

Számítógépes Grafika

Hajder Levente
hajder@inf.elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2017/2018. II. félév

Tartalom

1

A fény és anyagok

- Anyagok
- Fényforrás modellek
- Fény-felület kölcsönhatás
- Fényvisszaverési modellek
- Buckatérkép
- Színmodellek

A fény és anyagok

- A fény elektromágneses hullám
- Az anyagokat olyan színűnek látjuk, amilyen színű fényt visszavernek
 - A visszaverés egyaránt függ az anyag és a megvilágítás "színétől"
- Különböző anyagok különböző módon viselkednek a fénnel szemben

Tartalom

1

A fény és anyagok

- Anyagok
 - Fényforrás modellek
 - Fény-felület kölcsönhatás
 - Fényvisszaverési modellek
 - Buckatérkép
 - Színmodellek

Felületek osztályozása

- Emittáló felületek

- Fénykibocsátó felületek emittáló anyagnak hívjuk
- Ezeket hívjuk fényforrásoknak, ilyen a Nap, a lámpa stb.

- Diffúz felületek

- A diffúz vagy matt felületeket minden irányból nézve ugyanolyan színűnek látjuk
- Ilyen például a frissen meszelt fal vagy a homok stb.
- A diffúz felület a beérkező fénysugár energiáját minden irányban azonos intenzitással veri vissza

Felületek osztályozása

- Spekuláris felületek
 - Tükörző felületek, az ideális fénytörés irányába verik vissza nagyrészt a beérkező fényt
- Átlátszó felületek
 - Ezekben a felületeken áthalad a fény, a beérkező fénysugár energiájának java részét áteresztik

Felületek osztályozása

- Áttetsző felületek

- Ezek a beérkező fény nagy részét magukba engedik, de csak kis része lép ki az anyagból
- Pl. tej, bőr

- Anizotróp felületek

- A felületet a tengelye körül forgatva, a beeső és visszaverődő szögeket tartva is változik a színe
- Mint például a CD

Tartalom

1

A fény és anyagok

- Anyagok
- Fényforrás modellek
- Fény-felület kölcsönhatás
- Fényvisszaverési modellek
- Buckatérkép
- Színmodellek

Fényforrás modellek

Fény

A fény elektromágneses hullám

Absztrakt fényforrások

- Ambiens fény
- Irány fényforrás
- Pont fényforrás
- Reflektorfény (spotlight)

Fényforrás modellek

Fény

A fény elektromágneses hullám

Absztrakt fényforrások

- Ambiens fény
- Irány fényforrás
- Pont fényforrás
- Reflektorfény (spotlight)

Fényforrás modellek

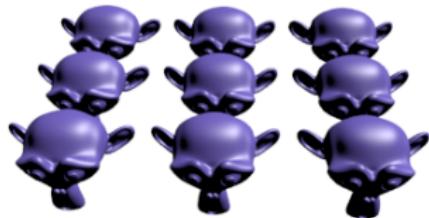
Ambiens fény

- Fénysugarak minden irányba egyenlő mértékben világítanak
- Távolság az intenzitást nem befolyásolja

Fényforrás modellek

Irány fényforrás

- Fénysugarak párhuzamosak
- Távolsággal a fény intenzitása nem csökken



Fényforrás modellek

Pont fényforrás

- Egy adott pontból indulnak ki a fénysugarak
- Fizika: a távolság négyzetével fordítottan arányos a fény intenzitás
- Szimulációkban a fakulást (falloff) meg lehet adni skaláris, lineáris, kvadratikus tagokkal



Fényforrás modellek

Reflektorfény (spotlight)

- Egy adott pontból indulnak ki a fénysugarak
- A fénnyalábot egy kör alapú végtelen gúla határozza meg
- A távolság négyzetével fordítottan arányos a fény intenzitás (valóságban)



Tartalom

1

A fény és anyagok

- Anyagok
- Fényforrás modellek
- Fény-felület kölcsönhatás**
- Fényvisszaverési modellek
- Buckatérkép
- Színmodellek

BRDF

- Legyen L^{in} egy adott irányból a felület egy pontjára beérkező, L pedig az onnan visszavert fény intenzitása
- Jelölje \mathbf{l} a fényforrás felé mutató egységvektort, \mathbf{v} a nézőpont felé mutató egységvektort, \mathbf{n} pedig a felületi normálist az adott pontban. A θ legyen az \mathbf{n} és \mathbf{l} által bezárt szög
- Ekkor a kétirányú visszaverődéses eloszlási függvény, BRDF (bi-directional reflection distribution function) a következő:

$$f_r(\mathbf{l}, \mathbf{v}) = \frac{L}{L^{in} \cos \theta}$$

Jelölések

- $\mathbf{v} := \omega$ a nézeti irány, azaz a szem/kamera fele mutató vektor
- $\mathbf{l} := -\omega'$ a megvilágító, a fényt "adó" pont fele mutató vektor, ekkor a beesési irány $-\mathbf{l}$ ($= \omega'$)
- \mathbf{n} a felületi normális
- $\mathbf{v}, \mathbf{l}, \mathbf{n}$ egységvektorok
- θ a \mathbf{l} és a \mathbf{n} által bezárt szög

Helmholtz-törvény

- Helmholtz-féle szimmetria: a fénysugár megfordítható
- Azaz: $f_r(\omega', \omega) = f_r(\omega, \omega')$
- Ez két dologért is jó:
 - Garantálja, hogy végső soron a radiancia csökken.
 - Nézhetjük "visszafelé" a sugarakat.

Ideális visszaverődés

Visszaverődési törvény

A beesési irány ($-I$), a felületi normális (n), és a kilépési irány (r) egy síkban van, valamint a beesési szög (θ) megegyezik a visszaverődési szöggel (θ').

Ideális visszaverődés

- Az ideális tükör csak az \mathbf{r} tükörirányba ver vissza.

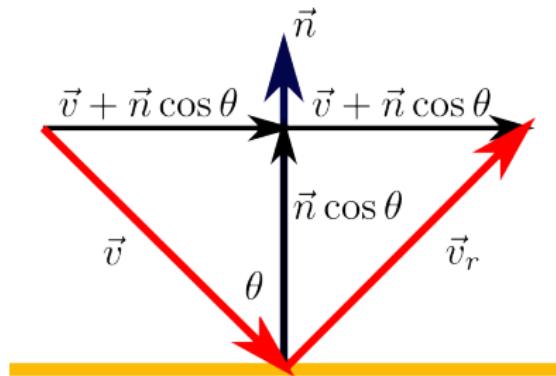
-

$$f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}) = k_r \frac{\delta(\mathbf{r} - \mathbf{v})}{\cos \theta}$$

- δ a *Dirac-delta* függvény, ami egy általánosított függvény, amely minden nem nulla paraméterre nullát ad, de a valós számok felett vett integrálja 1.
- A k_r visszaverődési együttható a *Fresnel*-együttható. Ez függ az anyag törésmutatójából, és az elektromos vezetési képességeből származik.
- A *Fresnel*-együttható a visszavert és beeső energia hányadát fejezi ki.

Visszaverődési irány

- Általános esetben, egy \mathbf{v} beeső vektorból a visszaverődési- vagy tükrirány:
- $\mathbf{v}_r = \mathbf{v} - 2\mathbf{n}(\mathbf{n} \cdot \mathbf{v})$
- Mivel $\cos \theta = -\mathbf{n} \cdot \mathbf{v}$
- Mindez csak akkor igaz, ha \mathbf{n} és \mathbf{v} vektorok egységnyi hosszúak!
- Általános esetben:
$$\mathbf{v}_r = \mathbf{v} + 2\mathbf{n} \cos \theta$$



Ideális törés

Snellius-Descartes törvény

A beesési irány ($-I$), a felületi normális (n), és a törési irány (t) egy síkban van, valamint $\eta = \frac{\sin \theta}{\sin \theta'}$, ahol η az anyagok relatív törésmutatója.

Néhány törésmutató

- Vákuum 1.0
- Levegő 1.0003
- Víz 1.3333
- Üveg 1.5
- Gyémánt 2.417



Ideális törés

Snellius-Descartes törvény

A beesési irány ($-I$), a felületi normális (n), és a törési irány (t) egy síkban van, valamint $\eta = \frac{\sin \theta}{\sin \theta'}$, ahol η az anyagok relatív törésmutatója.

Néhány törésmutató

- Vákuum 1.0
- Levegő 1.0003
- Víz 1.3333
- Üveg 1.5
- Gyémánt 2.417



Ideális törés

- Jelölje \mathbf{t} az ideális törési irányt.
- Az ideális tükörhöz hasonlóan kapjuk:

$$f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}) = k_t \frac{\delta(\mathbf{t} - \mathbf{v})}{\cos \theta}$$

Ideális törés

- Jelölje \mathbf{t} az ideális törési irányt.
- Az ideális tükörhöz hasonlóan kapjuk:

$$f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}) = k_t \frac{\delta(\mathbf{t} - \mathbf{v})}{\cos \theta}$$

Törési irány

- ### • Snellius-Descartes

törvény: $\eta = \frac{\sin \alpha}{\sin \beta}$

$$\bullet \mathbf{v}_t = \mathbf{n} \perp \sin \beta - \mathbf{n} \cos \beta$$

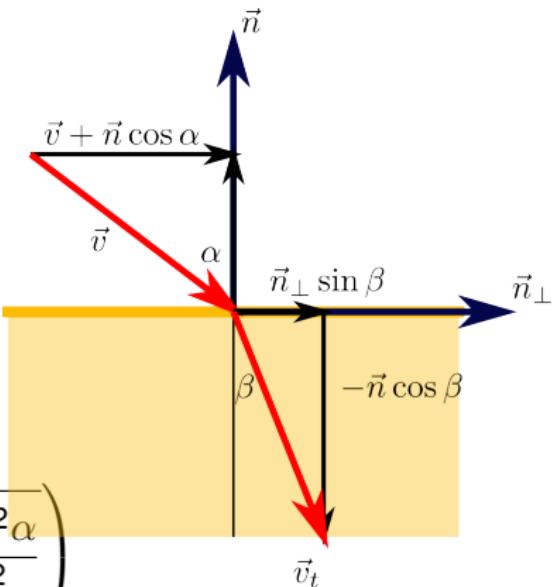
- $\mathbf{n}_{\perp} = \frac{\mathbf{v} + \mathbf{n} \cos \alpha}{\sin \alpha}$

$$\bullet \quad \mathbf{v}_t = \frac{\mathbf{v}}{\eta} + \mathbf{n} \left(\frac{\cos \alpha}{\eta} - \cos \beta \right)$$

$$\bullet \cos \beta = \sqrt{1 - \sin^2 \beta} =$$

$$\sqrt{1 - \frac{\sin^2 \alpha}{\eta^2}}$$

$$\mathbf{v}_t = \frac{\mathbf{v}}{\eta} + \mathbf{n} \left(\frac{\cos \alpha}{\eta} - \sqrt{1 - \frac{\sin^2 \alpha}{\eta^2}} \right)$$



Tartalom

1

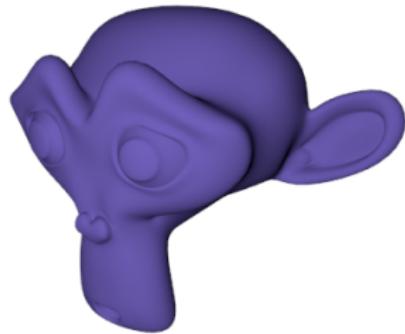
A fény és anyagok

- Anyagok
- Fényforrás modellek
- Fény-felület kölcsönhatás
- Fényvisszaverési modellek**
- Buckatérkép
- Színmodellek

Lambert-törvény

- Optikailag durva, *diffúz* felületek leírására jó.
- Feltételezés: a visszavert fénymennyiség nem függ a nézeti iránytól.
- Helmholtz-törvényt miatt akkor a bejövő iránytól sem függhet, azaz konstans:

$$f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}) = k_d$$



Spekuláris visszaverődés - Phong modell

- A tükörirányban intenzíven visszaverő, de attól távolodva gyorsan elhaló "csillanás" adható meg vele.
- Legyen ϕ az \mathbf{r} tükörirány és a \mathbf{v} nézeti irány által bezárt szög.
- Ekkor $\cos \phi = \mathbf{r} \cdot \mathbf{v}$
- Olyan függvényt keresünk, ami $\phi = 0$ -ra nagy, de gyorsan elhal.
-

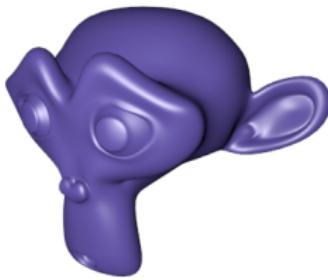
$$f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}) = k_s \frac{\cos^n \phi}{\cos \theta}$$

Nem szimmetrikus!

Spekuláris visszaverődés - Phong modell



$n = 5$



$n = 25$



$n = 50$

Spekuláris visszaverődés - Phong-Blinn modell

- Legyen \mathbf{h} a nézeti irány és a megvilágító pont fele mutató vektorok felezővektora.

-

$$\mathbf{h} = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

- Legyen δ a \mathbf{h} és az \mathbf{n} normálvektor által bezárt szög.
- Ekkor $\cos \delta = \mathbf{h} \cdot \mathbf{n}$

-

$$f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}) = k_s \frac{\cos^n \delta}{\cos \theta}$$

- Nagyon hasonló az egyszerű Phong modellhez, kicsit gyorsabban számítható.

Spektrális képszintézis

- Különböző hullámhosszú fény máshogyan viselkedik a felületeken.
- Színérzet a látható tartományban levő elektromágneses hullámok integrálja a három érzékelőnek megfelelően.
- Fényjelenségeket minden hullámhosszon külön kellene nézni.
 - Rendkívül számításigényes.
 - R,G,B komponensekkel jól közelíthetjük.

Tartalom

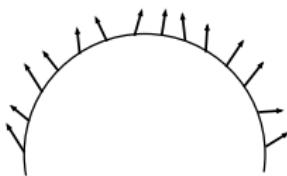
1

A fény és anyagok

- Anyagok
- Fényforrás modellek
- Fény-felület kölcsönhatás
- Fényvisszaverési modellek
- Buckatérkép**
- Színmodellek

Érdes felületek képzése

- Érdes felületek rengeteg poligonnal (háromszöghálóval) képezhetők.
 - Modell bonyolult, nehéz módosítani.
 - Renderelést lassítja.
- Trükk: egyszerű modellhez finom sűrű normálvektormezőt adunk meg

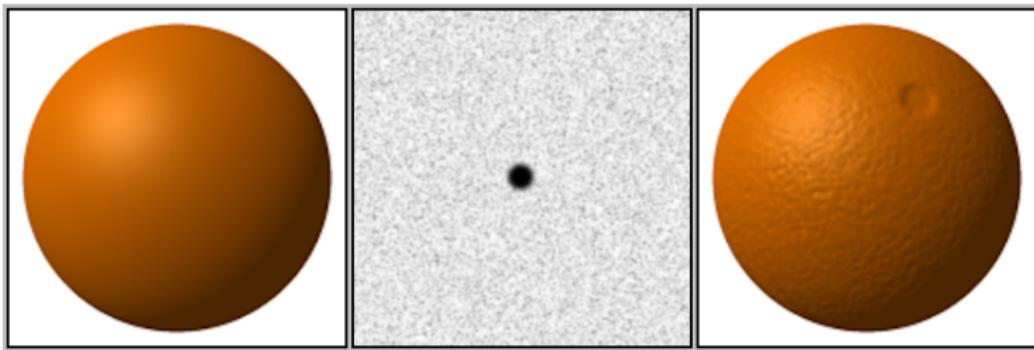


Érdes felületek képzése

- Durva pozíció + finom normálvektorok jó közelítés, ha
 - a felület nagyjából folytonos
 - mélységingadozás kicsi a felületen
- Mélység megadás képként: buckatérkép (bump map)
 - Textúraleképzéshez hasonlóan, képként szokás megadni
 - Buckatérkép leírhatja a mélységváltozást vagy a normálvektorokat (3D: 3 színkomponens).

Esettanulmány: narancs

- Példa (eredeti modell, buckatérkép, új modell):



Tartalom

1

A fény és anyagok

- Anyagok
- Fényforrás modellek
- Fény-felület kölcsönhatás
- Fényvisszaverési modellek
- Buckatérkép
- Színmodellek

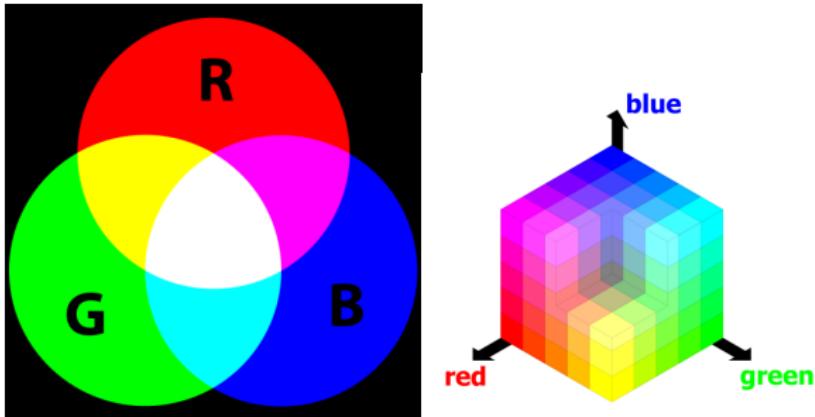
Színmodellek

- Fény: elektromágneses hullám
- Emberi szem által látható fény: alapszínek keveréséből
- Alapszínek: szivárvány színei



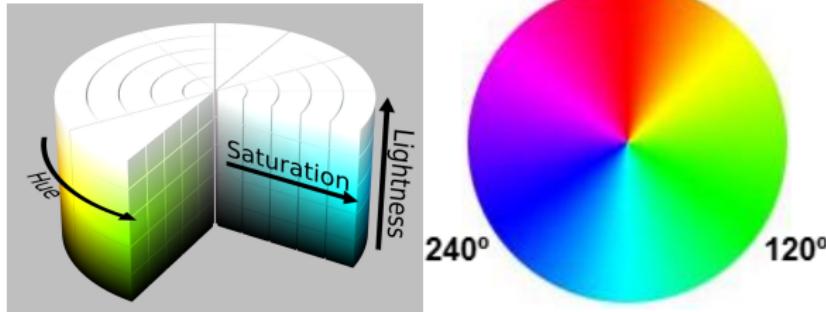
Színmodellek

- Monitor színmodellje: RGB
- Majdnem az összes látható szín kikeverhető



Színmodellek

- "Emberibb" színmodell: HSL (HSB,HSV)
- Három komponens: hue (színárnyalat), saturation (telítettség), lightness (fényesség)



Számítógépes Grafika

Hajder Levente
hajder@inf.elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2017/2018. II. félév

Tartalom

1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- Metszések
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

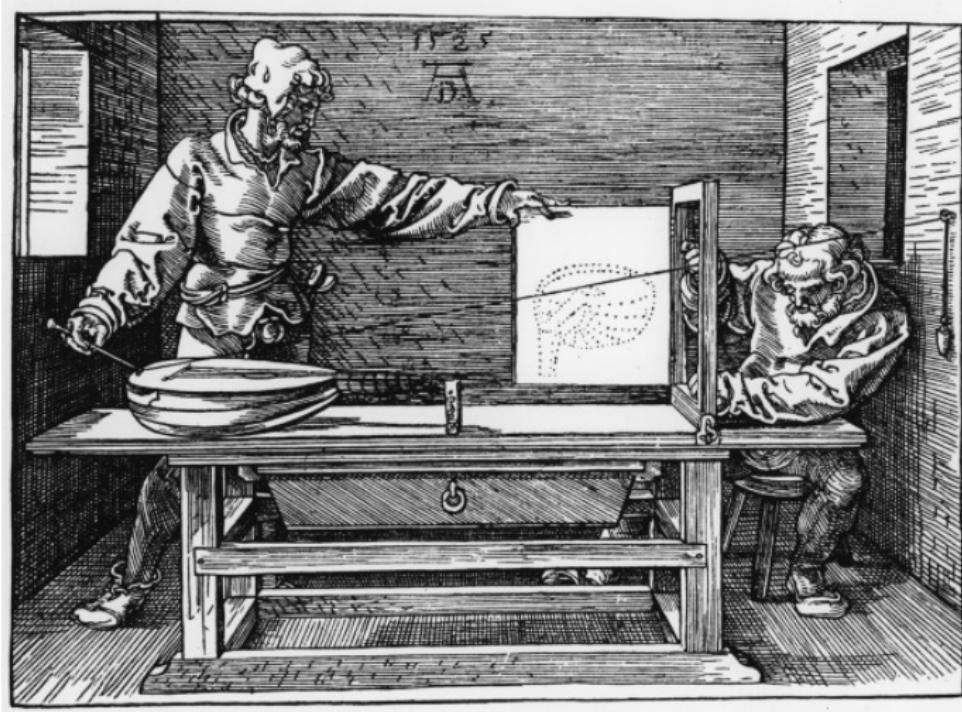
2 Rekurzív sugárkövetés

Tartalom

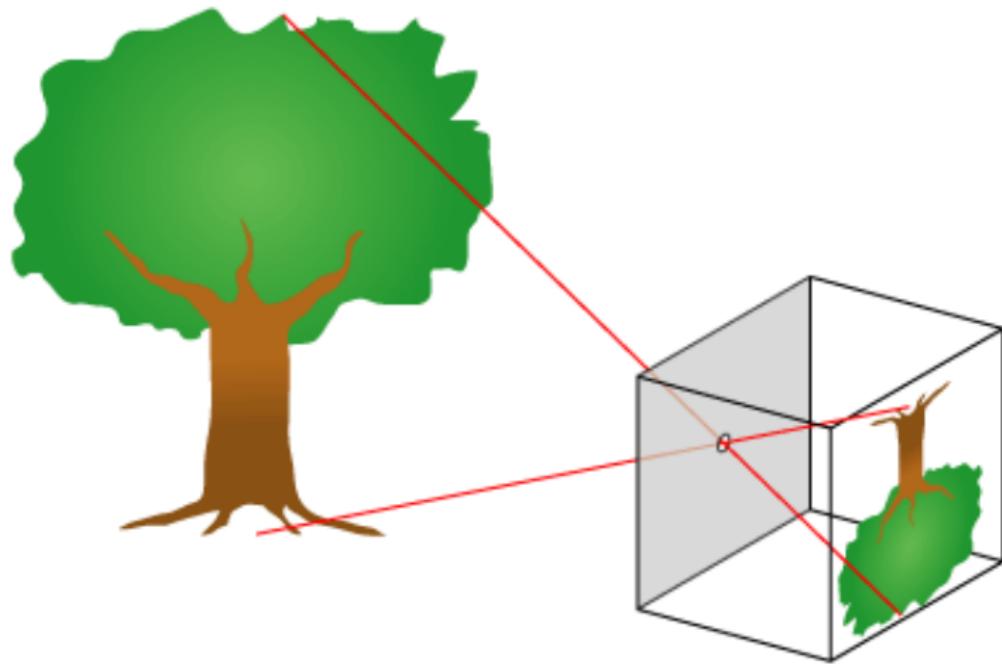
1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- Metszések
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

2 Rekurzív sugárkövetés

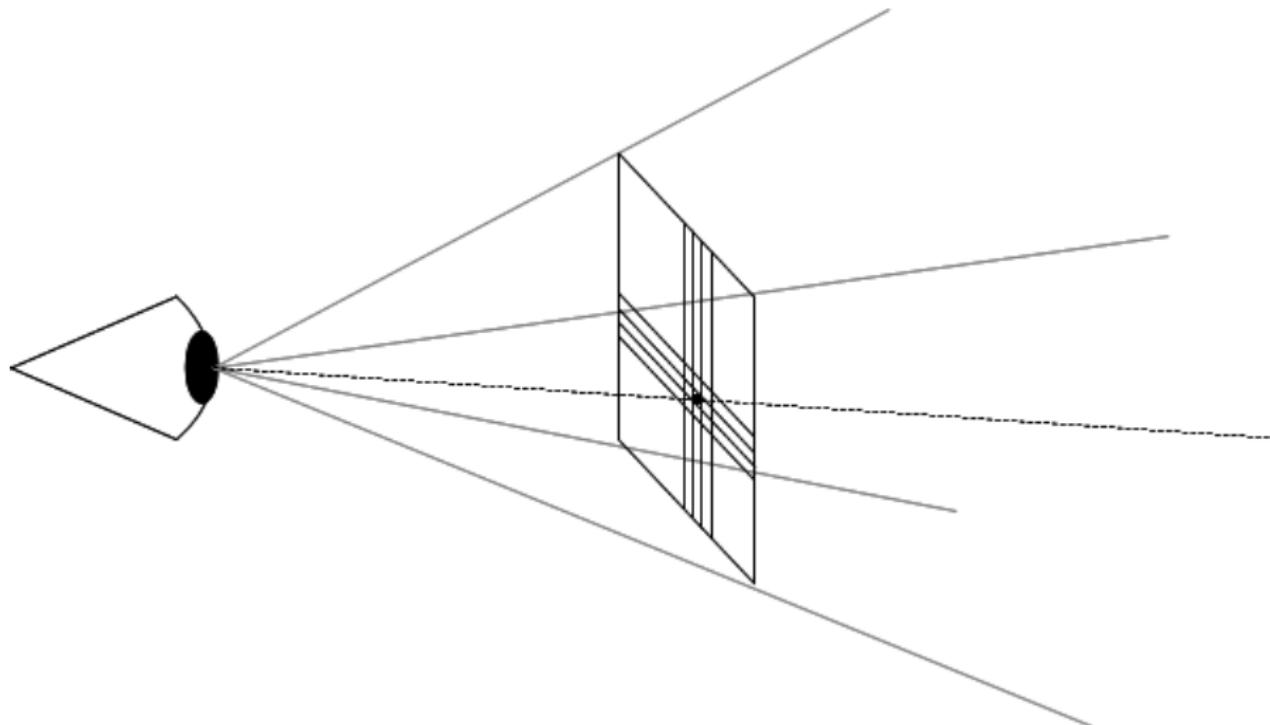


Albrecht Dürer, 1525



Raycasting
Rekurzív sugárkövetés

Motiváció
Raycasting
Sugarak indítása
Metszések



Motiváció

- Tekintsünk minden pixelre úgy, mint egy kis ablakra a világra
- Milyen színértéket vegyen fel ez a pixel? → Nézzük meg, mi látszik onnan a világból és az alapján rendeljünk hozzá a pixelhez egy színt!

Tartalom

1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- Metszések
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

2 Rekurzív sugárkövetés

Raycasting

Minden pixelre:

Indítsunk egy sugarat a színtérbe

Minden objektumra a színtérben:

Nézzük meg, hogy metszi-e a sugár az objektumot

A legközelebbi metszett objektum
színével színezzük ki a pixelt

Sugár

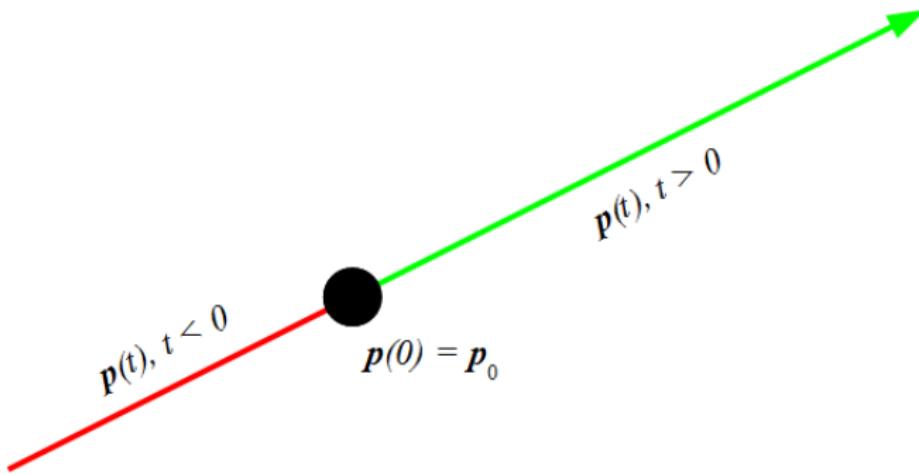
- A sugárnak van
 - egy \mathbf{p}_0 kiindulási pontja
 - és egy \mathbf{v} iránya
- A parametrikus sugár:

$$\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v},$$

ahol $t > 0$ (félegyenes!).

- $t = 0?$, $t < 0?$ sugár kezdőpontja, sugár mögötti részek

Sugár



Kérdés

- Honnan indítsuk a sugarat?
- Milyen irányba küldjük a sugarat?
- Hogyan metszük el a sugarat akármivel?

Tartalom

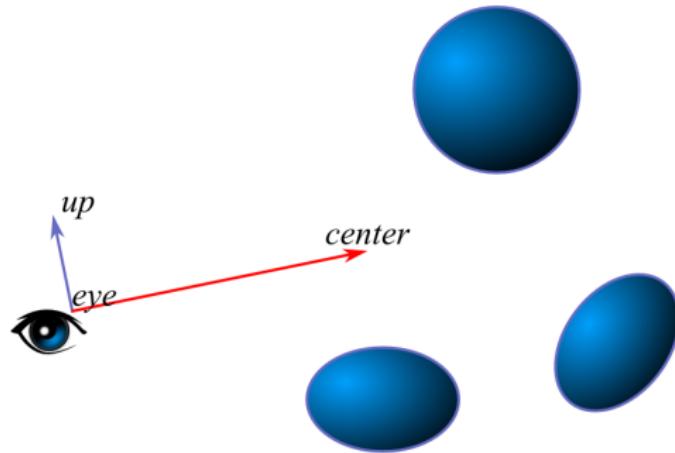
1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- Metszések
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

2 Rekurzív sugárkövetés

Sugarak indítása

- A szempozicióból indítunk sugarakat minden pixel középpontján keresztül
- Most: középpontosan szeretnénk vetíteni egy szembe, a vetítési sík egy négyszögletes részét megfeleltetve a képernyőnek
- Szem/kamera tulajdonságok:
 - szempozíció (**eye**),
 - egy pont amire néz (**center**),
 - felfele irányt megadó vektor a világban (**up**),
 - nyílásszög, amekkora szögtartományt lát (*fov_x*, *fov_y*).
 - (vetítővászon mérete. Most legyen adott:
$$2 \tan\left(\frac{\text{fov}_x}{2}\right) \times 2 \tan\left(\frac{\text{fov}_y}{2}\right)$$
nagyságú)
- Ezek segítségvel fogjuk megadni az (i, j) pixel világbeli koordinátáit



Sugarak indítása

Keressük a kamera saját $\mathbf{u}, \mathbf{v}, \mathbf{w}$ (jobbkezes!) koordinátarendszerét!

- Nézzen a kamera $-Z$ irányba!

$$\mathbf{w} = \frac{\mathbf{eye} - \mathbf{center}}{|\mathbf{eye} - \mathbf{center}|}$$

- Az X tengely legyen merőleges mind \mathbf{w} -re, mind az \mathbf{up} irányra!

$$\mathbf{u} = \frac{\mathbf{up} \times \mathbf{w}}{|\mathbf{up} \times \mathbf{w}|}$$

- Az Y tengely merőleges \mathbf{u} -ra és \mathbf{w} -re is:

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

(i, j) pixel koordinátái

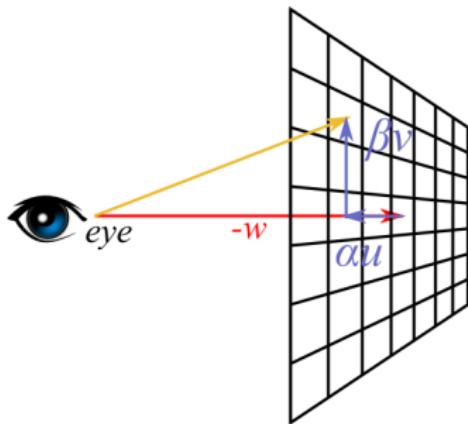
- Legyen \mathbf{p} az i, j pixel középpontja, a vetítő sík egységnyi távolságra a nézőponttól! Ekkor

$$\mathbf{p}(i, j) = \mathbf{eye} + (\alpha\mathbf{u} + \beta\mathbf{v} - \mathbf{w}).$$

- Ahol

$$\alpha = \tan\left(\frac{\text{fov}x}{2}\right) \cdot \frac{i - \text{width}/2}{\text{width}/2},$$

$$\beta = \tan\left(\frac{\text{fov}y}{2}\right) \cdot \frac{\text{height}/2 - j}{\text{height}/2}.$$



A sugár egyenlete

- A sugár egy félegyenes, amit kezdőpontjával és irányvektorával adhatunk meg.
- Legyen \mathbf{p}_0 a sugár kezdőpontja, \mathbf{v} pedig az irányvektora, ekkor

$$\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}, \quad t \geq 0$$

megadja a sugár összes pontját.

- Most a sugarak kezdőpontját az előbbieknél megfelelően számoljuk, azaz $\mathbf{p}_0 = \mathbf{p}(i, j)$
- A sugár irányvektora pedig $\mathbf{v} = \frac{\mathbf{p}(i, j) - \text{eye}}{|\mathbf{p}(i, j) - \text{eye}|}$

Tartalom

1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- **Metszések**
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

2 Rekurzív sugárkövetés

Metszések

- A sugárkövető programok futásidejük döntő részében metszéseket fognak végezni
- Nézzük meg néhány egyszerű geometriai elemmel vett metszetét a sugárnak
- A sugarunk mindenkor a fent is látott $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$ alakú, ahol feltesszük a továbbiakban, hogy $|\mathbf{v}| = 1$

Metszések: parametrikus sugár-implicit felület

- Legyen adva egy $f(\mathbf{x}) = 0$ implicit egyenlet, ami meghatározza a metszeni kívánt felületünket ($\mathbf{x} \in \mathbb{R}^3$)
- A sugarunk egyenlete $\forall t \in [0, \infty)$ -re meghatároz egy pontot a térben \rightarrow tegyük be ezt a képletet az implicit egyenletbe!
- Tehát a következő egyenletet kell megoldanunk t -re:

$$f(\mathbf{p}(t)) = 0$$

- A kapott t -től függően a következő esetek állhatnak fenn:
 - Ha $t > 0$, akkor a sugarunk előtt van a felület és metszi
 - Ha $t = 0$ a sugár a felületről indul
 - Ha $t < 0$, akkor a sugár "mögött" van a felület és metszi a sugár egyenese a felületet (de nekünk $t > 0$ kell!)

