LÁTHATÓ FELÜLETEK MEGHATÁROZÁSA

Kétváltozós függvények ábrázolása

A látható felszín meghatározására szolgáló általános algoritmusok

Látható felszín algoritmusok

1

LÁTHATÓ FELÜLETEK MEGHATÁROZÁSA

Adott 3D tárgyak egy halmaza, és egy projekció specifikációja.

Mely vonalak és felületek lesznek láthatók?
Melyek lesznek takarva?
Nehéz feladat (időigényes)

Kétféle megközelítés:

2

LÁTHATÓ FELÜLETEK MEGHATÁROZÁSA

```
1. for minden képpontra do begin
     határozzuk meg azt a tárgyat,
     amelyet a nézőpontból a
     képponton keresztül húzott
     egyenes leghamarabb metsz;
       rajzoljuk ki a képpontot a
         megfelelő színben
   end
```

A szükséges idő: $O(np)$
n: a tárgyak száma
p: a képpontok száma

3

LÁTHATÓ FELÜLETEK MEGHATÁROZÁSA

```
2. for minden tárgyra do begin
     határozzuk meg a tárgynak
     azokat a részeit, amelyek
     nincsenek takarásban saját
     maga vagy más tárgyak által;
     ezeket a részeket rajzoljuk
     ki a megfelelő színnel
   end
```

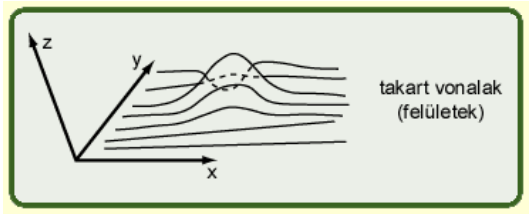
A szükséges idő: $O(n^2)$

4

Kétváltozós függvények ábrázolása

plotterrel

$y = f(x,z)$



5

Kétváltozós függvények ábrázolása

Tegyük fel, hogy f-et egy m x n-es Y mátrixszal közelíthetjük.

Drótvázas rajzot készíthetünk szakaszonként lineáris görbéket előállítva x és z irányban is.

Keressünk olyan algoritmust, amely a takart vonalakat nem rajzolja ki.

6

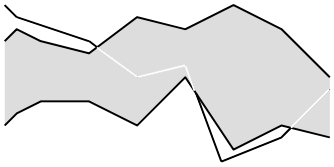
Kétváltozós függvények ábrázolása

1. Ha csak az *x*-tengellyel párhuzamos egyenesek menti értékeket kötjük össze:
   - Haladjunk elölről hátra (a távolabbi vonalak irányába, csak arra kell vigyázni, hogy a már megrajzolt látható felületeket ne "keresztezzük".
   - Elegendő az eddig rajzolt vonalak "sziluettjét" őrizni: csak az látható az új vonalból, ami ez alatt vagy fölött van.
   - Tároljuk minden törésponthoz az eddig rajzolt vonalak maximális és minimális *y* értékét (sziluett), és az új vonal *y* értékeinek megfelelően módosítsuk

Horizont-vonal algoritmus

7

Kétváltozós függvények ábrázolása

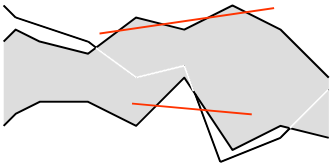


Ha az új vonal valamely szakaszának mindkét végpontja láthatatlan, akkor a szakasz sem látszik.

A részlegesen takart szakaszoknál metszéspontot kell számolni.

8

Kétváltozós függvények ábrázolása

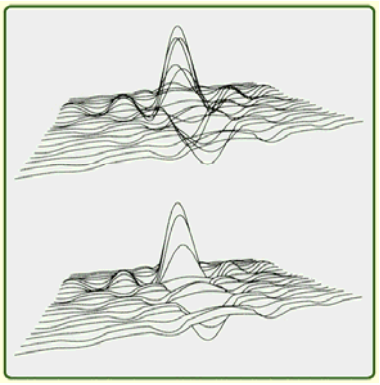


A szakasz nem töréspontól töréspontig, hanem a sziluett töréspontjától töréspontjáig terjed!

