

Számítógépes Grafika

9. előadás: árnyalás, textúrák, GPU

Bán Róbert

`rob.ban@inf.elte.hu`

Eötvös Loránd Tudományegyetem
Informatikai Kar

2025-2026. tavaszi félév

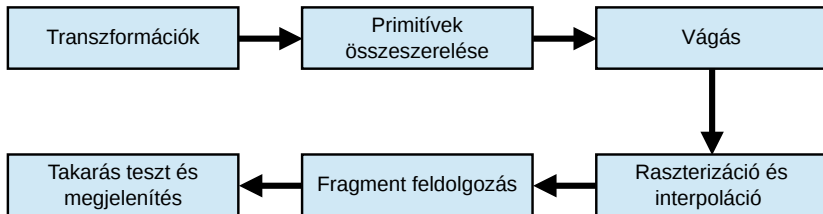
Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Grafikus szerelősorozat



Fragmentek feldolgozása

- Ebben a lépésben kell meghatározni a fragmentek színét

Fragmentek feldolgozása

- Ebben a lépésben kell meghatározni a fragmentek színét
- Most megnézzük, hogy miképp tehetjük ezt a fény-anyag kölcsönhatások modellezésével

Fragmentek feldolgozása

- Ebben a lépésben kell meghatározni a fragmentek színét
- Most megnézzük, hogy miképp tehetjük ezt a fény-anyag kölcsönhatások modellezésével
- Illetve megvizsgáljuk, hogy mire használható a textúrázás

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Lokális illumináció

- Valamilyen egyszerűsített megvilágítási modell szerint határozzuk meg a fragment színét
- Ehhez szükségünk van

Lokális illumináció

- Valamilyen egyszerűsített megvilágítási modell szerint határozzuk meg a fragment színét
- Ehhez szükségünk van
 - absztrakt fényforrásmodellekre

Lokális illumináció

- Valamilyen egyszerűsített megvilágítási modell szerint határozzuk meg a fragment színét
- Ehhez szükségünk van
 - absztrakt fényforrásmodellekre
 - absztrakt anyagmodellekre

Lokális illumináció

- Valamilyen egyszerűsített megvilágítási modell szerint határozzuk meg a fragment színét
- Ehhez szükségünk van
 - absztrakt fényforrásmodellekre
 - absztrakt anyagmodellekre
- Ezek kombinációjával közelítjük majd a kívánt megvilágítást

Tartalom

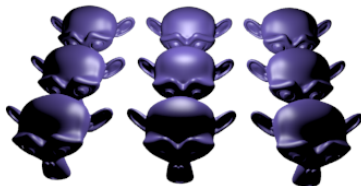
- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Fényforrás modellek



Írány fényforrás

Fényforrás modellek



Pont fényforrás

Fényforrás modellek



Spot fényforrás

Absztrakt fényforrások

- Irányfényforrás
 - csak iránya van
 - egyetlen (világ k.r.-beli) vektor megadja
 - a fénysugarakat párhuzamosnak tekintjük
 - távoli fényforrások megadására használható

Absztrakt fényforrások

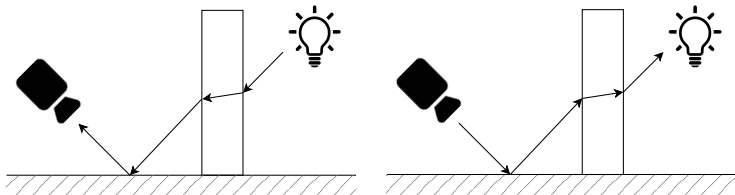
- Irányfényforrás
 - csak iránya van
 - egyetlen (világ k.r.-beli) vektor megadja
 - a fénysugarakat párhuzamosnak tekintjük
 - távoli fényforrások megadására használható
- Pont fényforrás
 - csak pozíciója van
 - egy ponttal adjuk meg, világ k.r.-ben
 - Pl.: villanykörte

Absztrakt fényforrások

- Irányfényforrás
 - csak iránya van
 - egyetlen (világ k.r.-beli) vektor megadja
 - a fénysugarakat párhuzamosnak tekintjük
 - távoli fényforrások megadására használható
- Pont fényforrás
 - csak pozíciója van
 - egy ponttal adjuk meg, világ k.r.-ben
 - Pl.: villanykörte
- *Spot* fényforrás
 - kúp: csúcspozíciója, tengelyiránya és „fényköre” van
 - egy pont és egy vektor (világ k.r.!) és két szög adja meg
 - Pl.: asztali lámpa

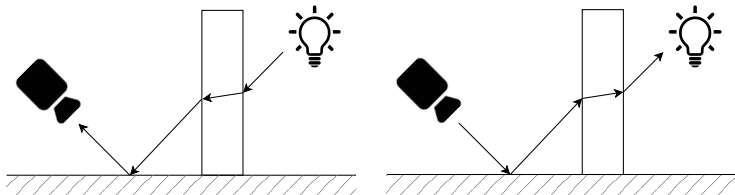
Helmholtz-törvény

- Helmholtz-féle szimmetria: a fénysugár megfordítható



Helmholtz-törvény

- Helmholtz-féle szimmetria: a fénysugár megfordítható
- Ez két dolog miatt jó nekünk:
 - Garantálja, hogy végső soron a radiancia csökken.
 - Nézzhetjük „visszafelé” a sugarakat.



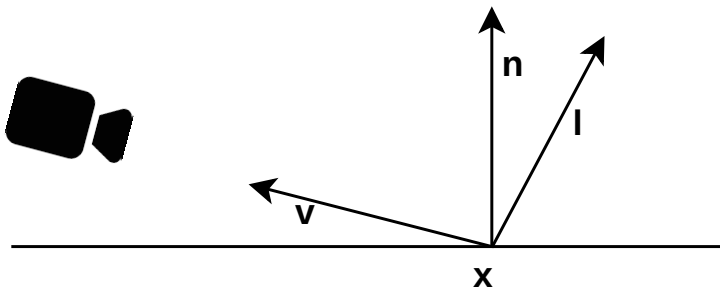
Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - **Anyagmodellek**
 - Megvalósítás
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

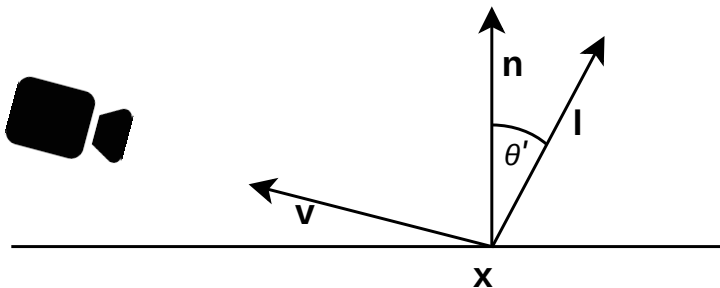
Megvilágítási modellek – jelölések

- $\mathbf{v} := \omega$ a nézőpont felé mutató vektor
- $\mathbf{l} := \omega'$ a megvilágító, a fényt „adó” pont felé mutató vektor
- $\mathbf{n}$ a felületi normális
- $\mathbf{r}$ az ideális visszaverődés iránya az $\mathbf{l}$ beesési irányból, $\mathbf{n}$ normális mellett
- $\mathbf{v}, \mathbf{l}, \mathbf{n}, \mathbf{r}$ egységvektorok
- θ' az $\mathbf{l}$ és $\mathbf{n}$ által bezárt szög
- ϕ az $\mathbf{r}$ és $\mathbf{v}$ által bezárt szög

Megvilágítási modellek – jelölések



Megvilágítási modellek – jelölések



BRDF

- A *kétirányú visszaverődéses eloszlási függvény* (BRDF – bi-directional reflection distribution function) segítségével modellezzük a fény-anyag kölcsönhatásokat

BRDF

- A *kétirányú visszaverődéses eloszlási függvény* (BRDF – bi-directional reflection distribution function) segítségével modellezzük a fény-anyag kölcsönhatásokat
- Legyen L^{in} egy adott irányból a felület egy pontjára beérkező, L pedig az onnan visszavert fény intenzitása (igazából: radianciája, azaz egységi felületből egységnyi térszögbe kibocsátott teljesítmény)

BRDF

- A *kétirányú visszaverődéses eloszlási függvény* (BRDF – bi-directional reflection distribution function) segítségével modellezzük a fény-anyag kölcsönhatásokat
- Legyen L^{in} egy adott irányból a felület egy pontjára beérkező, L pedig az onnan visszavert fény intenzitása (igazából: radianciája, azaz egységi felületből egységnyi térszögbe kibocsátott teljesítmény)
- A fenti jelölések mellett a BRDF a következő:

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

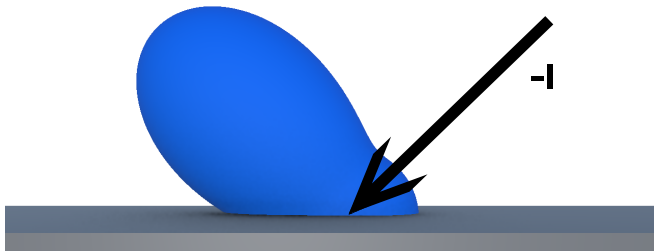
BRDF

- A *kétirányú visszaverődéses eloszlási függvény* (BRDF – bi-directional reflection distribution function) segítségével modellezzük a fény-anyag kölcsönhatásokat
- Legyen L^{in} egy adott irányból a felület egy pontjára beérkező, L pedig az onnan visszavert fény intenzitása (igazából: radianciája, azaz egységi felületből egységnyi térszögbe kibocsátott teljesítmény)
- A fenti jelölések mellett a BRDF a következő:

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

- Megjegyzés: a $\cos \theta'$ azonos a globális illuminációs képletben (6. előadás) is szereplő mennyiséggel ($-\omega_\ell \cdot \mathbf{n}$).

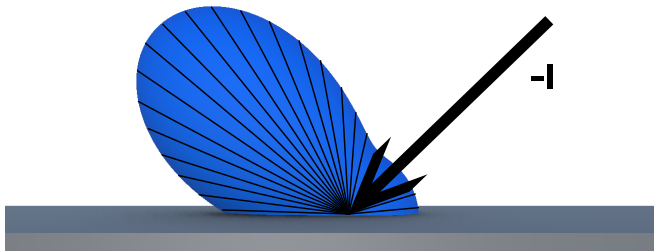
BRDF példa



Példa BRDF-re egy adott pontban az adott beérkezési iránnyal

- A függvény értékét a kék felület ábrázolja
- Minél messzebb van egy pont a beérkezési ponttól, annál valószínűbb, hogy arra fog a fény továbbhaladni
- Itt például a legnagyobb valószínűséggel az ideális tükörirány rendelkezik

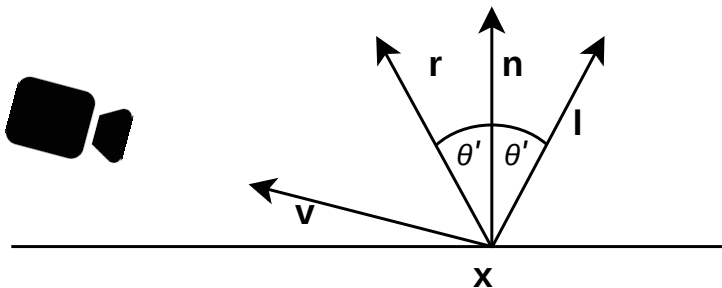
BRDF példa



Példa BRDF-re egy adott pontban az adott beérkezési iránnyal

- A függvény értékét a kék felület ábrázolja
- Minél messzebb van egy pont a beérkezési ponttól, annál valószínűbb, hogy arra fog a fény továbbhaladni
- Itt például a legnagyobb valószínűséggel az ideális tükörirány rendelkezik

Ideális visszaverődés



Ideális visszaverődés

- Az ideális tükör csak az $\mathbf{r}$ tükörirányba ver vissza. (ld. 4. EA)

Ideális visszaverődés

- Az ideális tükör csak az $\mathbf{r}$ tükörirányba ver vissza. (ld. 4. EA)



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

Ideális visszaverődés

- Az ideális tükör csak az $\mathbf{r}$ tükörirányba ver vissza. (ld. 4. EA)



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

- δ a *Dirac-delta* függvény, ami egy általánosított függvény, amely minden nemnulla paraméterre nullát ad, de a valós számok felett vett integrálja 1.

