Számítógépes Grafika

Valasek Gábor
valasek@inf.elte.hu

Eötvös Loránd Tudományegyetem
Informatikai Kar

2011/2012. őszi félév

Tartalom

Tartalom

Absztrakt fényforrások

- Irányfényforrás

Absztrakt fényforrások

- Irányfényforrás
  - csak iránya van

Absztrakt fényforrások

- Irányfényforrás
  - csak iránya van
  - egyetlen (világ k.r. beli) vektor megadja

Absztrakt fényforrások

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

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

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

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

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

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
  - iránya, pozíciója és ”fényköre” van

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
  - iránya, pozíciója és ”fényköre” van
  - két szög + egy pont és egy vektor adja meg (világ k.r. !)

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
  - iránya, pozíciója és ”fényköre” van
  - két szög + egy pont és egy vektor adja meg (világ k.r. !)
  - Pl.: asztali lámpa

Tartalom

Jelölések

- $\mathbf{v} := \omega$ a nézeti irány

Jelölések

- $\mathbf{v} := \omega$ a nézeti irány
- $\mathbf{l} := \omega'$ a megvilágító, a fényt ”adó” pont fele mutató vektor

Jelölések

- $\mathbf{v} := \omega$ a nézeti irány
- $\mathbf{l} := \omega'$ a megvilágító, a fényt ”adó” pont fele mutató vektor
- $\mathbf{n}$ a felületi normális

Jelölések

- **v** $:= \omega$ a nézeti irány
- **l** $:= \omega'$ a megvilágító, a fényt ”adó” pont fele mutató vektor
- **n** a felületi normális
- **r**: az ideális visszaverődés iránya az **l** beesési irányból, **n** normális mellett

Jelölések

- $\mathbf{v} := \omega$ a nézeti irány
- $\mathbf{l} := \omega'$ a megvilágító, a fényt ”adó” pont fele mutató vektor
- $\mathbf{n}$ a felületi normális
- $\mathbf{r}$: az ideális visszaverődés iránya az $\mathbf{l}$ beesési irányból, $\mathbf{n}$ normális mellett
- $\mathbf{v}, \mathbf{l}, \mathbf{n}$ egységvektorok

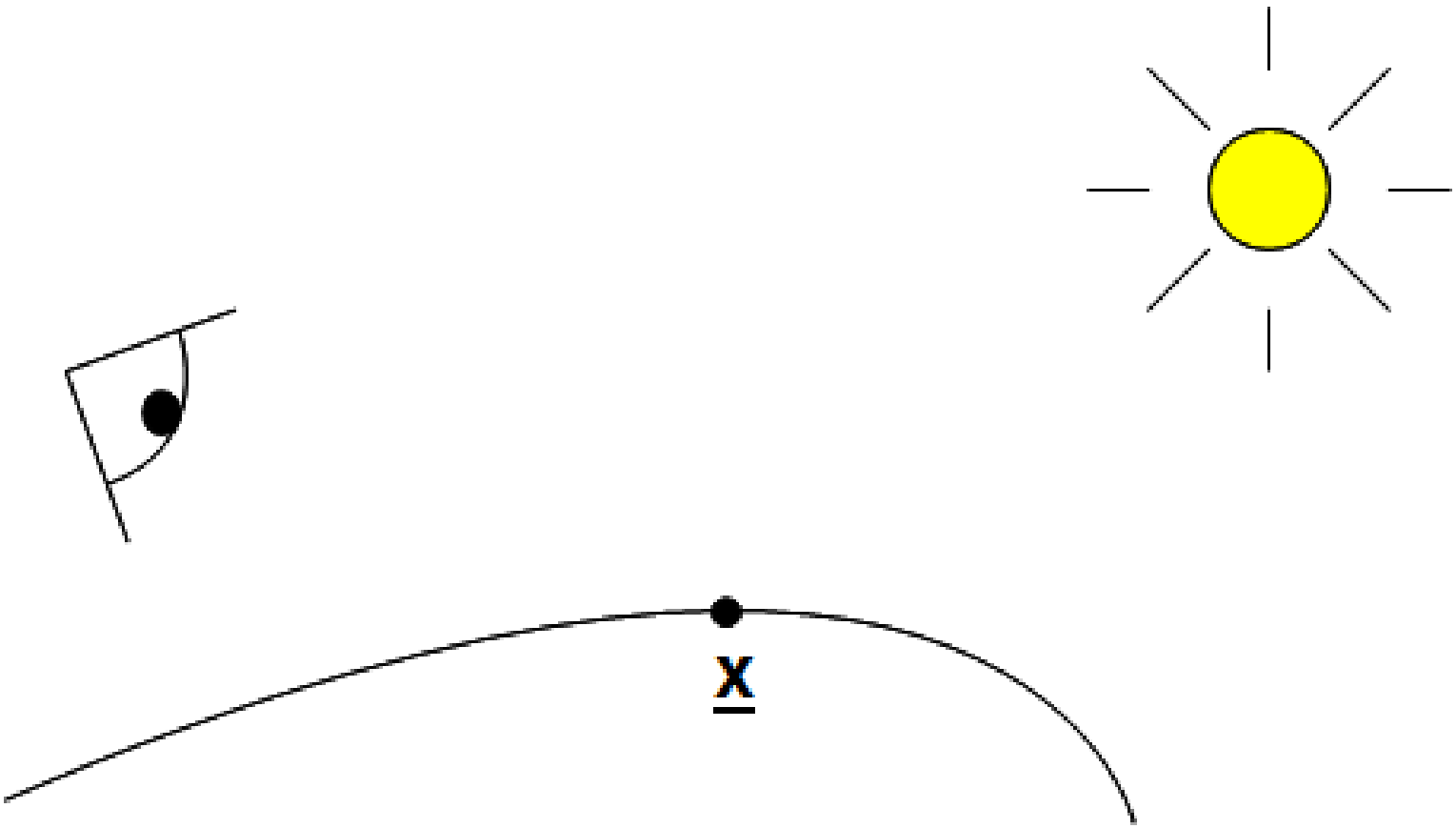
Jelölések

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

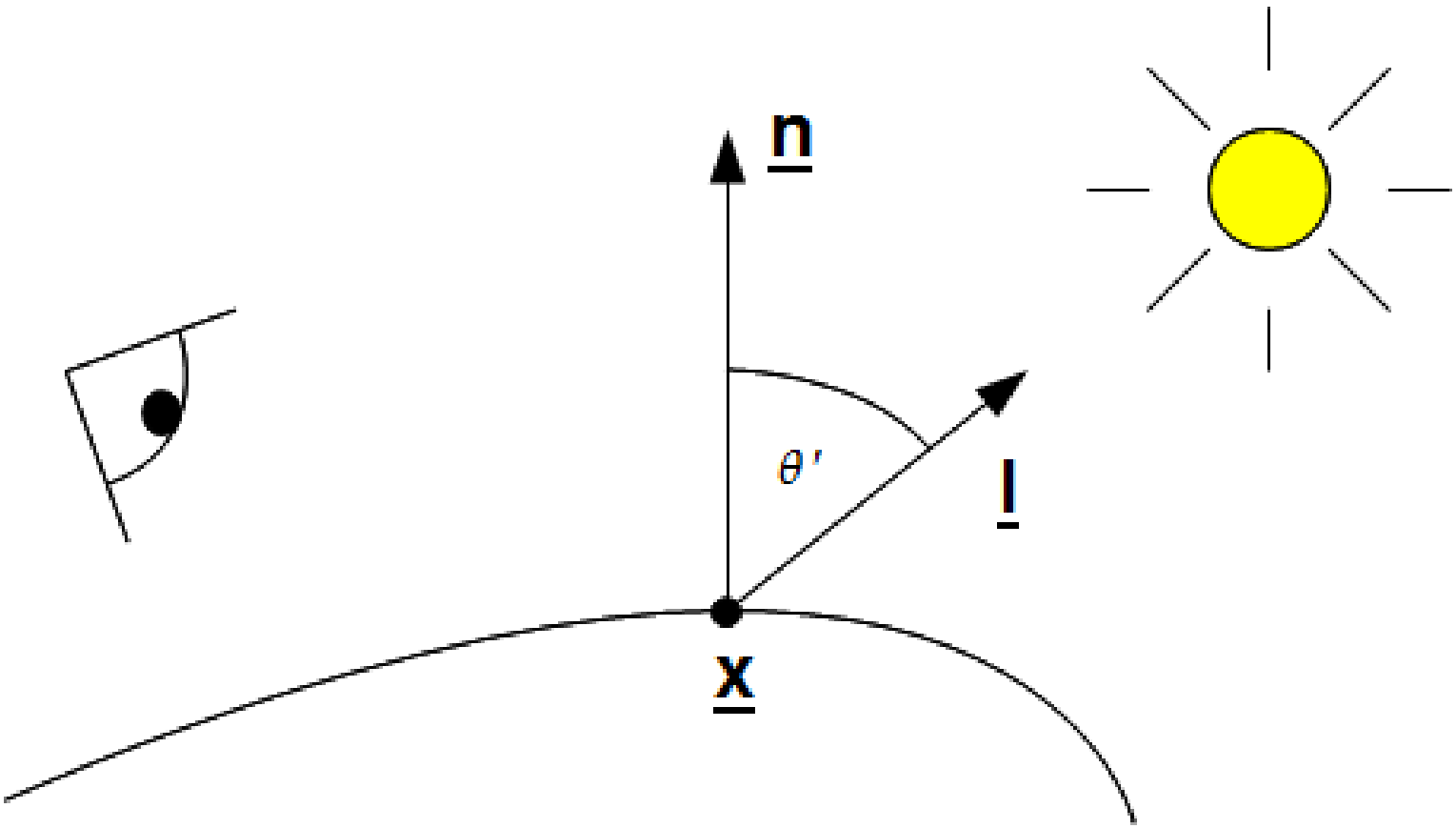
Jelölések

- **v** $:= \omega$ a nézeti irány
- **l** $:= \omega'$ a megvilágító, a fényt ”adó” pont fele mutató vektor
- **n** a felületi normális
- **r**: az ideális visszaverődés iránya az **l** beesési irányból, **n** normális mellett
- **v**, **l**, **n** egységvektorok
- θ' a **l** és a **n** által bezárt szög
- ϕ az **r** és **v** által bezárt szög

