Debreceni Egyetem
Informatikai Kar

Térbeli objektumok vetítésének módszerei

Témavezető:
Dr. Kovács Emőd
tanszékvezető főiskolai docens

Készítette:
Hank Sándor
programtervező
matematikus

Tartalomjegyzék

Bevezetés

A számítógépes grafika, a testek háromdimenziós modellezése – anélkül, hogy észrevennénk – jelen van a közvetlen környezetünkben. A mindenapjainkban használt tárgyak megtervezéséhez, legyártásához komoly grafikai programokat használnak. A filmek, műsorok, reklámok készítéséhez, melyeket a TV-ben sugároznak, szintén grafikai eszközöket használnak. Számtalan területen alkalmazzák ezt a tudományágat: a térképkészítéstől a design tervezésen át a multimédiás eszközök gyártásáig… A mai számítógépes technológiát kihasználva olyan minőségű képeket tudnak a szakemberek előállítani, melyek megtévesztően hasonlítanak a valós világ képeihez.

A komputergrafika világához harmadéves hallgatóként kerültem közelebb Dr. Kovács Emőd előadásain és gyakorlatain. Olyan alapokra tettem szert, melyek eszközként szolgálta a szakterület komolyabb megismerésében.

A diplomamunka témájának választásakor a célom az volt, hogy a hasonlóan „kezdő cipőben” járó, de a grafika iránt érdeklődő hallgatóknak segítséget nyújthasson a dolgozatom. Magamból kiindulva próbáltam úgy összeállítani ezeket az oldalakat és a program szövegét, hogy az olvasó ne vesszen el a komoly matematikai háttér tanulmányozásában, a gyakorlati megvalósításra fektettem a hangsúlyt. Azonban néhány alapvető matematikai összefüggést feltétlen ismernünk kell ahhoz, hogy egy jól működő grafikai programot tudjunk írni. Ez volt az oka annak, hogy az első fejezetet matematikai bevezetőnek szántam, viszont itt is arra törekedtem, hogy ne fussak bele az összefüggések rejtelmeibe, elegendőnek találtam az algoritmusok lényegének és folyamatának leírását. Természetesen, akit komolyabban érdekel a matematikai háttér, az utána tud nézni azokban a forrásokban, amelyeket az Irodalomjegyzékben gyűjtöttem össze. A program írásánál is arra törekedtem, hogy minél érthetőbben kódoljam le az algoritmusokat. Eddigi munkáim által szerzett tapasztalatokat felhasználva a program forrását magyarázatokkal láttam el, hogy ne csak számomra legyen érthető hetekkel, hónapokkal később, mit, miért alkalmaztam, hanem azok számára is, akik szeretnék a tanulmányaikban, munkáikban felhasználni a szoftver forrását.

A dolgozat végigvezeti az olvasót a testek számítógépes reprezentálásától a különböző transzformációkon keresztül a főbb megjelenítési technikákig. Leginkább azokra a módszerekre fektettem hangsúlyt, melyeket én is alkalmaztam az oktatóprogram készítése során. Bízok abban, hogy a munkám segíteni fog azoknak, akik szeretnék megismerni a komputergrafika alapjait. Akkor fogjunk is hozzá…

1. A komputergrafika matematikai alapjai

Ebben a fejezetben röviden felsorolom azokat a matematikai műveleteket, amelyek nélkül nem tudunk boldogulni a komputergrafikában. Fontos szerepet játszanak a mátrixok, melyekkel transzformációkat, objektumokat tudunk leírni, és a vektorok, melyekkel pontokat, határoló vonalakat, irányokat tudunk megadni.

A vektorok és mátrixok pontos matematikai definícióját megtalálhatjuk Dr. Bácsó Sándor Diszkrét matematika című jegyzetében. A most tárgyalt témakörben leegyszerűsített formában vegyük át azokat a vektor és mátrix műveleteket, tulajdonságaikat, amiket használni fogunk.

1.1. Vektorok

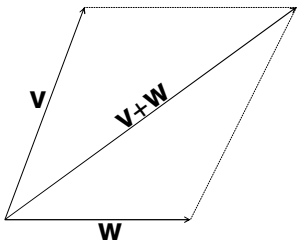
A három dimenziós térben a pontokat legalább három dimenziós vektorok segítségével tudjuk megadni:

$\mathbf{v} = (x, y, z)$

Vektorok összeadása, kivonása:

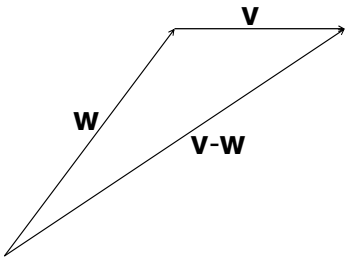
Legyen adott egy $\mathbf{v} = (x_v, y_v, z_v)$ és $\mathbf{w} = (x_w, y_w, z_w)$ vektorok.

$\mathbf{x} = \mathbf{v} + \mathbf{w} = (x_v + x_w, y_v + y_w, z_v + z_w)$ (1. ábra)



1. ábra

$\mathbf{x} = \mathbf{v} - \mathbf{w} = (x_v - x_w, y_v - y_w, z_v - z_w)$ (2. ábra)



2. ábra

Vektor skalárral való szorzása:

$\mathbf{v} = (x_v, y_v, z_v)$; s$\in R$

s $\cdot$ $\mathbf{v} = (s \cdot x_v, s \cdot y_v, s \cdot z_v)$

Vektorok hossza:

Legyen adott egy $\mathbf{v} = (x_v, y_v, z_v)$.

$|\mathbf{v}| = \sqrt{x_v^2 + y_v^2 + z_v^2}$

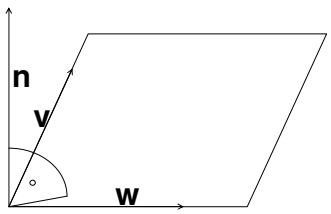
Ha a **v** vektor hossza 1, akkor a **v** vektor normalizált vektor. A programjaink során sokszor csak egy irányt akarunk leírni, ehhez viszont elegendő, ha a vektor normáját tároljuk.

A **v** vektor normája:

$\mathbf{u} = \mathbf{v} / |\mathbf{v}|$.

Normál vektorok:

A síkban két különböző irányú vektor meghatároz egy síkot. A komputergrafikában ha fel akarjuk írni egy test egy oldalát, nem elég, ha megadjuk a határoló poligonját, tudnunk kell, hogy az oldal melyik irányba néz. Ezt könnyen tárolhatjuk a lap síkjára merőleges vektor segítségével, vagyis a normálvektorával. (3. ábra)



3. ábra

Két nem egybeeső vektor normálvektorát a két vektor vektoriális – más néven keresztszorzatával tudjuk felírni:

Adott: $\mathbf{v} = (x_v, y_v, z_v)$ és $\mathbf{w} = (x_w, y_w, z_w)$.

$\mathbf{n} = \mathbf{v} \times \mathbf{w} = (y_v z_w - z_v y_w)\mathbf{i} + (z_v x_w - x_v z_w)\mathbf{j} + (x_v y_w - y_v x_w)\mathbf{k}$, ahol

$\mathbf{i} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, $\mathbf{j} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, $\mathbf{k} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ egységvektorok.

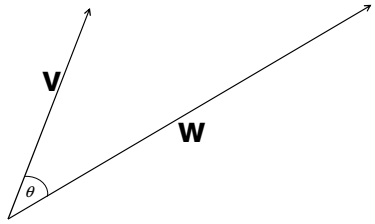
Két vektor által bezárt szög, skaláris szorzat:

Ha egy test egy lapját árnyalni szeretnénk, és ismerjük az oldal normálvektorát, valamint a fényforrás és az oldal elhelyezkedését, ki tudjuk számítani a fénysugár, milyen szögben esik az oldalra. Minél jobban közelít a derékszögöz, annál világosabb lesz az oldal.

Adott két geometriai vektor: $\mathbf{v} = (x_v, y_v, z_v)$ és $\mathbf{w} = (x_w, y_w, z_w)$.

A két vektor skaláris szorzata:

$\mathbf{v} \cdot \mathbf{w} = |\mathbf{v}| \cdot |\mathbf{w}| \cdot \cos\theta$, ahol $|\mathbf{v}|$ és $|\mathbf{w}|$ a két vektor hossza, $\cos\theta$ pedig a vektorok által bezárt szög. (4. ábra)



4. ábra

Három dimenziós vektorok esetén a vektorok derékszögű koordinátái segítségével az alábbi módon is megkaphatjuk két vektor skaláris szorzatát:

$$\mathbf{x} = \mathbf{v} \cdot \mathbf{w} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = v_1 w_1 + v_2 w_2 + v_3 w_3$$

Tulajdonságok:

$\mathbf{v} \cdot \mathbf{w} > 0 \Leftrightarrow \theta < 90°$

$\mathbf{v} \cdot \mathbf{w} = 0 \Leftrightarrow \theta = 90°$

$\mathbf{v} \cdot \mathbf{w} < 0 \Leftrightarrow \theta > 90°$

1.2. Mátrixok

Mátrixok segítségével írjuk fel a különböző transzformációkat. Néhány fontosabb művelet mátrixokkal:

Legyen:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \vdots \\ a_{n1} & & \dots & a_{nn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & & \\ \vdots & & \ddots & \vdots \\ b_{n1} & & \dots & b_{nn} \end{pmatrix}, \quad \mathbf{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_4 \end{pmatrix}$$

$$\mathbf{A} + \mathbf{B} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & & \\ \vdots & & \ddots & \vdots \\ a_{n1} + b_{n1} & & \dots & a_{nn} + b_{nn} \end{pmatrix}$$

$$\mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \vdots \\ a_{n1} & & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & & \\ \vdots & & \ddots & \vdots \\ b_{n1} & & \dots & b_{nn} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & & \\ \vdots & & \ddots & \vdots \\ c_{n1} & & \dots & c_{nn} \end{pmatrix},$$

ahol $c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$.

A mátrixműveletek megvalósítása a 6.1-es mellékletben található.

$$\mathbf{A} \cdot \mathbf{v} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \vdots \\ a_{n1} & & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} c_1 & c_2 & \cdots & c_n \end{pmatrix}, \text{ ahol } c_i = \sum_{j=1}^{n} a_{ij} \cdot v_j .$$

1.3. Egyenesek, síkok

1.3.1 Az egyenes paraméteres egyenlete

A program írása során gyakran meg kell határoznunk élek, lapok metszéspontját, ehhez szükségünk van az egyenes és a sík egyenletére.

A lap egy éléhez tartozó egyenes egyenletét fel tudjuk írni, ha tudjuk az él két végpontjának koordinátáját. Ha az egyenest a paraméteres egyenletével írjuk le, akkor könnyedén meg tudjuk mondani egy pontjáról, hogy az az él által meghatározott szakaszba esik-e, vagy sem.

Legyen a szakasz két végpontja:

$A = (x_a, \quad y_a, \quad z_a)$, $B = (x_b, \quad y_b, \quad z_b)$.

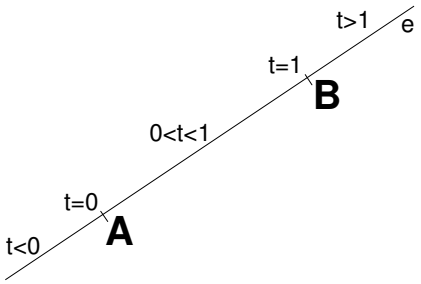
Az A és B pontok által meghatározott egyenes paraméteres egyenlete:

$$e(t): \begin{array}{l} x = (x_b - x_a)t + x_a \\ y = (y_b - y_a)t + y_a \\ z = (z_b - z_a)t + z_a \end{array}$$, mátrix-reprezentációja: $\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{pmatrix}$.

