fehér
    - Kompozitálás után: 70% fehér + 30% piros
- Front-to-Back (F2B):
  - Akkumulált szín: $C_{akk} = C_{akk} + (1 - \alpha_{akk}) C_i\alpha_i$
  - Akkumulált opacitás: $\alpha_{akk} = \alpha_{akk} + (1 - \alpha_{akk}) \alpha_i$

21 / 81

Összehasonlítás

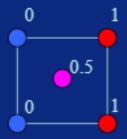
- Back-to-Front (B2F) két voxelre:
  - Háttér: In_0
  - 1. voxel: $Out_0 = In_0 (1 - \alpha_0) + C_0\alpha_0 = In_1$
  - 2. voxel: $Out_1 = In_1 (1 - \alpha_1) + C_1\alpha_1$, $In_{i+1} =$ $In_0 (1 - \alpha_0)(1 - \alpha_1) + C_0\alpha_0(1 - \alpha_1) + C_1\alpha_1$
- Front-to-Back (F2B) két voxelre:
  - Inicializálás: $C_{akk} = 0$, $\alpha_{akk} = 0$
  - 2. voxel: $C_{akk} = C_{akk} + (1 - \alpha_{akk}) C_1\alpha_1 = C_1\alpha_1$ $\alpha_{akk} = \alpha_{akk} + (1 - \alpha_{akk}) \alpha_1 = \alpha_1$
  - 1. voxel: $C_{akk} = C_1\alpha_1 + (1 - \alpha_1) C_0\alpha_0$ $\alpha_{akk} = \alpha_1 + (1 - \alpha_1) \alpha_0$
  - Háttér: $C_{akk} = C_1\alpha_1 + (1 - \alpha_1) C_0\alpha_0 + (1 - \alpha_1)(1 - \alpha_0) In_0$

$1-(\alpha_1+(1-\alpha_1)\cdot\alpha_0)$

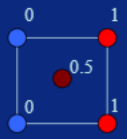
22 / 81

Opacitással modulált interpoláció

Interpoláció opacitás-moduláció nélkül



Interpoláció opacitás-modulációval



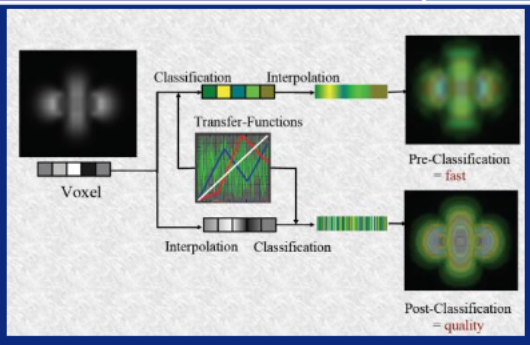
23 / 81

Preklasszifikáció vs. Posztklasszifikáció

- Preklasszifikáció
  - A szín és opacitás értékeket a voxelekhez rendeljük hozzá
  - A mintavételi pontokban ezeket az értékeket interpoláljuk (opacitás-moduláció)
- Posztklasszifikáció
  - a sűrűségértékeket (és a gradienseket) interpoláljuk
  - A mintavételi pontokhoz rendelünk színt az átviteli függvény szerint
  - A mintavételi pontokat árnyaljuk az interpolált gradiens alapján