A sziluett töréspontjai sűrűsödnek!

9

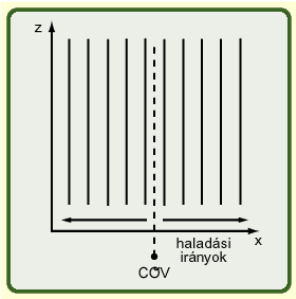
Kétváltozós függvények ábrázolása



10

Kétváltozós függvények ábrázolása

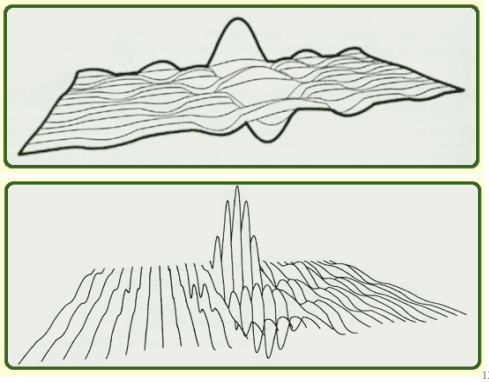
2. Ha csak a *z*-tengellyel párhuzamos egyenesek menti értékeket kötjük össze:



hasonló algoritmus

11

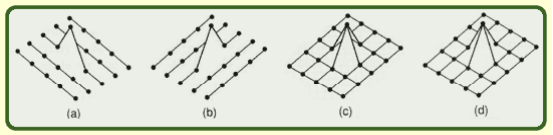
Kétváltozós függvények ábrázolása



12

Kétváltozós függvények ábrázolása

Drótvázas rajz konstans *x*- és *z*-menti görbékből

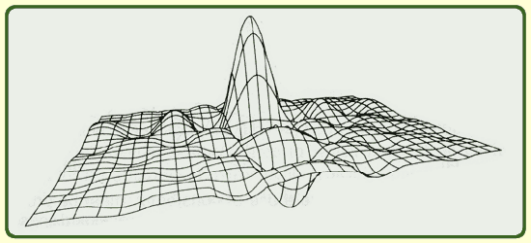


x-menti kép | z-menti kép | A két kép egymásra másolva | A korrekt kép

Nem lehet egyszerűen egymásra rakni a két képet

13

Kétváltozós függvények ábrázolása



Először az *x* irányú vonalakat rajzoljuk meg közelről távolra haladva (mint korábban), de minden vonal megrajzolása után megrajzoljuk a *z* irányú vonalaknak a két utolsó x irányú vonal közti szakaszait (mint korábban), ha látszanak.

14

A látható felszín meghatározására szolgáló általános algoritmusok

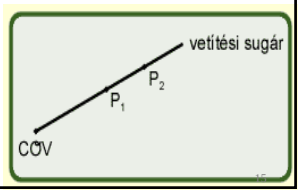
A tárgyak takarják-e egymást?

Mely tárgy látható?

Pontokra: $P_1 = (x_1, y_1, z_1)$ és $P_2 = (x_2, y_2, z_2)$;

Takarja-e egyik a másikat?

Ha ugyanazon a vetítési sugáron vannak, akkor a közelebbi takarja a másikat, különben nem.



Nehéz probléma

15

A látható felszín meghatározására szolgáló általános algoritmusok

Mélységbeli összehasonlítás

Helye: a normalizálási transzformáció után, ekkor

a. **párhuzamos vetítésnél**: a vetítési sugarak párhuzamosak a *z-tengellyel*, ekkor P_1 és P_2 ugyanazon a vetítési sugáron van, ha

$$x_1 = x_2 \text{ és } y_1 = y_2$$

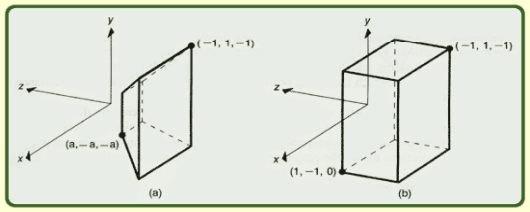
b. **perspektív vetítésnél**: a vetítési sugarak *COV*-ből indulnak ki, ekkor P_1 és P_2 ugyanazon a vetítési sugáron van, ha