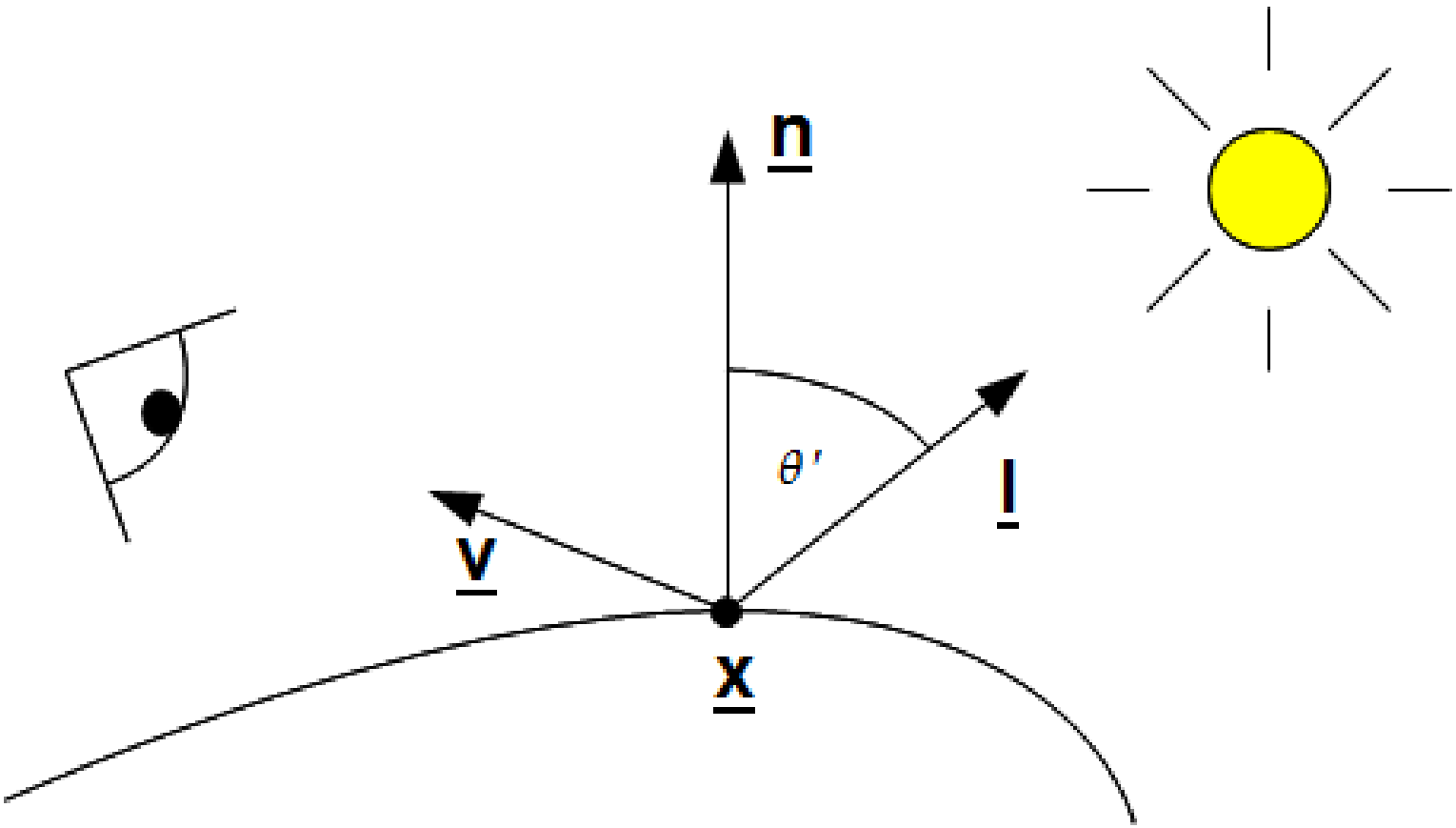
Jelölések



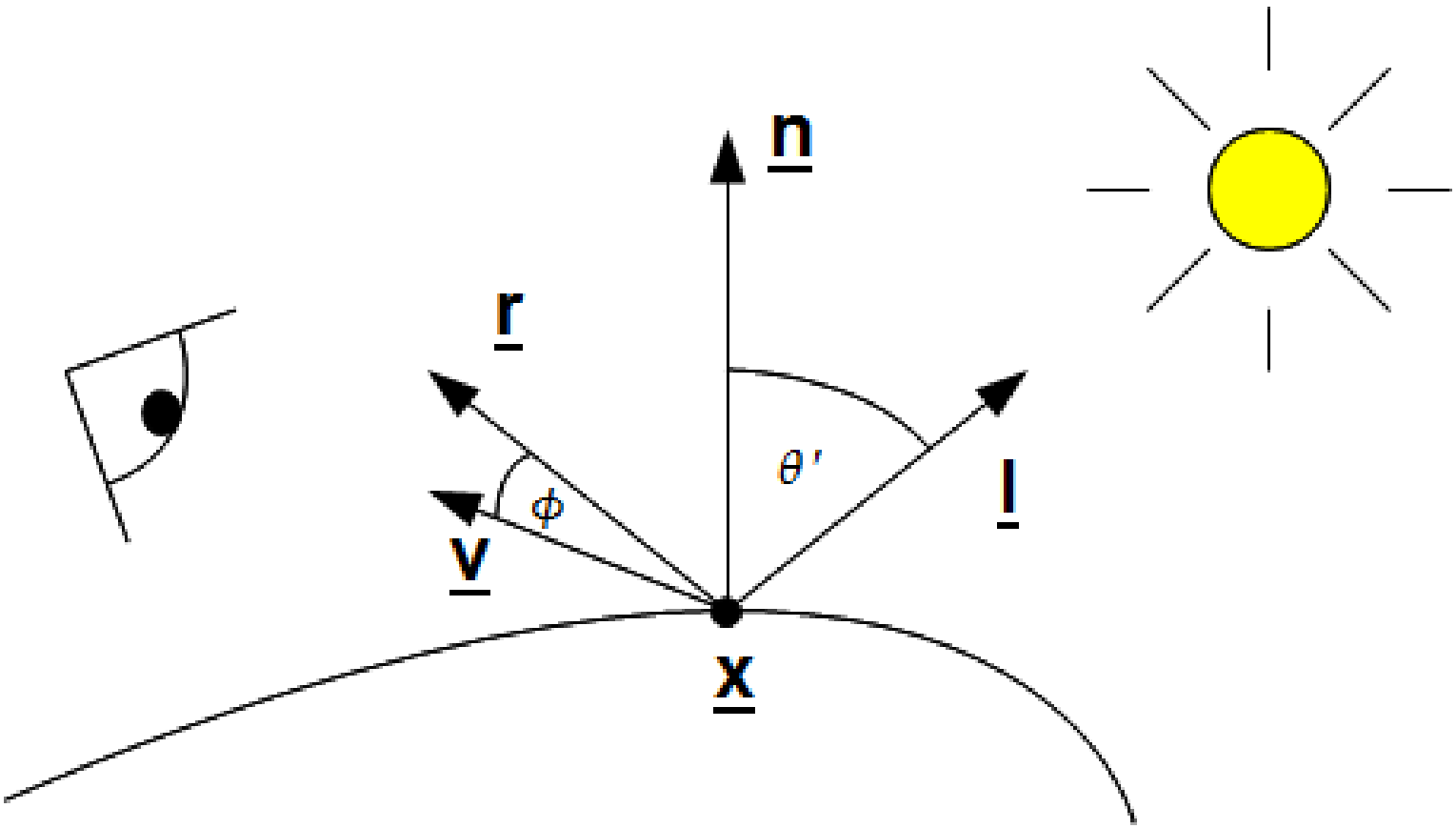
Jelölések



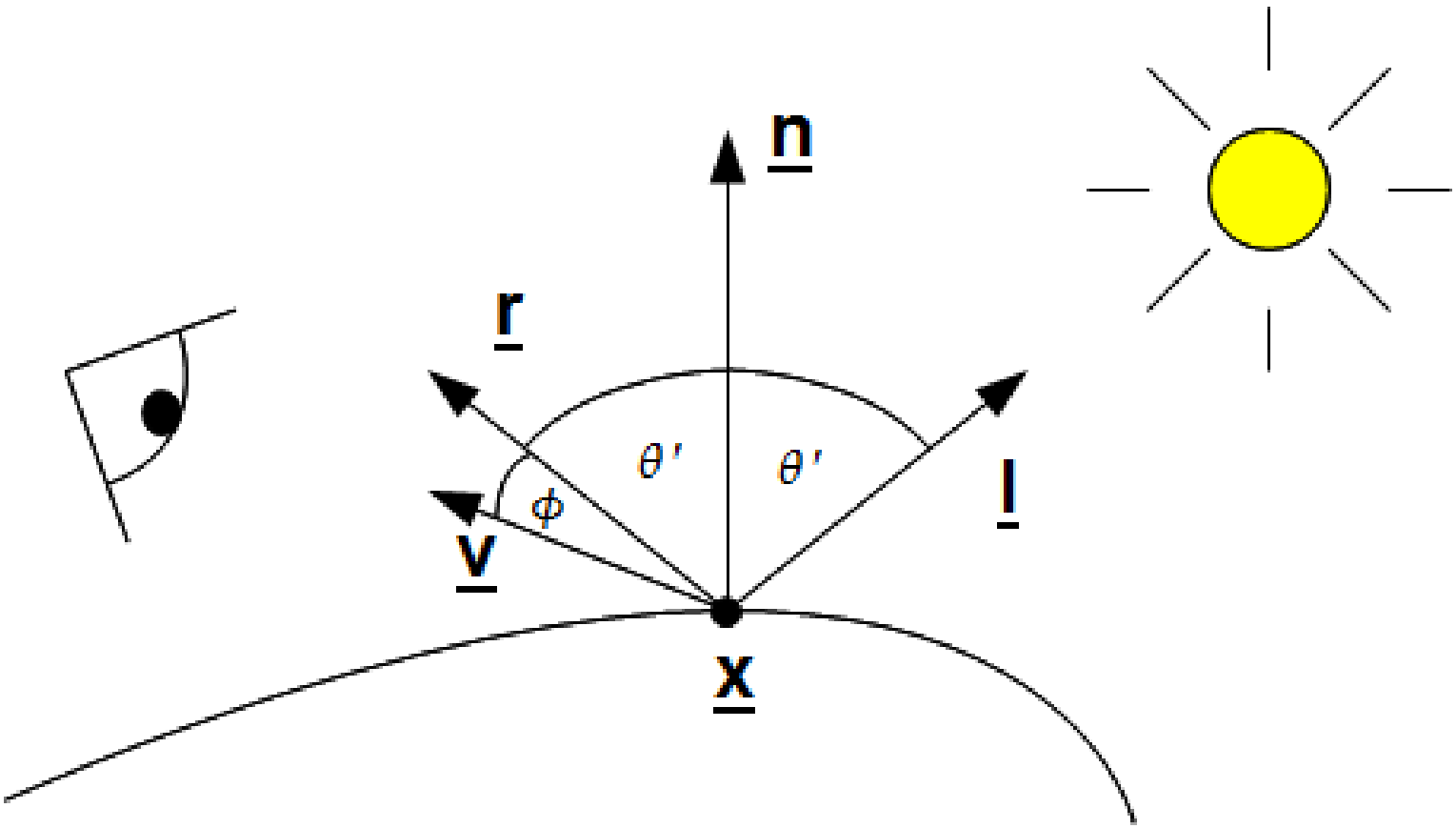
Jelölések



Jelölések



Jelölések



Ambiens tag

- A színtéren mindenütt jelenlevő fénymennyiség.

Ambiens tag

- A színtéren mindenütt jelenlevő fénymennyiség.
- Képlet: $k_a \cdot L_a$.

Ambiens tag

- A színtéren mindenütt jelenlevő fénymennyiség.
- Képlet: $k_a \cdot L_a$.
- k_a a felülettől függ, legyen `float4 ambientColor`.

Ambiens tag

- A színtéren mindenütt jelenlevő fénymennyiség.
- Képlet: $k_a \cdot L_a$.
- k_a a felülettől függ, legyen `float4 ambientColor`.
- A `float4` egy *rgba* négyes, ahol *a* az átlátszóság

Ambiens tag

- A színtéren mindenütt jelenlevő fénymennyiség.
- Képlet: $k_a \cdot L_a$.
- k_a a felülettől függ, legyen `float4 ambientColor`.
- A `float4` egy *rgba* négyes, ahol *a* az átlátszóság
- Itt a k_a számnégyese azt határozza meg, hogy a beeső fényintenzitás hányad részét veri vissza az anyag az R, G, B-nek megfelelő hullámhosszokon

Ambiens tag

- A színtéren mindenütt jelenlevő fénymennyiség.
- Képlet: $k_a \cdot L_a$.
- k_a a felülettől függ, legyen `float4 ambientColor`.
- A `float4` egy *rgba* négyes, ahol *a* az átlátszóság
- Itt a k_a számnégyese azt határozza meg, hogy a beeső fényintenzitás hányad részét veri vissza az anyag az R, G, B-nek megfelelő hullámhosszokon
- L_a fénytől, felülettől független, az állandó háttérmegvilágítás intenzitása az RGB hullámhosszokon. Szintén legyen `float4 ambientLight`, de $a \equiv 1$.

Ambiens tag

- A színtéren mindenütt jelenlevő fénymennyiség.
- Képlet: $k_a \cdot L_a$.
- k_a a felülettől függ, legyen `float4 ambientColor`.
- A `float4` egy *rgba* négyes, ahol *a* az átlátszóság
- Itt a k_a számnégyese azt határozza meg, hogy a beeső fényintenzitás hányad részét veri vissza az anyag az R, G, B-nek megfelelő hullámhosszokon
- L_a fénytől, felülettől független, az állandó háttérmegvilágítás intenzitása az RGB hullámhosszokon. Szintén legyen `float4 ambientLight`, de $a \equiv 1$.