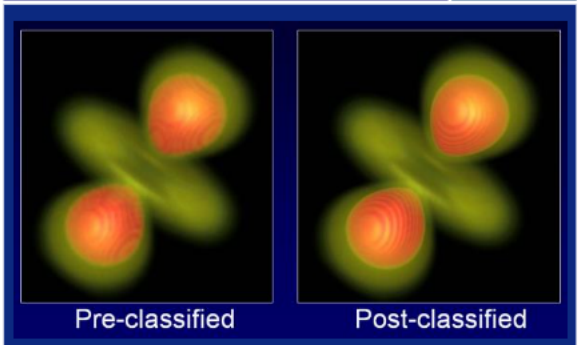
24 / 81

Preklasszifikáció vs. Posztklasszifikáció



25 / 81

Preklasszifikáció vs. Posztklasszifikáció



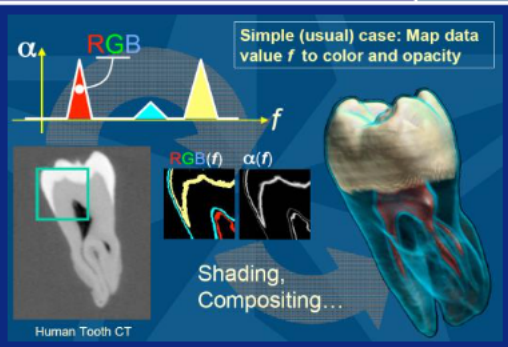
26 / 81

Átviteli függvények – Transfer functions

- Input:
  - sűrűség
  - gradiens (elsőrendű deriváltakból)
  - görbület (másodrendű deriváltakból)
  - pozíció (szegmentáló maszk alapján)
- Output:
  - szín (pl. RGB formátum)
  - opacitás

27 / 81

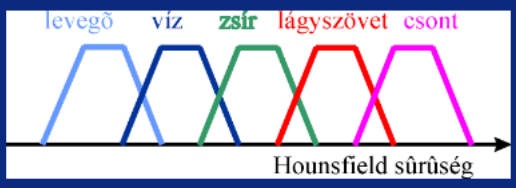
Átviteli függvények – Transfer functions



28 / 81

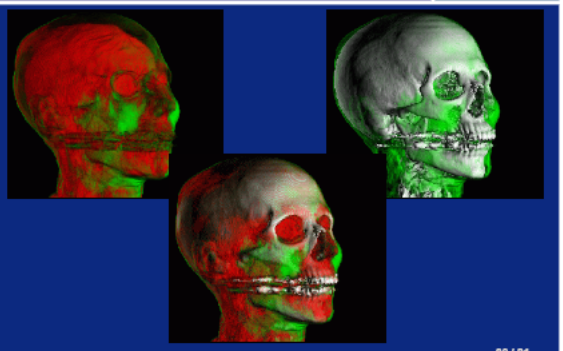
Fuzzy átviteli függvények

- Feltétel: Minden cellában legfeljebb két anyag keveréke lehet
- Bázisfüggvények lineáris kombinációja



29 / 81

Sugárkövetéssel generált képek

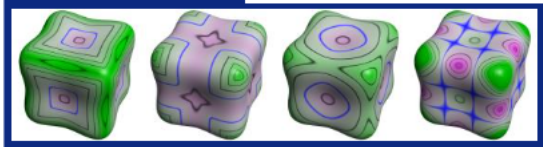


30 / 81

Görbület alapú átviteli függvények

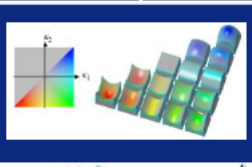
- Hesse mátrix – a másodrendű parciális deriváltakat tartalmazza
- A színeket a Hesse mátrix sajátértékei alapján rendeljük hozzá a voxelekhez

$$\mathbf{H} = \begin{bmatrix} \partial^2 f/\partial x^2 & \partial^2 f/\partial x \partial y & \partial^2 f/\partial x \partial z \\ \partial^2 f/\partial x \partial y & \partial^2 f/\partial y^2 & \partial^2 f/\partial y \partial z \\ \partial^2 f/\partial x \partial z & \partial^2 f/\partial y \partial z & \partial^2 f/\partial z^2 \end{bmatrix}$$



κ_1 κ_2 $(\kappa_1+\kappa_2)/2$ $\kappa_1\kappa_2$

Görbület alapú átviteli függvények



Brute-force Ray Casting (B2F)

```
RGB Volume::RayCasting(Vector pos, Vector dir)
{
    RGB color_acc;
    DBL tmin = this->EntryPoint(), t = this->ExitPoint();
    while(t > tmin)
    {
        Vector sample = pos + dir * t;
        DBL density = this->Resample(sample); // trilinear interpolation
        Vector gradient = this->ResampleGradient();
        opacity = OpacityFunction(density, gradient.Magnitude());
        color = ColorFunction(density) * Shading(gradient);
        color_acc = c_acc * (1 – opacity) + color * opacity; t–;
    }
    retrun color_acc;
}
```


