Pirkó József

3D

Perspektívikus grafika IBM PC-n Turbo-Pascalban

A TÉRLÁTÁS ELMÉLETE ÉS BIOLÓGIAI MODELLJE

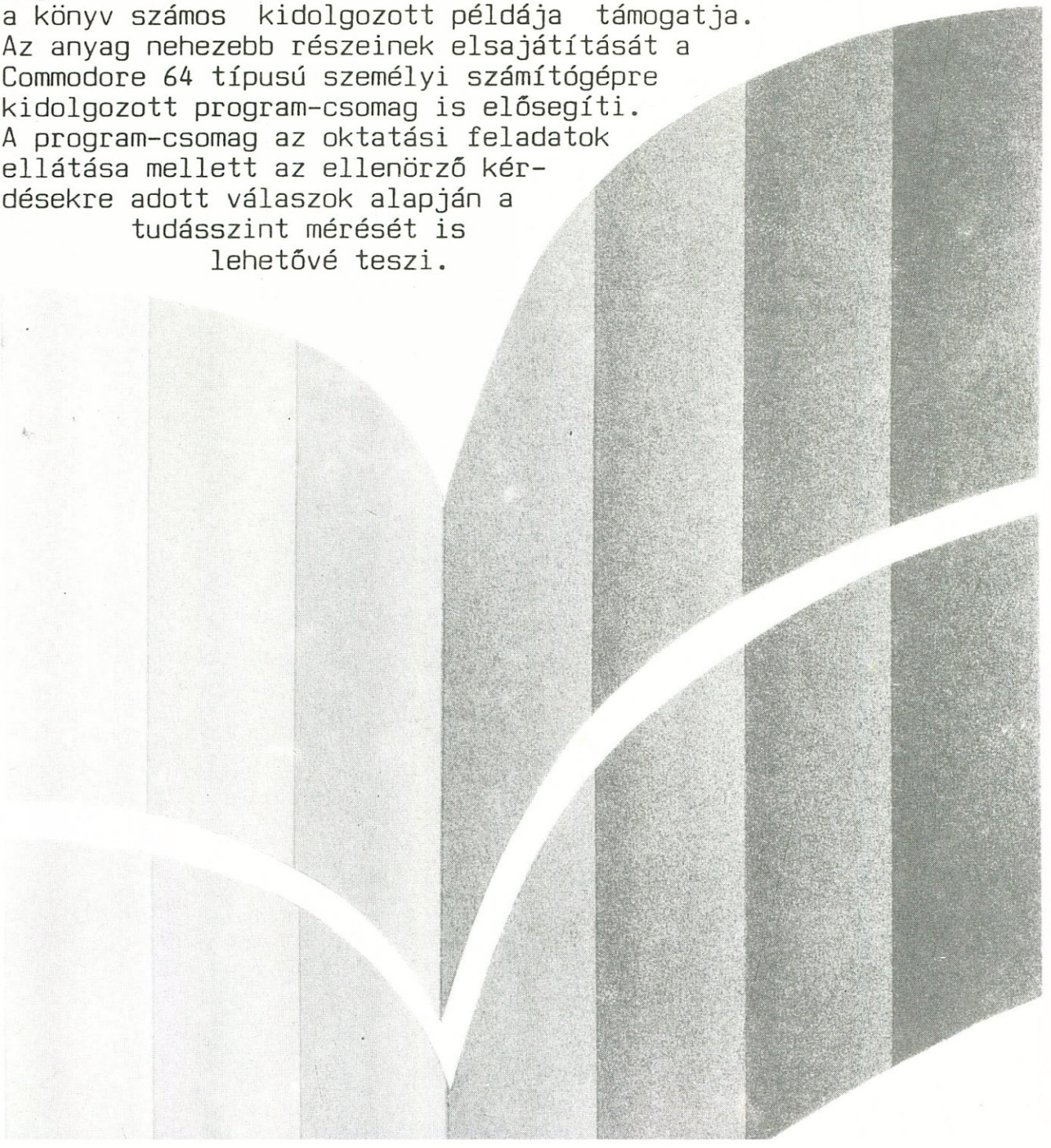
DIGITÁLIS VIDEOFILM-TECHNIKA

LSI ALKALMAZÁSTECHNIKAI TANÁCSADÓ SZOLGÁLAT

DR. OBÁDOVICS J. GYULA

MATEMATIKA

A könyv két kötete a középiskolai, technikumi matematika anyagot rendszerezett formában tárgyalja. Segítséget kíván nyújtani a korábban tanult ismeretek felfrissítéséhez, újratanulásához, illetve kibővítéséhez. A " Nyitott Egyetem " matematikai és szaktárgyi anyagának elsajátításához a könyv elméleti anyagának biztos tudása mellett, annak gyakorlati alkalmazásában is kellő jártasságot kell szerezni. Az Olvasót ebben való törekvésében a könyv számos kidolgozott példája támogatja.
Az anyag nehezebb részeinek elsajátítását a Commodore 64 típusú személyi számítógépre kidolgozott program-csomag is elősegíti. A program-csomag az oktatási feladatok ellátása mellett az ellenörző kérdésekre adott válaszok alapján a tudásszint mérését is lehetővé teszi.



Pirkó József

3D

Perspektívikus grafika IBM PC-n Turbo Pascalban

A TÉRLÁTÁS ELMÉLETE ÉS BIOLÓGIAI MODELLJE
DIGITÁLIS VIDEOFILM-TECHNIKA

LEKTORÁLTA: HORVÁTH LÁSZLÓ

ALKALMAZÁSTECHNIKAI TANÁCSADÓ SZOLGÁLAT
BUDAPEST, 1988

Készült: Révai Nyomda Egri Gyáregység
Felelős vezető: Horváth Józsefné dr.
Ív: 8,5 (A/5) ív
Példányszám: 4 000
Kiadó: LSI ATSZ
Felelős kiadó: Dr. Kovács Magda
Témafelelős: Székely László
ISBN: 963 592 716 9
Engedélyszám: 60 241

T A R T A L O M

1. B E V E Z E T É S

Már az ókori egyiptomiakat is foglalkoztatta a látás gondolata, a szem felépítése. Megállapították, hogy a minden fény - és így a látás - eredete a Nappal hozható összefüggésbe. Ezt az általuk már fontosnak tartott kapcsolatot kifejezi az is, hogy maga Ozirisz napisten a hieroglifákkal teleírt papíruszokon egy szem formájában jelenik meg. Később a szem, mint szimbólum már csak e legfontosabb isten nevének kezdőbetűjét, azaz az O-betűt jelképezte. Így alakult ki tehát az 'o' betű, mely formájánál fogva kifejezi a szem gömb alakját is. A szem fontosságát jelzi az is, hogy a görög mondákban felbukkanó Európa - a föníciai királylány - volt Zeusz szerelme, mivel szép nagy szemei voltak. Róla kapta kontinensünk a nevét, melynek jelentése szabad fordításban nem más, mint nagy szem, egész pontosan: Eurus jelentése: tág, 'ops' pedig szemet jelent. A kettő összevonásából keletkezik az 'Európa' szó. Mint látjuk, a 'szem' itt 'o'-kezdetű szó. Sok más nyelvben is 'o' betűvel kezdődik, például a héberekné az O-betű megfelelője szemet is jelent, a szem franciául oeil, oroszul oko (régiesen), görögül ophthalmos, latinul oculus, olaszul occhio, hollandul ooge.

A szem fontosságát tehát már igen korán felismerték és a következő korokban egyre többen, mind alaposabban kezdték megismerni szerkezetét, tulajdonságait. Ezzel párhuzamosan elkezdték a látás javítását is. Néró császár már csiszolt drágaköveket használt látása javítására. Ibn Ali Haitham már 1000 körül leírta az optika alaptörvényeit. Rogerius Bacon (1214-1294, angol) kidolgozta a szemüveg elméletének tekintélyes részét, melyet a gyakorlatban Salvino d'Armato (XIII-XIV. sz., olasz) valósított meg. Ekkor még csak plánkonvex lencsék voltak, bikonvexek a XV. századtól. Az első biztosan konkáv lencse Raffael híres, X. Leo pápát ábrázoló festményén látható. A pápa - mint a Medici család sok más tagja - örökletes rövidlátásban szenvedett. A lencséket egyre tökéletesítették, ebben nagy szerepük volt Duncker német és Wollaston angol tudósoknak, majd a bifokális szemüveg feltalálójának, Benjamin Franklinnak. A szem szerkezetének megismerésében igen nagy szerepe volt a múlt században Helmholtz német fiziológusnak, aki 1851-ben felfedezte a szemtükröt, ezzel utat nyitott a szem megismerése felé. A kutatás másik iránya a valóság mind tökéletesebb rögzítését tűzte ki célul maga elé. A cél az volt, hogy a valóságot valamilyen képsíkon maradandóan rögzíthessük, így egy adott pillanatbeli állapotot bármikor ismét megnézhessük. Már Leonardo da Vinci (1452-1519) két igen fontos szerkezetet tervezett.

Az olasz polihisztor 5300 tollrajza között található a Codex Atlanticus-ban megjelent Camera Obscura és a Laterna Magica is. A Camera Obscura - a tulajdonképpeni sötétkamra - már képes volt arra, hogy a valóságról alkotott képet egy képsíkon megjelentesse. Eleinte ezt a fordított állású képet csupán pontosan körülrajzolták, később pedig a képsík helyére fényérzékeny lemez került. Így jött létre a fényképezőgép, melynek feltalálásában és tökéletesítésében igen nagy szerepe volt többek közt Daguerre-nak, Talbotnak, Herschel-nek, Niépec-nek és Bayard-nak. A sötétkamra és a szem szerkezete nagyon hasonlítanak egymásra, Leonardo a szem anatómiáját tanulmányozva alkotta meg ezt a szerkezetet. A Laterna Magica pedig a képeket ki tudta vetíteni. E két szerkezet nyomán később számtalan optikai eszköz jelent meg, mint pl. a mikroszkóp, a távcső és a kamera.

A következő lépcsőfokot a számítógépek megjelenése jelentette. Míg az eddigi eszközök a látás folyamatának megismerését és kiszélesítését célozták, a számítógéppel már modellezhetjük is a szem működését. A szem és az optikai eszközök működésének ismeretében - azoknak matematikai pontosságú leírása után - az általuk előállított képet szimulálni is tudjuk. Így a szem ideghártyáján és a sötétkamrában keletkező kép hasonló matematikai szabályok szerint épül fel, ezeket definiálva a számítógép képernyőjén is elő tudjuk állítani ezeket a képeket. A különbség annyi, hogy ehhez már nem is szükséges feltétlenül a valóság, hanem 'csupán' a valóságot - vagy számításokkal, adatokkal megadott, elméleti testeket - kell leírni valamilyen formában. Így a valóság és az elméleti vagy tervezett objektumok egyaránt megjeleníthetők. Az ily módon készült számítógépes ábrázolás fontos jellemzője a három dimenziós hatás elmélyülése, mivel az ábrázolt objektumokat bármilyen szögből és közelségből megfigyelhetjük. Ez a megjelenítési mód leginkább a holografikus ábrázolással mutat rokonságot, bár a kettő között igen sok eltérést is megfigyelhetünk.

Ez tehát többek közt a számítógépes grafika egy fontos területe. Ezen kívül sok más probléma megoldására és szemléltetésére igen hasznos eszköz a számítógépes grafika. A problémák megoldása során gyakran szükségünk van a mennyiségek szemléltetésére és bemutatására. A mennyiségek, adatok szemléltetésének gyakori módja a függvények, grafikonok, oszlop diagrammok ábrázolása, mivel ezek által képszerűen magunk előtt láthatjuk eredményeinket. Általában többet mond számunkra a grafikus eredmény, mint egy számhalmaz.

A számítógépes grafikának igen sok alkalmazási területe van. Sokszor maga a grafika a végtermék, de leginkább valamilyen köztes funkciót tölt be. Nagy szerepe van a mérnöki tervezésben, a CAD (Computer Aided Design, számítógépes műszaki tervezés) rendszereknek önmagukban is sok alkalmazási területük van. Műszaki rajzok, áramkörök tervezésére használják őket többek közt. Más alkalmazások pl. az animáció, a molekula-modellezés, a szimuláció, a modellezés, gyógyszerkutatás, képfeldolgozás, alakfelismerés, oktatás, mozgó képek tárolása, elemzése és a művészet. Mindezek *napjaink dinamikusan fejlődő területei.*

A felsorolt alkalmazások többsége igényli is a három dimenziós megjelenítést, mivel két dimenzió nem mindig eléggé szemléletes ábrázolási forma, sokkal vizuálisabb a három dimenziós megjelenítési mód, s ezen belül is a mozgó, térbeli grafikák, mivel ezek állnak legközelebb a látásunkhoz. Ez esetben legfőbb törekvésünk, hogy a térbeli objektumok, testek ábrázolása minél valósághűbb legyen. Egy létező - álló,vagy mozgó objektumról - bármely irányból, távolságból mozgó filmet készíthetünk. Ugyanezt szeretnénk megvalósítani nemlétező - csak a számítógép memóriájában definiált - testek esetén is, az optikai filmekhez mérhető minőségben.

Célunk tehát egy olyan, számítógépre alkalmazható algoritmus létrehozása,amely a *lehető legnagyobb mértékben igazodik látásunkhoz.* Ahhoz, hogy a látásunknak ezt a tipusú szimulációját meg tudjuk valósítani, meg kell ismernünk alaposan szemünk szerkezetét, látásunk mechanizmusát, s ezt - az alkamazott hardware és software lehetőségeit figyelembe véve - számítógépen megvalósítva, olyan algoritmust kell előállítanunk, amely szimulálja, lemásolja a látási mechanizmust, az optikai eszközöket. Másként fogalmazva: a látást számítógáppel szimuláljuk.

Kövessük tehát végig ezt az utat lépésről lépésre, a kérdést különböző oldalakról megvilágítva, folyamatosan haladva a fentebb megfogalmazott cél, az általános számítógépes térlátás felé. Induljunk ki először tehát a szemünk szerkezetének *biológiai modelljéből.* Nem célunk a szem teljes anatómiai ismertetése, csupán azokat a részeket emeljük ki, amelyeket a számítógépes modell alkotásához szükséges. A felvázolt biológiai alapokra - a fény fizikai természetét figyelembe véve - épül fel a *matematikai modell.* Ez nem más, mint a látásunk leírása, egzakt képletekben megfogalmazva a felismert biológiai és fizikai törvényszerűségeket.

Több lényeges kérdést kell tisztázni, mely fontos részét alkotja a látás elméletének, mint pl. a fény-árnyék, a láthatóság, a takarás, a tónusok. Nem érdektelen ezeket több oldalról is elemzés alá vetni, mivel a három dimeziós tér tulajdonságai, annak analitikus geometriája sok érdekességet rejt magában. A különféle matematikai módszerek segítségével a felmerülő problémákat megoldhatjuk, elemezhetjük. Nemcsak a matematikai modellt állítjuk fel, hanem az abban szereplő leképezéseket is elemezzük, megállapítva tulajdonságaikat, érvényességi köreiket. Ezek az elméleti vizsgálatok alkotják ennek a könyvnek jelentős részét. A matematikai modell célja nemcsak a tér és a térlátás tulajdonságainak vizsgálata, hanem az is, hogy utat nyisson a *számítógépes megvalósítás* felé.

Az objektumokat mozgó, dinamikus formában a leghatásosabb ábrázolni, mivel ez a forma áll a legközelebb az általunk megszokott térlátáshoz. Ezért ehhez kapcsolódva egyfajta számítógépes digitális *videotechnikáról* is olvashatunk a könyv 7. fejezetében. A vizsgált modellek számítógépes alkalmazását szemléltetve néhány program is bemutásra kerül.

Nem az a célunk, hogy bemutassunk minden módszert és algoritmust, mely a térbeli grafikus ábrázoláshoz kapcsolódik, bár ez is egy igen szép feladat lenne, láthatnánk a kidolgozott témák sokszínűségét, a megoldások ötletgazdagságát. Az érintett témakörök nagy terjedelme miatt ez nem lehetséges e könyv keretein belül. A témánk szempontjából viszont minden lényeges kérdést és megoldást érdemes elemezni, tehát ilyen értelemben viszont szeretnénk *teljes képet* adni.

Mivel az IBM - vagy azzal kompatibilis - személyi számítógépek nálunk egyre több teret hódítanak, ezért a bemutatott eljárások általában ilyen típusú számítógépekre készültek, többnyire a nálunk (is) egyre népszerűbb *Turbo Pascal* programozási nyelven.