Ezáltal felírtuk az AB szakasz egyenletét is. Ha $0 \le t \le 1$, akkor az $(x, \quad y, \quad z)$ pont az A és B pontok által meghatározott szakasz eleme. (5. ábra)

$t = 0 \Rightarrow A$

$t = 1 \Rightarrow B$



5. ábra

Az egyenes paraméteres egyenletének előállításáról forráskód a 6.2-es mellékletben található.

1.3.2 A sík paraméteres egyenlete

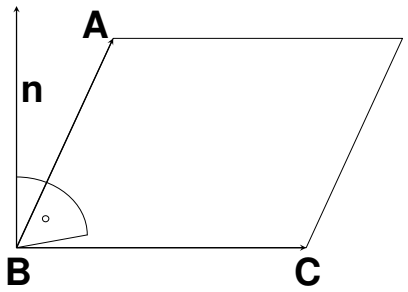
Ha egy lap síkjának az egyenletét szeretnénk meghatározni, azt is egyszerűen fel tudjuk írni, ha ismerjük a lap legalább három nem egybeeső pontját (6. ábra).

$A=(x_a,\quad y_a,\quad z_a),\ B=(x_b,\quad y_b,\quad z_b),\ C=(x_c,\quad y_c,\quad z_c).$

Az A, B és C pontok által meghatározott sík paraméteres egyenlete:

$$s(t_1,t_2):\begin{aligned} x &= (x_a - x_b)t_1 + (x_c - x_b)t_2 + x_b \\ y &= (y_a - y_b)t_1 + (y_c - y_b)t_2 + y_b \\ z &= (z_a - z_b)t_1 + (z_c - z_b)t_2 + z_b \end{aligned}$$

, mátrix-reprezentációja: $\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix}$.



6. ábra

A 6.3-as mellékletben található, hogyan valósítjuk meg a sík paraméteres egyenletének előállítását.

Ahhoz, hogy az egyenesek és síkok metszéspontját meghatározzuk a térben, az egyenleteikből felírt lineáris egyenletrendszert kell megoldanunk.

1.3.3 Lineáris egyenletrendszerek numerikus megoldása

A programom írása során kevés egyenletből és ismeretlenből álló lineáris egyenletrendszert kellett megoldanom, elégségesnek találtam a Gauss-elimináció módszerének használatát.

Írjuk fel az egyenletrendszer egyenleteit:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots a_{2n}x_n &= b_2 \\ &\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots a_{nn}x_n &= b_n \end{aligned}$$

Az egyenletrendszer mátrix-reprezentációja:

$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$, vagyis $\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \vdots \\ a_{n1} & & \cdots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$

Az egyenletrendszert felírhatjuk egyetlen mátrixban is:

$\mathbf{A} \mid \mathbf{b} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & & & b_2 \\ \vdots & & \ddots & \vdots & \vdots \\ a_{n1} & & \cdots & a_{nn} & b_n \end{pmatrix}$

Hozzuk a Gauss-elimináció módszerével az egyenletrendszer mátrixát háromszög alakra. Vigyázzunk a lebegőpontos számokkal végzett műveletek pontatlanságára.

Megjegyzés: A programban egy egyszerű módszert alkalmazok ennek a kiküszöbölésére. Ha egy szám abszolút értéke kisebb egy meghatározott küszöbszámnál, akkor azt nullának veszem.

$\mathbf{A} \mid \mathbf{b} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ 0 & a_{22} & & & b_2 \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{nn} & b_n \end{pmatrix}$

Majd a visszahelyettesítés módszerével megkaphatjuk az $\underline{x}$ vektort:

$$x_i = \frac{a_{ii}}{b_i - \sum_{j=i+1}^{n} x_j}$$

A Gauss-elimináció programnyelvi megvalósításáról a 6.4-es mellékletben olvashatunk.

1.3.4 Egyenesek metszéspontjának meghatározása

Mivel térben lévő egyenesekkel dolgozunk, előfordulhat, hogy két egyenesnek nincs pontosan egy metszéspontja. Lehetnek kitérő, párhuzamos, és egybeeső egyenesek is.

Legyen a két egyenes egyenletének mátrix-reprezentációja:

$e: \begin{pmatrix} a_{e1} & b_{e1} \\ a_{e2} & b_{e2} \\ a_{e3} & b_{e3} \end{pmatrix} \quad f: \begin{pmatrix} a_{f1} & b_{f1} \\ a_{f2} & b_{f2} \\ a_{f3} & b_{f3} \end{pmatrix}$

Oldjuk meg az alábbi egyenletrendszert:

$$a_{e1}t_1 + b_{e1} = a_{f1}t_2 + b_{f1}$$
$$a_{e2}t_1 + b_{e2} = a_{f2}t_2 + b_{f2}$$

Ha az egyenletrendszer lineárisan függő, akkor nincs pontosan egy metszéspontjuk.

Ha a kapott t_1, t_2 értékek kielégítik a harmadik egyenletet: $a_{e3}t_1 + b_{e3} = a_{f3}t_2 + b_{f3}$, akkor létezik az egyenesnek egy darab közös pontja. Ha szakaszok metszéspontjáról beszélünk, akkor a $0 \le t_1 \le 1$ és a $0 \le t_2 \le 1$ egyenlőtlenségeket is vizsgálnunk kell. Visszahelyettesítve például az első egyenes paraméteres egyenletébe a t_1 számot, megkapjuk az $\mathbf{X}(x, y, z)$ metszéspontot.

1.3.5 Sík és egyenes döféspontja

Hasonlóan járunk el, mint az előbb:

$$s: \begin{pmatrix} a_{s1} & b_{s1} & c_{s1} \\ a_{s2} & b_{s2} & c_{s2} \\ a_{s3} & b_{s3} & c_{s3} \end{pmatrix} \quad e: \begin{pmatrix} a_{e1} & b_{e1} \\ a_{e2} & b_{e2} \\ a_{e3} & b_{e3} \end{pmatrix}$$

A megoldandó egyenletrendszer:

$$a_{s1}s_1 + b_{s1}s_2 + c_{s1} = a_{e1}t_1 + b_{e1}$$
$$a_{s2}s_1 + b_{s2}s_2 + c_{s2} = a_{e2}t_1 + b_{e2}$$
$$a_{s3}s_1 + b_{s3}s_2 + c_{s3} = a_{e3}t_1 + b_{e3}$$

A sík és egyenes metszéspont-kiszámításának algoritmusa a 6.5-ös mellékletben van leírva.

1.3.6 Síkok metszésvonala

Találnunk kell két olyan pontot, mely mind a két síkon rajta fekszik.

Síkok metszésvonalának meghatározásához egy 3 egyenletből álló 4 ismeretlenes egyenletet kell megoldanunk. Egy apró trükköt bevetve viszont leredukálhatjuk az ismeretlenek számát 3-ra.

$$s: \begin{pmatrix} a_{s1} & b_{s1} & c_{s1} \\ a_{s2} & b_{s2} & c_{s2} \\ a_{s3} & b_{s3} & c_{s3} \end{pmatrix} \quad t: \begin{pmatrix} a_{t1} & b_{t1} & c_{t1} \\ a_{t2} & b_{t2} & c_{t2} \\ a_{t3} & b_{t3} & c_{t3} \end{pmatrix}$$

$$a_{s1}s_1 + b_{s1}s_2 + c_{s1} = a_{t1}s_3 + b_{t1}s_4 + c_{t1}$$
$$a_{s2}s_1 + b_{s2}s_2 + c_{s2} = a_{t2}s_3 + b_{t2}s_4 + c_{t2}$$
$$a_{s3}s_1 + b_{s3}s_2 + c_{s3} = a_{t3}s_3 + b_{t3}s_4 + c_{t3}$$

Írjunk s_3 helyére 0-t, ha így nem kapunk eredményt, akkor s_4 helyére 0-t. Ha egyik helyettesítéssel sem kaptunk egy konkrét megoldást, akkor a két sík vagy párhuzamos, vagy egybeesik. Ha van pontosan egy megoldás, akkor már meg is találtuk az egyik közös pontot. Járjunk el még egyszer ugyanígy, most viszont 1-et helyettesítsünk be. Ekkor megkaphatjuk a második pontot, és a két pont által fel tudjuk írni a metszésvonaluk egyenletét.

Az algoritmus megvalósításáról a 6.6-os mellékletben olvashatunk.

2. Pont-transzformációk

2.1. Koordinátarendszerek

A testek a definiálástól a síkban való megjelenítésig három különböző koordinátarendszerben léteznek. A definiálásukat a világ koordinátarendszerében végezzük, majd a pont-transzformációk elvégzése után kerülnek át a kamera koordináta rendszerébe. A vetítés során pedig „elhagyva” Z koordinátájukat a vetítési sík koordinátarendszerébe kerülnek.

A térben a Descartes-koordinátarendszerek egy osztályozása szerint a körbejárásuk alapján különböztetjük meg őket: bal- és jobbsodrású koordinátarendszerek (7. ábra):

7. ábra

Alapkonvenció szerint a világ rendszere, melyben az objektumokat definiáljuk, jobbsodrású, míg a kamera koordinátarendszere – a világ rendszerével ellentétesen – balsodrású.

2.2. Homogén koordináták

A transzformációkat mátrixokkal reprezentáljuk. Az egységes mátrix-reprezentáció érdekében bevezetjük a homogén koordinátákat.

Eddig a térbeli pontokat háromdimenziós vektorok segítségével adtuk meg. A homogén koordinátarendszerbe való áttéréskor a pontot nem egy rendezett számhármas, hanem rendezett számnégyes határoz meg. $P = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow P = \begin{pmatrix} w \cdot x \\ w \cdot z \\ w \cdot z \\ w \end{pmatrix}$

Tehát, ha adott egy $\mathbf{P}(x, \quad y, \quad z)$ Descartes koordinátarendszerbeli pont, akkor a homogén koordinátarendszerben felírt $\mathbf{P'}(x, \quad y, \quad z, \quad 1)$ pont vele megegyező.

Arányosság: az $(x, \quad y, \quad z, \quad w)$ ugyanazt a pontot határozza meg, mint a $(\lambda x, \quad \lambda y, \quad \lambda z, \quad \lambda w)$.

A $(0, \quad 0, \quad 0, \quad 0)$ pont nem létezik.[1]

Visszatérés homogén koordinátarendszerből Descartes koordinátarendszerbe:

$$P = \begin{pmatrix} x_H \\ z_H \\ z_H \\ w_H \end{pmatrix} \rightarrow P = \begin{pmatrix} x_D \\ y_D \\ z_D \end{pmatrix}$$

Ha $w_H \neq 0$, akkor

$x_D = x_H / w_H$

$y_D = y_H / w_H$

$z_D = z_H / w_H$

Ha $w = 0$, akkor az a projektív sík végtelen távoli pontja.

Pont-transzformációk megvalósítása mátrix-szorzással:

[1] Dr. Kovács Emőd, Hernyák Zoltán, Radványi Tibor, Király Roland: A C# programozási nyelv a felsőoktatásban

P’ = **M** · P, ahol a P a transzformálni kíván pont, **M** a transzformáció mátrixa. **M** [4x4]-es, determinánsa nem nulla.

A itt felsorolt kölcsönösen egyértelmű transzformációkat valósítottam meg a programomban.