- *Pixel shaderben* használható: `ambientColor*ambientLight`

Lambert-törvény

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

Lambert-törvény

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

Lambert-törvény

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

```
float4 diffuseColor
float4 diffuseLight
```

Lambert-törvény

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

```
float4 diffuseColor
float4 diffuseLight
```

- Kéne még: $\cos^+ \theta'$.

Lambert-törvény

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

```
float4 diffuseColor
float4 diffuseLight
```

- Kéne még: $\cos^+ \theta'$.
- Kiszámítása: `saturate(dot(normal, toLight))`

Lambert-törvény

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

```
float4 diffuseColor
float4 diffuseLight
```

- Kéne még: $\cos^+ \theta'$.
- Kiszámítása: `saturate(dot(normal, toLight))`
- Ismerni kell hozzá `normal` = **n**-t és `toLight` =**l**-t.

Lambert-törvény

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

```
float4 diffuseColor
float4 diffuseLight
```

- Kéne még: $\cos^+ \theta'$.
- Kiszámítása: `saturate(dot(normal, toLight))`
- Ismerni kell hozzá `normal` = **n**-t és `toLight` =**l**-t.
- *Spot* fényforrásnál a fénykört is figyelembe kell majd venni.

Spekuláris visszaverődés - Phong modell

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

Spekuláris visszaverődés - Phong modell

- BRDF: $L_{ref} = L_i k_s (\cos^+ \phi)^n$, ahol ϕ az **r** tükörirány és a **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:

```
float4 specularColor
float4 specularLight
```

Spekuláris visszaverődés - Phong modell

- BRDF: $L_{ref} = L_i k_s (\cos^+ \phi)^n$, ahol ϕ az **r** tükörirány és a **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:

```
float4 specularColor
float4 specularLight
```

- n felület függő konstans, legyen `float specularPower`

Spekuláris visszaverődés - Phong modell

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

- Kéne $\cos^+ \phi$, amihez meg kéne **r** és **v**.

