GeoGL3D v2.0 – térgeometriai szerkesztőprogram

Szerző:
Rana Dávid

Komárom **2009**

Tartalomjegyzék

1. Bevezetés

1.1. A dolgozat célja

A matematikaórákon gyakran okoz gondot a tanulóknak, hogy elképzeljék a feladatban szereplő térgeometriai alakzatokat. A dolgozatom célja egy olyan szerkesztőprogram létrehozása volt, mely segítségével e problémák könnyen orvosolhatóak. A tanulók az előre elkészített fájlokat megnyitva maguk vizsgálhatják meg a kérdéses testeket, sőt a megfelelő háttérismeret birtokában akár maguk is megszerkeszthetik azokat, növelve ezzel térlátásukat. Kitűnő segítséget nyújthat elsősorban ábrázoló-, ill. analitikus geometria tantárgyakhoz, mivel a szerkesztőprogramban síkokkal, ill. vektorokkal dolgozhatunk.

1.2. A dolgozat eredménye

A "GeoGL3D v2.0" Delphi2005 (OpenGL)-ben megírt térgeometriai szerkesztőprogram, mely a különböző sík-, ill. téridomok megjelenítésére szolgál.

Amit ennek a szerkesztőprogramnak a működéséről tudni kell az az, hogy bázispontokat veszünk fel, majd azokra objektumokat, pl. két felvett pontra szakaszt, egyenest, kúpot, hengert, stb. illeszthetünk. A legfontosabb a számítógépes szerkesztésben az, hogy a kész szerkesztés elemei változtathatók, miközben a származási viszonyok megmaradnak. Különféle színeket és vonalvastagságokat használhatunk a szebb ábra és a jobb áttekinthetőség érdekében. Továbbá lehetőség van az objektumok láthatóságát is módosítani, így az egyes objektumok áttetszők, ill. akár teljesen elrejthetők. Az elkészült szerkesztéseket saját fájlformátumban (*.gl2) menthetjük el.

Az alakzatok (testek) megjelenítését is szabályozhatjuk. Lehetőség van csúcsok-, élek- és lapok megjelenítésére. Az alakzatok kirajzolásánál használhatjuk a konstans és a folytonos árnyalást. Konstans árnyalás esetén a rendszer egyetlen csúcspont színével színezi ki az egész objektumot, folytonos árnyalás esetén a csúcspontok színét lineárisan interpolálja. Folytonos árnyalás alkalmazásával el tudjuk tüntetni a képről a görbült felületeket közelítő poligonháló éleit. A kamera mozgását az alap, csúszás, forgatás, +/– eszköztár nyomógombjaival változtathatjuk (vagy menüpontokból). E lehetőségeket kihasználva akár az alakzat belsejében is "sétálgathatunk".

2. A Delphi programozási nyelvről

A dolgozatom alapját képző GeoGL3D v2.0 szoftver Delphi programozási nyelv felhasználásával készült.

A Delphi alapja az Objekt Pascal programozási nyelv, amely az ismert Turbo Pascal objektumos felépítménye. Éppen ezért a Delphiben fel tudjuk használni a Turbo Pascal ciklusait, elágazásait, változók alaptípusait, illetve parancsait.

Többek közt azért választottam a Delphi programozási nyelvet, mert a Delphi fejlesztői környezet szerintem jobb más hozzá hasonló programozási nyelvtől. A Delphi az egyik legeffektívebb eszköz, mellyel Windows operációs rendszer alatt alkalmazásokat hozhatunk létre. Az alkalmazás létrehozásának maximálisan leegyszerűsített fázisa van. A programokat jóval kevesebb idő alatt meg lehet írni Delphiben, mint Turbo Pascalban, a rengeteg vizuális eszköznek és integrált környezetnek köszönhetően.

A Windows alatti programozás eseményekkel irányított programozás. A programozónak csak a rendszer különféle eseményeire kell reagálnia. A program irányítását az operációs rendszer végzi.

Jómagam a Delphi 2005-ös verzióját választottam, ebben készült a „GeoGL3D v2.0“ c. térgeometriai szerkesztőprogram.

3. Az OpenGL grafika

„Az OpenGL hardverfüggetlen programozási felületet biztosító háromdimenziós grafikus alprogramrendszer, melyet a Silicon Graphics Inc. fejlesztett ki. A GL (Graphics Library) több programozási nyelvből elérhető függvénykönyvtárakon keresztül. Az SG (Silicon Graphics) munkaállomások közismert előnye a gyors és igényes grafika, amit hardver oldalról a grafikus kártyába épített egy vagy több geometriai társprocesszor támogat. Ez a koncepció olyan sikeresnek bizonyult, hogy vezető hardver- és szoftvergyártó cégek (többek között a DEC, IBM, Intel, Microsoft és Silicon Graphics) összefogásával létrehoztak egy munkacsoportot, amely ez alapján specifikálta az OpenGL-t.

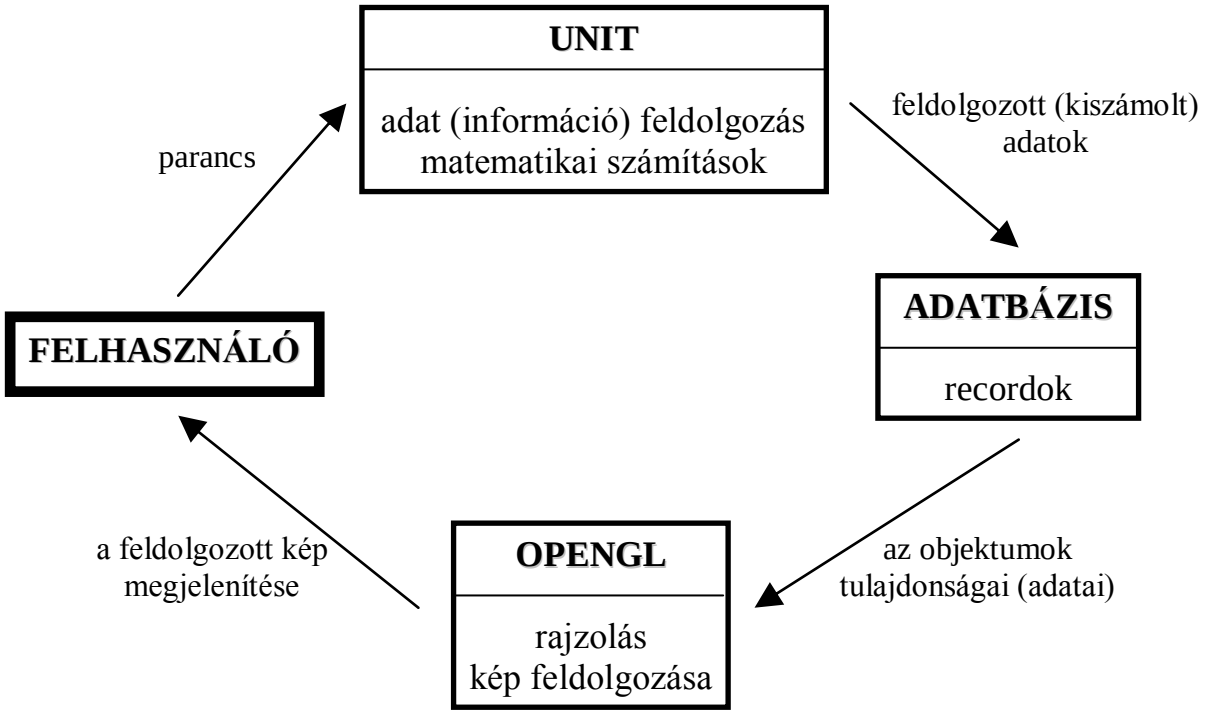
A legtöbb munkaállomás kategóriájú számítógép támogatja az OpenGL-t, de a PC-s világban használt WIN'95, WIN'98, Windows NT, Windows 2000 és Windows XP operációs rendszerek alatt az IBM PC (és vele kompatibilis) gépeken is futtathatunk ilyen alkalmazásokat. Természetesen a UNIX típusú operációs rendszereknek a PC-ken használatos különböző változatai (pl. Linux) is támogatják az OpenGL alkalmazások futtatását. Az egyes grafikus kártyák eltérő képességűek a beépített geometriai társprocesszorok (grafikai gyorsítók) típusától és számától függően.