2.3. Eltolás

Adott a világ koordinátarendszerében a $P(x, \quad y, \quad z)$ pont. Toljuk el az x tengely mentén t_x, az y tengely mentén t_y, a z tengely mentén t_z -vel.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x + t_x \\ y + t_y \\ z + t_z \end{pmatrix}$$

Ugyanezt megtehetjük az eltolás mátrix-reprezentációjával is, ha felírjuk a P pont homogén koordinátáit, majd a felírt vektort balról szorozzuk az eltolás mátrixával:

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{P'} = \mathbf{T} \cdot \text{P}$$

2.4. Skálázás

Skálázás alatt a P pont koordinátáinak skalárokkal való szorzását értjük. Ha mind a három skalár megegyezik, akkor arányos kicsinyítésről vagy nagyításról van szó.

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \mathbf{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{P'} = \mathbf{S} \cdot \text{P}$$

2.5. Forgatás

A forgatás alatt most a tengelyek mentén adott szöggel való elforgatást értek. Mind a három tengely körüli forgatásnak külön mátrixa van.

Forgatás az x tengely körül θ-val:

$$\mathbf{R_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Forgatás az y tengely körül θ-val:

$$\mathbf{R_z} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Forgatás az z tengely körül θ-val:

$$\mathbf{R_z} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.6. Transzformációk egymás utáni végrehajtása

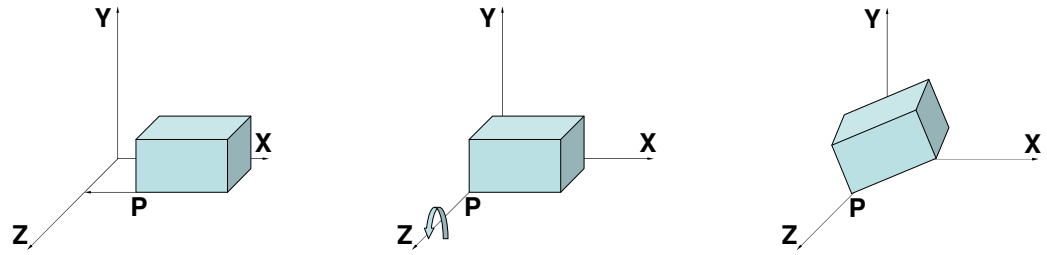
A transzformációk egymás utáni végrehajtásához szorozzuk össze az alap transzformációs mátrixokat jobbról balra, majd a szorzattal szorozzuk meg a transzformálni kívánt pont homogén koordinátájából álló vektort. Tehát a transzformációk sorrendje szerint az összeszorzandó mátrixok fordított sorrendben vannak felírva.

$\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{M_3}$, mátrix-transzformációk egymás utáni végrehajtása: $\mathbf{M} = \mathbf{M_3} \cdot \mathbf{M_2} \cdot \mathbf{M_1}$.

$\mathbf{M_1} \cdot \mathbf{M_2} \neq \mathbf{M_2} \cdot \mathbf{M_1}$

Vegyünk például egy **P**(20, 0, 10) pontot, toljuk el az x tengely mentén -20 egységgel, majd forgassuk a z tengely körül 20°-al (8. ábra).

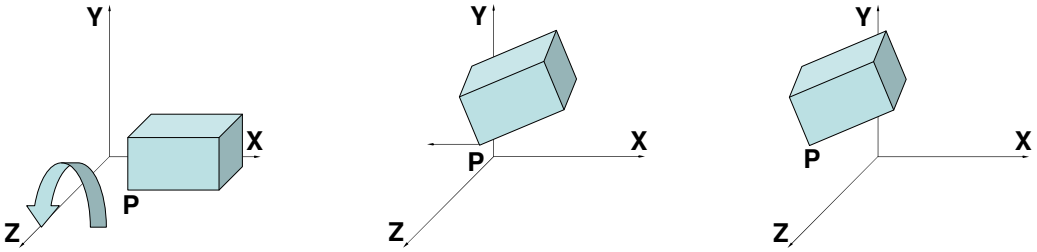
$\mathbf{M} = \mathbf{R} \cdot \mathbf{T}$; $P' = \mathbf{M} \cdot P$



8. ábra

Ezután cseréljük fel a transzformációkat. Ugyan ezt a pontot először forgassuk el a z tengely mentén 20°-al, majd toljuk el az x tengely mentén -20 egységgel (9. ábra).

$\mathbf{M} = \mathbf{T} \cdot \mathbf{R}$; $P' = \mathbf{M} \cdot P$



9. ábra

3. Testek reprezentálása

A programban a testeket valamilyen adatszerkezettel tárolnunk kell. Több reprezentálási mód is létezik. Mindegyiknek van előnyös és hátrányos oldala, attól függően válasszuk ki a megfelelőt, hogy mi a programunk célja. Ebben a fejezetben négy főbb reprezentálási módot mutatok be: B-rep, CSG, NURBS.

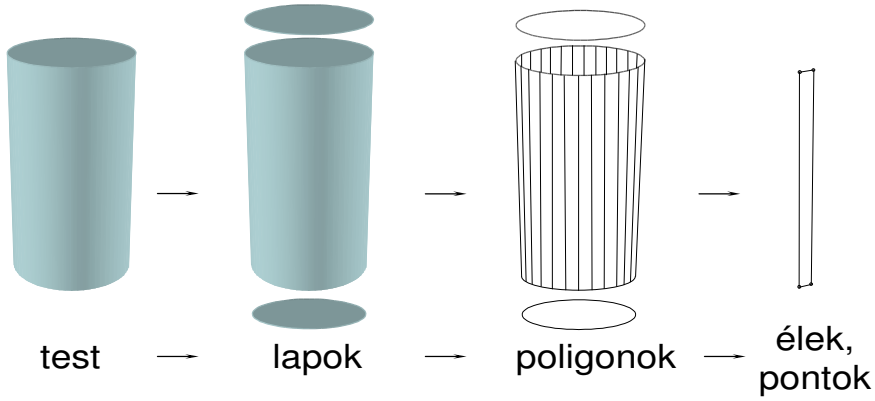
3.1. B-rep (Boundary-representation)

A térben lévő testek modellezéséhez a B-rep technika a legegyszerűbb, a mellékelt programban is ez az eljárás lett megvalósítva. A testek elemeit hierarchikusan tároljuk. Hátránya, hogy minél finomabb, és bonyolultabb testet akarunk tárolni és megjeleníteni, annál nagyobb adatstruktúrát kell használnunk, illetve ezzel számolnunk, ami tár és időigényes.

Meg kell adnunk:

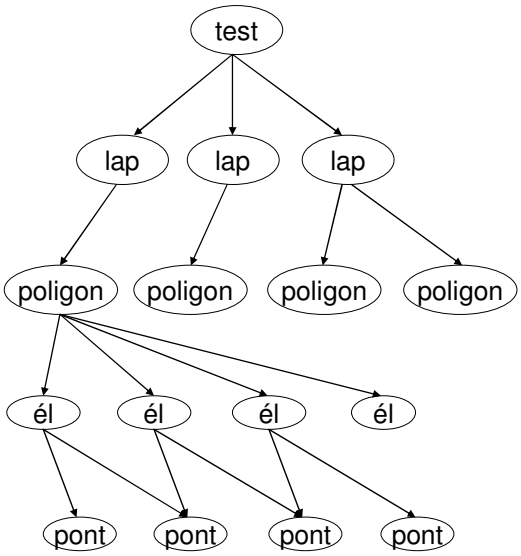
- a testet határoló pontok koordinátáit.

- az éleket,
- a lapokat határoló poligonokat,
- és a lap egyéb vizuális tulajdonságait: a színt és a felületi jellemzőket



10. ábra

A következő struktúrában tudjuk tárolni a test elemeit:

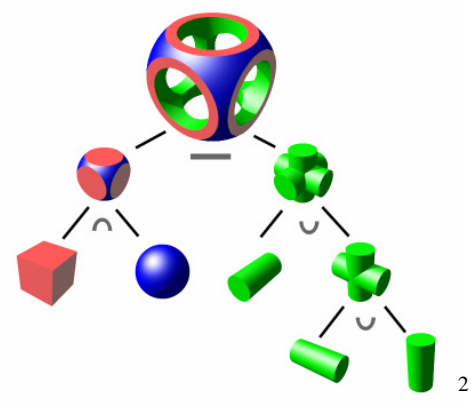


11. ábra

A programomban XML adatfájlban tárolom a testeket, amellyel átlátható struktúrát lehet kialakítani.

3.2. CSG (Constructive Solid Geometry)

Ez az az eljárás, amit a tervezőprogramok leggyakrabban használnak. A CSG módszer szintén hierarchikusan tárolja a testeket, viszont a B-reppel ellentétben nem pontokból, élekből, poligonokból, lapokból építi fel a testet, hanem primitív testekből (12. ábra). A primitív testeket regularizált halmazműveletekkel kapcsolják össze. A regularizáció azért szükséges, hogy ne keletkezzenek kiterjedés nélküli lapok.



[2]

12. **ábra**

3.3. NURBS

A mai nagykapacitású számítógépek megjelenése előtt a fenti eljárások nem biztosítottak arra lehetőséget, hogy jó minőségű képet állítsunk elő. A testek szögletesek voltak, és ha közelebb vittük a kamerát, akkor ezek a pontatlanságok még jobban kiéleződtek. Az eljárás matematikai függvényeken, görbéken alapul. A modellezni kívánt testnek meghatározzuk a főbb kontrollpontjait, és ezekre a pontokra görbéket illesztünk. Ha megfelelő mennyiségű görbével rendelkezünk, akkor ezzel a módszerrel jó minőségű felületeket tudunk előállítani. Az animációk készítésekor, ha a testnek a kontrolpontjait mozgatjuk, a felületet alkotó görbék a ponttal együtt mozognak. Az igen elterjedt Maya grafikus program poliéder-modellel dolgozik, amiben a B-rep és a CSG mellett a testeket NURBS felületként is elő tudjuk állítani.

[2] Kép forrása: Wikipedia; http://en.wikipedia.org/wiki/Image:Csg_tree.png

4. Testek síkra vetítése

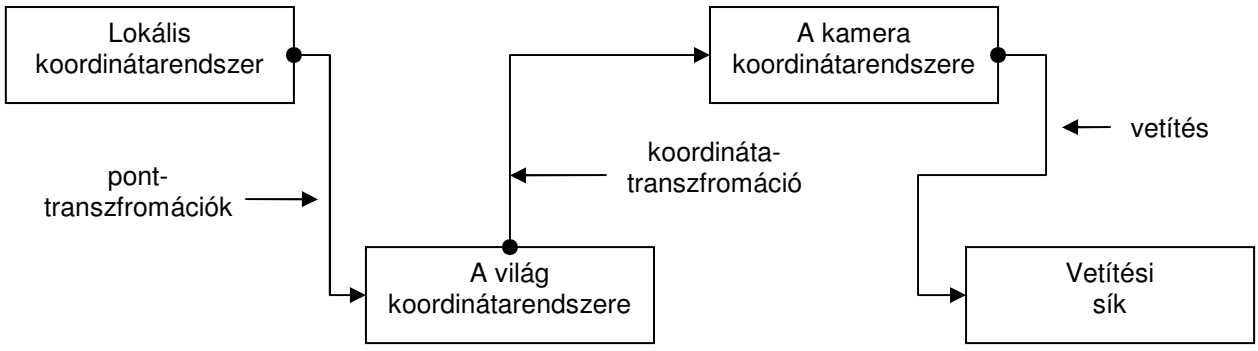
A test definiálásától a képi megjelenítésig hosszú út vezet.