$$x_1 / z_1 = x_2 / z_2 \text{ és } y_1 / z_1 = y_2 / z_2$$

16

A látható felszín meghatározására szolgáló általános algoritmusok

Perspektív vetítésnél azt a transzformációt használjuk, amely a perspektív kanonikus térfogatot átviszi párhuzamos kanonikus térfogatba



perspektív kanonikus térfogat | párhuzamos

17

A látható felszín meghatározására szolgáló általános algoritmusok

Perspektív kanonikus térfogatot párhuzamos kanonikus térfogatba transzformáló mátrix:

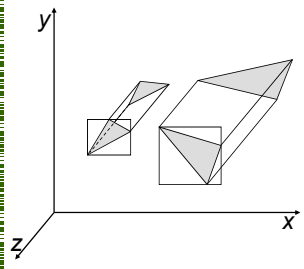
$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \dfrac{1}{1+z_{\min}} & \dfrac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{pmatrix}, \quad z_{\min} \neq -1$$

Ekkor a vetítési sugarak már párhuzamosak a *z*-tengellyel. Egyszerűbb a láthatóság.

18

A látható felszín meghatározására szolgáló általános algoritmusok

Tárgyak kiterjedése, határoló téglalapok, testek



Határoló-téglalap teszt

Ha a határoló téglalapok nem fedik egymást, akkor a vetületek sem fedik egymást, különben további vizsgálat szükséges

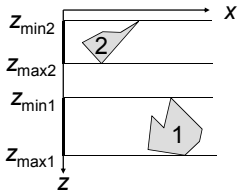
19

A látható felszín meghatározására szolgáló általános algoritmusok

1-dimenziós kiterjedés (határoló intervallum)

minmax-teszt:

A kiterjedés minimális és maximális értékeinek összehasonlításával döntjük el a takarást



A kiterjedés meghatározása:

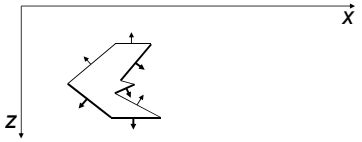
a tárgy (csúcs)pontjaihoz tartozó koordináták *min.* és *max.* értékeiből

20

A látható felszín meghatározására szolgáló általános algoritmusok

Hátra néző lapok kiválogatása

Tegyük fel, hogy poligon határú síklapokkal határolt a tárgy, és adottak a síklapoknak a tárgyból kifelé mutató normálisai. Ekkor azok a lapok nem láthatók, amelyek normálisai a "megfigyelőtől ellentétes" irányba mutatnak



21

A látható felszín meghatározására szolgáló általános algoritmusok

Projektív vetítés esetén:

$\underline{n}$: a poligon normálisa (n_x, n_y, n_z)

$\underline{v}$: *COV*-ből a poligon tetszőleges pontjába mutató vektor

Ha $\underline{n} \cdot \underline{v} < 0$ előre néz

> 0 hátra néz

$= 0$ csak az éle látszik

Az (*x*,*y*) síkra történő ortografikus vetítés esetén:

$n_z < 0$ hátra néz

> 0 előre néz

$= 0$ csak az éle látszik

22

LÁTHATÓ VONALAK MEGHATÁROZÁSA

Robert-féle algoritmus

Apple-féle algoritmus

Megszakított vonalak

23

LÁTHATÓ VONALAK MEGHATÁROZÁSA

Robert-féle algoritmus:

Síklapokkal határolt konvex testek éleire

1. A hátrafelé néző lapok meghatározása
2. A hátrafelé néző lapok közös élei elhagyhatók (azok nem láthatók)
3. Minden megmaradt élt minden testtel összehasonlítunk (kiterjedés vizsgálattal sok test triviálisan kizárható)

A fennmaradó esetek:

Az élet egy test teljesen eltakarja

Az élnek egy szakasza látszik a test mögül

Az élnek két szakasza látszik a (konvex) test mögül

24

LÁTHATÓ VONALAK MEGHATÁROZÁSA

Apple-féle algoritmus

Az élek pontjaihoz hozzárendel egy egész számot a pontot takaró előre néző lapok számát (kvantitatív láthatatlanság)

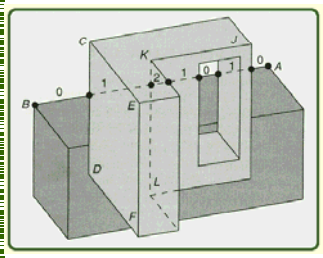
A kvantitatív láthatatlanság csak akkor változik, ha az él egy un. kontúr vonal mögött halad.

