LÁTHATÓ FELÜLETEK MEGHATÁROZÁSA

Kétváltozós függvények ábrázolása

A látható felszín meghatározására szolgáló általános algoritmusok

Látható felszín algoritmusok

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:

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

LÁTHATÓ FELÜLETEK MEGHATÁROZÁSA

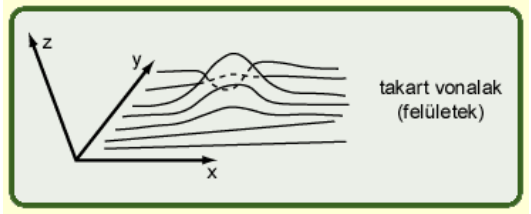
```
2. for minden tárgyra do begin
     határozzuk meg a tárgynak
     azokat a részeit, amelyek
     nincsenek takarásban saját
     maga vagy más tárgyak miatt;
     a kiválasztott részeket
     rajzoljuk ki a megfelelő
     színben
   end
```

A szükséges idő: $O(n^2)$

Kétváltozós függvények ábrázolása

plotterrel

$$y = f(x,z)$$



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.

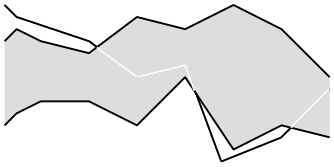
Keressünk olyan algoritmust, amely a takart vonalakat nem rajzolja ki.

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

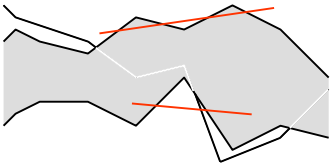
Kétváltozós függvények ábrázolása



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

A részlegesen takart szakaszoknál metszéspontot kell számolni.

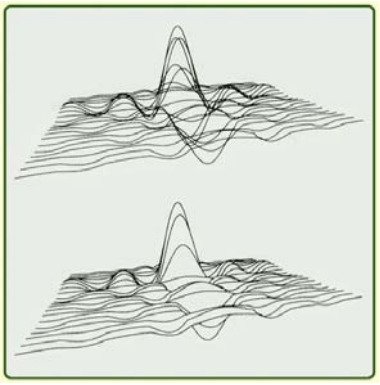
Kétváltozós függvények ábrázolása



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

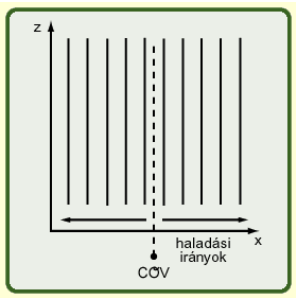
A sziluett töréspontjai sűrűsödnek!

Kétváltozós függvények ábrázolása



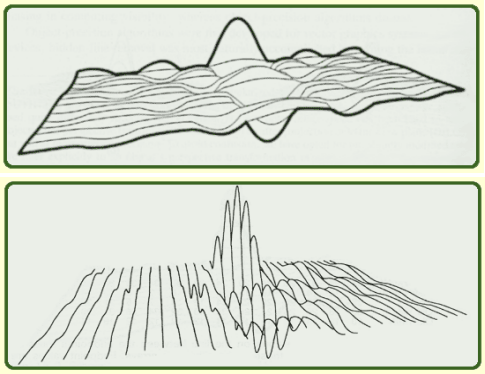
Kétváltozós függvények ábrázolása

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



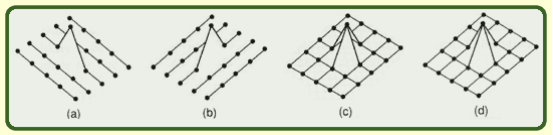
hasonló algoritmus

Kétváltozós függvények ábrázolása



Kétváltozós függvények ábrázolása

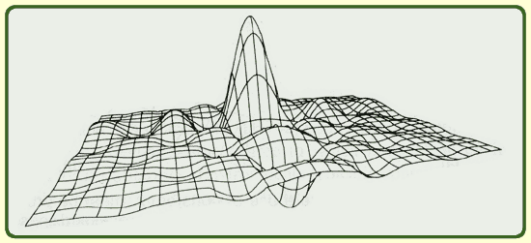
Drótvázas rajz konstans *x*- és *z*-menti görbékből