Az OpenGL nem geometriai modellezőrendszer, tehát nincsenek benne összetett geometriai objektumok megadására alkalmas parancsok. Bár – mint mondtuk – az OpenGL nem interaktív rajzoló- vagy grafikus rendszer, azonban rendkívül alkalmas a geometriai modellezőrendszerek, CAD rendszerek, grafikus rendszerek, de még az interaktív játékprogramok grafikai igényeinek kielégítésére is. Ennek köszönhetően széles körben elterjedt. Az OpenGL-t használó alkalmazások több géptípuson, így minden munkaállomás kategóriájú gépen és a nálunk legnépszerűbb PC-ken is futtathatók, még a programok forrásnyelvi szintű portabilitása is megoldható.“ [9]

„Az OpenGL segítségével egyszerű, térben elhelyezkedő 3D-s grafikus primitíveket jeleníthetünk meg a képernyőn. Ennek során megválaszthatjuk a térből síkba való leképezés módját, az objektumok színét, megvilágítását és mintázatát. A síkbeli képen többféle módon figyelembe vehetjük azt is, hogy az elemek a térbeli elhelyezkedésből következően takarják egymást.“ [10]

3.1. A szerkesztőprogram adatfeldolgozása és a GL kapcsolata

Ha a felhasználó kiad egy parancsot az objektum megszerkesztésére az adatlapon, a komponensekből kiolvasott adatok feldolgozásra kerülnek. A feldolgozott (kiszámolt) adatok belekerülnek az adatbázisba (recordokba), melyet a kirajzolásnál a GL felhasznál. Az OpenGL a képet feldolgozza és megjeleníti a felhasználónak.



Mivel a szerkesztőprogramban a származási viszonyok megmaradnak, ezért elengedhetetlen egy jól átlátható és igényes adatbázis létrehozása, melynek tartalmát a GL kirajzoláskor felhasználja. A *record* (adatbázis) lényegében egy „adatbank“, adatok szervezett gyűjteménye, amelyben a kívánt információk kikereshetők, rendezhetők és módosíthatók. A recordok mindig egy objektum tulajdonságait határozzák meg. Az „objektumok:objektum“ record tartalmazza a további recordokat és az összes objektum tulajdonságait. A következőkben egy pont adatbázisát mutatom be:

```
.....
pont = RECORD
        nev : string[6];
        rgb : array[1..3] of real;
        szin : integer;
```

```
        lathatosag : integer;
        vastagsag : integer;
        koo1 : array[1..3] of real;
        END;
.....
objektum = RECORD
            .....
            pontok : array [1..n] of pont;
            .....
            END;
…….
var objektumok : objektum;
…….
```

A „pont“ recordból a következő adatok (információk) olvashatóak ki:

- *objektumok.pontok[n].nev* – az objektum neve kerül bele, típusa string, max. 6 karakter.
- *objektumok.pontok[n].rgb* – az objektum RGB (red, green, blue) színei. A tömb értéke 0 és 1 közötti értéket vehet fel, mely arányban van az eredeti szín 0..255 közötti értékével, mivel a GL ezt a számolási módot támogatja. Pl. a piros szín GL-ben: rgb[1]=1 {piros}, rgb[2]=0 {zöld}, rgb[3]=0 {kék}, mivel eredeti értéke: rgb[1]=255, rgb[2]=0, rgb[3]=0 lenne.
- *objektumok.pontok[n].szin* – az objektum eredeti színének a száma 16-os számrendszerben.
- *objektumok.pontok[n].lathatosag* – az objektum láthatósága, vagyis áttetszősége, mely 0 és 1 közötti értéket vehet fel. Ha értéke 0, az objektum teljesen áttetsző, vagyis láthatatlan.
- *objektumok.pontok[n].vastagsag* – az objektum vastagsága (pont nagysága), mely 1 és 10 közötti értéket vehet fel.
- *objektumok.pontok[n].koo1* – az objektum x (koo1[1]), y (koo1[2]) és z (koo1[3]) koordinátája.

Tehát egy objektumhoz kapcsolódó record tartalmazza az összes olyan adatot, melyből az objektum számítások nélkül kirajzolható, ezzel növelve a grafika kirajzolásának gyorsaságát. A következő programrészlet az összes pontot kirajzolja, amit az adatbázisban talál:

```
i:=1;
WHILE objektumok.pontok[i].nev<>'' DO
```

```
BEGIN
glLoadIdentity();
glTranslatef(xx,yy,zz);
glrotatef(yszog,0,1,0);
glrotatef(xszog,1,0,0);
glpointsize(objektumok.pontok[i].vastagsag*5);
glcolor4f(objektumok.pontok[i].rgb[1],objektumok.pontok[i].rgb[2],
          objektumok.pontok[i].rgb[3],objektumok.pontok[i].lathatosag/10);
.........
glBegin(GL_points);
glVertex3f(objektumok.pontok[i].koo1[1],objektumok.pontok[i].koo1[2],
           objektumok.pontok[i].koo1[3]);
glEnd;
inc(i);
END;
```

A GL a következő lista alapján rajzolja ki az objektumokat: koordinátarenszer, segédvonalak, pontok, egyenesek, szakaszok, merőleges egyenesek, párhuzamos egyenesek, körök, sokszögek, háromszögek, kockák, hasábok, gömbök, hengerek, gúlák, kúpok és végül a síkok. A kirajzolásnál a „while“ ciklus csak azokat az objektumokat engedi a GL-nek kirajzolni, melyeknek van neve, tehát léteznek. Ha ez nem így lenne, a ciklus az egész adatbázist átnézné fölöslegesen és időigényesen, ezzel rontva a grafika kirajzolásának gyorsaságán. Mivel a while ciklus értelmezése „csináld, amíg“, tehát addig foglalkozik az adott objektum recordjával, míg szükséges.

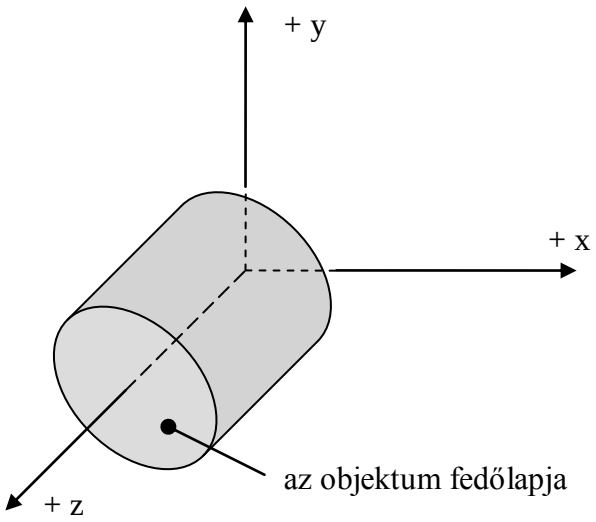
4. Vektoralgebra használata a szerkesztőprogramban

4.1. Az OpenGL lehetőségei

Az OpenGL nem rendelkezik külön paranccsal az objektumok forgatására pontokhoz viszonyítva, ezért egy objektum forgatását az x, y ill. a z tengely körül egy adott szög alatt értelmezzük.

```
.....
gltranslatef(0, 0, 0);
glrotatef(szög, 1, 0, 0);
.....
```

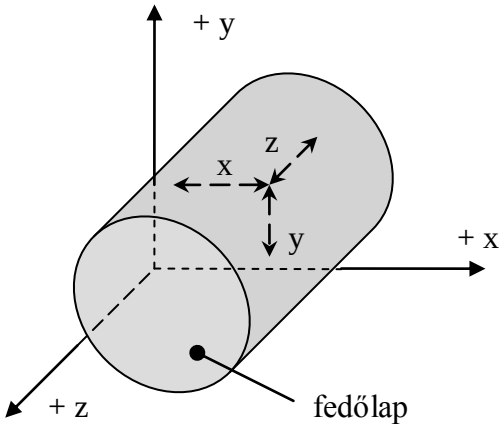
Ezt a parancssort a GL a következőképpen értelmezi: az adott objektumot helyezd a [0, 0, 0] pontba, majd forgasd el az x tengely körül "*szög"*-gel. Amikor az objektumot elhelyezzük valamilyen pontban (gltranslatef), az objektum először mindig a kezdő állapotot veszi fel és csak ez után tolhatjuk-, ill. forgathatjuk el. A következő ábra egy henger kezdő állapotát mutatja be:



Az ábrából látszódik, hogy egy objektum a kezdő állapotban az origóba (O =[0, 0, 0]) kerül és a "+z" (+zvr =(0, 0, 3) vektor) irányába néz. Vagyis egy térelemnél az alaplap középpontja az „origóban" van és a fedőlap középpontja a „z tengelyen" helyezkedik el. Tehát kezdő állapotban a forgatás szöge 0:

```
gltranslatef(0, 0, 0);  {eltolás}
glrotatef(0, 1, 0, 0);  {elforgatás}
```

Ha egy térelemet egy pontba **tolunk** el és a forgatás nagysága az x, y és z tengelyek körül 0, akkor a fedőlap középpontja egy konkrét pontba kerül és az objektum a „+z” vektor felé néz:



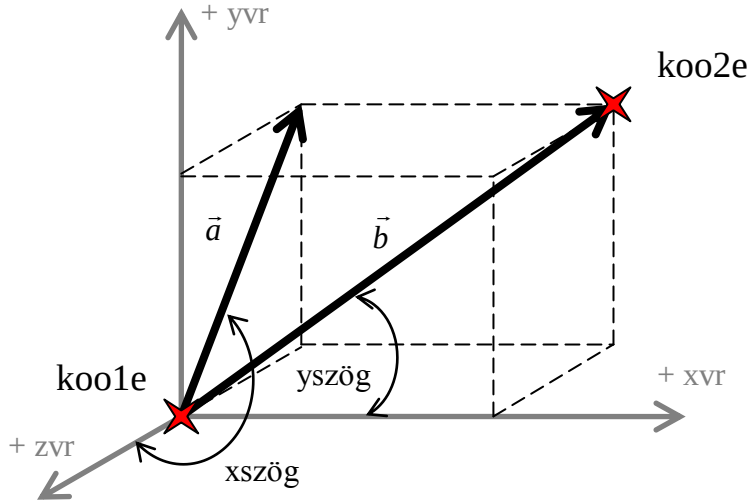

```
{ eltolás az [x,y,z] pontba: }
gltranslatef(x, y, z);

{ elforgatás az x, y és z tengelyen: }
glrotatef(0, 1, 0, 0);
glrotatef(0, 0, 1, 0);
glrotatef(0, 0, 0, 1);
```

Ha egy objektumot eltolunk x, y vagy z irányba, az OpenGL először az objektumot mindig a kezdő állapotában rajzolja ki, majd csak ezután transzformálja (tolja, forgatja, stb.). Gyengébb grafikus kártyánál észrevehetőek a GL transzformációk, vagyis az objektumot egy pillanatra a kezdő állapotban láthatjuk.