Kontúr vonal: előre és hátra néző lap közötti él.

25

Apple-féle algoritmus

A kvantitatív láthatatlanság számítása: ++, ha az él előre néző poligon mögé megy,
- -, ha az él előre néző poligon mögül jön ki.



Az él akkor látszik, ha a kvantitatív láthatatlansága = 0

26

Apple-féle algoritmus

Egymáson átható poliéderek nem megengedettek!

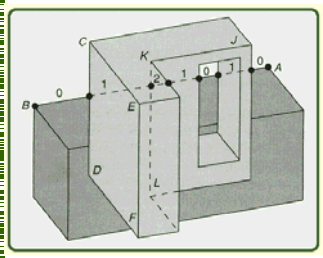
Az algoritmus megvalósítása:

1. Válasszunk ki egy csúcspontot, határozzuk meg a kvantitatív láthatatlanságát (direkt módszer)
2. Haladjunk az éleken, és közben módosítsuk a kvantitatív láthatatlansági értéket, *0* érték esetén rajzolunk

27

Apple-féle algoritmus

A csúcspont kvantitatív láthatatlansága nem egyszerű: pl. a *KJ* él látszik, a *KL* nem! A *K* csúcspont látszik vagy nem?



Az előre néző fölső lap *K* csúcsa látszik, tehát látszik a *KJ* él is, de a *KL* élet tartalmazó hátra néző lap *K* csúcsa nem látszik (takarja a fölső lap), így a *KL* él sem látszik.

28

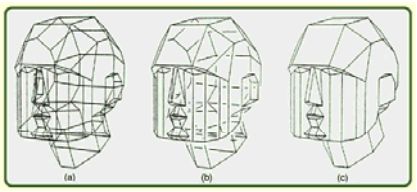
LÁTHATÓ VONALAK MEGHATÁROZÁSA

A látható vonal algoritmusok arra is használhatók, hogy a nem látható vonalak szaggatottak, pontozottak, halványabbak legyenek

29

LÁTHATÓ VONALAK MEGHATÁROZÁSA

Megszakított vonalak



(*a*): minden vonal látszik
(*b*): mintha minden vonalnak lenne egy takaró sávja, ami eltakarja a mögötte lévő részeket
(*c*): csak a látható vonalak látszanak

30

Megszakított vonalak

Az algoritmus az élek vetületének metszéspontja körül csak a közelebbit rajzolja, a távolabbit megszakítja

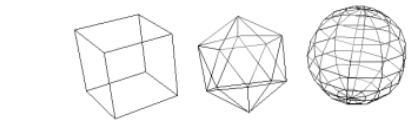
Algoritmus:

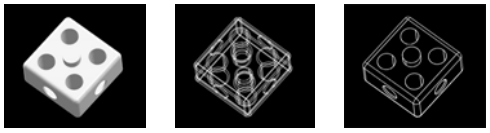
Minden vonalhoz megkeressük az előtte levőket
Csak a látható szakaszokat őrizzük meg

Ha minden metszésponttal végeztük, akkor rajzolunk.