Ideális visszaverődés

- Az ideális tükör csak az $\mathbf{r}$ tükörirányba ver vissza. (ld. 4. EA)



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

- δ a *Dirac-delta* függvény, ami egy általánosított függvény, amely minden nemnulla paraméterre nullát ad, de a valós számok felett vett integrálja 1.
- A k_r visszaverődési együttható a *Fresnel-együttható*. Ez függ az anyag törésmutatójától és az elektromos vezetési képességétől.

Ideális visszaverődés

- Az ideális tükör csak az $\mathbf{r}$ tükörirányba ver vissza. (ld. 4. EA)

-

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

- δ a *Dirac-delta* függvény, ami egy általánosított függvény, amely minden nemnulla paraméterre nullát ad, de a valós számok felett vett integrálja 1.
- A k_r visszaverődési együttható a *Fresnel-együttható*. Ez függ az anyag törésmutatójától és az elektromos vezetési képességétől.
- A *Fresnel-együttható* a visszavert és beeső energia hányadát fejezi ki. Számolható fizikai mennyiség.

Ideális törés

- Jelölje $\mathbf{t}$ az ideális törési irányt. (ld. 4. EA)

Ideális törés

- Jelölje $\mathbf{t}$ az ideális törési irányt. (ld. 4. EA)
- Az ideális tükrökhöz hasonlóan kapjuk:

Ideális törés

- Jelölje $\mathbf{t}$ az ideális törési irányt. (ld. 4. EA)
- 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. (ld. 4. EA)
- Az ideális tükörhöz hasonlóan kapjuk:



$$f_r(\mathbf{x}, \mathbf{v}, \mathbf{l}) = k_t \frac{\delta(\mathbf{t} - \mathbf{v})}{\cos \theta'}$$

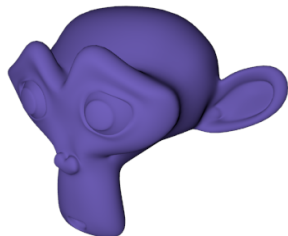
- k_t a törési *Fresnel-együttható*.

Lambert-törvény

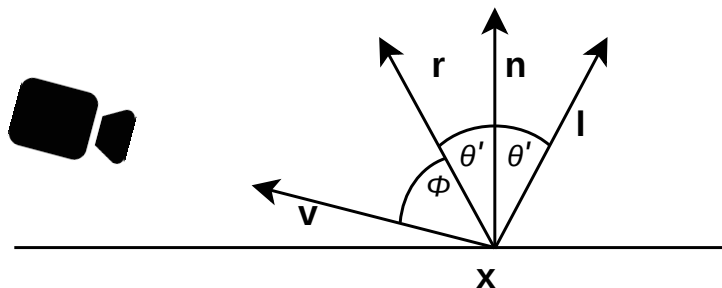
- Optikailag durva, *diffúz* felületek leírására jó.
- Feltételezés: a visszavert fény mennyisége nem függ a nézeti iránytól.
- Helmholtz-törvényt miatt akkor a bejövő iránytól sem függhet, azaz konstans:

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

- Csak ezt nézve: $L_{ref} = L_i k_d \cos^+ \theta'$



Spekuláris visszaverődés – Phong modell



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!

- Csak ezt nézve: $L_{ref} = L_i k_s (\cos^+ \phi)^n$

Spekuláris visszaverődés – Phong modell



$n = 5$

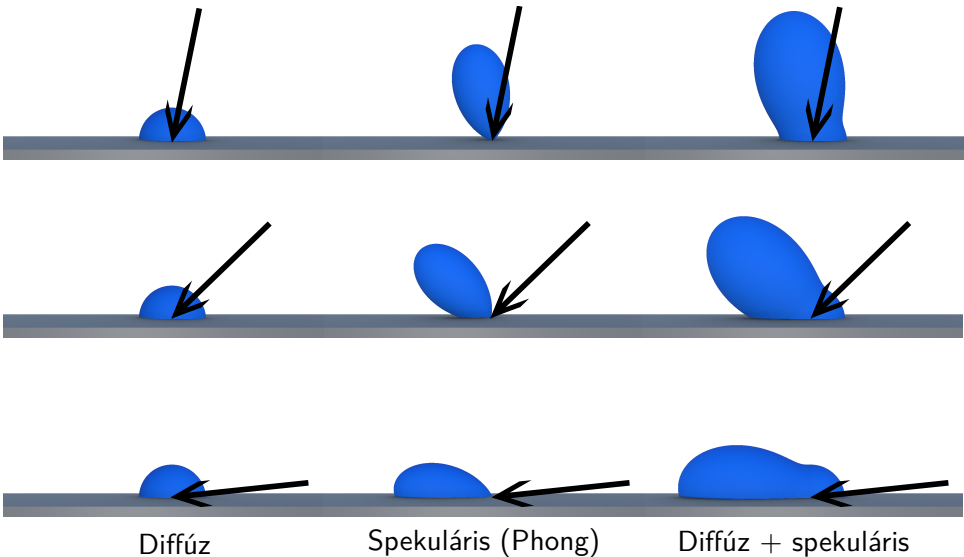


$n = 25$

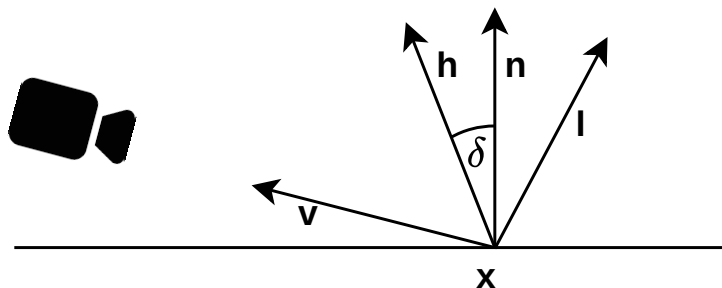


$n = 50$

BRDF – diffúz és spekuláris komponens



Spekuláris visszaverődés – Phong-Blinn modell



Spekuláris visszaverődés – Phong-Blinn modell

- Legyen $\mathbf{h}$ a nézeti irány és a megvilágító pont felé 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 $\cos \delta = \mathbf{h} \cdot \mathbf{n}$

-

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

- Csak ezt nézve: $L_{ref} = L_i k_s (\cos^+ \delta)^n$

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - **Megvalósítás**
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Megvilágítási modellek megvalósítása

- A korábbi képletek a fény egyetlen hullámhosszának viselkedését írják le.

Megvilágítási modellek megvalósítása

- A korábbi képletek a fény egyetlen hullámhosszának viselkedését írják le.
- A teljes spektrumot kellene leírunk, de egyszerűsítünk: csak az RGB hullámhosszokkal számolunk.

Megvilágítási modellek megvalósítása

- A korábbi képletek a fény egyetlen hullámhosszának viselkedését írják le.
- A teljes spektrumot kellene leírunk, de egyszerűsítünk: csak az RGB hullámhosszokkal számolunk.
- A legtöbb felület nem feleltethető meg egyetlen modellnek, több modellel együttesen írjuk le az anyag és a fény kölcsönhatását.

Megvilágítási modellek megvalósítása

- A korábbi képletek a fény egyetlen hullámhosszának viselkedését írják le.
- A teljes spektrumot kellene leírnunk, de egyszerűsítünk: csak az RGB hullámhosszokkal számolunk.
- A legtöbb felület nem feleltethető meg egyetlen modellnek, több modellel együttesen írjuk le az anyag és a fény kölcsönhatását.
- A különböző modellekből származó fénymennyiséget *összegezzük*.

Megvilágítási modellek megvalósítása

- A korábbi képletek a fény egyetlen hullámhosszának viselkedését írják le.
- A teljes spektrumot kellene leírnunk, de egyszerűsítünk: csak az RGB hullámhosszokkal számolunk.
- A legtöbb felület nem feleltethető meg egyetlen modellnek, több modellel együttesen írjuk le az anyag és a fény kölcsönhatását.
- A különböző modellekből származó fénymennyiséget összegezzük.
- Mivel csak lokális illuminációt számítunk, a többszörös visszaverődésekből származó fényt elveszítjük. Ezt külön pótoljuk (*ambiens fény*).

Ambiens tag

- A színtéren mindenütt jelenlevő fény mennyiség.

Ambiens tag

- A színtéren mindenütt jelenlevő fény mennyiség.
- Képlet: $k_a \cdot L_a$, ahol „ $\cdot$ ” most a koordinátánkénti szorzás:
 $[a, b, c] \cdot [x, y, z] = [ax, by, cz]$

Ambiens tag

- A szintéren mindenütt jelenlevő fénymennyiség.
- Képlet: $k_a \cdot L_a$, ahol „ $\cdot$ ” most a koordinátánkénti szorzás:
 $[a, b, c] \cdot [x, y, z] = [ax, by, cz]$
- k_a a felülettől függ, legyen
`vec3 ambientColor`

Ambiens tag

- A szintéren mindenütt jelenlevő fény mennyiség.
- Képlet: $k_a \cdot L_a$, ahol „ $\cdot$ ” most a koordinátánkénti szorzás:
 $[a, b, c] \cdot [x, y, z] = [ax, by, cz]$
- k_a a felülettől függ, legyen
`vec3 ambientColor`
- Itt a k_a számhármast az határozza meg, hogy a beeső fényintenzitás hányad részét veri vissza az anyag az R, G, B-nek megfelelő hullámhosszokon. k_a a *felület tulajdonsága*.

Ambiens tag

- A szintéren mindenütt jelenlevő fény mennyiség.
- Képlet: $k_a \cdot L_a$, ahol „ $\cdot$ ” most a koordinátánkénti szorzás:
 $[a, b, c] \cdot [x, y, z] = [ax, by, cz]$
- k_a a felülettől függ, legyen
`vec3 ambientColor`
- Itt a k_a számhármass azt határozza meg, hogy a beeső fényintenzitás hányad részét veri vissza az anyag az R, G, B-nek megfelelő hullámhosszokon. k_a a *felület tulajdonsága*.
- L_a fényforrástól, felülettől független, az állandó háttér-megvilágítás intenzitása az RGB hullámhosszokon. L_a a *színtér tulajdonsága*.