4.2. Egyéni megoldások az OpenGL lehetőségeit kihasználva

Ha egy objektumot (testet) szeretnénk megszerkeszteni, többek közt szükségünk van legalább 2 pontra, melyek meghatározzák az alaplap- és a fedőlap középpontját. E két pontból vektorok segítségével kiszámolható a **forgatás** nagysága a tengelyek körül:



+ xvr = (3, 0, 0)
+ yvr = (0, 3, 0)
+ zvr = (0, 0, 3)

koo1e – az objektum alaplapjának középpontja
koo2e – az objektum fedőlapjának középpontja

$\vec{a}$, $\vec{b}$ – vektorok (koordinátáik a „koo1e“ és „koo2e“ pontokból számolódnak ki)

xszög – az $\vec{a}$ vektor és a „+ zvr“ vektor hajlásszöge

yszög – a $\vec{b}$ vektor és a „+ xvr“ vektor hajlásszöge

Az $\vec{a}$ és $\vec{b}$ vektorok koordinátáit a következő képletből számítjuk ki [2]:

$$v = (x_{koo2e} - x_{koo1e}, y_{koo2e} - y_{koo1e}, z_{koo2e} - z_{koo1e})$$

Az „*xszög*“ nagyságát az $\vec{a}$ vektor és a „+zvr“ vektor hajlásszögéből, az „*yszög*“ nagyságát a $\vec{b}$ vektor és a „+xvr“ vektor hajlásszögéből számítjuk ki. A következő képlet a q és w vektorok hajlásszögét mutatja be [2]:

$$\frac{q_1 w_1 + q_2 w_2 + q_3 w_3}{\sqrt{q_1^2 + q_2^2 + q_3^2}\sqrt{w_1^2 + w_2^2 + w_3^2}}$$

- ha $\vec{a}$ = (0, 0, 0), akkor az *xszög* = 0, mert $\vec{a}$ nullvektor
- ha koo2e[y] > koo1e[y], akkor az xszög = 180 +(180–xszög), mert a két vektor hajlásszöge 0 és 180^o közé esik és az OpenGLben a forgatás nagysága 0 és 360^o között lehet.

A képlet alkalmazása után a szögek 0 és 1 közötti értéket vesznek fel, ezért:
„$\cos^{-1}$ szög“ használatával már 0 és 180 közötti értéket kapnak. A programban az „arcos“ (arkusz koszinusz) függvényt használtam: „arcos(0 és 1 közötti érték)*(180/pi);”

Az „xszög“ és az „yszög“ bekerül a GL parancsok közé:

```
gltranslatef(koo1e[1], koo1e[2], koo1e[3]); {koo1e[1] – a koo1e “x” koordinátája, …}
glrotatef(xszog, 1, 0, 0); {az x tengely körüli elforgatás xszog nagysággal}
glrotatef(yszog, 0, 1, 0); {az y tengely körüli elforgatás yszog nagysággal}
```

A következő Delphi parancssor a hengerek tulajdonságait számolja ki, majd a kiszámolt adatok az adatbázisba kerülnek:

```
.....
procedure szamol.hengerszerkeszt;
var j:integer;
    a,b,pz,px:array[1..3]of real; {a,b:vektorok; pz a +z irányvektor; px a +x irányvektor}
    magas:real; {magas-henger magassága}
    m,mm,mmm:real; {segédváltozók a henger magasságának kiszámításához}
    xsz,ysz:real; {fokok kerülnek bele}
begin
pz[1]:=0; pz[2]:=0; pz[3]:=3; {pz=(0,0,3) +z irányvektor}
px[1]:=3; px[2]:=0; px[3]:=0; {px=(3,0,0) +x irányvektor}
j:=1;
WHILE objektumok.hengerek[j].nev<>'' DO
BEGIN
m:=objektumok.hengerek[j].koo1e[1] – objektumok.hengerek[j].koo2e[1];;
mm:=objektumok.hengerek[j].koo1e[2] – objektumok.hengerek[j].koo2e[2];;
mmm:=objektumok.hengerek[j].koo1e[3] – objektumok.hengerek[j].koo2e[3];;
magas:=sqrt((m*m)+(mm*mm)+(mmm*mmm));
objektumok.hengerek[j].magassag:=magas; {2 pont távolsága lesz a magasság}
{az xszog es az yszog számítása}
a[1]:=0;
a[2]:=objektumok.hengerek[j].koo2e[2] – objektumok.hengerek[j].koo1e[2];
a[3]:=objektumok.hengerek[j].koo2e[3] – objektumok.hengerek[j].koo1e[3];
b[1]:=objektumok.hengerek[j].koo2e[1] – objektumok.hengerek[j].koo1e[1];
b[2]:=objektumok.hengerek[j].koo2e[2] – objektumok.hengerek[j].koo1e[2];
b[3]:=objektumok.hengerek[j].koo2e[3] – objektumok.hengerek[j].koo1e[3];
{a két vektor hajlásszögének számítása:}
xsz:=(a[1]*pz[1]+a[2]*pz[2]+a[3]*pz[3])/(sqrt((a[1]*a[1])+(a[2]*a[2])+(a[3]*a[3]))*sqrt((pz[1
]*pz[1])+(pz[2]*pz[2])+(pz[3]*pz[3])));
ysz:=(b[1]*px[1]+b[2]*px[2]+b[3]*px[3])/(sqrt((b[1]*b[1])+(b[2]*b[2])+(b[3]*b[3]))*sqrt((px[1
]*px[1])+(px[2]*px[2])+(px[3]*px[3])));
objektumok.hengerek[j].xszog:=arccos(xsz)*(180/pi);
objektumok.hengerek[j].yszog:=90 – (arccos(ysz)*(180/pi));
if (a[1]=0)and(a[2]=0)and(a[3]=0) then objektumok.hengerek[j].xszog:=0; {mert 0 vektor}
if objektumok.hengerek[j].koo2e[2]>objektumok.hengerek[j].koo1e[2] then
  objektumok.hengerek[j].xszog:=180+(180 – objektumok.hengerek[j].xszog);
inc(j);
END;
end;  .....
```

4.3. A szerkesztőprogram és az OpenGL kapcsolata

Az OpenGL segítségével egyszerű, térben elhelyezkedő 3D-s grafikus primitíveket jeleníthetünk meg a képernyőn. A primitívek segítségével bármilyen alakzatot megtudunk jeleníteni, csupán az OpenGL parancsok jól összehangolt sorára van szükség, ezért egy jó működő adatbázis nélkülözhetetlen. A szerkesztőprogram adatbázisa recordokból épül fel, melyek külön-külön tartalamazzák az egyes objektumok adatait, melyeket a GL parancsoknál használtam fel. A henger kirajzolása GL parancsal:

... var:henger:GLUquadricObj;

gomb:GLUquadricObj;

lemez:GLUquadricObj; ...

... Glucylinder(henger, {alapkör sugara}, {fedőlap sugara}, {magassága}, {alapkör és a fedőlap oldalainak darabszáma}, 1); ...

A „Glucylinder“ GL parancsot kihasználva nemcsak hengert, hanem kockát, hasábot, gúlát és kúpot is megtudunk szerkeszteni. A kockánál az alapkör és a fedőlap oldalainak darabszáma 4, hasábnál pedig n (tetszőleges, a felhasználótól függ) lehet. Gúlánál és kúpnál a fedőlap sugara 0. Ezek után ki kell rajzoltatni az alap- és a fedőlapot:

... Gludisk(lemez, {középső kör sugara}, {külső kör sugara}, {a lemez oldalainak darabszáma}, 1); ...

A „Gludisk“ parancs nélkül üreges objektumot kapnánk. A „lemez“ tulajdonságait a körnél és a sokszögnél is kihasználtam, melyek a síkidomokhoz tartoznak.