31

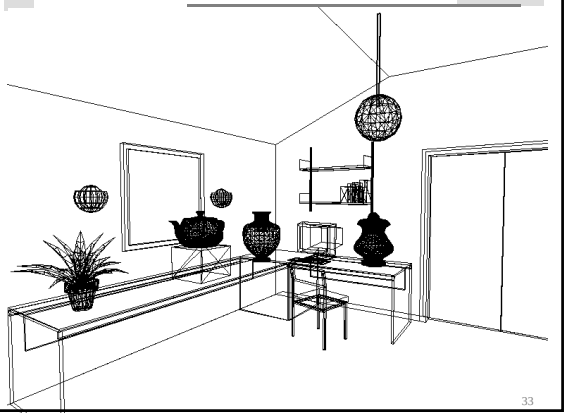
Példák





32

Példák



33

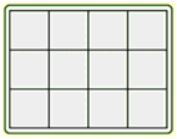
Példák

34

A látható felszín meghatározására szolgáló általános algoritmusok

Térbeli partícionálás

Észrevétel: nem minden tárgynak van minden vetítési sugárral metszéspontja (pl. távol vannak, más irány) → osszuk fel (particionáljuk) a képernyőt

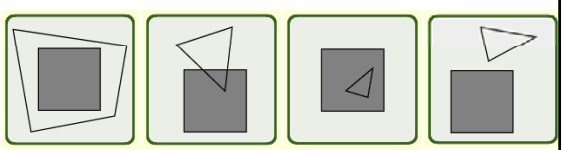


Meghatározzuk, hogy mely tárgyak vetülete van benne a megfelelő részben (partícióban) és csak azokkal keresünk metszéspontokat

35

Látható felszín algoritmusok

4 lehetőség egy poligon és egy téglalap alakú terület között:



Tartalmazó poligon | Metsző poligon | Tartalmazott poligon | Idegen poligon

36

Látható felszín algoritmusok

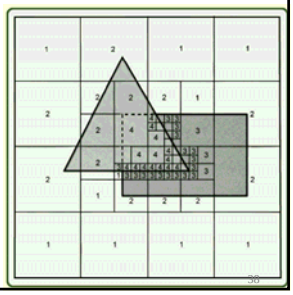
Mikor dönthető el könnyen, hogy mi rajzolható?

1. Minden poligon idegen a területtől (háttér)
2. Egyetlen metsző vagy tartalmazott poligon (háttér + pásztázással poligon)
3. Egyetlen tartalmazó poligon (rajz a poligon színével)
4. Van olyan tartalmazó poligon, amelyik a többi előtt van (rajz a poligon színével)

37

A látható felszín meghatározására szolgáló általános algoritmusok

Ez jó módszer, ha a tárgyak vetületei egyenletesen oszlanak el a teljes képernyőn, különben különböző méretű partíciókat érdemes készíteni: kisebb partíciót ott, ahol több tárgy vetülete van

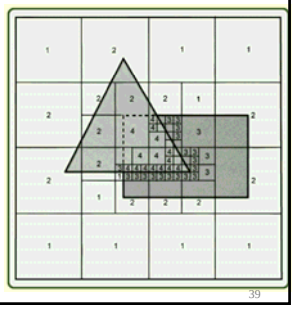


38

Látható felszín algoritmusok

Terület-osztó algoritmus látható felszín meghatározására: "oszd meg és uralkodj"

Ha egy területen könnyen eldönthető, hogy melyik poligon jeleníthető meg, akkor azt rajzoljuk ki, különben osszuk fel a területet, és alkalmazzuk az eljárást a rész területekre

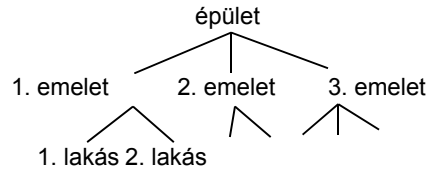


39

A látható felszín meghatározására szolgáló általános algoritmusok

Hierarchikus struktúrák alkalmazása

pl.



Ha a vetítési sugár nem metszi az épületet, akkor az emeleteit és az emeletek lakásait sem (tehát nem kell vizsgálni azokat)

40

Látható felszín algoritmusok

***Z - buffer* vagy *mélység – puffer* algoritmus**

(kép alapú)

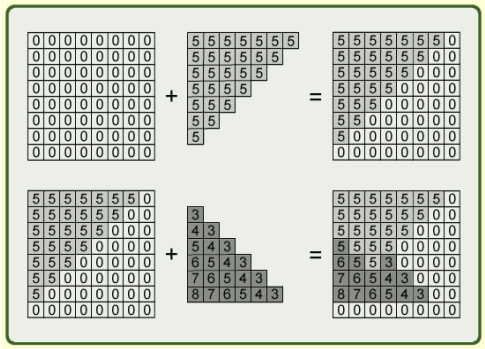
F : kép-puffer (képpontok tárolására) kezdeti értéke: háttérszín

Z : mélység-puffer (minden pontban a megfelelő *z* érték), kezdeti értéke: z-max (hátsó vágási sík)

Pásztázás közben *F*-be és *Z*-be bekerül az új pont, ha nincs messzebb, mint az eddigi *z* érték.