Metszések: parametrikus sugár-parametrikus felület

- Legyen adva egy $\mathbf{r}(u, v) = [r_x(u, v), r_y(u, v), r_z(u, v)]^T$ parametrikus felület
- Kell: találni egy olyan t sugárparamétert, amihez létezik (u, v) , hogy

$$\mathbf{r}(t) = \mathbf{r}(u, v)$$

- Ez három ismeretlenes (t, u, v) , három egyenletes (x, y, z) koordinátánként egy) egyenletrendszer
- A t ugyanúgy ellenőrizendő, mint előbb, de most az (u, v) -re is figyeljünk, hogy a felületünk paramétertartományának megengedett részén van-e (általában $(u, v) \in [0, 1]^2$ kell)!

Tartalom

1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- **Metszések**
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

2 Rekurzív sugárkövetés

Egyenes és implicit sík metszéspontja

- Síkot megadhatunk implicit alakban: $Ax + By + Cz + D = 0$
- A

$$\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + t \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

sugár egyenese metszi a síkot, ha

$$A(x_0 + tx) + B(y_0 + ty) + C(z_0 + tz) + D = 0$$

Egyenes és implicit sík metszéspontja

- Ezt t -re átrendezve adódik

$$t(Ax + By + Cz) + x_0 + y_0 + z_0 + D = 0$$

$$t = -\frac{x_0 + y_0 + z_0 + D}{Ax + By + Cz}$$

- Látható a sík a nézőpontunkból, ha $t > 0$

Egyenes és normálvektoros sík metszéspontja

- Legyen \mathbf{q}_0 a sík egy pontja, \mathbf{n} a normálvektora,
- Legyen \mathbf{p}_0 ez egyenes egy pontja, \mathbf{v} az irányvektora.
- Az egyenes egyenlete:

$$\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$$

- A sík egyenlete:

$$\langle \mathbf{n}, \mathbf{q} - \mathbf{q}_0 \rangle = 0$$

- minden \mathbf{q} pontja a síknak kielégíti ezt az egyenletet

Egyenes és normálvektoros sík metszéspontja

- Behelyettesítve $\mathbf{p}(t)$ -t a \mathbf{q} helyére:

$$\langle \mathbf{n}, \mathbf{p}_0 + t\mathbf{v} - \mathbf{q}_0 \rangle = 0,$$

$$\langle \mathbf{n}, \mathbf{p}_0 \rangle + t\langle \mathbf{n}, \mathbf{v} \rangle - \langle \mathbf{n}, \mathbf{q}_0 \rangle = 0,$$

$$t = \frac{\langle \mathbf{n}, \mathbf{q}_0 \rangle - \langle \mathbf{n}, \mathbf{p}_0 \rangle}{\langle \mathbf{n}, \mathbf{v} \rangle} = \frac{\langle \mathbf{n}, \mathbf{q}_0 - \mathbf{p}_0 \rangle}{\langle \mathbf{n}, \mathbf{v} \rangle},$$

ha $\langle \mathbf{n}, \mathbf{v} \rangle \neq 0$.

- A sugár metszi a síkot, ha: $t > 0$.
- Ha $\langle \mathbf{n}, \mathbf{v} \rangle = 0$, akkor az egyenes párhuzamos a síkkal, és így vagy nincs metszéspontjuk, vagy az egyenes a síkon fut

Egyenes és parametrikus sík metszéspontja

- Síkot megadhatunk egy \mathbf{q} pontjával és \mathbf{i}, \mathbf{j} kifeszítő vektorokkal is: $\mathbf{s}(u, v) = \mathbf{q} + u\mathbf{i} + v\mathbf{j}$
- Metszéspont a $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$ sugár egyenesével: keressük t és u, v -t úgy, hogy

$$\mathbf{p}(t) = \mathbf{s}(u, v)$$

- Beírva a képleteket adódik

$$\mathbf{p}_0 + t\mathbf{v} = \mathbf{q} + u\mathbf{i} + v\mathbf{j}$$

- Átrendezve kapjuk, hogy

$$\mathbf{p}_0 - \mathbf{q} = -t\mathbf{v} + u\mathbf{i} + v\mathbf{j}$$

Egyenes és parametrikus sík metszéspontja

- Ez három ismeretlenes, három lineáris egyenletből álló egyenletrendszer, ami megoldható, ha $\mathbf{v}, \mathbf{i}, \mathbf{j}$ lineárisan nem összefüggő
- Mátrix alakban:

$$\begin{bmatrix} p_{0x} - q_x \\ p_{0y} - q_y \\ p_{0z} - q_z \end{bmatrix} = \begin{bmatrix} -v_x & i_x & j_x \\ -v_y & i_y & j_y \\ -v_z & i_z & j_z \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix}$$

- Látjuk a síket, ha $t > 0$ (most $u, v \in \mathbb{R}$ a felület paramétertartománya, ez teljesülni fog)

Tartalom

1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- **Metszések**
 - Sugár és sík metszéspontja
 - **Sugár és háromszög metszéspontja**
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

2 Rekurzív sugárkövetés

Háromszög megadása

- Egyértelműen megadható három csúcsával.
- Ha $\mathbf{a}, \mathbf{b}, \mathbf{c}$ a háromszög csúcsai, akkor a hozzáartozó sík pont-normálvektoros implicit megadásához a sík
 - egy pontja $\mathbf{a}, \mathbf{b}, \mathbf{c}$ bármelyike
 - normálvektora

$$\mathbf{n} = \frac{(\mathbf{c} - \mathbf{a}) \times (\mathbf{b} - \mathbf{a})}{\|(\mathbf{c} - \mathbf{a}) \times (\mathbf{b} - \mathbf{a})\|},$$

ahol \times a vektoriális szorzást jelöli, és ekkor \mathbf{n} egységnyi hosszúságú.

Háromszög és egyenes metszéspontja

- Először számítsuk ki az egyenes és a háromszög síkjának metszéspontját, ez legyen \mathbf{p} (már ha létezik).
- Legyenek $\lambda_1, \lambda_2, \lambda_3$ a \mathbf{p} pont háromszögön belüli baricentrikus koordinátái, úgy hogy

$$\mathbf{p} = \lambda_1 \mathbf{a} + \lambda_2 \mathbf{b} + \lambda_3 \mathbf{c}$$

- p Akkor, és csak akkor van a \triangle -ön belül, ha

$$0 \leq \lambda_1, \lambda_2, \lambda_3 \leq 1.$$

Pont a háromszögön vizsgálat

- Tudjuk, hogy $\mathbf{p} = [x, y, z]^T = \lambda_1 \mathbf{a} + \lambda_2 \mathbf{b} + \lambda_3 \mathbf{c}$. Ekkor

$$x = \lambda_1 a_x + \lambda_2 b_x + \lambda_3 c_x$$

$$y = \lambda_1 a_y + \lambda_2 b_y + \lambda_3 c_y$$

$$z = \lambda_1 a_z + \lambda_2 b_z + \lambda_3 c_z,$$

$$\text{ill. } \lambda_1 + \lambda_2 + \lambda_3 = 1 \Rightarrow \lambda_3 = 1 - \lambda_1 - \lambda_2$$

Pont a háromszögön vizsgálat

- Tudjuk, hogy $\mathbf{p} = [x, y, z]^T = \lambda_1 \mathbf{a} + \lambda_2 \mathbf{b} + \lambda_3 \mathbf{c}$. Ekkor

$$x = \lambda_1 a_x + \lambda_2 b_x + \lambda_3 c_x$$

$$y = \lambda_1 a_y + \lambda_2 b_y + \lambda_3 c_y$$

$$z = \lambda_1 a_z + \lambda_2 b_z + \lambda_3 c_z,$$

$$\text{ill. } \lambda_1 + \lambda_2 + \lambda_3 = 1 \Rightarrow \lambda_3 = 1 - \lambda_1 - \lambda_2$$

- A gyorsabb számolásért vegyük a fentinek egy síkra vett vetületét

Pont a háromszögön vizsgálat

- Tudjuk, hogy $\mathbf{p} = [x, y, z]^T = \lambda_1 \mathbf{a} + \lambda_2 \mathbf{b} + \lambda_3 \mathbf{c}$. Ekkor

$$x = \lambda_1 a_x + \lambda_2 b_x + \lambda_3 c_x$$

$$y = \lambda_1 a_y + \lambda_2 b_y + \lambda_3 c_y$$

$$z = \lambda_1 a_z + \lambda_2 b_z + \lambda_3 c_z,$$

$$\text{ill. } \lambda_1 + \lambda_2 + \lambda_3 = 1 \Rightarrow \lambda_3 = 1 - \lambda_1 - \lambda_2$$

- A gyorsabb számolásért vegyük a fentinek egy síkra vett vetületét
- A koordinátaíkok közül (XY , XZ vagy YZ) arra vegyük a háromszög 2D vetületét, amelyre a háromszög vetületének területe a legnagyobb! \rightarrow a háromszög és a sík normálisa leginkább "egyállású"

Pont a háromszögön vizsgálat

- Tudjuk, hogy $\mathbf{p} = [x, y, z]^T = \lambda_1 \mathbf{a} + \lambda_2 \mathbf{b} + \lambda_3 \mathbf{c}$. Ekkor

$$x = \lambda_1 a_x + \lambda_2 b_x + \lambda_3 c_x$$

$$y = \lambda_1 a_y + \lambda_2 b_y + \lambda_3 c_y$$

$$z = \lambda_1 a_z + \lambda_2 b_z + \lambda_3 c_z,$$

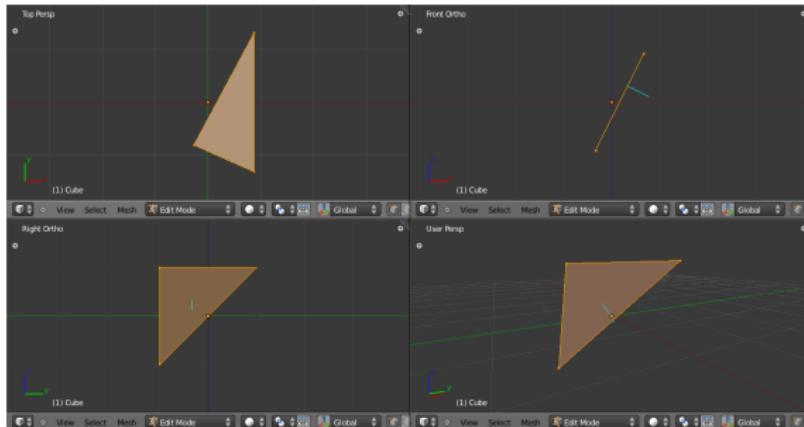
$$\text{ill. } \lambda_1 + \lambda_2 + \lambda_3 = 1 \Rightarrow \lambda_3 = 1 - \lambda_1 - \lambda_2$$

- A gyorsabb számolásért vegyük a fentinek egy síkra vett vetületét
- A koordinátaíkok közül (XY , XZ vagy YZ) arra vegyük a háromszög 2D vetületét, amelyre a háromszög vetületének területe a legnagyobb! \rightarrow a háromszög és a sík normálisa leginkább "egyállású"
- A vetülethez egyszerűen elhagyjuk z , y vagy x egyenletét, megfelelően.

Pont a háromszögön vizsgálat

Azt tengely kell választani, amelyik mentén a legnagyobb a háromszög normálvektorának abszolút értéke.

(Így biztos nem fordulhat elő, hogy a háromszög merőleges a síkra, és csak egy szakasz marad belőle!)



Pont a háromszögön vizsgálat

- Pl. legyen a z a választott tengely. Ekkor

$$x = \lambda_1 a_x + \lambda_2 b_x + \lambda_3 c_x$$

$$y = \lambda_1 a_y + \lambda_2 b_y + \lambda_3 c_y$$

Pont a háromszögön vizsgálat

- Pl. legyen a z a választott tengely. Ekkor

$$x = \lambda_1 a_x + \lambda_2 b_x + \lambda_3 c_x$$

$$y = \lambda_1 a_y + \lambda_2 b_y + \lambda_3 c_y$$

- Behelyettesítve $\lambda_3 = 1 - \lambda_1 - \lambda_2$ -t, és rendezve:

$$x = \lambda_1(a_x - c_x) + \lambda_2(b_x - c_x) + c_x$$

$$y = \lambda_1(a_y - c_y) + \lambda_2(b_y - c_y) + c_y$$

Pont a háromszögön vizsgálat

- Rendezve λ_1, λ_2 -re kapjuk:

$$\lambda_1 = \frac{(b_y - c_y)(x - c_x) - (b_x - c_x)(y - c_y)}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)}$$
$$\lambda_2 = \frac{-(a_y - c_y)(x - c_x) - (a_x - c_x)(y - c_y)}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)}$$

Pont a háromszögön vizsgálat

- Rendezve λ_1, λ_2 -re kapjuk:

$$\lambda_1 = \frac{(b_y - c_y)(x - c_x) - (b_x - c_x)(y - c_y)}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)}$$
$$\lambda_2 = \frac{-(a_y - c_y)(x - c_x) - (a_x - c_x)(y - c_y)}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)}$$

- A nevező csak degenerált háromszög esetén lehet nulla.

Pont a háromszögön vizsgálat

- Rendezve λ_1, λ_2 -re kapjuk:

$$\lambda_1 = \frac{(b_y - c_y)(x - c_x) - (b_x - c_x)(y - c_y)}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)}$$
$$\lambda_2 = \frac{-(a_y - c_y)(x - c_x) - (a_x - c_x)(y - c_y)}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)}$$

- A nevező csak degenerált háromszög esetén lehet nulla.
- **p** akkor és csak akkor van a háromszögön belül, ha

$$0 \leq \lambda_1, \lambda_2, \lambda_3 \leq 1.$$

Tartalom

1 Raycasting

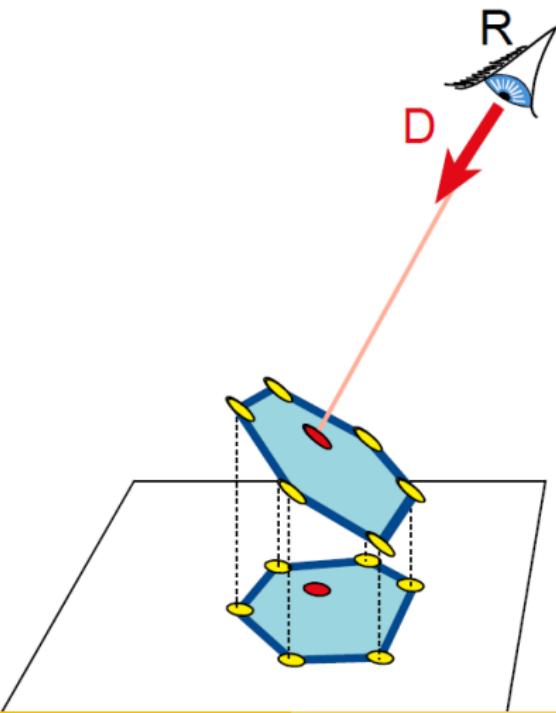
- Motiváció
- Raycasting
- Sugarak indítása
- **Metszések**
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - **Sugár és poligon metszéspontja**
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

2 Rekurzív sugárkövetés

Sugár metszése poligonnal

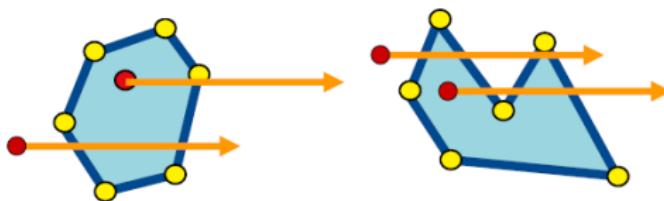
- Tegyük fel, hogy a poligonunk csúcsai egy síkban vannak, ekkor a metszés két lépésben
 - A sugarunkat metszük el a poligon síkjával
 - Döntsük el, hogy a metszéspont a poligonon belül van-e
- A másodikat egy síkban érdemes csinálni (vagy a poligon síkjában, vagy a poligon valamely koordinátatengelyre vett vetületének síkjában)

Sugár metszése poligonnal



Pont-poligon tartalmazás teszt síkban

- A pont a poligonon belül van, ha tetszőleges irányú, belőle indított sugárnak páratlan számú metszéspontja van a poligon oldalaival (azaz a sugarat a poligon összes oldalszakaszával el kell metszeni)
- Konkáv és csillag alagú poligonra is működik



Sugár metszése szakasszal

- A poligon $\mathbf{d}_i = (x_i, y_i), \mathbf{d}_{i+1} = (x_{i+1}, y_{i+1})$ csúcspontjai közötti szakasz parametrikus alakja:
$$\mathbf{d}_{i,i+1}(s) = (1 - s)\mathbf{d}_i + s\mathbf{d}_{i+1} = \mathbf{d}_i + s(\mathbf{d}_{i+1} - \mathbf{d}_i), s \in [0, 1]$$
- Ezt kell metszeni a $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$ alakú sugárral
- Most: a $\mathbf{p}_0 = (x_0, y_0)$ pont az a pont, amiről el akarjuk döntení, hogy a poligonon belül van-e, \mathbf{d} tetszőleges
- Legyen $\mathbf{v} = (1, 0)!$
- Így a $\mathbf{p}(t) = \mathbf{d}_{i,i+1}(s)$ egyenletet csak y koordinátára kell megoldani

Sugár metszése szakasszal

- Keressük meg, hogy hol metszi a $\mathbf{d}_{i,i+1}(s)$ oldal egyenese a sugarat (=melyik s -re lesz $d_{i,i+1}(s)_y = y_0$?)
- Azaz $y_0 = y_i + s(y_{i+1} - y_i)$
- s -t kifejezve: $s = \frac{y_0 - y_i}{y_{i+1} - y_i}$
- Innen megkapjuk azt az x koordinátát $\mathbf{d}_{i,i+1}(s)$ -be behelyettesítve, ahol a sugár metszi a szakaszt.
- Ha $s \notin [0, 1]$: a sugár nem metszi a szakaszt (csak az egyenesét)
- Ha $t \leq 0$: a sugár egybeesik a szakasszal, vagy mögötte van a metszéspont

Tartalom

1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- **Metszések**
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - **Sugár és gömb metszéspontja**
 - Sugár és doboz metszéspontja
 - Transzformált objektumok

2 Rekurzív sugárkövetés

A gömb egyenlete

- Az r sugarú, $\mathbf{c} = (c_x, c_y, c_z)$ középpontú gömb implicit egyenlete:

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 - r^2 = 0$$

- Ugyanez skalárszorzattal felírva:

$$\langle \mathbf{p} - \mathbf{c}, \mathbf{p} - \mathbf{c} \rangle - r^2 = 0,$$

ahol $\mathbf{p} = (x, y, z)$.

Gömb és egyenes metszéspontja

- Legyen \mathbf{p}_0 ez egyenes egy pontja, \mathbf{v} az irányvektora.
- Ekkor az egyenes egyenlete:

$$\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$$

- Behelyettesítve a gömb egyenletébe, kapjuk:

$$\langle \mathbf{p}_0 + t\mathbf{v} - \mathbf{c}, \mathbf{p}_0 + t\mathbf{v} - \mathbf{c} \rangle - r^2 = 0$$

- Kifejtve:

$$t^2 \langle \mathbf{v}, \mathbf{v} \rangle + 2t \langle \mathbf{v}, \mathbf{p}_0 - \mathbf{c} \rangle + \langle \mathbf{p}_0 - \mathbf{c}, \mathbf{p}_0 - \mathbf{c} \rangle - r^2 = 0$$

$$t^2 \langle \mathbf{v}, \mathbf{v} \rangle + 2t \langle \mathbf{v}, \mathbf{p}_0 - \mathbf{c} \rangle + \langle \mathbf{p}_0 - \mathbf{c}, \mathbf{p}_0 - \mathbf{c} \rangle - r^2 = 0$$

- Ez másodfokú egyenlet t -re (minden más ismert).
- Legyen $D = (2\langle \mathbf{v}, \mathbf{p}_0 - \mathbf{c} \rangle)^2 - 4\langle \mathbf{v}, \mathbf{v} \rangle(\langle \mathbf{p}_0 - \mathbf{c}, \mathbf{p}_0 - \mathbf{c} \rangle - r^2)$
- Ha $D > 0$: két megoldás van, az egyenes metszi a gömböt.
- Ha $D = 0$: egy megoldás van, az egyenes érinti a gömböt.
- Ha $D < 0$: nincs valós megoldás, az egyenes nem metszi a gömböt.

Tartalom

1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- **Metszések**
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - **Sugár és doboz metszéspontja**
 - Transzformált objektumok

2 Rekurzív sugárkövetés

Sugár metszése AAB-vel

- AAB = axis aligned box, olyan téglalap, aminek az oldallapjai a koordináta síkjainkkal párhuzamosak
- Legyen a sugarunk $\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{v}$ alakú, ahol $\mathbf{p}_0 = (x_0, y_0)$, $\mathbf{v} = (v_x, v_y)$ a téglalapot pedig adjuk meg átlójának két pontjával, **a** és **b** segítségével ($\mathbf{a} < \mathbf{b}$)!
- Tegyük fel, hogy a sugár kiindulópontja a doboztól balra helyezkedik el (ezt megtehetjük)

Sugár metszése AAB-vel

- Ha $v_x = 0$: vízszintes a sugarunk, nincs metszéspont, ha $x_0 \notin [a_x, b_x]$, különben triviális eldönteni.
- Ha $v_x \neq 0$, akkor a két paraméter (négyzet bal és jobb oldala): $t_1 := \frac{a_x - x_0}{v_x}$, $t_2 := \frac{b_x - x_0}{v_x}$
- y és z koordinátákra hasonlóan el kell végezni a számítást.
- Ezek alapján megállapítható, hogy melyik két lapját a téglatestnek hol metszi a sugár.

Tartalom

1 Raycasting

- Motiváció
- Raycasting
- Sugarak indítása
- **Metszések**
 - Sugár és sík metszéspontja
 - Sugár és háromszög metszéspontja
 - Sugár és poligon metszéspontja
 - Sugár és gömb metszéspontja
 - Sugár és doboz metszéspontja
 - **Transzformált objektumok**

2 Rekurzív sugárkövetés

Transzformált objektumok

- Legyen \mathbf{M} egy adott objektum transzformációs mátrixa.
- Feladat: Keressük \mathbf{r} sugár és az \mathbf{M} -mel transzformált objektum metszéspontját!
- Probléma: Hogyan transzformálunk egy gömböt?
Pontonként? Képletet írjuk át? ...
- Megoldás: Transzformáljuk inkább a sugarat!

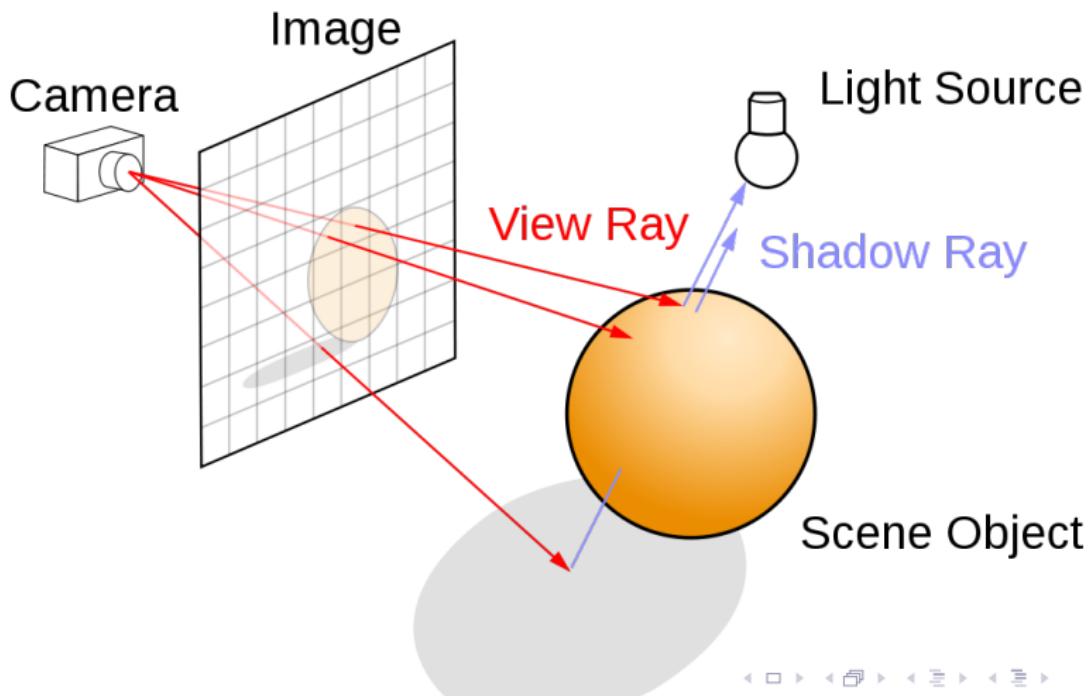
Tétel

Az r sugár és az \mathbf{M} -mel transzformált objektum metszéspontja \equiv az \mathbf{M}^{-1} -zel transzformált \mathbf{r} sugár és az objektum metszéspontja.

- $\mathbf{M} \in \mathbb{R}^{4 \times 4}$, homogén transzformáció
- Sugár kezdőpontja: $\mathbf{p}_0 = (p_x, p_y, p_z) \rightarrow [p_x, p_y, p_z, 1]$
- Sugár iránya: $\mathbf{v} = (v_x, v_y, v_z) \rightarrow [v_x, v_y, v_z, 0]$. Így nem hat rá az eltolás.
- Transzformált sugár $\mathbf{r}'(t) : \mathbf{M}^{-1}\mathbf{p} + t \cdot \mathbf{M}^{-1}\mathbf{v}$

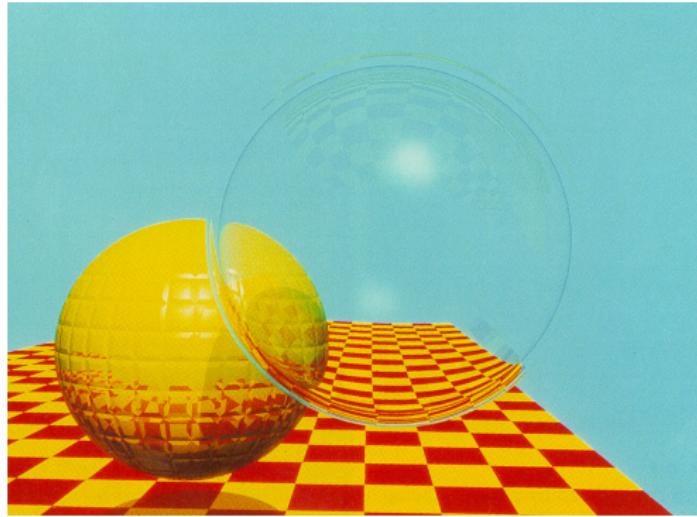
- Metszésvizsgálat: használjuk $\mathbf{r}'(t)$ -t!
- Metszéspont: \mathbf{q} , akkor az eredeti térben $\mathbf{M} \cdot \mathbf{q}$.

Rekurzív sugárkövetés



Egyszerűsített illuminációs egyenlet

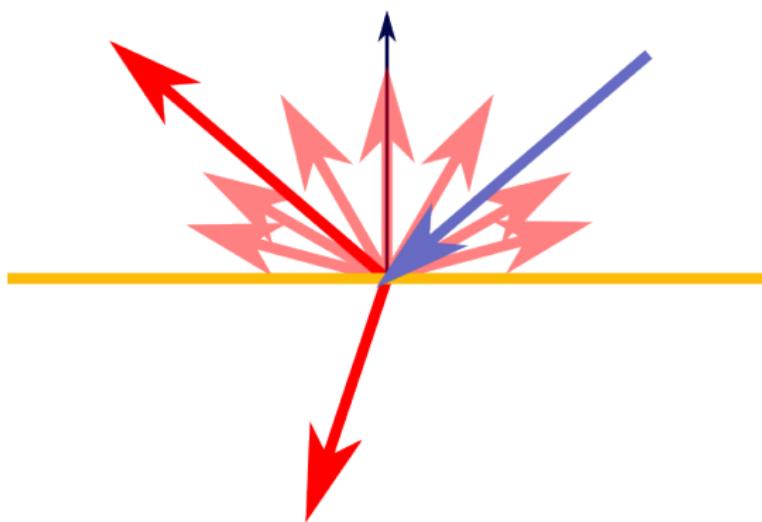
Minden pixelre egymástól függetlenül határozzuk meg azok színét
– oldjuk meg az árnyalási és takarási feladatot.



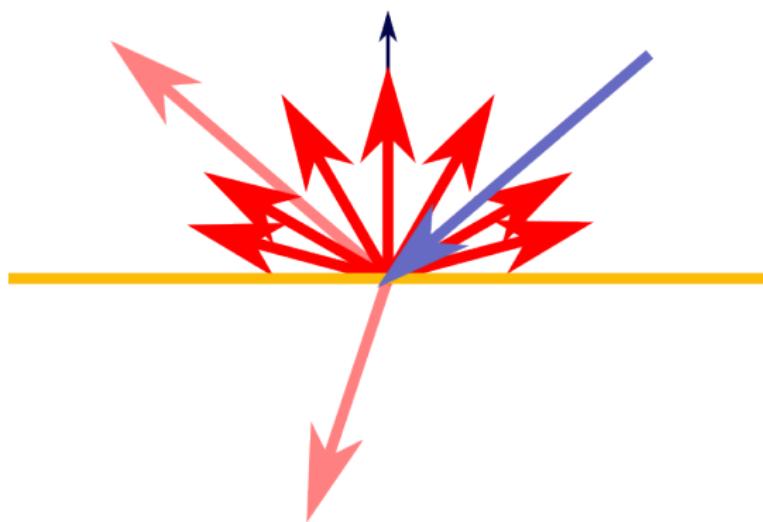
Turner Whitted, 1980

- A fény útját két féle komponensre bontjuk: *koherens* és *inkoherens* komponensre
- *Koherens* eset
 - Az optikának megfelelő ideális visszaverődés ("tükröződés") és törés
 - Tovább követjük a fény útját
- *Inkoherens* eset
 - minden egyéb
 - Csak az absztrakt fényforrás direkt megvilágítását vesszük figyelembe

Koherens komponens



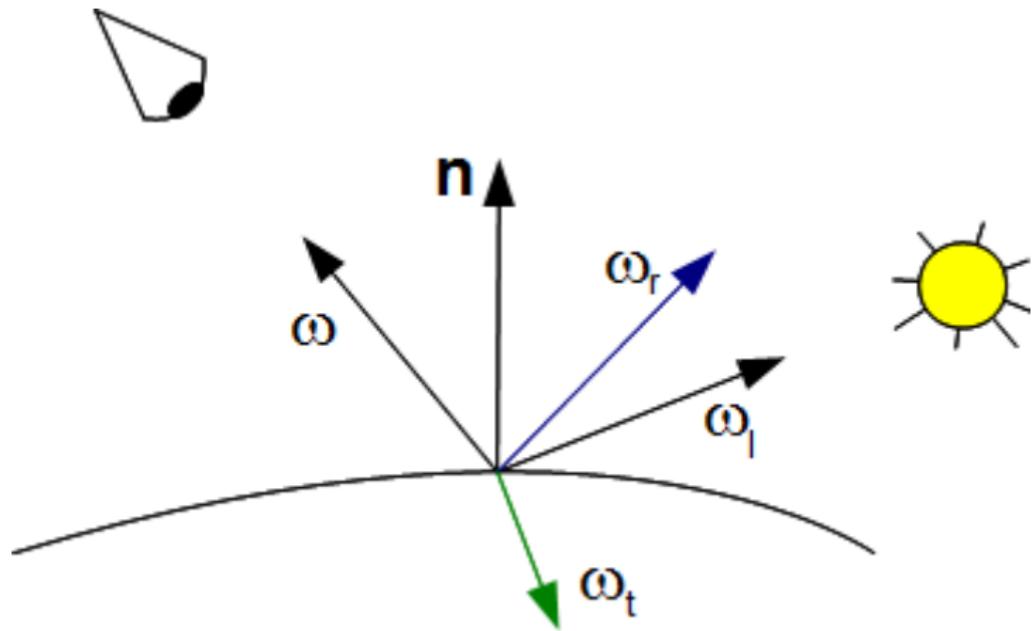
Inkoherens komponens



Egyszerűsített illuminációs egyenlet

A következő, egyszerűsített megvilágítási egyenletet oldjuk meg:

$$\begin{aligned} L(\mathbf{x}, \omega) = & L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{I \in \text{Lights}} f_r(\mathbf{x}, \omega_I, \omega) L_i(\mathbf{x}, \omega_I) (-\omega_I \cdot \mathbf{n}) \\ & + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t) \end{aligned}$$



Sugárkövetés

$$L(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{I \in \text{Lights}} f_r(\mathbf{x}, \omega_I, \omega) L_i(\mathbf{x}, \omega_I) (-\omega_I \cdot \mathbf{n}) + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t)$$

- A szempozicóból sugarakat indítunk minden pixel középpontján keresztül.
- Ennek a sugárnak az irányát adja meg $-\omega$ -t (**minusz omega!**).
- A sugar és a színtér objetumainak szemhez legközelebbi metszéspontja adja meg \mathbf{x} -et.

Emisszió

$$L(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{l \in \text{Lights}} f_r(\mathbf{x}, \omega_l, \omega) L_i(\mathbf{x}, \omega_l) (-\omega_l \cdot \mathbf{n}) + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t)$$

Az \mathbf{x} felületi pontból, az ω nézeti irányból érkező radiancia, a felület saját sugárzása – *emissziója* – miatt.

Ambiens fény

$$L(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{I \in \text{Lights}} f_r(\mathbf{x}, \omega_I, \omega) L_i(\mathbf{x}, \omega_I) (-\omega_I \cdot \mathbf{n}) + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t)$$

k_a a felület, L_a a környezet *ambiens* együtthatója.

Az egyenlet *ambiens* tagja közelíti azt a fénymennyiséget, ami általánosan jelen van, minden felületet ér, azok helyzetétől és az absztrakt fényforrásoktól függetlenül.