- Az objektumok különböző transzformációkon mennek keresztül.
- Meg kell határozni minden lap felületének megjelenítési módját.
- Melyek azok az objektumok, amelyeket meg kell jeleníteni a képernyőn.
- Mely objektum van előtérben, háttérben.
- Azt is meg kell határoznunk, hogy az egyes objektumok hol és hogyan helyezkednek el a vetítési síkon.

Ennek a folyamatnak a bemutatására a grafikus csővezetéket használom fel…

4.1. Grafikus csővezeték

A grafikus csővezeték segítségével lépésről lépesre nyomon követhetjük, hogy mik azok a műveletek, amiket végre kell hajtanunk annak érdekében, hogy az objektumaink megjelenjenek a képernyőn.



4.1.1 Helyi koordinátarendszer

A helyi vagy lokális koordinátarendszerben definiáljuk a testet. A test reprezentálására több módszer létezik, néhányat röviden már ismertettem az előző fejezetben. Itt már ismertek a testet határoló pontok, élek, poligonok.

4.1.2 Pont-transzformációk

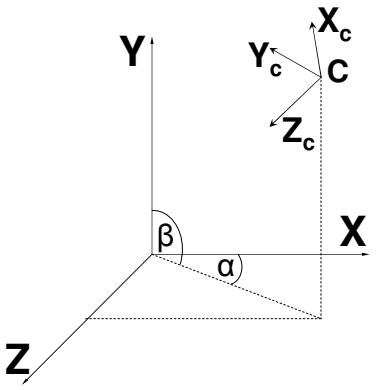
A lokális koordinátarendszerben definiált test pontjain elvégzett különböző transzformációk elvégzése után a test pontjainak koordinátái ismertté válnak a világ koordináta rendszerében. A mellékelt programban a 2. fejezetben említett transzformációkat valósítom meg.

4.1.3 A világ koordinátarendszere

A testet határoló pontok meghatározása után definiáljunk egy kamerát, fényforrást (esetleg többet is), határozzuk meg a lapok színét, árnyalását.

4.1.3.1 Kamera definiálása

A kamera elhelyezéséhez használjunk polár koordinátákat. Ha Descartes koordinátákkal határoznánk meg a kamera pozícióját, akkor könnyedén tudjuk mozgatni a világ koordinátarendszerében x, y, z irányba egyszerű eltolással, de a gyakorlatban általában az origó vagy objektum körüli körbeforgatást alkalmazzuk (13. ábra).



13. ábra

Ez által a kamera helyzetét nem egy $(x,\ y,\ z)$ számhármas határozza meg, hanem gömbi koordináták, két szög és az origótól való távolság: $(\alpha,\ \beta,\ R)$. Tehát az α és a β módosításával az origó körüli R sugarú körben tudjuk mozgatni a kamerát, az R módosításával pedig az origótól való távolságot tudjuk állítani.

Polár koordinátarendszerből Descarets koordinátarendszerbe való áttérés:

$X = R \cdot \cos(\beta) \cdot \sin(\alpha)$

$Y = R \cdot \sin(\beta)$

$$Z = R \cdot \cos(\beta) \cdot \cos(\alpha)$$

A kamera koordinátarendszerét állítsuk balsodrásúra:

$X : (1, 0, 0);\ Y : (0, 1, 0);\ Z : (0, 0, -1).$

Ha a kamera koordinátarendszerét is polár koordinátákkal adjuk meg, a kamera forgatásánál könnyebb dolgunk lesz beállítani a látóirányt, hogy ugyanarra a pontra nézzen:

Legyen $R_{tengely} = 1;\ \alpha_{tengely} = \alpha_{kamera};\ \beta_{tengely} = \beta_{kamera}$, ekkor

$$Y_{tengely} = PolarToDescartes(\alpha_{tengely}, \beta_{tengely} + 90°, 1)$$

$$Z_{tengely} = PolarToDescartes(\alpha_{tengely}, \beta_{tengely} + 180°, 1)$$

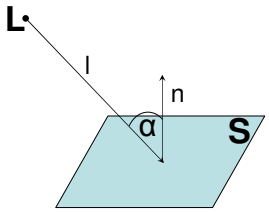
$$X_{tengely} = Z_{tengely} \times Y_{tengely}$$

4.1.3.2 Világítás definiálása

Határozzuk meg a fényforrás helyét is. Ha nem akarjuk gömb felszínén mozgatni a fényforrásunkat, akkor elegendő, ha Descartes koordinátákkal definiáljuk a helyzetét, ha a kamerához hasonlóan szeretnénk mozgatni, akkor az előző módszerrel használjunk polár koordinátákat .

4.1.3.3 Árnyalás

Ha adtunk meg fényforrást, lehetőségünk van a megjelenítendő lapok árnyalására. Az egyszerűség kedvéért tételezzük fel, hogy a fényforrásunk nem irányított fénysugárral rendelkezik, hanem nap-szerű, tehát minden irányba ugyanolyan intenzitással világít (14. ábra).



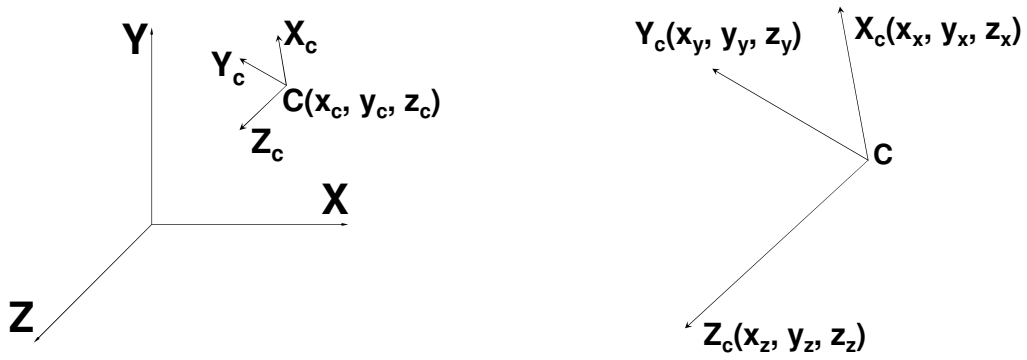
14. ábra

Konstans árnyalás: ha a fényforrást elég távolinak tekintjük, a lap teljes felületére közel ugyanolyan szögben érkeznek a fénysugarak. Ekkor elegendő kiválasztani a lap egy pontját, például a súlypontját, és a fényforrásból arra a pontra érkező fénysugarat.
Lambert törvénye szerint a lap intenzitása fordítottan arányos a beeső fény szögének koszinuszával.
Legyen ez a vektor L. Számítsuk ki a lap síkjának normálvektorát, N-t (ha az eddig nem lett volna tárolva), és határozzuk meg az L és N által bezárt szög (α) koszinuszát. Ezt a két vektor skaláris szorzatából kapjuk.
Akkor van a lap legjobban megvilágítva, ha $\alpha = 90°$, vagyis $\cos(\alpha) = 0$. Ekkor vegyük a lap színének 100%-át. Legkevésbé megvilágítva $\alpha = 0°$, azaz $\cos(\alpha) = 1$ esetén van.
Ha RGB színrendszerrel dolgozunk, határozzuk meg az R, G és B komponensek értékét $\cos(\alpha)$-val fordítottan arányosan.

Ha a fényforrás közeli, a lap különböző pontjára máshogy esik a fény. Minden pontra kiszámolni a beeső fény sugarának szögét - illetve annak koszinuszát – igen nagy erőforrás igényű művelet lenne.
Helyette alkalmazhatjuk a Gouraud, vagy Phong módszerét. Számoljuk ki minden csúcs normálvektorát úgy, hogy a csúcsokba összefutó lapok normálvektorát átlagoljuk. Ezek alapján meghatározhatjuk a csúcsokban lévő intenzitást. Az élek pontjainak, illetve a belső pontok intenzitását interpolációval kapjuk. A két eljárás között a különbség az interpoláció módjában van. Bővebben olvashatunk erről a témáról Dr. Szirmay – Kalos László: Számítógépes grafika című könyvében.

4.1.4 Koordináta-transzformáció

Miután meghatároztuk a világ koordinátarendszerében a testek tulajdonságait, át kell vinnünk azt a kamera koordinátarendszerébe. Ehhez szükségünk van a kamera helyzetére, illetve tengelyeinek koordinátáira. (15. ábra)



15. ábra

A kamera a $C(X_C, Y_C, Z_C)$ pontban helyezkedik el. A Z tengely ($Z(X_Z, Y_Z, Z_Z)$) fele néz, a kamera szemszögéből nézve pedig az Y tengely ($Y(X_y, Y_y, Z_y)$) felfele mutat. Ha balsodrású a koordinátarendszere, akkor az X ($X(X_x, Y_x, Z_x)$) tengely pozitív része balkéz fele van.

Első lépésben eltoljuk a transzformálni kívánt pontot a kamera origójába (**T** mátrix), majd elvégezzük a koordináta transzformációt (**R** mátrix).

M = RT

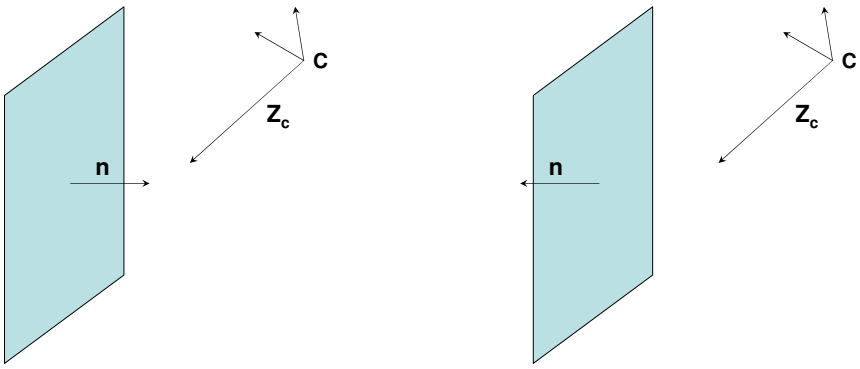
$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & -X_C \\ 0 & 1 & 0 & -Y_C \\ 0 & 0 & 1 & -Z_C \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{R} = \begin{pmatrix} X_X & X_Y & X_Z & 0 \\ Y_X & Y_Y & Y_Z & 0 \\ Z_X & Z_Y & Z_Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \mathbf{M} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}$$

4.1.5 Kamera koordinátarendszere

Meghatároztuk a testet határoló pontok nézeti rendszerben lévő koordinátáit. Ebben a szakaszban kell eldöntenünk, hogy pontosan melyik lapok látszanak majd a képernyőn, és azoknak mely részei.

4.1.5.1 Hátsó lapok eltávolítása (Back face culling)

A lapok normálvektorát célszerű már az objektum definiálásakor meghatározni, és azt a testtel együtt transzformálni a világ koordinátarendszerében, mert ezt az információt a megjelenítésig többször használjuk. Azonban míg a lapok pontjait átvittük a kamera koordináta-rendszerébe, a normálvektorát nem szükséges.



16. ábra

Számoljuk ki a kamera Z tengelye és a lap normálvektora (**n**) által bezárt szög koszinuszát a skaláris szorzatot használva.
(16. ábra).