Ambiens tag

- A szintéren mindenütt jelenlevő fény mennyiség.
- Képlet: $k_a \cdot L_a$, ahol „ $\cdot$ ” most a koordinátánkénti szorzás:
 $[a, b, c] \cdot [x, y, z] = [ax, by, cz]$
- k_a a felülettől függ, legyen
`vec3 ambientColor`
- Itt a k_a számhármass az határozza meg, hogy a beeső fényintenzitás hányad részét veri vissza az anyag az R, G, B-nek megfelelő hullámhosszokon. k_a a *felület tulajdonsága*.
- L_a fényforrástól, felülettől független, az állandó háttér-megvilágítás intenzitása az RGB hullámhosszokon. L_a a *színtér tulajdonsága*.
- *Fragment shaderben* (DirectX-ben pixel shader) használható:
`ambientColor * ambientLight`

Diffúz komponens – Lambert-törvény

- BRDF alapján: $L_{ref} = L_i k_d \cos^+ \theta'$

Diffúz komponens – Lambert-törvény

- BRDF alapján: $L_{ref} = L_i k_d \cos^+ \theta'$
- Ezt neveztük *diffúz színnek*.

Diffúz komponens – Lambert-törvény

- BRDF alapján: $L_{ref} = L_i k_d \cos^+ \theta'$
- Ezt neveztük *diffúz színnek*.
- k_d és L_i mint az előbb, de L_i az aktuális *fényforrás tulajdonsága*:
 `vec3 diffuseColor`
 `vec3 diffuseLight`

Diffúz komponens – Lambert-törvény

- BRDF alapján: $L_{ref} = L_i k_d \cos^+ \theta'$
- Ezt neveztük *diffúz színnek*.
- k_d és L_i mint az előbb, de L_i az aktuális *fényforrás tulajdonsága*:
 `vec3 diffuseColor`
 `vec3 diffuseLight`
- Kell még: $\cos^+ \theta'$. ($\cos^+$: ahol negatív a cos, ott 0.)

Diffúz komponens – Lambert-törvény

- BRDF alapján: $L_{ref} = L_i k_d \cos^+ \theta'$
- Ezt neveztük *diffúz színnek*.
- k_d és L_i mint az előbb, de L_i az aktuális *fényforrás tulajdonsága*:
 `vec3` diffuseColor
 `vec3` diffuseLight
- Kell még: $\cos^+ \theta'$. ($\cos^+$: ahol negatív a $\cos$, ott 0.)
- Kiszámítása: `clamp(dot(normal, toLight), 0, 1)`

Diffúz komponens – Lambert-törvény

- BRDF alapján: $L_{ref} = L_i k_d \cos^+ \theta'$
- Ezt neveztük *diffúz színnek*.
- k_d és L_i mint az előbb, de L_i az aktuális *fényforrás tulajdonsága*:
 `vec3 diffuseColor`
 `vec3 diffuseLight`
- Kell még: $\cos^+ \theta'$. ($\cos^+$: ahol negatív a $\cos$, ott 0.)
- Kiszámítása: `clamp(dot(normal, toLight), 0, 1)`
- Ismerni kell hozzá `normal = n`-t és `toLight = l`-t.

Diffúz komponens – Lambert-törvény

- BRDF alapján: $L_{ref} = L_i k_d \cos^+ \theta'$
- Ezt neveztük *diffúz színnek*.
- k_d és L_i mint az előbb, de L_i az aktuális *fényforrás tulajdonsága*:
 `vec3 diffuseColor`
 `vec3 diffuseLight`
- Kell még: $\cos^+ \theta'$. ($\cos^+$: ahol negatív a $\cos$, ott 0.)
- Kiszámítása: `clamp(dot(normal, toLight), 0, 1)`
- Ismerni kell hozzá `normal = n`-t és `toLight = l`-t.

- $$\text{clamp}(x, a, b) = \begin{cases} a & , \text{ha } x < a \\ x & , \text{ha } x \in [a, b] \\ b & , \text{ha } x > b \end{cases}$$

Diffúz komponens – Lambert-törvény

- BRDF alapján: $L_{ref} = L_i k_d \cos^+ \theta'$
- Ezt neveztük *diffúz színnek*.
- k_d és L_i mint az előbb, de L_i az aktuális *fényforrás tulajdonsága*:
 `vec3 diffuseColor`
 `vec3 diffuseLight`
- Kell még: $\cos^+ \theta'$. ($\cos^+$: ahol negatív a $\cos$, ott 0.)
- Kiszámítása: `clamp(dot(normal, toLight), 0, 1)`
- Ismerni kell hozzá `normal = n`-t és `toLight = l`-t.

- $$\text{clamp}(x, a, b) = \begin{cases} a & , \text{ha } x < a \\ x & , \text{ha } x \in [a, b] \\ b & , \text{ha } x > b \end{cases}$$

- *Spot* fényforrásnál a fénykört is figyelembe kell majd venni.

Spekuláris visszaverődés – Phong modell

- BRDF alapján: $L_{ref} = L_i k_s (\cos^+ \phi)^n$, ahol ϕ az $\mathbf{r}$ tükörirány és a $\mathbf{v}$ nézeti irány által bezárt szög.

Spekuláris visszaverődés – Phong modell

- BRDF alapján: $L_{ref} = L_i k_s (\cos^+ \phi)^n$, ahol ϕ az $\mathbf{r}$ tükörirány és a $\mathbf{v}$ nézeti irány által bezárt szög.
- k_d és L_i (ez egy másik L_i) megint mint az előbb, L_i szintén az aktuális *fényforrás tulajdonsága*:

`vec3 specularColor`

`vec3 specularLight`

Spekuláris visszaverődés – Phong modell

- BRDF alapján: $L_{ref} = L_i k_s (\cos^+ \phi)^n$, ahol ϕ az $\mathbf{r}$ tükörirány és a $\mathbf{v}$ nézeti irány által bezárt szög.
- k_d és L_i (ez egy másik L_i) megint mint az előbb, L_i szintén az aktuális *fényforrás tulajdonsága*:
`vec3 specularColor`
`vec3 specularLight`
- n felület függő konstans, legyen
`float specularPower`

Spekuláris visszaverődés – Phong modell

$L_{ref} = L_i k_s (\cos^+ \phi)^n$; $\mathbf{r}$ tükörirány és a $\mathbf{v}$ nézeti irány.

- Kell $\cos^+ \phi$, amihez szükséges $\mathbf{r}$ és $\mathbf{v}$.

Spekuláris visszaverődés – Phong modell

$L_{ref} = L_i k_s (\cos^+ \phi)^n$; $\mathbf{r}$ tükörirány és a $\mathbf{v}$ nézeti irány.

- Kell $\cos^+ \phi$, amihez szükséges $\mathbf{r}$ és $\mathbf{v}$.
- $\mathbf{r}$ a $\mathbf{l}$ vektor $\mathbf{n}$ -re vett tükörképe. Számítása:
`vec3 refl = reflect(-toLight, normal)`

Spekuláris visszaverődés – Phong modell

$L_{ref} = L_i k_s (\cos^+ \phi)^n$; $\mathbf{r}$ tükörirány és a $\mathbf{v}$ nézeti irány.

- Kell $\cos^+ \phi$, amihez szükséges $\mathbf{r}$ és $\mathbf{v}$.
- $\mathbf{r}$ a $\mathbf{l}$ vektor $\mathbf{n}$ -re vett tükörképe. Számítása:
`vec3 refl = reflect(-toLight, normal)`
- $\mathbf{v}$ a nézeti irány, azaz a felületi pontból a kamerába mutató egységvektor.
`vec3 toEye = normalize(eyePosition - worldPos)`

Spekuláris visszaverődés – Phong modell

$L_{ref} = L_i k_s (\cos^+ \phi)^n$; $\mathbf{r}$ tükörirány és a $\mathbf{v}$ nézeti irány.

- Kell $\cos^+ \phi$, amihez szükséges $\mathbf{r}$ és $\mathbf{v}$.
- $\mathbf{r}$ a $\mathbf{l}$ vektor $\mathbf{n}$ -re vett tükörképe. Számítása:
`vec3 refl = reflect(-toLight, normal)`
- $\mathbf{v}$ a nézeti irány, azaz a felületi pontból a kamerába mutató egységvektor.
`vec3 toEye = normalize(eyePosition - worldPos)`
- $(\cos^+ \phi)^n$ számítása:
`pow(clamp(dot(refl, toEye), 0, 1), specularPower)`

toLight számítása

- Irány fényforrás
 - Fény iránya, normalizált irányvektor: `vec3 lightDirection`
 - `toLight = -lightDirection`

toLight számítása

- Irány fényforrás
 - Fény iránya, normalizált irányvektor: `vec3 lightDirection`
 - `toLight = -lightDirection`
- Pont fényforrás
 - Fény pozíciója, helyvektor: `vec3 lightPosition`
 - `toLight = normalize(lightPosition - worldPos)`

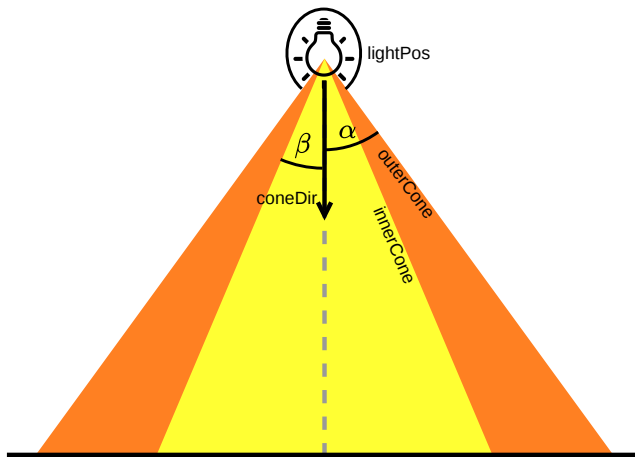
toLight számítása

- Irány fényforrás
 - Fény iránya, normalizált irányvektor: `vec3 lightDirection`
 - `toLight = -lightDirection`
- Pont fényforrás
 - Fény pozíciója, helyvektor: `vec3 lightPosition`
 - `toLight = normalize(lightPosition - worldPos)`
- *Spot* fényforrás
 - Kúp tengelyiránya, normalizált irányvektor: `vec3 coneDirection`
 - Csúcs pozíciója, helyvektor: `vec3 lightPosition`
 - `toLight = normalize(lightPosition - worldPos)`
mint pont fényforrás esetén

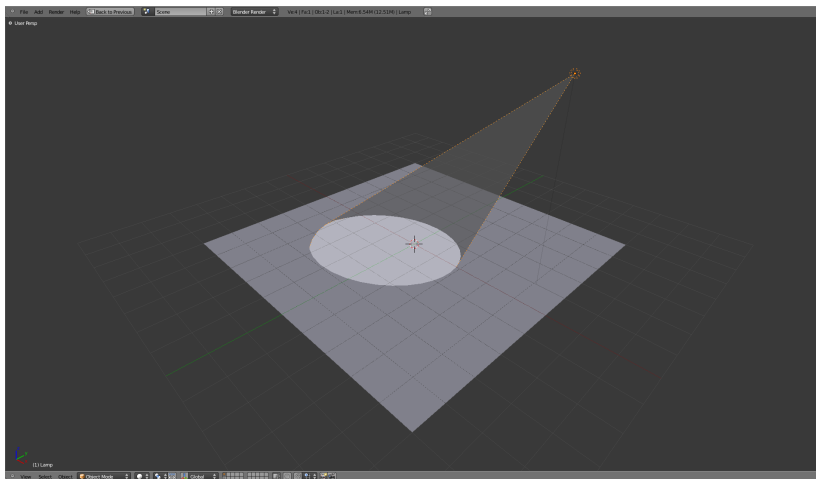
Spot fényforrás hatása

- Két extra paraméter:
 - belső fénykör: amin belül teljes intenzitással hat
 - külső fénykör: amin kívül abszolút nem hat
- A kettő között folyamatosan csökken a fény intenzitása.
- A fényköröket a fényforrásból induló, a fény irányával megegyező állású (végtelen)kúpoknak tekintjük.
- Egy felületi pont akkor van benne egy kúpban, ha a pontot a fényforrás pozíciójával összekötő egyenes a kúpon belül van.