Spekuláris visszaverődés - Phong modell

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

- Kéne $\cos^+ \phi$, amihez meg kéne **r** és **v**.
- **r** a **l** vektor **n**-re vett tükörképe. Számítása:

```
float3 reflection = reflect(-toLight,
normal)
```

Spekuláris visszaverődés - Phong modell

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

- Kéne $\cos^+ \phi$, amihez meg kéne **r** és **v**.
- **r** a **l** vektor **n**-re vett tükörképe. Számítása:
  ```
  float3 reflection = reflect(-toLight,
  normal)
  ```
- **v** a nézeti irány, azaz a feületi pontból a kamerába mutató egységvektor.
  ```
  float3 directionToEye =
  normalize(eyePosition-worldPos)
  ```

Spekuláris visszaverődés - Phong modell

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

- Kéne $\cos^+ \phi$, amihez meg kéne **r** és **v**.
- **r** a **l** vektor **n**-re vett tükörképe. Számítása:
  ```
  float3 reflection = reflect(-toLight,
  normal)
  ```
- **v** a nézeti irány, azaz a feületi pontból a kamerába mutató egységvektor.
  ```
  float3 directionToEye =
  normalize(eyePosition-worldPos)
  ```
- $(\cos^+ \phi)^n$ számítása:
  ```
  pow(saturate(dot(reflection,
  directionToEye)), specularPower)
  ```

Összefoglalva

Összefoglalva

Összefoglalva

Felület tulajdonságai

Összefoglalva

Felület tulajdonságai

- `ambientColor`

Összefoglalva

Felület tulajdonságai

- `ambientColor`
- `diffuseColor`

Összefoglalva

Felület tulajdonságai

- `ambientColor`
- `diffuseColor`
- `specularColor`

Összefoglalva

Felület tulajdonságai

- `ambientColor`
- `diffuseColor`
- `specularColor`
- `specularPower`

Összefoglalva

Felület tulajdonságai

- `ambientColor`
- `diffuseColor`
- `specularColor`
- `specularPower`
- `normal`

Összefoglalva

Felület tulajdonságai

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

Összefoglalva

Felület tulajdonságai

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

Összefoglalva

Felület tulajdonságai

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

Összefoglalva

Felület tulajdonságai

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

Fényforrás tulajdonságai

Összefoglalva

Felület tulajdonságai

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

Fényforrás tulajdonságai

- `diffuseLight`

Összefoglalva

Felület tulajdonságai

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

Fényforrás tulajdonságai

- `diffuseLight`
- `specularLight`

Összefoglalva

Felület tulajdonságai

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

Fényforrás tulajdonságai

- `diffuseLight`
- `specularLight`
- `toLight`

Összefoglalva

Felület tulajdonságai

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

Fényforrás tulajdonságai

- `diffuseLight`
- `specularLight`
- `toLight`

Összefoglalva

Felület tulajdonságai

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

Fényforrás tulajdonságai

- `diffuseLight`
- `specularLight`
- `toLight`

Színtér tulajdonságai

Összefoglalva

Felület tulajdonságai

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

Fényforrás tulajdonságai

- `diffuseLight`
- `specularLight`
- `toLight`

Színtér tulajdonságai

- `ambientLight`

Összefoglalva

Felület tulajdonságai

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

Fényforrás tulajdonságai

- `diffuseLight`
- `specularLight`
- `toLight`

Színtér tulajdonságai

- `ambientLight`
- `eyePosition`

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!

Tartalom

Adatstruktúrák

```
struct VS_INPUT
{
    float4 pos      : POSITION;
    float3 normal   : NORMAL;
};

struct VS_OUTPUT
{
    float4 pos      : POSITION;
    float3 normal   : TEXCOORD0;
    float3 worldPos : TEXCOORD1;
};
```

Vertex shader

```
VS_OUTPUT LightingVS(VS_INPUT input)
{
  VS_OUTPUT output;
  output.pos = mul(input.pos, WorldViewProjection);
  output.worldPos = mul(input.pos, World).xyz;
  output.normal = mul(InvWorld, float4(input.normal

  return output;
}
```

Pixel shader

```
float4 LightingPS(float3 normal   : TEXCOORD0,
                  float3 worldPos : TEXCOORD1)
                    : COLOR0
{
  normal = normalize(normal);
  float3 toLight = ??? // Fenyforras fuggo
  float diffuseIntensity =
    saturate(dot(normal, toLight));
  float3 reflection = reflect(-toLight, normal);
  float3 directionToEye =
    normalize(eyePosition-worldPos);
  // ...
```

Pixel shader

```
    // ...
    float specularIntensity =
      pow(saturate(dot(reflection, directionToEye)),
        specularPower);
    if (diffuseIntensity <= 0)
      specularIntensity = 0;

    return ambientColor*ambientLight +
diffuseIntensity*diffuseColor*diffuseLight +
specularIntensity*specularColor*specularLight;
}
```

`toLight` számítása

- Irány fényforrás

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - `toLight = -lightDirection`

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - `toLight = -lightDirection`
- Pont fényforrás

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - `toLight = -lightDirection`
- Pont fényforrás
  - Fény pozíciója, helyvektor: `float3 lightPosition`

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - `toLight = -lightDirection`
- Pont fényforrás
  - Fény pozíciója, helyvektor: `float3 lightPosition`
  - `toLight = normalize(lightPosition-worldPos)`

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - `toLight = -lightDirection`
- Pont fényforrás
  - Fény pozíciója, helyvektor: `float3 lightPosition`
  - `toLight = normalize(lightPosition-worldPos)`
- *Spot* fényforrás

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - `toLight = -lightDirection`
- Pont fényforrás
  - Fény pozíciója, helyvektor: `float3 lightPosition`
  - `toLight = normalize(lightPosition-worldPos)`
- *Spot* fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - `toLight = -lightDirection`
- Pont fényforrás
  - Fény pozíciója, helyvektor: `float3 lightPosition`
  - `toLight = normalize(lightPosition-worldPos)`
- *Spot* fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - Fény pozíciója, helyvektor: `float3 lightPosition`

`toLight` számítása

- Irány fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - `toLight = -lightDirection`
- Pont fényforrás
  - Fény pozíciója, helyvektor: `float3 lightPosition`
  - `toLight = normalize(lightPosition-worldPos)`
- *Spot* fényforrás
  - Fény iránya, normalizált irányvektor: `float3 lightDirection`
  - Fény pozíciója, helyvektor: `float3 lightPosition`
  - `toLight = normalize(lightPosition-worldPos)` mint pont fényforrás esetén

Spot fényforrás hatása

- Két extra paraméter:

Spot fényforrás hatása

- Két extra paraméter:
  - belső fénykör: amin belül teljes intenzitással hat

Spot fényforrás hatása

- Két extra paraméter:
  - belső fénykör: amin belül teljes intenzitással hat
  - külső fénykör: amin kívül abszolút nem hat

Spot fényforrás hatása

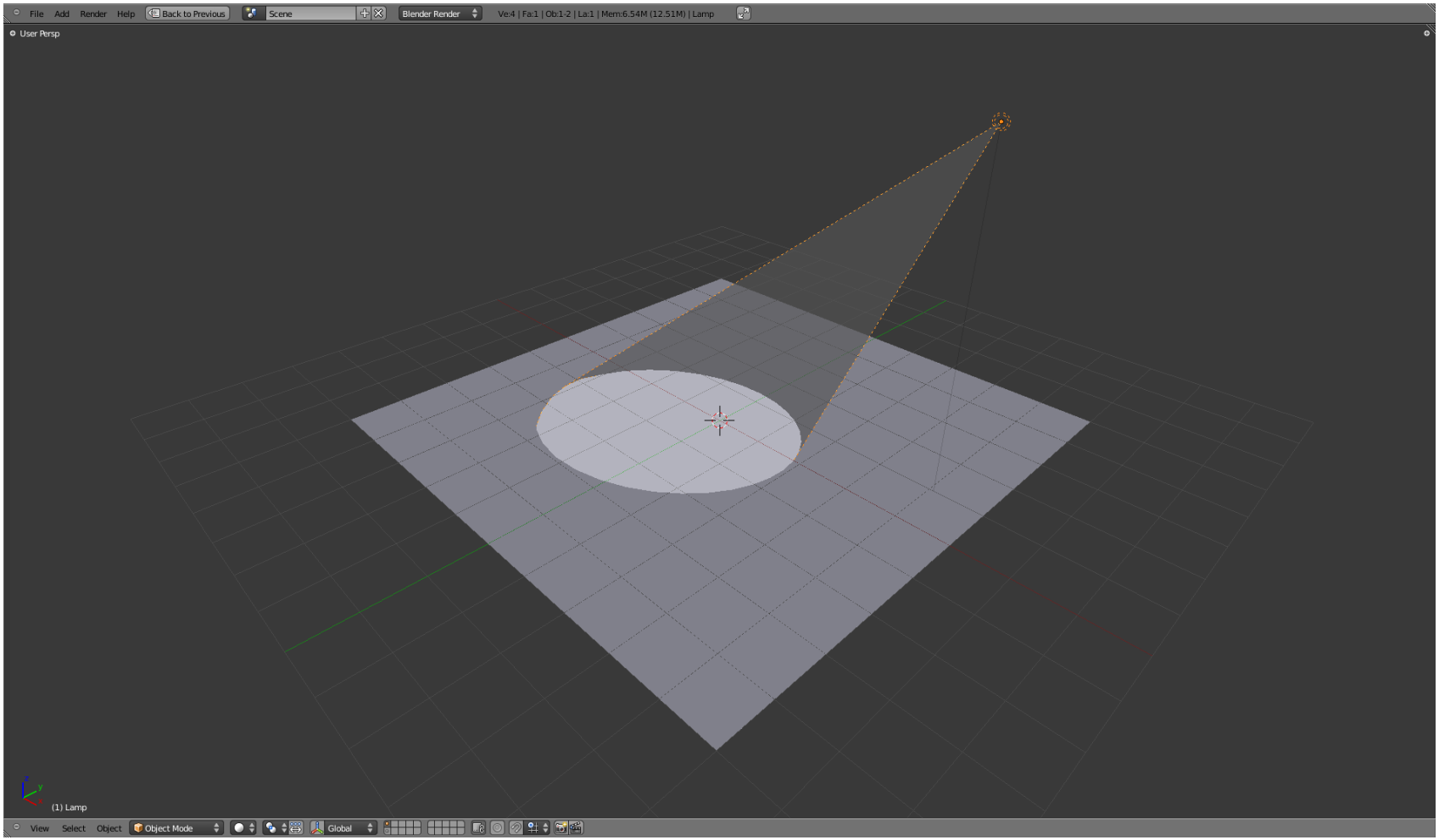
- Két extra paraméter:
  - belső fénykör: amin belül teljes intenzitással hat
  - külső fénykör: amin kívül abszolút nem hat
- A kettő között folyamatosan csökken a fény intenzitása.