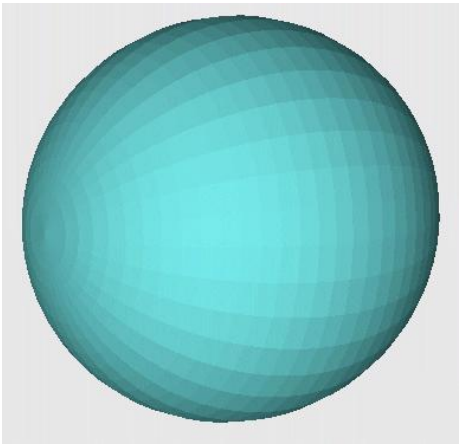
Miután kitöltöttük az adatlapot, az adatok felhasználásra kerülnek és betöltődnek a recordba. A rekordokból kiolvasott adatokat használom fel az OpenGL parancsoknál, növelve ezzel a program gyorsaságát. Tehát amikor az objektumok kirajzolásra kerülnek a GL ablakon, már az objektum összes tulajdonsága ismert.

A szerkesztőprogramban lehetőség van a **minőség** értékének változtatására. Ez azt jelenti, hogy pl. egy kört hány darabra osztjon, vagyis alapköre hány oldalú szabályos poligonból álljon. A GL a kört n oldalú szabályos poligonnal ábrázolja, melyet n darab egybevágó háromszögekre oszt szét. Lehetőség van alacsony, közepes, normális és magas minőség választására. Az alacsony minőségnél 20, a közepesnél 40, a normálisnál 60 és a magasnál 100 darabra osztja szét az objektumot. A programban ezt egy változó (*minoseg*) segítségével oldottam meg, melynek típusa integer, mely felveszi a 20, 40, 60 vagy 100

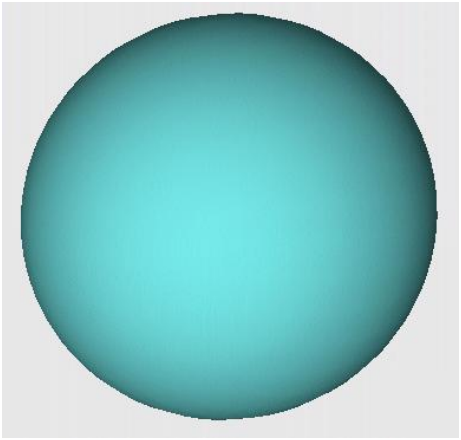
értéket. Ez a változó megjelenik a kör, gömb, henger és kúp GL parancsainál. A minőség nagyban befolyásolja a szerkesztőprogram gyorsaságát, mivel minél több poligon kirjzolása esetén az OpenGLnek több időre van szüksége a kép feldolgozásához és megjelenítéséhez. Azért választottam az 100-as értéket a legnagyobbnak, mert e érték fölött az emberi szem már nem veszi észre a különbséget és fölöslegesen lassítaná ezzel a kirajzolás (grafika) gyorsaságát.

A szakasz és poligon alapelemeket megrajzolhatjuk egyetlen színnel (konstans árnyalás), vagy sok különbözővel (folytonos árnyalás, élsimítás). Az árnyalás az összes objektumot érinti, ezt egy gömb felületén a legmegfelelőbb szemléltetni:

glShadeModel(GL_Flat);



glShadeModel(GL_Smooth);



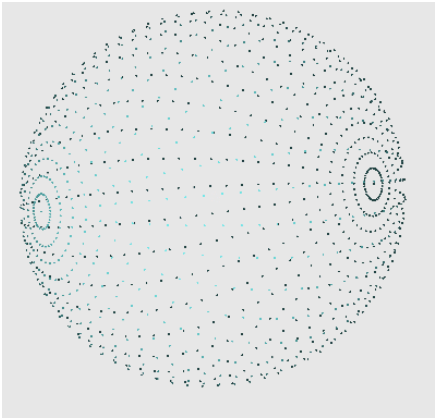
Ezt a lehetőséget a programban egyetlen változó (*arnyalas*) segítségével oldottam meg, melynek típusa byte. Ha az *arnyalas* változó értéke 0 konstans, ha 1 folytonos árnyalást hajt végre a program.

A poligon alapelem megjelenhet egy kitöltött síkidomként, a határát reprezentáló zárt töröttvonalként vagy a csúcspontjait jelölő pontsorozatként is.

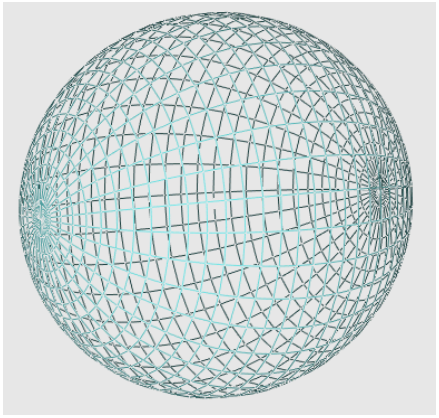
```
glPolygonMode(GL_FRONT_AND_BACK, mode );
```

A mode értéke GL_Point, GL_Line vagy GL_Fill lehet, mely azt jelzi, hogy a poligonnak csak a csúcspontjait kell megrajzolni, a határát kell megrajzolni vagy a belsejét ki kell tölteni. A GL_Fill megjelenítése függ az árnyalástól. A poligon alapelem megjelenítését is egy gömb felületén lehet a legmegfelelőbben szemléltetni:

csúcsok megjelenítése



élek megjelenítése



Ezt a lehetőséget a programban 3 változó (*csucsok, elek, lapok*) segítségével oldottam meg, melyeknek típusa byte. A program indításakor a *lapok* változó felveszi az 1 értéket, a többi 0-át. A program figyeli, hogy mely változó értéke módosul 1-re, mert ez alapján rajzolja ki az objektum megjelenítési stílusát. Ha valamely változó értéke 1-re módosult, akkor a többi felveszi a 0-át, azaz egyidejűleg csak az egyik megjelenítés lehet aktív.

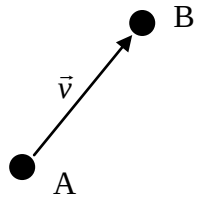
4.4. Az objektumok szerkesztése vektoralgebra segítségével

Egy objektumot többféleképpen is meg tudunk szerkeszteni. A szerkesztéshez bázispontokra van szükségünk, melyek segítségével vektoralgebrai képletek alkalmazásával kiszámolhatóak a kellő tulajdonságok és így megszerkeszthetőek az objektumok.

Az alábbiakban néhány objektum tulajdonságainak számítását mutatom be, melyek közül a többi objektum tulajdonsága is kiszámítható.

Vektor

- 2 pontból:



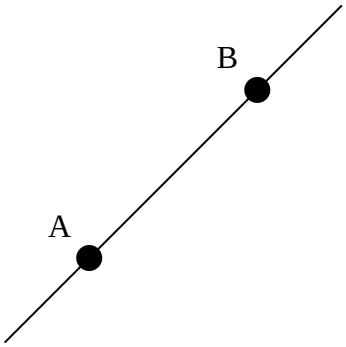
$\vec{v} = (X_B – X_A, Y_B – Y_A, Z_B – Z_A)$

A, *B* – már létező bázispontok, melyek a szerkesztés után az adatbázisban a vektoroknál a koo1e és koo2e felveszik az A és B pontok koordinátáit

Mivel 2 pont egyértelműen meghatároz egy vektort, ezért a fenti képlet alkalmazásával kiszámíthatóak a vektor koordinátái, miközben ábrázoláskor az első (A) pont a vektor kezdőpontja.

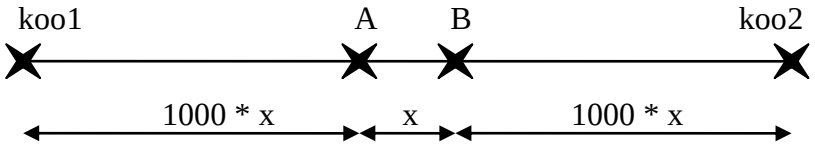
Egyenes

- 2 pontból:



A, B – már létező bázispontok, melyek a szerkesztés után az adatbázisban az egyeneseknél a koo1e és koo2e felveszik az A és B pontok koordinátáit

A 2 bázispont (A, B) felvétele után kiszámolható az egyenes koo1 és koo2 pontjainak koordinátái, mellyel az egyenest a GL megrajzolja. Erre azért van szükség, mert a GL-ben nincs külön parancs az egyenes megszerkesztésére, csak szakaszt szerkeszt 2 pontból, melyeket összeköt. Mivel egyenesről van szó, ezért szükségünk van 2 olyan pontra, melyek összekötése után az általuk meghatározott „szakasz" áthalad a 2 meghatározó bázisponton (A-n és B-n).