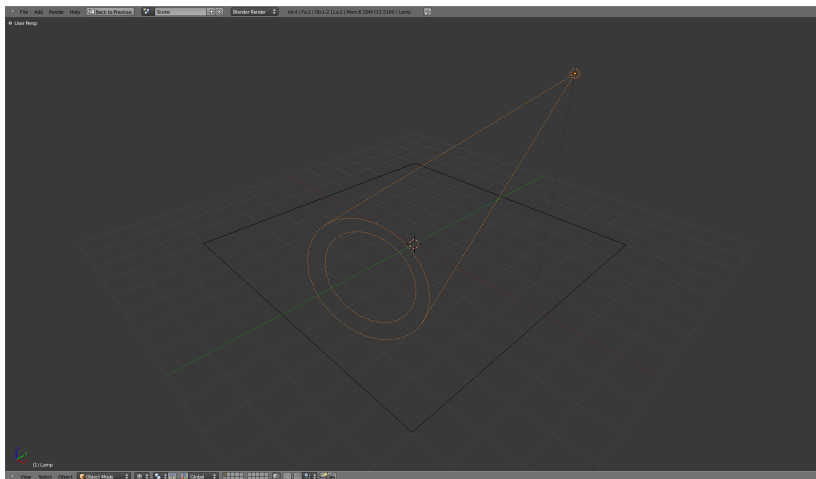
Spot fényforrás



Spot fényforrás



Spot fényforrás



Spot fényforrás hatása

- Ha a `coneDirection` és a `-toLight` által bezárt szög, kisebb, mint a kúp nyílásszögének a fele, akkor a szakasz benne van a kúpban, azaz a felületi pont is

Spot fényforrás hatása

- Ha a `coneDirection` és a `-toLight` által bezárt szög, kisebb, mint a kúp nyílásszögének a fele, akkor a szakasz benne van a kúpban, azaz a felületi pont is
- A szögek helyett elég vizsgálni azok $\cos$ -át, ha a max. nyílásszög $\leq 180^\circ$

Spot fényforrás hatása

- Ha a `coneDirection` és a `-toLight` által bezárt szög, kisebb, mint a kúp nyílásszögének a fele, akkor a szakasz benne van a kúpban, azaz a felületi pont is
- A szögek helyett elég vizsgálni azok $\cos$ -át, ha a max. nyílásszög $\leq 180^\circ$
- Adjuk meg a két fénykört, a hozzájuk tartozó kúpok nyílásszögének felének $\cos$ -ával:

Spot fényforrás hatása

- Ha a `coneDirection` és a `-toLight` által bezárt szög, kisebb, mint a kúp nyílásszögének a fele, akkor a szakasz benne van a kúpban, azaz a felületi pont is
- A szögek helyett elég vizsgálni azok $\cos$ -át, ha a max. nyílásszög $\leq 180^\circ$
- Adjuk meg a két fénykört, a hozzájuk tartozó kúpok nyílásszögének felének $\cos$ -ával:
 - belső fénykör: `cosInnerCone`

Spot fényforrás hatása

- Ha a `coneDirection` és a `-toLight` által bezárt szög, kisebb, mint a kúp nyílásszögének a fele, akkor a szakasz benne van a kúpban, azaz a felületi pont is
- A szögek helyett elég vizsgálni azok $\cos$ -át, ha a max. nyílásszög $\leq 180^\circ$
- Adjuk meg a két fénykört, a hozzájuk tartozó kúpok nyílásszögének felének $\cos$ -ával:
 - belső fénykör: `cosInnerCone`
 - külső fénykör: `cosOuterCone`

Spot fényforrás hatása

- `smoothstep(min, max, x)`:
 - 0, ha $x < \min$
 - 1, ha $x > \max$
 - egy átmenet 0 és 1 között (kübös Hermite, helyette lehetne használni lineáris interpolációt is)

Spot fényforrás hatása

- `smoothstep`(min, max, x):
 - 0, ha $x < \text{min}$
 - 1, ha $x > \text{max}$
 - egy átmenet 0 és 1 között (kübös Hermite, helyette lehetne használni lineáris interpolációt is)
- `float` spotFactor = `smoothstep`(cosOuterCone, cosInnerCone, `dot`(coneDirection, -toLight))

Spot fényforrás hatása

- `smoothstep`(min, max, x):
 - 0, ha $x < \text{min}$
 - 1, ha $x > \text{max}$
 - egy átmenet 0 és 1 között (kübös Hermite, helyette lehetne használni lineáris interpolációt is)
- `float` spotFactor = `smoothstep`(cosOuterCone, cosInnerCone, `dot`(coneDirection, -toLight))
- Ezzel kell beszorozni a diffúz és a spekuláris tagot

Összefoglalva

Felület tulajdonságai

- ambientColor
- diffuseColor
- specularColor
- specularPower
- normal
- worldPos

Fényforrás tulajdonságai

- diffuseLight
- specularLight
- toLight

Színtér tulajdonságai

- ambientLight
- eyePosition

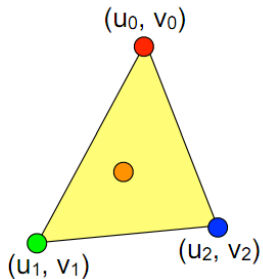
Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 **Textúrázás**
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

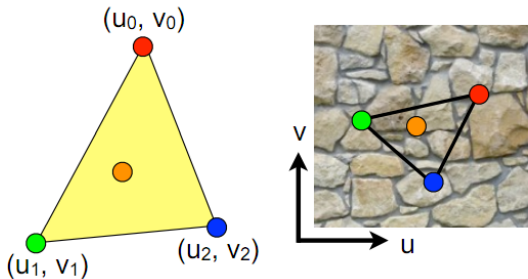
Mi az a textúrázás?

- 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*.
- A textúrák segítségével tehát a felület pontjaihoz további információt rendelünk a vertexektől függetlenül.

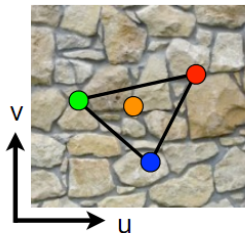
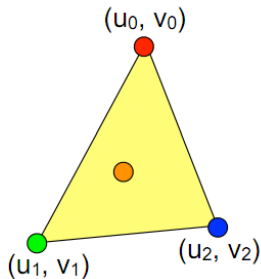
Textúrázás



Textúrázás



Textúrázás



Textúra megadási módok

- „Tömbbel”:
 - valamilyen 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
- Függvénnyel adjuk meg:
 - valamilyen $f: (x, y, z) \rightarrow \text{szín}$ vagy $f: (u, v) \rightarrow \text{szín}$ függvény segítségével
 - ezt nevezzük *procedurális textúrázásnak*

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás**
 - **Textúraleképezés**
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Leképezési módok

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

Leképezési módok

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

Leképezési módok

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

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 *pixel*t.
 - ⊕ Hatékony.

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 *pixel*t.
 - ⊕ Hatékony.
 - ⊖ Nem biztos, hogy minden pixel sorra kerül, és ami mégis, nem biztos, hogy csak egyszer.

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 *pixel*t.
 - ⊕ Hatékony.
 - ⊖ Nem biztos, hogy minden pixel sorra kerül, és ami mégis, nem biztos, hogy csak egyszer.
- Képtér → textúratér: *képtér alapú 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 *pixel*t.
 - ⊕ 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 *texel*t.

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 *pixel*t.
 - ⊕ 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 *texel*t.
 - ⊕ Inkrementális képszintézishez jól passzol.

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 *pixel*t.
 - ⊕ 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 *texel*t.
 - ⊕ Inkrementális képszintézishez jól passzol.
 - ⊖ Szükség van hozzá a paraméterezési és vetítési transzformációk inverzére.

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás**
 - Textúraleképezés
 - Paraméterezés**
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Paraméterezés

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

Paraméterezés

- Hogyan tudjuk eldönteni, 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

- Hogyan tudjuk eldönteni, 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

- Hogyan tudjuk eldönteni, 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.

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 $\mathcal{D}_F \neq \mathcal{D}_{\text{tex}}$, 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.
- Pl:
 - Henger palást

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 $\mathcal{D}_F \neq \mathcal{D}_{\text{tex}}$, akkor u, v -t transzformálni kell.
- Pl:
 - Henger palást
 - $\mathcal{D}_F = [0, 2\pi] \times [0, h]$, $F(u, v) := (\cos u, v, \sin u)$

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.
- Pl:
 - Henger palást
 - $\mathcal{D}_F = [0, 2\pi] \times [0, h], F(u, v) := (\cos u, v, \sin u)$
 - Textúra tér: $(\bar{u}, \bar{v}) \in [0, 1] \times [0, 1]$

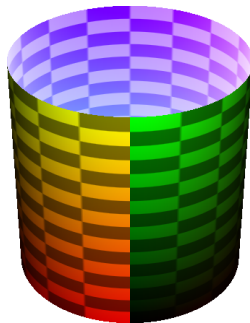
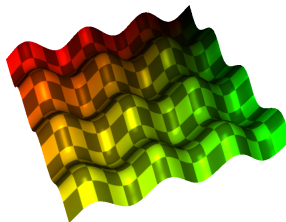
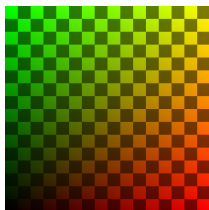
Parametrikus felületek paraméterezése

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

Parametrikus felületek paraméterezése



Háromszögek paraméterezése

- Legyen adott a háromszög három csúcsa:

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

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

Háromszögek paraméterezése

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

Háromszögek paraméterezése

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

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

Háromszögek paraméterezése

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

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

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!

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:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{P}^{-1} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Textúrázás és sugárkövetés

- Felület-sugár metszés: világ-koordináta rendszerben
- Inverz transzformációk:
 - Világ KR $\rightarrow$ Model KR
 - Model KR $\rightarrow$ textúra tér

Model KR $\rightarrow$ 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 (azaz a geometria nem animált), akkor $\mathbf{P}^{-1}$ állandó.