Spot fényforrás hatása

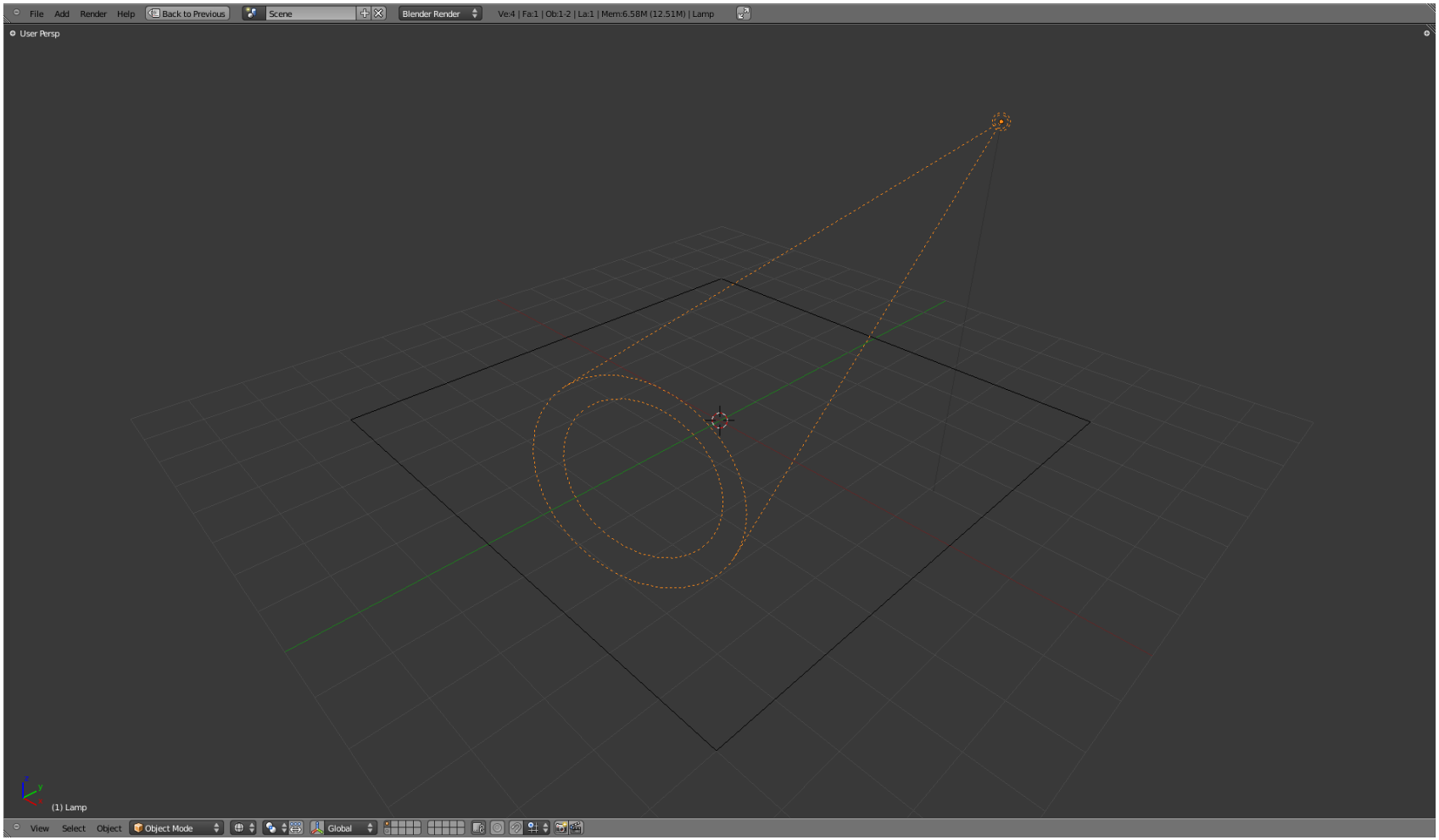
- Két extra paraméter:
  - belső fénykör: amin belül teljes intenzitással hat
  - külső fénykör: amin kívül abszolút nem hat
- A kettő között folyamatosan csökken a fény intenzitása.
- A fényköröket a fényforrásból induló, a fény irányával megeggyező állású (végtelen)kúpoknak tekintjük.

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 fénysugarakat a fényforrásból induló, a fény irányával megegyező állású (végtelen)kúpoknak tekintjük.
- Egy felületi pont akkor van benne egy kúpban, ha a pontot a fényforrás poziciójával összekötő egyenes a kúpon belül van.


File Add Render Help Back to Previous Scene Blender Render Ve:4 | Fa:1 | Ob:1-2 | La:1 | Mem:6.54M (12.51M) | Lamp
User Persp
(1) Lamp
View Select Object Object Mode Global

Shader programok



Spot fényforrás hatása

- Az előbbi szakasz iránya pont a `toLight`.

Spot fényforrás hatása

- Az előbbi szakasz iránya pont a `toLight`.
- Ha a `lightDirection` 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

- Az előbbi szakasz iránya pont a `toLight`.
- Ha a `lightDirection` 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 $\leq 180°$.

Spot fényforrás hatása

- Az előbbi szakasz iránya pont a `toLight`.
- Ha a `lightDirection` 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 $\leq 180°$.
- Adjuk meg a két fénykört, a hozzájuk tartozó kúpok nyílásszögének felének cos-áva:

Spot fényforrás hatása

- Az előbbi szakasz iránya pont a `toLight`.
- Ha a `lightDirection` 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 $\leq 180°$.
- Adjuk meg a két fénykört, a hozzájuk tartozó kúpok nyílásszögének felének cos-áva:
  - belső fénykör: `cosInnerCone`

Spot fényforrás hatása

- Az előbbi szakasz iránya pont a `toLight`.
- Ha a `lightDirection` 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 $\leq 180°$.
- Adjuk meg a két fénykört, a hozzájuk tartozó kúpok nyílásszögének felének cos-áva:
  - belső fénykör: `cosInnerCone`
  - külső fénykör: `cosOutterCone`

Spot fényforrás hatása

- `smoothstep(min, max, x)`:

Spot fényforrás hatása

- `smoothstep(min, max, x)`:
  - 0, ha `x<min`

Spot fényforrás hatása

- `smoothstep(min, max, x)`:
  - 0, ha `x<min`
  - 1, ha `x>max`

Spot fényforrás hatása

- `smoothstep(min, max, x)`:
  - 0, ha `x<min`
  - 1, ha `x>max`
  - lin. átmenet 0 és 1 között kül.

Spot fényforrás hatása

- `smoothstep(min, max, x)`:
  - 0, ha `x<min`
  - 1, ha `x>max`
  - lin. átmenet 0 és 1 között kül.
- `float spotFactor = smoothstep(cosOutterCone, cosInnerCone, dot(lightDirection, -toLight))`

Spot fényforrás hatása

- `smoothstep(min, max, x)`:
  - 0, ha `x<min`
  - 1, ha `x>max`
  - lin. átmenet 0 és 1 között kül.
- `float spotFactor = smoothstep(cosOutterCone, cosInnerCone, dot(lightDirection, -toLight))`
- Ezzel kell beszorozni a diffúz és a spekuláris tagot.

Spot fényforrás hatása

- `smoothstep(min, max, x)`:
  - 0, ha `x<min`
  - 1, ha `x>max`
  - lin. átmenet 0 és 1 között kül.
- `float spotFactor = smoothstep(cosOutterCone, cosInnerCone, dot(lightDirection, -toLight))`
- Ezzel kell beszorozni a diffúz és a spekuláris tagot.
- Pl. `diffuseIntensity`-t és `specularIntensity`

Tartalom

Virtuális világ tárolása - kérdések

- Hol táruljuk az adatokat? *Mem.* vagy HDD?

Virtuális világ tárolása - kérdések

- Hol táruljuk az adatokat? *Mem.* vagy HDD?
- Mire optimalizálunk? Rajzolás vagy szerkesztés?

Virtuális világ tárolása - kérdések

- Hol táruljuk az adatokat? *Mem.* vagy HDD?
- Mire optimalizálunk? Rajzolás vagy szerkesztés?
- Milyen koordináta-rendszerben tároljuk őket? Világ vagy modell kr?

Tartalom

Geometria *"brute force"* tárolása

- Legyenek a primitíveink poligonok (bejárási irány megegyezés szerint)

Geometria *"brute force"* tárolása

- Legyenek a primitíveink poligonok (bejárási irány megegyezés szerint)
- Legyünk lusták, és minden poligont tartsuk nyilván az összes csúcsával.