2. Szemünk felépítése, látásunk biológiai modellje

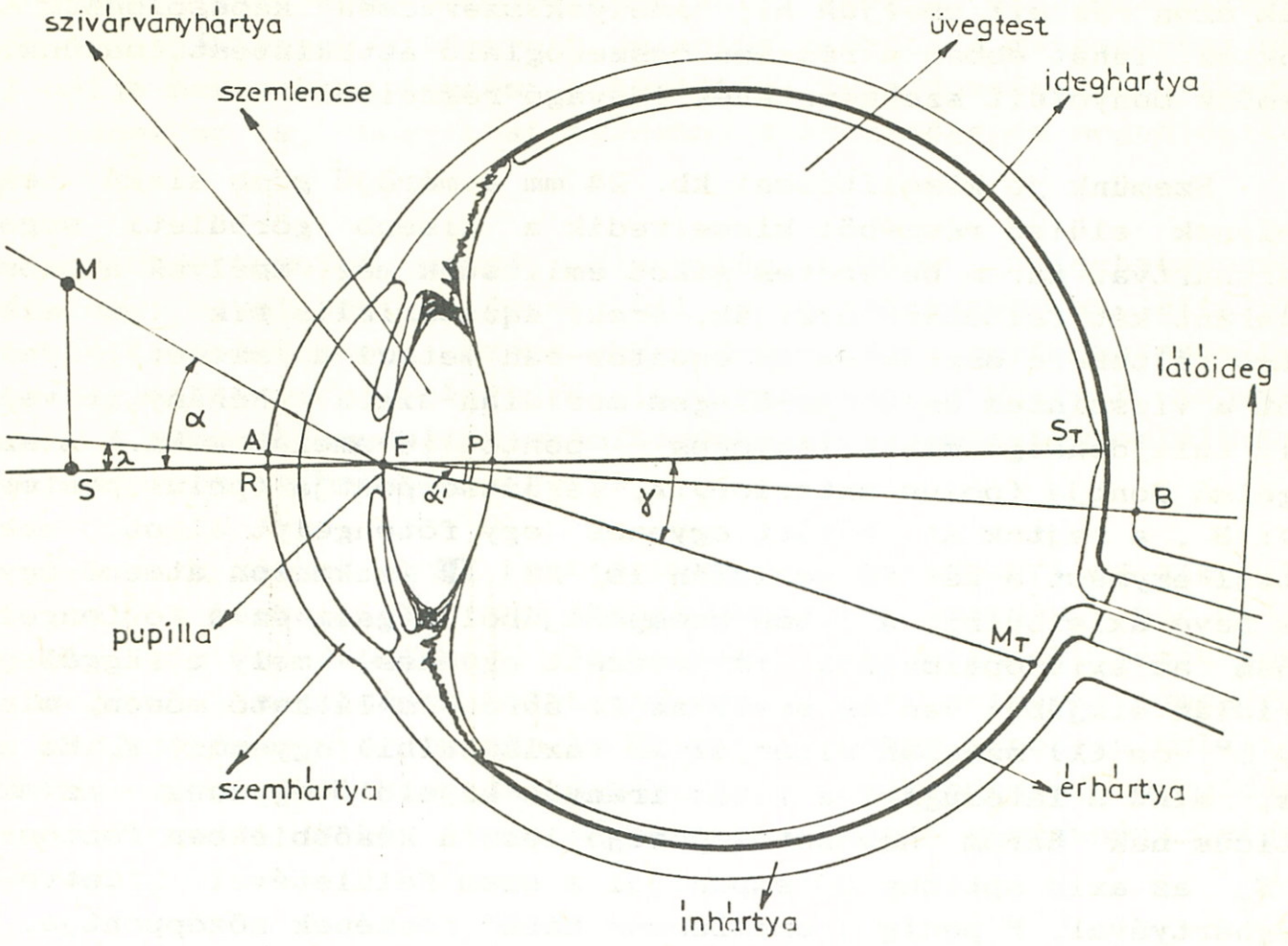
2.1. A szem részei, felépítése

Olyan algoritmust szeretnénk létrehozni, amely a szemünk működését szimulálja a lehető legteljesebb mértékben, s hogy ezt megvalósíthassuk, meg kell ismernünk szemünk felépítését, de csak a kellő mértékben, azaz az anatómiai és fiziológiai leírásoknak csak *azon részeit emeljük ki*, amelyek szervesen kapcsolódnak témánkhoz. Tehát ebben a részben összefoglaló áttekintést teszünk a szemünk bonyolult szerkezetének idevágó részeiről.

Szemünk jó közelítéssel kb. 24 mm átmérőjű gömb alakú test, amelynek elülső részéből kiemelkedik a kisebb görbületi sugarú szaruhártya. Három nevezetes síkot említsünk meg, amelyek a szemet egyaránt két féltekére osztják, ezek: equatorialis sík (ez első-hátsó féltekére osztja és az equator-ban metszi a szemet), valamint a vízszintes és a függőleges meridián-síkok. Néhány - valamely tulajdonsága miatt lényeges - pontot is emeljünk ki. A szem legelső pontja (polus anterior): A, leghátsó pontja (polus posterior): B , a rajtuk át húzott egyenes egy főtengelyt alkot, ebben metszi egymást a két fő meridián is. Az $\overline{AB}$ szakaszon átmenő egyenes neve axis bulbi. A látás szempontjából mégsem ez a fő tengely, hanem az axis opticusnak is nevezett egyenes, mely a függőleges meridián síkjában van és amely az 1. ábrán is látható módon, mintegy 5° -os (λ) szögben eltér az $\overline{AB}$ (axis bulbi) egyenestől. Ez nem más, mint a látósugár, a látás irányát kijelölő egyenes. Az axis opticus-nak három nevezetes pontja lesz a későbbiekben fontos: R és S_T az axis opticus döféspontjai a szem felületével, illetve az ideghártyával, P pedig a szemlencse hátsó részének középpontja, ez az optikai középpont. Később részletesebben is tárgyaljuk. Az S_T pont környeztének neve sárgafolt, másnéven fovea centralis.

A szemlencse 9 mm átmérőjű, változtatható mélységi átmérőjű, majdnem bikonvex lencse, amelynek hátsó fele domborúbb az elsőnél. Ez rugalmas, fókusza állítható, pl. a körkörös Müller-izom segítségével. Ilymódon a beeső fénysugarak a különböző távolságban lévő tárgyakról is éles képet adhatnak, a távolabbihoz laposabb, a közelebbihez domborúbb lencse szükséges. A gyújtótávolság mértékegysége a dioptria, a gyújtótávolság méterben kifejezett reciptroka. Nyugalmi helyzetben ez kb. 60 dioptria, használat közben - egészségesen, fiatal korban - kb. 14-74 dioptria (alkalmazkodási szélesség) között változik.

A közelre való alkalmazkodás (akkomodáció), ezzel együtt az alkalmazkodási szélesség mértéke idősebb korban egyre csökken, a közelpont - a legkisebb távolság, ahonnan még élesen látunk - egyre növekszik, előbb évente néhány millimétert, majd egyre többet, a fiatalkori néhány cm -ről elérve az átlag 60-100 cm körüli értéket is. A következő (1.) ábrán tanulmányozhatjuk a szem szerkezetét:



1.ábra: a szem szerkezete, felépítése

$\overline{AB}$: *axis bulbi*
A : *polus anterior, legelső pont*
B : *polus posterior, leghátsó pont*
P : *az optikai középpont*
α : *fénysugár beesési szöge*
α': *fénysugár törési szöge*
γ : *az* $S_T PM_T$ *szög*
λ : *az axis bulbi és a látósugár által bezárt szög*

$\overline{RS_T}$: *axis opticus (látósugár)*
R : *axis opticus elülső pontja*
S_T: *fovea centralis, sárgafolt*
S : *a szem által nézett pont*
M : *másik térbeli pont, melyre*
$\overline{SM}$: *az axis opticusra merőleges szakasz*
M_T: *M képe az ideghártyán*

A következő, tájékoztató jellegű táblázat, néhány átlagos adatot közöl a szemnek az 1. ábrán is látható részeiről, milliméterben:

A szaruhártya (cornea) vastagsága:	0,5
A szaruhártya távolsága a lencse első felületétől	3,6
A szaruhártya első felének sugara	7,7
A szaruhártya hátsó felének sugara	6,8
A szemlencse vastagsága (nyugalmi helyzetben)	3,6
ebből: a lencse magjának vastagsága	2,419
A lencse első felének sugara	10,0
A lencse hátsó felének sugara	6,0
A lencse magja (nucleus) első felének sugara	7,911
A lencse magja hátsó felének sugara	5,76
Az optikai középpont és a sárgafolt távolsága (PS_T)	15

2.2. Szem és fény viszonya, fénytörés és Helmholtz-állandó

A fénysugarak tehát a szaruhártyán, a szemcsarnokon és a szivárványhártya átlag 4 mm átmérőjű nyílásán, a pupillán keresztül a érik el a szemlencsét. Onnan - a lencse fénytörésének megfelelően módosulva - összetartó sugarakként az üvegtesten áthaladva elérik az ideghártyát, másnéven retinát. Az itt keletkező kép fordított állású, tartalmaz szín- és tónusinformációkat is. Ezt vizsgáljuk meg közelebbről, később pedig pontosítjuk a matematikai modellhez való hozzárendelést.

A látás szempontjából igen fontos tényező a fénytörés. Ha a fénysugár az egyik átlátszó anyagból egy másikba halad át, akkor iránya legtöbbször megtörik. Így van ez a szem esetén is, ahol az 1. ábrán látható módon az α beesési szöggel érkező fénysugár α' törési szöggel halad tovább. Ezek a beesési merőlegestől mért szögek. Érvényes rájuk a Snellius-Descartes törvény:

$$n \cdot \sin(\alpha) = n' \cdot \sin(\alpha')$$

esetünkben n: a levegő, n': a szaruhártya törésmutatói. A törésmutató az anyagra adott frekvencián jellemző állandó, melyet a levegő törésmutatójához - mint egységhez - viszonyítunk. Így pl. a szemlencse törésmutatója: 1,45; az üvegtesté vagy a vízcsarnoké: 1,3333 (akár a tiszta vízé).

Ebből egy igen fontos körülmény adódik, melyet vizsgáljunk meg részletesebben. A tárgyakról érkező fénysugarakat egy - a célra alkalmas - ernyőn fel tudjuk fogni. Ha a fény homogén közegben halad, akkor a testről - nevezzük így - *szabályos* képet kapunk. Legyen azonban a test a kisebb törésmutatójú - azaz optikailag ritkább - közegben és a róla érkező fénysugarak rövidesen egy optikailag sűrűbb közegbe. Ez esetben az előző, 'szabályos' képhez hasonló, nála kisebb képet kapnánk az ernyőn. Így történik ez a szem esetén is. Ha a második közeg optikailag ritkább, akkor a kapott kép a szabályosnál nagyobb lesz.

A fény útját befolyásolhatjuk még különböző lencsékkel: a domború lencse a kétszeres fókusznál nagyobb távolságban lévő testekről valódi, kicsinyített képet ad, a homorú pedig nagyít. Minden lencsére igaz viszont az un. lencseegyenlet: $\frac{1}{k} + \frac{1}{t} = \frac{1}{f}$, ahol f: a fókusz, k: a kép, t: a tárgy távolsága. A szem esetén a levegőből az optikailag sűrűbb anyagba jut a fény, de a törésmutató a szemen belül sem állandó, anyagonként különböző. Meg kell vizsgálni még azt is, hogy a szem azon részei, melyek a fénysugár útjába esnek, mint lencsék, hogy viselkednek. Jelöljük "-"-jellel a homorú, "+" jellel pedig a domború lencséket. Ilyenformán a fénysugár útjába kerülő részek ebből a szempontból a következő tulajdonságúak: szaruhártya: - , vízcsarnok: + , szemlencse héja: - , szemlencse magja: + , üvegtest: - .

Összevetve a fent említett körülményeket - melyeknek hatása sokszor ellentétes - azt mondhatjuk, hogy egy testről érkező, és az ideghártyán keletkező kép mindezen hatások miatt valamilyen mértékben kisebbedik. Ezt a mértéket kell pontosan meghatározni ahhoz, hogy az ideghártyán keletkező képet matematikailag konkrét formában, egyenletekkel is le tudjuk írni.

Látható, hogy igen sok tényező befolyásolja a vizsgált kérdést. Az 1. ábrán lévő γ -szög keletkezése a következő. Adott az $\overline{SM}$ szakasz, ahol S a szem által nézett pont, tehát az axis opticuson van, $\overline{SM}$ pedig az axis opticusra merőleges szakasz, az ábra szerint. S és M képe az ideghártyán: S_T (sárgafolt) és M_T. Ez a jelölés is azt kívánja érzékeltetni, hogy S_T és M_T pontok az S és M pontokból származtathatóak, valamilyen T transzformáció segítségével. Ezt a transzformációt fogjuk meghatározni a matematikai modellt leíró részben. Az S_TPM_T szöget pedig nevezzük γ -nak.

A legfontosabb kérdés a γ/α arány értékének meghatározása, mert ez azt mutatja, milyen mértékben kisebbedik az ideghártyára vetülő kép. A későbbiekben, a matematikai modell felépítésénél játszik majd ez a szám szerepet. Az arány meghatározásához igen sok tényezőt kellett figyelembe venni, mint a fentiekből is látható. Elvégezve az ehhez szükséges számításokat és méréseket, ezt a központi fontosságú tényezőt elsőként Helmholtz -nak sikerült meghatározni:

$$\gamma/\alpha = 0.81$$

Ez a szemünk működésére jellemző egyik legfontosabb arány a számítások, mérések és kísérletek tanúsága szerint nem változik, értéke *állandó*. Ez is hozzájárul szemünk precíz működéséhez. Mivel a szem közel gömb alakú, ezért az ideghártyán keletkező kép is jó *közelítéssel ugyanilyen arányban kissebbedik*.

A matematikai modell felírásakor azonban ezt a kérdést tovább egyszerűsíthetjük. Az ideghártya felé haladó fénysugarak közül azok, amelyek a 'P' optikai középponton - amely a szemlencse hátsó részében, középen van - haladnak át, töretlenül megtartják irányukat. Az optikai középpont 15 mm távolságra van a retinától. Ebből könnyen kiszámíthatjuk az ideghártyán megjelenő kép nagyságát. Ha pl. egy méteres nagyságú testet 15 m-ről nézünk, annak a képe 1 mm-es lesz a retinán. A párhuzamos szelők tétele alapján - amelyet ebben a speciális esetben alkalmazhatunk - a tárgy és a kép nagyságának aránya azonos a távolságuk arányával. A tárgy távolságán egész pontosan az optikai középponttól (P) mért távolságot értjük, a kép távolságán az ideghártya - azaz a képsík - és az optikai középpont távolságát. Ezt a 15 mm -es képsík-távolságot a matematikai modell felírásánál is figyelembe vesszük, mert ott a szemet egyetlen pontba - az optikai középpontba - fogjuk transzformálni.

Az említett képsík-távolság körülbelül 80 százaléka a fénysugár szemlencsébe való belépési pontja és az ideghártya távolságának. Ez az érték a Helmholtz-állandó egy becslését szemlélteti, amelyet úgy is tekinthetünk, hogy a különböző fénytörések miatt a képsík közelebb jön (0,81 százalékára) ahhoz képest, mintha a fénytörések nem lennének. Így tehát mindkét megközelítésből kiindulva (a Helmholtz-állandóval és az optikai középponttal) gyakorlatilag azonos eredményre jutunk a későbbi számításokban, tehát egyszerűbb az optikai középponttal számolni.

2.3. *Az ideghártya felépítése, a látótér. A szem szerkezete alapján tervezett, optimális felépítésű képernyő*

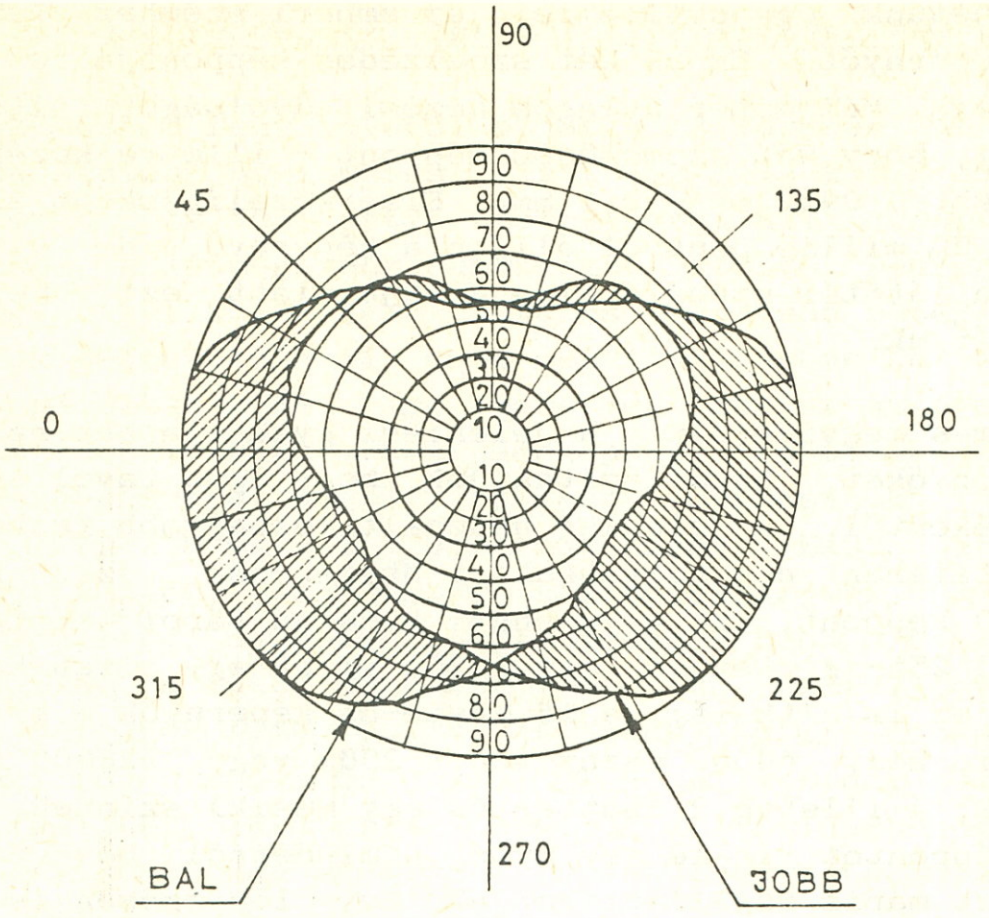
Nézzük most a fény további útját. Az ideghártyán keletkező kép tulajdonságainak megismerése miatt vizsgáljuk meg az ideghártya felépítését. Az ideghártya tíz rétegből áll. Bonyolult szerkezetéből csak a lényegesebb funkciókat betöltő részeket emeljük ki. Az üvegtesttel közvetlenül határos fényvédő réteg a pigmenthám, melynek sejtnyúlványai erős megvilágításban bevándorolnak a csapok és a pálcikák közé, így védve meg azokat az erős fénytől. A csapok és a pálcikák a tulajdonképpeni fotoreceptorok, ezek alkotják az ideghártya második rétegét; összesen kb. 5-7 millió a csapok és 125-130 millió a pálcikák száma. Ezután a kapcsolódó neuronok következnek, majd pedig a látóidegsejtek. A kapcsolódó idegsejtek feladata: összeköttetést tartanak fenn a csapok és pálcikák, valamint a látóidegsejtek között. Ezáltal továbbítják a csapok és a pálcikák által felfogott látási információkat a látóidegsejteknek. Ez utóbbiakból indulnak ki a látóideg rostjai. Minden látóidegsejtből egy rost, amelyek a látóidegben egyesülnek. Ezen át jut a látott kép az agyba. A látóidegsejtek száma kb. 50 millió.