Fényforrások

$$L(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{I \in \text{Lights}} f_r(\mathbf{x}, \omega_I, \omega) L_I(\mathbf{x}, \omega_I) (-\omega_I \cdot \mathbf{n}) + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t)$$

- Az inkohérent visszarödéseket foglalja össze a szummás tag
- Csak a fényforrások direkt hatását vesszük figyelembe
- És csak akkor, ha az az \mathbf{x} felületi pontból látszik

Fényforrások

$$\begin{aligned} L(\mathbf{x}, \omega) = & L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{l \in \text{Lights}} f_r(\mathbf{x}, \omega_l, \omega) L_i(\mathbf{x}, \omega_l) (-\omega_l \cdot \mathbf{n}) \\ & + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t) \end{aligned}$$

- ω_l a fényforrásból a felületi pontba mutató egységvektor.
- $f_r(\mathbf{x}, \omega_l, \omega)$ most csak a diffúz és spekuláris visszaverődést jellemző BRDF.
- $-\omega_l \cdot \mathbf{n}$ a felületi normális és a fényforrás fele mutató vektor által bezárt szög koszinusza.

Fényforrások

$$\begin{aligned} L(\mathbf{x}, \omega) = & L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{I \in \text{Lights}} f_r(\mathbf{x}, \omega_I, \omega) L_i(\mathbf{x}, \omega_I) (-\omega_I \cdot \mathbf{n}) \\ & + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t) \end{aligned}$$

- Ha az I fényforrás teljesítménye Φ_I , és pozíciója \mathbf{x}_I akkor

$$L_i(\mathbf{x}, \omega_I) = v(\mathbf{x}, \mathbf{x}_I) \cdot \frac{\Phi_I}{\|\mathbf{x} - \mathbf{x}_I\|^2}.$$

- $v(\mathbf{x}, \mathbf{x}_I) \in [0, 1]$ függvény: *Mi van a felületi pont és a fényforrás között?*

Fényforrások

$$\begin{aligned} L(\mathbf{x}, \omega) = & L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{I \in \text{Lights}} f_r(\mathbf{x}, \omega_I, \omega) L_I(\mathbf{x}, \omega_I) (-\omega_I \cdot \mathbf{n}) \\ & + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t) \end{aligned}$$

$v(\mathbf{x}, \mathbf{x}_I) \in [0, 1]$ függvény

- $= 0$, ha a fényforrás nem látható \mathbf{x} -ből,
- $= 1$, ha igen,
- $\in (0, 1)$, ha átlátszó objektumok vannak a kettő között.
- v kiszámításához úgynevezett árnyéksugarat indítunk \mathbf{x} -ből \mathbf{x}_I -fele, és az objektumokkal való metszését nézzük.

Tükroződés

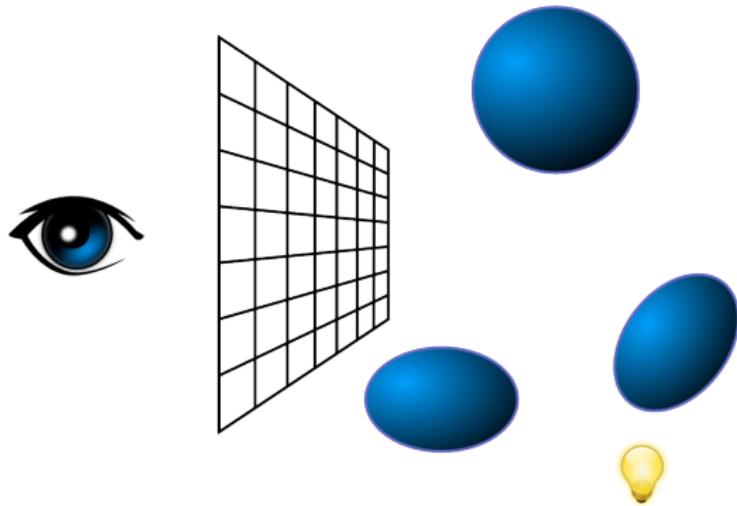
$$\begin{aligned} L(\mathbf{x}, \omega) = & L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{I \in \text{Lights}} f_r(\mathbf{x}, \omega_I, \omega) L_I(\mathbf{x}, \omega_I) (-\omega_I \cdot \mathbf{n}) \\ & + k_r \cdot L(\mathbf{x}, \omega_r) + k_t \cdot L(\mathbf{x}, \omega_t) \end{aligned}$$

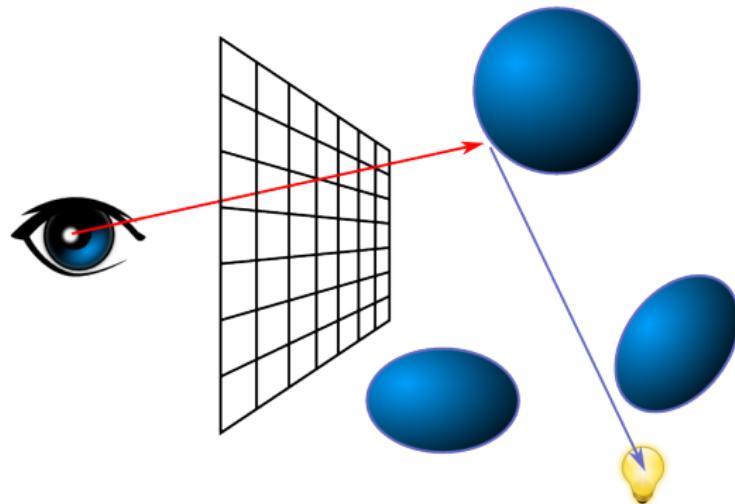
- A tükörirányból érkező fényt k_r arányban vesszük figyelembe.
- ω_r az ideális tüköriránynak megfelelő **beeső** vektor.
- $L(\mathbf{x}, \omega_r)$ kiszámítása azonos $L(\mathbf{x}, \omega)$ kiszámításával (rekurzió!).
- Új sugár: szempozíció helyett \mathbf{x} , és a sugár iránya $-\omega_r$.

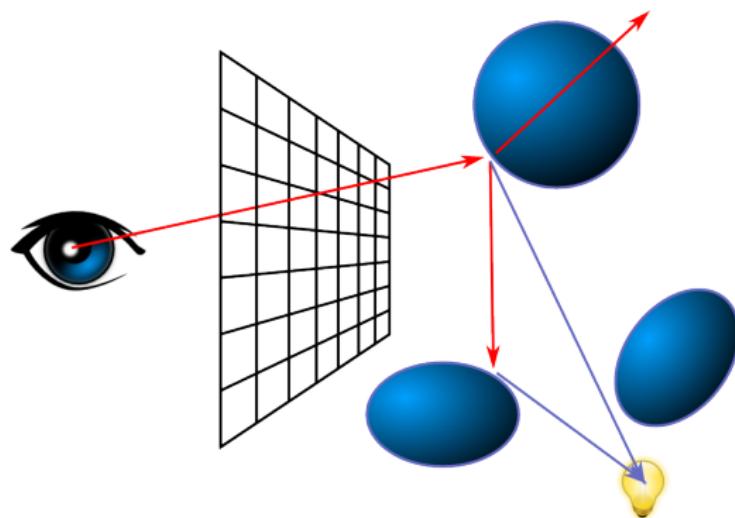
Fénytörés

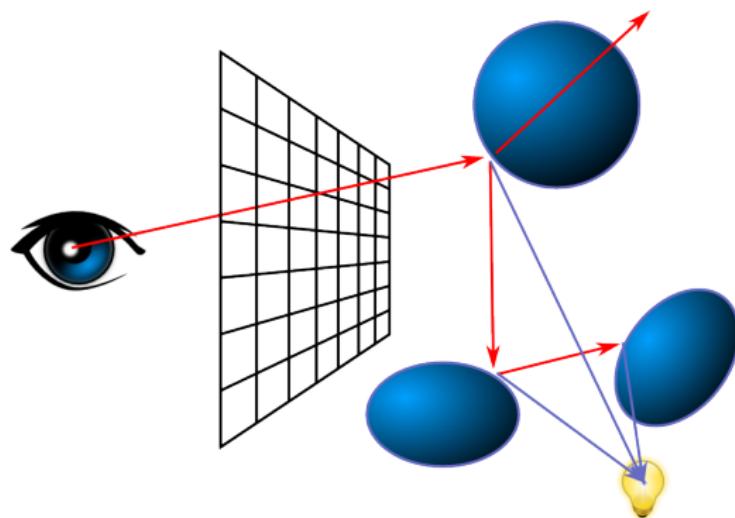
$$\begin{aligned} L(\mathbf{x}, \omega) = & L_e(\mathbf{x}, \omega) + k_a \cdot L_a + \sum_{l \in \text{Lights}} f_r(\mathbf{x}, \omega_l, \omega) L_i(\mathbf{x}, \omega_l) (-\omega_l \cdot \mathbf{n}) \\ & + k_r \cdot L(\mathbf{x}, \omega_r) + \color{red}k_t \cdot L(\mathbf{x}, \omega_t) \end{aligned}$$

- A törési-irányból érkező fényt k_t arányban vesszük figyelembe.
- ω_t a törésiránynak megfelelő **beeső** vektor.
- $L(\mathbf{x}, \omega_t)$ kiszámítása megint azonos $L(\mathbf{x}, \omega)$ kiszámításával (rekurzió!).
- Új sugár: szempozíció helyett \mathbf{x} , és a sugár iránya $-\omega_t$.









Számítógépes Grafika

Hajder Levente
hajder@inf.elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2017/2018. II. félév

Tartalom

1 Tartalom

- Motiváció

2 Grafikus szerelőszalag

- Áttekintés
 - Modellezési transzformáció
 - Nézeti transzformáció
 - Perspektív transzformáció
 - Vágás
 - Raszterizáció
 - Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer

3 Lokális illumináció

Tartalom

1 Tartalom

- #### • Motiváció

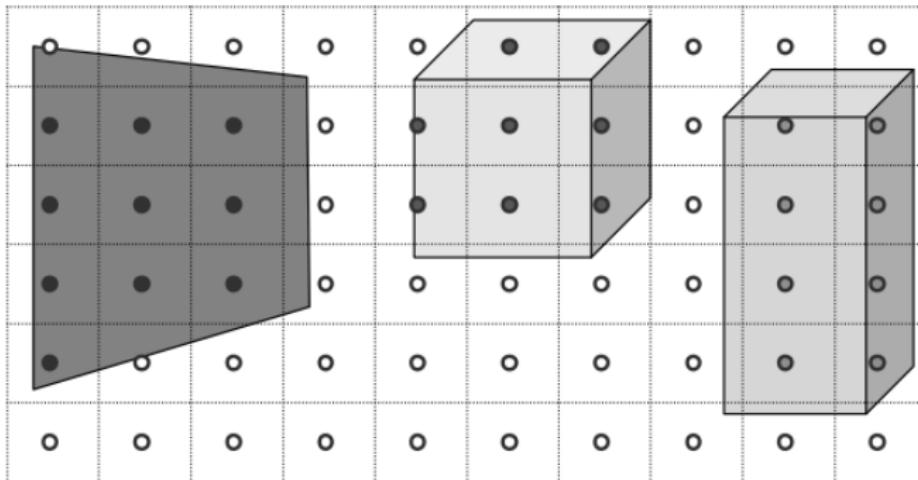
2 Grafikus szerelőszalag

- Áttekintés
 - Modellezési transzformáció
 - Nézeti transzformáció
 - Perspektív transzformáció
 - Vágás
 - Raszterizáció
 - Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer

3 Lokális illumináció

Emlékeztető

- Múlt órán megismerkedtünk a sugárkövetéssel
 - Előnyei:
 - A színtér benépesítésére minden használható, ami metszhető sugárral
 - Rekurzióval könnyen implementálható programmal
 - A fény részecske tulajdonságaiból következő hatások szimulálása
 - Ugyanakkor láttuk, hogy vannak hátrányai is:
 - minden pixelnél minden primitívvel kellett tesztelnünk → ezen próbáltunk gyorsítani (dobozolás, térfelosztás)
 - Globális jellegű az algoritmus, nehezen gyorsítható hardveresen
 - A fény hullám természetéből adódó jelenségeket nem tudja visszaadni
 - Valósidejű alkalmazásokhoz túl lassú



Valósidejű grafika

- Sugárkövetésnél tehát ez volt: \forall pixelre indítsunk sugarat: \forall objektummal nézzük van-e metszés
 - Ehelyett próbáljuk meg ezt: \forall objektumra (primitívre): számoljuk ki, mely pixelekre képeződik le és végül csak a legközelebbi jelenítsük meg!

Inkrementális képszintézis – Fogalmak

- *Koherencia*: Pixelek helyett nagyobb logikai egységekből, *primitívekből* indulunk ki
 - *Pontosság*: *objektum tér pontosság* ("pixel pontosság" helyett)
 - *Vágás*: képernyőből kilógó elemekre ne számoljunk feleslegesen
 - *Inkrementális elv*: az árnyalási és takarási feladatnál kihasználjuk a nagyobb egységenként szerzett információkat.

Összehasonlítás

Sugárkövetés

- pixelenként számol

Inkrementális képszintézis

- primitívenként számol

Összehasonlítás

Sugárkövetés

- pixelenként számol
 - amit lehet sugárral metszeni, az használható

Inkrementális képszintézis

- primitívenként számol
 - ami nem primitív, azt azzal kell közelíteni

Összehasonlítás

Sugárkövetés

- pixelenként számol
 - amit lehet sugárral metszeni, az használható
 - van tükröződés, fénytörés, vetett árnyékok

Inkrementális képszintézis

- primitívenként számol
 - ami nem primitív, azt azzal kell közelíteni
 - külön algoritmus kell ezekhez

Összehasonlítás

Sugárkövetés

- pixelenként számol
 - amit lehet sugárral metszeni, az használható
 - van tükröződés, fénytörés, vetett árnyékok
 - takarási feladat triviális

Inkrementális képszintézis

- primitívenként számol
 - ami nem primitív, azt azzal kell közelíteni
 - külön algoritmus kell ezekhez
 - külön meg kell oldani

Összehasonlítás

Sugárkövetés

- pixelenként számol
 - amit lehet sugárral metszeni, az használható
 - van tükröződés, fénytörés, vetett árnyékok
 - takarási feladat triviális
 - sok pixel, sok sugár miatt nagy számításigény

Inkrementális képszintézis

- primitívenként számol
 - ami nem primitív, azt azzal kell közelíteni
 - külön algoritmus kell ezekhez
 - külön meg kell oldani
 - a koherencia miatt kisebb számításigény

Áttekintés

Tartalom

1 Tartalom

- #### • Motiváció

2 Grafikus szerelőszalag

- Áttekintés
 - Modellezési transzformáció
 - Nézeti transzformáció
 - Perspektív transzformáció
 - Vágás
 - Raszterizáció
 - Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer

3 Lokális illumináció

Grafikus szerelőszalag

- A színtérünkről készített kép elkészítésének műveletsorozatát nevezik grafikus szerelőszalagnak (angolul *graphics pipeline*)
 - A valósidejű alkalmazásoknak lényegében a színtérünk leírását kell csak átadnia, a képszintézis lépései a grafikus szerelőszalag végzi
 - A szerelőszalagban több koordináta-rendszer váltás is történik - minden feladatot a hozzá legjobban illeszkedő rendszerben próbálunk elvégezni

Grafikus szerelőszalag

- Lépései főbb műveletek szerint:
 - Modellezési transzformáció
 - Nézeti transzformáció
 - Perspektív transzformáció
 - Vágás
 - Homogén osztás
 - Raszterizáció
 - Megjelenítés
 - A grafikus szerelőszalag eredménye egy kép (egy kétdimenziós pixeltömb, aminek minden elemében egy színérték található)

Transzformációk

- A szerelőszalag transzformációinak feladata: a modelltérben adott objektumot "eljuttatni" a képernyő-térbe
 - Lépései:

modell k.r.

→ világ k.r.

→ kamera k.r.

→ normalizált eszköz k.r.

→ képernyő k.r.

Szerelőszalag bemeneti adatai

- Ábrázolandó tárgyak *geometriai modellje*
 - *Virtuális kamera adatai* (nézőpont és látógúla)
 - A képkeret megadása (az a pixeltömb, amire a színtérünk síkvetületét leképezzük)
 - A színtérben található fényforrásokhoz és anyagokhoz tartozó *megvilágítási adatok*

Pipeline

- minden egyes primitív végigmegy az összes lépésen
 - az egyes lépések az eredményüket a következőnek továbbítják
 - többféleképpen megadható és csoportosítható - mi csak egy példát írtunk fel (pl. hol "színezünk")

Koordináta-rendszer

- Modell KR:
Az objektumok saját koordináta-rendszerükben adottak.
 - Világ KR:
Az objektumok egymáshoz viszonyított helyzete itt adott, ahogy a kamera/szem pozicó és az absztrakt fényforrások pozíciója is.
 - Kamera KR:
A koordináták a kamera pozíciója és orientációjához relatíven adottak.

Áttekintés

Koordináta-rendszer

- Normalizált eszköz KR:
A hardverre jellemző, $[-1, 1] \times [-1, 1] \times [0, 1]$ kiterjedésű KR.
 - Képernyő KR:
A megjelenítendő képnek (képernyő/ablak) megfelelő KR
(balkezes, bal-felső "sarok" az origó).

Modellezési transzformáció

Tartalom

- 1 Tartalom
 - Motiváció
 - 2 Grafikus szerelőszalag
 - Áttekintés
 - Modellezési transzformáció
 - Nézeti transzformáció
 - Perspektív transzformáció
 - Vágás
 - Raszterizáció
 - Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer
 - 3 Lokális illumináció

Modellezési transzformáció

- A saját (modell) koordináta-rendszerben adott modelleket a világ-koordináta rendszerben helyezi el
 - Tipikusan minden modellre különböző (lehet a színtérünk két eleme csak a világtrafóban különbözik!)
 - Jellemzően affin transzformációk
 - Gyakorlatban: ez a *Model* (vagy *World*) mátrix a kódjainkban

Nézeti transzformáció

Tartalom

1 Tartalom

- #### • Motiváció

2 Grafikus szerelőszalag

- Áttekintés
 - Modellezési transzformáció
 - **Nézeti transzformáció**
 - Perspektív transzformáció
 - Vágás
 - Raszterizáció
 - Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer

3 Lokális illumináció

Nézeti transzformáció

- A világ koordináta-rendszert a kamerához rögzített koordináta-rendszerbe viszi át.
 - A transzformáció a kamera tulajdonságaiból adódik.
 - Gyakorlatban: ez a *View* mátrix.

Kamera transzformáció

- Tulajdonságok, mint sugárkövetés esetén:
eye, center, up
 - Ebből kapjuk a nézeti koordináta-rendszer tengelyeit:

$$w = \frac{\text{eye} - \text{center}}{|\text{eye} - \text{center}|}$$

$$\mathbf{u} = \frac{\mathbf{up} \times \mathbf{w}}{|\mathbf{up} \times \mathbf{w}|}$$

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

Kamera transzformációs mátrix

- Mátrixot kapjuk: áttérés az **–eye** origójú, **u, v, w** koordinátarendszerbe:

$$T_{View} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspektív transzformáció

Tartalom

- 1 Tartalom
 - Motiváció
 - 2 Grafikus szerelőszalag
 - Áttekintés
 - Modellezési transzformáció
 - Nézeti transzformáció
 - Perspektív transzformáció
 - Vágás
 - Raszterizáció
 - Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer
 - 3 Lokális illumináció

Párhuzamos vetítés

- A mátrix ami megadja egyszerű, például az XY síkra való vetítés

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

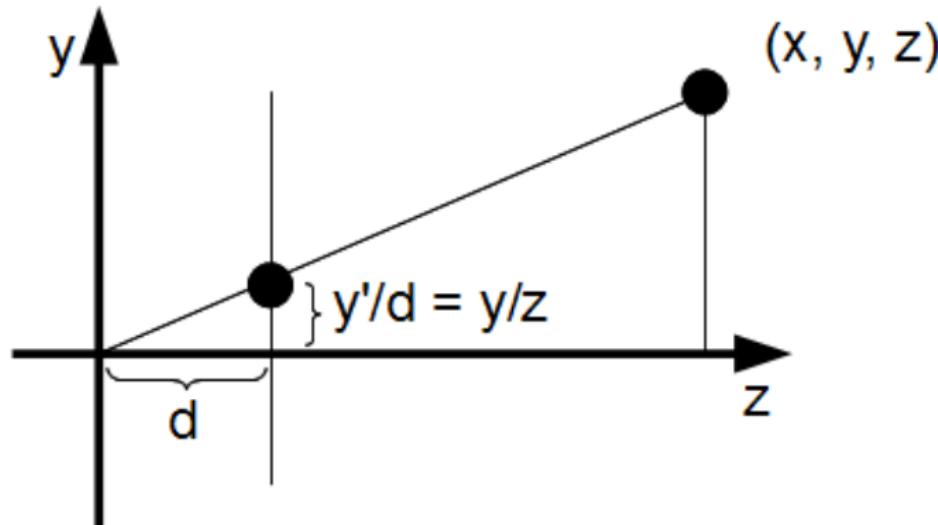
Perspektív transzformáció

- Emlékeztető: 3. EA
 - A nézeti csonkagúla által határolt térrészt normalizált eszköz KR-be viszi át
 - Ami benne volt a csonkagúlában, az lesz benne a $[-1, 1] \times [-1, 1] \times [0, 1]$ (vagy $[-1, 1] \times [-1, 1] \times [-1, 1]$) tartományban
 - A transzformáció a kamerán átmenő *vetítő sugarakból* párhuzamosokat csinál
 - A transzformáció a kamerapozíciót a végtelenbe tolja
 - Gyakorlatban: ez a *Projection* mátrix

Perspektív transzformáció

- Emlékeztető: tulajdonságok
 - függőleges és vízszintes nyílásszög (*fov_x, fovy*) vagy az alap oldalainak az aránya és a függőleges nyílásszög (*fovy, aspect*),
 - a közel vágósík távolsága (*near*),
 - a távoli vágósík távolsága (*far*)

Középpontos vetítés



Középpontos vetítés

- Vagyis:

$$x' = \frac{x}{z}d$$

$$y' = \frac{y}{z}d$$

$$z' = \frac{z}{z}d = d$$

Középpontos vetítés

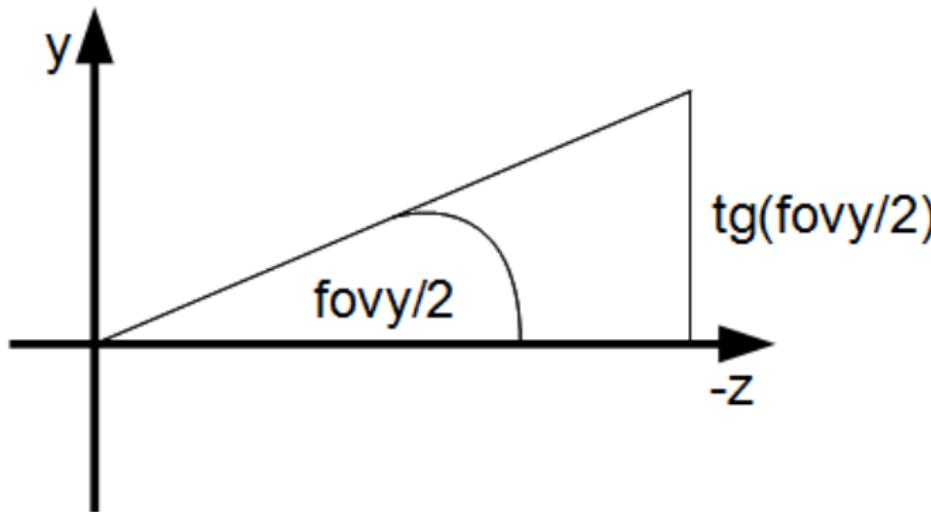
- Az origó, mint vetítési középpont és egy, attól a Z tengely mentén d egységre található, XY síkkal párhuzamos vetítő síkra való vetítés mátrixa:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}$$

- Homogén osztás után ($\frac{z}{d}$ -vel) a fentit kapjuk

Normalizált látogúla

- Figyeljünk: a szerelőszalagunk ezen pontján a kamera $-Z$ felé néz és az origóba van
 - A fenti térből térjünk át egy "normalizáltabb" gúlába - aminek nyílásszöge x és y mentén is 90 fokos!



Normalizált látogúla

- Mátrix alakban:

$$\begin{bmatrix} 1/\tan \frac{fov_x}{2} & 0 & 0 & 0 \\ 0 & 1/\tan \frac{fov_y}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Normalizált látogúla

- Ezután már csak a közelí és a távoli vágosík z koordinátáit kell a normalizálásnak megfelelően átképezni ($-1, 1$ vagy $0, 1$ -re):

$$T_{Projection} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{far}{far - near} & \frac{near * far}{near - far} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Vágás

Tartalom

1 Tartalom

- ## • Motiváció

2 Grafikus szerelőszalag

- Áttekintés
 - Modellezési transzformáció
 - Nézeti transzformáció
 - Perspektív transzformáció
 - **Vágás**
 - Raszterizáció
 - Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer

3 Lokális illumináció

Vágás

- Cél: ne dolgozzunk feleslegesen azokkal az elemekkel, amikből nem lesz pixel (nem jelennek majd meg).
 - Megoldás (kísérlet):
 - ➊ Végezzük el a homogén osztást!
 - ➋ Vágunk le, ami kilóg a $[-1, 1] \times [-1, 1] \times [0, 1]$ -ből!
 - Probléma: vegyük egy olyan szakaszt, ami átlóg a kamera mögé!
 - Ez a szakasz átmegy egy ideális ponton \Rightarrow a szakasz képe nem egyezik meg a transzformált pontokat összekötő szakasszal!
 - Ez az *átfordulási probléma*.

Vágás homogén koordinátákban

- Megoldás (valóban): Vágás homogén koordinátákban
 - Legyen: $[x_h, y_h, z_h, h] = \mathbf{M}_{\text{Proj}}[x_c, y_c, z_c, 1]$
 - Cél: $(x, y, z) := (x_h/h, y_h/h, z_h/h) \in [-1, 1] \times [-1, 1] \times [0, 1]$, azaz

Legyen $h > 0$, és

Ebből kapjuk:

$$-1 < x < 1$$

$$-h < x_h < h$$

$$-1 < y < 1$$

$$-h < y_h < h$$

$$0 < z < 1$$

$$0 < z_h < h$$

Raszterizáció

Tartalom

- 1 Tartalom
 - Motiváció
 - 2 Grafikus szerelőszalag
 - Áttekintés
 - Modellezési transzformáció
 - Nézeti transzformáció
 - Perspektív transzformáció
 - Vágás
 - Raszterizáció
 - Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer
 - 3 Lokális illumináció

Raszterizáció

Raszterizáció

- Ne feledjük: eddig minden primitív, amiről beszéltünk folytonos objektum volt
 - Azonban nekünk egy diszkrét térben, a képernyő képpontjain kell dolgoznunk
 - A primitívek folytonos teréből át kell térní ebbe a diszkrét térbe, ezt hívják *raszterizáció*nak

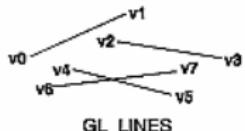
Raszterizáció

- Olyan geometriai primitíveket kell választanunk, amelyeket gyorsan tudunk raszterizálni
 - Mi lehet ilyen? Jó lenne pl. ha egyik pixelhez tartozó felületi pontjának koordinátái alapján könnyen számítható lenne a szomszédos pixelekhez tartozó pontok koordinátái, illetve ha síkbeli is lenne...
 - A háromszög ilyen!
 - minden egyéb felületet ilyen primitívekkel (lényegében: síklapokkal) közelítünk ← tesszeláció

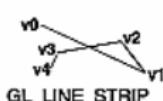
Raszterizáció

Raszterizáció

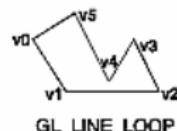
• v4
v0 • • v3
 v1 • • v2
GL POINTS



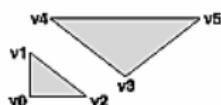
GL LINES



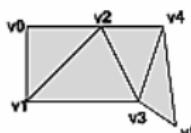
GL LINE STRIP



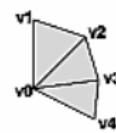
GL LINE LOOP



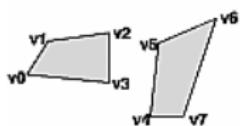
GL_TRIANGLES



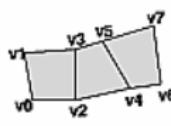
GL_TRIANGLE_STRIP



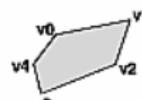
GL_TRIANGLE_FAN



GL_QUADS



GL QUAD STRIP



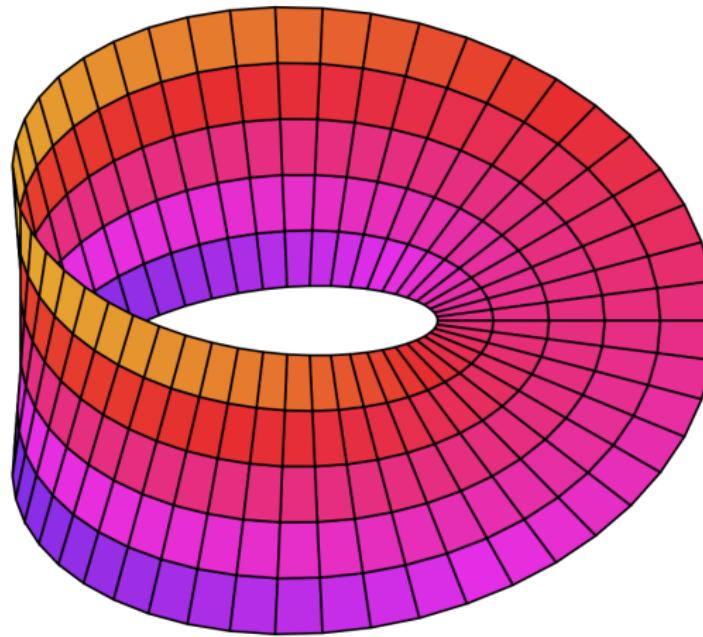
GL_POLYGON

Tesszeláció

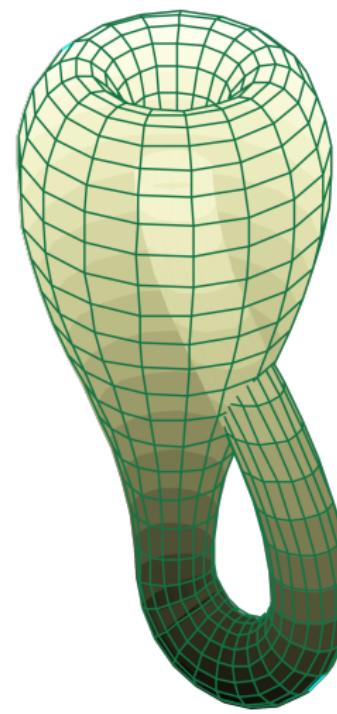
- Vigyázzunk: "szép" (teljes oldalakban illeszkedő), 6-reguláris háromszög vagy "szép", 4-reguláris négyszöghálóval nem lehet bármit lefedni degenerált esetek nélkül!
- A fenti reguláris topológiákkal a végtelen síklap, vagy a végtelen hengerpalást, vagy pedig a tórusz topológiának megfelelő felületek írhatóak le.

Raszterizáció

Tesszeláció



Tesszeláció



Megjelenítés

Tartalom

1 Tartalom

- Motiváció

2 Grafikus szerelőszalag

- Áttekintés
- Modellezési transzformáció
- Nézeti transzformáció
- Perspektív transzformáció
- Vágás
- Raszterizáció
- Megjelenítés
 - Triviális hátlapeldobás
 - Festő algoritmus
 - Z-buffer

3 Lokális illumináció

Takarási feladat

- Feladat: eldönteni, hogy a kép egyes részein milyen felületdarab látszik.
- Objektum tér algoritmusok:
 - Logikai egységenként dolgozunk, nem függ a képernyő felbontásától.
 - Rossz hír: nem fog menni.
- Képtér algoritmusok:
 - Pixelenként döntjük el, hogy mi látszik.
 - Ilyen a sugárkövetés is.

Triviális hátlapeldobás, *Back-face culling*

- Feltételezés: Az objektumaink "zártak", azaz ha nem vagyunk benne az objektumban, akkor seholnan sem láthatjuk a felületét belülről.
 - Körüljárási irány: rögzítsük, hogy a poligonok csúcsait milyen sorrendben *kell* megadni:
 - óramutató járásával megegyező (*clockwise, CW*)
 - óramutató járásával ellentétes (*counter clockwise, CCW*)
 - Ha a transzformációk után a csúcsok sorrendje nem egyezik meg a megadással, akkor a lapot *hátulról* látjuk \Rightarrow nem kell kirajzolni, *eldobható*.

Festő algoritmus

- Rajzoljuk ki hátulról előre haladva a poligonokat!
- Ami közelebb van, azt később a rajzoljuk \Rightarrow ami takarva van, takarva lesz.
- Probléma: hogyan rakjuk sorrendbe a poligonokat?
- Már háromszögeknél is van olyan eset, amikor nem lehet sorrendet megadni.

Festő algoritmus

Z-buffer algoritmus

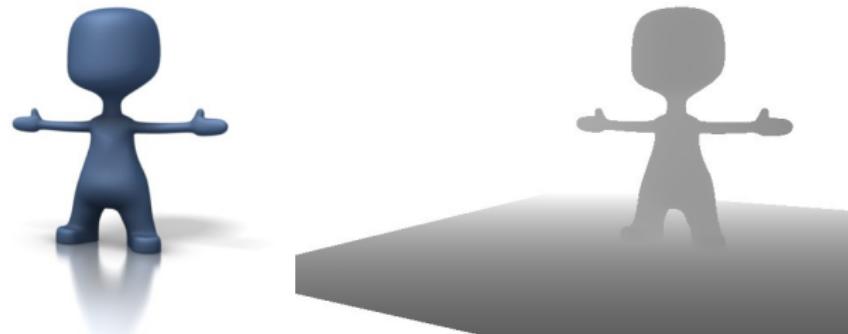
- Képtérbeli algoritmus
- minden pixelre nyílvántartjuk, hogy ahhoz milyen mélységek tartozott.
- Ha megint újra erre a pixelre rajzolnánk (*Z-test*):
 - Ha az új Z érték mélyebben van, akkor ez e pont takarva van
⇒ nem rajzolunk
 - Ha a régi Z érték mélyebben van, akkor az új pont kitakarja azt ⇒ rajzolunk, eltároljuk az új Z értéket.