Geometria *"brute force"* tárolása

- Legyenek a primitíveink poligonok (bejárási irány megegyezés szerint)
- Legyünk lusták, és minden poligont tartsuk nyilván az összes csúcsával.
- Poligonokkal kapcsolatos feladatok:
  - tárolás
  - transzformálás

Geometria *"brute force"* tárolása

- Legyenek a primitíveink poligonok (bejárási irány megegyezés szerint)
- Legyünk lusták, és minden poligont tartsuk nyilván az összes csúcsával.
- Poligonokkal kapcsolatos feladatok:
  - tárolás
  - transzformálás
  - szomszédsági lekérdezések

Geometria *"brute force"* tárolása

```
struct triangle {
  float x1,y1,z1;
  float x2,y2,z2;
  float x3,y3,z3;
};
```

A *"brute force"* tárolás elemzése

- *Tárolás:* ha vannak poligonoknak közös csúcsai, akkor ezeket többször tároljuk – feleslegesen $\rightarrow$ nem túl jó

A *"brute force"* tárolás elemzése

- *Tárolás:* ha vannak poligonoknak közös csúcsai, akkor ezeket többször tároljuk – feleslegesen $\rightarrow$ nem túl jó
- *Transzformálás:* a közös csúcsokra annyiszor fogjuk elvégezni a transzformációkat, ahányszor szerepelnek $\rightarrow$ nem hatékony

A *"brute force"* tárolás elemzése

- *Tárolás:* ha vannak poligonoknak közös csúcsai, akkor ezeket többször tároljuk – feleslegesen → nem túl jó
- *Transzformálás:* a közös csúcsokra annyiszor fogjuk elvégezni a transzformációkat, ahányszor szerepelnek → nem hatékony
- *Lekérdezések:* fogalmunk sincs, ki kinek a szomszédja, csak az összes csúcs bejárásával tudunk eredményre jutni → katasztrófa

A *"brute force"* tárolás elemzése

- *Tárolás:* ha vannak poligonoknak közös csúcsai, akkor ezeket többször tároljuk – feleslegesen $\rightarrow$ nem túl jó
- *Transzformálás:* a közös csúcsokra annyiszor fogjuk elvégezni a transzformációkat, ahányszor szerepelnek $\rightarrow$ nem hatékony
- *Lekérdezések:* fogalmunk sincs, ki kinek a szomszédja, csak az összes csúcs bejárásával tudunk eredményre jutni $\rightarrow$ katasztrófa
- Egyetlen előnye, hogy ennél egyszerűbben már nem is lehetne tárolni.

Index pufferek

- Alapötlet: tároljunk minden csúcsot egyszer, egy nagy közös tömbben!

Index pufferek

- Alapötlet: tároljunk minden csúcsot egyszer, egy nagy közös tömbben!
- A poligonok csak hivatkozzanak a csúcsok tömbjének elemeire.

*Index puffer*ek

- Alapötlet: tároljunk minden csúcsot egyszer, egy nagy közös tömbben!
- A poligonok csak hivatkozzanak a csúcsok tömbjének elemeire.
- Ez az *index puffer*.

*Index puffer*ek

- Alapötlet: tároljunk minden csúcsot egyszer, egy nagy közös tömbben!
- A poligonok csak hivatkozzanak a csúcsok tömbjének elemeire.
- Ez az *index puffer*.
- Minden GPU támogatja.

Index buffer-ek

```
struct triangle {
  unsigned int a,b,c;
};

struct vec3 {
  float x,y,z;
};

std::vector<vec3> vertexBuffer;
std::vector<int> indexBuffer;
```

Példa

- Vegyünk egy $N \times N$ db négyzetből álló rácsot! Hány darab csúcsot kell tárolni?

Példa

- Vegyünk egy $N \times N$ db négyzetből álló rácsot! Hány darab csúcsot kell tárolni?
- Mérete *index puffer* nélkül: 4 csúcs/négyzet, $N \times N$ négyzet: $4N^2$ *csúcspont*.

Példa

- Vegyünk egy $N \times N$ db négyzetből álló rácsot! Hány darab csúcsot kell tárolni?
- Mérete *index puffer* nélkül: 4 csúcs/négyzet, $N \times N$ négyzet: $4N^2$ *csúcspont*.
- Mérete *index puffer*-rel: $(N+1) \times (N+1) = N^2 + 2N + 1$ *csúcs* (+ $4N^2$ egész szám, *index*)

Példa

- Vegyünk egy $N \times N$ db négyzetből álló rácsot! Hány darab csúcsot kell tárolni?
- Mérete *index puffer* nélkül: 4 csúcs/négyzet, $N \times N$ négyzet: $4N^2$ *csúcspont*.
- Mérete *index puffer*-rel: $(N+1) \times (N+1) = N^2 + 2N + 1$ *csúcs* (+ $4N^2$ egész szám, *index*)
- $4N^2$ vs. $N^2 + 2N + 1$

$$\begin{aligned} 4N^2 &> N^2 + 2N + 1 \\ 0 &> -3N^2 + 2N + 1 \\ N &> 1, \text{ha } N \in \mathbb{Z}^+ \end{aligned}$$

Példa – folyt.

- Ha több mint egyetlen négyzetünk van, már megéri.

Példa – folyt.

- Ha több mint egyetlen négyzetünk van, már megéri.
- Pl. ha $N = 10$
- Mérete *index puffer* nélkül: 400 csúcsot tárolunk és transzformálunk
- Mérete *index puffer*-rel: 121 csúcsot tárolunk és transzformálunk

index puffer-ek GPU-n

- Minden videókártya, amit még nem gyűjtenek a múzeumok támogatja az *index puffer*-eket.

index puffer-ek GPU-n

- Minden videókártya, amit még nem gyűjtenek a múzeumok támogatja az *index puffer*-eket.
- A csúcspontok tömbje (*vertex buffer*) nem csak pozicókat tartalmaz, hanem normálvektorokat, textúra-koordinátákat, és még sok mást.

index puffer-ek GPU-n

- Minden videókártya, amit még nem gyűjtenek a múzeumok támogatja az *index puffer*-eket.
- A csúcspontok tömbje (*vertex buffer*) nem csak pozicókat tartalmaz, hanem normálvektorokat, textúra-koordinátákat, és még sok mást.
- Egy hivatkozás a *vertex buffer*-ra mindezekre együtt hivatkozik.

Kocka probléma

- Ha egy kockát rakunk össze háromszögekből, akkor egy sarokban lévő csúcs min. három, max. hat háromszögnek a csúcsa.

Kocka probléma

- Ha egy kockát rakunk össze háromszögekből, akkor egy sarokban lévő csúcs min. három, max. hat háromszögnek a csúcsa.
- *index puffer*-rel elég lenne 8 csúcsot nyilvántartani.

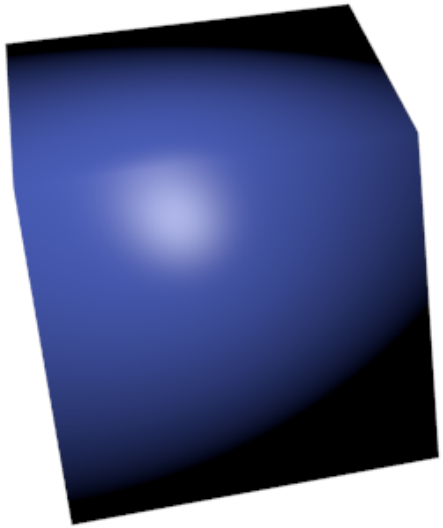
Kocka probléma

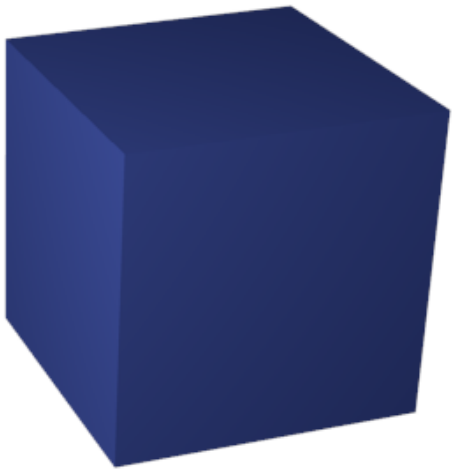
- Ha egy kockát rakunk össze háromszögekből, akkor egy sarokban lévő csúcs min. három, max. hat háromszögnek a csúcsa.
- *index puffer*-rel elég lenne 8 csúcsot nyilvántartani.
- Mi lesz a normálisokkal? Hogy éles sarkai legyenek a kockának a normálisoknak merőlegesnek kell lennie a lapokra!

Kocka probléma

- Ha egy kockát rakunk össze háromszögekből, akkor egy sarokban lévő csúcs min. három, max. hat háromszögnek a csúcsa.
- *index puffer*-rel elég lenne 8 csúcsot nyilvántartani.
- Mi lesz a normálisokkal? Hogy éles sarkai legyenek a kockának a normálisoknak merőlegesnek kell lennie a lapokra!
- Oldalanként külön meg kell adni a csúcsokat: összesen 3×8 csúcs kerül az *vertex buffer*-ba.