Háromszögek paraméterezése

- Gyakorlatban ez gyorsan számítható inkrementális algoritmussal.

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.

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.

Háromszögek paraméterezése

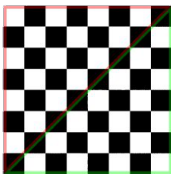
- Gyakorlatban ez gyorsan számítható inkrementális algoritmussal.
- Ha a három csúcsban adottak a textúra koordináták, akkor ezeket a háromszögek kitöltésénél használt algoritmussal kiszámíthatjuk minden pixelre.
- Minden pontra legyen a felület egy pontja
 $\mathbf{p} = \alpha \mathbf{p}_1 + \beta \mathbf{p}_2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

Perspektivikusan korrekt textúrázás

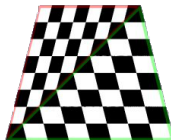
- A textúra koordináták lineáris interpolációja hibás képet ad, ha a megjelenítendő háromszögön nem csak affin transzformációt végzünk.

Perspektivikusan korrekt textúrázás

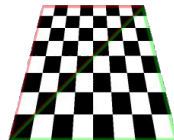
- A textúra koordináták lineáris interpolációja hibás képet ad, ha a megjelenítendő háromszögön nem csak affin transzformációt végzünk.
- Értsd: esetek 99%-a.



Sík

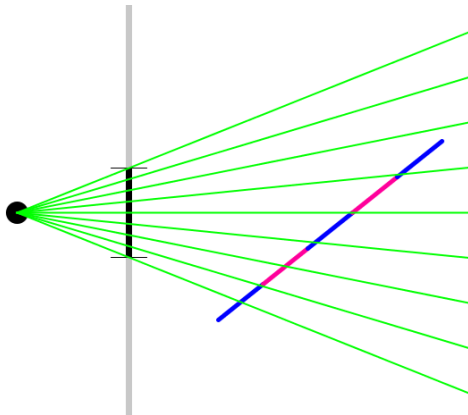


Affin



Helyes

Perspektivikusan korrekt textúrázás



Perspektivikusan korrekt textúrázás

- Tehát u, v lineáris interpolációja nem lesz jó nem affin transzformációk esetén – ugyanis ezek 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.

*Perspektivikusan korrekt textúrázás - bizonyítás 1/3

- Jelölje most a képernyő-koordinátákat X, Y , a térbeli koordinátákat pedig x, y, z , amelyek között fennáll a következő:

$$X = \frac{x}{z}, \quad Y = \frac{y}{z}$$

amiből nyilván $x = Xz$, $y = Yz$.

- Tegyük fel, hogy a fenti pont egy háromszög (vagy általában poligon) egy csúcspontja és legyen ezen primitív síkjának implicit egyenlete

$$Ax + By + Cz = D$$

- Jelölje u, v a térbeli primitív pontjainak textúrakoordinátáit, amiket tegyük fel, hogy kifejezhetünk az x, y, z koordináták lineáris függvényeként:

$$u = ax + by + cz + d, \quad v = ex + fy + gz + h$$

*Perspektivikusan korrekt textúrázás - bizonyítás 2/3

Lássuk be, hogy $1/z$ lineáris függvénye X, Y -nak:

- Induljunk ki a primitívünk síkjának implicit egyenletéből:

$$Ax + By + Cz = D$$

amibe beírva x és y a képernyő-koordinátákkal (X, Y -nal) való kifejezését kapjuk, hogy

$$AXz + BYz + Cz = D$$

- Ezt osszuk el zD -vel (azaz fejezzük ki $1/z$ -t):

$$\frac{1}{z} = \frac{A}{D}X + \frac{B}{D}Y + \frac{C}{D}$$

- Mivel A, B, C, D konstansok (azaz függetlenek X, Y -tól), ezért a fenti valóban $1/z$ kifejezése az X, Y lineáris függvényeként

*Perspektivikusan korrekt textúrázás - bizonyítás 3/3

Lássuk be, hogy u/z , v/z lineáris függvénye X , Y -nak:

- Feltettük, hogy a textúrakoordináták a térbeli koordinátákban lineárisak, azaz

$$u = ax + by + cz + d$$

$$v = ex + fy + gz + h$$

amibe behelyettesítve a térbeli koordináták képkoordinátákkal való kifejezését kapjuk, hogy

$$u = aXz + bYz + cz + d$$

$$v = eXz + fYz + gz + h$$

*Perspektivikusan korrekt textúrázás - bizonyítás 3/3

Lássuk be, hogy u/z , v/z lineáris függvénye X , Y -nak:

- Osszunk z -vel és helyettesítsük be az előbb kapott összefüggést $1/z$ -re a jobboldalon:

$$\frac{u}{z} = aX + bY + c + d \left(\frac{A}{D}X + \frac{B}{D}Y + \frac{C}{D} \right)$$

$$\frac{v}{z} = eX + fY + g + h \left(\frac{A}{D}X + \frac{B}{D}Y + \frac{C}{D} \right)$$

azaz valóban lineárisak, mert a fentit átrendezve

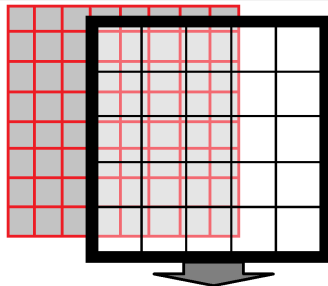
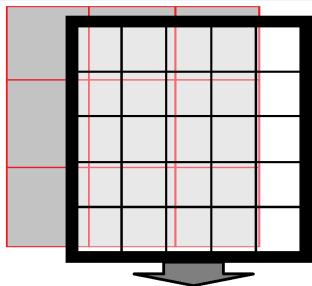
$$\frac{u}{z} = \left(a + d \frac{A}{D} \right) X + \left(b + d \frac{B}{D} \right) Y + \left(c + d \frac{C}{D} \right)$$

$$\frac{v}{z} = \left(e + h \frac{A}{D} \right) X + \left(f + h \frac{B}{D} \right) Y + \left(g + h \frac{C}{D} \right)$$

Tartalom

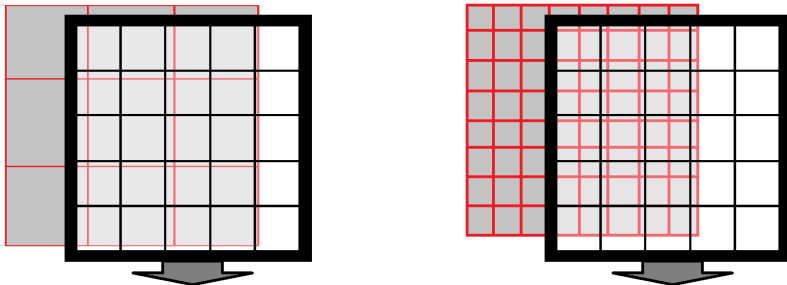
- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 **Textúrázás**
 - Textúraleképezés
 - Paraméterezés
 - **Textúra szűrés**
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Textúrák szűrése



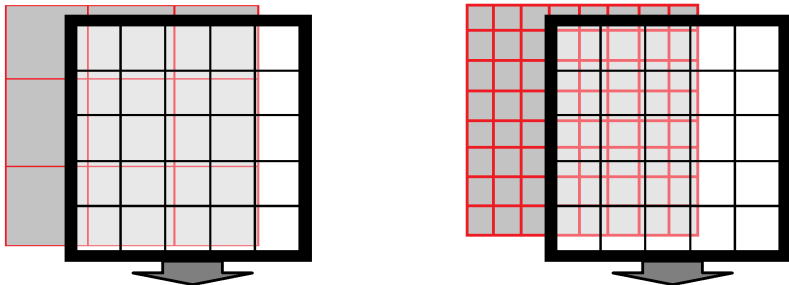
- Ritkán esik pontosan egy texel egy pixelre.

Textúrák szűrése



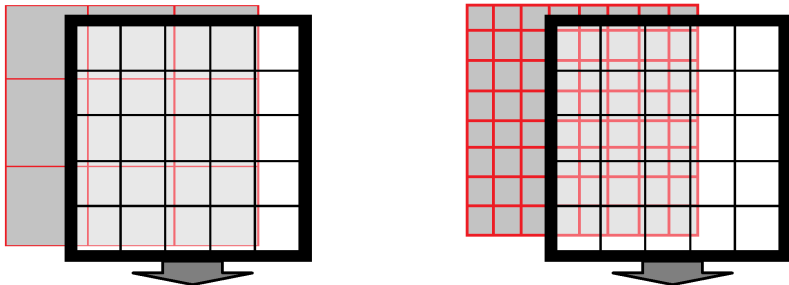
- Ritkán esik pontosan egy texel egy pixelre.
- Nagyítás: a pixel mérete kisebb a texelénél – több pixel jut egyetlen texelre. OpenGL: `GL_TEXTURE_MAG_FILTER`

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`

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.

Nagyítás

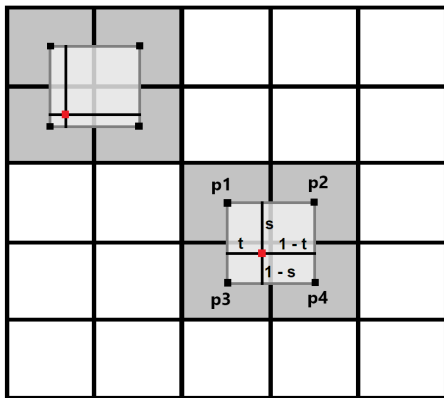
Egy pixel mérete kisebb egy texelénél – több pixel jut egyetlen texelre

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

Bilineáris szűrés

$$\mathbf{b}(s, t) = (1 - t)((1 - s)\mathbf{p}_1 + s\mathbf{p}_3) + t((1 - s)\mathbf{p}_2 + s\mathbf{p}_4)$$

$$s, t \in [0, 1]$$



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.

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.

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 használjunk kevesebb texelt, vagy *MIP-map*-eket

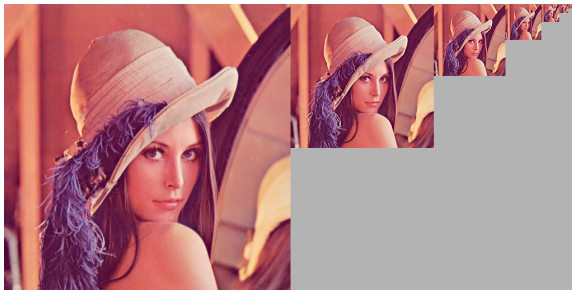
Kicsinyítés

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 használjunk kevesebb texelt, vagy *MIP-map*-eket
- Kevesebb texel:
 - Szűrés nélkül (Nearest neighbour filtering): a pixel középpontjához legközelebb eső texel értékét használjuk.
 - Bilineáris szűrés: a legközelebbi négy texel súlyozott átlagát vesszük.