Brute-force Ray Casting (F2B)

```
RGB Volume::RayCasting(Vector pos, Vector dir)
{
    RGB color_acc; DBL opacity_acc = 0;
    DBL t = this->EntryPoint(), tmax = this->ExitPoint();
    while(t < tmax)
    {
        Vector sample = pos + dir * t;
        DBL density = this->Resample(sample); // trilinear interpolation
        Vector gradient = this->ResampleGradient();
        opacity = OpacityFunction(density, gradient.Magnitude());
        color = ColorFunction(density) * Shading(gradient);
        color_acc += (1 – opacity_acc) * color * opacity;
        opacity_acc += (1 – opacity_acc) * opacity; t++;
    }
    retrun color_acc;
}
```


First-Hit Ray Casting

```
RGB Volume::RayCasting(Vector pos, Vector dir, DBL threshold)
{
    DBL t = this->EntryPoint(), tmax = this->ExitPoint();
    while(t < tmax)
    {
        Vector sample = pos + dir * t;
        DBL density = this->Resample(sample); // trilinear interpolation
        if(density > threshold)
        {
            Vector gradient = this->ResampleGradient();
            color = Shading(gradient);
            return color;
        }
        t++;
    }
    retrun BACKGROUND;
}
```


Gyorsító módszerek

- Primitív műveletek optimalizálása SIMD utasításokkal (SSE)
- Koherencia kihasználása (adat-koherencia, sugár-koherencia, frame-koherencia)
- Hierarchikus adatstruktúrák (piramis, octree)
- Üres régiók átugrása
- Potenciálmezők alkalmazása
- Hibrid módszerek (PARC – Polygon Assisted Ray Casting)
- GPU implementáció

Vektor osztály

```
struct Vector {
    float x, y, z, w;

    Vector(float x0, float y0, float z0, float w0 = 0) {  x = x0; y = y0; z = z0; w = w0; }

    Vector operator*(float a) {  return Vector(x * a, y * a, z * a, w * a); }

    Vector operator+(Vector& v) { return Vector(x + v.x, y + v.y, z + v.z, w + v.w); }

    Vector operator-(Vector& v) { return Vector(x - v.x, y - v.y, z - v.z, w - v.w); }

    float operator*(Vector& v) { return (x * v.x + y * v.y + z * v.z); }

    Vector operator%(Vector& v) {
        return Vector(y*v.z-z*v.y, z*v.x - x*v.z, x*v.y-y*v.x);
    }

    float Length() { return sqrt(x * x + y * y + z * z); }
};
```


Vektor osztály SSE utasításokkal

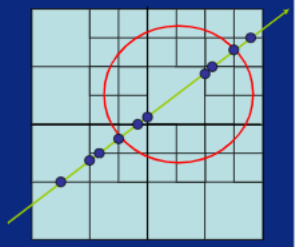
```
struct Vector {
    float x, y, z, w;
public:
    Vector operator+( Vector& v ) {
    __declspec(align(16)) Vector res;
    __asm {
      mov       esi, this
      mov       edi, v
      movups    xmm0, [esi]
      movups    xmm1, [edi]
      addps     xmm0, xmm1
      movaps    res, xmm0
    }
    return res;
  }
};
```


Early ray termination

```
RGB Volume::RayCasting(Vector pos, Vector dir)
{
    RGB color_acc; DBL opacity_acc = 0;
    DBL t = this->EntryPoint(), tmax = this->ExitPoint();
    while(t < tmax)
    {
        Vector sample = pos + dir * t;
        DBL density = this->Resample(sample); // trilinear interpolation
        Vector gradient = this->ResampleGradient();
        opacity = OpacityFunction(density, gradient.Magnitude());
        color = ColorFunction(density) * Shading(gradient);
        color_acc += (1 – opacity_acc) * color * opacity;
        opacity_acc += (1 – opacity_acc) * opacity; t++;
        if(opacity_acc > 0.999) break;
    }
    retrun color_acc;
}
```

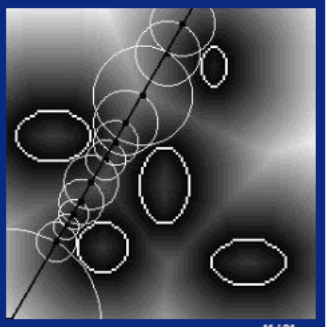

Min/Max Octree

- Minden szinten tároljuk a minimális d_{min} és a maximális d_{max} sűrűségeket
- A szintfelületet definiáló küszöbérték legyen t
- Megkeressük az adott mintavételi pontot tartalmazó legnagyobb blokkot, melyre $t < d_{min}$ vagy $t > d_{max}$.
- Ezt a blokkot a szintfelület biztosan nem metszi, így a sugarat léptetni tudjuk a kilépési pontra