A – koo1e pont

B – koo2e pont

1 lépés: $X_B - X_A = w$, $Y_B - Y_A = q$, $Z_B - Z_A = o$;

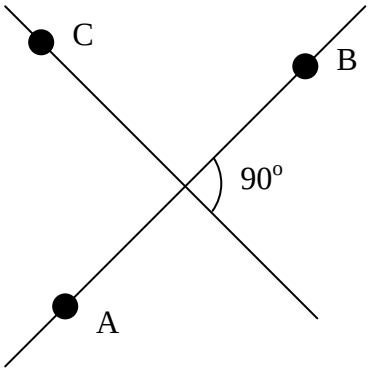
2 lépés: $kx = X_A + 1000*w$, $ky = Y_A + 1000*q$, $kz = Z_A + 1000*o$; ebből:

koo1 = [kx, ky, kz];

Az A és B pontok közötti távolság 1000 szeresére nő, ebből kapjuk meg a koo1-et. A koo2-t ugyanezzel az eljárással oldottam meg, de itt már a két pont koordinátái felcserélődnek a számítások során. A GL a koo1-et és a koo2-t köti össze, ebből egy szakaszt kapunk, mely áthalad a koo1e és koo2e pontokon.

Merőleges egyenes

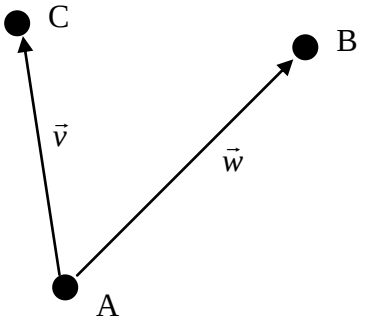
- 1 adott ponton keresztül (C) egy adott egyenesre, vagy szakaszra (AB):



C – egy bázispont, mely nem fekszik az egyenesen vagy szakaszon

A, B – az egyenes vagy szakasz pontjai. Bázispontok, melyekből az egyenes vagy szakasz lett megszerkesztve

A 3 bázispont segítségével 2 vektort tudunk felvenni:



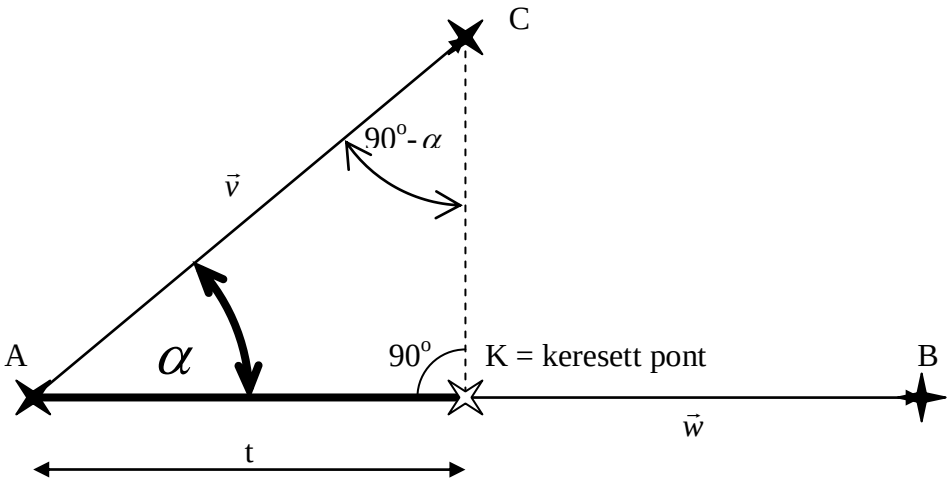
$\vec{v}$ – vektor, koordinátáit a „X_{koo1e}-X_{koo2e}, Y_{koo1e}-Y_{koo2e}, Z_{koo1e}-Z_{koo2e}“ -ből kapjuk

$\vec{w}$ – vektor, koordinátáit a „X_{koo1e}-X_{koo2e}, Y_{koo1e}-Y_{koo2e}, Z_{koo1e}-Z_{koo2e}“ -ből kapjuk

Miután felvettük a két vektort, kiszámoljuk azok hosszát a következő képlet alapján [3]:

$$|v| = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

A $\vec{v}$ és $\vec{w}$ vektor hosszára a derékszögű háromszög számításához van szükség:



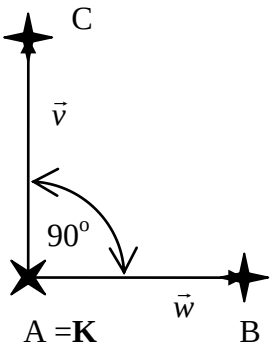
Kiszámoljuk a két vektor ($\vec{v}$ és $\vec{w}$) hajlásszögét („α“). Mivel ismerjük a derékszögű háromszög egy oldalát ($\vec{v}$ hossza) és szögeit („α“, „$90^o - \alpha$“, 90^o), a következő képket alapján kiszámolható a „t“ hossza [3]:

$$t = |v1| * \sin(90 - \alpha)$$

Ha „t” hosszát osztjuk $\vec{w}$ hosszával, kapunk egy “**q**” számot (q = │t│ / │ $\vec{v}$ │), melyet a későbbiekben felhasználunk (arányosság).

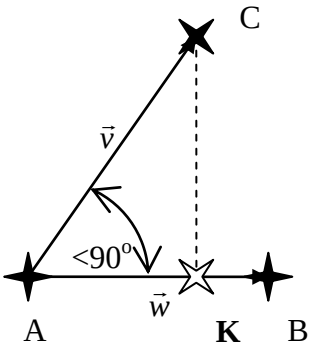
3 eset fordulhat elő:

- 1. eset: $\alpha = 90^o$



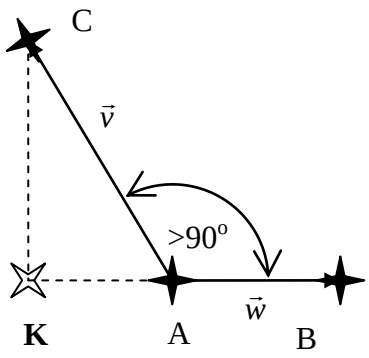
A “K” (keresett) pont koordinátái egyenlőek lesznek az A (koo2e) pont koordinátáival “± (│ $\vec{w}$ │*q)” az α foktól függően.

- 2. eset: $\alpha < 90^o$



A “K” (keresett) pont az AB szakaszon (félegyenesen) fekszik, vagyis a koo2e és a koo3e pontok között.

- 3. eset: $\alpha > 90^o$

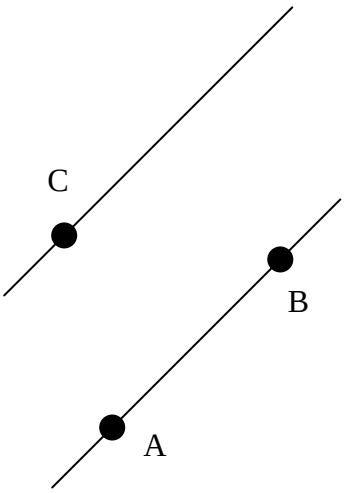


A “K” (keresett) pont nem fekszik az AB szakaszon.

A „C“ (koo1e) és a „K“ pontot összekötve éppúgy járunk el, mint az egyenesnél, vagyis mindkét irányba a köztük lévő távolságot 1000-szeresére növelve megkapjuk a „koo1“ és „koo2“-t, melyet a GL paranccsal összekötünk (gl_line [7]).

Párhuzamos egyenes

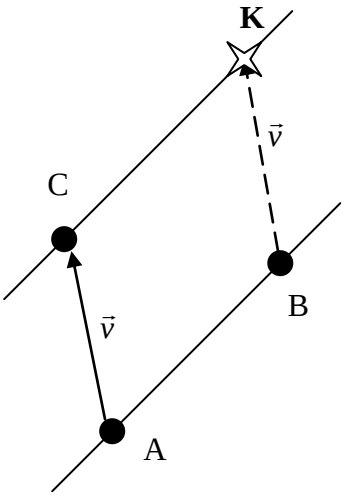
- 1 adott ponton keresztül (C) egy adott egyenesre, vagy szakaszra (AB):



C – egy bázispont, mely nem fekszik az egyenesen vagy szakaszon

A, B – az egyenes vagy szakasz pontjai. Bázispontok, melyekből az egyenes vagy szakasz lett megszerkesztve

A „C“ és az „A“ koordinátáiból egy vektort (“v1“) kapunk:



K – keresett pont, melyet a „C“ ponttal összekötve megkapjuk a párhuzamos egyenest

A $\vec{v}$ vektor koordinátáit hozzáadjuk a „koo3e“ koordinátáihoz. Az összeadás után megkapjuk a „K“ pont koordinátáit:

$X_K = X_{koo3e} + X_{v1}$, $Y_K = Y_{koo3e} + Y_{v1}$, $Z_K = Z_{koo3e} + Z_{v1}$