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

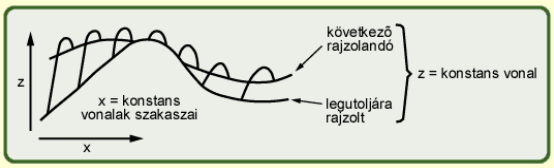
Nem lehet egyszerűen egymásra rakni a két képet

Kétváltozós függvények ábrázolása



Először az *x* tengellyel párhuzamosak vonalakat közelről távolra haladva rajzoljuk (mint korábban), de a *z* irányú vonalakat szakaszonként (közelről távolra haladva) a szakasz közelebbi végpontjának z értékéhet tartozó korábbi sziluettet használva.

Kétváltozós függvények ábrázolása

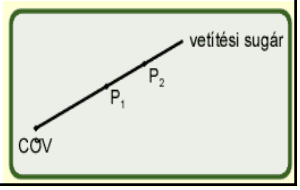


Ezeket az eljárásokat általában csak akkor használjuk, ha a rajzolandó vonalak *x* = *konstans* vagy *z* = *konstans* menti értékekből állnak)

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

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

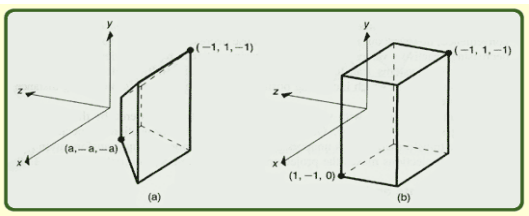
$$x_1 = x_2 \text{ és } y_1 = y_2$$

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

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

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

A látható felszín meghatározására szolgáló általános algoritmusok

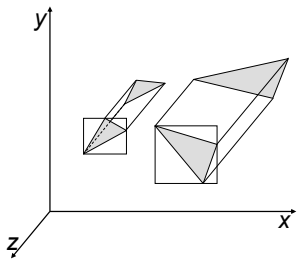
Perspektív kanonikus térfogatot párhuzamos kanonikus térfogatba transzformáló mátrix:

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{min}} & \frac{-z_{min}}{1+z_{min}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Ekkor a vetítési sugarak már párhuzamosak a *z*-tengellyel. Egyszerűbben végezhető a vágás

A látható felszín meghatározására szolgáló általános algoritmusok

Tárgyak kiterjedése, határoló téglalapok, testek



Határoló-téglalap teszt

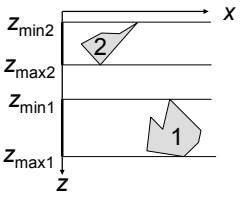
Ha a határoló téglalapok nem fedik egymást, akkor a vetületek sem fedik egymást, különben további vizsgálat szükséges

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



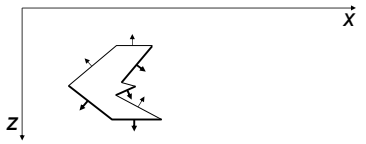
A kiterjedés meghatározása:

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

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



A látható felszín meghatározására szolgáló általános algoritmusok

Azonosításuk:

$\underline{n}$: normális (n_x, n_y, n_z)

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

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

> 0 hátra néz

$= 0$ csak az éle látszik

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

$n_z < 0$ hátra néz

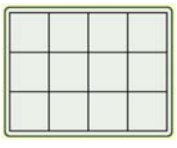
> 0 előre néz

$= 0$ csak az éle látszik

A látható felszín meghatározására szolgáló általános algoritmusok

Térbeli partícionálás

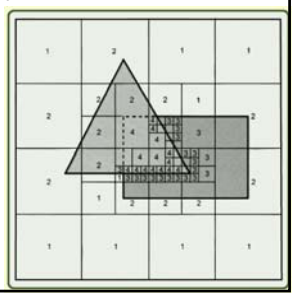
Észrevétel: nem minden tárgynak van minden vetítési sugárral metszéspontja (pl. távol vannak, más irány) ►►► osszuk fel (partícionáljuk) a képernyőt