Kocka probléma





Kocka probléma

- A probléma: bár egyetlen térbeli pontot jelöl egy sarok, valójában három különböző felület *felületi pontja*

Kocka probléma

- A probléma: bár egyetlen térbeli pontot jelöl egy sarok, valójában három különböző felület *felületi pontja*
- Mivel a csúcspontokban pozíciókon kívül felületi tulajdonságokat is tárolunk (normális), ezért ilyenkor még index pufferrel sem spróolhatjuk ki a redundanciát!

Az *index puffer*es tárolás elemzése

- *Tárolás:* ált. hatékony.

Az *index puffer*es tárolás elemzése

- *Tárolás:* ált. hatékony.
- *Transzformálás:* hatékony.

Az *index puffer*es tárolás elemzése

- *Tárolás:* ált. hatékony.
- *Transzformálás:* hatékony.
- *Lekérdezések:* közös csúcsokat már tudunk, de igazából még mindig fogalmunk sincs.

Tartalom

Szomszédsági viszonyok

- Néha kellenek a szomszédok, pl. felület-felosztásoknál, degenerált primitívek kiszűrésekor, egyes felhasználói inputok kezelésekor stb.

Szomszédsági viszonyok

- Néha kellenek a szomszédok, pl. felület-felosztásoknál, degenerált primitívek kiszűrésekor, egyes felhasználói inputok kezelésekor stb.
- Ismertek a csúcsok $\Rightarrow$ számítható mi, minek a szomszédja!

Szomszédsági viszonyok

- Néha kellenek a szomszédok, pl. felület-felosztásoknál, degenerált primitívek kiszűrésekor, egyes felhasználói inputok kezelésekor stb.
- Ismertek a csúcsok $\Rightarrow$ számítható mi, minek a szomszédja!
- Egy csúcsban tetszőleges számú poligon találkozhat $\Rightarrow$ dinamikus adatszerkezet kéne

Szomszédsági viszonyok

- Néha kellenek a szomszédok, pl. felület-felosztásoknál, degenerált primitívek kiszűrésekor, egyes felhasználói inputok kezelésekor stb.
- Ismertek a csúcsok $\Rightarrow$ számítható mi, minek a szomszédja!
- Egy csúcsban tetszőleges számú poligon találkozhat $\Rightarrow$ dinamikus adatszerkezet kéne
- Jobb megoldás: Szárnyas-él (*winged-edge*) adatszerkezet!

Szárnyas-él adatszerkezet

- Az alakzatokat határukkal leíró (*B-rep*) (boundary representation) reprezentáció egyik gyakran használt topológiatároló adatszerkezete manifold poliéderek esetén

Szárnyas-él adatszerkezet

- Az alakzatokat határukkal leíró (*B-rep*) (boundary representation) reprezentáció egyik gyakran használt topológiatároló adatszerkezete manifold poliéderek esetén
- Tárolás során *csúcsokat*, *éleket* és *lapokat* különböztet meg

Szárnyas-él adatszerkezet

- Az alakzatokat határukkal leíró (*B-rep*) (boundary representation) reprezentáció egyik gyakran használt topológiatároló adatszerkezete manifold poliéderek esetén
- Tárolás során *csúcsokat*, *éleket* és *lapokat* különböztet meg
- Az élek szempontjából tároljuk a felületet

Szárnyas-él adatszerkezet

- Az alakzatokat határukkal leíró (*B-rep*) (boundary representation) reprezentáció egyik gyakran használt topológiatároló adatszerkezete manifold poliéderek esetén
- Tárolás során *csúcsokat*, *éleket* és *lapokat* különböztet meg
- Az élek szempontjából tároljuk a felületet
- Minden élhez fix számú adat tartozik

Szárnyas-él adatszerkezet

- Az alakzatokat határukkal leíró (*B-rep*) (boundary representation) reprezentáció egyik gyakran használt topológiatároló adatszerkezete manifold poliéderek esetén
- Tárolás során *csúcsokat*, *éleket* és *lapokat* különböztet meg
- Az élek szempontjából tároljuk a felületet
- Minden élhez fix számú adat tartozik
- Segítségével pl. gyorsan körbe lehet járni egy poligon éleit, közben megkapva minden szomszédot

Szárnyas-él adatszerkezet

- Minden lapot egy élsorozat határol - minden laphoz tároljunk egy, az élsorozatához tartozó tetszőleges élre mutató pointert

Szárnyas-él adatszerkezet

- Minden lapot egy élsorozat határol - minden laphoz tároljunk egy, az élsorozatához tartozó tetszőleges élre mutató pointert
- A csúcspontok élekhez illeszkednek (vagy belőle indul ki, vagy ő a célja) - tároljuk valamelyiket a csúcsokhoz!

Egyetlen él adatai

- Egy él két csúcsot köt össze - tároljuk ezeket az élben

Egyetlen él adatai

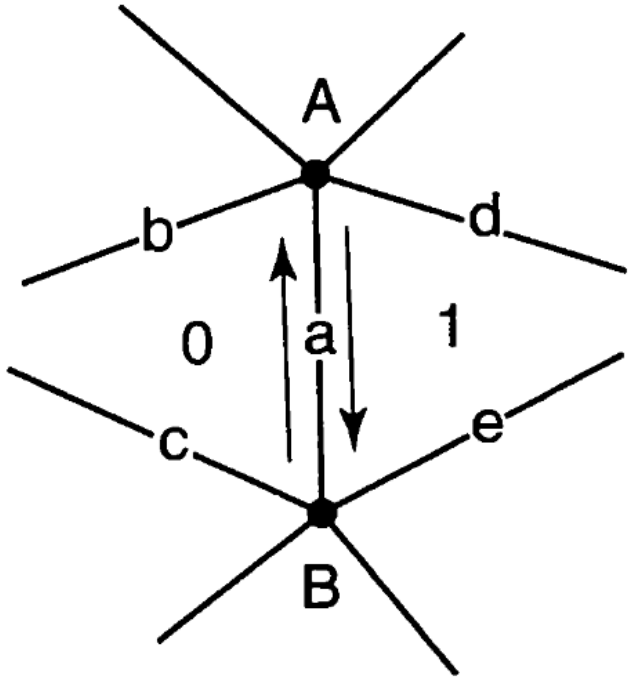
- Egy él két csúcsot köt össze - tároljuk ezeket az élben
- Egy él legfeljebb két laphoz tartozhat - az egyik a baloldali, a másik a jobboldali lap lesz, ezekre mutató pointereket (vagy indexeket) tárolunk

Egyetlen él adatai

- Egy él két csúcsot köt össze - tároljuk ezeket az élben
- Egy él legfeljebb két laphoz tartozhat - az egyik a baloldali, a másik a jobboldali lap lesz, ezekre mutató pointereket (vagy indexeket) tárolunk
- A fenti két lapon egyúttal egy-egy élsorozat (az adott lapot alkotó élsorozat) része is az adott él - mindkét élsorozatban tároljuk a rákövetkezőjét és megelőzőjét *az adott élnek az adott lap bejárási irányának megfelelően* (!)

Egyetlen él adatai

2*él	csúcs		lap		balra		jobbra	
	start	vég	bal	jobb	előző	köv.	előző	köv.
a	B	A	0	1	c	b	d	e



Egyéb táblázatok

Egyéb táblázatok

Egyéb táblázatok

Csúcsok táblája

Egyéb táblázatok

Csúcsok táblája

- csúcs ID

Egyéb táblázatok

Csúcsok táblája

- csúcs ID
- csúcsból induló él

Egyéb táblázatok

Csúcsok táblája

- csúcs ID
- csúcsból induló él

Egyéb táblázatok

Csúcsok táblája

- csúcs ID
- csúcsból induló él

Lapok táblája

Egyéb táblázatok

Csúcsok táblája

- csúcs ID
- csúcsból induló él

Lapok táblája

- lap ID

Egyéb táblázatok

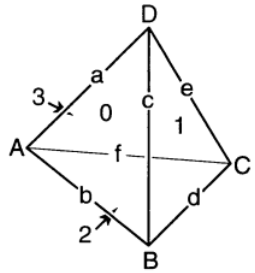
Csúcsok táblája

- csúcs ID
- csúcsból induló él

Lapok táblája

- lap ID
- lap egy éle

Példa: tetraéder



edge	vertex 1	vertex 2	face left	face right	pred left	succ left	pred right	succ right
a	A	D	3	0	f	e	c	b
b	A	B	0	2	a	c	d	f
c	B	D	0	1	b	a	e	d
d	B	C	1	2	c	e	f	b
e	C	D	1	3	d	c	a	f
f	C	A	3	2	e	a	b	d

vertex	edge
A	a
B	d
C	d
D	e

face	edge
0	a
1	c
2	d
3	a

Shirley, Fundamentals of Computer Graphics

Pl.: Egy lap összes szomszéd lapjának felsorolása

```
def allNeighbours(face, edges, vertices, faces):
  startEdge = faces[face]
  edge = startEdge
  if edges[startEdge].faceLeft == face:
    while edges[edge].succLeft != startEdge:
      print edges[edge].faceRight
      edge = edges[edge].succLeft
  else:
    while edges[edge].succRight != startEdge:
      print edges[edge].faceLeft
      edge = edges[edge].succRight
```