Z-buffer

- Z-buffer vagy *depth buffer*: külön memóriaterület.
- Képernyő/ablak méretével megegyező méretű tömb.
- Pontosság: a közeli és távoli vágósík közti távolságtól függ.
- minden pixelhez tartozik egy érték a bufferból.
- Ezzel kell összehasonlítani, és ide kell írni, ha a pixel átment a *Z-teszten*.
- Gyakorlatban:
 - 16-32 bites elemek
 - Hardveres gyorsítás
 - Pl: közeli vágósík: t , távoli: $1000t$, akkor a Z-buffer 98%-a a tartomány első 2%-át írja le.

Megjelenítés

Z-buffer



by macouno, macouno.com

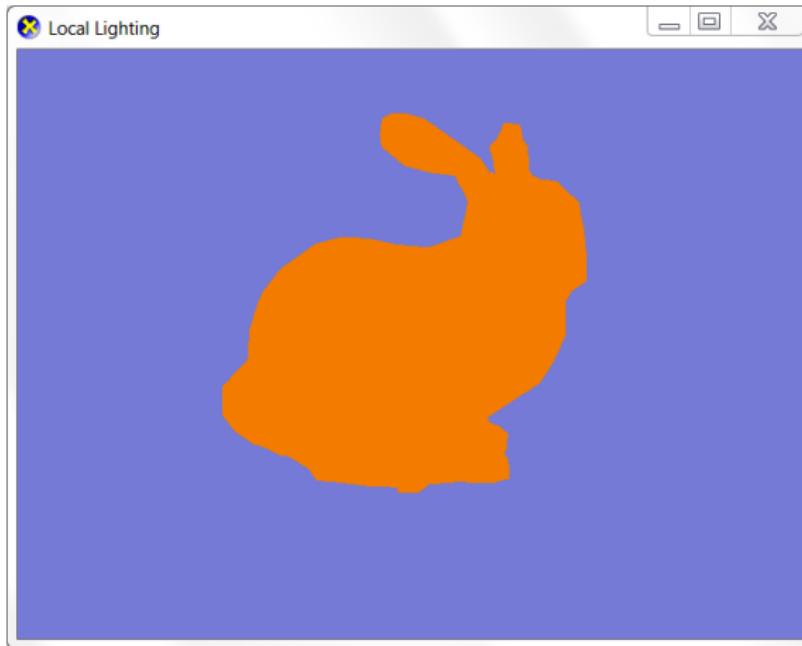
Lokális illumináció

- Ha már megvannak a primitívjeink pixelekre való leképezései, valahogyan számítsunk színeket

Saját színnel árnyalás

- minden objektumhoz/primitívhez egy színt rendelenünk, és kirajzoláskor el lesz a pixelek értéke.
 - Leggyorsabb: az ilumináció gyakorlatilag egyetlen értékadás.
 - Borzasztó: se nem valósághű, se nem szép.

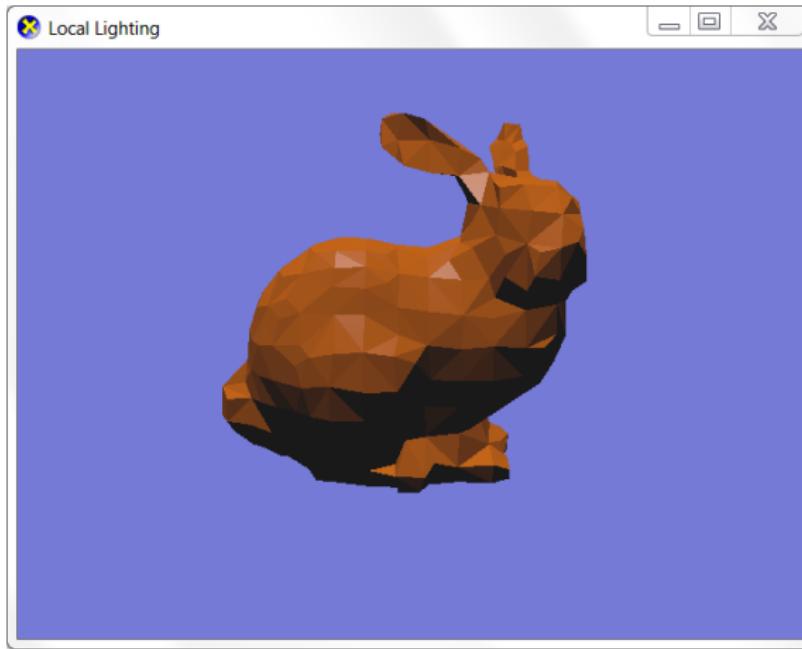
Saját színnel árnyalás



Konstans árnyalás, *Flat shading*

- A megvilágítást poligononként egyszer számítjuk ki, a szín homogén a lapon belül.
 - Gyors: a műveletek száma a poligonok számától függ, a pixelek számától független.
 - Van hogy használható: íves részeket nem tartalmazó, diffúz, egyszínű objektumokra

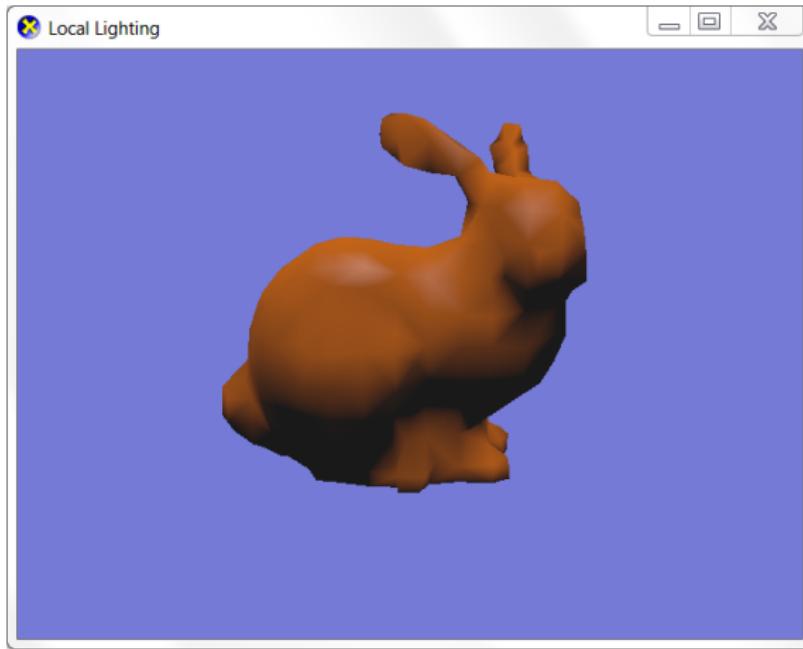
Konstans árnyalás



Gouraud árnyalás

- A megvilágítást csúcspontonként számítjuk ki, a lapon lineáris interpolációval számítjuk a színeket.
 - Lassabb: N db megvilágítás számítás + minden pixelre interpoláció.
 - Szebb: az árnyalás minősége nagyban függ a poligonok számától. Nagy lapokon nem tud megjelenni a csillanás.

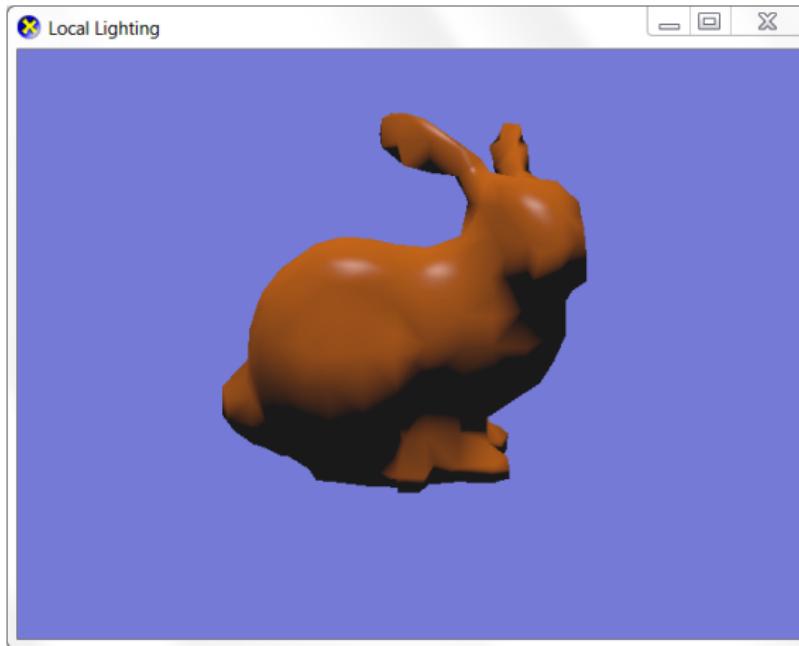
Gouraud árnyalás



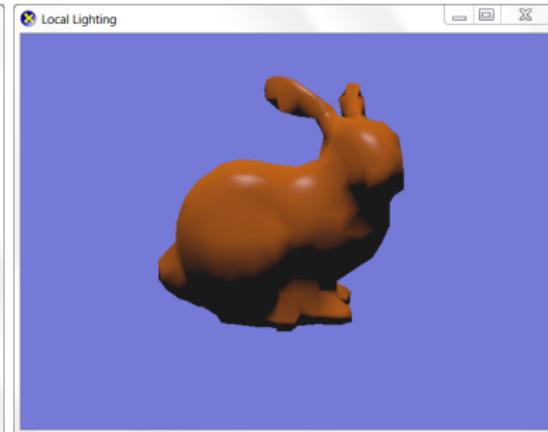
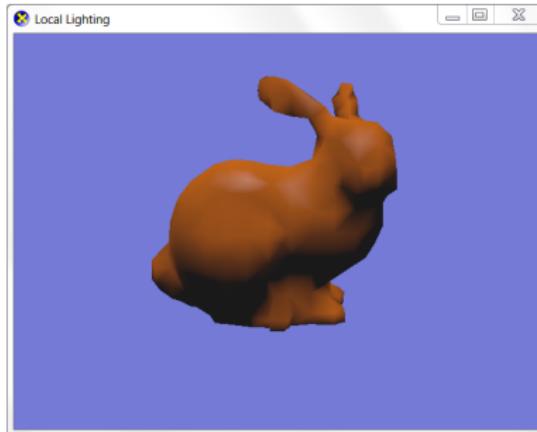
Phong árnyalás

- Csak a normálvektorokat interpoláljuk, a megvilágítást minden pixelre kiszámítjuk.
 - Leglassabb: *pixelek száma* db megvilágítás számítás.
 - Legszebb: az árnyalás minősége nem függ a poligonok számától. Csillanás akár poligon közepén is meg tud jelenni.

Phong árnyalás



Gouraud vs Phong árnyalás



Számítógépes Grafika

Hajder Levente

hajder@inf.elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2017/2018. II. félév

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
- Pont és szakaszvágás
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció

Inkrementális képszintézis

- Inkrementális elv
- A transzformációk szemszögéből végignéztük a grafikus szerelőszalagot
- Lényegében egy pont útját követtük végig, a transzformációkon át a képernyőig
- Most az inkrementális képszintézis szerelőszalagját vizsgáljuk tovább

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás

3 Raszterizálás

- Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció





Vágás

- A vágás során a nézeti csonkagúlán kívül geometriai elemeket szűrjük ki
- A nézeti csonkagúla határozza meg a színterünknek azt a részét, amely majd leképeződik a képernyőre (ld. múlt óra)
- Miért érdemes egyáltalán vagni?
 - Degenerált esetek kiszűrése (ld. pl. múlt óra középpontos vetítés)
 - Ne számoljunk feleslegesen (amit úgyse látunk, ne számoljuk sokat)

Vágás az ablakra

- Pont vágás: $(x, y) \in \mathbb{R}^2$. Akkor tartjuk meg a pontot, ha $(x, y) \in [x_{min}, x_{max}] \times [y_{min}, y_{max}]$.
Megoldás: a határoló egyenesekkel vágás (ha tengelypárhuzamosak: csak összehasonlítás)
 - Szakasz vágás: a szakasznak csak azokat a pontjai akarjuk megtartani, amik benne vannak $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ -ban.
Megoldás: az ablak négy élével, mint négy félsíkkal vágjuk a szakaszt.
 - Poligon vágás: a poligonból egy új poligont akarunk csinálni, ami nem lóg ki az ablakból.
Megoldás: A poligon minden oldalát (mint szakaszt) vágjuk.



Pont és szakaszvágás

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
- Pont és szakaszvágás
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció

Pont és szakaszvágás

Pontok vágása

- Itt most pontokat vágunk egyenesre, síkra
- Azaz, azt akarjuk meghatározni, hogy a bemeneti pont az egyenes vagy sík normálisával megegyező irányban fekszik-e az egyenes/sík pontjaihoz képest ("előtte" van-e).
- Ami mögötte van, nem kell nekünk → ha rajta fekszik, azt is tartsuk még meg
- Síkra 3D-ben vágunk → a nézeti csonkgúla 6 sík "előtti" rész (6 féltér metszete)
- Egyenesre 2D-ben → a képernyőn a monitorra kerülő rész 4 egyenes "előtti" rész (4 félsík metszete)



Pont és szakaszvágás

Az egyenes normálvektoros egyenlete a síkban

- Az egyenes megadható egy $P(p_x, p_y)$ pontjával és egy, az egyenes irányára merőleges $\mathbf{n} = [n_x, n_y]^T \neq \mathbf{0}$ normálvektorral:
- Az egyenes pontjai azon $Q(x, y)$ pontok, amelyek kielégítik a

$$\langle X - P, \mathbf{n} \rangle = 0$$
$$(x - p_x)n_x + (y - p_y)n_y = 0$$

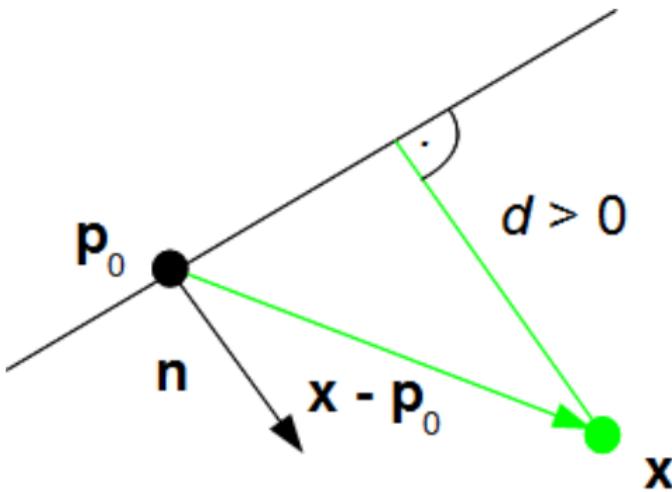
egyenletet.

- Az $\langle X' - P, \mathbf{n} \rangle < 0$ és $\langle X' - P, \mathbf{n} \rangle > 0$ az egyenesünk által meghatározott két félsík egyenlete.

Pont vágása pont-normálvektoros egyenesre

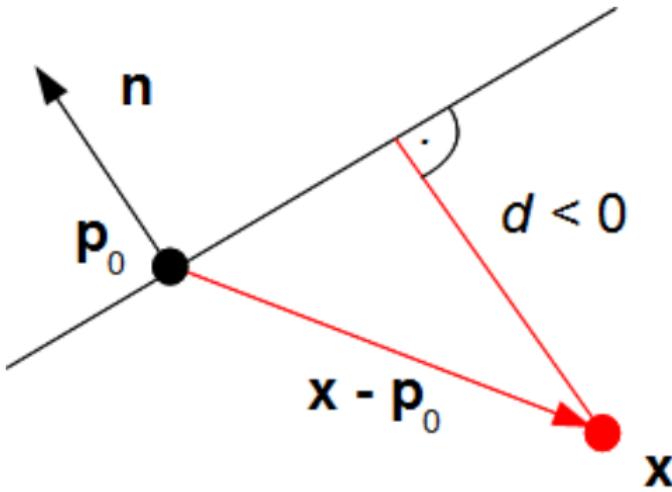
- Feladat: adott \mathbf{p} pont és egy e egyenes (\mathbf{p}_0 pontjával és \mathbf{n} normálvektorával, $|\mathbf{n}| = 1$) a síkban. Vágjuk a pontot az e egyenesre!
 - Megtartjuk, ha: $\langle \mathbf{p} - \mathbf{p}_0, \mathbf{n} \rangle \geq 0$
 - Ilyenkor a skaláris szorzat eredménye az egyenes egy pontjából az adott pontba mutató vektor előjeles vetülete a normálisra $|\mathbf{n}| = 1 \rightarrow$ előjeles távolság az egyenestől

Pont-egyenes távolsága



Pont és szakaszvágás

Pont-egyenes távolsága



Pont vágása vonalkoordinátás egyenesre

- Használjuk ki, hogy a vágásnál már homogén koordinátákban dolgozik a rendszer!
 - Feladat: adott $\hat{\mathbf{p}}$ pont homogén koordinátás alakja és egy e egyenes e vonalkoordinátáival, úgy, hogy $e_1^2 + e_2^2 + e_3^2 = 1$
 - Megtartjuk, ha: $e \cdot \hat{\mathbf{p}} \geq 0$

A sík normálvektoros egyenlete

- A sík megadható egy $P(p_x, p_y, p_z)$ pontjával és a síkra merőleges $\mathbf{n} = [n_x, n_y, n_z]^T$ normálvektorával:

$$\langle X - P, \mathbf{n} \rangle = 0$$

- Félterek: $\langle X - P, \mathbf{n} \rangle < 0$, $\langle X - P, \mathbf{n} \rangle > 0$



Pont és szakaszvágás

Pont vágása síkra

- Pont-normálisal adott sík esetén megtartjuk a pontot, ha:

$$\langle \mathbf{p} - \mathbf{p}_0, \mathbf{n} \rangle \geq 0$$

- Sík-koordinátás (\mathbf{s}) megadás esetén (ha $s_1^2 + s_2^2 + s_3^2 = 1$) megtartjuk a pontot, ha:

$$\mathbf{s} \cdot \hat{\mathbf{p}} \geq 0$$



Szakaszok vágása

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
- Pont és szakaszvágás
- Szakaszok vágása**
- Szakaszvágás
- Poligonvágás

3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció



Szakaszok vágása

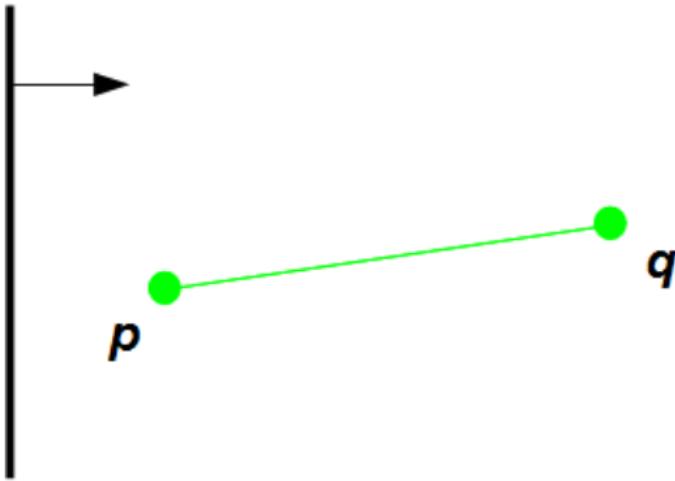
Szakasz vágása félsíkra

- A szakasz **p** és **q** végpontjait vágjuk az e egyenes és normálisa által meghatározott félsíkra!
 - Az előbb látott módon végezhetjük a vágást!
 - Az eredmény viszont most bonyolultabb egy kicsit

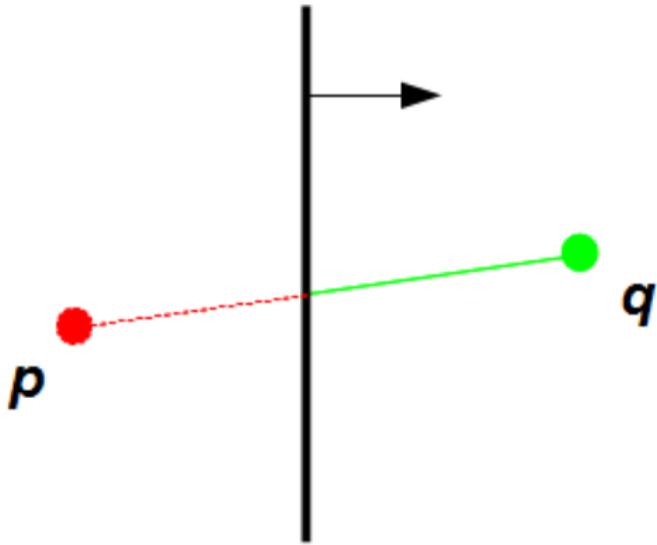
Szakasz vágása félsíkra



Szakasz vágása félsíkra



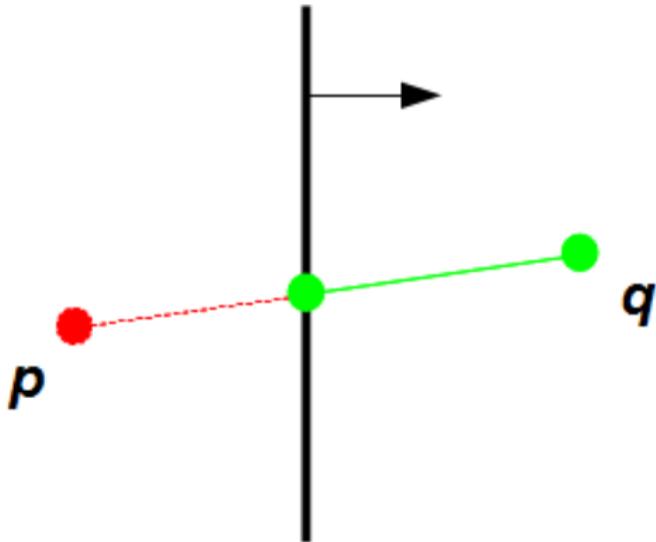
Szakasz vágása félsíkra



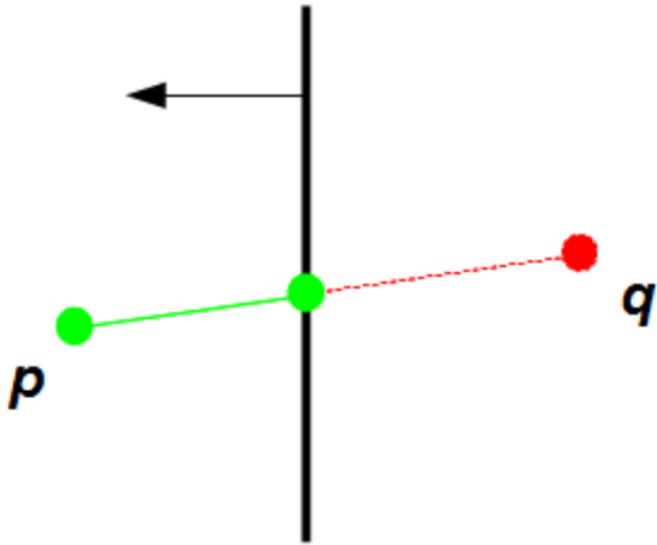


Szakasz vágása

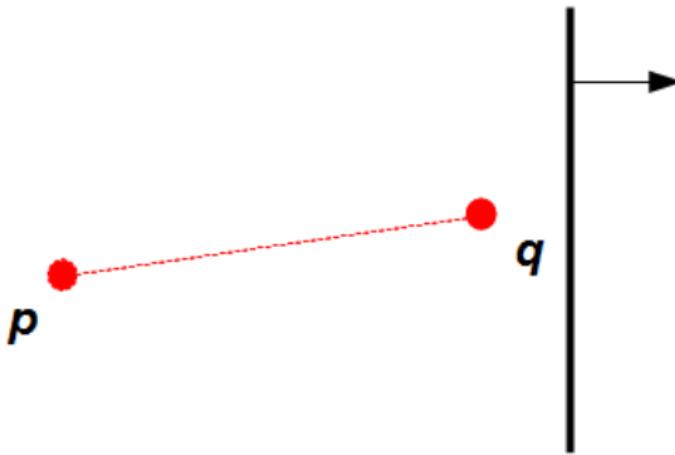
Szakasz vágása félsíkra - új "kezdőpont"!



Szakasz vágása félsíkra



Szakasz vágása félsíkra





Szakaszvágás

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
- Pont és szakaszvágás
- Szakaszok vágása
- Szakaszvágás**
- Poligonvágás

3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció



Szakaszvágás

Szakasz vágása félsíkra

- Csak azt a speciális esetet nézzük, amikor a vágó egyenes párhuzamos valamelyik tengellyel.
- $(x_1, y_1) - (x_2, y_2)$ szakasz egyenlete:

$$x(t) = x_1 + t(x_2 - x_1)$$

$$y(t) = y_1 + t(y_2 - y_1)$$

- Vágó egyenes: pl. $x = x_{min}$
- Megoldás: $x_{min} = x_1 + t(x_2 - x_1) \Rightarrow t = \frac{x_{min} - x_1}{x_2 - x_1}$
- Metszéspont: $x = x_{min}, y = y_1 + \frac{x_{min} - x_1}{x_2 - x_1}(y_2 - y_1)$

Szakasz vágása - nehézségek

- Eredmény kezelése: $t \leq 0$ vagy $t \geq 1 \rightarrow$ nincs is igazi vágás
- Kivételek: tengelyekkel párhuzamos szakaszok vágása
- Képernyőre vágásnál is rengeteg eset: négy félsík, két végpont, minden kettő eshet bárhova
- Feladat: Kell-e vágni? Melyik félsík(ok)ra kell vágni?
Triviálisan bent/kint van-e a szakasz?
- Megoldás: *Cohen-Sutherland* vágás

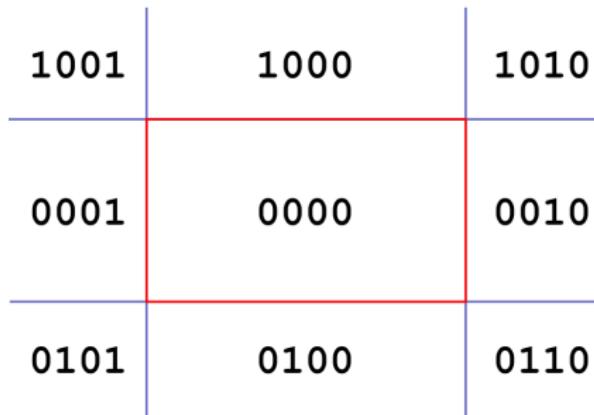
Cohen-Sutherland vágás

- A képernyő széleit reprezentáló egyenesekkel osszuk kilenc részre a síkot!
- Mindegyik síkrészhez rendeljünk egy 4 bitből álló bit-kódot: TBRL
- Számítsuk ki ezt a kódot a szakasz végpontjaira!
 - $T = 1$, ha a pont az képernyő fölött van, különben 0
 - $B = 1$, ha a pont az képernyő alatt van, különben 0
 - $R = 1$, ha a pont az képernyőtől jobbra van, különben 0
 - $L = 1$, ha a pont az képernyőtől balra van, különben 0



Cohen-Sutherland vágás

code = Top, Bottom, Right, Left





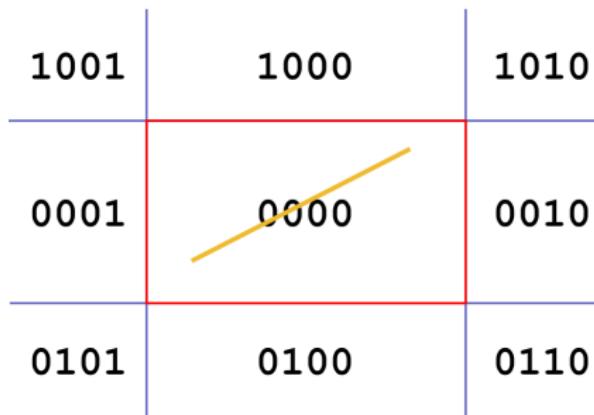
Cohen-Sutherland váágás

- $\text{code} = (y > y_{max}, y < y_{min}, x > x_{max}, x < x_{min})$
- Vizsgáljuk a két végpont bit-kódjait, code_a -t és code_b -t!
 - Ha $\text{code}_a \text{ OR } \text{code}_b == 0$: a szakasz egésze (mindkét végpontja) az ablakban van \Rightarrow megtartjuk.
 - Ha $\text{code}_a \text{ AND } \text{code}_b != 0$: a szakasz az ablak valamelyik oldalán (fölül, alul, jobbra, balra), kívül van \Rightarrow eldobjuk.
 - Különben vagni kell: az 1-es bitek adják meg, hogy melyik félsíkra, utána újra számoljuk a bit-kódokat, és az új szakasszal újrakezdjük az algoritmust.



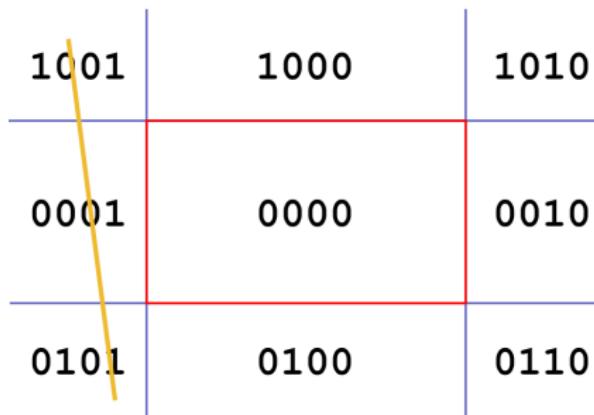
Cohen-Sutherland vágás

$\text{code}_a \text{ OR } \text{code}_b == 0:$



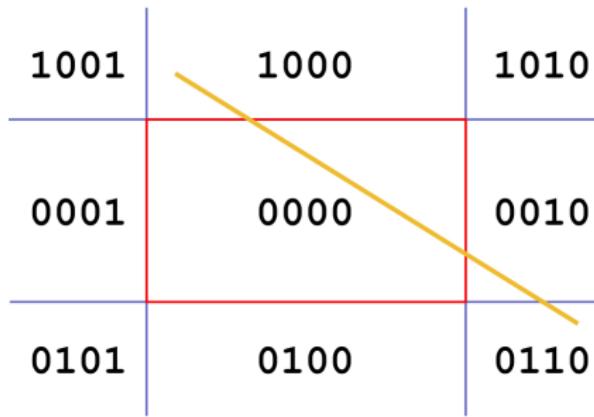
Cohen-Sutherland vágás

$\text{code}_a \text{ AND } \text{code}_b \neq 0:$



Cohen-Sutherland vágás

Különben:





Poligonvágás

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
- Pont és szakaszvágás
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás**

3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció

Poligonvágás

Feladat

Adott egy vágandó és egy vágó poligon. Keressük azt a poligont, amit a vágandóból kapunk, ha csak a vágó poligonba tartozó részeit vesszük. Röviden: keressük a kettő közös részét.



Sutherland-Hodgman poligonvágás

- Legyen p tömb a bemeneti-, q a kimeneti-poligon csúcsainak tömbje!
- Legyen n a csúcsok száma, és $p[0] = p[n]!$
- Vágás egyenesre:
 - Ha $p[i]$ bent van az alablakban, és $p[i + 1]$ is:
 \Rightarrow adjuk hozzá $p[i]$ -t a q -hoz.
 - Ha $p[i]$ bent van az alablakban, de $p[i + 1]$ nincs:
 \Rightarrow adjuk hozzá $p[i]$ -t a q -hoz, számítsuk ki $p[i], p[i + 1]$ szakasz metszéspontját a vágó egyenessel, majd ezt is adjuk hozzá q -hoz.
 - Ha $p[i]$ nincs bent az alablakban, de $p[i + 1]$ benne van:
 \Rightarrow számítsuk ki $p[i], p[i + 1]$ szakasz metszéspontját a vágó egyenessel, és ezt adjuk hozzá q -hoz.
 - Ha $p[i]$ nincs bent az alablakban, és $p[i + 1]$ sincs:
 \Rightarrow SKIP

Poligonvágás

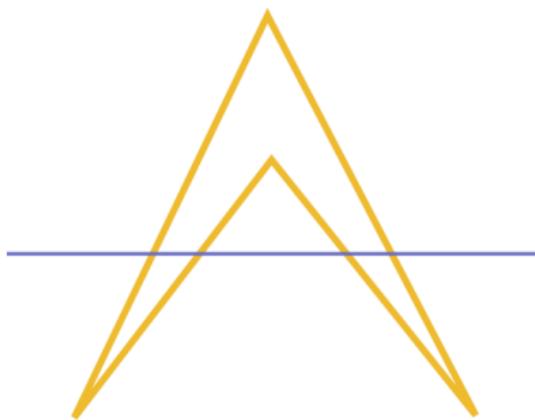
Sutherland-Hodgman poligonvágás - Pszeudó-kód

```
PolygonClip(in p[n], out q[m], in line) {
    m = 0;
    for( i=0; i < n; i++) {
        if (IsInside(p[i])) {
            q[m++] = p[i];
            if (!IsInside(p[i+1]))
                q[m++] = Intersect(p[i], p[i+1], line);
        } else {
            if (IsInside(p[i+1]))
                q[m++] = Intersect(p[i], p[i+1], line);
        }
    }
}
```



Sutherland-Hodgeman: konkáv poligonok

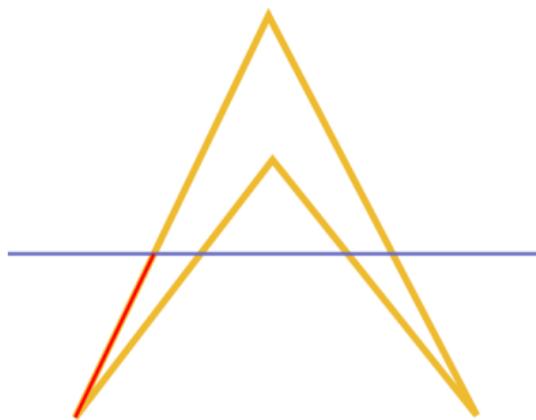
Konkáv poligonokra átfedő éleket hoz létre.