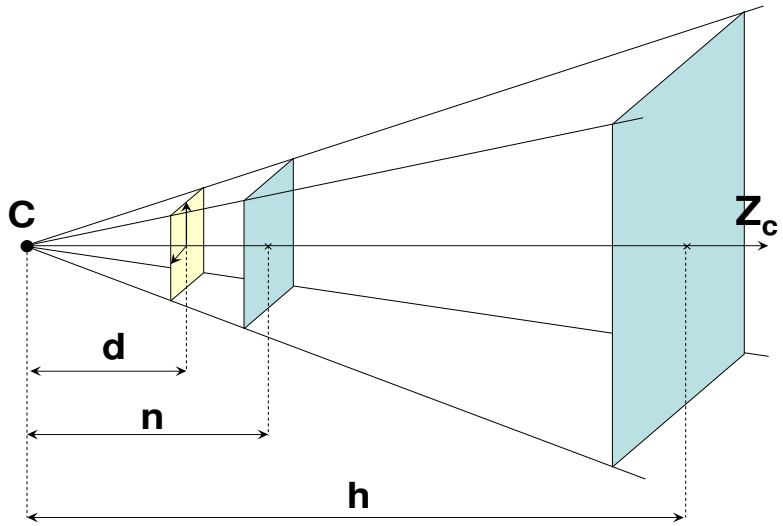
$$\begin{cases} \cos\alpha > 0 \Rightarrow \text{a lap hátsó lap} \\ \cos\alpha <= 0 \Rightarrow \text{a lap látható lap} \end{cases}$$

A hátsó lapokat távolítsuk el, hogy ezekkel már ne kelljen tovább számolni, és ezzel is processzor-időt takaríthatunk meg.

4.1.5.2 Vetítési gúla

A láthatósági gúla határozza meg, hogy a testek mely rész látható a képernyőn. A gúla alsó-, felső-, jobb- és baloldala, valamint a közelebbi és távolabbi vágósík által határolt csonka gúlába eső részek láthatóak, a többi a kamera szemszögéből láthatatlan objektumok.
A kamera tulajdonságai közé vegyük fel a következő értékeket (17. ábra):

- d : a kamera és a vetítési sík távolsága
- m_1 : a képernyő szélessége
- m_2 : a képernyő magassága
- n : a közelebbi vágósík távolsága a kamerától
- h : a távolabbi vágósík távolsága a kamerától



17. ábra

Ezekből az adatokból már meghatározható mind a hat vágósík paraméteres egyenlete, mert mindegyik oldalnak ismerjük legalább három pontját.

A képernyőt határoló pontok koordinátái:

- bal felső: $(-m_1/2,\quad m_2/2,\quad d)$
- bal alsó: $(-m_1/2,\quad -m_2/2,\quad d)$
- jobb felső: $(m_1/2,\quad m_2/2,\quad d)$
- bal alsó: $(m_1/2,\quad -m_2/2,\quad d)$

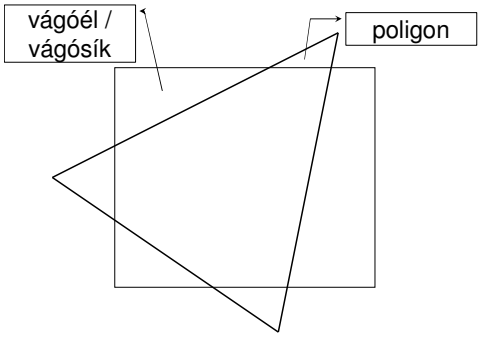
A láthatósági csonka gúla közelebbi lapjának koordinátáit megkaphatjuk, ha a képernyő koordinátáit megszorozzuk az (n/d) skalárral.

A távolabbi lap koordinátáinak kiszámolásánál (n/h) –val szorzunk.

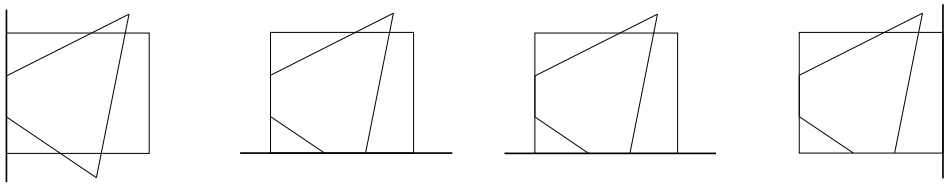
Írjuk fel a vágósíkok egyenleteit, és a következő alfejezetben tárgyalt módon vágjuk le a testek nem látható részeit.

4.1.5.3 Vágás

Ezzel az eljárással eltávolíthatjuk az objektum azon részeit, melyek nem láthatóak. Vegyük sorba a 18. és a 19. ábrán látható módon a vágóéleket / vágósíkokat, és sorba mindegyikkel hajtsuk végre a vágási műveletet.

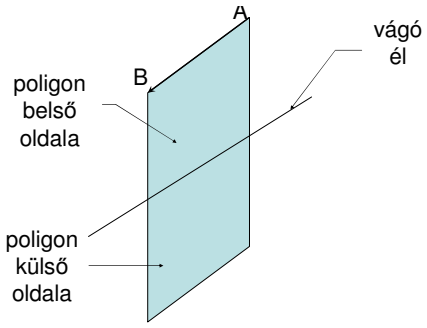


18. ábra

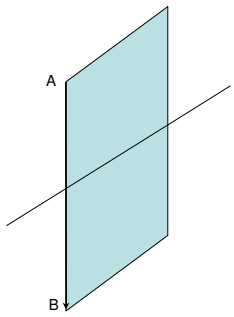


19. ábra

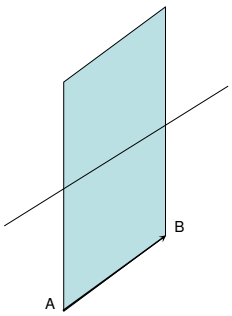
A vágás során körbemegyünk a poligon élein, és minden élt vágjuk az éppen aktuális vágóelemmel. Készítsünk egy új poligont, és fűzzük hozzá az aktuális művelet végeredményét. Négy esetet különböztetünk meg:



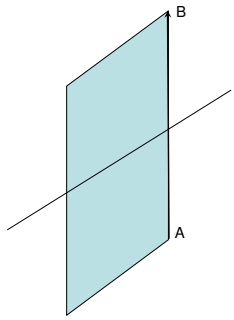
20. ábra



21. ábra



22. ábra



23. ábra

- (a) az él mindkét pontja belül helyezkedik el, ekkor fűzzük hozzá az új poligonhoz az él végpontját (20. ábra).
- (b) az él első pontja belül, a második pont kívül helyezkedik el. Új pontként vegyük fel az él és a vágóelem metszéspontját (21. ábra).
- (c) az él mindkét pontja kívül helyezkedik el, nem veszünk fel új pontot a poligonba (22. ábra).
- (d) az él első pontja kívül, a második pontja belül van. Visszaadjuk a metszéspontot, majd az él végpontját (23. ábra).

A metszéspontokat az 1.3.4. fejezetben leírt módon meg tudjuk határozni.

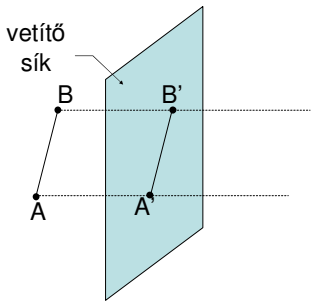
4.1.6 Vetítés

Eljutottunk addig a pontig, hogy tudjuk, milyen objektumokat milyen színnel, színekkel szeretnénk megjeleníteni a képernyőn. Ahhoz, hogy három dimenzióból átvigyük az objektumokat két dimenzióba, le kell vetítenünk őket egy síkra, a képernyő síkjára. Többféle vetítési módszer létezik, melyekkel más és más képet kapunk. Vetítés előtt a homogén koordinátarendszerből térjünk vissza Descartes koordinátarendszerbe. A leggyakrabban használtak a párhuzamos vetítés, az axonometria és a centrális projekció. A mellékelt programban a centrális projekciót alkalmazom. A következő alfejezetekben ezek közül kettőt részletesebben bemutatok.

4.1.6.1 Párhuzamos vetítés

A párhuzamos vetítésnél a nevéből adódóan a háromdimenziós pontokat párhuzamosan vetítjük rá az [XY] síkra (24. ábra). (A kamera a Z tengely irányába néz).
A gyakorlatban ez annyit jelent, hogy a pontok elhagyják a Z koordinátájukat.

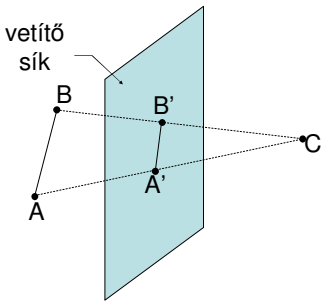
$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow P' = \begin{pmatrix} x \\ y \end{pmatrix}$$



24. ábra

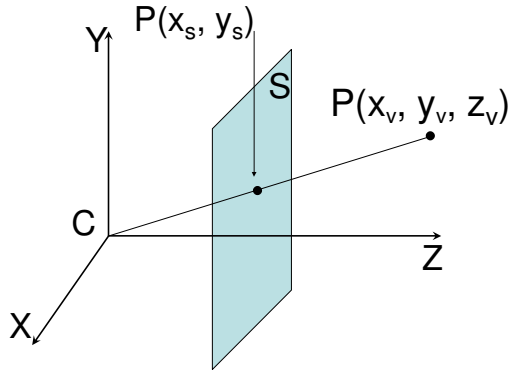
4.1.6.2 Centrális projekció

Ha centrális projekciót alkalmazunk, sokkal valósághűbb képet kapunk, mint az előző eljárásnál. Ha elvégeztük a vágást a vetítőgúlával, akkor a kamera a vetítési sík egyik oldalán helyezkedik el, a megjelenítendő test a sík másik oldalán. (Akkor is kaphatunk képet az objektumról, ha azok a sík azonos oldalán vannak.) A kamerából egy vetítendő pontra sugarat indítunk, és ahol ez a sugár elmetszi a képernyő síkját, ott lesz a pont képe (25. ábra).



25. ábra

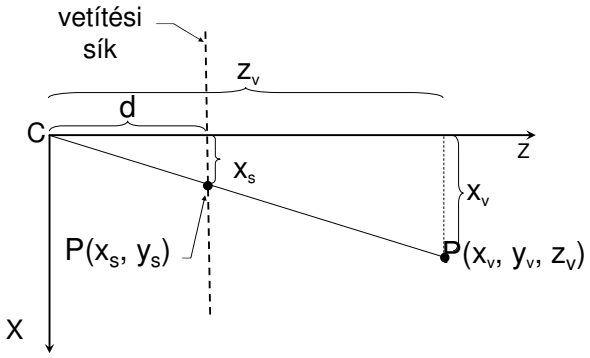
A levetített pont koordinátáinak kiszámítása:



26. ábra

A 26. ábrának megfelelően legyen a megjelenítendő pont $P(x_v, y_v, z_v)$. A vetítősík a Z tengelyen helyezkedjen el, éa a kamerától való távolsága legyen d. A pont vetített képe pedig $P'(x_s, y_s)$. Ekkor:

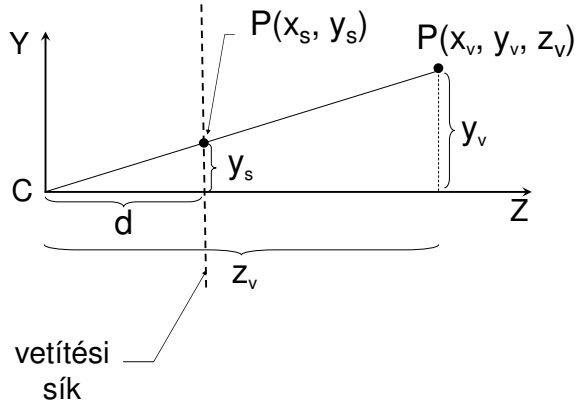
Az Y tengely irányából nézve:



27. ábra

$\frac{x_s}{d} = \frac{x_v}{z_v}$.