Sík

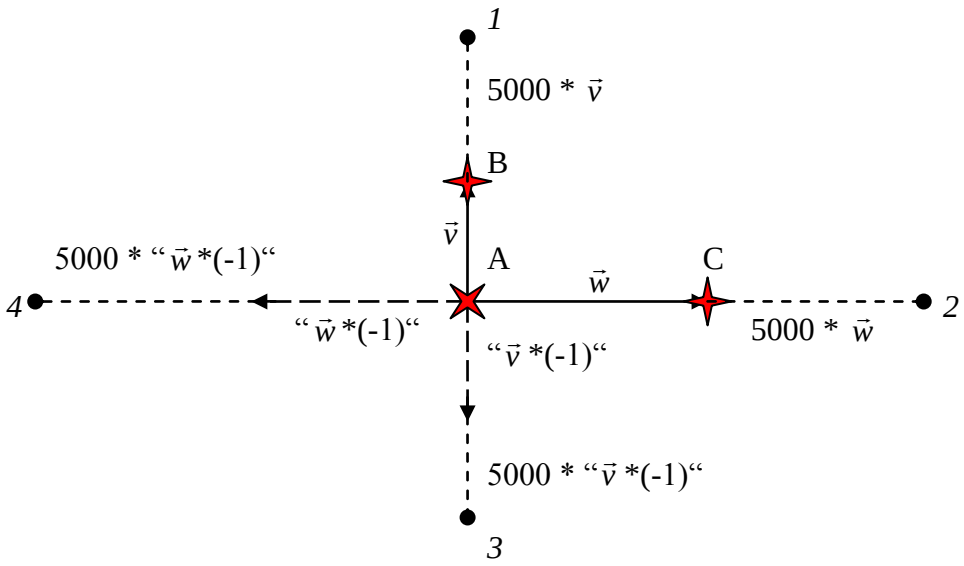
A sík poligon segítségével van szemléltetve. Az adott poligont (síkot) 4 pont összekötésével kapjuk. A 4 pont koordinátáit az A, B és C pontok koordinátáiból számítjuk ki vektorok alkalmazásával. Ezek a pontok olyan messze vannak egymástól a térben, hogyha összekötjük őket és ábrázoljuk, a felhasználó úgy érzékeli a poligont, mintha végtelen nagyságú lenne. Bármely szerkesztési lehetőséget használjuk ki, a pontok kiszámításához mindig az „A“ (koo1e) lesz a kezdőpont.

- 3 pontból:

– az „A“ és a „B“ pontokból egy vektort kapunk ($\vec{v}$), „A“ a kezdőpont.
– az „A“ és a „C“ pontokból egy vektort kapunk ($\vec{w}$), „A“ a kezdőpont.
– mindkét vektor nagyságát 5000-szeresére növelve megkapjuk a poligon 2 csúcspontját (*1*, *2*).
– mindkét vektor koordinátáit beszorozzuk „–1“-el, így ellentett irányú vektorokat kapunk.
– az ellentett irányú vektorokat a „koo1e“ ponthoz hozzáadva és 5000-szeresére növelve nagyságukat, megkapjuk a poligon hiányzó 2 csúcspontját (*3*, *4*).
– a kiszámolt 4 pontból GL parancsok segítségével megszerkeszthető a poligon.

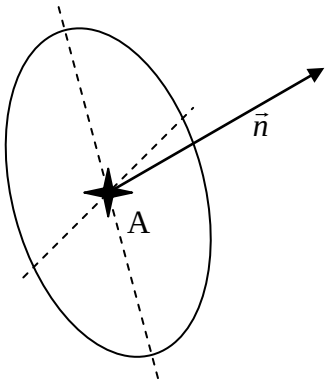
- 1 adott pont és 2 vektor koordinátáiból:

Ha 1 pont és 2 vektor segítségével szeretnénk megszerkeszteni a poligont, akkor a „B“ és a „C“ pontok koordinátáit az „A“ és a két vektor koordinátái összeadása után kapjuk meg. A poligon szerkesztése a fenti felsorolás alapján történik.



Kör, sokszög

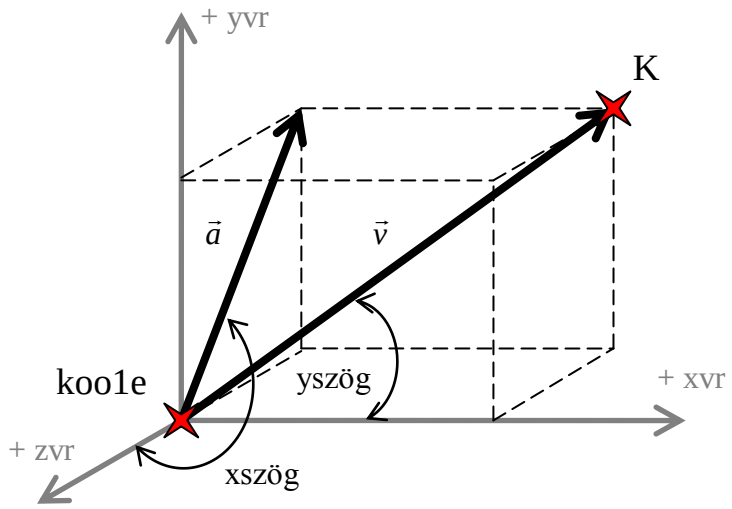
- 1 adott pontból, sugárból és 1 vektorból:



A – középpont

$\vec{n}$ – normálvektor, mely merőleges az átmérőkre

Az $\vec{n}$ normálvektort hozzárendeljük a „koo1e“ ponthoz, így megkapjuk a „K“ pontot, melyből további vektorokat vehetünk fel. A felvett vektorokból meghatározzuk azokat a szögeket, melyek nagyságával az adott objektum az x és y tengelyek körül el lesz forgatva.



+ xvr = (3, 0, 0)

+ yvr = (0, 3, 0)

+ zvr = (0, 0, 3)

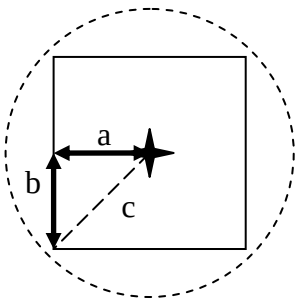
A sokszögnél is az említett elv alapján járunk el, azzal a különbséggel, hogy mi magunk állíthatjuk be a sokszög alapkörének az oldalainak számát.

Az objektum forgatásának módjáról a 4. fejezetben (Vektoralgebra használata a szerkesztőprogramban) olvashatunk.

Kocka

- 2 adott pontból és 1 irányvektorból:

A kockát a henger tulajdonságaiból szerkesztjük meg. Az alaplap- és a fedőlap középpontja közti távolságból megkapjuk a kocka magasságát (élének hossza), melyből Pitagorasz tételének segítségével kiszámítható az a sugár, melyből a kocka alap- és fedőlapja meg lesz szerkesztve. Mivel a és b is egyenlő a kocka magasságának felé vel, a kocka alap- és fedőlapján lévő átlójának fele: $c = \sqrt{a^2 + b^2}$, ahol a = b. A kért irányvektorral a kocka egyik alapélének irányát adjuk meg.



Gömb

- 2 adott pontból és 1 irányvektorból:

Ha a gömböt 2 pontból szeretnénk megszerkeszteni, a két pont közötti távolság lesz a sugár, melyet a következő képlet alapján számolunk ki [3]:

$$távolság = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + (a_3 - b_3)^2}$$

5. A GeoGL3D v2.0 - térgeometriai szerkesztőprogram

A szerkesztőprogram neve a **Geo** (geometria), **GL** (OpenGL), **3D** (3 dimenzió -tér) származik. A **v2.0** a szerkesztőprogram verziójának a számát jelöli. A program megírása Delphi 2005-ös verzióban OpenGL parancsok segítségével történt. Célja a különböző sík-, ill. téridomok megjelenítése különböző vetítési módokban.