41

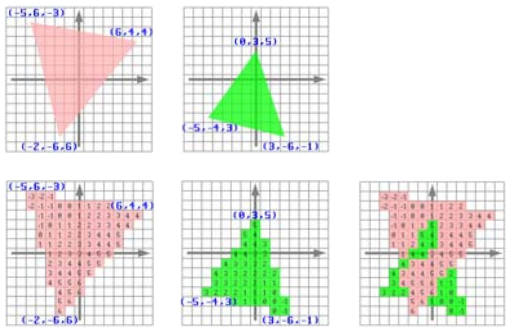
Z-buffer algoritmus



A hátsó vágási sík a $z = 0$

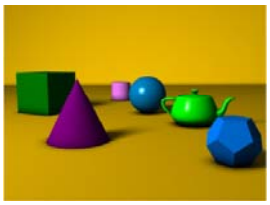
42

Z-buffer algoritmus

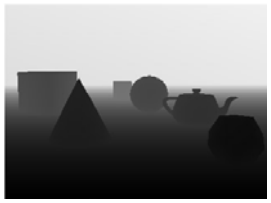


43

Z-buffer algoritmus



Kép



z-buffer

44

Z-buffer algoritmus

Tulajdonságai:

- Nincs tárgyak rendezése, összehasonlítása, metszéspontok számítása
- Poligononként végezhető el,
- Nem csak poligonokra jó,
- Nagy helyigény, de lehet sávonként haladni,
- Könnyű implementálni,
- Könnyű egy újabb tárgy képét hozzávenni
- A *z*-értékek felhasználhatók terület és térfogat számításra

45

Látható felszín (OpenGL)

OpenGL-ben: a ***mélységi-*** vagy ***z-buffer*** algoritmus

A mélységbeli összehasonlítást

```
glEnable (GL_DEPTH_TEST)      // engedélyezi
glDisable(GL_DEPTH_TEST)      // tiltja
```

46

Sokszögek oldalai (OpenGL)

Sokszögek (első és hátsó) oldalai OpenGL-ben:
Első és hátsó oldalak explicit megadása:

```
glFrontFace(GLenum mode);
```

`mode:`

`GL_CCW` (alapértelmezés)
az az oldal lesz első oldal, amelyen a csúcspontokat az óramutató járásával ellenkező irányban adtuk meg,

`GL_CW` az ellenkezője.

47

Sokszögek oldalai (OpenGL)

```
void glPolygonMode(enum face, enum mode);
face: GL_FRONT_AND_BACK, GL_FRONT, GL_BACK
```

Megmondja, hogy a poligon mindkét, első vagy hátsó oldalát kell rajzolni

`mode:` Megmondja, hogy

`GL_POINT` csak a poligon csúcsait,

`GL_LINE` határvonalát kell kirajzolni, vagy

`GL_FILL` ki is kell tölteni (alapértelmezés)

48

Sokszögek oldalai (OpenGL)

A sokszög meghatározott oldalán letiltja a világítási, árnyalási és szín-számítási műveleteket (láthatatlan oldal)

```
glCullFace(GLenum mode);
mode: GL_FRONT, GL_BACK
```

A culling-ot
`glEnable (GL_CULL_FACE)` engedélyezhetjük illetve
`glDisable(GL_CULL_FACE)` tilthatjuk

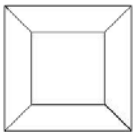
49

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
#include<glut.h>

void glutWireCube(GLdouble size);
```

`size` : a kocka élhossza (a kocka középpontja az origóban lesz)



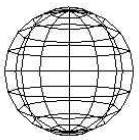
50

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
void glutWireSphere(GLdouble radius,
     GLint slices, Glint stacks);
```