Az X tengely irányából nézve:



28. ábra

$$\frac{y_s}{d} = \frac{y_v}{z_v}$$

$$\Rightarrow \begin{matrix} x_v = d\frac{x_w}{z_w} \\ y_v = d\frac{y_w}{z_w} \end{matrix}$$

4.2. Vetítési sík

Adottak az objektumaink kétdimenziós koordinátái. Viszont a megjelenítendő lapok, élek egymást takarhatják. Kérdés, hogy melyik van előrébb, a képernyő egyes pontjaiban melyik objektum színe fog megjelenni. A lapok sorba rendezése igen nagy erőforrást igényel. A mai forgalomban lévő videokártyákat már felkészítették ennek a problémának a hardveres megoldására. A grafikus programok ezt a lehetőséget kihasználva nem is terhelik a processzort feleslegesen ezen művelet elvégzésével.

4.2.1 Objektumok sorba rendezése

Tételezzük fel, hogy a felbontásunk 1024x768-as, melyen meg szeretnénk jeleníteni a levetített objektumainkat. Minden egyes pont színének kiszámításához szükségünk van arra az információra, hogy az adott képpontban melyik objektum látszik. Ennek az eldöntésére a két legismertebb algoritmus az objektumok sorba rendezésére a Z-buffer és a Ray-tracing.

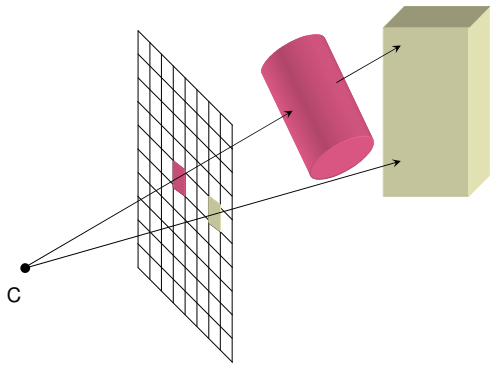
4.2.1.1 Z-buffer

Készítsünk egy 1024x768 méretű tömböt, melynek elemeiben tárolni tudunk objektumra mutató referenciát, és egy lebegőpontos számot. Vegyük az első objektumunkat, és ne egyből a képernyőre rajzoljuk, hanem a kétdimenziós tömbünkbe úgy, hogy a tömb adott elemének objektumreferenciáját állítsuk az aktuális objektumra, a lebegőpontos értékbe pedig töltsük be az éppen kirajzolt pont Z koordinátáját. Menjünk sorba a az objektumokon, de a bufferbe töltés előtt vizsgáljuk meg, hogy az adott tömbelembe nem-e „rajzoltunk” már. Ha üres az elem, akkor töltsük fel az eddigi módszerrel, ha nem, hasonlítsuk össze a már tárolt Z értéket az aktuális pont értékével. Csak akkor írjuk felül a tömb elemét, ha az új pontnak a mélységi koordinátája kisebb, mint a már eddigi. Az eljárás végére feltöltünk egy a képernyő felbontásával megegyező méretű tömböt azzal az információval, hogy mely ponton melyik objektum látszik. Ezután csak az objektumok színét kell visszakeresni, és a képernyő pontjait feltölteni.

4.2.1.2 Sugárkövetés (Ray-tracing)

Vegyük sorba a képernyő pontjait, és a kamerából minden képpont irányába indítsunk el egy sugarat. Amelyik objektumot először elmetszi az aktuális sugár, azt jelenítjük

meg az adott képpontban. Ha a sugár nem találkozik objektummal, akkor az a képpont üres marad. Ez az eljárás is elég időigényes, mert egy 1024x768-as felbontású képhez közel 800 ezer sugarat kell kilőnünk, és metszéspontokat számolnunk.



29. ábra

4.2.2 Window to viewport

A vetítési síkon a képernyő mérete nem feltétlen egyezik meg a megjelenítendő kép felbontásával. A síkon lévő kép szélességével és magasságával meghatározott ablakban lévő képpontok képernyőnk felbontása szerinti koordinátáját ezzel az eljárással tudjuk meghatározni.

Input:

- A képen lévő pont koordinátái: $P(x_p, y_p)$
- A megjeleníteni kívánt kép ablaka: Window.Left, Window.Top, Window.Width, Window.Height
- A képernyőnk ablaka: View.Left, View.Top, View.Width, View.Height

Output:

- A képernyőn lévő pont koordinátái: $Q(x_q, y_q)$

Az eljárás: A két ablak méretének arányaival számolva:

$$x_q = (x_p - Window.Left) \cdot (View.Width / Window.Width) + View.Left$$

$$y_q = (-y_p - Window.Top) \cdot (View.Height / Window.Height) + View.Top$$

Megjegyzés: az y_p-t azért negáltuk, mert a képernyő Y tengelye lefelé néz, vagyis az $y = 0$ pontok a képernyő tetején helyezkednek el, míg az $y = \max_y$-1 a képernyő alján.

5. A program leírása

A mellékelt programot Visual Studio 2005-ben C# nyelven fejlesztettem. A forrását bárki szabadon felhasználhatja.

5.1. A program felépítése

A program felépítése megközelítőleg követi a háromrétegű alkalmazások struktúráját. Három fő és egy mellék projektből áll:

- adatelérési,
- logikai,
- megjelenítési és
- egy mindegyik által látott réteg (Globals).

5.1.1 Globals

Ebbbe a projektbe helyeztem el azokat az osztályokat, melyeket legalább két fő-projekt használ. Ide tartozik

- háromdimenziós testek reprezentálását leírása,
- kamerát és a fényforrást definiáló osztályok,
- vektorok és mátrixok struktúrákat, műveleteket és tulajdonságokat leíró osztály,
- transzformációk definiálása,
- rajzelemek osztályai.

5.1.2 Adatelérési réteg (Data Access Layer - DAL)

A program a térbeli testeket B-rep technikával reprezentálja. Az objektumok B-rep struktúráját XML leíró fájlokban tárolom, és a program innen olvassa ki az adatokat. Egyetlen statikus class-ból áll a DAL, az XmlIO.cs-ből.

5.1.3 Logikai réteg (Business Logic Layer – BLL)

Itt található annak a megvalósítása, amiket az előző fejezetekben tárgyaltam. Ez a réteg valósítja meg a:

- transzformációs mátrixok elkészítését,
- pont-transzformációkat,
- a vetítési gúla síkjaival való vágást,
- a centrális projekciót,
- egy speciális Ray-tracing-et,
- Z-buffer-t,
- a window to viewport-ot.

5.1.4 Megjelenítési réteg (Presentation Layer – CentProj)

A felhasználó a megjelenítési réteg által nyújtotta lehetőségeket tudja használni. Ez az a réteg, ami kommunikál a Userrel. Itt határoztam meg a program felületét alkotó főformot, illetve egyéb message és input formokat. A program használatára a következő fejezetben térek ki.

5.2. A program használata

Töltsük be a megjeleníteni kívánt térbeli testet/testeket a Fájl/Betöltés menüpontban. Ha egymás után több ugyanolyan nevű testet töltöttünk be, például több „kocka”-t, akkor a második a objektum neve „kocka(2)”, a harmadiké „kocka(3)” lesz, és így tovább... Ha valamelyik objektumra már nincs szükségünk, a File/Eldobás/*eldobandó test neve* menüpontban eltávolíthatjuk.

A főablak felosztása:

- a bal felső részben transzformációk generálhatók testekhez
- a bal alsó részen a nézeti tulajdonságokat állíthatjuk
- a középső területen a főkamera szemszögéből látjuk a levetített objektumokat
- a jobboldali részen pedig egy külső kamerával figyelhetjük a testek és a kamera egymáshoz viszonyított helyzetét.

Miután betöltöttünk legalább egy objektumot, adhatunk hozzá különböző pont-transzformációkat:

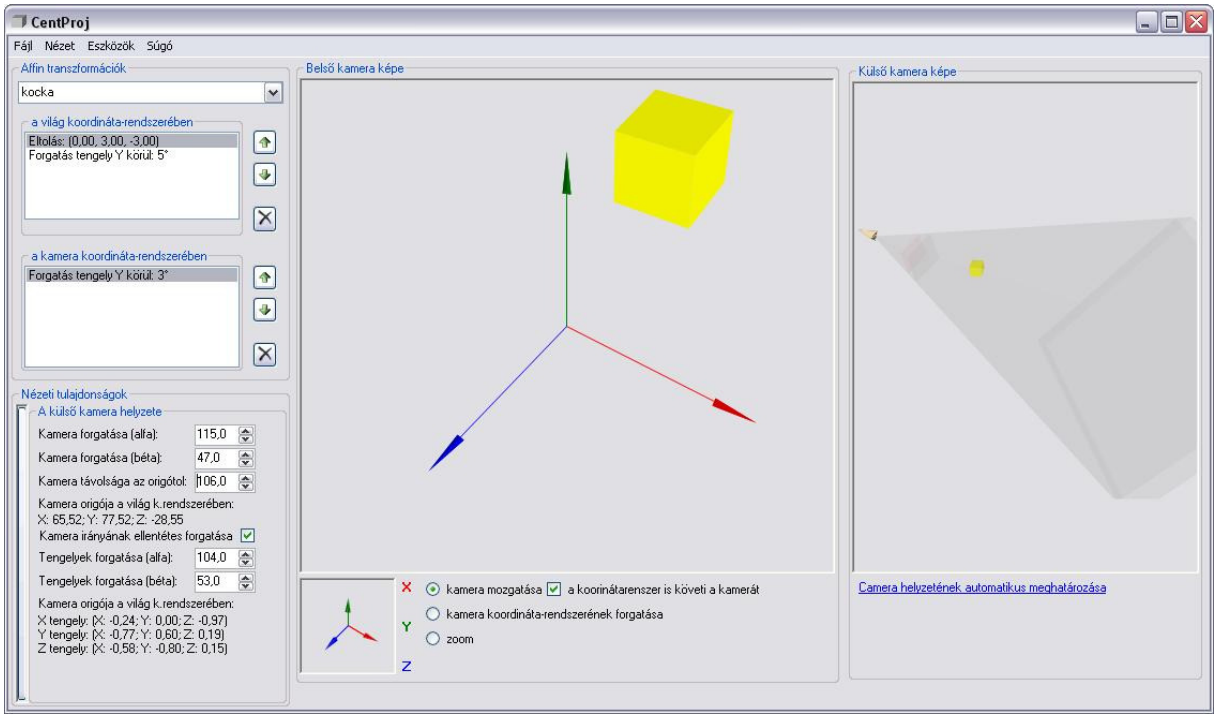
- forgatás három a koordináta-tengely körül,
- eltolás tengelyenként,
- skálázás.

Megfigyelhetjük, hogy ha felcseréljük a transzformációkat, miben változik meg a vetített kép.
A pont - transzformációkat két nagy csoportra osztottam:

- a koordináta-transzformáció előtti, a világ koordináta-rendszerében való transzformációk,
- a koordináta-transzformáció utáni, vagyis a kamera koordináta-rendszerében való transzformációk.

A nézeti tulajdonságoknál megadhatjuk:

- a kameránkként a pocíciót és a kamera-koordinátarendszereket,
- a vetítőgúlák adatait,
- a testek megjelenítési formáját (pontok, élek, lapok, hátsó lapok, lapok átlátszósága),
- valamint beállíthatjuk, hogy az egyes kameranézetekben milyen segédelemek jelenjenek meg (világ és kamera rendszerének koordinátatengelyei, kamera, mint test, vetítőgúla),
- árnyalás ki/be kapcsolása.