5.1. A szerkesztőprogram bemutatása

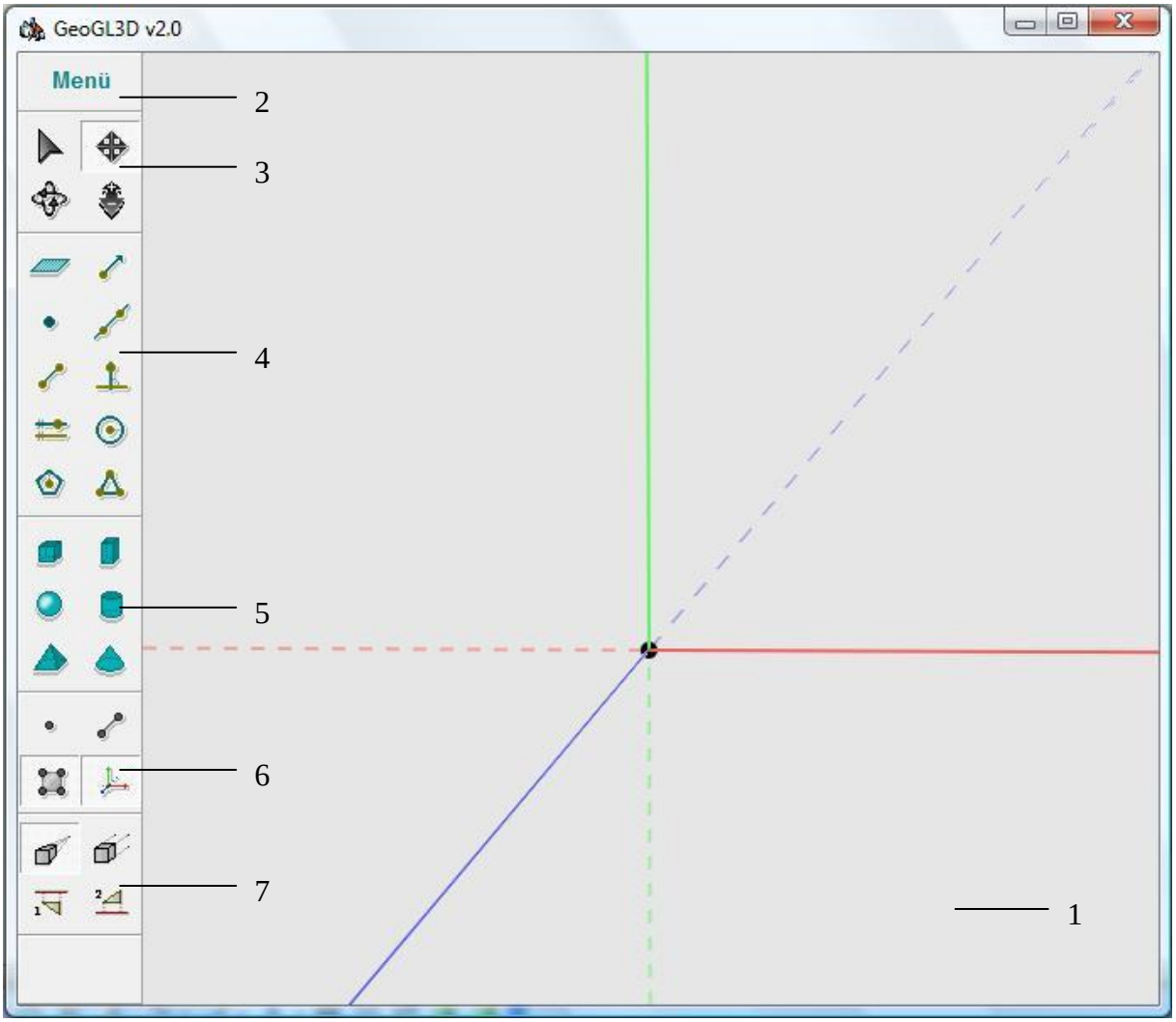
A legfontosabb dolog az, hogy először ún. „bázispontokat“ kell létrehoznunk, majd ezekre épülhetnek az objektumok, pl. 2 pontra szakasz, egyenes, stb. Mivel térben dolgozunk, fontos, hogy az objektumokat bármilyen kameraállásból meg tudjuk jelenítteni, ezért a szerkesztőprogramban magunk választhatjuk ki a kamera stílusát. Választhatunk az *alap, csúszás* (a térben lévő csúszás a képernyő x,y tengelyéhez viszonyítva), *forgatás* (az OpenGL koordinátarendszer x és y tengelye körül) és +/– (nagyítás, kicsinyítés: a térben előre ill. hátra lévő mozgás) közül. A jobb áttekinthetőséget és egyértelműséget a programban a vonalvastagság, a szín és a láthatóság (átlátszóság) szabályozza. Továbbá kiválaszthatjuk azt is, hogy az objektumokat milyen stílusban szeretnénk megjeleníttetni. Lehetőség van csúcsok-, élek- és lapok megjelenítésére (egyidejűleg csak az egyikre), továbbá a koordinátatengelyek és a segédvonalak megjelenítésére is.

Magunk választhatjuk ki az egész „munkaablak“ színét és stílusát. A munkaablak (főablak) tartalmaz eszköztárat, melyet szintén a saját munkastílusunkhoz igazíthatjuk, mivel lehetőség van az eszköztár színének és elhelyezkedéének módosítására, melyek a program indításakor visszaolvasódnak.

Mivel a szerkesztőprogram alkalmas ábrázoló geometriai elemek megjelenítésére, ezért a Monge – projekció első és második képsík beépítése elengedhetetlen volt a szerkesztőprogram fejlesztése során. A Monge projekción kívül választhatunk még centrális ill. merőleges vetítések közül.

5.2. A szerkesztőprogram használata

Indításkor a fő- és az OpenGL (szerkesztés) ablak felveszi a képernyő maximális méretét. A szerkesztőprogram indítása után a következőt kell, hogy lássuk:



1 – Szerkestés ablak: itt rajzolódik ki a szerkesztés. Ez egyben az OpenGL ablaka

2 – Menü megjelenítése

3 – Kamera eszköztár. A kameramódokat tartalmazza

4 – Sík eszköztár: A síkidomokat tartalmazza

5 – Tér eszköztár: A térelemeket tartalmazza

6 – Megjelenítés eszköztár. A megjelenítéseket tartalmazza

7 – Vetítés eszköztár. A vetítéseket tartalmazza

5.3. Menüpontok

 **Új projekt** – Egy új projektet hoz létre, mely a kezdő objektumokat tartalmazzza, melyeket egy külső fájlból olvas be. A kezdő objektumok a következők:

- 6 vektor: +xvr {O pont, (3,0,0) koordináták}, –xvr {O pont, (–3,0,0)}, +yvr {O pont, (0,3,0)}, –yvr {O pont, (0,–3,0)}, +zvr {O pont, (0,0,3)}, –zvr {O pont, (0,0,–3)}
- 1 pont: origó (O=[0,0,0])

 **Projekt beolvasása** – Egy létező GeoGL3D v2.0-es fájlt (*.gl2) nyit meg.

 **Projekt mentése** – A projektet GeoGL3D v2.0-es fájlformátumba (*.gl2) menti el.

 **Kilépés** – Kilép a szerkesztőprogramból.

 **Színek** – A felhasználó testreszabhatja a szerkesztőprogram munkafelületét a színek beállításával. Beállíthatja a háttérszínt, ablak háttérszínét, az x,y,z tengelyek-, rács-, üzenetablakok-, eszköztárak színét. Az “Eredeti” nyomógomb segítségével visszaállítja az eredeti színeket. A megváltoztatott tulajdonságok külső fájlban tárolódnak. A szerkesztőprogram indításakor ezek az adatok visszaolvasódnak.

 **Eszköztár elhelyezkedése** – Az eszköztár pozícióját változtathatjuk a fő ablakon belül.

 **Kameramód** – A felhasználó 4 kameramód közül választhat: alap, csúszás(a térben lévő csúszást teszi lehetővé a képernyő x,y tengelyéhez viszonyítva), forgatás(a koordinátarendszer x és y tengelyek körüli forgatást teszi lehetővé), +/–(nagyítás, kicsinyítés: a térben előre ill. hátra lévő mozgást teszi lehetővé).

 **Megjelenítés** – Az objektum megjelenítési stílusát teszi lehetővé. A felhasználó szabályozhatja az alakzat megjelenítését. Lehetőség van a csúcsok-, élek- és lapok megjelenítésére. Továbbá kiválaszthatja a koordinátarendszer és a segédvonalak elrejtését, ill. megjelenítését.

 **Nyelv** – A szerkesztőprogram beépített nyelvei közül választhatunk (magyar és angol). A beállított nyelv a szerkesztőprogram következő indítása után is aktuális lesz.

 **Tengelyek/Rács módosítása** – A tengelyek és a rács módosítására szolgál.

- A *tengelyek* (x,y,z) láthatóságát, vonalvastagságát, ill. vonaltípusát lehet megváltoztatni.
- A *rács* beosztási távolságát x,y,z tengelyek szerint, a vonalvastagságukat, vonaltípusukat, ill. láthatóságukat lehet megváltoztatni.

Az "Eredeti" nyomógomb segítségével visszaállítja az eredeti tulajdonságokat (adatokat). A megváltoztatott tulajdonságok külső fájlban tárolódnak. A szerkesztőprogram indításakor ezek az adatok visszaolvasódnak.

 **Átlátszóság** – Az átlátszóság ki- és bekapcsolása. Ha nincs bekapcsolva az átlátszóság, akkor az összes objektum látható lesz, attól függetlenül hogy a "láthatóság" tulajdonsága milyen értékű.

 **Pontok stílusa** – A bázispontok stílusát határozza meg. Lehetnek kör és négyzet alakúak.

 **Árnyalás** – Meghatározza, hogy a szakasz és poligon alapelemek hány színben jelenjenek meg. A konstans árnyalásnál egy, a folytonos árnyálásnál több sok különböző színben jelenik meg az objektum.

 **Minőség** – A kirajzolandó objektum minőségét lehet beállíttani. A felhasználó választhat az alacsony, közepes és magas minőségek közül.