Sutherland-Hodgeman: konkáv poligonok

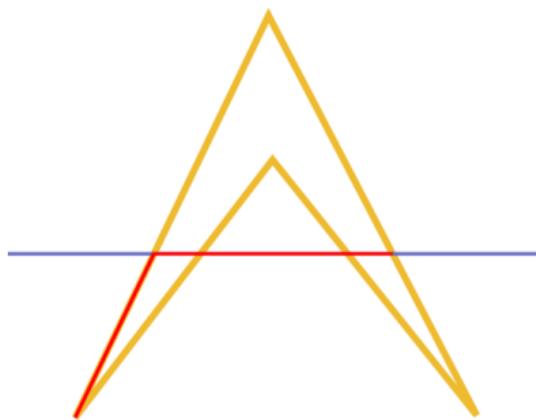
Konkáv poligonokra átfedő éleket hoz létre.





Sutherland-Hodgeman: konkáv poligonok

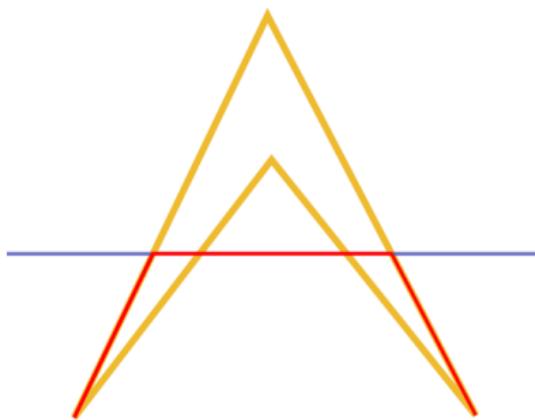
Konkáv poligonokra átfedő éleket hoz létre.





Sutherland-Hodgeman: konkáv poligonok

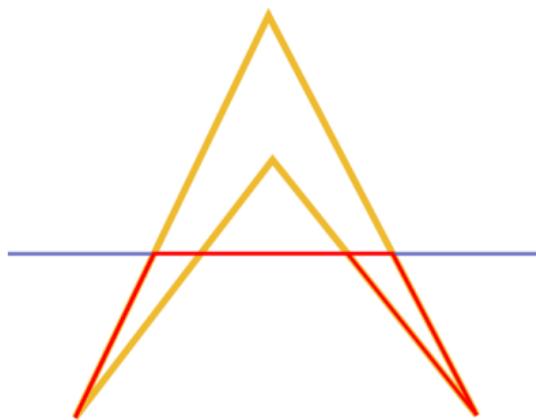
Konkáv poligonokra átfedő éleket hoz létre.





Sutherland-Hodgeman: konkáv poligonok

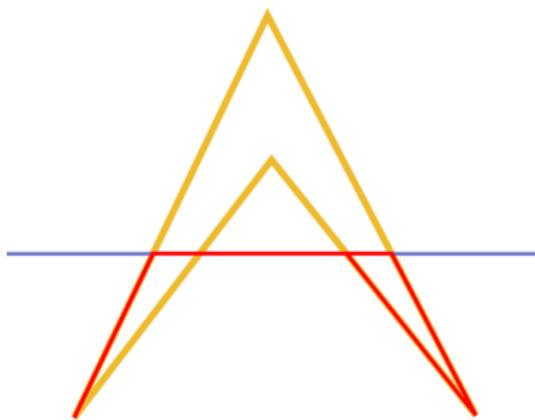
Konkáv poligonokra átfedő éleket hoz létre.





Sutherland-Hodgeman: konkáv poligonok

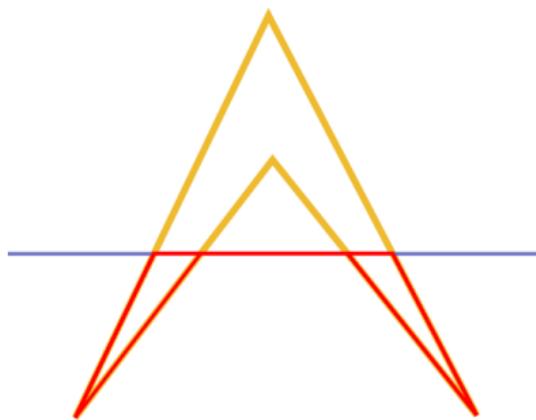
Konkáv poligonokra átfedő éleket hoz létre.





Sutherland-Hodgeman: konkáv poligonok

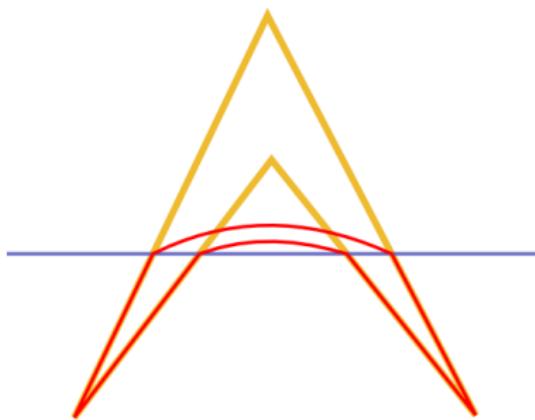
Konkáv poligonokra átfedő éleket hoz létre.





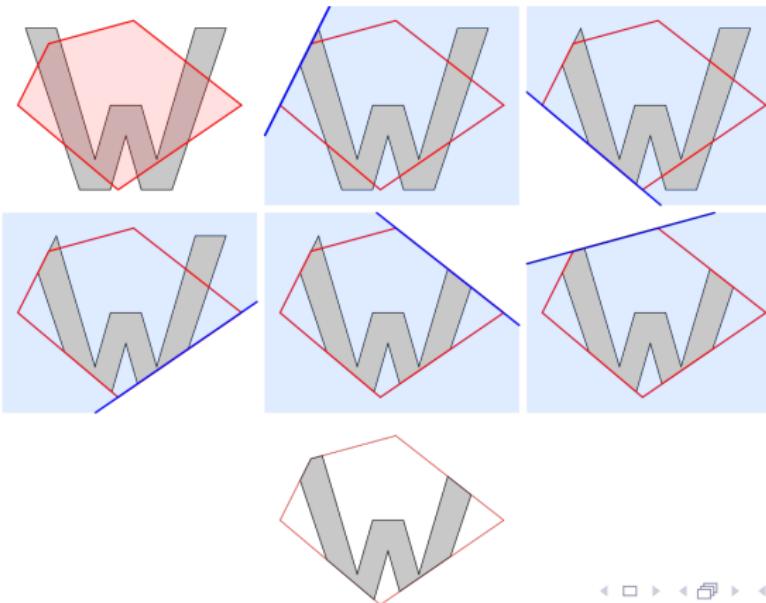
Sutherland-Hodgeman: konkáv poligonok

Konkáv poligonokra átfedő éleket hoz létre.



Sutherland-Hodgeman: poligonra vágása poligonnak

A vágópoligon minden élére vágunk, az előző vágás eredmény éllistáját felhasználva kiindulásként





Mikor vágunk?

- Projektív transzformáció előtt → a vágósíkok világtérbeli megadásával (mik az egyenletei?)
- NPKR-ben (projektív trafó után, homogén osztás előtt) → a homogén koordináták "ellenére" ez a legegyszerűbb!
- Transzformált 3D térben (homogén osztás után) → vetítési síkon átmenő objektumok...

Szakasz rászterizálása

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás

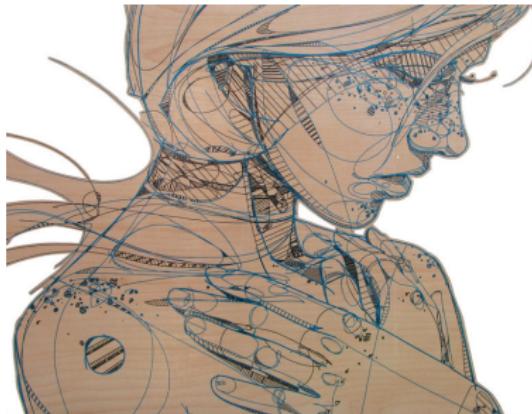
3 Raszterizálás

- Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

Szakasz rászterizálása

Szakasz rajzolás

- Egyik leggyakrabben előforduló primitív
 - Fontos, hogy szépen tudjuk rajzolni
 - Mégjobb, ha gyorsan is

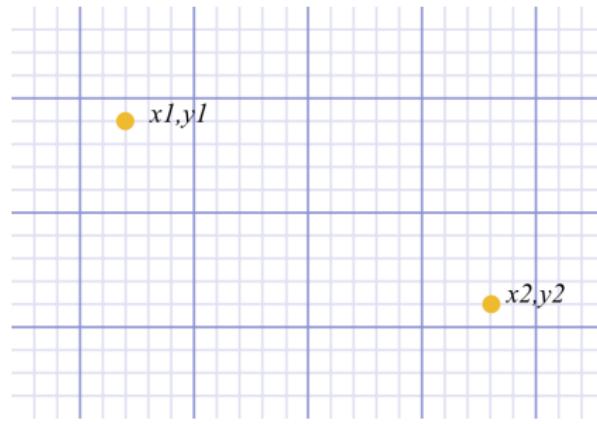


Jason Thielke, jasonthielke.com

Szakasz rászterizálása

Hogyan rajzolunk szakaszt?

- Adott a két végpont.

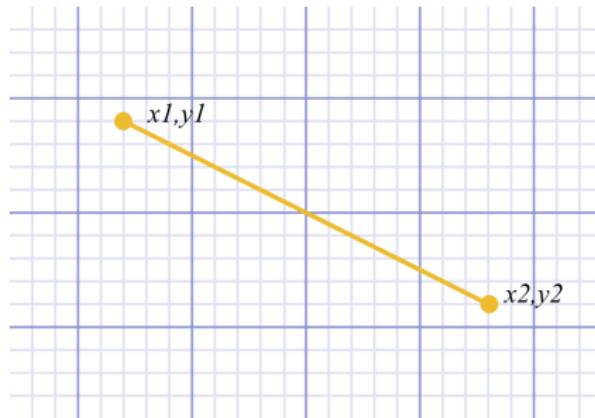




Szakasz riasztására

Hogyan rajzolunk szakaszt?

- Adott a két végpont.
- Hogyan tudjuk összekötni őket?

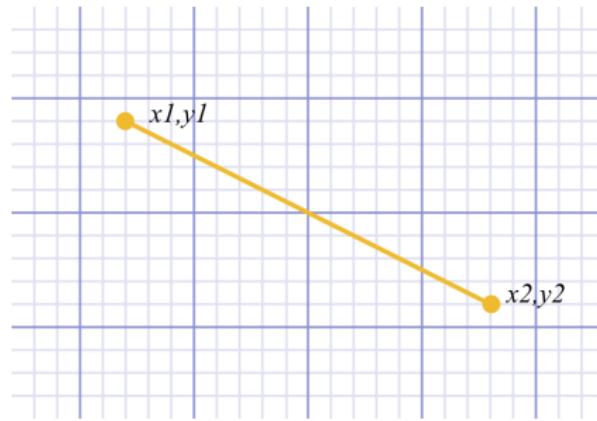




Szakasz raszterizálása

Hogyan rajzolunk szakaszt?

- Adott a két végpont.
- Hogyan tudjuk összekötni őket?
- Csak miniatűr téglalapjaink vannak (amiket pixelnek nevezünk).

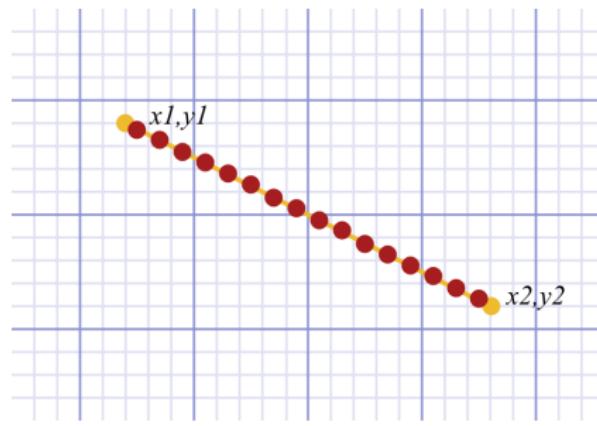




Szakasz raszterizálása

Hogyan rajzolunk szakaszt?

- Adott a két végpont.
- Hogyan tudjuk összekötni őket?
- Csak miniatűr téglalapjaink vannak (amiket pixelnek nevezünk).

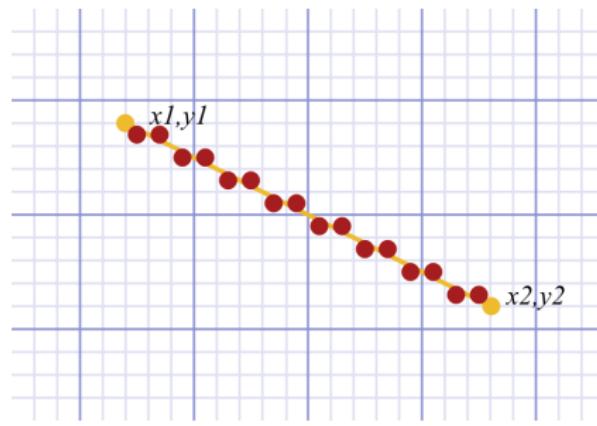




Szakasz raszterizálása

Hogyan rajzolunk szakaszt?

- Adott a két végpont.
- Hogyan tudjuk összekötni őket?
- Csak miniatűr téglalapjaink vannak (amiket pixelnek nevezünk).



Szakasz raszterizálása

Szakasz megadása (sokadszor)

- Végpontok: $(x_1, y_1), (x_2, y_2)$
 - Tfh. nem függőleges: $x_1 \neq x_2$.
 - Szakasz egyenlete:

$$y = mx + b, x \in [x_1, x_2]$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - mx_1$$

Szakasz rászterizálása

Naív algoritmus

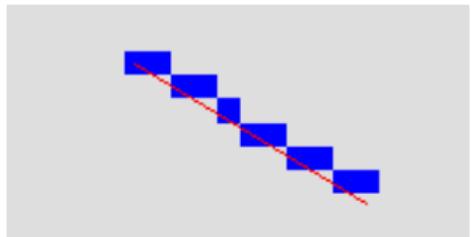
```
def line1(x1,y1,x2,y2, draw):  
    m = float(y2-y1)/(x2-x1)  
    x = x1  
    y = float(y1)  
    while x<=x2:  
        draw.point((x,y))  
        x += 1  
        y += m
```



Szakasz rászterizálása

Naív algoritmus

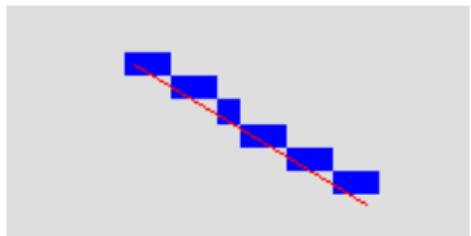
- $m = \text{float}(y_2 - y_1) / (x_2 - x_1)$
nem pontos



Szakasz rászterizálása

Naív algoritmus

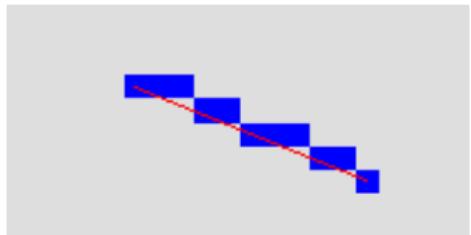
- $m = \text{float}(y2-y1) / (x2-x1)$
nem pontos
 - $y += m \rightarrow$ a hiba gyűlik y-ban



Szakasz rászterizálása

Naív algoritmus

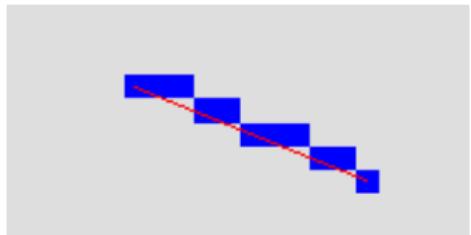
- $m = \text{float}(y2-y1) / (x2-x1)$
nem pontos
 - $y += m \rightarrow$ a hiba gyűlik y-ban



Szakasz rászterizálása

Naív algoritmus

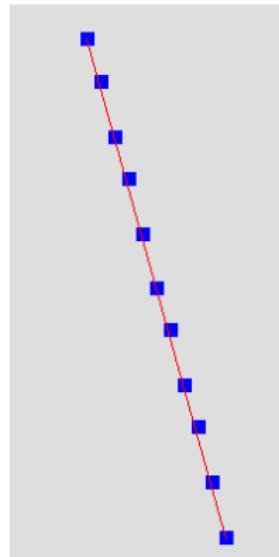
- $m = \text{float}(y_2 - y_1) / (x_2 - x_1)$
nem pontos
 - $y += m \rightarrow$ a hiba gyűlik y -ban
 - `draw.point((x, y))` →
int-eket vár, lassú a konverzió



Szakasz raszterizálása

Naív algoritmus

- $m = \text{float}(y2-y1) / (x2-x1)$
nem pontos
 - $y += m \rightarrow$ a hiba gyűlik y -ban
 - `draw.point((x, y))` →
int-eket vár, lassú a konverzió
 - csak $|m| < 1$ -re működik
helyesen



Szakasz raszterizálása

Javítsuk az algoritmust 1.

```
def line2(x1,y1,x2,y2, draw):
    m = float(y2-y1)/(x2-x1)
    x = x1
    y = y1
    e = 0.0
    while x<=x2:
        draw.point((x,y))
        x += 1
        e += m
        if e >= 0.5:
            y += 1
            e -= 1.0
```

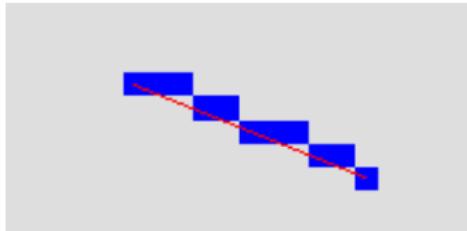


Szakasz raszterizálása

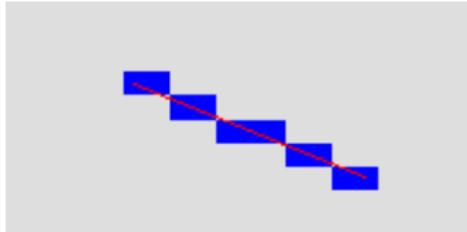
Javítsuk az algoritmust 1.

- Jó: Mindig "eltalálja" a végpontokat
- Jó: Egyenletesebben lép az y irányban.
- Rossz: Még mindig használunk float-okat

Naív:



Javítás:



Szakasz raszterizálása

Javítsuk az algoritmust 2.

```
def line3 (x1 ,y1 ,x2 ,y2 , draw ):  
    x = x1  
    y = y1  
    e = -0.5←  
    while x<=x2:  
        draw . point ((x ,y ))  
        x += 1  
        e += float (y2-y1 )/(x2-x1 )←  
        if e >= 0.0:←  
            y += 1  
            e -= 1.0
```

Szakasz raszterizálása

Javítsuk az algoritmust 3.

```
def line4 (x1 ,y1 ,x2 ,y2 , draw ):  
    x = x1  
    y = y1  
    e = -0.5*(x2-x1 )←  
    while x<=x2 :  
        draw . point ((x ,y ))  
        x += 1  
        e += y2-y1 ←  
        if e >= 0.0 :  
            y += 1  
            e -= (x2-x1 )←
```

Szakasz raszterizálása

Javítsuk az algoritmust 4.

```
def line5(x1,y1,x2,y2, draw):
    x = x1
    y = y1
    e = -(x2-x1)←
    while x<=x2:
        draw.point((x,y))
        x += 1
        e += 2*(y2-y1)←
        if e >= 0:
            y += 1
            e -= 2*(x2-x1)←
```

Szakasz raszterizálása

Javítsuk az algoritmust 4.

- Ez a *Bresenham* algoritmus (egyik speciális esete)
 - Külön gyűjtjük a hibát e -ben
 - Nem használunk `float`-okat
 - Tetszőleges meredekségű szakaszokra általánosítható.

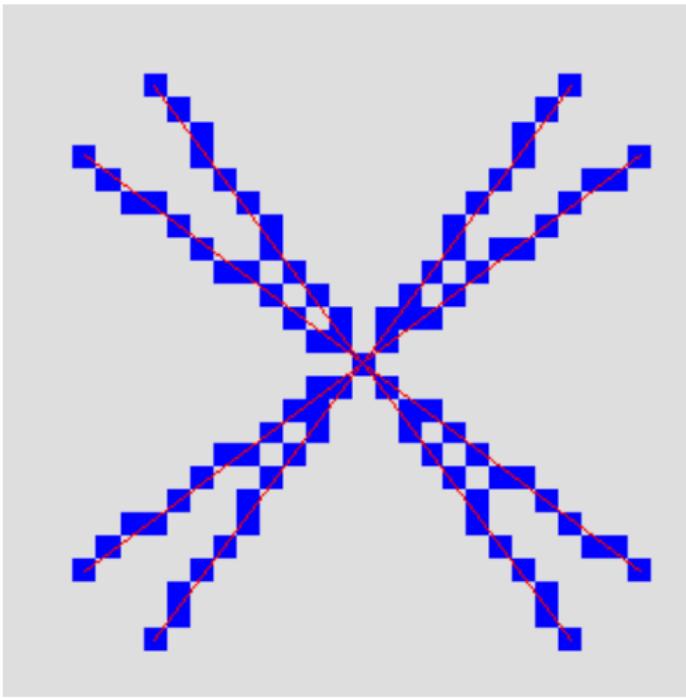
Szakasz raszterizálása

Bresenham algoritmus

- Nyolcadokra kéne felbontani a síkot, minden eset.
- (Előzők végig: jobbra-le)
- El kell döntenünk, hogy $|x_2 - x_1|$ vagy $|y_2 - y_1|$ a nagyobb (merre meredekebb a szakasz).
- Ha $|y_2 - y_1|$ a nagyobb, cseréljük fel $x_i \leftrightarrow y_i$, és rajzolásnál is fordítva használjuk!
- Ha $x_1 > x_2$, akkor csere: $x_1 \leftrightarrow x_2$, $y_1 \leftrightarrow y_2$.
- Az e hibatagot $|y_2 - y_1|$ -nal növeljük minden lépésben
- y -nál $y_2 - y_1$ előjele szerint haladunk.

Szakasz raszterizálása

Bresenham algoritmus



Szakasz raszterizálása

Teljes *Bresenham* algoritmust 1.

```
def Bresenham(x1,y1,x2,y2, draw):
    steep = abs(y2-y1)>abs(x2-x1)
    if steep:
        x1, y1 = y1, x1
        x2, y2 = y2, x2
    if x1>x2:
        x1, x2 = x2, x1
        y1, y2 = y2, y1
    Dy = abs(y2-y1)
    if y1<y2:
        Sy = 1
    else:
        Sy = -1
```

Szakasz raszterizálása

Teljes *Bresenham* algoritmust 2.

```
x = x1
y = y1
e = -(x2-x1)
while x<=x2:
    if steep:
        draw.point((y,x))
    else:
        draw.point((x,y))
    x += 1
    e += 2*Dy
    if e >= 0:
        y += Sy
        e -= 2*(x2-x1)
```

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
 - Pont és szakaszvágás
 - Szakaszok vágása
 - Szakaszvágás
 - Poligonvágás

3 Raszterizálás

- Szakasz raszterizálása
 - Háromszög raszterizálása
 - Poligon raszterizáció

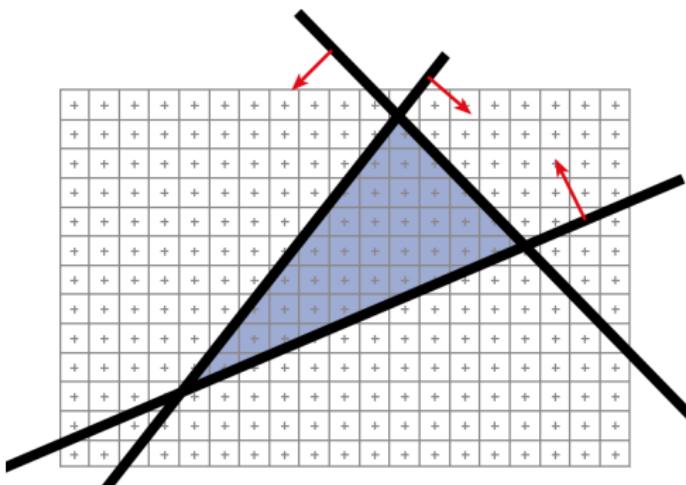
Háromszög raszterizálása

Háromszög raszterizáció

- A háromszög oldalait tudjuk vágni - most töltük ki a belsejét!
 - Ha egy meghatározott bejárási irányban adtuk meg az összes háromszög csúcsát, tudunk félsíkokat adni (tudjuk irányítani az éleket) → u.i. ha (t_x, t_y) az irányvektora az oldalnak, akkor $(-t_y, t_x)$ egy normális lesz
 - minden pixelre menjünk végig a képernyőn és nézzük meg, hogy a háromszög oldalai által meghatározott síkok jó oldalán van-e!

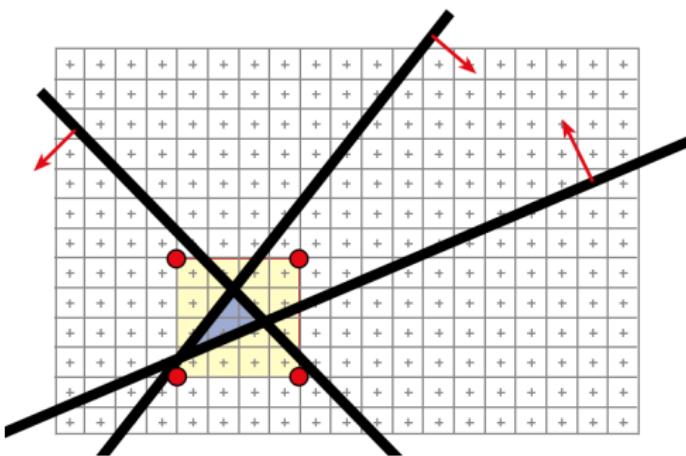
Háromszög raszterizálása

Háromszög raszterizáció



Háromszög raszterizálása

Háromszög raszterizáció - okosabban



Háromszög raszterizálása

Háromszög raszterizáció

- Lehetne még okosabban is csinálni, de: gyakorlatban ez a brute-force megközelítés nagyon jól alkalmazható!

Háromszög raszterizáció

- Nem egy rögzített értékkel akarunk kitölteni, a hanem a csúcsokban adott értékeket akarjuk interpolálni.
 - Felhasználásai: szín (Gouraud-árnyalás), textúra koordináták, normálvektorok
 - Legyen a felület egy pontja $p = \alpha p_1 + \beta p_2 + \gamma p_3$, az α, β, γ baricentrikus koordinákkal adott.
 - Ekkor bármilyen más értéket is végig tudunk interpolálni ugyan így:

$$c = \alpha c_1 + \beta c_2 + \gamma c_3$$

- Ez az úgy nevezett *Gouraud interpoláció* (nem véletlenül)

Háromszög raszterizálása

Háromszög kitöltés 1.

```

for all x:
  for all y:
     $\alpha, \beta, \gamma = \text{barycentric}(x, y)$ 
    if  $\alpha \in [0, 1]$  and  $\beta \in [0, 1]$  and  $\gamma \in [0, 1]$ :
       $c = \alpha c_1 + \beta c_2 + \gamma c_3$ 
      draw.point((x, y), c)

```

Háromszög rászterizálása

Háromszög kitöltés 2.

- Gyorsítás: felesleges minden x, y -t vizsgálni, elég a háromszöget tartalmazó téglalapon végig menni.
- Inkrementálissá tétele:
 - Most még lassú, nem használjuk, ki hogy sorban megyünk x -en, y -on.
 - Milyenek is ezek az f -ek?
 - Mind $f(x, y) = Ax + By + C$ alakú.
 - Ekkor $f(x + 1, y) = f(x, y) + A$, ill.
 - $f(x, y + 1) = f(x, y) + B$
- Megvalósítás: *házi feladat*



Poligon raszterizáció

Tartalom

1 Emlékeztető

2 Vágás

- Motiváció
- Pont és szakaszvágás
- Szakaszok vágása
- Szakaszvágás
- Poligonvágás

3 Raszterizálás

- Szakasz raszterizálása
- Háromszög raszterizálása
- Poligon raszterizáció

Flood-fill – elárasztásos kitöltés

- Tetszőleges, már raszterizált poligon kitöltésére alkalmas.
 - Bemenet: raszter kép + annak egy pontja
 - Brute-force: a megadott pontból kiindulva rekurzívan haladunk:
 - Az aktuális pont színe megegyezik a kiindulási pont színével?
 - Nem: megállunk;
 - Igen: átszínezzük, és minden szomszédra újrakezdjük az algoritmust.

Flood-fill – szomszédságok

- Négy szomszéd: fent, lent, jobbra, balra
 - Nyolc szomszéd: az előző négy + a sarkak
 - Rekurzió nagyon durva: gyakorlatban ennél okosabb algoritmusok is vannak: → aktív éllista stb. (Haladó Grafika)

Számítógépes Grafika

Valasek Gábor

valasek@inf.elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2011/2012. őszi félév

Tartalom

1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés
- Textúra szűrés
- Procedurális textúrák
- Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- Grafikus gyorsítókártya generációk

Tartalom

1 Textúrázás

- Bevezetés
 - Textúra leképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
 - DirectX
 - OpenGL
 - Grafikus gyorsítókártya generációk

Bevezetés

Mi az a textúrázás?

- Eddig: egyszínű anyag modellek, azaz a teljes felület azonos színű.

Bevezetés

Mi az a textúrázás?

- Eddig: egyszínű anyag modellek, azaz a teljes felület azonos színű.
 - Kevés ilyen van a valóságban.

Bevezetés

Mi az a textúrázás?

- Eddig: egyszínű anyag modellek, azaz a teljes felület azonos színű.
 - Kevés ilyen van a valóságban.
 - Finom részleteket szeretnénk megadni.

Bevezetés

Mi az a textúrázás?

- Eddig: egyszínű anyag modellek, azaz a teljes felület azonos színű.
 - Kevés ilyen van a valóságban.
 - Finom részleteket szeretnénk megadni.
 - Változó paraméterekre van szükségünk a BRDF-ben.

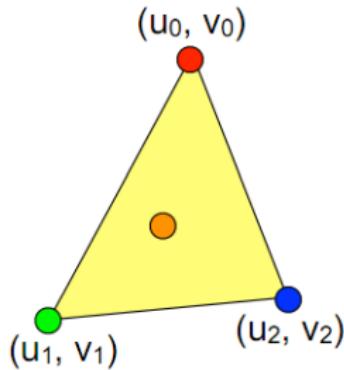
Bevezetés

Mi az a textúrázás?

- Eddig: egyszínű anyag modellek, azaz a teljes felület azonos színű.
 - Kevés ilyen van a valóságban.
 - Finom részleteket szeretnénk megadni.
 - Változó paraméterekre van szükségünk a BRDF-ben.
 - Ezeket a paramétereket, elsősorban színt, adjuk meg a *textúrákban*.

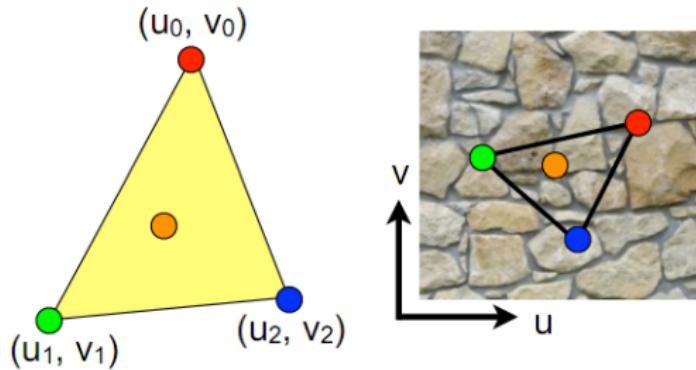
Bevezetés

Textúrázás



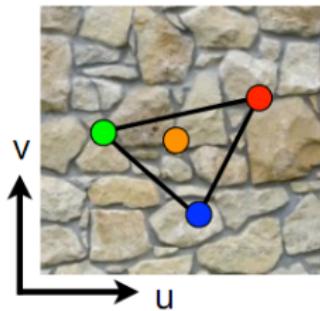
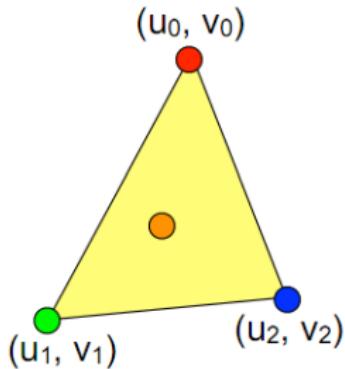
Bevezetés

Textúrázás



Bevezetés

Textúrázás



Bevezetés

Textúra megadási módok

- "Tömbbel":

Bevezetés

Textúra megadási módok

- "Tömbbel":
 - valamelyen 1/2/3 dimenziós tömbből olvasunk, elemei a *texel*-ek

Textúra megadási módok

- "Tömbbel":
 - valamelyen 1/2/3 dimenziós tömbből olvasunk, elemei a *texel*-ek
 - 2D szemléletesen: a geometriánkat a tömbben tárolt "képpel" "fedjük/csomagoljuk be"

Textúra megadási módok

- "Tömbbel":
 - valamelyen 1/2/3 dimenziós tömbből olvasunk, elemei a *texel*-ek
 - 2D szemléletesen: a geometriánkat a tömbben tárolt "képpel" "fedjük/csomagoljuk be"
 - *textúratérben* vett koordinátákkal azonosítjuk a texeleket, a koordináták [0, 1] intervallumon vannak értelmezve

Bevezetés

Textúra megadási módok

- "Tömbbel":
 - valamelyen 1/2/3 dimenziós tömbből olvasunk, elemei a *texel*-ek
 - 2D szemléletesen: a geometriánkat a tömbben tárolt "képpel" "fedjük/csomagoljuk be"
 - *textúratérben* vett koordinátákkal azonosítjuk a texeleket, a koordináták [0, 1] intervallumon vannak értelmezve
 - Függvénynel adjuk meg:

Bevezetés

Textúra megadási módok

- "Tömbbel":
 - valamelyen 1/2/3 dimenziós tömbből olvasunk, elemei a *texel*-ek
 - 2D szemléletesen: a geometriánkat a tömbben tárolt "képpel" "fedjük/csomagoljuk be"
 - *textúratérben* vett koordinátkkal azonosítjuk a texeleket, a koordináták $[0, 1]$ intervallumon vannak értelmezve
 - Függvénynel adjuk meg:
 - valamelyen $f(x, y, z) \rightarrow$ szín függvény segítségével

Bevezetés

Textúra megadási módok

- "Tömbbel":
 - valamelyen 1/2/3 dimenziós tömbből olvasunk, elemei a *texel*-ek
 - 2D szemléletesen: a geometriánkat a tömbben tárolt "képpel" "fedjük/csomagoljuk be"
 - *textúratérben* vett koordinátákkal azonosítjuk a texeleket, a koordináták $[0, 1]$ intervallumon vannak értelmezve
 - Függvénynel adjuk meg:
 - valamelyen $f(x, y, z) \rightarrow$ szín függvény segítségével
 - ezt nevezzük *procedurális textúrázásnak*

Textúra leképezés

Tartalom

1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés
- Textúra szűrés
- Procedurális textúrák
- Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- Grafikus gyorsítókártya generációk

Textúra leképezés

Leképezési módok

- Két terünk van: képtér (képernyő pixelei) és *textúratér* (textúra texelei)

Textúra leképezés

Leképezési módok

- Két terünk van: *képtér* (képernyő pixelei) és *textúratér* (textúra texelei)
- Honnan, hova képezzünk?