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

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

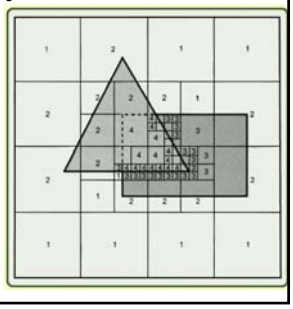


Látható felszín algoritmusok

<u>Terület-osztó algoritmus</u> látható felszín meghatározására

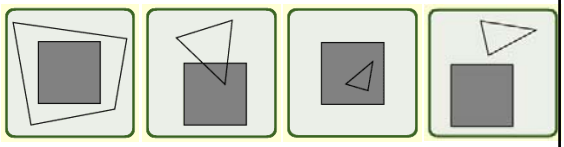
"oszd meg és uralkodj"

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



Látható felszín algoritmusok

4 lehetőség egy poligon és egy téglalap alakú terület között:



Tartalmazó poligon | Metsző poligon | Tartalmazott poligon | Idegen poligon

Látható felszín algoritmusok

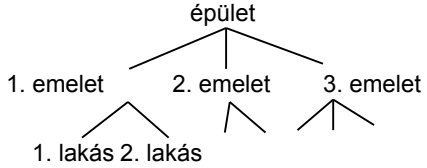
Mikor dönthető el könnyen, hogy mi rajzolható?

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.

A látható felszín meghatározására szolgáló általános algoritmusok

<u>Hierarchikus struktúrák alkalmazása</u>

pl.



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

Látható felszín algoritmusok

***<u>Z - buffer</u>* <u>vagy *mélység – puffer* algoritmus</u>**

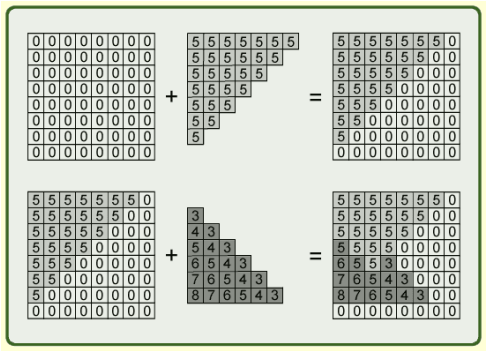
(kép alapú)

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

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

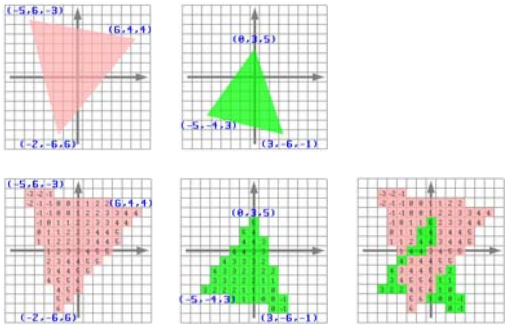
Pásztázás közben *F*-be és *Z*-be bekerül az új pont, ha nincs messzebb, mint az eddigi *z* érték.

Z-buffer algoritmus

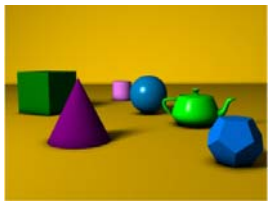


A hátsó vágási sík a $z = 0$

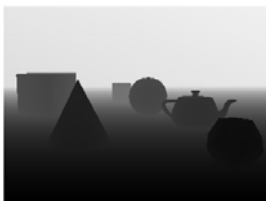
Z-buffer algoritmus



Z-buffer algoritmus



Kép



z-buffer

Z-buffer algoritmus

Tulajdonságai:

- Nincs tárgyak rendezése, összehasonlítása, metszéspontok számítása
- Poligononként végezhető el, ha nincsenek átható poligonok,
- Nem csak poligonokra jó,
- Nagy helyigény, de lehet sávonként haladni,
- Könnyű implementálni,
- Könnyű egy újabb tárgy képét hozzávenni

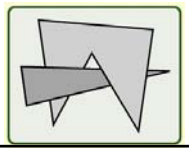
Látható felszín algoritmusok

Lista-prioritás algoritmusok

Meghatározzák a tárgyaknak azt a sorrendjét, ami a kép kirajzolásához kell.

Pl.: Ha a *z* értékekben nincs átfedés, akkor a tárgyakat növekvő *z* értékük szerint kell rendezni, és utána távolról közelre haladva megjeleníteni.