Ez határozza meg a szem felbontóképességét, mert egy látóidegsejt egy képpontnak felel meg. A látás alkalmával a képet mintegy 50 millió képpontra bontjuk fel. Mivel ez a ponthálózat igen finom, ezért a látott kép egyöntetűnek tűnik. A recehártya kb. 4 cm^2 felületén oszlik el az 50 millió látósejt, így egy látósejtre átlagosan $8 \cdot 10^{-6}$ mm^2 terület jut. Ez megfelel mintegy 0,00283 mm oldalú négyzetnek; ennyi tehát a távolság két látósejt - pontosabban két látósejt középpontja - között. A látósejtek távolsága az optikai középponttól mintegy 15 mm; így ha két - az optikai középpont felől érkező - fénysugár két szomszédos látósejtre jut, akkor az általuk bezárt szög: $\mu \backsim 0{,}648$ szögperc, azaz: $\mu \backsim \mathrm{arctg}(0{,}00283/15)$. Ezt a tényt úgy is kifejezhetjük, hogy az optikai középpontból nézve az ideghártyán két látósejt átlagos távolságának látószöge: $\mu \backsim 0{,}648$ perc. Így pl. két csillag akkor látszik kettőnek, ha két látóidegsejtet hoz ingerületbe, azaz legalább μ látószöget zárnak be; ha ennél kisebbet, a két csillag egynek látszik. Minden adat, amelyet itt említünk, átlagos, azaz lehetnek ettől kisebb-nagyobb eltérések, még teljesen egészséges szemek esetén is.

Ennek és még több más - később meghatározott - adatnak alapján tervezhetünk egy optimális, az emberi szemhez legjobban illeszkedő képernyőt. Ennek két szomszédos képpontja is μ látószög alatt látszik. Fél méter átlagos nézési távolságot feltételezve ez azt jelenti, hogy két szomszédos képpont - illetve középpontjaik - távolsága kb. 0,0942 mm ~ 0,1 mm. Első közelítésként azt mondhatjuk, hogy 50 millió pontból álljon a képernyő, de - mint később látjuk - a látótér szabálytalan alakja miatt ezt az első közelítést módosítjuk.

Érdemes megvizsgálni a jelenlegi gyakorlatban használt grafikus képernyőket. Megállapíthatjuk, hogy igen távol esnek a fent kapott értékektől. A jelenleg használt legnagyobb felbontású képernyők általában nem nagyobbak 1 Mbyte -nál, azaz felbontásuk 1000 x 1000 képpont, ezt ugyanennyi byte-on tárolják. Egy byte értéke 0-tól 255-ig terjed, azaz 256 színű egy ilyen képernyő. De nálunk még ez is ritkaság, a 16 Kbyte-os képernyők a gyakoriak. Ez azt jelenti, hogy 64000 (azaz 320 x 200) vagy 128000 (640 x 200) képpontosak, 4 illetve 2 (háttér és egy másik) színűek. Itt 4 illetve 8 képpontot ír le egy byte. (minderről bővebbet az 5.2. részben. Azt máris megállapíthatjuk, hogy lényegesen le vannak még maradva a számítógépek display-jei a szemhez képest.

A fent említett S_T pont, másnéven fovea centralis vagy sárgafolt területén van csak éleslátás, itt a retina elvékonyodott. Annak a pontnak a képe, amelyre nézünk - jelöljük ezt S -sel - , a fovea centralis-ra kerül, ezért a fovea centralis környezete lesz a matematikai modellben a képsík, maga a fovea centralis pedig a képsíkon felvett koordináta-rendszer középpontja. A retina többi részén nincs éleslátás, de a látótér maga a sárgafoltnál lényegesen nagyobb, vízszintes irányban halánték felé 90°, az orr felé 50°, felfelé 60°, lefelé 70°. Ennél pontosabban mutatja ezt a 2. ábra, amely mindkét szem látóterét szemlélteti. A szemmel, mint középponttal vegyünk fel előre, az axis opticus -szal, mint főtengellyel egy félgömböt, melynek felületére rajzoljuk be a látótér határát. A kapott felületet vetítsük a félgömb alapsíkjára. Így keletkezik a 2. ábrán is látható, az emberi szem látótereit szemléltető kép is. Vizsgáljuk meg azt a kérdést, hogy a látótérhez legjobban illeszkedő téglalaphoz milyen módon juthatunk, másként szólva a látóteret téglalappal akarjuk közelíteni, a lehető legjobban. Ez azért fontos kérdés, mert ez is elősegíti egy képernyő optimális tervezését.



2. ábra: A két szem látóterének sémája, jobbra dől vonalkázás a jobb szem, a balra dőlt a bal szem látóterét szemlélteti.

Többféleképpen is megközelíthetjük ezt a kérdést. Ha matematikailag, akkor kiszámítjuk a látótérbe rajzolható legnagyobb területű téglalapot. Azt tapasztaljuk, hogy e téglalap két oldalának aránya jó közelítéssel 1:2, ettől - egészséges szem esetén - kisebb egyéni eltérés lehet. Mivel azonban a látótér különböző részein különböző a látás élessége, érzékenysége, a matematikai módszer itt csak irányadó lehet. A látótérnek eme tulajdonságait is figyelembevéve, ezt az arányt finomíthatjuk. Igen sok kísérlet, mérés, számítás eredményeképpen határozták meg az átlagos emberi szemre ezt az igen fontos arányt, amely 1 : 2.55 -nek adódott. Már évtizedek óta figyelembe is veszik ezt a tényt a szélesvásznú filmek készítésénél. A szélesvásznú film - melynek megvalósítása Henri Chrétien francia tanár nevéhez fűződik - egy képkockájának, mint téglalapnak, az oldalai aránya 1 : 2.55 egységesen. A számítógépek képernyői és a TV -képernyők tervezésekor egyaránt nem vették figyelembe az emberi szem ilyen irányú adottságait. A ma leginkább használatos display -eken (pl. IBM XT, AT, stb.) ugyanis ez az arány: 320 : 200.

Nézzük a 200 soros képernyők esetét. Már az előbb említett matematikai megközelítés szerint is a 400 : 200 arány lett volna helyesebb, azaz a használatos display -ek szélessége csak 80 % -a ennek. A helyesen tervezett, széles képernyő 510 oszlopos lenne, azaz az 510 : 200 arány jellemezné. Kb. 40 % -kal lenne szélesebb a mostaniaknál. De ez még mindig nem az optimális képernyő, mert kicsi a felbontása.

Az eddigi adatainkat összegezve eljuthatunk a szem szerkezete alapján megtervezett, optimális képernyőhöz. A függőleges és vízszintes oldalainak aránya: 1:2,55. Az most a kérdés, hogy az ilyen típusú képernyők közül melyik az, amelynek képe a legtöbb pontot foglalja el a retinán. Csak becslést adhatunk, hogy hány pontot jelent ez, az ideghártya pontjainak hányad részét. A becslés esetünkben elegendő is, hiszen most inkább e pontok nagyságrendje érdekel minket. Az említett arányú képernyőre nézve, az elfoglalhatja az ideghártyán lévő látósejtek felét. Mivel a két szem látótere nem esik teljesen egybe, lesznek olyan pontok is, amelyek csak az egyik szem ideghártyáján tűnnek fel, de többségük mindkettőn. Tehát legyen kb. feleannyi pont a display-en, mint ahány látósejt egy szemben., azaz legyen 25,5 millió. Így a képernyő méretei (vízszintes, illetve függőleges irányban): 7650 x 3000 pont. Így tényleg kb. 25,5 millió pont van a képernyőn és a függőleges, illetve a vízszintes oldalak aránya: 1:2,55. Ha a képernyő kissé homorú, az célszerű a szem szempontjából, de ha mértani pontosságot is elvárunk tőle, akkor jobb a teljesen sík képernyő.

Vessünk tehát egybe minden adatot, hogy megtervezhessük a szem szerkezetéhez leginkább közelálló, optimális képernyőt. A következő táblázatban összefoglaljuk ennek adatait, összehasonlítva a szemmel és a hagyományos 16 Kbyte-os monitorokkal. Néhány olyan adat is szerepel a táblázatban, amelyet az eddigiekben még nem említettünk, de ezek a többi ismeretében kiszámíthatók. Az ideális képernyő adatait figyelve, azt láthatjuk, hogy még igen sok fejlesztés szükséges ahhoz, hogy a display-ek minősége a szemünk paramétereihez mérhető legyen. A jelenlegi - még a táblázatban nem szereplő 'nagy' felbontású - grafikus monitoroknál lényegesen tökéletesebb a szemünk felépítése. A táblázatban minden adat közelítő érték. A hagyományos 16 Kbyte-os képernyő méretei a táblázatbelitől eltérhetnek. Több féle van forgalomban, a táblázatbeli adatok az eredeti IBM AT -re jellemzőek. A képernyőkön most nem a teljes képernyőt, hanem csak annak érdemi részét értjük, amelybe a háttér nem számít bele.

Táblázat a szem, a szem szerkezete alapján felépített optimális képernyő és a hagyományos (IBM AT) képernyő paramétereiről:

Paraméterek:	Szem:	Optimális képernyő:	Hagyományos képernyő:
Mérete: (mm) vízszintes:	28	760,3	245
Mérete: (mm) függőleges:	14	298,1	153
Pontok száma vízszintes:	10000	8064	320 (640)
Pontok száma függőleges:	5000	3162	200
Látószöge vízszintes:	140° (180°, a szemek együtt)	74° 29'	27° 32'
Látószöge függőleges:	130°	33° 12'	17° 25'
Területe:	4 cm^2	22,67 dm^2	3,75 dm^2
Képpontok száma:	50 millió	25498368 (25,5 millió)	64000 (vagy 128000)
Függ.-vízsz. arány	1 : 2	1 : 2,55	1 : 1,66
Szomszédos képpontok távolsága (mm)	$2,83 \cdot 10^{-3}$	$9,43 \cdot 10^{-2}$ ($\approx 0,1$)	0,766
Képpont területe:	$8 \cdot 10^{-6}$	$8,88 \cdot 10^{-3}$	0,59
Két szomszédos pont látószöge:	0,648'	0,648'	5,264'
Színek száma: (egyszerre:)	5 millió	5 millió	4

A táblázatban szereplő valamennyi adatot érintettük, vagy a többi adatból számítható, kivéve a színek számára vonatkozó információ. Erről azonban az alábbiakban bővebben is szólunk.

2.4. *Az ideghártya viszonya a fényhez*

A retina többi rétegét hagyjuk figyelmen kívül, a témánk szempontjából ezek nem érdekesek. Az ideghártyán keletkező kép tulajdonságai a fontosabbak. Az itt keletkező kép mozaikszerűen felbontott. A pálcikák a fény intenzitására, a fény-árnyék hatásokra, *a formákra* érzékenyek, a csapok pedig a színekre és a gyengébb fényben való látásra. Működik még egy kontraszt-erősítő hatás is a szemben. A megvilágított pont körüli sejtek gátolt állapotba kerülnek, tehát a kontrasztok jobban kiemelődnek. A csapok és a pálcikák eloszlása is különböző: a fovea centralis területén csak csapok vannak, ezért itt (és csak itt) van éleslátás.

A szem nagyon erős és gyenge fényben is lát, e képességét adaptációnak nevezzük. Szürkületben a fényre nagyon érzékeny, de színlátásra alkalmatlan pálcikák fogják fel az ingerületeket, erős fényben pedig a kevésbé fényérzékeny csapok. Ez az oka annak, hogy egy fényerősség-küszöb alatt a színeket nem tudjuk egymástól megkülönböztetni.

A szem érzékenységét szabályozó mechanizmust segíti a pupilla összeszűkülése illeteve tágulása is. A szem fényérzékenységére jellemző, hogy 3 km távolságból 0,05 mp-re felvillanó 1,5 candela erősségű fényt vesz észre, folyamatos fénynél ennél kisebb is elegendő. Az, hogy mennyi fény éri az ideghártyát, igen sok tényezőtől függ. Ezt szemlélteti a következő képlet:

$$E = B \cdot t \cdot A \cdot \cos(\beta)$$

Ez a képlet a fény erősségét lux-ban adja meg; a szimbólumok jelentése a következő:

B : Fényerősség az adott irányból (candela, 1 candela a fényerőssége az 1769 °C -os - ez a platina dermedési hőmérséklete - , 1/600000 m^2 -es fekete testnek, 1 atmoszféra nyomáson)

t : A szem átviteli együtthatója. A szem különböző közegein áthaladó fény hány százaléka érkezik el az ideghártyáig. A többi elnyelődik, vagy visszaverődik.

A : a pupilla területe (m^2)

β : a beeső fény és a pupilla síkja normálvektorának szöge

Einstein 1905-ben bevezette a foton fogalmát, amely a fényáram legkisebb egysége. Kutatásokat végzett a fény-kvantum elmélet terén is, többek között a fény szerkezetét vizsgálta. A retinával összefüggésben.

Az eddigiekből látható, hogy a szem magasabbrendű működésű, mint egy kamera, mivel nem csak lefényképezi azt, amit lát, hanem az alakzatok elemzésének, az alakfelismerésnek az első lépése is itt történik. A folyamat további lépéseit az agy valósítja meg.

2.5. Színek, tónusok, mozgás

A szem csupán a 390-780 nm -es, azaz 0,00039-0,00079 mm -es frekvenciatartományban lévő beeső sugarakat érzékeli, az ezen kívüleső hullámhosszú sugárzás a szem számára láthatatlan (pl. az infravörös: nagyobb, mint 780 nm és az ibolyántúli: kisebb, mint 390 nm). Ez azt jelenti, hogy az emberi szem az elektromágneses sugarak tartományából éppen egy oktávnyit lát. Frekvenciában kifejezve - kiszámítva azon tényből, hogy a hullámhossz és a frekvencia szorzata a fény esetén éppen a fénysebesség - ez azt jelenti, hogy a piros: $3{,}84 \cdot 10$ Hz, a kék: $7{,}68 \cdot 10$ Hz körüliek. A spektrális érzékenység maximuma 555 nm körül van, azaz ez az a frekvencia, amelyre a legjobban érzékeny a szemünk. Ezért egy világoszöld test különösen világosnak tűnik. Oldalról pedig a sárga színt vesszük leghamarább észre, majd a kéket és vöröset, legkésőbb a zöldet. A szemnek ezek a perifériális területei a mozgásra is igen érzékenyek. A retinában csak három színérzékelő elem van: vörös, zöld, kék. Ezek a különböző színekre különbözőképpen érzékenyek. A többi szín ezek keveredésével áll elő, melyeknek külön szabályai vannak. Nézzünk egy táblázatot a színekről, amelyeket hullámhosszuk alapján a következőképpen csoportosítjuk:

szín:	ibolya:	kék:	zöld:	sárga:	narancs:	vörös:
hullámhossz (nm):	390 - - 436	436 - - 500	500 - - 560	560 - - 595	595 - - 627	627 - - 780

Természetesen a táblázatbeli értékeket nem lehet mereven kezelni, ugyanis a színek lágy átmenetet képezve mennek át egymásba, ezeknek az áthajlásoknak a színe a két szomszédos szín között áll.

A színeket különböző mértékben keverve kb. 17000 féle - a szemünk által is különbözőnek ítélt - színt tudunk maximum előállítani. Minden színnek kb. 300 árnyalatot adhatunk (a legvilágosabbtól a legsötétebbig), amelyet még különbözőnek látunk. Mindez azt jelenti, hogy kb. 5 millió színárnyalatot tud a szemünk megkülönböztetni. Így a szem paramétereit maradéktalanul megvalósító képernyővel is ennyi színt kell tudnunk előállítani. Talán ez a legnehezebb feladat az ezen elméleti képernyő paramétereinek megvalósításai közül. A színekkel, a szem színlátásával kapcsolatos vizsgálataikért 1967-ben Granit, Haldan Hartline és Wald kapott megosztott Nobel-díjat.

Ezért a tónusok, a színek telítettsége, a kontrasztok igen sok információt hordoznak magukban. Ezek nemcsak a test tulajdonságaitól, hanem annak helyzetétől, alakjától, a megvilágítás erősségétől, helyétől, a fény-árnyék hatásoktól, a testeknek a térben elfoglalt helyzetétől és egymásra hatásuktól, a szem térbeli elhelyezkedésétől is nagymértékben függenek. Mindezen tényezők együttes hatására alakítja ki a szem a végleges képet a testekről.

Ezek a hatások matematikailag szintén megvizsgálhatók, egzakt képletekké formálhatók, vizsgálatuk éppoly elengedhetetlen része a látás elméletének, mint a perspektív leképezések értelmezése.

Egy kép "lefényképezéséhez" kb. 1/6 másodperc szükséges, azaz másodpercenként 6-nál több állóképet vetítve, azt mozgóképnek látjuk. Ezen alapul a filmkockák egybeolvasztása mozgóképpé a film levetítésekor. Tehát egy testet sűrűn lefényképezve, a filmkockákat egymás után vetítve, folyamatos mozgást látunk. Ezeket az állapotokat a számítógépben tárolva, nem létező, csak kiszámított testről is mozgó képet kaphatunk.

Az eddigiekben minden - a témánkhoz kapcsolódóan - lényeges kérdést megvizsgáltunk. A következő részekben ehhez kapcsolódóan felépítjük a matematikai modellt.

3. A LÁTÁS MATEMATIKAI MODELLJE (1):
A három dimenziós objektumok perspektív leképezésének elmélete.

3.1. *A testek felszínének közelítése:*

Az átlátszó (áttetsző) testeket kivéve minden testnek csupán a *felszínét látjuk*; nem nézhetünk bele az objektumok belsejébe. Legyen egy általában vett test felszínének jele: F, közelítsük a felszínt poligonokkal. Ez azt is jelenti, hogy a testet egy poliéderrel közelítjük. Világos, hogy mennél több sokszöggel közelítjük a test felszínét, ez a közelítés annál pontosabb lesz. Ezt a tényt fogalmazzuk meg matematikailag pontosított formában is.