Textúra leképezés

Leképezési módok

- Két terünk van: képtér (képernyő pixelei) és *textúratér* (textúra texelei)
- Honnan, hova képezzünk?
- Textúratér → képtér: *textúra alapú leképezés*

Textúra leképezés

Leképezési módok

- Két terünk van: *képtér* (képernyő pixelei) és *textúratér* (textúra texelei)
- Honnan, hova képezzünk?
- Textúratér → képtér: *textúra alapú leképezés*
 - » minden *texel*hez keressünk a neki megfelelő *pixelt*.

Textúra leképezés

Leképezési módok

- Két terünk van: *képtér* (képernyő pixelei) és *textúratér* (textúra texelei)
- Honnan, hova képezzünk?
- Textúratér → képtér: *textúra alapú leképezés*
 - » minden *texel*hez keressünk a neki megfelelő *pixelt*.
 - ⊕ Hatékony.

Textúra leképezés

Leképezési módok

- Két terünk van: *képtér* (képernyő pixelei) és *textúratér* (textúra texelei)
- Honnan, hova képezzünk?
- Textúratér → képtér: *textúra alapú leképezés*
 - » minden *texel*hez keressünk a neki megfelelő *pixelt*.
 - ⊕ Hatékony.
 - ⊖ Nem biztos, hogy minden pixel sorra kerül, és ami mégis, nem biztos, hogy csak egyszer.

Textúra leképezés

Leképezési módok

- Két terünk van: *képtér* (képernyő pixelei) és *textúratér* (textúra texelei)
- Honnan, hova képezzünk?
- Textúratér → képtér: *textúra alapú leképezés*
 - » minden *texel*hez keressünk a neki megfelelő *pixelt*.
 - ⊕ Hatékony.
 - ⊖ Nem biztos, hogy minden pixel sorra kerül, és ami mégis, nem biztos, hogy csak egyszer.
- Képtér → textúratér: *képtér alapú leképezés*

Textúra leképezés

Leképezési módok

- Két terünk van: **képtér** (képernyő pixelei) és **textúratér** (textúra texelei)
- Honnan, hova képezzünk?
- Textúratér → képtér: **textúra alapú leképezés**
 - » minden *texel*hez keressünk a neki megfelelő *pixelt*.
 - ⊕ Hatékony.
 - ⊖ Nem biztos, hogy minden pixel sorra kerül, és ami mégis, nem biztos, hogy csak egyszer.
- Képtér → textúratér: **képtér alapú leképezés**
 - » minden *pixel*hez keresünk a neki megfelelő *texelt*.

Textúra leképezés

Leképezési módok

- Két terünk van: *képtér* (képernyő pixelei) és *textúratér* (textúra texelei)
- Honnan, hova képezzünk?
- Textúratér → képtér: *textúra alapú leképezés*
 - » minden *texel*hez keressünk a neki megfelelő *pixelt*.
 - ⊕ Hatékony.
 - ⊖ Nem biztos, hogy minden pixel sorra kerül, és ami mégis, nem biztos, hogy csak egyszer.
- Képtér → textúratér: *képtér alapú leképezés*
 - » minden *pixel*hez keresünk a neki megfelelő *texelt*.
 - ⊕ Inkrementális képszintézishez jól passzol.

Textúra leképezés

Leképezési módok

- Két terünk van: *képtér* (képernyő pixelei) és *textúratér* (textúra texelei)
- Honnan, hova képezzünk?
- Textúratér → képtér: *textúra alapú leképezés*
 - » minden *texel*hez keressünk a neki megfelelő *pixelt*.
 - ⊕ Hatékony.
 - ⊖ Nem biztos, hogy minden pixel sorra kerül, és ami mégis, nem biztos, hogy csak egyszer.
- Képtér → textúratér: *képtér alapú leképezés*
 - » minden *pixel*hez keresünk a neki megfelelő *texelt*.
 - ⊕ Inkrementális képszintézishez jól passzol.
 - ⊖ Szükség van hozzá a paraméterezési és vetítési transzformációk inverzére.

Paraméterezés

Tartalom

1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés**
- Textúra szűrés
- Procedurális textúrák
- Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- Grafikus gyorsítókártya generációk

Paraméterezés

Paraméterezés

- Hogyan tudjuk eldönten, hogy az egyes felületi pontokhoz a tömb melyik elemét választjuk, vagy a procedurális textúra függvényét milyen paraméterekkel értékeljük ki?

Paraméterezés

Paraméterezés

- Hogyan tudjuk eldönten, hogy az egyes felületi pontokhoz a tömb melyik elemét választjuk, vagy a procedurális textúra függvényét milyen paraméterekkel értékeljük ki?
- A felület minden pontjához *textúra koordinátákat* rendelünk.

Paraméterezés

Paraméterezés

- Hogyan tudjuk eldönten, hogy az egyes felületi pontokhoz a tömb melyik elemét választjuk, vagy a procedurális textúra függvényét milyen paraméterekkel értékeljük ki?
- A felület minden pontjához *textúra koordinátákat* rendelünk.
- A textúra koordináták hozzárendelését a felülethez nevezzük most *paraméterezésnek*.

Paraméterezés

Paraméterezés

- Hogyan tudjuk eldönten, hogy az egyes felületi pontokhoz a tömb melyik elemét választjuk, vagy a procedurális textúra függvényét milyen paraméterekkel értékeljük ki?
- A felület minden pontjához *textúra koordinátákat* rendelünk.
- A textúra koordináták hozzárendelését a felülethez nevezzük most *paraméterezésnek*.
- A továbbiakban 2D textúrákról beszélünk.

Paraméterezés

Parametrikus felületek paraméterezése

- Parametrikus felület:

$$F \in \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad F(u, v) := (x, y, z)$$

Parametrikus felületek paraméterezése

- Parametrikus felület:

$$F \in \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad F(u, v) := (x, y, z)$$

- u, v paraméterek természetes módon használhatóak textúra koordinátáknak.

Parametrikus felületek paraméterezése

- Parametrikus felület:

$$F \in \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad F(u, v) := (x, y, z)$$

- u, v paraméterek természetes módon használhatóak textúra koordinátáknak.
- Ha $\mathcal{D}_F \neq \mathcal{D}_{tex}$, akkor u, v -t transzformálni kell.

Parametrikus felületek paraméterezése

- Parametrikus felület:

$$F \in \mathbb{R}^2 \rightarrow \mathbb{R}^3, \quad F(u, v) := (x, y, z)$$

- u, v paraméterek természetes módon használhatóak textúra koordinátáknak.
- Ha $\mathcal{D}_F \neq \mathcal{D}_{tex}$, akkor u, v -t transzformálni kell.
- Pi:
 - Henger palást
 - $\mathcal{D}_F = [0, 2\pi] \times [0, h]$, $F(u, v) := (\cos u, h, \sin u)$
 - Textúra tér: $(\bar{u}, \bar{v}) \in [0, 1] \times [0, 1]$
 - Transzformáció: $\bar{u} := u/2\pi$, $\bar{v} := v/h$

Háromszögek paraméterezése

- Legyen adott a háromszög három csúcsa:

$p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, $i \in \{1, 2, 3\}$, valamint az ezeknek megfelelő csúcsok textúra térben:

$$t_i = (u_i, v_i) \in \mathbb{R}^2, i \in \{1, 2, 3\}.$$

Háromszögek paraméterezése

- Legyen adott a háromszög három csúcsa:
 $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, $i \in \{1, 2, 3\}$, valamint az ezeknek megfelelő csúcsok textúra térben:
 $t_i = (u_i, v_i) \in \mathbb{R}^2$, $i \in \{1, 2, 3\}$.
- Olyan $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ leképezést keresünk, amire $p_i \mapsto t_i$, és háromszöget háromszögbe visz át.

Paraméterezés

Háromszögek paraméterezése

- Legyen adott a háromszög három csúcsa:
 $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, $i \in \{1, 2, 3\}$, valamint az ezeknek megfelelő csúcsok textúra térben:
 $t_i = (u_i, v_i) \in \mathbb{R}^2$, $i \in \{1, 2, 3\}$.
- Olyan $\mathbb{R}^3 \rightarrow \mathbb{R}^2$ leképezést keresünk, amire $p_i \mapsto t_i$, és háromszöget háromszögbe visz át.
- A legegyszerűbb ilyen leképezés a lineáris leképezés, ami megadható egy 3×3 -as mátrixszal.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Paraméterezés

Háromszögek paraméterezése

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- Kilenc ismeretlen, kilenc egyenlet

Paraméterezés

Háromszögek paraméterezése

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- Kilenc ismeretlen, kilenc egyenlet
- Ez textúra alapú leképezés!

Paraméterezés

Háromszögek paraméterezése

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} A_x & A_y & A_z \\ B_x & B_y & B_z \\ C_x & C_y & C_z \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

- Kilenc ismeretlen, kilenc egyenlet
- Ez textúra alapú leképezés!
- Képtér alapú leképezéshez inverz trafó kell:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P}^{-1} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Paraméterezés

Textúrázás és sugárkövetés

- Felület-sugár metszés: világ-koordináta rendszerben

Paraméterezés

Textúrázás és sugárkövetés

- Felület-sugár metszés: világ-koordináta rendszerben
- Inverz transzformációk:

Paraméterezés

Textúrázás és sugárkövetés

- Felület-sugár metszés: világ-koordináta rendszerben
- Inverz transzformációk:
 - Világ KR → Model KR

Paraméterezés

Textúrázás és sugárkövetés

- Felület-sugár metszés: világ-koordináta rendszerben
- Inverz transzformációk:
 - Világ KR → Model KR
 - Model KR → textúra tér

Paraméterezés

Model KR → textúra tér

- Parametrikus felületek

A metszéspont számítás során adódik u, v , nem kell külön számítani.

Paraméterezés

Model KR → textúra tér

- Parametrikus felületek

A metszéspont számítás során adódik u, v , nem kell külön számítani.

- Háromszögek

Az előbb levezetett \mathbf{P}^{-1} szükséges.

Gyorsítási lehetőség: ha sem maga a háromszög, sem a textúra koordináták nem változnak a csúcsokban, akkor \mathbf{P}^{-1} állandó.

Paraméterezés

Háromszögek paraméterezése

- Gyakorlatban ez gyorsan számítható inkrementális algoritmussal.

Paraméterezés

Háromszögek paraméterezése

- Gyakorlatban ez gyorsan számítható inkrementális algoritmussal.
- Ha a három csúcsban adottak a textúra koordináták, akkor ezeket a háromszögek kitöltésénél használt algoritmussal kiszámíthatjuk minden pixelre.

Paraméterezés

Háromszögek paraméterezése

- Gyakorlatban ez gyorsan számítható inkrementális algoritmussal.
- Ha a három csúcsban adottak a textúra koordináták, akkor ezeket a háromszögek kitöltésénél használt algoritmussal kiszámíthatjuk minden pixelre.
- minden pontra legyen a felület egy pontja
 $\mathbf{p} = \alpha\mathbf{p}_1 + \beta\mathbf{p}_2 + \gamma\mathbf{p}_3$, az α, β, γ baricentrikus koordinátákkal adott.

Paraméterezés

Háromszögek paraméterezése

- Gyakorlatban ez gyorsan számítható inkrementális algoritmussal.
- Ha a három csúcsban adottak a textúra koordináták, akkor ezeket a háromszögek kitöltésénél használt algoritmussal kiszámíthatjuk minden pixelre.
- minden pontra legyen a felület egy pontja
 $\mathbf{p} = \alpha\mathbf{p}_1 + \beta\mathbf{p}_2 + \gamma\mathbf{p}_3$, az α, β, γ baricentrikus koordinákkal adott.
- Ekkor a \mathbf{p} -hez tartozó textúra koordináta is megkapható
 $t = \alpha t_1 + \beta t_2 + \gamma t_3$ alapján

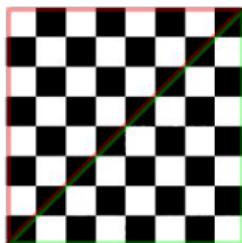
Paraméterezés

Perspektívikusan korrekt textúrázás

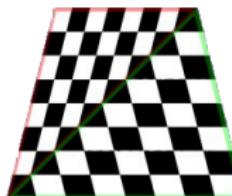
- A textúra koordináták lineáris interpolációja hibás képet ad, ha a megjelenítendő háromszögön nem csak affin transzformációt végzünk.

Perspektívikusan korrekt textúrázás

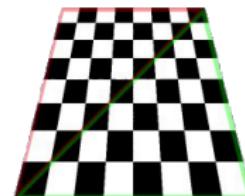
- A textúra koordináták lineáris interpolációja hibás képet ad, ha a megjelenítendő háromszögön nem csak affin transzformációt végzünk.
- Értsd: esetek 99%-a.



Sík



Affin



Helyes

Paraméterezés

Perspektívikusan korrekt textúrázás

- Tehát u, v lineáris interpolációja nem lesz jó nem affin transzformációk esetén - nem lineárisak a képernyőtérben

Perspektívikusan korrekt textúrázás

- Tehát u, v lineáris interpolációja nem lesz jó nem affin transzformációk esetén - nem lineárisak a képernyőtérben
- Transzformálva, homogén osztás előtt a koordinátáink legyenek $[x_t, y_t, z_t, w_t]$

Perspektívikusan korrekt textúrázás

- Tehát u, v lineáris interpolációja nem lesz jó nem affin transzformációk esetén - nem lineárisak a képernyőtérben
- Transzformálva, homogén osztás előtt a koordinátáink legyenek $[x_t, y_t, z_t, w_t]$
- Homogén osztás után: $[x_s, y_s, z_s, 1] = [\frac{x_t}{w_t}, \frac{y_t}{w_t}, \frac{z_t}{w_t}, 1]$

Perspektívikusan korrekt textúrázás

- Tehát u, v lineáris interpolációja nem lesz jó nem affin transzformációk esetén - nem lineárisak a képernyőtérben
- Transzformálva, homogén osztás előtt a koordinátáink legyenek $[x_t, y_t, z_t, w_t]$
- Homogén osztás után: $[x_s, y_s, z_s, 1] = [\frac{x_t}{w_t}, \frac{y_t}{w_t}, \frac{z_t}{w_t}, 1]$
- Ha $[x_s, y_s, z_s, 1]$ szerint interpoláljuk a textúra koordinátákat, akkor az nem lesz jó.

Paraméterezés

Perspektívikusan korrekt textúrázás

- Tehát u, v lineáris interpolációja nem lesz jó nem affin transzformációk esetén - nem lineárisak a képernyőtérben
- Transzformálva, homogén osztás előtt a koordinátáink legyenek $[x_t, y_t, z_t, w_t]$
- Homogén osztás után: $[x_s, y_s, z_s, 1] = [\frac{x_t}{w_t}, \frac{y_t}{w_t}, \frac{z_t}{w_t}, 1]$
- Ha $[x_s, y_s, z_s, 1]$ szerint interpoláljuk a textúra koordinátákat, akkor az nem lesz jó.
- Ezzel szemben: ha $\frac{1}{w}$ -t interpoláljuk, az helyes marad! Sőt, bármilyen $\frac{q}{w}$ érték jól interpolálódik!

Perspektívikusan korrekt textúrázás

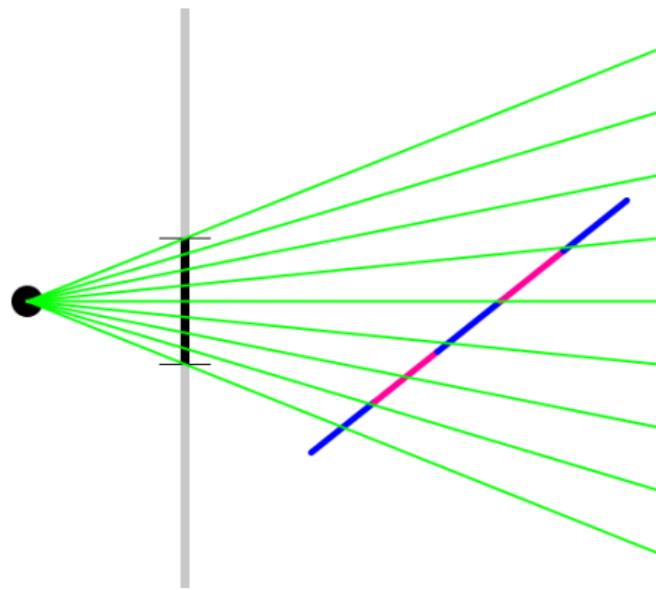
- Tehát u, v lineáris interpolációja nem lesz jó nem affin transzformációk esetén - nem lineárisak a képernyőtérben
- Transzformálva, homogén osztás előtt a koordinátáink legyenek $[x_t, y_t, z_t, w_t]$
- Homogén osztás után: $[x_s, y_s, z_s, 1] = [\frac{x_t}{w_t}, \frac{y_t}{w_t}, \frac{z_t}{w_t}, 1]$
- Ha $[x_s, y_s, z_s, 1]$ szerint interpoláljuk a textúra koordinátákat, akkor az nem lesz jó.
- Ezzel szemben: ha $\frac{1}{w}$ -t interpoláljuk, az helyes marad! Sőt, bármilyen $\frac{q}{w}$ érték jól interpolálódik!
- α, β, γ helyett használjuk a világ koordináta-rendszerbeli $\alpha_w, \beta_w, \gamma_w$ koordinátákat, és

Perspektívikusan korrekt textúrázás

- Tehát u, v lineáris interpolációja nem lesz jó nem affin transzformációk esetén - nem lineárisak a képernyőtérben
- Transzformálva, homogén osztás előtt a koordinátáink legyenek $[x_t, y_t, z_t, w_t]$
- Homogén osztás után: $[x_s, y_s, z_s, 1] = [\frac{x_t}{w_t}, \frac{y_t}{w_t}, \frac{z_t}{w_t}, 1]$
- Ha $[x_s, y_s, z_s, 1]$ szerint interpoláljuk a textúra koordinátákat, akkor az nem lesz jó.
- Ezzel szemben: ha $\frac{1}{w}$ -t interpoláljuk, az helyes marad! Sőt, bármilyen $\frac{q}{w}$ érték jól interpolálódik!
- α, β, γ helyett használjuk a világ koordináta-rendszerbeli $\alpha_w, \beta_w, \gamma_w$ koordinátákat, és
- $\frac{\alpha_w}{w}, \frac{\beta_w}{w}, \frac{\gamma_w}{w}$ interpolálva helyes textúrázást kapunk.

Paraméterezés

Perspektívikusan korrekt textúrázás



Textúra szűrés

Tartalom

1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés
- **Textúra szűrés**
- Procedurális textúrák
- Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- Grafikus gyorsítókártya generációk

Textúra szürés

Textúrák szűrése

- Ritkán esik pontosan egy texel egy pixelre.

Textúra szürés

Textúrák szűrése

- Ritkán esik pontosan egy texel egy pixelre.
- Nagyítás: a pixel mérete kisebb a texelénél – több pixel jut egyetlen texelre. OpenGL: GL_TEXTURE_MAG_FILTER

Textúra szűrés

Textúrák szűrése

- Ritkán esik pontosan egy texel egy pixelre.
- Nagyítás: a pixel mérete kisebb a texelénél – több pixel jut egyetlen texelre. OpenGL: GL_TEXTURE_MAG_FILTER
- Kicsinyítés: a pixel mérete nagyobb a texelénél – több texel jut egyetlen pixelre. OpenGL: GL_TEXTURE_MIN_FILTER

Textúra szűrés

Textúrák szűrése

- Ritkán esik pontosan egy texel egy pixelre.
- Nagyítás: a pixel mérete kisebb a texelénél – több pixel jut egyetlen texelre. OpenGL: GL_TEXTURE_MAG_FILTER
- Kicsinyítés: a pixel mérete nagyobb a texelénél – több texel jut egyetlen pixelre. OpenGL: GL_TEXTURE_MIN_FILTER
- További probléma: ami a textúra térben lineáris, az nem az a képtérben a perspektíva miatt.

Textúra szűrés

Nagyítás

Egy pixel mérete kisebb egy texelénél – több pixel jut egyetlen texelre

- Szűrés nélkül: a pixel középpontjához legközelebb eső texel értékét használjuk.

Textúra szürés

Nagyítás

Egy pixel mérete kisebb egy texelénél – több pixel jut egyetlen texelre

- Szűrés nélkül: a pixel középpontjához legközelebb eső texel értékét használjuk.
- Bilineáris szűrés: a legközelebbi négy texel súlyozott átlagát vesszük.

Textúra szürés

Kicsinyítés

Egy pixel mérete nagyobb egy texelénél – több texel jut egyetlen pixelre

- Pontos mintavételezés lenne: a pixel négyzetét transzformáljuk el textúra térbe, és az ott kijelölt texelek átlagát vegyük.

Textúra szűrés

Kicsinyítés

Egy pixel mérete nagyobb egy texelénél – több texel jut egyetlen pixelre

- Pontos mintavételezés lenne: a pixel négyzetét transzformáljuk el textúra térbe, és az ott kijelölt texelek átlagát vegyük.
- Helyette: a textúra térben is négyzetet veszünk.

Textúra szürés

Kicsinyítés

Egy pixel mérete nagyobb egy texelénél – több texel jut egyetlen pixelre

- Pontos mintavételezés lenne: a pixel négyzetét transzformáljuk el textúra térbe, és az ott kijelölt texelek átlagát vegyük.
- Helyette: a textúra térben is négyzetet veszünk.
- Gyakorlatban: a textúra egy tetszőleges négyzetének a leátlagolása túl erőforrás igényes, helyette vagy használunk kevesebb texelt, vagy *MIP-map*-eket

Textúra szürés

Kicsinyítés

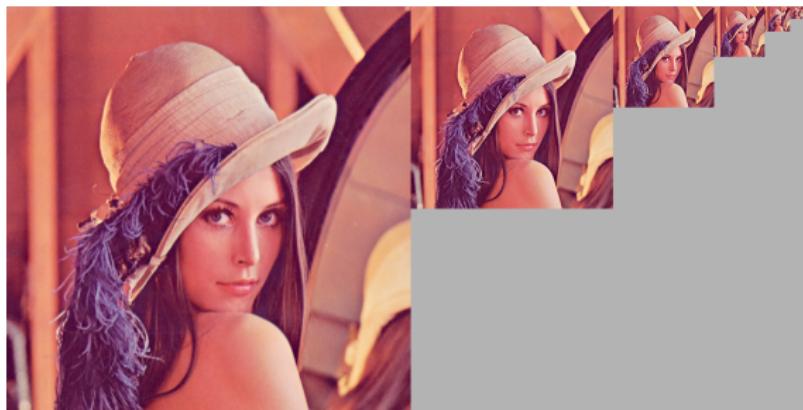
Egy pixel mérete nagyobb egy texelénél – több texel jut egyetlen pixelre

- Pontos mintavételezés lenne: a pixel négyzetét transzformáljuk el textúra térbe, és az ott kijelölt texelek átlagát vegyük.
- Helyette: a textúra térben is négyzetet veszünk.
- Gyakorlatban: a textúra egy tetszőleges négyzetének a leátlagolása túl erőforrás igényes, helyette vagy használunk kevesebb texelt, vagy *MIP-map*-eket
- Kevesebb pixel:
 - Szűrés nélkül: a pixel középpontjához legközelebb eső texel értékét használjuk.
 - Bilineáris szűrés: a legközelebbi négy texel súlyozott átlagát vesszük.

Textúra szürés

MIP-map-ek

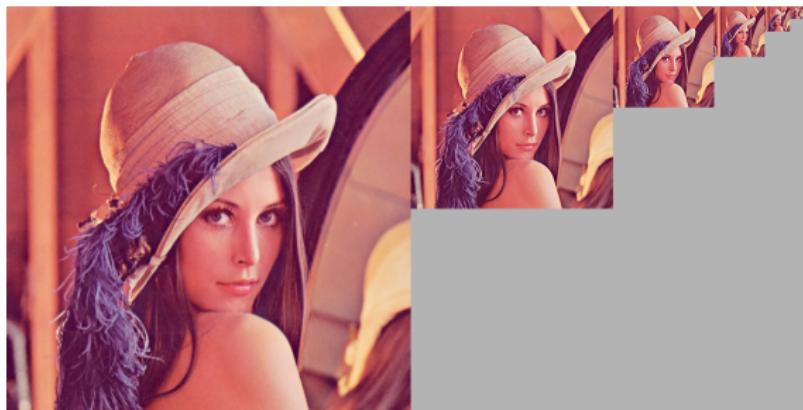
- MIP: *multum in parvo* – sok, kis helyen



Textúra szürés

MIP-map-ek

- MIP: *multum in parvo* – sok, kis helyen
- Ú.n. pirasmist generálunk a textúrából ből, minden szinten felezve a méretét



Textúra szürés

MIP-map-ek

- Szűrés során kiválasztjuk a pixel/texel terület aránynak megfelelő szintet és onnan olvasunk.

Textúra szürés

MIP-map-ek

- Szűrés során kiválasztjuk a pixel/texel terület aránynak megfelelő szintet és onnan olvasunk.
- Az adott MIP-map-en belül is lehet az olvasás szűrés nélküli, vagy lehet bilienárisan szűrt.

Textúra szürés

MIP-map-ek

- Szűrés során kiválasztjuk a pixel/texel terület aránynak megfelelő szintet és onnan olvasunk.
- Az adott MIP-map-en belül is lehet az olvasás szűrés nélküli, vagy lehet bilienárisan szűrt.
- *Trilineáris szűrés*: két szomszédos szintet használunk, azokon belül bilineáris szűréssel, és ezeknek vesszük a súlyoztott átlagát.

Procedurális textúrák

Tartalom

1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés
- Textúra szűrés
- **Procedurális textúrák**
- Nem-szín textúrák

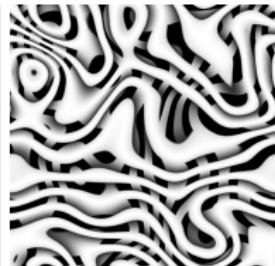
2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- Grafikus gyorsítókártya generációk

Procedurális textúrák

Procedurális textúrák

- A textúrákat "tömb" helyett megadhatjuk függvényel is.

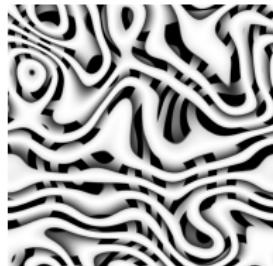
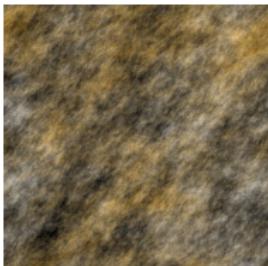


generated with Filter Forge

Procedurális textúrák

Procedurális textúrák

- A textúrákat "tömb" helyett megadhatjuk függvényteljesen.
- Textúra koordináták → a függvény paramétereit.



generated with Filter Forge

Procedurális textúrák

Tulajdonságai

- Előnyök:

Procedurális textúrák

Tulajdonságai

- Előnyök:
 - Sokkal kevesebb tárhely

Procedurális textúrák

Tulajdonságai

- Előnyök:

- Sokkal kevesebb tárhely
- Tetszőleges felbontás – csak a numerikus pontosság a korlát

Procedurális textúrák

Tulajdonságai

- Előnyök:

- Sokkal kevesebb tárhely
- Tetszőleges felbontás – csak a numerikus pontosság a korlát
⇒ Nagyításnál nem kell szűrés. (Kicsinyítést sajnos nem oldja meg.)

Procedurális textúrák

Tulajdonságai

- Előnyök:
 - Sokkal kevesebb tárhely
 - Tetszőleges felbontás – csak a numerikus pontosság a korlát
 - ⇒ Nagyításnál nem kell szűrés. (Kicsinyítést sajnos nem oldja meg.)
- Hátrányok:

Procedurális textúrák

Tulajdonságai

- Előnyök:
 - Sokkal kevesebb tárhely
 - Tetszőleges felbontás – csak a numerikus pontosság a korlát
 - ⇒ Nagyításnál nem kell szűrés. (Kicsinyítést sajnos nem oldja meg.)
- Hátrányok:
 - Nagy számítás igény.

Procedurális textúrák

Tulajdonságai

- Előnyök:
 - Sokkal kevesebb tárhely
 - Tetszőleges felbontás – csak a numerikus pontosság a korlát
 - ⇒ Nagyításnál nem kell szűrés. (Kicsinyítést sajnos nem oldja meg.)
- Hátrányok:
 - Nagy számítás igény.
 - Nem lehet vele bármit leírni.

Procedurális textúrák

Tulajdonságai

- Előnyök:
 - Sokkal kevesebb tárhely
 - Tetszőleges felbontás – csak a numerikus pontosság a korlát
 - ⇒ Nagyításnál nem kell szűrés. (Kicsinyítést sajnos nem oldja meg.)
- Hátrányok:
 - Nagy számítás igény.
 - Nem lehet vele bármit leírni.
 - Nem módosítható tetszőlegesen.

Nem-szín textúrák

Tartalom

1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés
- Textúra szűrés
- Procedurális textúrák
- Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- Grafikus gyorsítókártya generációk

Nem-szín textúrák

Nem-szín textúrák

- A textúrák segítségével a felületi pont bármilyen tulajdonságát leírhatjuk.

Nem-szín textúrák

Nem-szín textúrák

- A textúrák segítségével a felületi pont bármilyen tulajdonságát leírhatjuk.
- Ezek a tulajdonságok lehetnek pl.:
 - felületi normális – *Bucka ill. normál leképezés*

Nem-szín textúrák

Nem-szín textúrák

- A textúrák segítségével a felületi pont bármilyen tulajdonságát leírhatjuk.
- Ezek a tulajdonságok lehetnek pl.:
 - felületi normális – *Bucka ill. normál leképezés*
 - elmozdulás – *Displacement leképezés*

Nem-szín textúrák

Nem-szín textúrák

- A textúrák segítségével a felületi pont bármilyen tulajdonságát leírhatjuk.
- Ezek a tulajdonságok lehetnek pl.:
 - felületi normális – *Bucka ill. normál leképezés*
 - elmozdulás – *Displacement leképezés*
 - fényforrás láthatósága – *Árnyék térképek*

Nem-szín textúrák

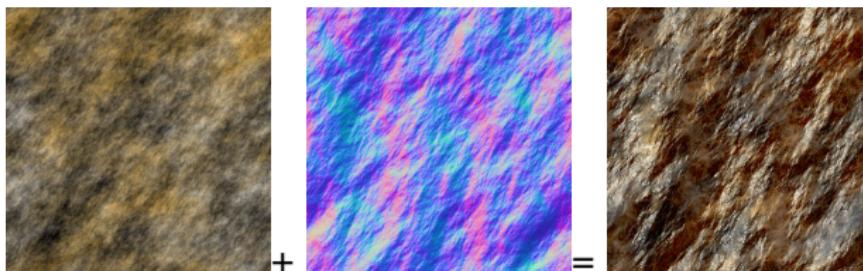
Nem-szín textúrák

- A textúrák segítségével a felületi pont bármilyen tulajdonságát leírhatjuk.
- Ezek a tulajdonságok lehetnek pl.:
 - felületi normális – *Bucka ill. normál leképezés*
 - elmozdulás – *Displacement leképezés*
 - fényforrás láthatósága – *Árnyék térképek*
 - tükröként visszavert fény – *Visszaverődés leképezés ill. Környezet leképezés*

Nem-szín textúrák

Bucka vagy normál leképezés

- A textúrával normál vektorokat adunk meg.

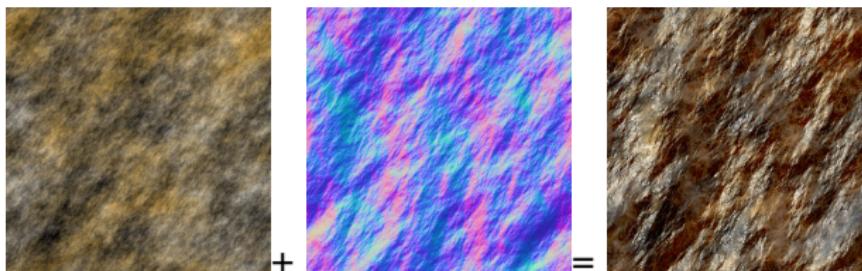


generated with Filter Forge

Nem-szín textúrák

Bucka vagy normál leképezés

- A textúrával normál vektorokat adunk meg.
- A felület eredeti normálisai helyett ezeket használjuk a megvilágítás számításakor.

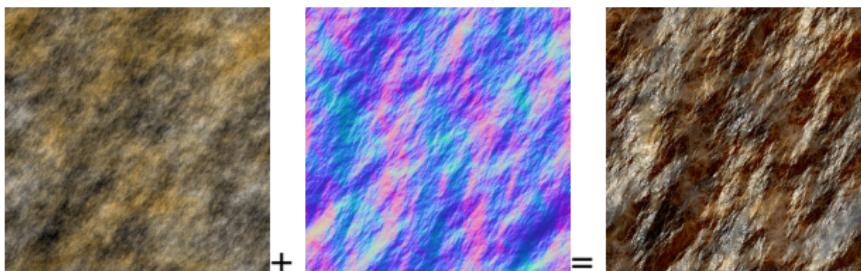


generated with Filter Forge

Nem-szín textúrák

Bucka vagy normál leképezés

- A textúrával normál vektorokat adunk meg.
- A felület eredeti normálisai helyett ezeket használjuk a megvilágítás számításakor.
- Rücskös/érdes felület látszatát adja, amíg nem nézünk rá túl lapos szögben.