Pl.: Egy lap összes szomszéd lapjának felsorolása

- Azaz: induljunk el az adott lap reprezentáns éléből (amit tárolunk a laphoz)

Pl.: Egy lap összes szomszéd lapjának felsorolása

- Azaz: induljunk el az adott lap reprezentáns éléből (amit tárolunk a laphoz)
- Ha ennek az élnek a baloldali lapja az adott lap: iteráljunk végig a baloldali éllistán és írjuk ki a jobboldali lapokat (a baloldali a lekérdezést kiváltó lap)

Pl.: Egy lap összes szomszéd lapjának felsorolása

- Azaz: induljunk el az adott lap reprezentáns éléből (amit tárolunk a laphoz)
- Ha ennek az élnek a baloldali lapja az adott lap: iteráljunk végig a baloldali éllistán és írjuk ki a jobboldali lapokat (a baloldali a lekérdezést kiváltó lap)
- Különben a jobboldali lap az adott lap, iteráljunk végig a jobboldali lap éllistáján

Pl.: Egy lap összes szomszéd lapjának felsorolása

- Azaz: induljunk el az adott lap reprezentáns éléből (amit tárolunk a laphoz)
- Ha ennek az élnek a baloldali lapja az adott lap: iteráljunk végig a baloldali éllistán és írjuk ki a jobboldali lapokat (a baloldali a lekérdezést kiváltó lap)
- Különben a jobboldali lap az adott lap, iteráljunk végig a jobboldali lap éllistáján
- Az iteráció érjen véget, amint visszaérünk az adott lap reprezentáns élébe

Pl.: Egy adott csúcsot tartalmazó összes lap felsorolása

```
def allFaces(vertex, edges, vertices, faces):
  startEdge = vertices[vertex]
  edge = startEdge
  done = False
  while not done:
    if edges[edge].vertStart == vertex:
      print edges[edge].faceLeft
      edge = edges[edge].predLeft
    else:
      print edges[edge].faceRight
      edge = edges[edge].predRight
```

Fél-él adatszerkezet

- A winged-edge élét vegyük szét két fél-élre!

Fél-él adatszerkezet

- A winged-edge élét vegyük szét két fél-élre!
- $\rightarrow$ lényegében az élek lapra vett vetületével dolgozunk!

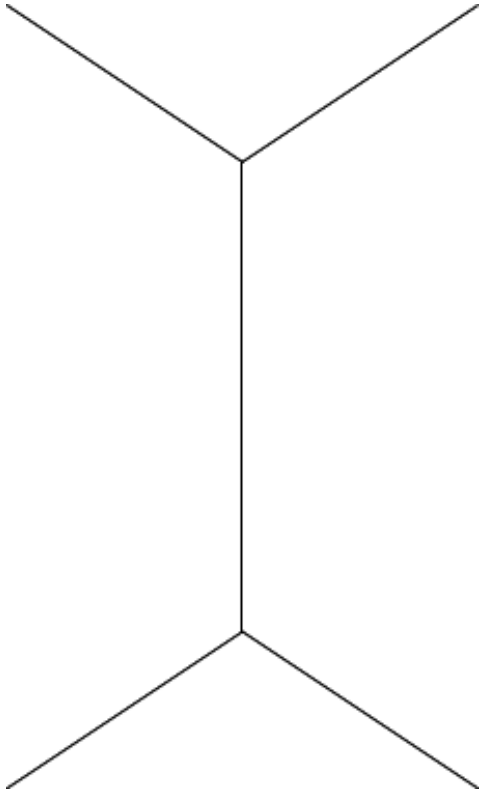
Fél-él adatszerkezet

- A winged-edge élét vegyük szét két fél-élre!
- $\rightarrow$ lényegében az élek lapra vett vetületével dolgozunk!
- A fél-élhez csak egy lap tartozhat + meg kell jegyeznünk a testvér fél-élét (az adott él másik oldali lapra vett vetületét)

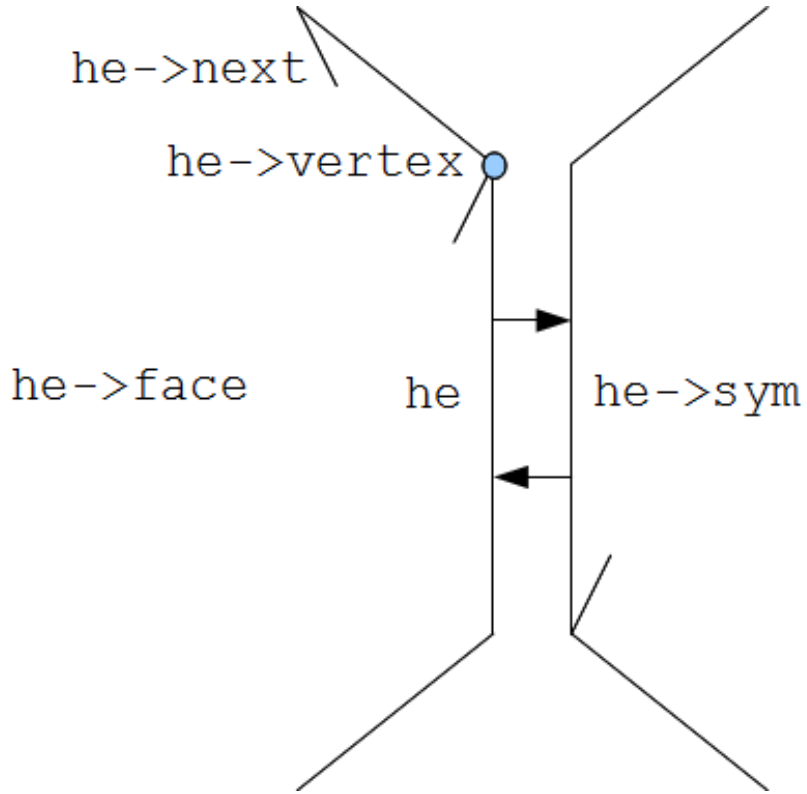
Fél-él adatszerkezet

- A winged-edge élét vegyük szét két fél-élre!
- $\rightarrow$ lényegében az élek lapra vett vetületével dolgozunk!
- A fél-élhez csak egy lap tartozhat + meg kell jegyeznünk a testvér fél-élét (az adott él másik oldali lapra vett vetületét)
- A reprezentáció központi eleme a fél-él

Half-edge



Half-edge



Fél-él adatszerkezet

- Szigorú értelemben véve egy fél-élhez pontosan egy csúcs, él és lap tartozik (de gyakorlatban ennél többet tárolni hasznos lehet)

Fél-él adatszerkezet

- Szigorú értelemben véve egy fél-élhez pontosan egy csúcs, él és lap tartozik (de gyakorlatban ennél többet tárolni hasznos lehet)
- A következőt tároljuk egy fél-élben: az fél-él cél csúcspontja (vertex), a fél-él testvére (sym), a fél-él lapja (face) és a lapot körbefogó fél-élsorozatban a rákövetkezője (next)

Fél-él adatszerkezet

- Szigorú értelemben véve egy fél-élhez pontosan egy csúcs, él és lap tartozik (de gyakorlatban ennél többet tárolni hasznos lehet)
- A következőt tároljuk egy fél-élben: az fél-él cél csúcspontja (vertex), a fél-él testvére (sym), a fél-él lapja (face) és a lapot körbefogó fél-élsorozatban a rákövetkezője (next)
- A lapokhoz egy tetszőleges alkotó fél-él pointerét jegyezzük fel

Fél-él adatszerkezet

- Szigorú értelemben véve egy fél-élhez pontosan egy csúcs, él és lap tartozik (de gyakorlatban ennél többet tárolni hasznos lehet)
- A következőt tároljuk egy fél-élben: az fél-él cél csúcspontja (vertex), a fél-él testvére (sym), a fél-él lapja (face) és a lapot körbefogó fél-élsorozatban a rákövetkezője (next)
- A lapokhoz egy tetszőleges alkotó fél-él pointerét jegyezzük fel
- A csúcspontokhoz egy befutó fél-élt

Fél-él adatszerkezet

- Szigorú értelemben véve egy fél-élhez pontosan egy csúcs, él és lap tartozik (de gyakorlatban ennél többet tárolni hasznos lehet)
- A következőt tároljuk egy fél-élben: az fél-él cél csúcspontja (vertex), a fél-él testvére (sym), a fél-él lapja (face) és a lapot körbefogó fél-élsorozatban a rákövetkezője (next)
- A lapokhoz egy tetszőleges alkotó fél-él pointerét jegyezzük fel
- A csúcspontokhoz egy befutó fél-élt
- $\rightarrow$ HE$\rightarrow$sym$\rightarrow$sym = HE, HE$\rightarrow$next$\rightarrow$sym$\rightarrow$vertex = HE$\rightarrow$vertex stb.

Pl.: Adott lap körbejárása

```
def faceLoop(HE):
  loop = HE

  do:
    print HE
    loop = HE.next
  while loop != HE
```