A test felszínét a $p_1, p_2, \ldots, p_n$ poligonokkal közelítjük, melyeknek területe rendre: $f_1, f_2, \ldots, f_n$,azaz:

$$(3.1.1.) \qquad F \backsim F_{\text{közelítő}} = f_1 + f_2 + \ldots + f_n = \sum_{i=1}^{n} f_i$$

Ha minden poligon területe tart a nullához, a poligonok száma pedig végtelenhez, akkor a (3.1.1.)-beli felszín-közelítés tart a tényleges felszínhez:

$$(3.1.2.) \qquad F := \lim_{\substack{n \to \infty \\ f_i \to 0}} \sum_{i=1}^{n} f_i$$

Amennyiben meg akarjuk szerkeszteni a fent említett test három dimenziós, transzformált képét, akkor a módszerünk a következő. Transzformáljuk egy poligon minden csúcsát, majd a kapott, transzformált csúcspontokat összekötve megkapjuk a poligon transzformált képét. Minden poligonnal - mely a test felszínét alkotja - végrehajtva ezt a műveletet, megszerkesztettük a test három dimenziós, perspektív képét. Ez a transzformáció még "nyers", mivel ezt a képet egyrészt láthatósági szempontból - azaz mely oldala, felszíni poligonja látható vagy takart és milyen mértékben - elemezni kell, másrészt meg kell vizsgálni a fény-árnyék hatásokat és a tónusokat is. Tehát minél nagyobb felbontásban, minél több poligonnal közelítjük a test felszínét, annál pontosabban tudjuk ábrázolni a test térbeli képét. Némely test esetén elegendő kevés poligonnal közelítenünk, pl. egy kocka felszínét hat négyzettel pontosan leírhatjuk. A későbbiekben is a fenti módszerrel dolgozunk.

3.2. *Képsík-definiálás*

Vezessük be a következő jelöléseket: (minden térbeli pontot három koordinátával, x,y,z-koordinátákkal adunk meg).

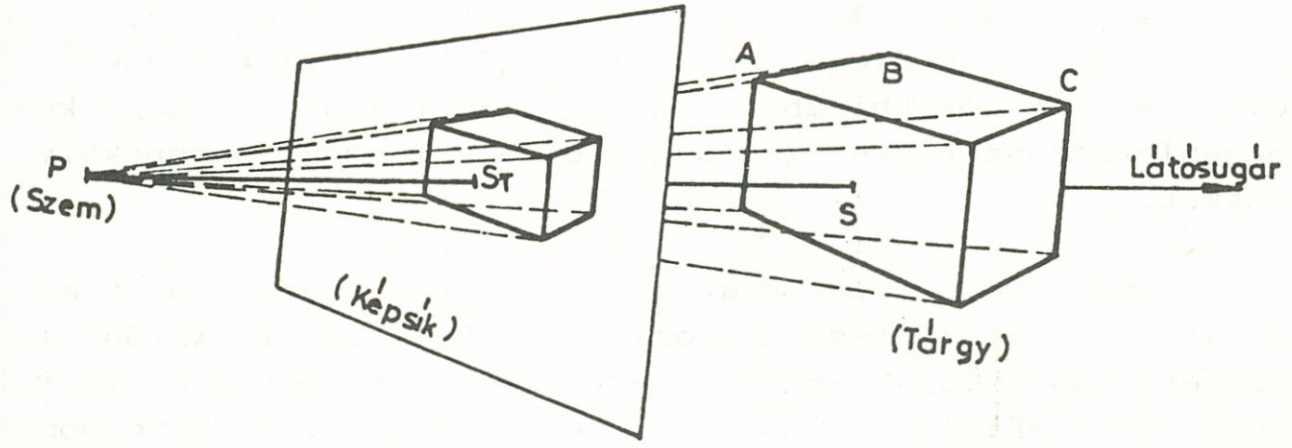
$A(a_1,a_2,a_3)$, $B(b_1,b_2,b_3)$, $C(c_1,c_2,c_3)$: egy tetszőleges, a test felszínét közelítő poligon három csúcspontja.

$P(p_1,p_2,p_3)$: A szem helye a térben, továbbiakban: szem, vagy szempont, később: az optikai középpont.

$S(b_1,b_2,b_3)$: A tárgy (vagy a tér) egy tetszőleges pontja, amelyet nézünk, a továbbiakban: tárgypont.

$S_T(s_1^T,s_2^T)$: Képpont, azaz S transzformációs képe, mivel síkban ábrázoljuk, ezért csak két dimenziós.

$H(h_1,h_2,h_3)$: A fényforrás helye a térben.



3. ábra

Nevezzük ezentúl a szem (P) és a nézett pont (tárgypont:S) által meghatározott, rajtuk átmenő, P pontból kiinduló félegyenest látósugárnak, mely a biológiai leírásban axis opticus néven szerepelt. Vegyünk fel egy üveglapot a szem és a tárgy között, a látósugárra merőlegesen. Ezen üveglapon jelöljük ki az S_t pontot, ott, ahol a látósugár metszi az üveglapot. A test tetszőleges Q pontjához rendeljük hozzá a Q_t pontot, amely legyen a PQ -szakasz és az üveglap metszéspontja. A test minden Q pontjához rendeljük így hozzá a Q_t pontot, azaz a test minden pontjához hozzárendeltük *egyértelműen* az üveglap egy pontját. Az üveglapon kapott kép kapjon olyan színt és tónust, amilyennek a test P -ből látszik, így tehát P-ből nézve a test képe és az üveglapon látott (rajzolt) kép megegyezik. Nevezzük ezután a szemléltetés céljából leírt üveglapot *képsík*nak. A leírt elvet először Leonardo alkalmazta a már említett, XV. századi szerkezet, a Camera Obscura megalkotásakor, azaz ő ismerte fel képsík fogalmát. A leírt elven működik a fényképezőgép és sok más optikai eszköz, valamint a szemünk is.

A leírt transzformáció a három dimenziós tér egy részhalmazáról a (testről) egy kétdimenziós tér (képsík) részhalmazára képez le. Ez a leképezés *odafelé egyértelmű*, mert minden pontnak van egyértelmű képe a képsíkon. *Visszafelé nem egyértelmű*, mert a képsík egy pontjához végtelen sok (kontinuum számosságú) pont tartozik, a ponton és a P ponton átmenő egyenes pontjai. Matematikailag fogalmazva a leírt transzformáció egy képelemének ősképe egy egyenes, tehát többelemű halmaz, de az értelmezési tartomány minden pontjához egyértelműen hozzárendeltük az értékkészlet egy elemét, tehát a leírt transzformáció *injektív*.

Nevezzük ezután a leírt transzformációt *perspektív leképezésnek*, *képsík-transzformációnak*, *jelöljük* t *-vel*. Matematikailag: $t : Q \rightarrow K$ injektív leképezés, ahol: Q: a vizsgált test, K: a képsík -tartomány (a test képe), $\mathbb{R}^3$: a három dimenziós tér, $\mathbb{R}^2$: a képsík, $Q \subseteq \mathbb{R}^3$ és $K \subseteq \mathbb{R}^2$. Látható, hogy a szem, a képsík és a tárgy igen sokféleképpen helyezkedhetnek el a térben. Ebből adódik - mint a későbbiekben ezt látni is fogjuk - , hogy különböző egyenletrendszerekkel írhatjuk le a különböző perspektív leképezéseket.

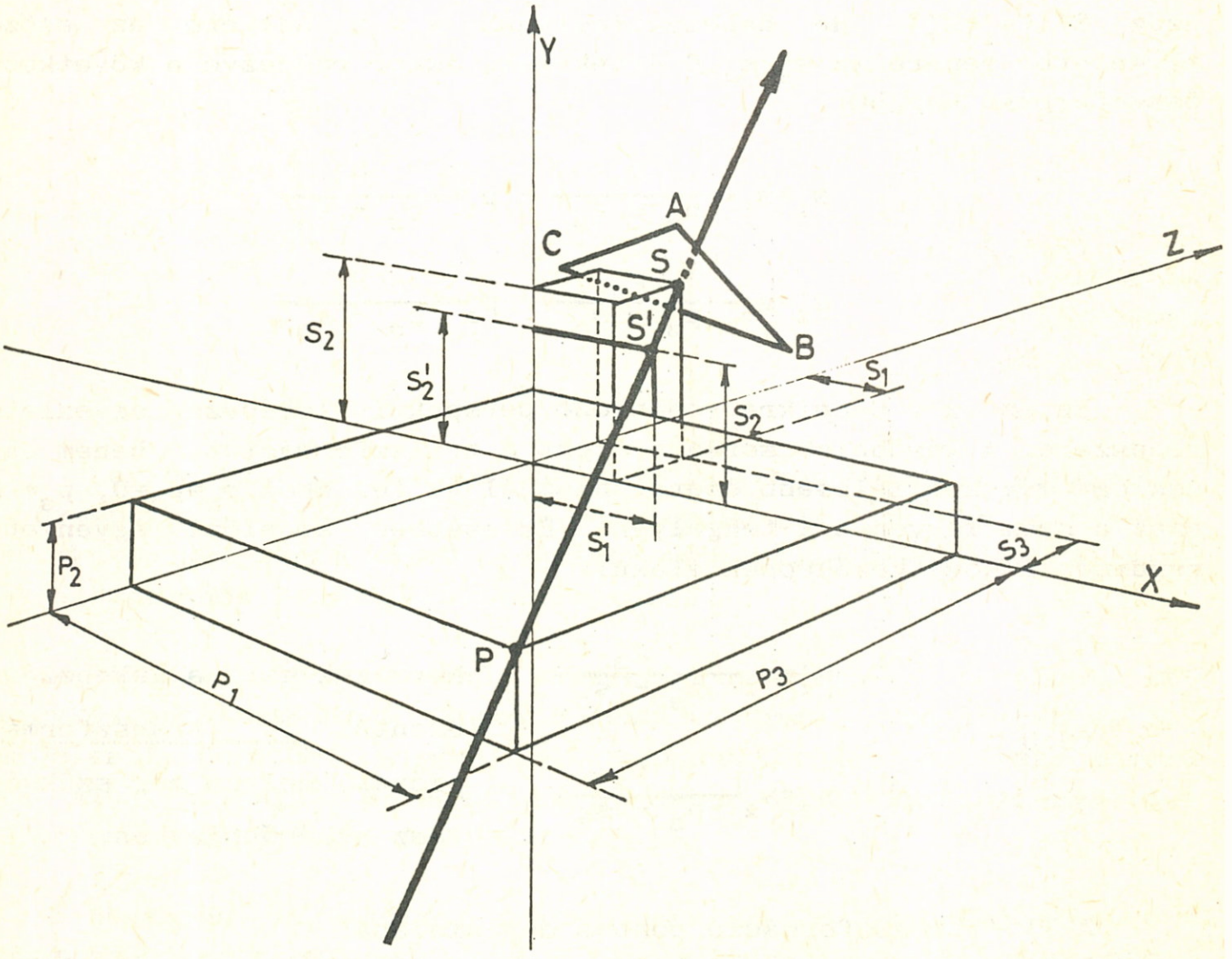
Visszatérve a *biológiai modell*hez, a következőket mondhatjuk: a szem esetén kissé módosítanunk kell az 1. ábrán látható helyzetet. Leszűkítjük az eddig értelmezett szemet a biológiai modellnél már definiált P pontra, azaz az optikai középpontra, a képsíkot pedig a fovea centralis környezetére. Ez nem sík terület, de egy síkkal közelítjük, amely a fovea centralis -ban érinti a szemgolyót.

A közelítés annál pontosabb, mennél kisebb környezetét vizsgáljuk a fovea centralis-nak. Jelöljük ezután ezt a pontot a biológiai modellel összhangban S_T-vel. A fent leírt képsík most nem a tárgy és a szem közé esik, hanem a P pont - az értelmezett szem leszűkítése - túlsó oldalára.

Bármely képsíkot használjuk, a test transzformált képei egymáshoz hasonlóak lesznek. Ha a képsík az S (nézett pont) és P pont közé esik, akkor egyenes állású; ha P-nek S-sel ellenkező oldalára, akkor fordított állású képet kapunk. Így van ez a szem esetén is, ahol fordított állású kép vetül az retinára. A szem képsík-transzformációját a többi (t) képsík-transzformációtól megkülönböztetve nevezzük T -transzformációnak. S pont T -transzformáltja tehát S_T, a fovea centralis.

Ebben a részben meghatározzuk t, majd a T transzformációt. Elegendő a t-transzformációk közül valamelyiket meghatároznunk, a hasonlóság miatt ebből már könnyen meghatározhatjuk bármelyik képsíkra vonatkozó, így a T -transzformációt is. Ezzel részben leírjuk: hogyan látunk. A pontos matematikai modellnek ez az első részrésze, mert ahhoz még a takarások, fény-árnyék, tónusok elmélete is hozzá tartozik.

3.3. *Perspektív leképezés az X-Y síkra*



4. ábra

A most leírt speciális transzformációra - mely a perspektivikus leképezés speciális esete - a későbbiekben lesz szükség, amikor az általános képsík-transzformációt definiáljuk. Ezt a speciális transzformációt szemlélteti a 4. ábra. Legyen az eddigiek szerint P:szempont, S:tárgypont, $P,S \in \mathbb{R}^3$, legyen a képsík az (x,y) sík. A transzformáció definiálásához írjuk fel a P,S pontokon átmenő egyenes egyenletét:

$$(3.3.1.) \qquad \frac{x - p_1}{s_1 - p_1} = \frac{y - p_2}{s_2 - p_2} = \frac{z - p_3}{s_3 - p_3}$$

Legyen S': a $\overline{PS}$ -szakaszon (illetve a meghosszabbításán) átmenő egyenes és az X-Y sík döféspontja. Mivel $s'_3=0$, ezért $S' \in \mathbb{R}^2$ azaz $S':(s'_1,s'_2)$. Ha behelyettesítjük x,y,z helyére az előző egyenletbe rendre az $s'_1,s'_2,0$ értékeket, akkor rendezve a következő összefüggést kapjuk:

$$(3.3.2.) \qquad \begin{aligned} s'_1 &= s_1 \frac{p_3}{p_3 - s_3} - p_1 \frac{p_3}{p_3 - s_3} \\ s'_2 &= s_2 \frac{p_3}{p_3 - s_3} - p_2 \frac{p_3}{p_3 - s_3} \end{aligned}$$

Ez az x-y képsíkra vonatkozó perspektív leképezés egyenletrendszere. A későbbiek során nem ezt a transzformációt, hanem ennek egy még speciálisabb esetét használjuk fel,ahol $p_1=p_2=0$, $p_3 \neq 0$ azaz a szem legyen a z-tengelyen. Ez esetben az előző egyenletrendszer a következőképpen alakul:

$$(3.3.3.) \qquad \begin{aligned} s'_1 &= s_1 \frac{p_3}{p_3 - s_3} \\ s'_2 &= s_2 \frac{p_3}{p_3 - s_3} \end{aligned}$$

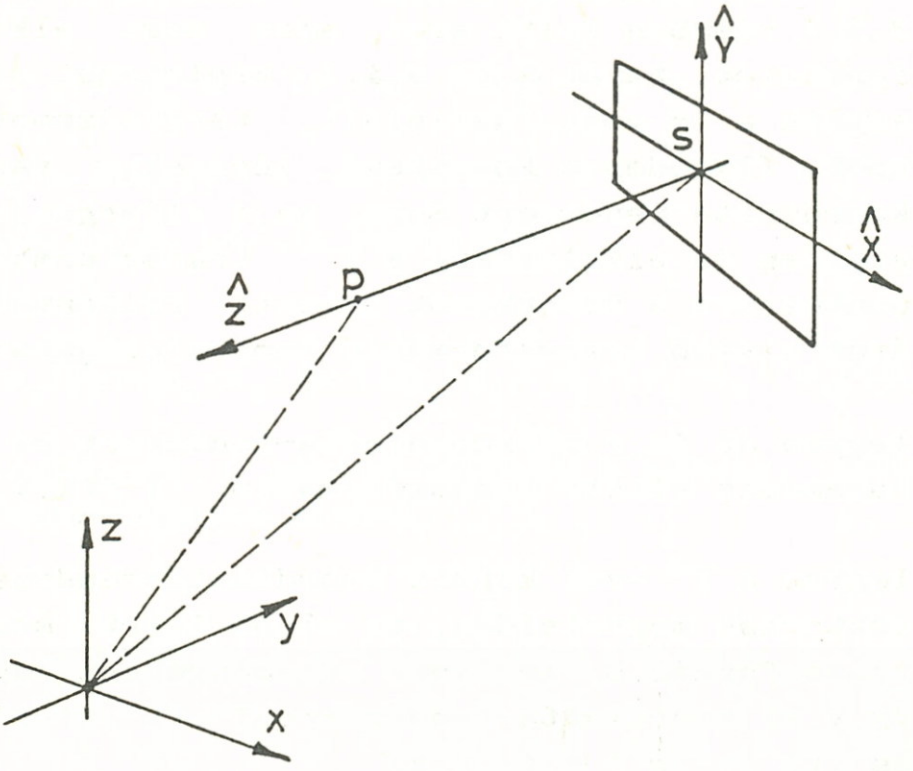
Nevezzük ezt a leképezést ezentúl T''' transzformációnak, melyre még szükség lesz a későbbiekben.

A T''' transzformáció pontos definiálása:
$T''':(x,y,z) \rightarrow (x''',y''')$, $T''' \in \mathbb{R}^3 \rightarrow \mathbb{R}^2$, ahol (3.3.3.) alapján:

$$(3.3.4.) \qquad \boxed{x''' = x \frac{p_3}{p_3 - z} \qquad y''' = y \frac{p_3}{p_3 - z}}$$

3.4. Leképezés tárgyközpontú képsíkra

3.4.1. Az alkalmazott transzformáció lényege, első közelítésben



P: szempont
S: tárgypont
ahol: P ≠ S

5. ábra

Legyen (x,y,z) jobbsodrású koordinátarendszer, S tárgypont, P szempont. A képsík $\overrightarrow{SP}$-re merőleges. Az $\overrightarrow{SP}$-re merőleges képsíkokon keletkező képek egymáshoz mértanilag hasonlóak, arányosak a képsíknak P-től mért távolságával.

Először az S-en átmenő képsíkot használjuk, nevezzük ezt tárgyközpontú képsíknak. Ennek S lesz az origója, de egyben ez azt is jelenti, hogy $S=S_t$,tehát S-et a transzformáció önmagára képezi le. Általánosságban is igaz, hogy minden képsík-transzformáció a képsíkon lévő pontokat helyükön hagyja.

Nevezzük az említett transzformációt - megkülönböztetve a többi képsík-transzformációtól - tárgyközpontú képsík-transzformációnak. Eddig t-vel jelöltünk egy képsík-transzformációt általában, legyen ezután 't' a tárgyközpontú képsík-transzformáció jele. Ahhoz, hogy a 't' transzformációt pontosan definiálhassuk, *áttérünk egy másik koordináta-rendszerre*, amelynek tengelyeit a következő módon állítjuk elő:

$\hat{z}$: $\overrightarrow{SP}$ irányú egyenes.