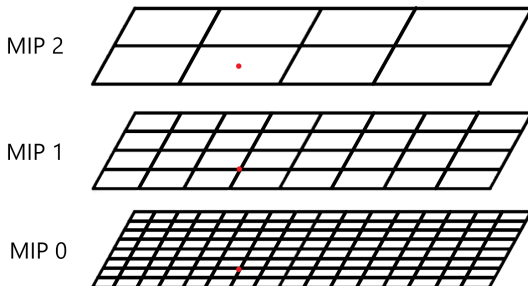
MIP-map-ek

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

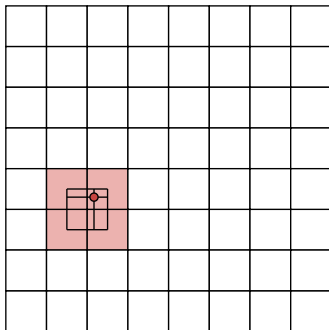


MIP-map-ek

- Szűrés során kiválasztjuk a pixel/textel terület arányának 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 súlyozott átlagát.

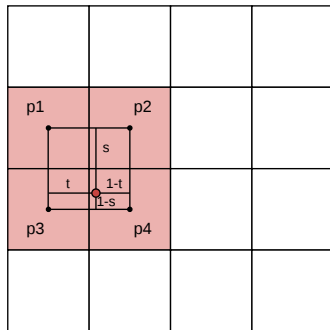


Trilineáris szűrés



Mipmap N. szint

Bilineáris eredmény

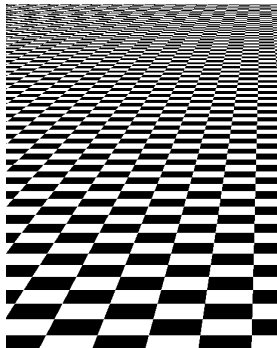


Mipmap N+1.szint

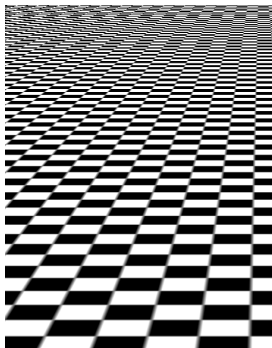
Bilineáris eredmény

Lineáris interpoláció folytonos N alapján

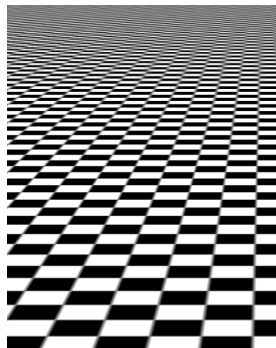
Szűrések – összehasonlítás



Nearest neighbour



Bilineáris



Trilineáris

Szűrések – összehasonlítás



Nearest neighbour



Bilineáris



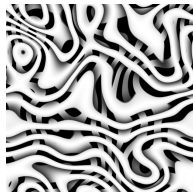
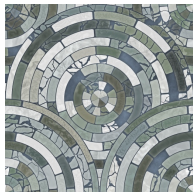
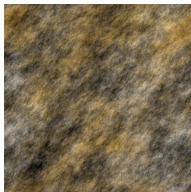
Trilineáris

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 **Textúrázás**
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - **Procedurális textúrák**
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Procedurális textúrák

- A textúrákat „tömb” helyett megadhatjuk függvénnyel is.
- Textúra koordináták: a függvény paraméterei.



generated with Filter Forge

Tulajdonságai

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

⇒ Nagyításnál nem kell szűrés. (Kicsinyítést sajnos nem oldja meg.)
- Hátrányok:
 - Nagy számításigény.
 - Bonyolultabban módosítható.

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 **Textúrázás**
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - **Nem-szín textúrák**
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Nem-szín textúrák

- A textúrák segítségével a felületi pont bármilyen tulajdonságát leírhatjuk.
- Ezek a tulajdonságok lehetnek pl.:

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

- 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

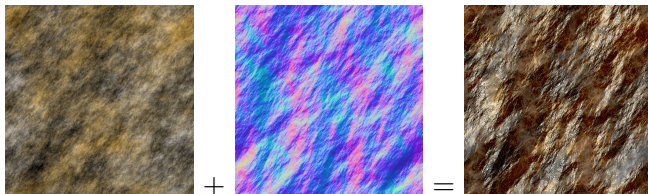
- A textúrák segítségével a felületi pont bármilyen tulajdonságát leírhatjuk.
- Ezek a tulajdonságok lehetnek pl.:
 - felületi normális – *Bucka ill. normál leképezés*
 - elmozdulás – Displacement *leképezés*
 - fényforrás láthatósága – *Árnyéktérképek*

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 – *Árnyéktérképek*
 - tükörként visszavert fény – *Visszaverődés-leképezés* ill. *Környezetleképezés*

Bucka vagy normál leképezés

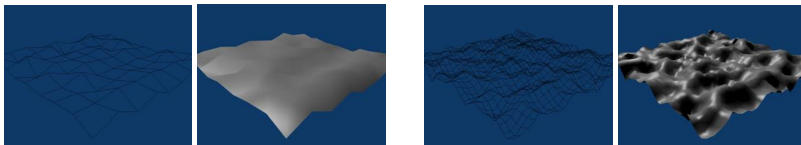
- A textúrával normál vektorokat adunk meg.
- A felület eredeti normálisai helyett ezeket használjuk a megvilágítás számításakor.
- Rűcskös/érdes felület látszatát adja, amíg nem nézünk rá túl lapos szögben.



generated with Filter Forge

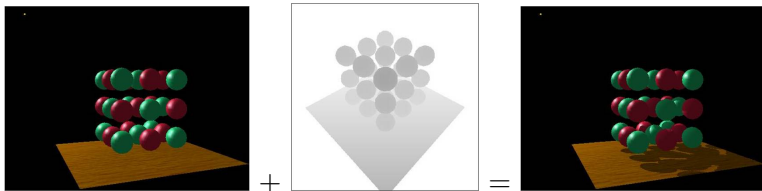
Displacement leképezés

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



Árnyéktérképek

- 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.

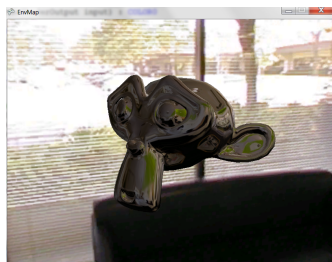
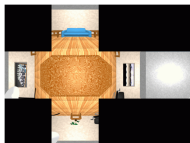
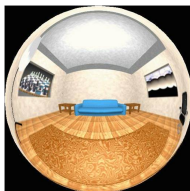


Visszaverődés-leképezés

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

Környezetleképezés

- A színterünk környezetét végtelenül távolinak tekintjük, a lekérdezésnél csak az irány számít, a pozíció nem.
- Speciális textúrában tároljuk. (Jellemzően *Cubemap*.)
- Rajzolásnál a tükröződő felületekhez ebből olvasunk a tükörirány szerint.



Összefoglalva

- Minden felületi-optikai tulajdonságot meg lehet adni konstanssal, vagy akár textúrával.

Összefoglalva

- Minden felületi-optikai tulajdonságot meg lehet adni konstanssal, vagy akár textúrával.
- (Sőt, még többet, ha ügyesek vagyunk!)

Összefoglalva

- Minden felületi-optikai tulajdonságot meg lehet adni konstanssal, vagy akár textúrával.
- (Sőt, még többet, ha ügyesek vagyunk!)
- Minden vektor és pont világkoordináta-rendszerben adott.

Összefoglalva

- Minden felületi-optikai tulajdonságot meg lehet adni konstanssal, vagy akár textúrával.
- (Sőt, még többet, ha ügyesek vagyunk!)
- Minden vektor és pont világkoordináta-rendszerben adott.
- eyePosition-t frissíteni kell, ha változik a nézet!

Összefoglalva

- Minden felületi-optikai tulajdonságot meg lehet adni konstanssal, vagy akár textúrával.
- (Sőt, még többet, ha ügyesek vagyunk!)
- Minden vektor és pont világkoordináta-rendszerben adott.
- `eyePosition`-t frissíteni kell, ha változik a nézet!
- Probléma: a modellünk modell k.r.-ben van, és a *Vertex Shader* normalizált eszköz k.r.-be visz át!

Összefoglalva

- Minden felületi-optikai tulajdonságot meg lehet adni konstanssal, vagy akár textúrával.
- (Sőt, még többet, ha ügyesek vagyunk!)
- Minden vektor és pont világkoordináta-rendszerben adott.
- `eyePosition`-t frissíteni kell, ha változik a nézet!
- Probléma: a modellünk modell k.r.-ben van, és a *Vertex Shader* normalizált eszköz k.r.-be visz át!
- Megoldás: számoltassunk a *Vertex Shader*-rel világ k.r.-beli koordinátákat is, és ezeket interpolálva adjuk át a *Fragment Shader*-nek!

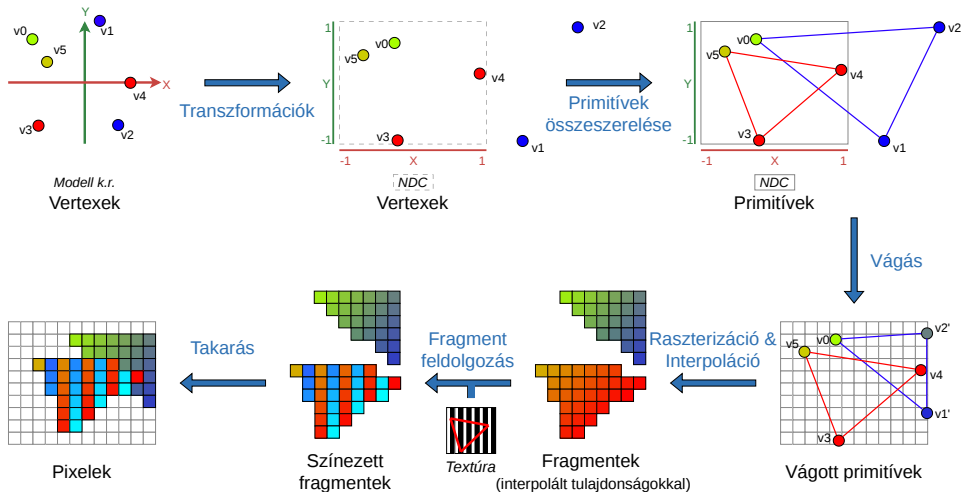
Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

Inkrementális képszintézis



Pipeline

- Az elvégzendő feladatokat részfeladatokra bontjuk
- Mindegyik részfeladatot más-más feldolgozóegység (processzor) dolgozza fel (ideális esetben)
- Minden egység inputja, a pipeline-ban őt megelőző egység outputja

Pipeline

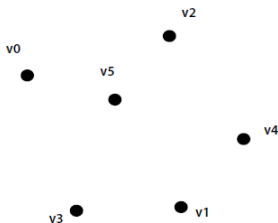
- Ahhoz, hogy egy egység el tudjon kezdeni dolgozni, nem kell megvárnia, hogy befejeződjön a munka az egész munkadarabon, csak az előtte lévő részek kell, hogy készen legyenek!
- → ha n pipeline fázist (**stage**-et) vezetünk be, akkor egy adott pillanatban egyszerre akár n elemen is dolgozhatunk (felfutás után)
- Nem csak GPU! Már a Pentium IV-nek is 20 pipeline stage-e volt (egy GeForce 3-nak 6-800)