Néha még akkor is lehet ilyen sorrendet megadni, ha a *z* értékekben van átfedés, de nem mindig



Ilyenkor szétvágjuk a tárgyakat és a darabokat rendezzük sorba

Látható felszín algoritmusok

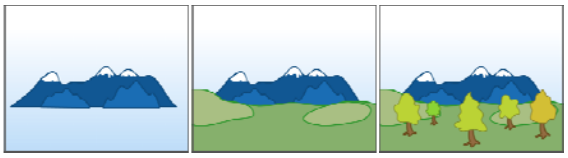
1. Mélység szerint rendező algoritmus

Lépések:

1. Rendezzük a poligonokat legtávolabbi *z* koordinátájuk szerint
2. Vágjuk szét az átfedő poligonokat (ha szükséges)
3. Pásztázzunk minden poligont hátulról előre haladva

Ha a poligonok párhuzamos síkokban fekszenek, akkor a 2. lépés kimaradhat

Festő algoritmus



Látható felszín algoritmusok

Tegyük fel hogy a *P* poligon legtávolabbi *z* koordinátája szerint a lista végén van. Pásztázás előtt össze kell hasonlítani a lista azon *Q* elemeivel, amelyeknek *z* irányú kiterjedése átfedi *P z* irányú kiterjedését, és meg kell vizsgálni, hogy

P átfedi-e *Q*-t?

Látható elszín algoritmusok

P átfedi-e *Q*-t?

1. ha *P* és *Q* *x* kiterjedése nem átfedő, akkor **nem**;
2. ha *P* és *Q* *y* kiterjedése nem átfedő, akkor **nem**;
3. ha *COV*-ból nézve *P* teljes terjedelmében a *Q* túlsó oldalán van, akkor **nem**;
4. ha *COV*-ból nézve *Q* teljes terjedelmében *P*-nek az innenső oldalán van, akkor **nem**;
5. ha *P* és *Q* (*x*,*y*) síkra való vetülete nem átfedő, akkor **nem**

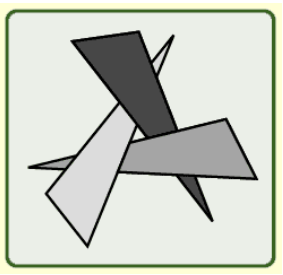
különben (hátha *Q*-t kell előbb rajzolni):

3‘ ha *COV*-ból nézve *Q* teljes terjedelmében a *P* sík túlsó oldalán van, akkor *P* *Q* csere

4‘ ha *COV*-ból nézve *P* teljes terjedelmében *Q*-nak az innenső oldalán van, akkor *P* *Q* csere

különben *P*-t vagy *Q*-t fel kell darabolni a másik síkkal, és a darabokat kell beilleszteni a listába

Látható felszín algoritmusok



Végtelen ciklust eredményezne, ezért megjelöljük azokat a poligonokat, amelyeket egyszer már a lista végére tettünk, és ha újra előjönnek, akkor darabolunk

Látható felszín algoritmusok

2. Bináris tér-partícionáló fa algoritmus

Ötlet: Ha van olyan sík, amely a tárgyakat (teljes egészükben) két féltérbe osztja, akkor a *COV*-t tartalmazó féltér tárgyait nem takarhatják el a másik féltér tárgyai

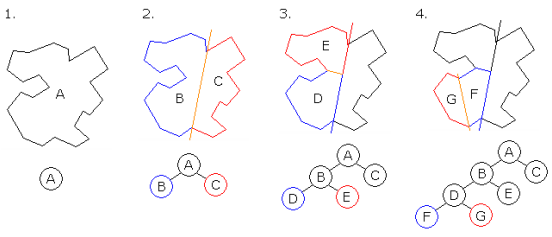
BSP fa: Csomópontok - poligonok

A csomóponthoz tartozó poligon síkjával darabolhatjuk a többi poligont, és azok darabjaival folytathatjuk a fát

bal oldalra: azok a poligonok, amelyek elöl vannak (később kell rajzolni)

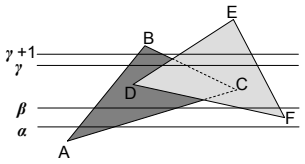
jobb oldalra: azok a poligonok, amelyek hátul vannak (korábban kell rajzolni)

Bináris tér-partícionálás



Látható felszín algoritmusok

Pásztázó vonal algoritmus látható felszín meghatározására (hasonló a poligonok kitöltését végző algoritmushoz)