$\hat{x},\hat{y}$: $\overrightarrow{SP}$-re és egymásra merőleges, úgy, hogy $(\hat{x},\hat{y},\hat{z})$ jobbsodrású legyen, ezenkívül $\hat{x}$ párhuzamos az (x,y) síkkal. Ebben a koordináta-rendszerben ábrázoljuk a testet. Észrevehetjük, hogy előállt a (3.3.) részben leírt eset, azaz most alkalmazhatjuk a T''' transzformációt, tehát a (3.3.4.) -képleteket alkalmazva megkapjuk a testnek a - most definiált, tárgyközpontú - képsíkon keletkező képét. Ilymódon a következő lépésekre van szükségünk: eltoljuk a koordináta-rendszert S-be, majd elforgatjuk a látósugárnak megfelelően és alkalmazzuk a T''' transzformációt. Ez matematikailag pontosítva a következőt jelenti: felbontjuk t -transzformációt három kisebb transzformáció kompozíciójára:

T' : legyen az (x,y,z) koordináta-rendszert az S pontba párhuzamosan eltoló transzformáció: $T':(x,y,z)\rightarrow(x',y'z')$

T" : legyen a T' -vel kapott koordináta-rendszert az $\overrightarrow{SP}$ vektornak megfelelően, az előbb leírt módon elforgató transzformáció, amelynek eredményeként áttérhetünk az $(\hat{x},\hat{y},\hat{z})$ koordináta-rendszerre: $T':(x',y',z')\rightarrow(x'',y'',z'')$ és $(x'',y'',z'') \equiv (\hat{x},\hat{y},\hat{z})$

T''': a (3.3.) részben definiált perspektív transzformáció

A t-transzformációt tehát három egyszerűbb transzformációból építjük fel. A T', T" transzformációk kompozícióját nevezzük ezentúl $\hat{T}$ transzformációnak, amely tehát megvalósítja az $(\hat{x},\hat{y},\hat{z})$ koordináta-rendszerre való áttérést. Tehát:

(3.4.1.) $$t = T'''(T''(T')) = T'''(\hat{T}) \quad \text{,ahol:} \quad \hat{T} = T''(T')$$

Ez a transzformáció képlete, melyet a következő részekben pontosítunk, innen már csak egy lépés a keresett, a szem helyzetétől függő perspektív transzformáció kiszámítása.

$T',T'',\hat{T} \in \mathbb{R}^3 \rightarrow \mathbb{R}^3$,bijektív, önmagára képezi a testet, de koordinátái megváltoznak, $t,T''' \in \mathbb{R}^3 \rightarrow \mathbb{R}^2$,injektív leképezések, ahol: $T':Q \rightarrow Q'$, $T'':Q' \rightarrow Q''$, $\hat{T}:Q \rightarrow \hat{Q}$, $T''':Q'' \rightarrow K$, $t:Q \rightarrow K$, ahol: Q: a vizsgált test, K: a képsík-tartomány (a test képe), $\mathbb{R}^3$: a három dimenziós tér, $\mathbb{R}^2$: a képsík, $Q \subseteq \mathbb{R}^3$ és $K \subseteq \mathbb{R}^2$, $Q',Q'',\hat{Q}$: a test koordináta-transzformáltjai.

3.4.2. *A tárgyponton átmenő képsík egyenlete*

Adott ponton átmenő, adott irányvektorra merőleges sík egyenletéből származtatjuk. Ha pl. adott az $S = (s_1,s_2,s_3)$ pont és a $V = (v_1,v_2,v_3)$ irányvektor, akkor a síkegyenlet:

$$v_1 \cdot x + v_2 \cdot y + v_3 \cdot z - (v_1 \cdot s_1 + v_2 \cdot s_2 + v_3 \cdot s_3) = 0$$

Ha a képsík átmegy S -en, akkor: $V := \overrightarrow{SP} = \overrightarrow{P} - \overrightarrow{S}$,ekkor az előző részben leírt képsík egyenlete:

(3.4.2.1.)

$$(p_1-s_1)\cdot x+(p_2-s_2)\cdot y+(p_3-s_3)\cdot z-[(p_1-s_1)\cdot s_1+(p_2-s_2)\cdot s_2+(p_3-s_3)\cdot s_3]=0$$

Ha a tetszőleges Q ponton megy át a képsík, akkor az előző egyenlet a következőképpen alakul:

(3.4.2.2.)

$$(p_1-q_1)\cdot x+(p_2-q_2)\cdot y+(p_3-q_3)\cdot z-[(p_1-q_1)\cdot q_1+(p_2-q_2)\cdot q_2+(p_3-q_3)\cdot q_3]=0$$

Tehát a (3.4.2.1.) és a (3.4.2.2.) adja a speciális és az általános esetekben a képsíkok egyenletét. Ezekre az egyenletekre a későbbiekben még szükségünk lesz.

3.4.3. *T' koordináta-transzformáció: párhuzamos eltolás*

Ez a transzformáció az (x,y,z) koordináta-rendszert eltolja párhuzamosan az S pontba: $T':(x,y,z) \rightarrow (x',y',z')$. Az ily módon kapott megváltozott koordináták értékeit az alábbi egyszerű összefüggésekkel kapjuk:

(3.4.3.1.) $x' := x-s_1 \quad y' := y-s_2 \quad z' := z-s_3$

Felírhatjuk a transzformáció mátrixát is - jelöljük szintén T'-vel - a következő egyenlet alapján: $(x',y',z') := (x,y,z)T'$

(3.4.3.2.)
$$(x',y',z') := (x,y,z)\begin{bmatrix} \frac{x-s_1}{x} & 0 & 0 \\ 0 & \frac{y-s_2}{y} & 0 \\ 0 & 0 & \frac{z-s_3}{z} \end{bmatrix}$$

Az ilyen típusú mátrix a gépi feldolgozás esetén előnytelen, mivel elemei között van olyan, amely függ az aktuális pont valamely koordinátájától. Így minden pontra újra kell számítani a mátrixot, ez aránytalanul nagy gépidőfelhasználást jelentene. Olyan mátrixokra van szükségünk, amelyeket az egy vagy több testre vonatkozó számítások kezdetén rögzíthetünk, majd a továbbiakban számolhatunk velük. Ezért tehát inkább a (3.4.3.1.) egyenletrendszert alkalmazzuk, azaz a transzformációnak feleltessünk meg egy vektort:

(3.4.3.4.) $(x',y',z') := (x,y,z) - (s_1,s_2,s_3)$,azaz:

(3.4.3.5.) $(x',y',z') := (x,y,z) - T'$,vagyis:

$T' := (s_1,s_2,s_3) = S$
$Q' := Q - S$

Ahol: $S=(s_1,s_2,s_3)$ a szem által nézett pont (tárgypont) és $Q=(q_1,q_2,q_3)$ tetszőleges pont a térben.

3.4.4. *T" koordináta-transzformáció: elforgatás $\overrightarrow{SP}$ szerint*

Ez a transzformáció az (x',y',z') koordináta-rendszert elforgatja $\overrightarrow{SP}$ szerint: $T'':(x',y',z') \rightarrow (x'',y'',z'')$; oly módon, hogy hogy a *látott kép egyenes állású legyen.* Ennek pontosabb meghatározásához szükséges mindaz, amely a (3.4.4.1.)-(3.4.4.4.) részekben található, ahol először meghatározzuk az (x",y",z") koordináta-rendszer egy bázisát, majd ennek segítségével definiáljuk a T" transzformáció mátrixát.

3.4.4.1. *Bázisvektorok keresése (x",y",z")-ben*

Teljes ortogonális vektorrendszert keresünk, amelynek három eleme - három bázisvektor - határozza meg az (x",y",z") koordinátákat. A térnek végtelen sok bázisa lehet, azonban elegendő ebből egyet meghatározni, a T" transzformáció definiálásához, sőt nem szükséges, hogy a bázis ortonormált legyen, elegendő, ha ortogonális. Jelöljük a bázisvektorokat a következőképpen: $\overrightarrow{v}_x$; $\overrightarrow{v}_y$; $\overrightarrow{v}_z$ Mindehhez egy speciális ortogonalizációs eljárást alkalmazunk, amelyben meghatározzuk a bázist, az alábbiak szerint:

z" (illetve: $\vec{v}_z$) konstrukciója:

Egy z" -nek megfelelő bázisvektor kézenfekvően $\vec{SP}$, amely a következő alakú: $\vec{SP} = \vec{P} - \vec{S} = (p_1-s_1,\ p_2-s_2,\ p_3-s_3)$. Tehát z" bázisvektora:

$$\vec{v}_z := (p_1-s_1,\ p_2-s_2,\ p_3-s_3)$$

x" (illetve: $\vec{v}_x$) konstrukciója:

A $\vec{v}_z$ vektorra merőleges, S-en átmenő sík nem más, mint az (x",y") sík. Ennek fel is írhatnánk az egyenletét, hasonlóan, mint a (3.4.2.) esetben. Azonban most más úton haladjunk a célunk felé, először az (x",y") síkot elmetsszük a $z=s_3$ egyenletű - az (x,y) síkkal párhuzamos, ('vízszintes') - síkkal. Ezt megtehetjük, ha az (x",y") sík nem párhuzamos az (x,y) síkkal, mivel két sík vagy párhuzamos, vagy egy egyenesben metszi egymást. A két sík párhuzamossága egyenértékű azzal, hogy z" párhuzamos z-vel. Ez azt jelenti, hogy $v_1=v_2=0$ és $v_3\neq 0$, azaz $p_1=s_1$, $p_2=s_2$ és $p_3\neq s_3$. Esetünkben tehát az, hogy z nem párhuzamos z" -vel, egyenértékű azzal, hogy $p_1\neq s_1$ vagy $p_2\neq s_2$. Először ezzel az esettel foglalkozunk, később pedig a párhuzamos esetre még visszatérünk. A $z=s_3$ egyenletű sík az (x,y) síkkal párhuzamos, S -en átmenő sík, ennek és a képsíknak a metszete egy egyenes (a párhuzamos esetet kivéve, amelyet egyelőre figyelmen kívül hagytunk). Ennek a metszésvonalnak az egyenlete a képsíknak a (3.4.2.1.) egyenletéből származtatható, amelybe a $z=s_3$ helyettesítést elvégezve, a kapott egyenlet teljesíti mindkét síkra vonatkozó egyenletet, tehát a síkok metszésvonalának (mint egyenesnek) az egyenletét kapjuk, amely a következő alakú lesz:

(3.4.4.1.1.) $$(p_1-s_1)\cdot x+(p_2-s_2)\cdot y-[(p_1-s_1)\cdot s_1+(p_2-s_2)\cdot s_2]=0$$

Legyen egy Q-val jelölt pont ezen az egyenesen, amelyet nevezzünk ezentúl x"-egyenesnek. A 'Q' pont koordinátáit úgy konstruáljuk meg, hogy először $q_1:=0$, azaz Q-nak x-irányú koordinátája legyen :0, ekkor az x"-egyenes előbb felírt egyenlete alapján (abban x=0 helyettesítéssel) y -ra a következő összefüggést kaphatjuk:

$$y = q_2 = \frac{p_1 - s_1}{p_2 - s_2}\, s_1 + s_2$$

Itt: $p_2 \neq s_2$, ez erősebb feltétel, mint az eddigi: $p_1 \neq s_1$ vagy $p_2 \neq s_2$, azaz $v_1 \neq 0$ vagy $v_2 \neq 0$, a párhuzamosságot kizáró feltétel. Ez azt jelenti, hogy $\mathbb{R}^3$-nak egy leszűkítésére értelmezzük a feladatot, olyan $(v_1, v_2, v_3) \in \mathbb{R}^3$ számhármasra, amelyek a szemtárgy vektor koordinátáit alkotva teljesítik a feltételt. Mivel a teljességre törekedünk, ezért a diszkussziót tartalmazó következő részben a teljes $\mathbb{R}^3$ -ra értelmezzük a feladatot. A Q pont a z" egyenesen van, amelynek minden pontjára: $z=s_3$, tehát $q_3=s_3$ teljesül. Most már felírhatjuk a Q pont által meghatározott vektor koordinátáit:

$$\overrightarrow{Q}: \left(0, \frac{p_1-s_1}{p_2-s_2}\, s_1+s_2, s_3\right)$$

$$\text{így: } \overrightarrow{SQ} = \overrightarrow{Q} - \overrightarrow{S} = \left(-s_1, \frac{p_1-s_1}{p_2-s_2}\, s_1, 0\right)$$

Ezután minden koordinátát megszorozva $\frac{p_2-s_2}{s_1}$ -gyel, az előzővel azonos irányú vektort kapunk. Legyen ez az x"-koordinátát meghatározó bázisvektor, amely esetén már a $p_2 \neq s_2$ feltétel sem szükséges (bővebben erről: később):

$$\boxed{\overrightarrow{v_x} := (-(p_2-s_2),\ p_1-s_1,\ 0)}$$

Eddig ismerjük az x",z" koordinátákat meghatározó bázisvektorokat, ezekből könnyen kiszámíthatunk egy y" -höz tartozó bázisvektort is. Felhasználjuk, hogy a bázisvektorok skaláris szorzata páronként nulla. Alkalmazzuk a következő *jelöléseket*: $v_i := p_i - s_i$ ahol i:=1, 2, 3. Ilymódon x",z" bázisvektorai: $(-v_2, v_1, 0)$ és (v_1, v_2, v_3).

Írjuk fel most az x",y", majd a z",y" bázisvektorainak skaláris szorzatát, ez két egyenletből álló háromismeretlenes egyenletrendszer. Ismeretlenek a y" bázisvektorainak koordinátái, jelöljük ezeket most x,y,z -vel. Az egyenletrendszer:

$$-v_2 \cdot x + v_1 \cdot y = 0$$

$$v_1 \cdot x + v_2 \cdot y + v_3 \cdot z = 0$$

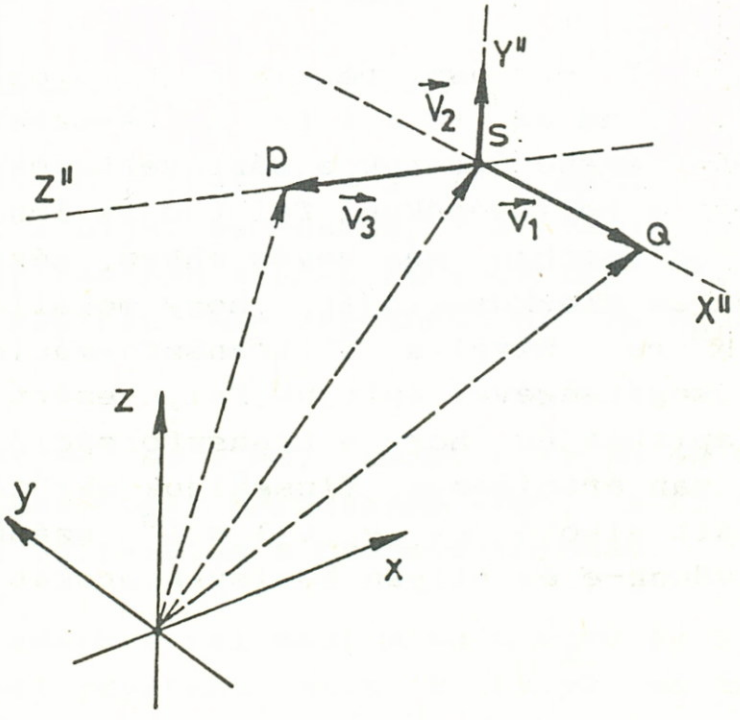
Ennek egy lehetséges megoldása: $(x,y,z)=(-v_1v_3,-v_2v_3,v_1^2+v_2^2)$ Ellenőrzéssel megállapíthatjuk, hogy a kapott y" -bázisvektor, valamint az x",z" bázisvektorainak páronkénti skaláris szorzatai nullát adnak. Összefoglalva, a kapott *bázisvektorok*:

(3.4.4.1.2.)

$\overrightarrow{v}_x := (-v_2,\ v_1,\ 0)$	ahol: $v_i=p_i-s_i$ $(i=1,2,3)$ és: $(x,y),(x",y")$ nem párhuzamos
$\overrightarrow{v}_y := (-v_1v_3,\ -v_2v_3,\ v_1^2+v_2^2)$	
$\overrightarrow{v}_z := (v_1,\ v_2,\ v_3)$	

Ezek közül $\overrightarrow{v}_x$ és $\overrightarrow{v}_y$ egyúttal az (x",y") képsíknak is bázisvektorai. Előnyös tulajdonságokkal rendelkeznek, mert $\overrightarrow{v}_x$ párhuzamos az (x,y) síkkal, tehát ha (x,y) vízszintes sík,akkor x" is vízszintes marad.

Ez a valós testek ábrázolásánál játszik szerepet, mert a P szempontból nézett test megőrzi irányát is a képsíkon, pl. egy álló épület a képsíkon is áll. Tehát a megszerkesztendő perspektív leképezés injektív és *ebben az értelemben iránytartó* lesz, mivel az (x",y") koordináták kiválasztásakor a végtelen sok lehetőség közül az optimálisat jelöltük ki. A kapott koordináta-rendszer is jobbsodrású, ezt szemlélteti a 6. ábra:



6. ábra

Térjünk most vissza arra az esetre, amikor az (x,y) és az (x",y") síkok párhuzamosak. Erre az esetre is jó a fenti ortogonalizációs eljárás, mégis érdemes megvizsgálni külön, mivel törekednünk kell a teljességre. A képsík ez esetben is átmegy S-en, tehát megegyezik azzal a síkkal, amellyel a képsíkot az előző konstrukció során metszettük. Ez esetben a közös rész nem egy egyenes, hanem maga a képsík és $p_1=s_1$, $p_2=s_2$, ezért: $\overrightarrow{v}_x=\overrightarrow{v}_y=\overrightarrow{0}$ és $\overrightarrow{v}_z=(0,0,v_3)$. A nullvektor minden vektorral párhuzamos, illetve merőleges, éppen ezért nem is lehet bázisvektora a térnek. Tehát a kapott vektorok ez esetben nem lesznek a térnek bázisai, más bázist kell találnunk ebben a speciális esetben.