Tartalom

- 1 Áttekintés
- 2 Árnyalás
 - Fényforrásmodellek
 - Anyagmodellek
 - Megvalósítás
- 3 Textúrázás
 - Textúraleképezés
 - Paraméterezés
 - Textúra szűrés
 - Procedurális textúrák
 - Nem-szín textúrák
- 4 Inkrementális képszintézis a GPU-n
 - Inkrementális képszintézis
 - Pipeline a hardveren

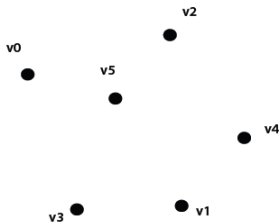
Párhuzamosítások a gyakorlatban – per vertex

- Amikor `glDrawArrays`, `glDrawElements` stb. hívást csinálunk, akkor a kirajzolandó primitív csúcspontjaira elindul a szerelőszalag
- Minden **csúcspontot** *egymástól függetlenül, párhuzamosan* transzformálunk

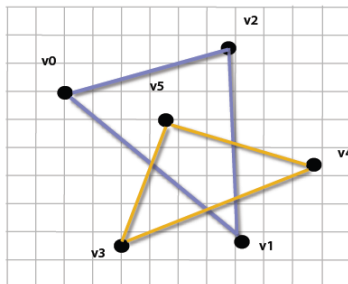


Párhuzamosítások a gyakorlatban – per primitív

- Minden kirajzolandó **primitívet** egymástól függetlenül, párhuzamosan vágunk



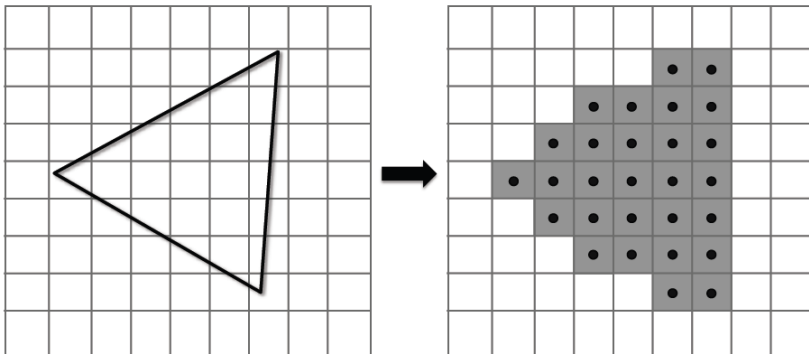
Vertices



**Primitives
(triangles)**

Párhuzamosítások a gyakorlatban – per primitív

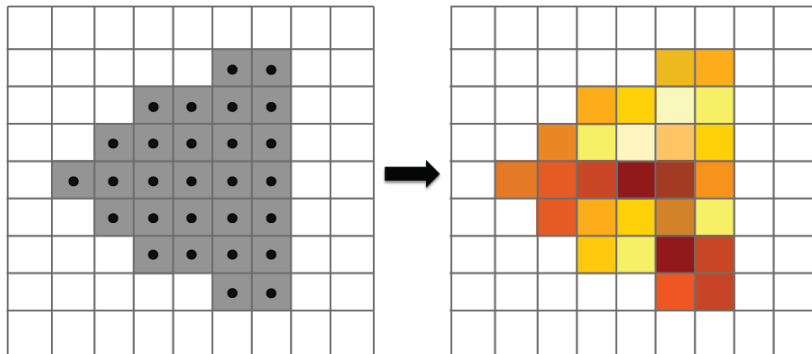
- Minden kirajzolandó **primitívet** egymástól függetlenül, párhuzamosan raszterizálunk



Fragments

Párhuzamosítások a gyakorlatban – per fragment

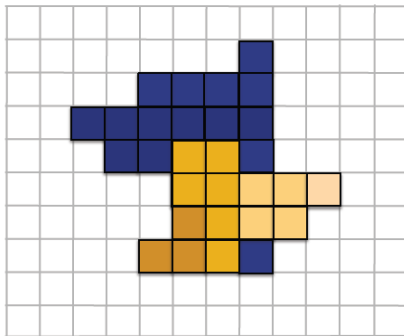
- Minden kirajzolandó **fragmentet** egymástól függetlenül, párhuzamosan színezzünk



Shaded fragments

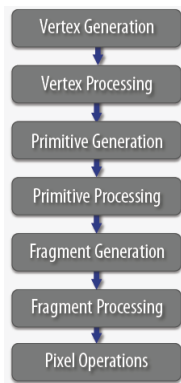
Párhuzamosítások a gyakorlatban – per pixel

- Minden fragmenthez tartozó **pixelre** meghatározzuk a képernyőn megjelenítendő képpontszínt (+Z-pufferrel láthatóságot)



Pixels

Szerelőszalag a hardveren



GPU-programozás

- Nincs explicit párhuzamosítás
- Mert az abból származik, hogy a különböző elemeket egymástól függetlenül, (akár időben) párhuzamosan dolgozzuk fel

Shaderek fordítása

```
sampler mySamp;  
Texture2D<float3> myTex;  
float3 lightDir;  
  
float4 diffuseShader(float3 norm, float2 uv)  
{  
    float3 kd;  
    kd = myTex.Sample(mySamp, uv);  
    kd *= clamp( dot(lightDir, norm), 0.0, 1.0);  
    return float4(kd, 1.0);  
}
```

1 input fragment record



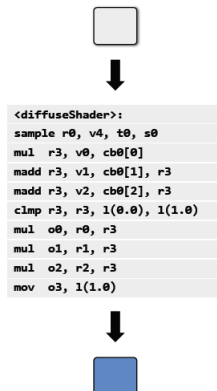
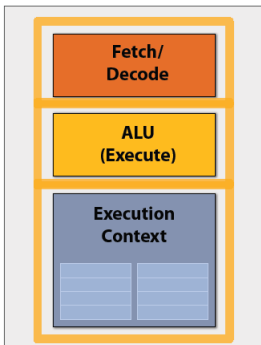
```
<diffuseShader>:  
sample r0, v4, t0, s0  
mul r3, v0, cb0[0]  
madd r3, v1, cb0[1], r3  
madd r3, v2, cb0[2], r3  
clmp r3, r3, 1(0.0), 1(1.0)  
mul o0, r0, r3  
mul o1, r1, r3  
mul o2, r2, r3  
mov o3, 1(1.0)
```



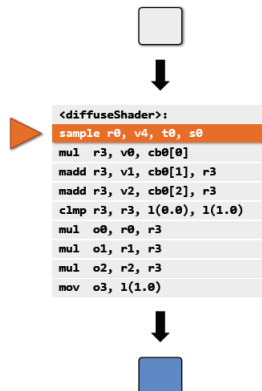
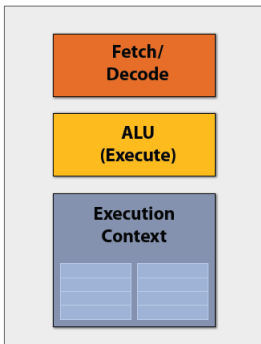
1 output fragment record



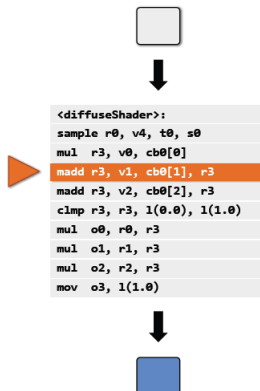
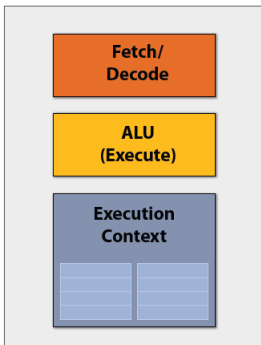
Shaderek végrehajtása



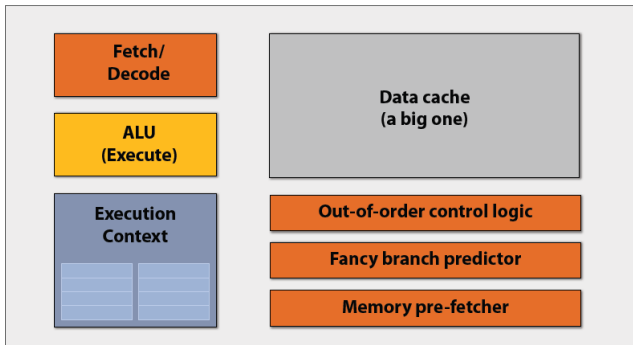
Shaderek végrehajtása



Shaderek végrehajtása

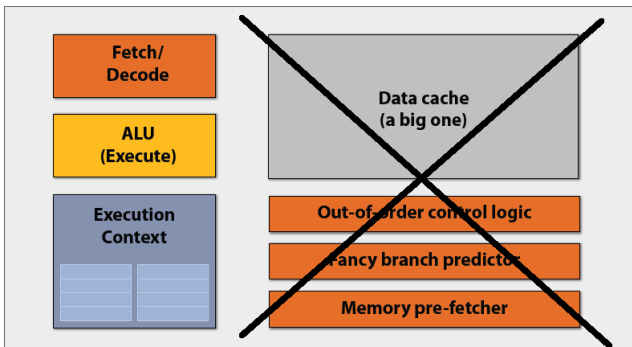


Egy tipikus CPU váz



GPU-s feldolgozó egység

A csak egyetlen egy szál gyors végrehajtását segítő komponenseket vegyük ki

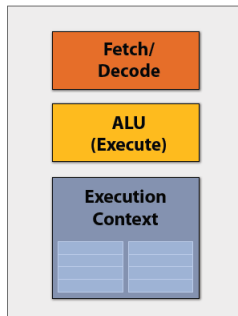
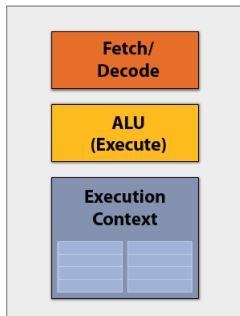


GPU – 2 feldolgozó egység, 2 shader

fragment 1



```
<diffuseShader>:  
sample r0, v4, t0, s0  
mul r3, v0, cb0[0]  
madd r3, v1, cb0[1], r3  
madd r3, v2, cb0[2], r3  
clmp r3, r3, l(0.0), l(1.0)  
mul o0, r0, r3  
mul o1, r1, r3  
mul o2, r2, r3  
mov o3, l(1.0)
```



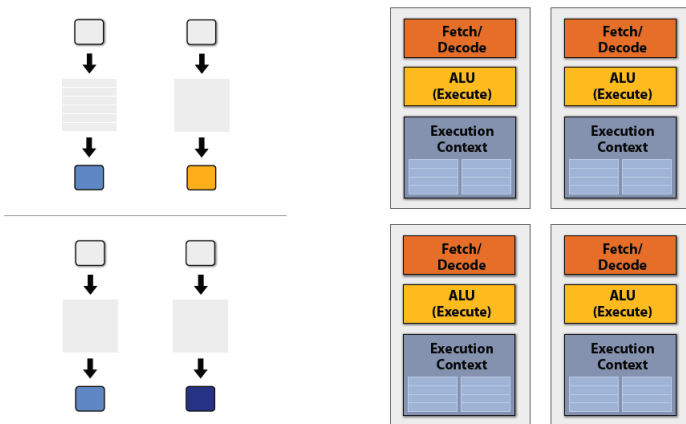
fragment 2



```
<diffuseShader>:  
sample r0, v4, t0, s0  
mul r3, v0, cb0[0]  
madd r3, v1, cb0[1], r3  
madd r3, v2, cb0[2], r3  
clmp r3, r3, l(0.0), l(1.0)  
mul o0, r0, r3  
mul o1, r1, r3  
mul o2, r2, r3  
mov o3, l(1.0)
```