 **Kezdő nézet** – A szerkesztőprogram előre definiált kamera pozíciója.

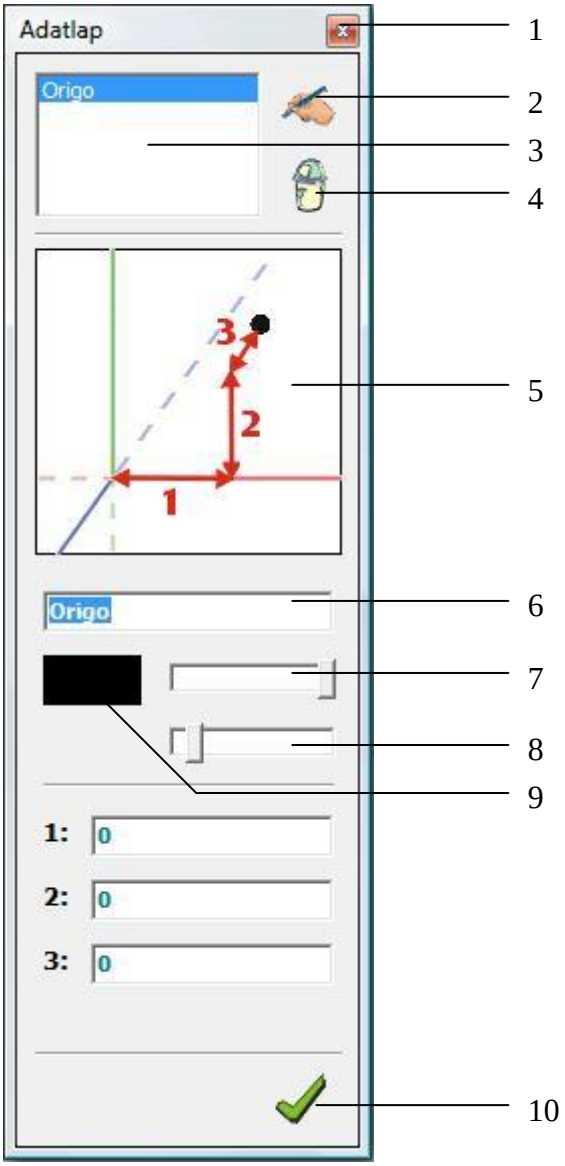
 **Egyéni forgatás** – A felhasználó maga választhatja ki a forgatás nagyságát az x és y tengelyek körül a kezdő nézethez viszonyítva.

 **Vetítés** – A vetítést (perspektívát) változtathatjuk. A felhasználó 4 vetítési mód közül választhat: centrális, merőleges, Monge projekció – első képsík és Monge projekció – második képsík.

 **Névjegy** – A szerző adatait jeleníti meg.

5.4. Adatlap

A fő ablak, vagy a szerkesztés ablak az egér jobb klikkje után, ill. a sík, vagy a tér eszköztár ikonjai klikkje után jelenik meg az adatlap. Az adatlap tartalmazza az összes felvett objektum tulajdonságait. Vagyis a nevét, színét, láthatóságát, vastagságát, ill. hogy mely más objektum(ok) segítségével lett megszerkesztve, vagy sajátos adatait (pl. pontnál). Lényegében az adatlap tartalmazza az adatbázist.



1 – Elutasítás, bezárás. Az adatlap elrejtésére szolgál.

2 – Új objektum hozzáadása. Egy új objektum kerül a listába. Az objektumot automatikusan elnevezi, beállítja az objektum színét, láthatóságát és vastagságát az adott objektumtól függően (pl. a síknál a láthatóság kezdő értéke 2).

3 – A kiválasztott objektumtípus neveinek listája. A névre jobb egérgomb klikkelés után a szerkesztés ablakban az objektum nagysága és színe megváltozik, így tájékoztatja a felhasználót, hogy mely objektum adatlapja látható.

4 – Az objektum törlése. Kitörli az adott objektumot a listából, továbbá minden olyan objektumot, mely a kitörölendő objektum segítségével lett megszerkesztve (pl. ha egy pont kerül törlésre és ez a pont egy kör középpontja, akkor a kör is törlésre kerül).

5 – Az adott objektum szerkesztéséhez szolgáló információs kép. A kép tájékoztatja a felhasználót az objektum megszerkesztéséhez szükséges információkra. Tájékoztat, hogy az adott objektum mely más objektum(ok) segítségével szerkeszthető meg, esetleg sajátos tulajdonságairól (pl. pont x,y,z koordinátái).

6 – Az objektum neve, mely utólag is megváltoztatható.

7 – Az objektum láthatósága. 0..10 között vehet fel értéket. Ez egyben az objektum átlátszóságát biztosítja (pl. ha értéke 0, az objektum teljesen átlátszó, tehát rejtve marad).

8 – Az objektum vastagsága (nagysága). A pontnál, egyenesnél, szakasznál, merőleges- és párhuzamos egyeneseknél jelenik meg. 1..10 között vehet fel értéket.

9 – Az objektum színe.

10 – Az objektum jóváhagyása, szerkesztése. Az adatlap adataiból megszerkeszti az objektumot.

5.5. Az objektumok szerkesztéseinek lehetőségei

Egy objektumot esetenként többféle képpen is meg lehet szerkeszteni.

 **Sík –** Kétféleképpen adhatjuk meg: 3 különböző ponttal, melyek nem fekszenek egy egyenesen, vagy 1 ponttal és 2 vektorral. Az utóbbinál az információ sor első képe alatt lévő listából kell kiválasztani a pontot, különben hibaüzenetet jelenik meg.

 **Vektor –** Kétféleképpen adhatunk meg egy vektort: 1 pont és 1 vektorral, vagy 2 ponttal. Az utóbbinál az információs sor első képe alatt lévő listából kell kiválasztani a pontot, különben hibaüzenetet jelenik meg, továbbá a szerkesztőprogram kiszámolja a vektor koordinátáit és megjeleníti azokat.

 **Pont –** x, y és z koordinátáival.

 **Egyenes,**  **szakasz –** 2 pont segítségével adhatjuk meg.

 **Merőleges-,**  **párhuzamos egyenes –** 1 egyenessel, vagy szakasszal és egy rajtuk kívül fekvő ponttal adhatjuk meg.

 **Kör,**  **sokszög –** 1 ponttal (középpont), sugárral, beosztással (sokszögnél, ez az oldalak számát jelenti, körnél a minőség értéke megegyezik az oldalak számával) és 1 vektorral (meghatározza hogy az objektum síkjára mely vektor legyen merőleges) adhatjuk meg.

 **Háromszög –** 3 ponttal adhatjuk meg, melyek nem fekszenek egy egyenesen.

 **Kocka –** 2 ponttal (az alaplap és a fedőlap középpontjai) és 1 vektorral (az alaplapján lévő egyik él iránya) adhatjuk meg.

 **Hasáb (szabályos n-oldalú hasáb) –** 2 ponttal (az alaplap ill. a fedőlap köréírt körének sugara), sugárral (az alaplap és a fedőlap egyik átlójának a fele), beosztással (az alaplap és fedőlap oldalainak száma) és 1 vektorral (az alaplapján lévő egyik él iránya) adhatjuk meg.

 **Gömb –** 2 ponttal (középpont és egy kerületi pont), vagy 1 ponttal (középpont) és sugárral szerkeszthetjük meg.

 **Henger –** 2 ponttal (az alaplap és a fedőlap középpontjai) és sugárral (az alaplap és a fedőlap sugara) adhatjuk meg.

 **Gúla (szabályos n-oldalú gúla) –** 2 ponttal (az alaplap középpontja és a csúcsa), sugárral (az alaplap egyik átlójának a fele), beosztással (az alaplap oldalainak száma) és 1 vektorral (az alaplapján lévő egyik él iránya) adhatjuk meg.

 **Kúp –** 2 ponttal (az alaplap középpontja és a csúcsa), sugárral (az alaplap sugara) adhatjuk meg.

6. Irodalomjegyzék

Irodalmak:

[1] STOFFOVÁ, V.: *Databázové systémy v sledovani úrovne vedomostí študentov.* Hradec Králové : Vysoká škola pedagogická Hradec Králové, 1995. 1. vyd.

[2] HAJÓS, GY.: *Bevezetés a geometriába*, Nemzeti tankönyvkiadó Rt., Budapest, 1999, ISBN 963 19 01165

[3] OBÁDOVICS, J. GY.: *Matematika.* Tizennyolcadik, javított kiadás, Scolar Kiadó, 1994, ISBN 963-9534-28-5

[4] MARTZ, P.: *OpenGL röviden*, Kiskapu Kft., 2007, ISBN 9789639637252

[5] ASTLE D.: *More OpenGL Game Programming*, Thomson, 2005, ISBN 9781592008308

Internetes források:

[6] PROG.HU: *Informatikáról szakszerűen*, forrás: http://www.prog.hu/, 2008.július

[7] SULACO: *OpenGL Projects*, forrás: www.sulaco.co.za/opengl.htm, 2006.október

[8] DELPHI PAGES, forrás: http://www.delphipages.com/, 2008.augusztus

[9] IIT, BME Irányítástechnikai és Informatika Tanszék, Benedek Balázs: *OpenGL*, forrás: http://www.iit.bme.hu/~benedek/mirrors/OpenGL/opengl/, 2006.július

[10] DELPHI FORFUN HOME, forrás: http://www.delphiforfun.org/, 2008.november