30. ábra (pillanatkép a programról)

6. Mellékletek

6.1. Mátrix-műveletek

Objects.Matrix.cs

```
        /// <summary>
        /// mátrixok összeadása
        /// </summary>
        /// <param name="mtxParam">a hozzá adandó mátrx</param>
        /// <returns></returns>
        private Matrix Add(Matrix mtxParam)
        {
            if (this.ColCount != mtxParam.ColCount || this.RowCount !=
mtxParam.RowCount)
            {
                throw new Exception("Összeadandó mátrixok nem azonos
méretűek!");
            }
            //az eredmény-mátrix
            Matrix mtxResult = this.Copy();

            //soronként, ozloponként az elemeket összeadjuk
            for (int intRow = 0; intRow < this.RowCount; intRow++)
            {
                for (int intCol = 0; intCol < this.ColCount; intCol++)
                {
                    mtxResult.mBody[intRow, intCol] +=
mtxParam.mBody[intRow, intCol];
                }
            }
            return mtxResult;
        }

        /// <summary>
        /// önmagából kivonja a paramétert
        /// </summary>
        /// <param name="mtxParam">a kivonadó mátrix</param>
        /// <returns></returns>
        private Matrix Sub(Matrix mtxParam)
        {
            if (this.ColCount != mtxParam.ColCount || this.RowCount !=
mtxParam.RowCount)
            {
                throw new Exception("Kivonandó mátrixok nem azonos
méretűek!");
            }
            //az eredmény-mátrix
            Matrix mtxResult = this.Copy();

            //soronként, ozloponként az elemeket kivonjuk
            for (int intRow = 0; intRow < this.RowCount; intRow++)
```

```
            {
                for (int intCol = 0; intCol < this.ColCount; intCol++)
                {
                    mtxResult.mBody[intRow, intCol] -=
mtxParam.mBody[intRow, intCol];
                }
            }
            return mtxResult;
        }

        /// <summary>
        /// mátrixok szorzása
        /// </summary>
        /// <param name="mtxParam">a szorzás előtagja</param>
        /// <returns>mtxParam * this</returns>
        private Matrix PreMultiplication(Matrix mtxParam)
        {
            if (this.RowCount != mtxParam.ColCount)
            {
                //ha a szorzás előtagjának oszlopainak száma nem egyezik
                //a szorzás utótagjának sorainak számával, kivételt dobunk.
                throw new Exception("Hiba a mátrixok szorzásánál! Nem
megfelelő a szorzandó mátrix oszlopainak száma!");
            }

            //az eredmény-mátrix
            int intNewRowCount = mtxParam.RowCount;
            int intNewColCount = this.ColCount;
            Matrix mtxResult = new Matrix(intNewRowCount, intNewColCount);
            for (int i = 0; i < intNewRowCount; i++)
            {
                for (int j = 0; j < intNewColCount; j++)
                {
                    Double d = 0;
                    for (int k = 0; k < mtxParam.ColCount; k++)
                    {
                        d += mtxParam.mBody[i, k] * this.mBody[k, j];
                    }
                    mtxResult.mBody[i, j] = d;
                }
            }
            return mtxResult;
        }

        /// <summary>
        /// mátrixok szorzása
        /// </summary>
        /// <param name="mtxParam">a szorzás utótagja</param>
        /// <returns>this * mtxParam</returns>
        private Matrix PostMultiplication(Matrix mtxParam)
        {
            if (this.ColCount != mtxParam.RowCount)
            {
                //ha a szorzás előtagjának oszlopainak száma nem egyezik
                //a szorzás utótagjának sorainak számával, kivételt dobunk.
                throw new Exception("Hiba a mátrixok szorzásánál! Nem
megfelelő a szorzó mátrix sorainak száma!");
            }
```

```
        //az eredmény-mátrix
        int intNewRowCount = this.RowCount;
        int intNewColCount = mtxParam.ColCount;
        Matrix mtxResult = new Matrix(intNewRowCount, intNewColCount);
        for (int i = 0; i < intNewRowCount; i++)
        {
            for (int j = 0; j < intNewColCount; j++)
            {
                Double d = 0;
                for (int k = 0; k < this.ColCount; k++)
                {
                    d += this.mBody[i, k] * mtxParam.mBody[k, j];
                }
                mtxResult.mBody[i, j] = d;
            }
        }
        return mtxResult;
    }

    /// <summary>
    /// Skaláris szorzás egésszel
    /// </summary>
    /// <param name="intScalar">egész skalár</param>
    /// <returns></returns>
    private Matrix MultiplicationWithSkalar(int intScalar)
    {
        //az eredmény-mátrix
        Matrix mtxResult = this.Copy();

        //soronként, ozloponként az elemeket megszorozzuk a skalárral
        for (int intRow = 0; intRow < this.RowCount; intRow++)
        {
            for (int intCol = 0; intCol < this.ColCount; intCol++)
            {
                mtxResult.mBody[intRow, intCol] *= intScalar;
            }
        }
        return mtxResult;
    }

    /// <summary>
    /// Skaláris szorzás lebekőpontos számmal
    /// </summary>
    /// <param name="dScalar">lebegőpontos skalár</param>
    /// <returns></returns>
    private Matrix MultiplicationWithSkalar(Double dScalar)
    {
        //az eredmény-mátrix
        Matrix mtxResult = this.Copy();
        //soronként, ozloponként az elemeket megszorozzuk a skalárral
        for (int intRow = 0; intRow < this.RowCount; intRow++)
        {
            for (int intCol = 0; intCol < this.ColCount; intCol++)
            {
                mtxResult.mBody[intRow, intCol] *= dScalar;
            }
        }
        return mtxResult;
    }
```

6.2. Egyenes paraméteres egyenlete

BLL.Equations.Line.cs

```
        /// <summary>
        /// az egyenes paraméteres enyenletét tároló mátrix
        /// </summary>
        private Matrix mEquation;
        public Matrix Equation
        {
            get { return mEquation; }
        }


        /// <summary>
        /// a konstruktor két pontból állítja elő az egyenes paraméteres
egyenletét
        /// </summary>
        /// <param name="vecP">a P pont</param>
        /// <param name="vecQ">a Q pont</param>
        public Line(Vector4D vecP, Vector4D vecQ)
        {
            mEquation = new Matrix(new Double[,]{
                {vecQ.X - vecP.X, vecP.X},
                {vecQ.Y - vecP.Y, vecP.Y},
                {vecQ.Z - vecP.Z, vecP.Z},
            });
        }
```

6.3. Sík paraméteres egyenlete

BLL.Equations.Plain.cs

```
        /// <summary>
        /// a sík paraméteres egyenletét tároló mátrix
        /// </summary>
        private Matrix mEquation;
        public Matrix Equation
        {
            get { return mEquation; }
        }


        /// <summary>
        /// a konstruktor 3 pontból állítja elő
        /// a sík paraméteres egyenletét
        /// </summary>
        /// <param name="vecP">P pont</param>
        /// <param name="vecQ">Q pont</param>
        /// <param name="vecR">R pont</param>
        public Plain(Vector4D vecP, Vector4D vecQ, Vector4D vecR)
        {
            //ha egybeeső pontok a paraméterek, kivételt dobunk.
            //az Eguals metódust felüldefiniáltam
            if (vecP.Equals(vecQ) ||
                vecP.Equals(vecR) ||
                vecQ.Equals(vecR))
            {
                throw new Exception("Egybeeső pontok a sík paraméteres
egyenletének meghatározásánál!");
            }
            mEquation = new Matrix(new Double[,]{
                {vecP.X-vecQ.X, vecR.X-vecQ.X, vecQ.X},
                {vecP.Y-vecQ.Y, vecR.Y-vecQ.Y, vecQ.Y},
                {vecP.Z-vecQ.Z, vecR.Z-vecQ.Z, vecQ.Z}
            });
        }
```

6.4. Gauss-elimináció

BLL.Equations.Equations.cs

```
    public static class Equations
    {
        public const Double ZERO = 0.000001;

        /// <summary>
        /// a közelítésből adódó hibák miatt
        /// a 0-közeli értéket nullának tekintem
        /// </summary>
        /// <param name="d"></param>
        /// <returns></returns>
        public static bool IsZero(Double d)
        {
            return Math.Abs(d) < ZERO;
        }

        /// <summary>
        /// Gauss-elimináció
        /// visszahelyettesítéssel
        /// </summary>
        /// <param name="mtxParam"></param>
        /// <param name="vecResult"></param>
        /// <returns></returns>
        public static bool Gauss(Matrix mtxParam, ref Vector vecResult)
        {
            if (mtxParam.RowCount + 1 != mtxParam.ColCount)
            {
                throw new Exception("Nem megfelelő a mátrix mérete!");
            }


            Matrix mtxGauss = mtxParam.Copy();
            for (int intRow = 1; intRow < mtxGauss.RowCount; intRow++)
            {
                //főelem kiválsztás
                if (IsZero(mtxGauss[intRow - 1, intRow - 1]))
                {
                    int intChangeRow;
                    for (intChangeRow = intRow; intChangeRow <
mtxGauss.RowCount; intChangeRow++)
                    {
                        if (!IsZero(mtxGauss[intChangeRow, intRow - 1]))
                        {
                            //találtunk olyan sort, aminek az aktuális
eleme nem 0, csere
                            for (int intCol = 0; intCol <
mtxGauss.ColCount; intCol++)
                            {
                                Double temp = mtxGauss[intChangeRow,
intCol];
                                mtxGauss[intChangeRow, intCol] =
mtxGauss[intRow - 1, intCol];
                                mtxGauss[intRow - 1, intCol] = temp;
                            }
```

```
                            break;
                        }
                    }
                    if (intChangeRow == mtxGauss.RowCount)
                    {
                        //ezt a sort nem kell eliminálni
                        continue;
                    }
                }

                Double norm = mtxGauss[intRow - 1, intRow - 1];
                if (!IsZero(norm))
                {
                    for (int intCol = 0; intCol < mtxGauss.ColCount;
intCol++)
                    {
                        //normalizálás
                        mtxGauss[intRow - 1, intCol] = mtxGauss[intRow - 1,
intCol] / norm;
                    }
                }

                for (int intReducateRow = intRow; intReducateRow <
mtxGauss.RowCount; intReducateRow++)
                {
                    if (IsZero(mtxGauss[intReducateRow, intRow - 1]))
                    {
                        continue;
                    }

                    Double l = mtxGauss[intReducateRow, intRow - 1] /
mtxGauss[intRow - 1, intRow - 1];
                    for (int intCol = 0; intCol < mtxGauss.ColCount;
intCol++)
                    {
                        mtxGauss[intReducateRow, intCol] =
mtxGauss[intReducateRow, intCol] - l * mtxGauss[intRow - 1, intCol];
                    }


                }

            }
            for (int intRow = 0; intRow < mtxGauss.RowCount; intRow++)
            {
                Double norm = mtxGauss[intRow, intRow];
                if (IsZero(norm))
                {
                    continue;
                }
                for (int intCol = 0; intCol < mtxGauss.ColCount; intCol++)
                {
                    //normalizálás
                    mtxGauss[intRow, intCol] = mtxGauss[intRow, intCol] /
norm;
                }
            }

            vecResult = new Vector(mtxGauss.RowCount, true);
```

```
            //visszahelyettesítés
            for (int intRow = mtxGauss.RowCount - 1; intRow >= 0; intRow--)
            {
                if (IsZero(mtxGauss[intRow, intRow]))
                {
                    return false;
                }
                Double b = mtxGauss[intRow, mtxGauss.ColCount - 1];
                for (int intCol = intRow + 1; intCol < mtxGauss.ColCount -
1; intCol++)
                {
                    b -= mtxGauss[intRow, intCol];
                }
                vecResult[intRow] = b / mtxGauss[intRow, intRow];
                for (int intRowRefresh = 0; intRowRefresh <
mtxGauss.RowCount; intRowRefresh++)
                {
                    mtxGauss[intRowRefresh, intRow] *= vecResult[intRow];
                }
            }
            return true;
        }

    }
```