A vizsgált esetben célszerű ezért eltekinteni a T" transzformációtól, hiszen elforgatásra nincs is szükség ebben az esetben, azaz $T"=\hat{T}$ és (x",y",z")=(x,y,z). Másként fogalmazva ez azt jelenti, hogy a T" az identikus transzformáció lesz esetünkben. A tér bázisát most a következő vektorok alkotják:

(3.4.4.1.3.)

$\overrightarrow{v}_x=(1,0,0)$	$\overrightarrow{v}_y=(0,1,0)$	$\overrightarrow{v}_z=(0,0,1)$
ha (x,y) és (x",y") párhuzamosak, azaz $v_1=v_2=0$ és $v_3\neq 0$		

Az általános T" -transzformáció meghatározásához tehát mind az (3.4.4.1.2.), mind az (3.4.4.1.3.) eseteket figyelembe kell venni. Eddig tehát meghatároztuk a bázisvektorokat. Ezekre azért volt szükség, mert a segítségükkel fel tudjuk írni a T" transzformáció mátrixát. Ez azonban még kevés ehhez, nézzük a fenti ortogonalizációs eljárás diszkusszióját, hogy megállapíthassuk, érvényes-e a teljes $\mathbb{R}^3$-re. Mivel a T" transzformációt az ortogonalizációs eljárás segítségével építjük fel, ezért a diszkusszióból kiindulva megállapíthatjuk, hogy a transzformáció mely szem-tárgy vektorok mellett van értelmezve. Vizsgáljuk meg tehát a szem-tárgy vektor koordinátáit alkotó $(v_1,v_2,v_3)\in\mathbb{R}^3$ számhármasokat minden esetben, hogy tudunk-e és milyen bázisvektorokat alkotni belőlük.

3.4.4.2. Az ortogonalizációs eljárás és a 't'-transzformáció érvényességi körének vizsgálata

Vizsgáljuk meg a szem-tárgy vektor - $\overrightarrow{V} = \overrightarrow{S} - \overrightarrow{P} = (v_1, v_2, v_3)$ - összes lehetséges helyzetét. A következő táblázat az összes lehetséges esetet megmutatja, aszerint, hogy v_1, v_2, v_3 értéke nulla vagy sem: Ha nem, ezt a tényt a táblázatban 1-essel jelöljük, egyébként 0-val:

eset:	v_1	v_2	v_3
a	0	0	0
b	0	0	1
c	0	1	0
d	0	1	1
e	1	0	0
f	1	0	1
g	1	1	0
h	1	1	1

a) $v_1=v_2=v_3=0$, azaz P = S értelmetlen, hiszen nem nézhetünk a saját szemünkbe, ez kiderül másrészt abból is, hogy ez esetben minden bázisvektor nullvektor lenne, azaz az ez esetben nem végezhető el az ortogonalizációs eljárás.

b) ha $v_1=v_2=0$ és $v_3 \neq 0$, azaz: $v_1^2+v_2^2=0$ és $v_3 \neq 0$, ez éppen a már kiszámított párhuzamos eset, a (3.4.4.1.3.) összefüggés definiálja a bázisvektorokat.

c,d,g,h) esetben teljesül mind a $v_2 \neq 0$ és a ($v_1 \neq 0$ vagy $v_2 \neq 0$) feltétel egyaránt, azaz az (x,y) és az (x",y") síkok nem párhuzamosak, ezen esetekben tehát a bázisvektorok megkonstruálására alkalmas a (3.4.4.1.2.) összefüggés.

e,f) esetben $v_2=0$, ekkor a metszésvonal (3.4.4.1.1.) egyenletéből: $v_1x - v_1s = 0$, tehát $x = s_1$ -et kaphatunk. Ez esetekben módosítsuk kissé az ortogonalizációs eljárást: Mivel $x = s_1$ teljesül minden y-érték mellett, ezért most válasszuk y értékéül az $s_2+ v_1$ mennyiséget. Innen az ortogonalizációs eljárás szerint továbbhaladva: $\overrightarrow{Q} = (s_1,s_2+v_1,s_3)$, ebből $\overrightarrow{SQ} = \overrightarrow{Q} - \overrightarrow{S} = (0,v_1,0) =: \overrightarrow{V}_x$ Ezután a vektorok skaláris szorzatára vonatkozó, már említett lineáris egyenletrendszerből: $\overrightarrow{V}_z := (-v_1v_3, 0, v_1^2)$ Ha $v_2=0$ -ra alkalmazzuk a (3.4.4.1.2.) képletet, akkor ugyanerre az eredményre jutunk. Tehát a leírt ortogonalizációs eljárás és eredménye az 'e' és az 'f' esetekben is helyes.

Figyelembe véve mindazt, amit ebben a részben megállapítottunk és kiszámítottunk, összegezzük a bázisvektorokra kapott eredményeinket a szem-tárgy vektor szerint:

(3.4.4.2.1.) Ha $v_1=v_2=v_3=0$, akkor a feladat értelmetlen. (P=S, saját szemünkbe nem nézhetünk.)

(3.4.4.2.2.)

$\overrightarrow{V}_x=(1,0,0)$	$\overrightarrow{V}_y=(0,1,0)$	$\overrightarrow{V}_z=(0,0,1)$
ha $v_1=v_2=0$ (azaz: $v_1^2+v_2^2=0$) és $v_3\neq0$ (párhuzamos esetben, $V \in \{0,0,\mathbb{R}\}$)		

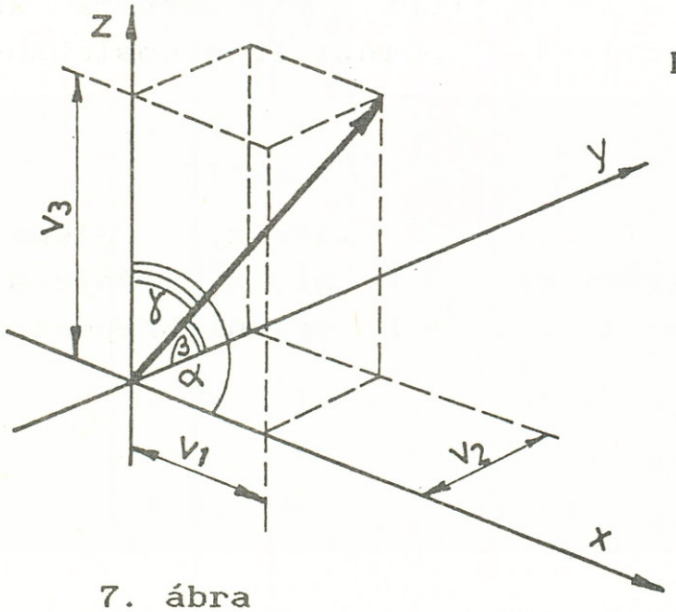
(3.4.4.2.3.)

$\overrightarrow{V}_x := (-v_2,\ v_1,\ 0)$	Minden más esetben. Ez az általános eset.
$\overrightarrow{V}_y := (-v_1v_3,\ -v_2v_3,\ v_1^2+v_2^2)$	
$\overrightarrow{V}_z := (v_1,\ v_2,\ v_3)$	

A most kapott bázisvektorokból fogjuk felírni az elforgatási mátrixot. Egyúttal áttekintjük az elforgatási mátrixok matematikai hátterét is. A T" elforgatási mátrix elemei éppen a bázisvektorok régi tengelyek szerinti irányicosinusai, ezért a 3.4.4.3. részben definiáljuk *általánosan*, mit értünk irányicosinus alatt, majd előbb a (3.4.4.2.3.), általános esetre, utóbb a (3.4.4.2.2.), párhuzamos esetre definiáljuk a transzformációt. A 'párhuzamos' és az 'általános' fogalmakat később is használjuk, megkülönböztetésre.

3.4.4.3. *Adott* $\vec{v}=(v_1,v_2,v_3)$ *vektor iránycosinusai*

Minden vektornak, mely a három dimenziós térben található, a tér három koordináta-tengelye szerinti három iránycosinusát határozhatjuk meg. Legyen a vizsgált $\vec{v}$-vektornak az x,y,z tengelyekkel bezárt szöge rendre: α,β,γ . Mindezt a 7. ábra is szemlélteti:



7. ábra

Legyen $d_v := |\vec{v}| = \sqrt{v_1^2 + v_2^2 + v_3^2}$

$$\cos(\alpha) = \frac{v_1}{d_v}$$

$$\cos(\beta) = \frac{v_2}{d_v}$$

$$\cos(\gamma) = \frac{v_3}{d_v}$$

A $\vec{v}$ vektornak az x,y,z tengelyekre vonatkozó iránycosinusai tehát rendre: $\cos(\alpha)$, $\cos(\beta)$, $\cos(\gamma)$. Képezzük ezeket az előző részben definiált $\vec{v}_x$, $\vec{v}_y$, $\vec{v}_z$ bázisvektorokra is. Ehhez szükséges kiszámítani ezeknek a vektoroknak az abszolút értékeit, amelyeket jelöljünk rendre d_x,d_y,d_z-vel. Ez nem más, mint az említett vektorok euklideszi normája. A (3.4.4.2.2.) -vel leírt párhuzamos esetben: $d_x=d_y=d_z=1$; a $v_1=v_2=v_3=0$ eset a transzformáció szempontjából értelmetlen, az általános (3.4.4.2.2) esetben pedig:

$$d_x := \sqrt{v_1^2 + v_2^2} \qquad (v_i := p_i - s_i \ , i=1,2,3)$$

$$d_y := \sqrt{v_1^2 v_3^2 + v_2^2 v_3^2 + (v_1^2 + v_2^2)^2} = \sqrt{(v_1^2 + v_2^2)(v_1^2 + v_2^2 + v_3^2)}$$

$$d_z := \sqrt{v_1^2 + v_2^2 + v_3^2} = \| P - S \|_E \qquad (P,S \text{ euklideszi normája})$$

A fentiekből kiderül a következő érdekes összefüggés is: $d_y=d_x d_z$, ez is gyorsabbá teheti az algoritmust, ha számítógépre alkalmazzuk. Általában véve a grafikus algoritmusoknál nagyon sok lehetőség közül választhatunk; nem mindegy, melyiket alkalmazzuk, mert igen nagy futási időt takaríthatunk meg az algoritmus apró változtatásával is, de egy látszólag elhanyagolható tényező is nagymértékben, sőt használhatatlanná téve lelassíthatja a grafikus algoritmusokat.

3.4.4.4. *Az elforgatási mátrix*

Az eddigiekben d_x, d_y, d_z értékeket definiáltuk, és az irány-cosinust felírtuk egy általános vektorra. A bázisvektorok irány-cosinusaival most felírjuk az elforgatás mátrixát. Tehát keressük az (x,y,z) tengelyről (x",y",z") tengelyekre való elforgatás mátrixát, azaz azt a mátrixot, amely a $\vec{v}$ vektort $\vec{v}$" vektorra képezi le. Ennek elemei a régi tengelyek szerinti iránycosinusok. Ezt egyenlettel megfogalmazva:

$\vec{v}" := \vec{v} \cdot T"$,másként: $(x",y",z") := (x,y,z) \cdot T"$

ahol: $T" \in \mathbb{R}^{3\times 3}$, azaz T": 3x3-as mátrix, elemei: $t_{i,j}$, ahol: i,j=1,2,3. Elemei az iránycosinusok, pl. x" iránycosinusai: $t_{1,1}, t_{1,2}, t_{1,3}$ Kifejtve ez a következő egyenletrendszert jelenti:

$$x" := t_{1,1} \cdot x + t_{2,1} \cdot y + t_{3,1} \cdot z$$
$$y" := t_{1,2} \cdot x + t_{2,2} \cdot y + t_{3,2} \cdot z$$
$$z" := t_{1,3} \cdot x + t_{2,3} \cdot y + t_{3,3} \cdot z$$

Ezt más formában írva, a transzformációs mátrix-szal:

$$(x",y",z") := (x,y,z) \cdot \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,1} & t_{3,2} & t_{3,3} \end{bmatrix}$$

Vegyük most figyelembe az iránycosinusok előbb leírt kiszámítási módját és a bázisvektorokra kapott értékeket, a (3.4.4.2.3) általános esetben a következő eredményre jutunk:

$$T" = \begin{bmatrix} t_{1,1} & t_{1,2} & t_{1,3} \\ t_{2,1} & t_{2,2} & t_{2,3} \\ t_{3,1} & t_{3,2} & t_{3,3} \end{bmatrix} := \begin{bmatrix} \frac{-v_2}{d_x} & \frac{-v_1 v_3}{d_y} & \frac{v_1}{d_z} \\ \frac{v_1}{d_x} & \frac{-v_2 v_3}{d_y} & \frac{v_2}{d_z} \\ 0 & \frac{v_1^2+v_2^2}{d_y} & \frac{v_3}{d_z} \end{bmatrix}$$

Ez tehát T" transzformáció keresett mátrixa, ahol az eddigiek szerint: $v_i := p_i - s_i$ (i:=1,2,3). A (3.4.4.2.2.) párhuzamos esetben T"=I, az identikus mátrix, mert ekkor forgatásra nincs szükség.

3.4.5. A 't' transzformáció definiálása, általános esetben

Az eddigiekben részeire bontottuk a képsík-transzformációt, majd ezeket a részeket külön kiszámítottuk, most a teendőnk egyesíteni számításaink eredményeit. Először írjuk fel a T' és T" transzformációk kompozícióját, a $\hat{T}$ -transzformációt: $\hat{T} = T"(T')$
A (3.4.4.2.2.) párhuzamos esettel a (3.4.6.) részben foglalkozunk.

(i)
$$\begin{cases} \hat{x} := x"(x') = \dfrac{-v_2}{d_x}\cdot(x-s_1) + \dfrac{v_1}{d_x}\cdot(y-s_2) \\ \hat{y} := y"(y') = \dfrac{-v_1 v_3}{d_y}\cdot(x-s_1) + \dfrac{-v_2 v_3}{d_y}\cdot(y-s_2) + \dfrac{v_1^2+v_2^2}{d_y}\cdot(z-s_3) \\ \hat{z} := z"(z') = \dfrac{v_1}{d_z}\cdot(x-s_1) + \dfrac{v_2}{d_z}\cdot(y-s_2) + \dfrac{v_3}{d_z}\cdot(z-s_3) \\ \text{Mindenütt: } v_i := p_i - s_i \quad (i:=1,2,3) \end{cases}$$

Ezt mátrixos alakban felírva: $\hat{Q} := (Q-T')\cdot T" = (Q-S)\cdot T"$
Ismét más alakban, tetszőleges $Q=(q_1,q_2,q_3)$ pontra:

(ii)
$$(\hat{q}_1,\hat{q}_2,\hat{q}_3) := [(q_1,q_2,q_3)-(s_1,s_2,s_3)]\cdot \begin{bmatrix} \dfrac{-v_2}{d_x} & \dfrac{-v_1 v_3}{d_y} & \dfrac{v_1}{d_z} \\ \dfrac{v_1}{d_x} & \dfrac{-v_2 v_3}{d_y} & \dfrac{v_2}{d_z} \\ 0 & \dfrac{v_1^2+v_2^2}{d_y} & \dfrac{v_3}{d_z} \end{bmatrix}$$

Ezzel tulajdonképpen *áttértünk az* $(\hat{x},\hat{y},\hat{z})$ *koordináta-rendszerre*. Hozzávéve ehhez a legegyszerűbb perspektív leképezést, már meg is határoztuk a tárgyközpontú perspektív leképezést, a t-transzformációt. (3.4.1.) alapján: $t=T'''(\hat{T})$, tehát a most definiált $\hat{Q}$ pontra írjuk fel a T''' transzformációt. Ehhez előbb fel kell írnunk az $(\hat{x},\hat{y},\hat{z})$ renszerben Q és P koordinátáit, melyeket $\hat{S}$ és $\hat{P}$ szimbólumokkal jelölhetünk. A rendszer tervezéséből és a $\hat{T}$ transzformációra fentebb kapott egyenletekbe való behelyettesítéssel egyaránt adódik:

(iii) $S=(s_1,s_2,s_3)=(0,0,0)$ és $P=(p_1,p_2,p_3)=(0,0,d_z)$

Ezt behelyettesítve a (3.3.4.) -egyenlőségekbe, a következő eredményre jutunk:

$$\text{(iv)}\quad \begin{cases} x_t = x''' := \hat{x}\cdot\dfrac{d_z}{d_z-\hat{z}} := \dfrac{\hat{x}\cdot d_z}{d_z-\hat{z}} \\ \\ y_t = y''' := \hat{y}\cdot\dfrac{d_z}{d_z-\hat{z}} := \dfrac{\hat{y}\cdot d_z}{d_z-\hat{z}} \end{cases}$$

Az (i) egyenletrendszerből kiderül, hogy $\hat{x},\hat{y}$ és $\hat{z}$ tulajdonképpen egy polinom: $ax+by+cz+d$ alakúak, ahol $a,b,c,d \in \mathbb{R}$, így az (iv) egyenletrendszerben szereplő x''' és y''' is két polinom hányadosaként írható fel: $\hat{x}\cdot d_z$ és $d_z-\hat{z}$, illetve $\hat{y}\cdot d_z$ és $d_z-\hat{z}$ hányadosaként. Írjuk fel ilymódon az (i) egyenletrendszerből származtatva ezeket az értékeket, rendezve polinomiális alakra és kissé átalakítva:

$$\text{(v)}\quad \begin{cases} \hat{x}\cdot d := -v_2\cdot\dfrac{d_z}{d_x}\cdot x + v_1\cdot\dfrac{d_z}{d_x}\cdot y + (s_1\cdot v_2 - s_2\cdot v_1)\cdot\dfrac{d_z}{d_x} \\ \\ \hat{y}\cdot d_z := -v_1\cdot v_3\cdot\dfrac{d_z}{d_y}\cdot x - v_2\cdot v_3\cdot\dfrac{d_z}{d_y}\cdot y + (v_1^2+v_2^2)\cdot\dfrac{d_z}{d_y}\cdot z + \\ \qquad + [(s_1\cdot v_1 + s_2\cdot v_2)\cdot v_3 - (v_1^2+v_2^2)\cdot s_3]\cdot\dfrac{d_z}{d_y} \\ \\ d_z-\hat{z} := -v_1\cdot\dfrac{1}{d_z}\cdot x - v_2\cdot\dfrac{1}{d_z}\cdot y - v_3\cdot\dfrac{1}{d_z}\cdot z + (v_1p_1+v_2p_2+v_3p_3)\cdot\dfrac{1}{d_z} \\ \\ \text{Mindenütt:}\quad v_i := p_i - s_i \quad (i:=1,2,3) \end{cases}$$