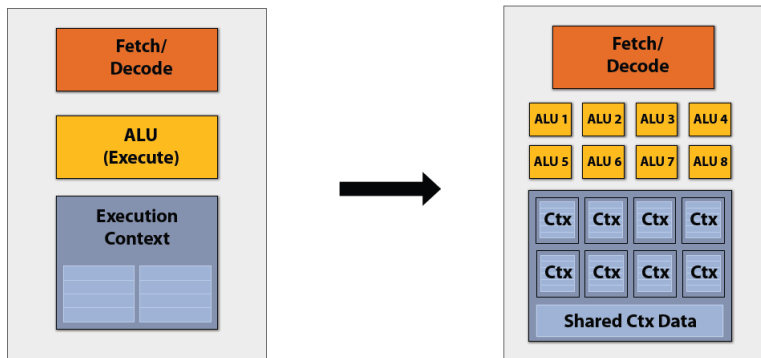
GPU – 4 feldolgozó egység, 4 shader



GPU feldolgozó egységek

- Előbbi ábrákon mindegyik feldolgozóegység különböző shader-kódokat is futtathatott
- De erre nincs feltétlen szükség!
- Egy háromszögből nagyon sok fragment is készülhet → mindre ugyanazt a shadert kell lefuttatni!
- ⇒ csökkentsük a költségeket úgy, hogy több ALU-hoz közös utasításszedőt (azaz: futtatott programot) rendelünk → **Single Instruction Multiple Data**

GPU feldolgozó egység



GPU feldolgozó egység

<diffuseShader>:

sample r0, v4, t0, s0

mul r3, v0, cb0[0]

madd r3, v1, cb0[1], r3

madd r3, v2, cb0[2], r3

clmp r3, r3, 1(0.0), 1(1.0)

mul o0, r0, r3

mul o1, r1, r3

mul o2, r2, r3

mov o3, 1(1.0)



<VEC8_diffuseShader>:

VEC8_sample vec_r0, vec_v4, t0, vec_s0

VEC8_mul vec_r3, vec_v0, cb0[0]

VEC8_madd vec_r3, vec_v1, cb0[1], vec_r3

VEC8_madd vec_r3, vec_v2, cb0[2], vec_r3

VEC8_clmp vec_r3, vec_r3, 1(0.0), 1(1.0)

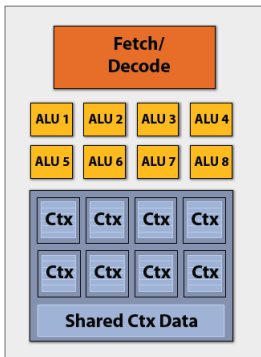
VEC8_mul vec_o0, vec_r0, vec_r3

VEC8_mul vec_o1, vec_r1, vec_r3

VEC8_mul vec_o2, vec_r2, vec_r3

VEC8_mov o3, 1(1.0)

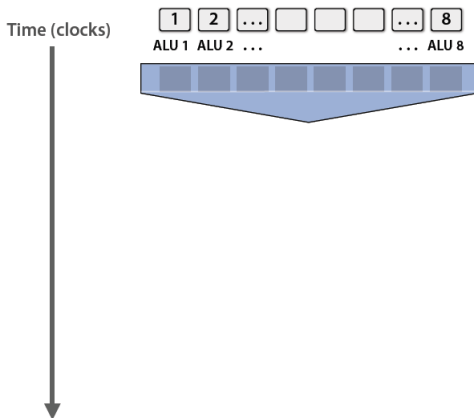
GPU feldolgozó egység



```
<VEC8_diffuseShader>:
VEC8_sample vec_r0, vec_v4, t0, vec_s0
VEC8_mul vec_r3, vec_v0, cb0[0]
VEC8_madd vec_r3, vec_v1, cb0[1], vec_r3
VEC8_madd vec_r3, vec_v2, cb0[2], vec_r3
VEC8_clmp vec_r3, vec_r3, 1(0.0), 1(1.0)
VEC8_mul vec_o0, vec_r0, vec_r3
VEC8_mul vec_o1, vec_r1, vec_r3
VEC8_mul vec_o2, vec_r2, vec_r3
VEC8_mov o3, 1(1.0)
```



Elágazások

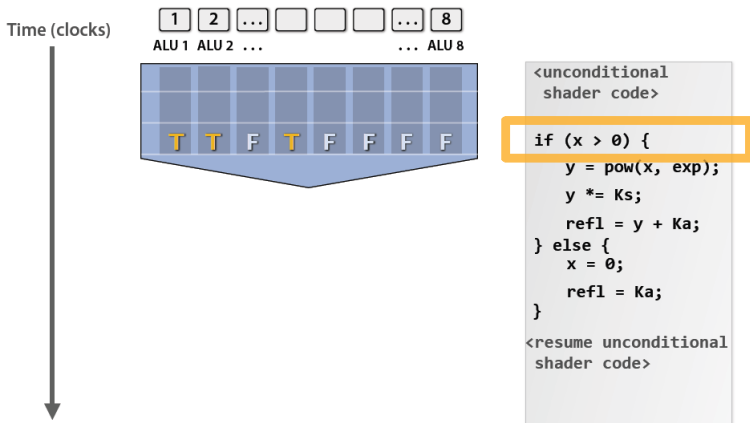


```
<unconditional  
shader code>
```

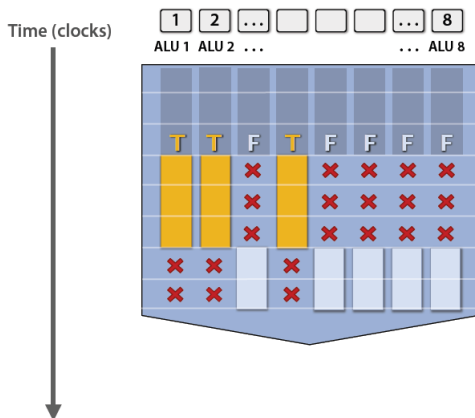
```
if (x > 0) {  
    y = pow(x, exp);  
    y *= Ks;  
    refl = y + Ka;  
} else {  
    x = 0;  
    refl = Ka;  
}
```

```
<resume unconditional  
shader code>
```

Elágazások



Elágazások



```
<unconditional  
shader code>
```

```
if (x > 0) {
```

```
    y = pow(x, exp);
```

```
    y *= Ks;
```

```
    refl = y + Ka;
```

```
} else {
```

```
    x = 0;
```

```
    refl = Ka;
```

```
}
```

```
<resume unconditional  
shader code>
```

Elágazások

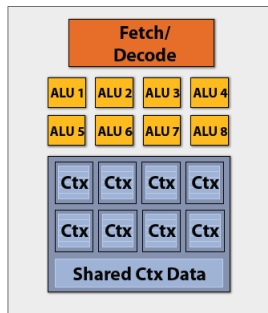
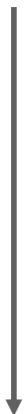
- Az ALU1-ALU8 a közös utasításfeldolgozó egységhez rendelt ALU-kat jelöli
- A T azoknál az ALU-knál jelent meg, akiknek az adatai alapján az elágazás feltétele *igaz*
- Az F pedig azoknál, akiknél *hamis*
- Mindkét ág utasításait (!) ki kell értékelni
- Legrosszabb esetben a nyolcadára esik a fenti ábrán a teljesítmény!
- Mai GPU-kon 16-64 ALU van közös utasítás-folyamon

Várakozások

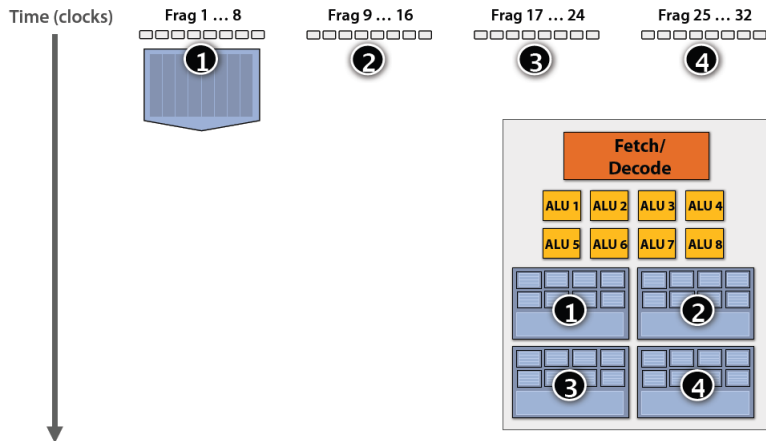
- Egy ALU-n átlapolva több fragmentet/vertexet/stb. dolgozzunk fel → több végrehajtási kontextus tárolása (kód, átmeneti adatok stb.)
- Mert bizonyos műveletek sokkal lassabbak, mint mások
- Például: egy textúralekérdezés shaderben 100x vagy akár 1000x lassabb, mint egy aritmetikai (vektor)művelet!

Átlapolás

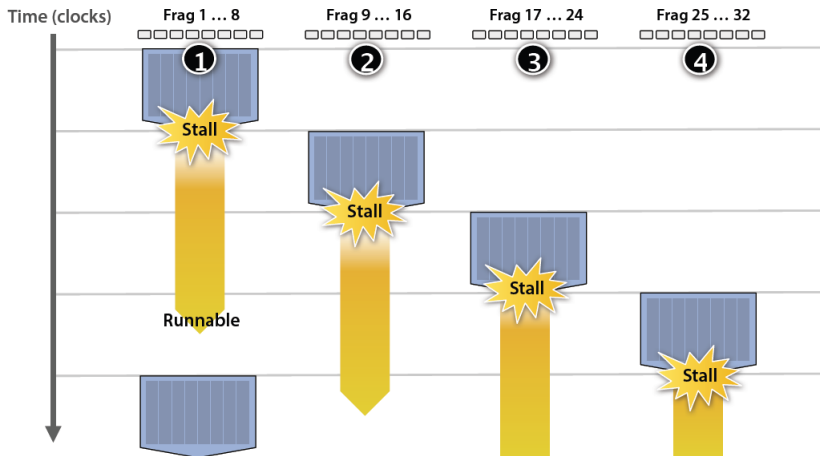
Time (clocks)



Átlapolás



Átlapolás



GPU feldolgozó egység

A GPU-s streamprocesszorok felépítésében tehát a következő észrevételek játszottak fontos szerepet:

- ❶ A csak egyetlen egy szál gyors végrehajtását segítő komponenseket vegyük ki
- ❷ $\Rightarrow$ csökkentsük a költségeket úgy, hogy több ALU-hoz közös utasításszedőt (azaz: futtatott programot) rendelünk $\rightarrow$ **Single Instruction Multiple Data**
- ❸ Egy ALU-n átlapolva több fragmentet/vertexet/stb. dolgozzunk fel $\rightarrow$ több végrehajtási kontextus tárolása (kód, átmeneti adatok stb.)