Vízszintesen pászázunk

Most több poligon lehet

Látható felszín algoritmusok

ÉT (élek táblázata):

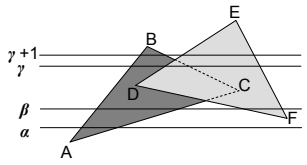
A poligonok nem vízszintes éleit tartalmazza az élek kisebbik *y*-koordinátája szerint növekvő, ezen belül az élek meredeksége szerint csökkenő sorrendbe rendezve (a vízszintes éleket kihagyjuk)

Egy elem részei:
- a kisebb *y*-koordinátájú csúcspont *x*-koordinátája
- a másik csúcspont *y*-koordinátája
- *x* növekménye: $\Delta x = 1/m$
- poligon azonosító (több poligon lehet)

Látható felszín algoritmusok

PT (poligonok táblázata):

egy elem részei:
- azonosító
- a sík egyenletének együtthatói
  a sík egyenlete: $Ax + By + Cz + D = O$
- árnyalati/színezési információ
- ki-be jelző (kezdő érték: ki)



ÉT: *AB*
AC
FD
FE
CB
DE

PT: *ABC*
DEF

Látható felszín algoritmusok

AÉT (aktív élek táblázata):

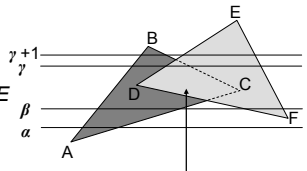
Balról jobbra, alulról felfelé haladva

α: *AB, AC* (*ABC*) — *AB*-nél **be**-, AC-nél **ki**kapcsolva

AB-től *AC*-ig *ABC* színe

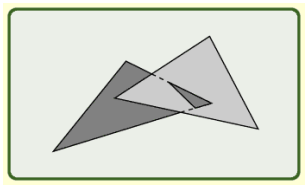
β: *AB, AC, FD, FE* (*ABC*, *DEF*)
be ki be ki

$\gamma + 1$: *AB, DE, CB, FE* (*ABC*, *DEF*)
be ki be ki



A síkok egyenletéből dönthető el, hogy melyik van közelebb

Látható felszín algoritmusok



Ha a poligonokat felvágjuk a metszeteik mentén, akkor nem kell minden pontban megvizsgálni a poligonok sorrendjét, elegendő csak akkor, ha egy "takaró" poligon véget ér.

Látható felszín (OpenGL)

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

A mélységbeli összehasonlítást

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

Sokszögek oldalai (OpenGL)

Sokszögek (elülső és hátulsó) oldalai OpenGL-ben:
Elülső oldal az, amelyen a csúcspontok az óramutató járásával ellentétes irányban vannak megadva

`void glPolygonMode(enum face, enum mode);`

`face: GL_FRONT_AND_BACK, GL_FRONT, GL_BACK`

Megmondja, hogy a poligon mindkét, elülső vagy hátulsó 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)

Sokszögek oldalai (OpenGL)

Elülső és hátulsó oldalak explicit megadása:

`glFrontFace(GLenum mode);`

`mode:`
- `GL_CW` az az oldal lesz elülső oldal, amelyen a csúcspontokat az óramutató járásával megegyező irányban adtuk meg,
- `GL_CCW` az ellenkezője.

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

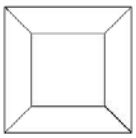
`glDisable(GL_CULL_FACE)` tilthatjuk

3D geometriai formák drótvázas megjelenítése (OpenGL)

```
#include<glut.h>

void glutWireCube(GLdouble size);
```

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

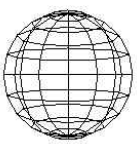


3D geometriai formák drótvázas megjelenítése (OpenGL)

```
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

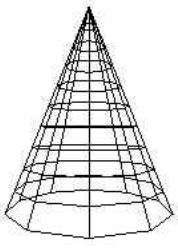
A középpont az origóban lesz



3D geometriai formák drótvázas megjelenítése (OpenGL)

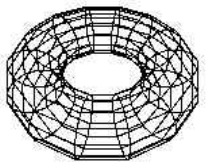
```
void glutWireCone(GLdouble base,
     GLdouble height,GLint slices,
     GLint stacks);
```

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



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.