Csak a $d_z-\hat{z}$ polinomban levő konstans tag igényelt több meggondolást. Itt összevonásokat, a $p_i = v_i+s_i$ egyenlőséget és a d_z-re is vonatkozó (3.4.4.3.)-beli egyenlőséget vettük figyelembe. Az egyenletrendszert ennél egyszerűbb alakra szeretnénk hozni, ezért bevezetünk néhány új jelölést, amelyek majd elősegítik a végső eredmény egyszerűsítését. Tehát alkalmazzuk a következő helyettesítéseket és jelöléseket, felhasználva a (3.4.4.3.) részben kapott egyenlőségeket:

$$\text{(vi)}\quad \begin{cases} c_x := \dfrac{d_z}{d_x} = \sqrt{\dfrac{v_1^2 + v_2^2 + v_3^2}{v_1^2 + v_2^2}} \\ c_y := \dfrac{d_z}{d_y} = \dfrac{d_z}{d_x d_z} = \dfrac{1}{d_x} = \dfrac{1}{\sqrt{v_1^2 + v_2^2}} \\ c_z := \dfrac{1}{d_y} = \dfrac{1}{\sqrt{v_1^2 + v_2^2 + v_3^2}} \end{cases}$$

Természetesen itt is: $v_i := p_i - s_i$ $(i=1,2,3)$. Ha számítógépes algoritmust szeretnénk írni erre a feladatra, akkor még célszerű két segédváltozót is bevezetni: $d:=v_1^2+v_2^2$ és $e:=v_1^2+v_2^2+v_3^2$ Vegyük észre azt is, hogy: $c_y=c_x c_z$, hasonlóan a $d_y=d_x d_z$ egyenlőséghez. Mindezt figyelembevéve az előző egyenlet a következőképpen alakul:

$$\text{(vii)}\quad \begin{cases} c_y := \dfrac{1}{\sqrt{d}} \qquad c_z := \dfrac{1}{\sqrt{e}} \\ c_x := \dfrac{c_y}{c_z} \end{cases}$$

Az (v) egyenletrendszeren elvégezve ezeket a helyettesítéseket, kis átalakítás után a következő eredményre jutunk:

$$\text{(viii)}\quad \begin{cases} \hat{x}\cdot d_z := -v_2 c_x x + v_1 c_x y + (s_1 v_2 - s_2 v_1) c_x \\ \hat{y}\cdot d_z := -v_1 v_3 c_y x - v_2 v_3 c_y y + (v_1^2 + v_2^2) c_y z + \\ \qquad + [(s_1 v_1 + s_2 v_2) v_3 - (v_1^2 + v_2^2) s_3] c_y \\ d_z - \hat{z} := -v_1 c_z x - v_2 c_z y - v_3 c_z z + (v_1 p_1 + v_2 p_2 + v_3 p_3)\cdot c_z \\ \text{Mindenütt:} \quad v_i := p_i - s_i \quad (i:=1,2,3) \end{cases}$$

Ezen a módon a t-transzformációt egyszerűen le tudjuk írni, mivel x_t és y_t egy-egy polinom hányadosaként áll elő, az (iv) egyenletrendszer alapján, a (viii) egyenlőségekben megadott értékeknek megfelelően. Ha a feladatot programozni akarjuk, akkor elegendő felírni az x,y,z -höz tartozó és a konstans együtthatókat, ezt minden objektumra csak egyszer kell elvégezni, a polinomokkal való osztást pedig minden pontra külön. Ez azt jelenti, hogy x_t és y_t alakja a következő lesz:

$$\text{(ix)} \quad \begin{cases} x_t := \dfrac{\hat{x}\cdot d_z}{d_z - \hat{z}} = \dfrac{0 + x_2 y + x_1 x + x_o}{z_3 z + z_2 y + z_1 x + z_o} \\ y_t := \dfrac{\hat{y}\cdot d_z}{d_z - \hat{z}} = \dfrac{y_3 z + y_2 y + y_1 x + y_o}{z_3 z + z_2 y + z_1 x + z_o} \end{cases}$$

Definiáljuk ebben az egyenletrendszerben az együtthatókat, (viii) és (ix) alapján:

$$\text{(x)} \quad \begin{cases} x_2 := v_1 \cdot c_x \qquad x_1 := -v_2 \cdot c_x \qquad x_o := (s_1 \cdot v_2 - s_2 \cdot v_1) \cdot c_x \\ y_3 := (v_1^2 + v_2^2) \cdot c_y \qquad y_2 := -v_2 \cdot v_3 \cdot c_y \qquad y_1 := -v_1 \cdot v_3 \cdot c_y \\ y_o := [(s_1 \cdot v_1 + s_2 \cdot v_2) \cdot v_3 - (v_1^2 + v_2^2) \cdot s_3] \cdot c_y \\ z_3 := -v_3 \cdot c_z \qquad z_2 := -v_2 \cdot c_z \qquad z_1 := -v_1 \cdot c_z \\ z_o := (v_1 \cdot p_1 + v_2 \cdot p_2 + v_3 \cdot p_3) \cdot c_z \\ \text{Mindenütt: } v_i := p_i - s_i \quad (i := 1,2,3) \end{cases}$$

Írjuk fel most egy tetszőleges $Q:(q_1,q_2,q_3)$ pontra is a (ix) egyenletet: $t: (q_1,q_2,q_3) \rightarrow (q_1^t,q_2^t,q_3^t)$

$$\text{(xi)} \quad \begin{cases} q_1^t := \dfrac{x_2 q_2 + x_1 q_1 + x_o}{z_3 q_3 + z_2 q_2 + z_1 q_1 + z_o} \\ q_2^t := \dfrac{y_3 q_3 + y_2 q_2 + y_1 q_1 + y_o}{z_3 q_3 + z_2 q_2 + z_1 q_1 + z_o} \end{cases}$$

Természetesen itt is érvényesek az előzőekben felírt együtthatók.

3.4.6. A 't' transzformáció definiálása, párhuzamos esetben

A (3.4.4.2.) részben meghatároztuk a perspektív transzformációk alapját képező bázisvektorokat. Három esetet különböztettünk meg. Eddig az általános esettel foglalkoztunk, most vizsgáljuk meg a párhuzamos esetet is. Célunk, hogy a szem-tárgy vektornak a transzformációkkal kapcsolatos értelmezési tartományát a lehető legteljesebb mértékben kiterjesszük, azaz az előző részben meghatározott (ix) egyenletrendszer - ha más együtthatókkal is - érvényes legyen a teljes értelmezési tartományon. Az egyenletrendszert meghatározó szem-tárgy vektort eddig a következő tartományon értelmeztük:

$$t_D^1 := \{V:(v_1,v_2,v_3) \mid V\in\mathbb{R}^3 \text{ és: } v_1^2+v_2^2\neq 0\} = \{\ \mathbb{R}^3 / \{0x0x\mathbb{R}\}\ \}$$

Tehát az $\{\mathbb{R}^3 \setminus t_D^1\} =: t_D^2$ halmazra is ki kell terjeszteni az értelmezési tartományt. A (3.4.4.2.) részben leírt párhuzamos esetre teljesül: (x",y",z")=(x',y',z'), azaz most elforgatásra nincs szükség, tehát: t = T'''(T'), szemben az általános esetre vonatkozó t = T'''(T"(T')) egyenlettel. Ez utóbbi is érvényes a párhuzamos esetre, mivel a T" transzformáció mátrixa az identikus mátrix, azaz a főátló 1-esekből, a többi elem nullákból áll. Mivel a párhuzamos eset kritériuma: $p_1=s_1$, $p_2=s_2$ (azaz: $v_1^2+v_2^2=0$) és $p_3\neq s_3$, ezért (3.4.3.4.) alapján:

$$(x',y',z')=(x-s_1,y-s_2,z-s_3)=(x-p_1,y-p_2,z-s_3)$$

Ezen képletet a (3.3.4.) összefüggéssel összevetve, felírhatjuk a t-transzformáció egyenletrendszerét a párhuzamos esetre, majd a kapott összefüggéseket olyan formára hozva, mint az általános esetet leíró (3.4.5.) rész (ix) egyenletrendszere:

(i)
$$\begin{cases} x_t := x' \cdot \dfrac{p_3}{p_3 - z'} = \dfrac{p_3 x - p_1 p_3}{-z + s_3 + p_3} \\[2ex] y_t := y' \cdot \dfrac{p_3}{p_3 - z'} = \dfrac{p_3 y - p_2 p_3}{-z + s_3 + p_3} \end{cases}$$

Ebből következően a párhuzamos esetben a transzformációs együtthatók - melyeket az általános esetben a (3.4.5.) részben definiáltunk - a következők lesznek:

$$
\text{(ii)}\quad \begin{cases} & x_2 = 0 & x_1 = p_3 & x_o = -p_1 p_3 \\ y_3 = 0 & y_2 = p_3 & y_1 = 0 & y_o = -p_2 p_3 \\ z_3 = -1 & z_2 = 0 & z_1 = 0 & z_o = s_3 + p_3 \end{cases}
$$

Ezeket az együtthatókat kell tehát alkalmaznunk abban az esetben, ha $V \in \{0x0x\mathbb{R}\}$, ahol 'V' a szem-tárgy vektor koordinátáiból alkotott számhármast jelenti. Mindez azt jelenti, hogy a szem-tárgy vektor értelmezési tartománya a következő:

$$V \in \mathbb{R}^3 \setminus (0,0,0)$$

Azaz a nullvektor kivételével minden vektorra - mint szem-tárgy vektorra - értelmezhetjük a tárgyközpontú perspektivikus transzformációt. Ezzel a leképezés definiálását és néhány, hozzá közelálló matematikai probléma bemutatását befejeztük. Reméljük, hogy ezek a vizsgálódások hasznosak voltak a három dimenziós grafika elméleti háttere bemutatása szempontjából.

3.4.7. *Néhány kiegészítő transzformáció*

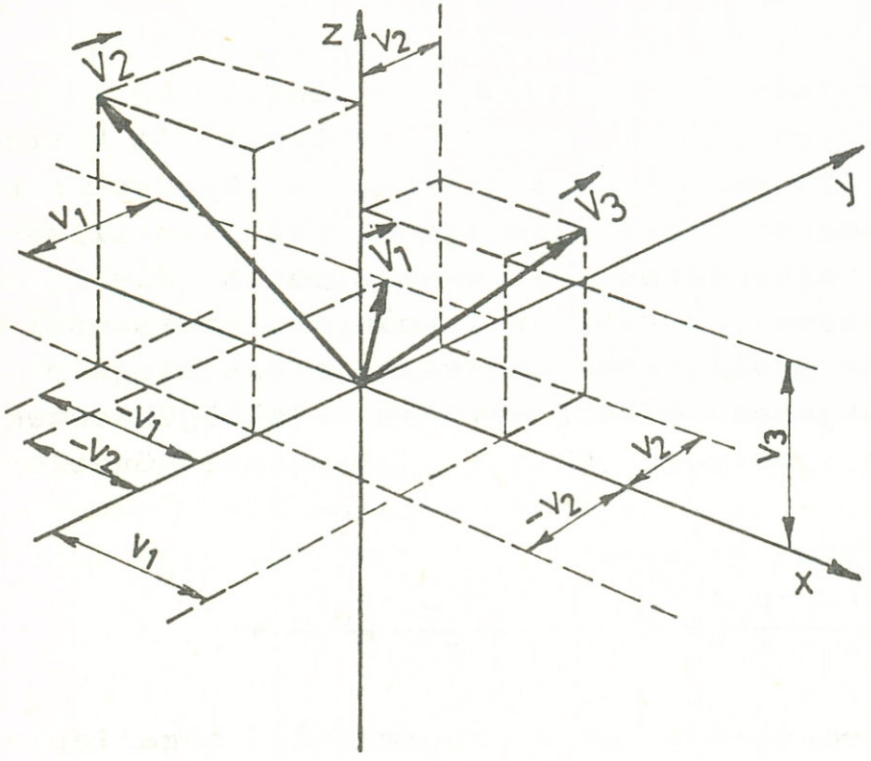
A (4.6.) részben sok kiegészítő transzformációval foglalkozunk, de most más megközelítésben említsünk meg néhány fontos transzformációt: a képernyőn vízszintes, illetve függőleges irányú nagyítást-kicsinyítést; valamint a szimpla, illetve dupla felbontás különbségeit áthidaló transzformációt. Ez utóbbihoz az (5.2.) rész adja meg az elméleti alapot. Legyen a vízszintes nagyítási (illetve kicsinyítési) tényező: l_x (ha $l_x > 0$, akkor nagyítás, ha $l_x < 0$, akkor kicsinyítés). A függőleges nagyítási tényező: l_y Ekkor a (3.4.5.)-beli (vi) alapján kapott c_i értékeket módosítva ezekkel a tényezőkkel, a perspektív transzformációval kapott kép l_x illetve l_y -szorosára növekszik. Ehhez elegendő c_x illetve c_y eredeti értékeit l_x illetve l_y -szorosára növelni.

A szimpla és dupla felbontás különbségeit áthidaló transzformáció definiálásához vezessük be a h_x tényezőt, melynek értéke dupla felbontásnál: 2, egyébként: 1. Így elegendő c_x eredeti értékét h_x-szeresére növelni. Összefoglalva mindez a következőket jelenti:

$$c_x := c_x \cdot l_x \cdot h_x \qquad c_y := c_y \cdot l_y$$

3.5. A szem perspektív transzformációjának definiálása

Eddig a tárgyponton átmenő képsíkot használtuk, az ehhez kapcsolódó transzformációt írtuk le. Most első lépésben általánosítsuk ezt a transzformációt a következőképpen: a képsík helyét ne határozzuk meg, legyen az tetszőleges helyen a térben, csupán azt a megkötést tegyük, hogy legyen merőleges a szem-tárgy vektorra.



8. ábra

Legyen a 8. ábra szerint a képsík a P szemponttól 'd' távolságra, $d:=\| P - R \|_E$ (euklideszi norma). Most is az S jelű tárgypontra nézünk. A szem-tárgy vektor és a képsík metszéspontja legyen: R, tehát olyan, mintha az R pontra néznénk. Az így leírt képsíkon keletkező képet előállító transzformációt úgy definiáljuk, hogy kiszámítjuk R koordinátáit és alkalmazzuk az eddig meghatározott transzformációkat R:=S helyettesítéssel. $R:(r_1,r_2,r_3)$; $S:(s_1,s_2,s_3)$ és $P:(p_1,p_2,p_3)$. A párhuzamos szelők tétele alapján:

$$\frac{r_i - p_i}{s_i - p_i} = \frac{\| R - P \|_E}{\| S - P \|_E}$$

Ha R az S pont felé eső irányban van.

$$\frac{p_i - r_i}{s_i - p_i} = \frac{\| R - P \|_E}{\| S - P \|_E}$$

Ha R az S ponttal ellenkező irányban, a szem mögött van.

Ebből rendezés után kiszámíthatók R koordinátái: (3.5.1.)

$$r_i = \pm \frac{\| R - P \|}{\| S - P \|} \cdot (s_i - p_i) + p_i = \pm \frac{d}{\| S - P \|} \cdot (s_i - p_i) + p_i$$

Itt '+' érvényes akkor, ha a képsík a szem *előtt* van, '-' pedig akkor, ha a képsík a szem mögött van. Módszerünk mármost a következő: a (3.5.1.) képlet alapján meghatározzuk R koordinátáit, majd az S:=R helyettesítést alkalmazzuk, ezután a tárgyközpontú perspektív transzformációra kapott összefüggéseket alkalmazzuk. Ilymódon eljutottunk az *általános perspektív* transzformációhoz. Ennek egyik speciális esete a szem perspektív transzformációja, melynek definiálásához mostmár csupán a szem leképezésére vonatkozó néhány adatot kell figyelembevenni.

A szem esetén a (3.5.1.) egyenlőségnél a '-' érvényes. Most ugyanis a P pont fogalmán az optikai középpontot kell értenünk, képsíkon pedig az ideghártyát, másnéven retinát. A kettő távolsága jó közelítéssel 15 mm, a képsík tehát a P (szűkebb értelemben vett szempont) mögött van. Számoljunk ezentúl milliméterben, ezért a (3.5.1.) egyenletben a szem esetén d:=15, a '-' előjel és az R:=S_T érvényes. Ugyanis a látósugár (axis opticus) és a képsík (retina) az eddigiekben S_T-vel jelölt sárgafolt (fovea centralis) középpontjában metszi egymást. Jelöljük ennek koordinátáit: $S_T(s_1^T, s_2^T, s_3^T)$, melyeket (3.5.1.) alapján a következőképpen számíthatunk ki:

$$(3.5.2.) \qquad s_i^T = -\frac{15\cdot(s_i - p_i)}{\| S - P \|} + p_i = \frac{15\cdot(p_i - s_i)}{\| S - P \|} + p_i$$

Már csak a Helmholtz-állandót kell értelmezni. Megállapítottuk a (2.2.) részben, hogy - a fénytörés, az optikai sűrűség, stb. tényezők együttes hatása miatt - az ideghártyán keletkező kép jó közelítéssel 0.81 arányban csökken. A kép tehát ugyanaz lesz, mintha 0.81-ára csökkenne a képsík és a fény szemlencsébe való belépési pontjának (F) távolsága ($f \sim 19$ mm). Ezért ha felveszünk $0.81 \cdot f$ távolságra F mögött egy képsíkot, akkor az ezen keletkező kép - a Helmholtz-állandót meghatározó hatások *nélkül* - egybevágó az F mögött f távolságra lévő olyan képpel, melynek keletkezése a Helmholtz-állandó függvénye. Tehát a P mögött mintegy 15 mm-re lévő, Helmholtz-állandó nélküli képsíkkal is számolhatunk a szem esetén. Ez azt jelenti, hogy mindkét lehetőség ugyanoda vezet matematikai szempontból:

1. d=19 -cel számolunk, majd az S_T-re, a képsík középpontjára kapott koordinátákra felírjuk a 't'-transzformációt és a kapott kép minden pontjának koordinátáit megszorozzuk a Helmholtz-állandóval.