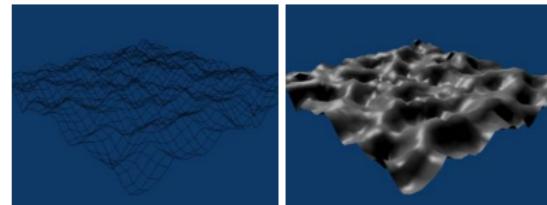
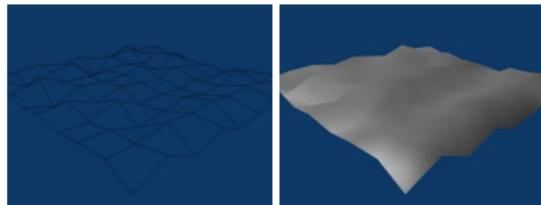


generated with Filter Forge

Nem-szín textúrák

Displacement leképezés

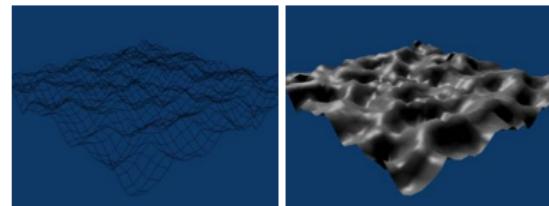
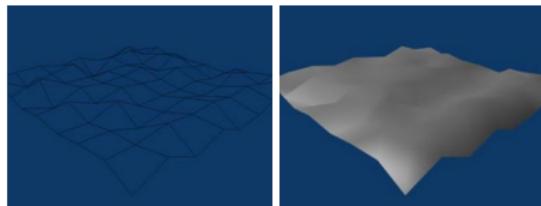
- A feületi pontot tényleges arrébb mozdítjuk.



Nem-szín textúrák

Displacement leképezés

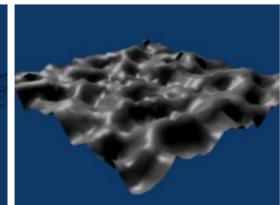
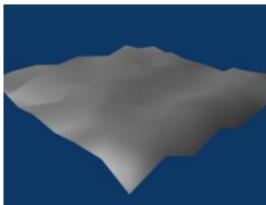
- A feületi pontot tényleges arrébb mozdítjuk.
- A csak háromszögek csúcsait tudjuk elmozdítani \Rightarrow függ a geometria felbontásától.



Nem-szín textúrák

Displacement leképezés

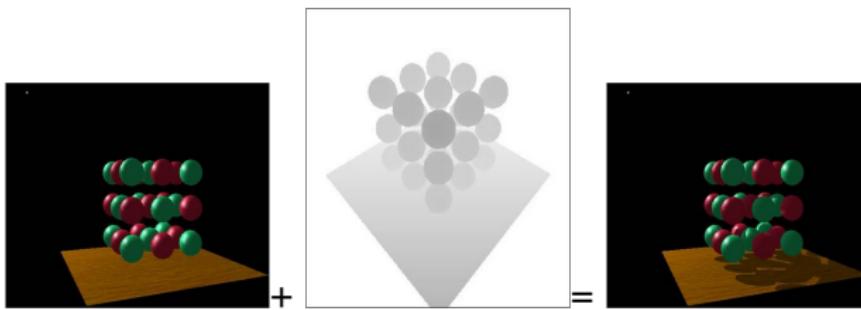
- A feületi pontot tényleges arrébb mozdítjuk.
- A csak háromszögek csúcsait tudjuk elmozdítani \Rightarrow függ a geometria felbontásától.
- Lapos szögben is helyes látványt kapunk.



Nem-szín textúrák

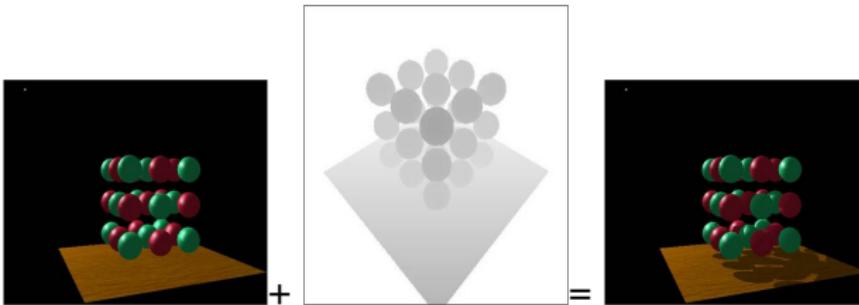
Árnyék térképek

- A fényforrás szemszögéből készítünk egy textúrát, amiben a fényforrástól vett távolságokat tároljuk el.



Árnyék térképek

- A fényforrás szemszögéből készítünk egy textúrát, amiben a fényforrástól vett távolságokat tároljuk el.
 - A tényleges rajzolás során ez alapján döntjük el minden pontra, hogy azt éri-e közvetlenül a fény vagy sem.



Nem-szín textúrák

Visszaverődés leképezés

- Sík tükrök esetén használható.

Nem-szín textúrák

Visszaverődés leképezés

- Sík tükrök esetén használható.
 - Külön képet készítünk, textúrába mentve, arról, hogy mi látszik tükrirányban.

Nem-szín textúrák

Visszaverődés leképezés

- Sík tükrök esetén használható.
- Külön képet készítünk, textúrába mentve, arról, hogy mi látszik tükrirányban.
- Ezt textúraként ráfeszítjük a felületre a végső kirajzoláskor.

Visszaverődés leképezés

- Sík tükrök esetén használható.
 - Külön képet készítünk, textúrába mentve, arról, hogy mi látszik tükörirányban.
 - Ezt textúraként ráfeszítjük a felületre a végső kirajzoláskor.
 - Csak egyszeres tükrözést ad.

Nem-szín textúrák

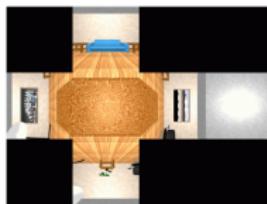
Visszaverődés leképezés

- Sík tükrök esetén használható.
- Külön képet készítünk, textúrába mentve, arról, hogy mi látszik tükrirányban.
- Ezt textúraként ráfeszítjük a felületre a végső kirajzoláskor.
- Csak egyszeres tükrözést ad.
- Tükrönként külön el kell végezni.

Nem-szín textúrák

Környezet leképezés

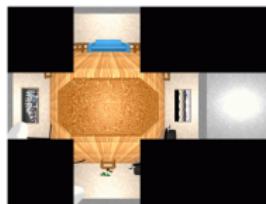
- A színtérünk környezetét végtelenül távolinak tekintjük.



Nem-szín textúrák

Környezet leképezés

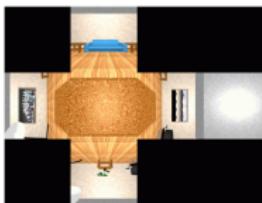
- A színtérünk környezetét végtelenül távolinak tekintjük.
- Speciális textúrában tároljuk.



Nem-szín textúrák

Környezet leképezés

- A színtérünk környezetét végtelenül távolinak tekintjük.
- Speciális textúrában tároljuk.
- Rajzolás a tükröződő felületekhez ebből olvasunk a tükrirány szerint.



Inkrementális képszintézis

Tartalom

1 Textúrázás

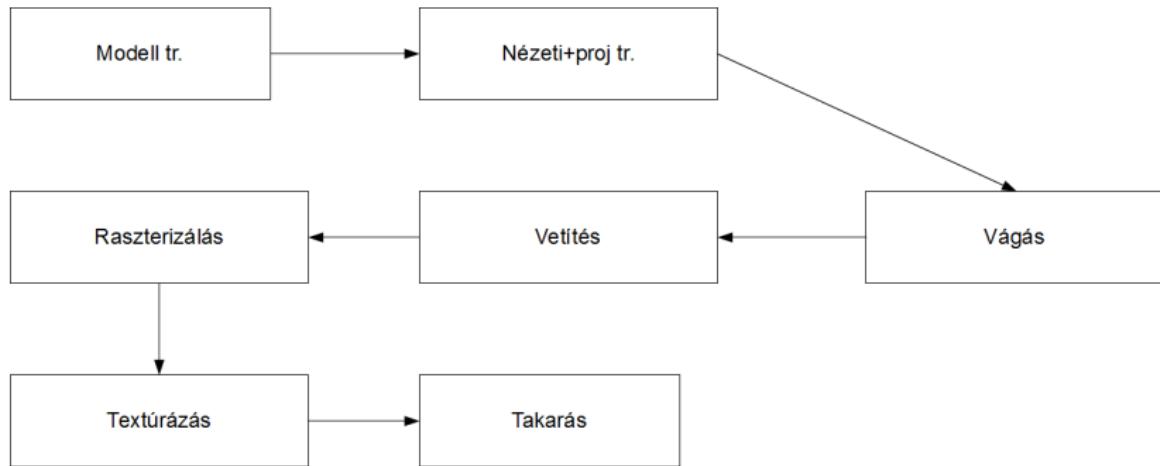
- Bevezetés
- Textúra leképezés
- Paraméterezés
- Textúra szűrés
- Procedurális textúrák
- Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- Grafikus gyorsítókártya generációk

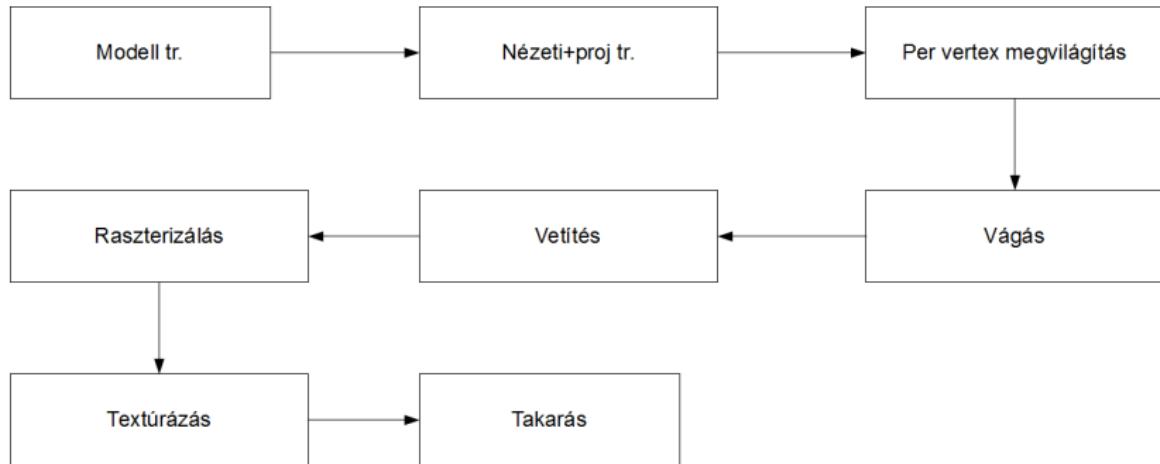
Inkrementális képszintézis

Inkrementális képszintézis



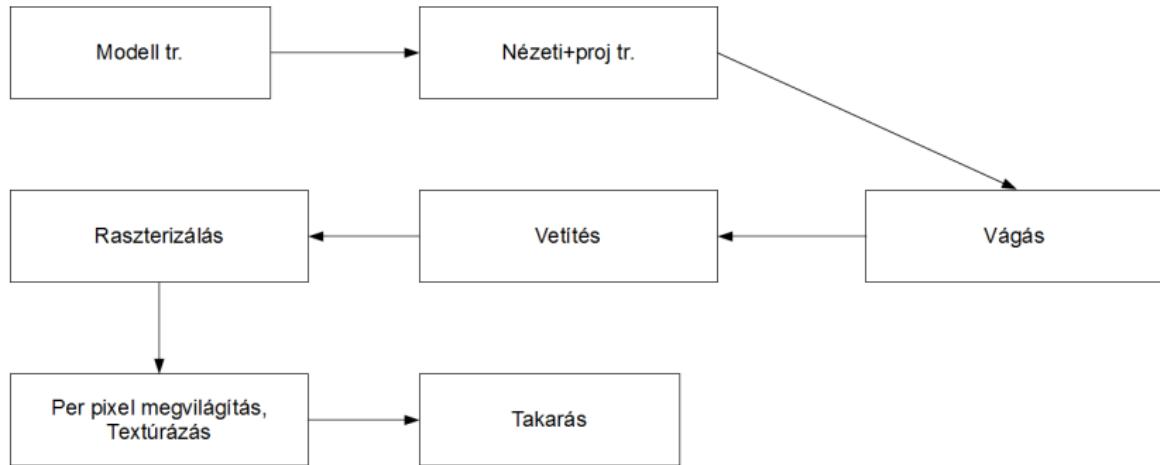
Inkrementális képszintézis

Inkrementális képszintézis



Inkrementális képszintézis

Inkrementális képszintézis



Tartalom

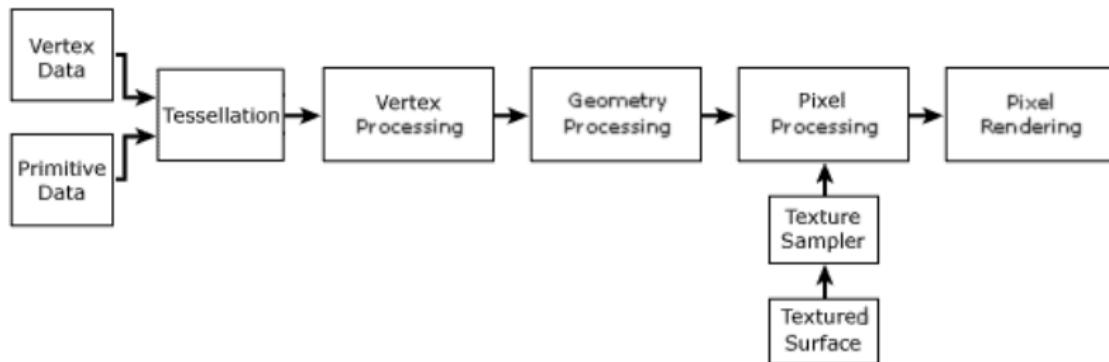
1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés
- Textúra szűrés
- Procedurális textúrák
- Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- Grafikus gyorsítókártya generációk

DirectX 9 pipeline



DirectX 9 pipeline

- Vertex data: a modell-koordinátarendszerbeli pozícióadatok

DirectX 9 pipeline

- Vertex data: a modell-koordinátarendszerbeli pozícióadatok
- Primitive data: összekötési szabályok, az előbbiekből hogyan kapunk rajzolási primitíveket

DirectX 9 pipeline

- Vertex data: a modell-koordinátarendszerbeli pozícióadatok
- Primitive data: összekötési szabályok, az előbbiekből hogyan kapunk rajzolási primitíveket
- Tesselation: magasabb rendű primitívek lineáris közelítése, a közelítések csúcspontjainak vertexpufferbe helyezése

DirectX 9 pipeline

- Vertex data: a modell-koordinátarendszerbeli pozícióadatok
- Primitive data: összekötési szabályok, az előbbiekből hogyan kapunk rajzolási primitíveket
- Tesselation: magasabb rendű primitívek lineáris közelítése, a közelítések csúcspontjainak vertexpufferbe helyezése
- Vertex processing: a világ, nézeti és projektív transzformációk elvégzése; a modell-koordinátarendszerből NPKR-be visz ($z \in [0, 1]$)

DirectX 9 pipeline

- Geometry processing: vágás, hátlapeldobás, raszterizáció

DirectX 9 pipeline

- Geometry processing: vágás, hátlapeldobás, raszterizáció
- Textured surface: 1/2/3D-s textúra

DirectX 9 pipeline

- Geometry processing: vágás, hátlapeldobás, raszterizáció
- Textured surface: 1/2/3D-s textúra
- Texture sampler: textúra mintavételező, szűrések stb.

DirectX 9 pipeline

- Geometry processing: vágás, hátlapeldobás, raszterizáció
- Textured surface: 1/2/3D-s textúra
- Texture sampler: textúra mintavételező, szűrések stb.
- Pixel processing: a bejövő fragment színének kiszámítása

DirectX 9 pipeline

- Geometry processing: vágás, hátlapeldobás, raszterizáció
- Textured surface: 1/2/3D-s textúra
- Texture sampler: textúra mintavételező, szűrések stb.
- Pixel processing: a bejövő fragment színének kiszámítása
- Pixel rendering: képernyőre kerülő pixel színének meghatározása (mélységi és egyéb tesztek)

DirectX 9 pipeline

- A fenti fázisok közül kettő programozható, a vertex és a pixel feldolgozás, a többi konfigurálható

DirectX 9 pipeline

- A fenti fázisok közül kettő programozható, a vertex és a pixel feldolgozás, a többi konfigurálható
- Vertex shader:

DirectX 9 pipeline

- A fenti fázisok közül kettő programozható, a vertex és a pixel feldolgozás, a többi konfigurálható
- Vertex shader:
 - bemenete: általában modell-KR-beli pont és egyéb attribútumok

DirectX 9 pipeline

- A fenti fázisok közül kettő programozható, a vertex és a pixel feldolgozás, a többi konfigurálható
- Vertex shader:
 - bemenete: általában modell-KR-beli pont és egyéb attribútumok
 - kimenete: NPKR-beli pont és egyéb attribútumok

DirectX 9 pipeline

- A fenti fázisok közül kettő programozható, a vertex és a pixel feldolgozás, a többi konfigurálható
- Vertex shader:
 - bemenete: általában modell-KR-beli pont és egyéb attribútumok
 - kimenete: NPKR-beli pont és egyéb attribútumok
- Pixel shader:

DirectX 9 pipeline

- A fenti fázisok közül kettő programozható, a vertex és a pixel feldolgozás, a többi konfigurálható
- Vertex shader:
 - bemenete: általában modell-KR-beli pont és egyéb attribútumok
 - kimenete: NPKR-beli pont és egyéb attribútumok
- Pixel shader:
 - bemenete: a raszterizálás végén előálló fragmentek - az összes attribútum elérhető, amit a vertex shaderben előállítottunk

DirectX 9 pipeline

- A fenti fázisok közül kettő programozható, a vertex és a pixel feldolgozás, a többi konfigurálható
- Vertex shader:
 - bemenete: általában modell-KR-beli pont és egyéb attribútumok
 - kimenete: NPKR-beli pont és egyéb attribútumok
- Pixel shader:
 - bemenete: a raszterizálás végén előálló fragmentek - az összes attribútum elérhető, amit a vertex shaderben előállítottunk
 - kimenete: egy színérték

DirectX 9 pipeline

- Régen a vertex és a pixel feldolgozási fázis sem volt programozható

DirectX 9 pipeline

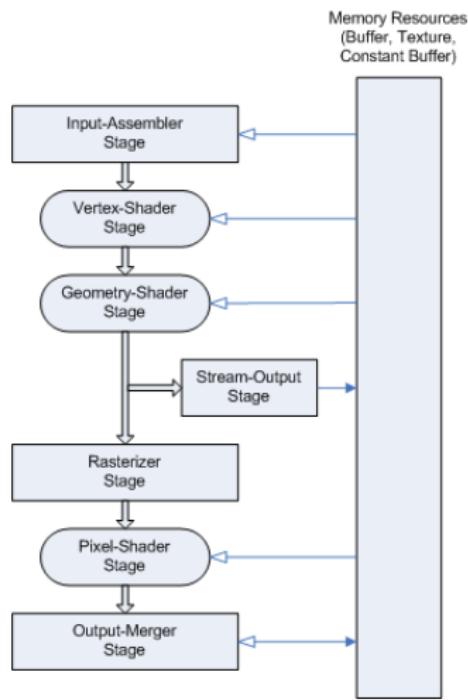
- Régen a vertex és a pixel feldolgozási fázis sem volt programozható
- Csak konfigurálni tudtuk őket → pl. transzformációs mátrixokat átadni, a DX-be bedrótozott megvilágítási modelleket felpáraméterezni stb.

DirectX 9 pipeline

- Régen a vertex és a pixel feldolgozási fázis sem volt programozható
- Csak konfigurálni tudtuk őket → pl. transzformációs mátrixokat átadni, a DX-be bedrótozott megvilágítási modelleket felpáraméterezni stb.
- Ezt hívták Fixed Function Pipeline-nak

DirectX

DirectX 10 pipeline



DirectX 10 pipeline

- Input-Assembler stage: a pipeline bemeneti adatainak előállítását végzi

DirectX 10 pipeline

- Input-Assembler stage: a pipeline bemeneti adatainak előállítását végzi
- Vertex-Shader stage: ld. mint korábban

DirectX 10 pipeline

- Input-Assembler stage: a pipeline bemeneti adatainak előállítását végzi
- Vertex-Shader stage: ld. mint korábban
- Geometry-Shader stage: bemenetként egész primitíveket kap (háromszögek esetén 3, szakaszoknál 2, pontknál 1 pont), eldobhatja a bejövő primitívet vagy egy vagy több új primitívet létrehozhat és a szerelőszalagra helyezheti. Ez egy új, programozható fázis

DirectX 10 pipeline

- Stream-Output Stage: a pipeline-ból a memóriába való adattovábbításra használható, a rasszterizálás előtt történik. Adatokat be- és ki is lehet írni rasszterizáláshoz.

DirectX 10 pipeline

- Stream-Output Stage: a pipeline-ból a memóriába való adattovábbításra használható, a rászterizálás előtt történik. Adatokat be- és ki is lehet írni rászterizáláshoz.
- Rasterizer Stage: rászterizálás

DirectX 10 pipeline

- Stream-Output Stage: a pipeline-ból a memóriába való adattovábbításra használható, a rászterizálás előtt történik. Adatokat be- és ki is lehet írni rászterizáláshoz.
- Rasterizer Stage: rászterizálás
- Pixel-Shader Stage: ld. korábban

DirectX 10 pipeline

- Stream-Output Stage: a pipeline-ból a memóriába való adattovábbításra használható, a rászterizálás előtt történik. Adatokat be- és ki is lehet írni rászterizáláshoz.
- Rasterizer Stage: rászterizálás
- Pixel-Shader Stage: ld. korábban
- Output-Merger Stage: a különböző kimeneti adatok (szín-, mélységi- és egyéb információk) felhasználásával a végső eredmény kiszámítása

Geometry Shader

- Feladata: új geometria generálása, (vagy régi eltüntetése)

Geometry Shader

- Feladata: új geometria generálása, (vagy régi eltüntetése)
- A vertex shader által már transzformált primitívekre fut le.

Geometry Shader

- Feladata: új geometria generálása, (vagy régi eltüntetése)
- A vertex shader által már transzformált primitívekre fut le.
- Bemenete: teljes primitív (szomszédsági infókkal, ha vannak)

Geometry Shader

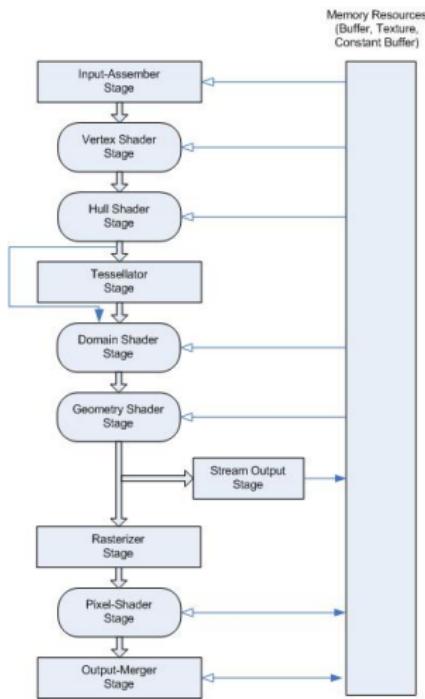
- Feladata: új geometria generálása, (vagy régi eltüntetése)
- A vertex shader által már transzformált primitívekre fut le.
- Bemenete: teljes primitív (szomszédsági infókkal, ha vannak)
- Kimenete: újabb primitív[ek], vagy semmi.

Geometry Shader

- Feladata: új geometria generálása, (vagy régi eltüntetése)
- A vertex shader által már transzformált primitívekre fut le.
- Bemenete: teljes primitív (szomszédsági infókkal, ha vannak)
- Kimenete: újabb primitív[ek], vagy semmi.
- Kimeneti típusok: *PointStream*, *LineStream*, vagy *TriangleStream*

DirectX

DirectX 11 pipeline



DirectX 11 pipeline

- A DX10-es pipeline-nak megfelelő, de itt a tesszeláció már programozható

DirectX 11 pipeline

- A DX10-es pipeline-nak megfelelő, de itt a tesszeláció már programozható
- Emellett a DX11-ben az általános számításokra való felhasználás is előtérbe került → ComputeShader

DirectX 11 pipeline

- A DX10-es pipeline-nak megfelelő, de itt a tesszeláció már programozható
- Emellett a DX11-ben az általános számításokra való felhasználás is előtérbe került → ComputeShader
- Régóta nagy az igény a GPU-kban rejlő számítási kapacitás kihasználására

DirectX 11 pipeline

- A DX10-es pipeline-nak megfelelő, de itt a tesszeláció már programozható
- Emellett a DX11-ben az általános számításokra való felhasználás is előtérbe került → ComputeShader
- Régóta nagy az igény a GPU-kban rejlő számítási kapacitás kihasználására
- Uj. pl.: Intel Core i7 980 XE: 109 GFLOPS, Nvidia GTX 480: 672 GFLOPS, dupla pontosságban

Tesszelációs lépések

- Speciális primitív tipusokat használ

Tesszelációs lépések

- Speciális primitív tipusokat használ
 - négyzetök foltok (*patches*)

Tesszelációs lépések

- Speciális primitív tipusokat használ
 - négyzetű foltok (*patches*)
 - háromszög foltok

Tesszelációs lépések

- Speciális primitív tipusokat használ
 - négyzet foltok (*patches*)
 - háromszög foltok
 - iso-vonalak

Tesszelációs lépések

- Speciális primitív tipusokat használ
 - négyzet foltok (*patches*)
 - háromszög foltok
 - iso-vonalak
- Alapötlet: a transzformációkat az alacsony felbontású foltokon végezzük, és tesszelációval, hardveresen bontjuk fel kisebb darabokra és tesszük részletgazdagtávba.

Tesszelációs lépések

- Speciális primitív tipusokat használ
 - négyzet foltok (*patches*)
 - háromszög foltok
 - iso-vonalak
- Alapötlet: a transzformációkat az alacsony felbontású foltokon végezzük, és tesszelációval, hardveresen bontjuk fel kisebb darabokra és tesszük részletgazdagtávba.
- Használató:

Tesszelációs lépések

- Speciális primitív tipusokat használ
 - négyzet foltok (*patches*)
 - háromszög foltok
 - iso-vonalak
- Alapötlet: a transzformációkat az alacsony felbontású foltokon végezzük, és tesszelációval, hardveresen bontjuk fel kisebb darabokra és tesszük részletgazdagtávba.
- Használató:
 - *displacement mapping*

Tesszelációs lépések

- Speciális primitív tipusokat használ
 - négyzetű foltok (*patches*)
 - háromszög foltok
 - iso-vonalak
- Alapötlet: a transzformációkat az alacsony felbontású foltokon végezzük, és tesszelációval, hardveresen bontjuk fel kisebb darabokra és tesszük részletgazdagtávba.
- Használató:
 - *displacement mapping*
 - nézet függő dinamikus LOD (*level of detail*)

Tesszelációs lépések

- Speciális primitív tipusokat használ
 - négyzetű foltok (*patches*)
 - háromszög foltok
 - iso-vonalak
- Alapötlet: a transzformációkat az alacsony felbontású foltokon végezzük, és tesszelációval, hardveresen bontjuk fel kisebb darabokra és tesszük részletgazdagtávba.
- Használató:
 - *displacement mapping*
 - nézet függő dinamikus LOD (*level of detail*)
 - *morph*-olások gyorsítása

Hull Shader Stage

Hull Shader Stage

Hull Shader Stage

- Foltonként dolgozza fel a bementi pontokat.

Hull Shader Stage

- Foltonként dolgozza fel a bementi pontokat.
- Konstansokat rendel a folthoz, amik a tesszelálás módját határozzák meg.

Hull Shader Stage

- Foltonként dolgozza fel a bementi pontokat.
- Konstansokat rendel a folthoz, amik a tesszelálás módját határozzák meg.
- Megadja a tesszeláció mértékét.

Hull Shader Stage

- Foltonként dolgozza fel a bementi pontokat.
- Konstansokat rendel a folthoz, amik a tesszelálás módját határozzák meg.
- Megadja a tesszeláció mértékét.
- Két külön kimenet:

Hull Shader Stage

- Foltonként dolgozza fel a bementi pontokat.
- Konstansokat rendel a folthoz, amik a tesszelálás módját határozzák meg.
- Megadja a tesszeláció mértékét.
- Két külön kimenet:
 - Vezérlő pontok a *Domain Shader*-nek

Hull Shader Stage

- Foltonként dolgozza fel a bementi pontokat.
- Konstansokat rendel a folthoz, amik a tesszelálás módját határozzák meg.
- Megadja a tesszeláció mértékét.
- Két külön kimenet:
 - Vezérlő pontok a *Domain Shader*-nek
 - Konstansok a *Tesselator Stage*-nek

Hull Shader Stage

- Foltonként dolgozza fel a bementi pontokat.
- Konstansokat rendel a folthoz, amik a tesszelálás módját határozzák meg.
- Megadja a tesszeláció mértékét.
- Két külön kimenet:
 - Vezérlő pontok a *Domain Shader*-nek
 - Konstansok a *Tesselator Stage*-nek



Tesselator Stage

- Nem programozható.

Tesselator Stage

- Nem programozható.
- Felbontja a tartományt kisebb darabokra (négyszög, háromszög, szakasz).

Tesselator Stage

- Nem programozható.
- Felbontja a tartományt kisebb darabokra (négyszög, háromszög, szakasz).
- u, v (opcionálisan w) koordinátákat állít elő kimenetként.

Tesselator Stage

- Nem programozható.
- Felbontja a tartományt kisebb darabokra (négyszög, háromszög, szakasz).
- u, v (opcionálisan w) koordinátákat állít elő kimenetként.
- "Láthatatlan" kimenet: topológiai információk a *primitive assembly*-nek

Domain Shader Stage

Domain Shader Stage

Domain Shader Stage

- A *Tesselator Stage* során előállított minden egyes pontra lefut.

Domain Shader Stage

- A *Tesselator Stage* során előállított minden egyes pontra lefut.
- Bemenete:

Domain Shader Stage

- A *Tesselator Stage* során előállított minden egyes pontra lefut.
- Bemenete:
 - A *Tesselator Stage*-től:
 $u, v, (w)$ -k.

Domain Shader Stage

- A *Tesselator Stage* során előállított minden egyes pontra lefut.
- Bemenete:
 - A *Tesselator Stage*-től:
 $u, v, (w)$ -k.
 - A *Hull Shader*-től:
vezérlő pontok

Domain Shader Stage

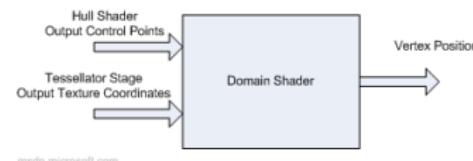
- A *Tesselator Stage* során előállított minden egyes pontra lefut.
- Bemenete:
 - A *Tesselator Stage*-től:
 $u, v, (w)$ -k.
 - A *Hull Shader*-től:
vezérlő pontok
 - A *Hull Shader*-től:
tesszelációs faktorok

Domain Shader Stage

- A *Tesselator Stage* során előállított minden egyes pontra lefut.
- Bemenete:
 - A *Tesselator Stage*-től:
 $u, v, (w)$ -k.
 - A *Hull Shader*-től:
vezérlő pontok
 - A *Hull Shader*-től:
tesszelációs faktorok
- A bemenetekből egyetlen csúcspontot állít elő, és az lesz a kimenet.