`radius`: a gömb sugara,
`slices`: a *z*-tengely körüli beosztások (hosszúsági körök) száma,
`stacks`: a *z*-tengely menti beosztások (szélességi körök) száma

A középpont az origóban lesz

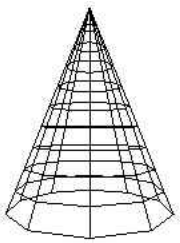


51

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
void glutWireCone(GLdouble base,
     GLdouble height,GLint slices,
     GLint stacks);
```

`base`: a kúp alapjának sugara,
`height` : a kúp magassága,
`slices`: a *z*-tengely körüli beosztások száma,
`stacks` : a *z*-tengely menti beosztások száma

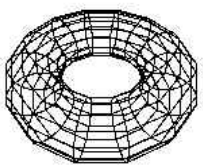


52

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
void glutWireTorus(GLdouble
     innerRadius,GLdouble outerRadius,
     GLint nsides, GLint rings);
```

`innerRadius`: a tórusz belső sugara,
`outerRadius`: a tórusz külső sugara,
`nsides`: a radiális részek oldalainak száma,
`rings`: a tórusz radiális beosztásainak száma.



53

MEGVILÁGÍTÁS

Világító tárgyak
Környezeti fény
Szórt visszaverődés
Környezeti fény és diffúz visszaverődés együtt
Tükröző visszaverődés
Poligonokból álló felületek fényességének meghatározása
Gouraud-féle fényesség
Phong-féle fényesség

54

MEGVILÁGÍTÁS

a. Világító tárgyak:

Minden tárgynak saját intenzitású fénye van

A megvilágítás egyenlete:

$I = k_i$

k_i - a tárgy saját fényének az intenzitása

független a pont helyzetétől

55

MEGVILÁGÍTÁS

b. Környezeti (szórt - ambient) fény:

Minden irányból egyenletes

$I = I_a k_a$

I_a: környezeti fény intenzitása

k_a: a környezeti fény visszaverődési együtthatója (anyagtól függ),

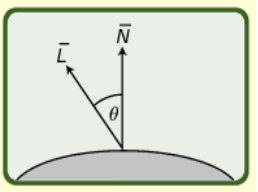
$0 \le k_a \le 1$

56

MEGVILÁGÍTÁS

c. Diffúz (diffuse) visszaverődés
(Lambert-féle visszaverődés)

Minden irányban ugyanannyi fényt ver vissza. A felület fényessége (I) függ a fényforrás iránya ($\underline{L}$) és a felület normálisa ($\underline{N}$) közötti szögtől:



$\underline{N}$, $\underline{L}$ egységvektorok

$I = I_p k_d \cos\Theta = I_p k_d (\underline{N}\,\underline{L})$

I_p: a pontforrás intenzitása

k_d: a szórt visszaverődés együtthatója (anyagtól függ), $0 \le k_d \le 1$.

57

MEGVILÁGÍTÁS

Környezeti fény (b) és diffúz visszaverődés (c) együtt:

$I = I_a k_a + I_p k_d (\underline{N}\,\underline{L})$

Ha a fényforrás és a tárgy közötti távolságot (d_L) is figyelembe vesszük, akkor:

$I = I_a k_a + \underbrace{I_p / d_L^2}_{= f_{att}} k_d (\underline{N}\,\underline{L})$ (gyengülési faktor)

58

MEGVILÁGÍTÁS

Színes fény és felületek esetén a komponensekre:

$I_R = I_{aR} k_a O_{dR} + f_{att} I_{pR} k_d O_{dR} (\underline{N}\,\underline{L})$

$I_G = ...$

$I_B = ...$

ahol O_{dR} a tárgy szórt vörös komponense

I_{pR} a megvilágítás vörös komponens

$k_d O_{dR}$ visszaverődési hányad komponense

...

Általában:

$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} k_d O_{d\lambda} (\underline{N}\,\underline{L})$

ahol λ a hullámhossz

59