Potenciálmező

- Minden cellában tároljuk a legközelebbi felületi pont távolságát
- Ezzel a távolsággal tudjuk léptetni a sugarat
- Nem hagyunk ki metszéspontot

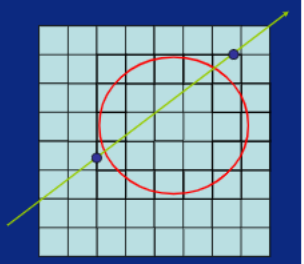


Hibrid gyorsítás

- Objektumsorrendi módszerek:
  - könnyű az üres régiókat kezelni
  - relatíve nagy overhead (vetítés, raszterizálás)
  - ritka térfogati adatokra hatékonyabb
- Képsorrendi módszerek:
  - nehéz az üres régiókat kezelni
  - a sebesség inkább a pixelek mint a voxelek számától függ
  - egyenletes kitöltöttség esetén hatékonyabb
- Hibrid módszerek:
  - rendszerint több lépés (objektum- és képsorrendi) kombinációja
  - a két megközelítés előnyeinek ötvözése

Polygon-Assisted Ray Casting

- Megkeressük azokat a cellákat, melyeket a szintfelület metsz
- A cellák lapjait a grafikus hardverrel két mélységi pufferbe (legközelebbi / legtávolabbi) rendereljük
- A mélységi puffer tartalma alapján meghatározható a sugarak kezdő és végpontja



43 / 81

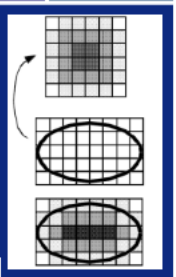
Pacázás - Splatting

- Objektumsorrendi megközelítés
- A voxeleket sorra egymásután vetítjük a képsíkra
- Az üres tartományok kezelésének nincs többletköltsége
- Cache koherencia kihasználása
- Hátrányok
  - A finom részletek elmosódnak
  - Szintfelület megjelenítésére csak korlátozottan alkalmas
  - A takart tartományokat is ki kell értékelni

44 / 81

Pacázás: alapötlet

- Az egyes voxelek nem csak egy pixelre vetülnek, hanem több pixelen hagyják a lenyomatukat (footprint kernel)
- Szétterítjük a voxel értékét a pixeleken
- A nagy pixel/voxel arányhoz nagyobb lenyomat kell de ez homályossá teszi a képet (blurring)



45 / 81

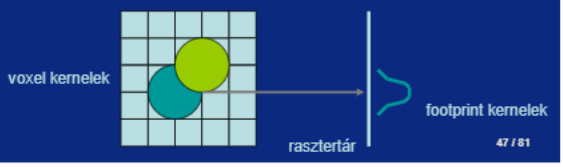
Pacázás - hatékonyság

- Lenyomat (footprint): kiintegrált rekonstrukciós kernel
- Ha a rekonstrukciós kernel gömbszimmetrikus, akkor a lenyomat kör alakú és független a párhuzamos vetítés irányától
- Perspektív vetítésnél a lenyomat ellipszis alakú, de szintén előre kiintegrálható
- Perspektív vetítésnél az ellipszisek irányát is számításba kell venni

46 / 81

Pacázás

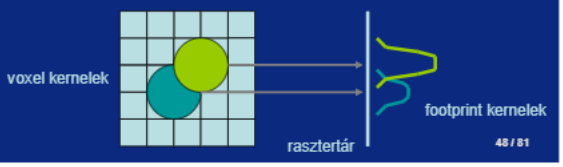
- Hatáskör (support): a 3D rekonstrukciós kernel által lefedett tartomány
- Minden voxelhez tarozik egy a voxel értékével súlyozott kernel
- Minden kernel egy 2D lenyomatot hagy a képen (szín, opacitás)
- A súlyozott lenyomatok akkumulálódnak a képen



47 / 81

Pacázás

- Hatáskör (support): a 3D rekonstrukciós kernel által lefedett tartomány
- Minden voxelhez tarozik egy a voxel értékével súlyozott kernel
- Minden kernel egy 2D lenyomatot hagy a képen (szín, opacitás)
- A súlyozott lenyomatok akkumulálódnak a képen



48 / 81