Domain Shader Stage

- A *Tesselator Stage* során előállított minden egyes pontra lefut.
- Bemenete:
 - A *Tesselator Stage*-től:
 $u, v, (w)$ -k.
 - A *Hull Shader*-től:
vezérlő pontok
 - A *Hull Shader*-től:
tesszelációs faktorok
- A bemenetekből egyetlen csúcspontot állít elő, és az lesz a kimenet.



msdn.microsoft.com

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.
- Felhasználásai:

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.
- Felhasználásai:
 - Képfeldolgozás/post-processing

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.
- Felhasználásai:
 - Képfeldolgozás/post-processing
 - Ray-tracing

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.
- Felhasználásai:
 - Képfeldolgozás/post-processing
 - Ray-tracing
 - Fizika

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.
- Felhasználásai:
 - Képfeldolgozás/post-processing
 - Ray-tracing
 - Fizika
 - AI

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.
- Felhasználásai:
 - Képfeldolgozás/post-processing
 - Ray-tracing
 - Fizika
 - AI
- "Testvérei"

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.
- Felhasználásai:
 - Képfeldolgozás/post-processing
 - Ray-tracing
 - Fizika
 - AI
- "Testvérei"
 - CUDA

Compute Shader

- Általános célú számításokat tesz lehetővé a GPU-n.
- Speciális adatszerkezeteket használ.
- Nincsenek megszorítások a ki- és bemenetekre.
- Felhasználásai:
 - Képfeldolgozás/post-processing
 - Ray-tracing
 - Fizika
 - AI
- "Testvérei"
 - CUDA
 - OpenCL

Tartalom

1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés
- Textúra szűrés
- Procedurális textúrák
- Nem-szín textúrák

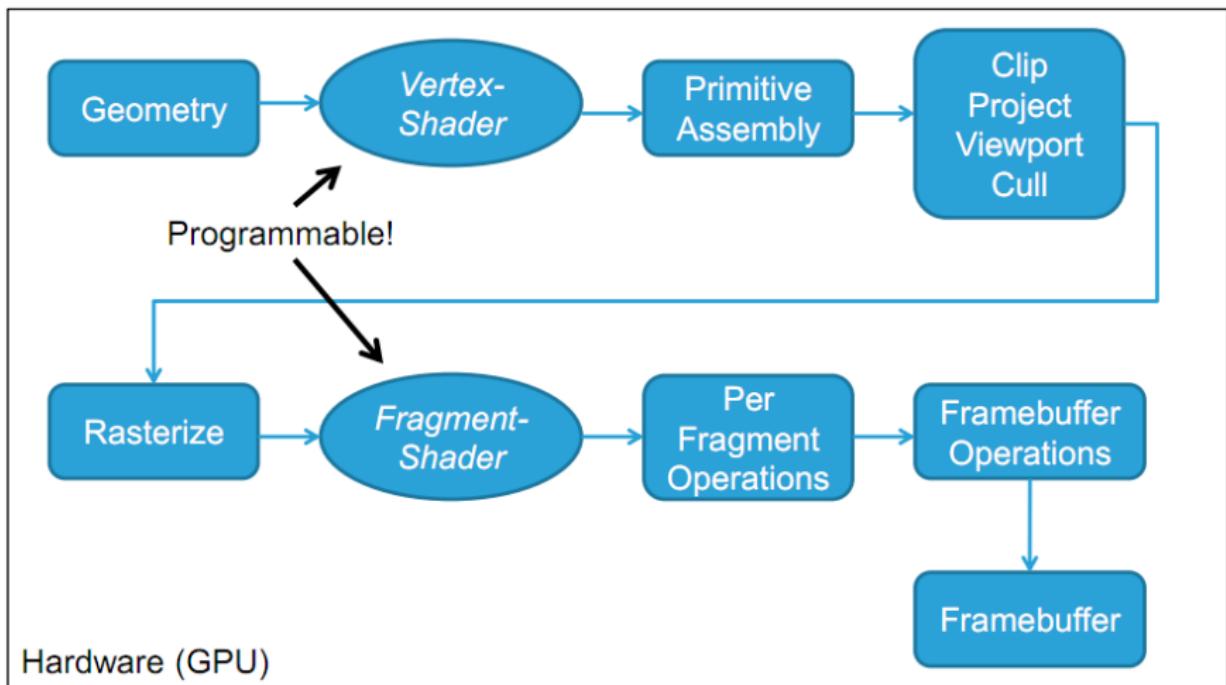
2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- **OpenGL**
- Grafikus gyorsítókártya generációk

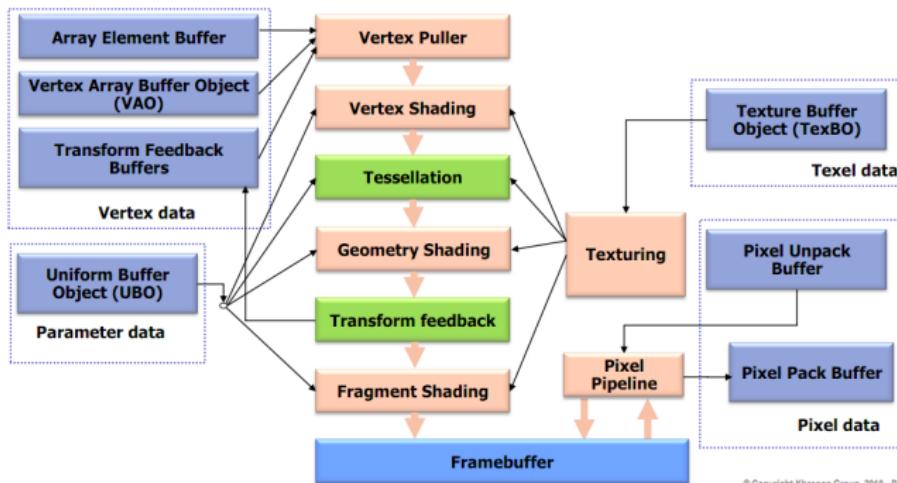
OpenGL pipeline

- Más elnevezésekkel, de nagyjából hasonlóak, mint a DX

OpenGL 3.x pipeline



OpenGL 4.1 pipeline



Grafikus gyorsítókártya generációk

Tartalom

1 Textúrázás

- Bevezetés
- Textúra leképezés
- Paraméterezés
- Textúra szűrés
- Procedurális textúrák
- Nem-szín textúrák

2 Inkrementális képszintézis a GPU-n

- Inkrementális képszintézis
- DirectX
- OpenGL
- **Grafikus gyorsítókártya generációk**

Grafikus gyorsítókártya generációk

1. Generáció

- NVIDIA TNT2, ATI Rage, 3dfx Voodoo3

Grafikus gyorsítókártya generációk

1. Generáció

- NVIDIA TNT2, ATI Rage, 3dfx Voodoo3
- A standard 2d-s videokártyák kiegészítése

Grafikus gyorsítókártya generációk

1. Generáció

- NVIDIA TNT2, ATI Rage, 3dfx Voodoo3
- A standard 2d-s videokártyák kiegészítése
- Csúcspont transzformációkat még a CPU csinálja!

Grafikus gyorsítókártya generációk

1. Generáció

- NVIDIA TNT2, ATI Rage, 3dfx Voodoo3
- A standard 2d-s videokártyák kiegészítése
- Csúcspont transzformációkat még a CPU csinálja!
- A kártya csak a textúrázást, Z-buffer kezelést végezte

2. Generáció (1999-2000)

- NVIDIA GeForce 256, GeForce 2, ATI Radeon 7500

Grafikus gyorsítókártya generációk

2. Generáció (1999-2000)

- NVIDIA GeForce 256, GeForce 2, ATI Radeon 7500
- Átveszik a transzformációk és az árnyalás kezelését a CPU-tól.

2. Generáció (1999-2000)

- NVIDIA GeForce 256, GeForce 2, ATI Radeon 7500
- Átveszik a transzformációk és az árnyalás kezelését a CPU-tól.
- Az OpenGL és DirectX 7 is támogatja a hardveres csúcspont transzformációkat

2. Generáció (1999-2000)

- NVIDIA GeForce 256, GeForce 2, ATI Radeon 7500
- Átveszik a transzformációk és az árnyalás kezelését a CPU-tól.
- Az OpenGL és DirectX 7 is támogatja a hardveres csúcspont transzformációkat
- Multi-textúrázás megjelenése: bump map, light map

2. Generáció (1999-2000)

- NVIDIA GeForce 256, GeForce 2, ATI Radeon 7500
- Átveszik a transzformációk és az árnyalás kezelését a CPU-tól.
- Az OpenGL és DirectX 7 is támogatja a hardveres csúcspont transzformációkat
- Multi-textúrázás megjelenése: bump map, light map
- Konfigurálható (driver szinten), de még nem programozható

Grafikus gyorsítókártya generációk

3. Generáció (2001)

- NVIDIA GeForce 3, GeForce 4 Ti, Xbox, ATI Radeon 8500

3. Generáció (2001)

- NVIDIA GeForce 3, GeForce 4 Ti, Xbox, ATI Radeon 8500
- A csúcspont pipeline korlátozott programozhatósága

3. Generáció (2001)

- NVIDIA GeForce 3, GeForce 4 Ti, Xbox, ATI Radeon 8500
- A csúcspont pipeline korlátozott programozhatósága
- Fejlettebb pixel szintű konfigurálás, de még nem programozás

3. Generáció (2001)

- NVIDIA GeForce 3, GeForce 4 Ti, Xbox, ATI Radeon 8500
- A csúcspont pipeline korlátozott programozhatósága
- Fejlettebb pixel szintű konfigurálás, de még nem programozás
- 3d-s textúrák, többszörös mintavételezés (antialias-hoz)

Grafikus gyorsítókártya generációk

4. Generáció (2002)

- NVIDIA GeForce FX, ATI Radeon 9700

Grafikus gyorsítókártya generációk

4. Generáció (2002)

- NVIDIA GeForce FX, ATI Radeon 9700
- A csúcspont és pixel pipeline teljesen programozható (erőforrás-korlátok azért még vannak)

Grafikus gyorsítókártya generációk

4. Generáció (2002)

- NVIDIA GeForce FX, ATI Radeon 9700
- A csúcspont és pixel pipeline teljesen programozható (erőforrás-korlátok azért még vannak)
- Magas szintű árnyaló nyelvek (shading languages) megjelenése (NVIDIA Cg, Microsoft HLSL, OpenGL GLSL)

Grafikus gyorsítókártya generációk

4. Generáció (2002)

- NVIDIA GeForce FX, ATI Radeon 9700
- A csúcspont és pixel pipeline teljesen programozható (erőforrás-korlátok azért még vannak)
- Magas szintű árnyaló nyelvek (shading languages) megjelenése (NVIDIA Cg, Microsoft HLSL, OpenGL GLSL)
- Shader Model 2.0 (simple branching)

Grafikus gyorsítókártya generációk

5. Generáció (2004)

- NVIDIA GeForce 6, ATI Radeon X, GeForce 7

Grafikus gyorsítókártya generációk

5. Generáció (2004)

- NVIDIA GeForce 6, ATI Radeon X, GeForce 7
- Több puffer szimultán renderelése

Grafikus gyorsítókártya generációk

5. Generáció (2004)

- NVIDIA GeForce 6, ATI Radeon X, GeForce 7
- Több puffer szimultán renderelése
- 64bites pipeline

Grafikus gyorsítókártya generációk

5. Generáció (2004)

- NVIDIA GeForce 6, ATI Radeon X, GeForce 7
- Több puffer szimultán renderelése
- 64 bites pipeline
- PCIe busz

Grafikus gyorsítókártya generációk

5. Generáció (2004)

- NVIDIA GeForce 6, ATI Radeon X, GeForce 7
- Több puffer szimultán renderelése
- 64 bites pipeline
- PCIe busz
- Több memória, hosszabb csúcspont árnyaló programok

Grafikus gyorsítókártya generációk

5. Generáció (2004)

- NVIDIA GeForce 6, ATI Radeon X, GeForce 7
- Több puffer szimultán renderelése
- 64 bites pipeline
- PCIe busz
- Több memória, hosszabb csúcspont árnyaló programok
- Shader Model 3.0 (branching and looping in the pixel shader (physics))

Grafikus gyorsítókártya generációk

5. Generáció (2004)

- NVIDIA GeForce 6, ATI Radeon X, GeForce 7
- Több puffer szimultán renderelése
- 64 bites pipeline
- PCIe busz
- Több memória, hosszabb csúcspont árnyaló programok
- Shader Model 3.0 (branching and looping in the pixel shader (physics))
- HDRI, SLI, TSAA, TMAA

Grafikus gyorsítókártya generációk

6. Generáció (2007)

- DirectX 10

Grafikus gyorsítókártya generációk

6. Generáció (2007)

- DirectX 10
- Shader Model 4.0 (Unified Shader Model, geometry shader)

Grafikus gyorsítókártya generációk

6. Generáció (2007)

- DirectX 10
- Shader Model 4.0 (Unified Shader Model, geometry shader)
- Unified Shading Architecture

Grafikus gyorsítókártya generációk

6. Generáció (2007)

- DirectX 10
- Shader Model 4.0 (Unified Shader Model, geometry shader)
- Unified Shading Architecture
- Shading performance 2x pixel, 12x vertex above G71

Grafikus gyorsítókártya generációk

6. Generáció (2007)

- DirectX 10
- Shader Model 4.0 (Unified Shader Model, geometry shader)
- Unified Shading Architecture
- Shading performance 2x pixel, 12x vertex above G71
- 700 Mtransistors

Grafikus gyorsítókártya generációk

6. Generáció (2007)

- DirectX 10
- Shader Model 4.0 (Unified Shader Model, geometry shader)
- Unified Shading Architecture
- Shading performance 2x pixel, 12x vertex above G71
- 700 Mtransistors
- 130W to 300W

Grafikus gyorsítókártya generációk

7. Generáció (2009-)

- DirectX 11, OpenGL 4.1

Grafikus gyorsítókártya generációk

7. Generáció (2009-)

- DirectX 11, OpenGL 4.1
- Shader Model 5.0 (Compute Shader, Tesselation Shaders: Hull Shader & Domain Shader)

Grafikus gyorsítókártya generációk

7. Generáció (2009-)

- DirectX 11, OpenGL 4.1
- Shader Model 5.0 (Compute Shader, Tesselation Shaders: Hull Shader & Domain Shader)
- Többszállúság

Grafikus gyorsítókártya generációk

7. Generáció (2009-)

- DirectX 11, OpenGL 4.1
- Shader Model 5.0 (Compute Shader, Tesselation Shaders: Hull Shader & Domain Shader)
- Többszállúság
- *Dynamic Shader Linkage*: OOP jellegű szolgáltatások HLSL-ben

Grafikus gyorsítókártya generációk

7. Generáció (2009-)

- DirectX 11, OpenGL 4.1
- Shader Model 5.0 (Compute Shader, Tesselation Shaders: Hull Shader & Domain Shader)
- Többszállúság
- *Dynamic Shader Linkage*: OOP jellegű szolgáltatások HLSL-ben
- 3000+ Mtransistors

Grafikus gyorsítókártya generációk

7. Generáció (2009-)

- DirectX 11, OpenGL 4.1
- Shader Model 5.0 (Compute Shader, Tesselation Shaders: Hull Shader & Domain Shader)
- Többszállúság
- *Dynamic Shader Linkage*: OOP jellegű szolgáltatások HLSL-ben
- 3000+ Mtransistors
- 1000 GFLOPs

Számítógépes Grafika

Hajder Levente
hajder@inf.elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2017/2018. II. félév

Tartalom

1 Animáció

2 Animálható tulajdonságok

3 Animáció típusok

4 Hierachikus rendszerek

Animáció

- Állókép helyett képsorozat
- Objektumok/kamera/világ tulajdonságait változatjuk
- Egy kép ≡ egy idő pillanat
- Képsorozat elég gyors → folyamatos mozgást érzékel az emberi szem

Mi az, amit változtatunk?

- Lehet: pozíció, orientáció, szín, normál, BRDF, stb.
- "Értelme" van: pozíció, orientáció, kamera
- Animációt alkalmazunk:
 - a modellezési transzformációra
 - a kamera transzformációra

Animáció szintézis

- Legyen minden o objetumra, a modellezési trafó:
 $M_o \in \mathbb{R}^{4 \times 4}$
- Legyen a kamera trafó: $V \in \mathbb{R}^{4 \times 4}$
- *Megjegyzés: ha V a View mátrix, akkor csak a kamera pozícióját, orientációját tartalmazza; ha V a View és a Projection együtt, akkor tudjuk vele a látószöget is állítani*
- Legyen mindkettő idő függő!
- $M_o \in \mathbb{R} \rightarrow \mathbb{R}^{4 \times 4}$
- $V \in \mathbb{R} \rightarrow \mathbb{R}^{4 \times 4}$

Általános animációs program váza

```
while keep_running:  
    t = get_time()  
    for o in objects:  
        Mo = Mo(t)  
        V = V(t)  
    render_scene()
```

Animáció szintézis

- Valósidejű/interaktív:

- rögtön meg is jelenítjük a képkockákat
- a számításnak elég gyorsnak kell lennie a folytonosság látszatához
- a felhasználói eseményekre ragálni kell
- ⇒ olyan részletgazdag lehet a színtér, amit még így meg lehet jeleníteni
- ⇒ inkrementális képszintézist használunk

Valósidejű animációs program váza

```
def update():
    # vagy idle , vagy onFrameMove
    t = get_time()
    for o in objects:
        Mo = Mo(t)
        V = V(t)

def render():
    # vagy display , vagy onFrameRender
    for o in objects:
        render_object(o)
```

Animáció szintézis

- Nem valós idejű/*offline*:

- "Nem számít", hogy mennyi ideig tart kiszámítani egy képkockát
- Elkülönöl a szintézis és a visszajátszás
 - Először elmentjük a képkockákat
 - Aztán majd videóként lehet visszanézni
- ⇒ a felhasználó nem tud belenyülni az animációba
- ⇒ olyan részletes színteret, és olyan bonyolult algoritmust használunk, amit ki tudunk várni

Offline rendering

```
 $\Delta t = 1/FPS$ 
for (t=t_start; t<t_end; t+= $\Delta t$ ):
    for o in objects:
         $M_o = M_o(t)$ 
         $V = V(t)$ 
        render_scene_to_disk()
```

Valósidejű animáció – rosszul

- Hogyan lehet a legkönnyebben elrontani az animáció számítást?
- Azzal, ha nem vesszük figyelembe, hogy mennyi idő telt el két képkocka között.
- Pl.: A középpontja körül akarjuk forgatni az objektumot állandó szögsebességgel.

```
model = rotation(phi, 0,1,0); phi +=  
phi_step;
```

- Az objektum olyan gyorsan fog forogni, amilyen gyakorisággal ez a kód részlet meghívódik.
- Gyorsabb gépen többször, lassabb gépen kevesebbszer.

Valósidejű animáció – jól

- Hogyal lehet a legkönnyebben ezt megelőzni?
- Sose azt tároljuk, hogy mennyivel kell változtatni, hanem, hogy mi a változás *sebessége*.
- minden számítás előtt kérjük le, hogy mennyi idő telt el az előző képkocka óta, és ezel szorozzuk a sebességet.
- Pl.: `phi += phi_step;` helyett `phi += get_time_since_last_frame() * phi_speed;`
- Gyorsabb gépen kevesebb, lassabb gépen több idő telik el két képkocka között ⇒ gyors gépen kiszebbeket lépünk, lassabban nagyobbakat.

Tartalom

1 Animáció

2 Animálható tulajdonságok

- Kamera animáció
- Pozíció és orientáció
- Roll, Pitch, Yaw

3 Animáció típusok

4 Hierachikus rendszerek

Kamera animáció

- A kamera tulajdonásgai:
 - szempozíció (*eye*),
 - egy pont amire néz (*center*),
 - felfele irányt megadó vektor (*up*),
 - a képernyő/ablak oldal aránya (*aspect*),
 - nyílásszög (*fovy*).
- Ezek mind külön-külön változtathatók az animáció létrehozásához.

Pozíció és orientáció

- Pozícó: "*Hol van az objektum?*"
- Orientáció: "*Hogy áll, merrefele néz az objektum?*"
- Elsősorban ezt a kettőt szerenénk változtatni.
- $M_o(t)$ megadja mindkettőt.
- Normális esetben

$$M_o(t) = \begin{bmatrix} A_{11} & A_{12} & A_{13} & 0 \\ A_{21} & A_{22} & A_{23} & 0 \\ A_{31} & A_{32} & A_{33} & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix},$$

- $\vec{p} = (p_x, p_y, p_z)$ a pozíció.
- Az **A** mátrix **tartalmazza** az orientációt.

Orientációs paraméterek

- Tegyük \vec{p} -t és \mathbf{A} -t is időfüggővé!
- \vec{p} tagjait leírhatjuk külön-külön függvénnyel.
- Pl.: *Valami esik, elég p_y -t változtatni.*
- \mathbf{A} -t tagjai összefügggenek, és csak az orientáció érdekel belőle minket (nem érdekel: méretezés, nyírás).
- Az orientáció megadható három tengely menti forgatással
→ három független függvénnyel.

Yaw, pitch, roll

- Egy objektum függőleges- (*yaw*), kereszt- (*pitch*) és hossztengelye (*roll*) menti elfordulásait egyszerre adjuk meg.
- Már találkoztunk vele, 3×3 -as mátrixszal megadható, három forgatás szorzata.
- Három tengely menti elfordulás szögével megadható \Rightarrow megadja az orientációt is.

Tartalom

1 Animáció

2 Animálható tulajdonságok

3 Animáció típusok

- Képlet animáció
- Kulcskocka animáció
- Pálya animáció

4 Hierachikus rendszerek

Képlet animáció

- Egy adott tulajdonság változását egy megfelelő függvénytel írjuk le.
- PI: Óra mutatói
 - Nagymutató: $yaw(t) = t/10$
 - Kismutató: $yaw(t) = t/120$
 - Ha t -t mp-ben adjuk meg, a forgatásokat pedig fokban.
- PI: Pattogó labda
 - $p_y(t) = |\sin(\omega t + \theta_0)| \cdot e^{-kt}$

Kulcskocka (*key frame*) animáció

- Egy bonyolult mozgást nehézkes lenne képlettel megadni.
- Inkább adjuk csak bizonyos időközönként, hogy *akkor* mit szeretnénk látni.
- Ezek a kulcskockák.
- Egy tulajdonságot két kulcskocka között *interpolációval* számolunk ki.

Kulcskocka (*key frame*) animáció

- Az interpolációval az objektum egyes paramétereire folytonos görbét illesztünk.
- Az animáció lejátszása/elmentése során a program minden képkockában a hozzá tartozó t értékkel kiértékeli az objektum paraméter-függvényeit.
- Ezekből számítja a transzformációs mátrixokat.
- A mátrixok felhasználásval előállíja a képet.

Lineáris interpoláció

- Legyen a két kulcskockánk időpontja t_0 és t_1 .
- Legyen az interpolálendő tulajdonság g .
- Lineráis interpolációval $\forall t \in [t_0, t_1]$ -re kapjuk

$$g(t) = \left(1 - \frac{t - t_0}{t_1 - t_0}\right) g(t_0) + \frac{t - t_0}{t_1 - t_0} g(t_1)$$

Interpoláció két kulcskocka között

K: Mi a baj a lineáris interpolációval?

V: Ritkán néz ki természetesen. Animáció során a sebesség konstans, előtte, utána nulla.

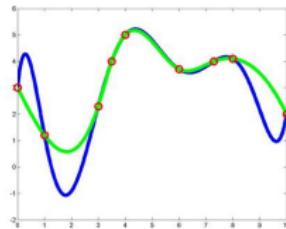
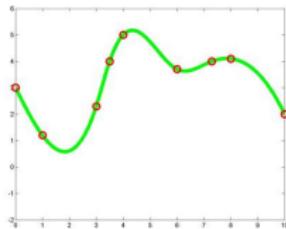
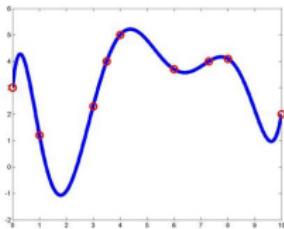
- Elgurított labda: folyamatosan lassul.
- Zuhanó zongora: folyamatosan gyorsul.
- Rakéta a Földről a Marsra: gyorul, halad, lassít.
- Ilyen interpolációra használhatunk:
 - Gyök függvényt.
 - Másodfokú függvényt.
 - Logisztikus függvényt/görbét.
- Gyakorlatban: Bézier görbe

Polinom interpoláció

- n kulcspontra fel tudunk írni $n - 1$ -ed fokú polinomot.
- Interpolációs polinom: minden kulcskockában az előírt értéket veszi fel.
- Együtthatók számíthatók Lagrange interpolációval.
- A lineáris interpoláció a Lagrange interpoláció speciális esete $n = 2$ -re.

Spline interpoláció

- A polinom interpolációval kapott fv. magas fokszám esetén a szomszédos pontok között "hullámzik", így elrontja az animációt.
- Spline interpoláció: használunk több, egymáshoz kapcsolódó, alacsony fokszámú polinomot az interpolációhoz!



Pálya animáció

- Egy objektum mozgását megadhatjuk a bejárandó pálya megadásával is.
- A pályát egy 3D görbével adjuk meg.
- A model ezen a görbén halad végig.

Orientáció megadása

- Hogyan adjuk meg az objetumunk orientációját?
- Egy *előre*, és egy *felfele* irány egyértelműen meghatározza ezt.
- *Megjegyzés:* v.ö. kamera esetén *center-eye ill. up* vektorok
- Ha a pályagörbe differenciálható, akkor az megadja a sebességvektort minden időpillanatban.
- A sebességvektor mindenkor előre fele mutat.

Orientáció megadása

- A *felfele* irány megadására két lehetőségünk is van.
- Ha van egy természetes *felfele*, akkor használjuk azt.
(Mindennél, ami nem ből be a kanyarban.)
- Ha ez az irány is változik, akkor ez megegyezik a gyorsulás irányával, azaz a pályagörbe második deriváltjának irányával.

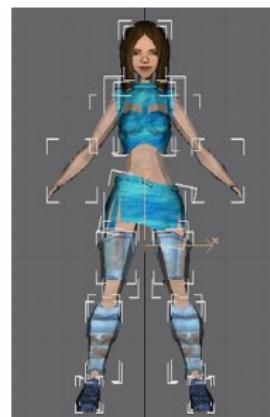
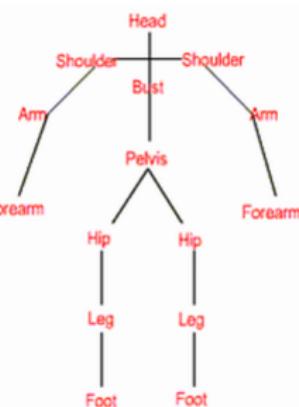
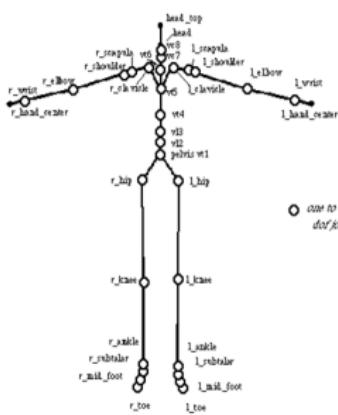
Tartalom

- 1 Animáció
- 2 Animálható tulajdonságok
- 3 Animáció típusok
- 4 Hierachikus rendszerek
 - Előrehaladó kinematika
 - Inverz kinematika

Hierachikus rendszerek

- Színtér gráfoknál már találkoztunk ilyenekkel.
- Egy gyerek objektum mozgását a szülőhöz viszonyítva adjuk meg.
- Gyerekeknek lehetnek újabb gyerekei, s.i.t.
- Hierachikus rendszert – fát – kapunk.

Példa: Emberi test



Kényszerek (*constraints*)

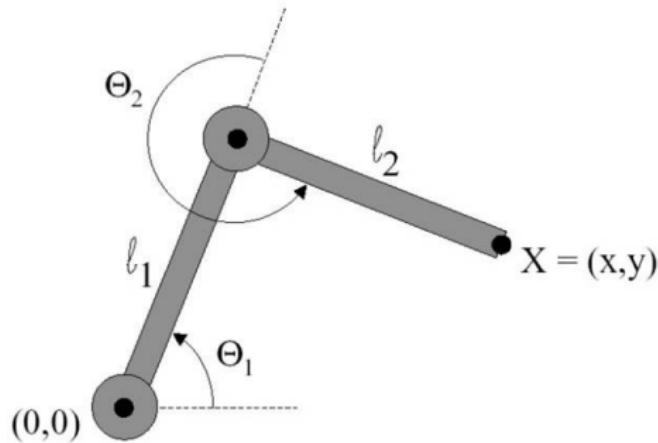
- Nem minden mozgást szeretnénk megengedni a szülőhöz képest.
- Ezeket a megszorításokat írhatjuk le *kényszerekkel*.
- Korlátozhatjuk a szabadságfokokat: pl. könyök csak egy tengely mentén tud forogni, de a csukló kettő
- Vagy a tartományokat: kevesen bírják, ha a fejük 90° -nél többet fordul.

Előrehaladó kinematika

- Végállapotot határozunk meg az állapotváltozók függvényében.
- Szimulációhoz jól használható.
- minden elemre megadjuk, a hozzá tartozó transzformációt.
- Ezeket a hierarchiában felülről lefele haladva értékeljük ki.
- Az adott elemhez tartozó transzformáció az összes ős és a saját transzformáció szorzata.

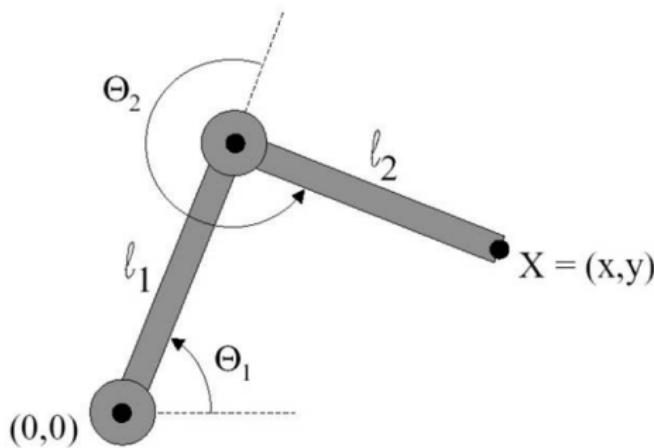
Példa

- Kétszabadságfokú, rotációs csuklókat tartalmazó rendszer.
- A csuklók csak a Z tengely körül fordulnak.



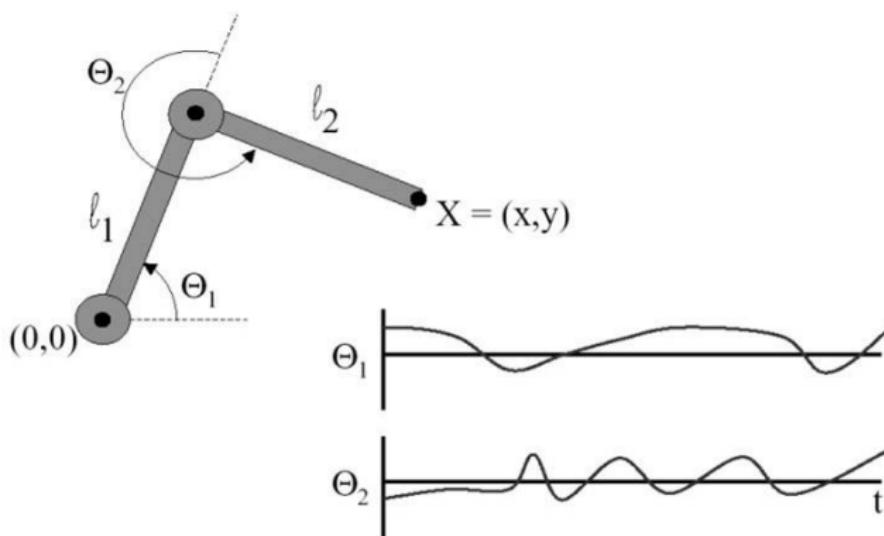
Példa – folyt.

- Állapotváltozók: Θ_1, Θ_2
- A végberendezés (X) pozícióját a gép számolja.
- $X = (l_1 \cos \Theta_1 + l_2 \cos(\Theta_1 + \Theta_2), l_1 \sin \Theta_1 + l_2 \sin(\Theta_1 + \Theta_2))$



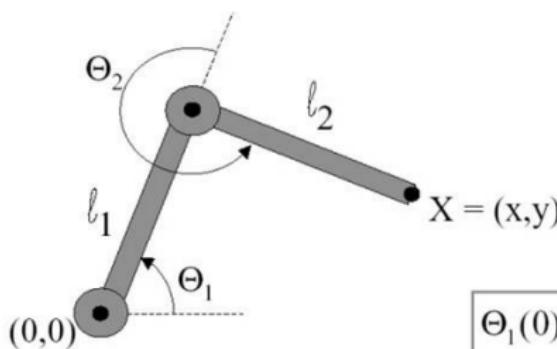
Példa – folyt.

- Az állapotváltozókat megadhatjuk (pl. spline) függvénnnyel.



Példa – folyt.

- Az állapotváltozókat megadhatjuk kezdeti értékkel és sebességgel.



$$\Theta_1(0) = 60^\circ \quad \Theta_2(0) = 250^\circ$$

$$\frac{d\Theta_1}{dt} = 1.2 \quad \frac{d\Theta_2}{dt} = -0.1$$

Mit nem tud az előrehaladó kinematika?

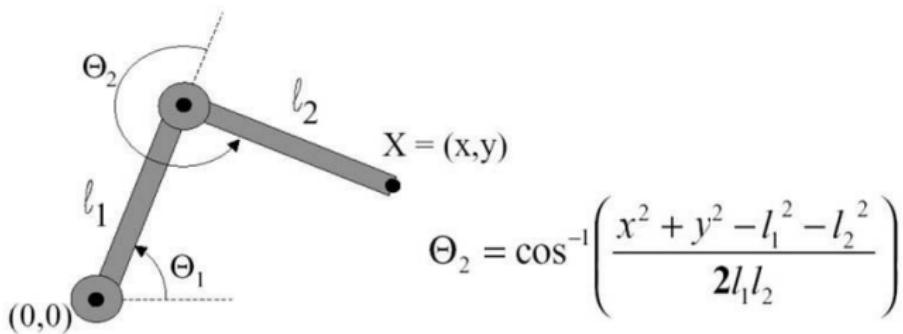
- Az előrehaladó kinematika nem használható, ha a strukturális összefüggés erősen nem lineáris
- Hiába interpolálunk egyenletesen az állapottérben, a végberendezés vadul kalimpálhat a kulcspontok között
- Problémás esetek:
 - Láb mozgása a talajon
 - Végállapot jó, de menet közben a berendezés részei átmehetnek egymáson.

Inverz kinematika

- Az inverz kinematika a kritikus végberendezés helyzetét interpolálja, majd az állapotváltozók értékét végberendezés interpolált helyzetéből számítja vissza.
- Az inverz kinematika másik neve a cél-orientált animáció.
- *"Ezt szeretném megfogni, hogyan forgassam az izületeimet?"*

Példa

- A végberendezés helyzetéből visszaszámoljuk az állapotváltozók értékét.

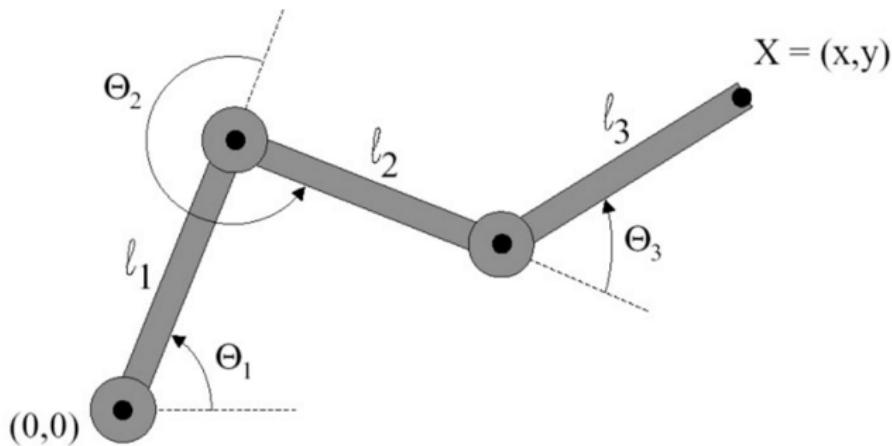


Problémák

- Nehéz "természetesnek látszó" mozgást leírni vele.
- Az inverz függvény kiszámítása nem triviális,
- és nem is egyértelmű (redundancia).

Példa

- Egyenletek száma: 2, ismeretlen változók száma: 3 \Rightarrow Végtelen sok megoldás!
- Rendszer DOF > végberendezés DOF
- Az emberi csontváz kb 70 DOF!



Példa

Nem egyértelmű, ill. nem létező megoldás