6.5. Sík és egyenes döféspontja

BLL.Equations.Plain

```
        /// <summary>
        /// sík és egyenes metszéspontja
        /// </summary>
        /// <param name="linParam">az egyens egyenlete</param>
        /// <param name="s1">ha van metszéspont,
        /// a sík első paramétere a metszéspont
        /// megadásához
        /// </param>
        /// <param name="s2">ha van metszéspont,
        /// a sík második paramétere a metszéspont
        /// megadásához
        /// </param>
        /// <param name="t">ha van metszéspont,
        /// az egyenes paramétere a metszéspont
        /// megadásához
        /// </param>
        /// <returns>true, ha pontosan egy darab
        /// metszéspnt létezik, különben false
        /// </returns>
        public bool Intersection(Line linParam, ref Double s1, ref Double
s2, ref Double t)
        {
            Vector vecIntersection = null;
            //az egyenletrendszer mátrixa 3x4-es
            Matrix mtxEquations = new Matrix(3, 4);

            for (int intRow = 0; intRow < this.mEquation.ColCount;
intRow++)
            {
                //az egyenletrendszer mátrixának feltöltése
                mtxEquations[intRow, 0] = this.mEquation[intRow, 0];
                mtxEquations[intRow, 1] = this.mEquation[intRow, 1];
                mtxEquations[intRow, 2] = -1 * linParam.Equation[intRow,
0];
                mtxEquations[intRow, 3] = linParam.Equation[intRow, 1] -
this.mEquation[intRow, 2];
            }

            if (!Equations.Gauss(mtxEquations, ref vecIntersection))
            {
                //ha nincs pontosan egy megoldás
                return false;
            }
            //visszaadjuk a paraméterek értékeit
            s1 = vecIntersection[0];
            s2 = vecIntersection[1];
            t = vecIntersection[2];
            return true;
        }
```

6.6. Síkok metszésvonala

BLL.Equations.Plain

```
        /// <summary>
        /// síkok metszésvonala
        /// </summary>
        /// <param name="plnParam">a paramétersík</param>
        /// <param name="line">visszaadjuk a metszésvonalat egyenletét,
        /// ha létezik
        /// </param>
        /// <returns>true, ha létezik megoldás
        /// különben false
        /// </returns>
        public bool Intersection(Plain plnParam, ref Line line)
        {
            Vector vecIntersection = null;
            //az egyenletrendszer mátrixa 3x4-es
            Matrix mtxEquations = new Matrix(3, 4);

            Double s1, s2, s3, s4;
            Vector4D vec1 = null;
            Vector4D vec2 = null;

            for (int intRow = 0; intRow < this.mEquation.ColCount;
intRow++)
            {
                //az egyenletrendszer mátrixának feltöltése s4 = 0 esetén
                mtxEquations[intRow, 0] = this.mEquation[intRow, 0];
                mtxEquations[intRow, 1] = this.mEquation[intRow, 1];
                mtxEquations[intRow, 2] = -1 * plnParam.Equation[intRow,
0];
                mtxEquations[intRow, 3] = plnParam.Equation[intRow, 2] -
this.mEquation[intRow, 2];
            }

            if (Equations.Gauss(mtxEquations, ref vecIntersection))
            {
                //ha van megoldás akkor adott az egyenes első pontja
                s1 = vecIntersection[0];
                s2 = vecIntersection[1];
                s3 = vecIntersection[2];
                s4 = 0;
                vec1 = this[s1, s2];
            }
            else
            {
                for (int intRow = 0; intRow < this.mEquation.ColCount;
intRow++)
                {
                    //az egyenletrendszer mátrixának feltöltése s3 = 0
esetén
                    mtxEquations[intRow, 0] = this.mEquation[intRow, 0];
                    mtxEquations[intRow, 1] = this.mEquation[intRow, 1];
                    mtxEquations[intRow, 2] = -1 *
plnParam.Equation[intRow, 1];
```

```
                    mtxEquations[intRow, 3] = plnParam.Equation[intRow, 2]
- this.mEquation[intRow, 2];
                }
                if (Equations.Gauss(mtxEquations, ref vecIntersection))
                {
                    //ha van megoldás akkor adott az egyenes első pontja
                    s1 = vecIntersection[0];
                    s2 = vecIntersection[1];
                    s3 = 0;
                    s4 = vecIntersection[2];
                    vec1 = this[s1, s2];
                }
                else
                {
                    return false;
                }
            }

            for (int intRow = 0; intRow < this.mEquation.ColCount;
intRow++)
            {
                //az egyenletrendszer mátrixának feltöltése s4 = 1 esetén
                mtxEquations[intRow, 0] = this.mEquation[intRow, 0];
                mtxEquations[intRow, 1] = this.mEquation[intRow, 1];
                mtxEquations[intRow, 2] = -1 * plnParam.Equation[intRow,
0];
                mtxEquations[intRow, 3] = plnParam.Equation[intRow, 2] -
this.mEquation[intRow, 2] + plnParam.Equation[intRow, 1];
            }

            if (Equations.Gauss(mtxEquations, ref vecIntersection))
            {
                //ha van megoldás akkor adott az egyenes második pontja is
                s1 = vecIntersection[0];
                s2 = vecIntersection[1];
                s3 = vecIntersection[2];
                s4 = 1;
                vec2 =this[s1, s2];
            }
            else
            {
                for (int intRow = 0; intRow < this.mEquation.ColCount;
intRow++)
                {
                    //az egyenletrendszer mátrixának feltöltése s3 = 1
esetén
                    mtxEquations[intRow, 0] = this.mEquation[intRow, 0];
                    mtxEquations[intRow, 1] = this.mEquation[intRow, 1];
                    mtxEquations[intRow, 2] = -1 *
plnParam.Equation[intRow, 1];
                    mtxEquations[intRow, 3] = plnParam.Equation[intRow, 2]
- this.mEquation[intRow, 2] + plnParam.Equation[intRow, 0];
                }
                if (Equations.Gauss(mtxEquations, ref vecIntersection))
                {
                    //ha van megoldás akkor adott az egyenes második pontja
is
                    s1 = vecIntersection[0];
                    s2 = vecIntersection[1];
```

```
                s3 = 1;
                s4 = vecIntersection[2];
                vec2 = this[s1, s2];
            }
            else
            {
                //nincs metszésvonal
                return false;
            }
        }

        //visszaadjuk a kép közös pont által határolt egyenest
        line = new Line(vec1, vec2);

        return true;
    }
```

Összefoglalás

A dolgozatom végére érve –érzésem szerint– egy olyan oktatási segédanyagot állítottam össze, mely segítséget nyújt mindazok számára, akik a komputergrafika alapjaival meg szeretnének ismerkedni. A fejezeteket sorrendjét úgy határoztam meg, hogy pontról pontra vegyem azokat a grafikai módszereket, amelyek szükségesek egy ilyen jellegű program elkészítéséhez.

A programom írása során én is hasonló sorrendben építettem fel az eljárásokat. A matematikai összefüggések leprogramozását követően a térbeli testek B-rep reprezentációjának megvalósítását és az adatelérési réteget állítottam össze. Célom volt, hogy az objektum-definiálástól a képi megjelenítésig minden szakaszt magam programozzak le, és a vetített képet DGI+ technológiával rajzoljam a képernyőre. Implementáltam a pont-transzformációkat, a koordináta-transzformációt, a térbeli testek síkra vetítését. Elkészítettem mindehhez a GUI felületet. A kezelőfelületen szinte minden, a modellezéshez szükséges paramétert beállítható, hogy a felhasználó érzékelhesse a különböző paraméterezettségek közötti különbséget. Ahhoz, hogy a testek egymáshoz és a kamerához viszonyított elhelyezkedését jobban átláthassa a felhasználó, egy külső kameranézetet is megvalósítottam. Működött is minden addig, amíg el nem jutottam a megjelenítendő lapok sorba rendezéséig. Beépítettem a programba a Z-buffer algoritmust, de az eljárásom úgy lelassította a programot, hogy az használhatatlanná vált. Próbáltam különböző optimalizálási módszereket, nagyobb tömböket, ezáltal több memóriát használva azt remélve, hogy gyorsabb lesz a megjelenítés, de látványos eredményt nem értem el. A probléma oka a nagyobb felületek bufferbe „rajzolása” volt. Váltottam Ray-tracing módszerre. Gyorsaságbeli különbséget nem értem el. Ha ezzel az eljárással nagyítottam a program ablakán, így a rajzfelület mérete megnőtt, ezáltal annyi metszéspontot kellett számolni, hogy a kirajzolás továbbra is lassú maradt. Megpróbáltam saját sorba rendező algoritmust gyártani, de tökéletesen működőt nem sikerült alkotnom. Emiatt döntöttem úgy, hogy felhagyok a GDI+ megjelenítéssel, és áttérek olyan megoldásra, ami a grafikus kártya adottságait kihasználja. Minden eljárást meghagytam a programban, de az elemek kirajzolásánál az OpenGL mellett döntöttem.

Azt a következtetést vontam le, hogy ha komolyabb animációt, grafikus programot szeretnénk készíteni, az alap Windows grafikai eszközök használata kevésnek bizonyulnak.

Nehéz olyan optimális algoritmus alkotni ennek a problémának a megoldására, ami tökéletesen működik, és emellett gyors is. Meghagytam viszont azt a lehetőséget, hogy a későbbiek során szerzett ismereteim alapján továbbfejleszthessem az alkalmazást.

Ezt a problémát leszámítva a program, és annak forrása alkalmas arra, hogy bemutassa a fent leírt eljárásokat, és ötleteket nyerjen a felhasználó a saját programjának elkészítéséhez.

A programot és a dokumentációt a nehézségek ellenére is igyekeztem a legjobb tudásom szerint megírni, mindvégig ügyelve arra, hogy a későbbiek során munkám, a téma iránt érdeklődőknek segítséget nyújtson.

Irodalomjegyzék

1. Alan Watt [2000]: 3D Computer Graphics
Addision-Wesley, Pearson Education, Harlow, England, Thrid edition

2. Bácsó Sándor [2002]: Diszkrét matematika programtervező matematikusok számára
Debreceni Egyetem, Matematikai és Informatikai Intézet, Debrecen

3. Bácsó Sándor [2003]: Fejezetek a geometriából
Eszterházi Károly Főiskola, Eger

4, Dr. Kovács Emőd, Hernyák Zoltán, Radványi Tibor, Király Roland [2005]:
A C# programozási nyelv a felsőoktatásban
Programozási tankönyv
Eszterházy Károly Főiskola, Eger

5, Dr. Szirmay – Kalos László [1999]: Számítógépes grafika
ComputerBooks, Budapest

6. Julian Templeman, David Vitter [2002]: Visual Studio .NET -
The .NET Framework Black Book
The Coriolis Group, LLC

7. Juval Löwy [2004]: .Net komponensek programozása
Kossuth kiadó, Budapest

Online irodalmak:

1. A CodeProject oldalán nagyon hasznos programozás-technikai megvalósításokat találhatunk, a regisztráció ingyenes: http://www.codeproject.com/

2. Szegedi Tudományegyetem, Informatikai Tanszékcsoport: Az OpenGL grafikai rendszer Utoljára frissítve: 2000. május 28.
http://www.inf.u-szeged.hu/oktatas/jegyzetek/KubaAttila/opengl/peldak/starthu.xml