2. Az optikai középponttal, azaz d=15 -tel számolunk, majd a képsík középpontjára kapott koordinátákra felírjuk a 't' transzformációt. Ez azért előnyösebb az előzőnél, mert ott minden pont mindkét koordinátáját meg kellett szorozni egy számmal, most nem. Ez nagyobb objektumok számításánál jelentős gépidőmegtakarítást eredményez. Esetünkben tehát d=15 milliméter, így tehát a (3.5.1.) egyenlet a következőképpen határozza meg a képsík középpontjának koordinátáit:

$$(3.5.3.) \qquad s_i^T = \frac{15\cdot(p_i - s_i)}{\| S - P \|} + p_i \qquad (i=1,2,3)$$

Ezzel definiáltuk a teljes T-transzformációt, szem által megvalósított leképezést, mely a körülöttünk levő három dimenziós világot két dimenzióra képezi le. Mielőtt összefoglalnánk ennek a transzformációnak a lényegét, említsünk meg egy műveletcsökkentési lehetőséget a számítógépes feldolgozás meggyorsítására. A 't'-transzformációt a (3.4.5.) rész (ix) egyenletrendszere írja le. Vezessük be a q_o jelet is, úgy, hogy $q_o \equiv 1$ és az (ix) egyenletrendszer minden együtthatóját osszuk el x_2-vel, ha $x_2 \neq 0$. Ekkor az (ix) egyenletrendszer a következő formájú lesz:

$$(3.5.4.)\begin{cases} q_1^t = \dfrac{\sum_{i:=o}^{2} \frac{x_i}{x_2} q_i}{\sum_{i:=o}^{3} \frac{z_i}{x_2} q_i} := \dfrac{\sum_{i:=o}^{2} x_i q_i}{\sum_{i:=o}^{3} z_i q_i} \\ q_2^t = \dfrac{\sum_{i:=o}^{3} \frac{y_i}{x_2} q_i}{\sum_{i:=o}^{3} \frac{z_i}{x_2} q_i} := \dfrac{\sum_{i:=o}^{3} y_i q_i}{\sum_{i:=o}^{3} z_i q_i} \end{cases}$$

Azaz a transzformációs együtthatókon a következő változást hajtjuk végre:

$$x_i := x_i / x_2 \quad (i=0,1,2)$$
$$y_i := y_i / x_2 \quad (i=0,..3)$$
$$z_i := z_i / x_2 \quad (i=0,..3)$$

Ekkor az objektum minden pontjának kiszámításakor egy szorzással csökkenthető a műveletek száma, ennek feltétele: $x_2 \neq 0$, azaz $p_1 \neq s_1$. Ezután foglaljuk össze mindazt, amit a látásunk matematikai modelljéről eddig leírtunk.

3.6. A látás elmélete (I): összefoglalás

Ebben a részben nem csak az eddigi eredményeinket foglaljuk össze, hanem a definiált matematikai összefüggéseket algoritmikus formába öntjük, mely alapja lehet a témához kapcsolódó számítógépes algoritmusnak is. Adott $P,S \in \mathbb{R}^3$ (szempont, tárgypont), $P \neq S$ Célszerű bevezetni a következő jelölést: $d := v_1^2 + v_2^2$. Ilyenformán - alakilag kissé módosítva az eddigieket - képezzük az algoritmust. Ha P=S, akkor a feladat értelmetlen, saját szemünkbe nem nézhetünk. A következő számításokat az algoritmus elején, egyszer kell elvégezni, s a kapott értékek mindaddig érvényben maradnak, amíg P és S értékét nem változtatjuk meg. Először kiszámítjuk a képsík középpontjának, a fovea centralis -nak (S_T) koordinátáit (3.5.3.) alapján:

(3.6.1.) $$s_i^T = \frac{15 \cdot (p_i - s_i)}{\| S - P \|} + p_i \qquad (i:=1,2,3)$$

Bevezetjük a következő jelöléseket:

$$v_i := p_i - s_i^T \qquad (i=1,2,3), \quad \text{azaz:} \quad V:=P-S_T$$

Ha $v_1=v_2=0$, azaz $v_1^2+v_2^2=0$ és $v_3 \neq 0$ teljesül: párhuzamos eset, egyébként általános eset. Általános esetben alkalmazzuk (3.4.5.) - ben a c_x, c_y, c_z-re vonatkozó (vi) és (vii) összefüggéseket:

(3.6.2.) $$\begin{cases} d := v_1^2 + v_2^2 \\ e := v_1^2 + v_2^2 + v_3^2 \end{cases}$$

(3.6.3.) $$\begin{cases} c_y := \dfrac{1}{\sqrt{d}} \qquad c_z := \dfrac{1}{\sqrt{e}} \\ c_x := \dfrac{c_y}{c_z} \end{cases}$$

Ezt követhetik a (3.4.7.) részben leírt nagyítási és felbontási transzformációk:

$$c_x := c_x \cdot l_x \cdot h_x \qquad c_y := c_y \cdot l_y$$

Ezután a (3.4.5.)-beli (x) alapján felírjuk a következő transzformációs együtthatókat, figyelembe véve: 'd' tényezőt is.

$$
(3.6.4.)\quad \begin{cases}
x_2 := v_1 \cdot c_x \qquad x_1 := -v_2 \cdot c_x \qquad x_o := (s_1 \cdot v_2 - s_2 \cdot v_1) \cdot c_x \\
y_3 := d \cdot c_y \qquad y_2 := -v_2 \cdot v_3 \cdot c_y \qquad y_1 := -v_1 \cdot v_3 \cdot c_y \\
y_o := [(s_1 \cdot v_1 + s_2 \cdot v_2) \cdot v_3 - d \cdot s_3] \cdot c_y \\
z_3 := -v_3 \cdot c_z \qquad z_2 := -v_2 \cdot c_z \qquad z_1 := -v_1 \cdot c_z \\
z_o := (v_1 \cdot p_1 + v_2 \cdot p_2 + v_3 \cdot p_3) \cdot c_z \\
\text{Mindenütt:} \quad v_i := p_i - s_i \quad (i := 1,2,3)
\end{cases}
$$

Ekkor egy tetszőleges $Q := (q_1, q_2, q_3)$ pontra írjuk fel a (3.4.5.)-beli (xi) egyenletrendszert, amelyben ezek a transzformációs együtthatók szerepelnek:

$$
(3.6.5.)\quad \begin{cases}
q_1^T := \dfrac{x_2 q_2 + x_1 q_1 + x_o}{z_3 q_3 + z_2 q_2 + z_1 q_1 + z_o} \\
q_2^T := \dfrac{y_3 q_3 + y_2 q_2 + y_1 q_1 + y_o}{z_3 q_3 + z_2 q_2 + z_1 q_1 + z_o}
\end{cases}
$$

Ez az egyenletrendszer definiálja a perspektivikus transzformációt, azaz megadja egy tetszőleges térbeli Q pontnak megfelelő Q^T pont koordinátáit a képsíkon: $T: Q(q_1, q_2, q_3) \rightarrow Q^T(q_1^T, q_2^T)$. Az itt leírt algoritmussal még konkrétabb formában, Turbo-Pascal nyelven találkozhatunk az (5.7.) részben.

4. A LÁTÁS MATEMATIKAI MODELLJE (2): takarás, tónus, fény és árnyék, valamint kiegészítő transzformációk.

4.1. *A láthatósági szögtartomány analízise*

Tekintsünk egy tetszőleges konvex testet. Vizsgáljuk meg a láthatóság szempontjából, azaz: a test mely részeit látjuk ? Először csupán logikai szemszögből vizsgáljuk meg ezt a kérdést, majd írjuk fel rá a matematikai modellt. Egy olyan módszert mutatunk most, amely algoritmikusan könnyen megfogható, másrészt igen szemléletes. Mi szükséges ahhoz, hogy a test egy bizonyos pontja látható legyen ? Először is - az átlátszó testeket kivéve - , hogy a pont az objektum felszínén legyen. A felszínt azonban egy bizonyos szemszögből nézve két részre oszthatjuk, látható és láthatatlan részre, egyszerűen szólva a szemlélő szempontjából elülső és hátsó részekre. Ez szemléletes, de matematikailag nehezen megfogható, pontosítsuk tehát ezt a képet. Vizsgáljuk tovább az adott pontot, húzzunk a testhez érintősíkot ezen a ponton. Ezt megtehetjük. hiszen a test konvex. E síkra állítsunk egy merőleges vektort, melynek iránya a testből kifelé mutasson; azaz így a testen a vektor nem halad át. Induljon ki ez a vektor az érintési pontból. Ezzel megkonstruáltuk a testfelszín egy adott pontjának normálvektorát. Ezt a tárgypontot akkor láthatjuk, ha a rajta átmenő normálvektor $-90°$-tól $90°$ -ig terjedő szöget zár be a szem-tárgy vektorral. Láthatjuk, mert ez *szükséges, de nem elégséges* feltétel. Ugyanis ez a feltétel csak az adott pontra vonatkozik, de nem veszi figyelembe a test többi részét és a többi testet. Így előfordulhat, hogy a pont - önmagából eredően - látható lenne, de pl. más testek eltakarhatják. Ezt a feltételt ezért nevezzük a láthatóság lokális feltételének, s mivel ezt a szem-tárgyvektor és a leírt normálvektor szöge - amelyet jelöljünk α-val - határozza meg, az említett szögtartományt nevezzük lokális láthatósági szögtartománynak és jelöljük Ω -val. Tehát:

$$\Omega := \{\ \alpha \mid\ -90° \leq \alpha \leq 90°\ \}$$

Ha a vizsgált ponthoz tartozó α szög ebben a szögtartományban van, az azt jelenti, hogy a pont teljesít egy láthatósági feltételt. Ahhoz, hogy biztosan tudjuk, hogy látható, más feltételeket is teljesítenie kell. Ha a vizsgált ponthoz tartozó α szög nincs benne az Ω szögtartományban, az azt jelenti, hogy a pont *nem látható*. Ez már egy biztos információ, mert az ilyen pontokat

nem kell újabb vizsgálatoknak alávetni, azaz a felszín pontjainak átlagosan a feléről már ezzel a módszerrel kiderül, hogy nem látható. Mivel erre igen gyors számítógépes algoritmus készíthető, a módszernek mindenképpen nagy jelentősége van, kiváltképpen, ha figyelembe vesszük, hogy konvex testeket viszonylag nagy távolságból nézve ez a kritérium elegendő a láthatóság eldöntésére, ha más testek zavaró, takarási hatása nem érvényesül. Az 'lokális' elnevezés azt takarja, hogy a pont saját tulajdonságai alapján eldöntjük, látható-e, de az még nem biztos, hogy teljesíti a globális láthatósági feltételt is, azaz más pontok nem takarják-e el. A takarás feltételeit később még definiáljuk. Egy pont csak akkor látható, ha mind a két feltételt teljesíti.

A 3. részben leírtaknak megfelelően a test felszínét az őt burkoló poligonokkal közelítjük. Így elegendő minden poligonra egyszer megvizsgálni ezt a kérdést, azaz definiálunk a poligon valamely pontjához egy normálvektort és megvizsgáljuk annak szögét a szem-tárgy vektorral. Ezeknek a vizsgálatoknak más helyen is nagy jelentőségük van. Ha a poligon látható, akkor a megvilágítása és fényessége a fényforrással és a szemmel alkotott szögétől is nagymértékben függ, tehát a tónusok és a fény-árnyék hatások területén is alkalmazható az alább leírt elmélet. Ezeket az alkalmazásokat majd későbbi fejezetekben taglaljuk. Végezzük el tehát a láthatósági tartomány analízisét az analitikus geometria eszközeivel.

4.1.1. *A poligon síkjának egyenlete*

Tekintsük a test felszínét közelítő poligonok egyikét. Legyen ennek három csúcspontja: A, B és C. Ha a test közelítése elég pontos, akkor ezek a pontok is közelítőleg rajta vannak a testen, sőt a közelítés pontosságát növelve határértékben maguk is a test pontjai lesznek. Tehát úgy is mondhatjuk, hogy A,B és C a testet közelítő háromszögek egy reprezentánsa. Az A,B,C pontokon átmenő sík egyenlete:

$$(4.1.1.1.) \qquad \begin{vmatrix} x - a_1 & y - a_2 & z - a_3 \\ b_1 - a_1 & b_2 - a_2 & b_3 - a_3 \\ c_1 - a_1 & c_2 - a_2 & c_3 - a_3 \end{vmatrix} = 0$$

azaz kifejtve:

(4.1.1.2.) $[(b_2-a_2)\cdot(c_3-a_3)-(b_3-a_3)\cdot(c_2-a_2)]\cdot x +$
$[(b_3-a_3)\cdot(c_1-a_1)-(b_1-a_1)\cdot(c_3-a_3)]\cdot y +$
$[(b_1-a_1)\cdot(c_2-a_2)-(b_2-a_2)\cdot(c_1-a_1)]\cdot z + const = 0$

Fejezzük ki ezt egyszerű alakban, behelyettesítve az előző egyenletbe:

(4.1.1.3.) $K\cdot x + L\cdot y + M\cdot z + N = 0$

4.1.2. *Az A,B,C háromszög és a szem-tárgy vektor szöge*

Jelöljük ezt a szöget β -val. Az A,B,C háromszögnek azt az oldalát vizsgáljuk, amely a testfelszín közelítését alkotja. Ennek és a $\overrightarrow{PS}$ vektornak a (β) szögére felírhatjuk:

(4.1.2.1.) $$\sin(\beta) = \frac{K\cdot(s_1 - p_1) + L\cdot(s_2 - p_2) + M\cdot(s_3 - p_3)}{\sqrt{D}}$$

ahol K,L,M: a (4.1.1.3.) egyenletben lett definiálva, D pedig egy pozitív konstans, értéke egyelőre nem lényeges. Az viszont lényeges körülmény, hogy $\sin(\beta) \geq 0$ mikor teljesül. (4.1.2.1.)-ből következően - mivel: $D > 0$ - a következőt mondhatjuk: ($\Leftrightarrow$ jelentése: akkor és csak akkor)

$\sin(\beta) \geq 0 \Leftrightarrow K\cdot(s_1-p_1) + L\cdot(s_2-p_2) + M\cdot(s_3-p_3) \geq 0$,azaz:

(4.1.1.2.)

$\sin(\beta) \geq 0 \Leftrightarrow K\cdot s_1 + L\cdot s_2 + M\cdot s_3 \geq K\cdot p_1 + L\cdot p_2 + M\cdot p_3$

4.1.3. *A láthatósági tartomány meghatározása*

Mint már említettük, α: a normálvektor és a szem-tárgy vektor szöge, azaz most az ABC háromszög egy normálvektorát vegyük, mely tehát 90° -os szöget zár be a háromszög síkjával, ezért:

(4.1.3.1.) $\alpha = 90^\circ - \beta$ ezért: $\sin(\beta) = \cos(\alpha)$

Funkciójából adódóan α értelmezési tartománya:

$D : \{ \alpha \mid -180° \leq \alpha \leq 180° \}$

Írjuk fel Ω definícióját és a cosinus függvénynek a 9. ábrán is szemléltetett tulajdonsága, valamint (4.1.3.1.) alapján:

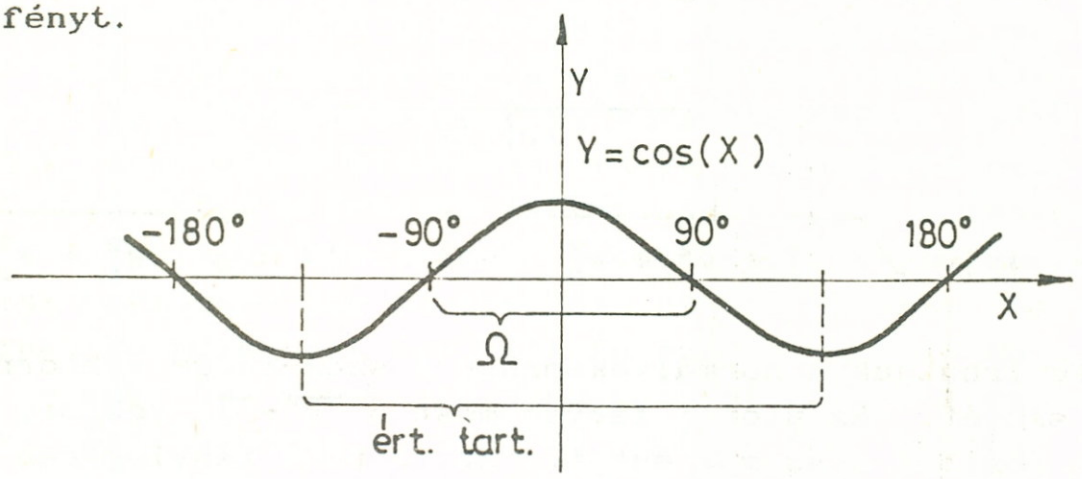
$\Omega = \{\alpha \mid -90° \leq \alpha \leq 90° \} = \{\alpha \mid \cos(\alpha) \geq 0 \} = \{\alpha \mid \sin\beta \geq 0 \} =$

$= \{ \alpha \mid$ teljesül: (4.1.1.2.)$\}$

Ennek alapján egy test felszínének ABC -részlete a lokális láthatóság kritériumát teljesíti, ha:

(4.1.3.2.) $\quad K \cdot s_1 + L \cdot s_2 + M \cdot s_3 \geq K \cdot p_1 + L \cdot p_2 + M \cdot p_3$

Ez könnyen meghatározható, egyszerű feltétel. Nézzük meg ezt a normálvektorok szempontjából is. Érdekes összefüggésekre derít-hetünk fényt.



9. ábra

4.1.4. *Az ABC normálvektor definiálása*

Legyen $\overrightarrow{N}$ az ABC háromszöghöz tartozó, a testből kifelé mutató normálvektor. $\overrightarrow{N}$: (n_1, n_2, n_3) Ezt úgy határozhatjuk meg, mint a háromszög két oldalára merőleges vektort:

$\overrightarrow{N} \perp \overrightarrow{B-A}$ és $\overrightarrow{N} \perp \overrightarrow{C-A}$

Írjuk fel az i, j, k egységvektorok segítségével $\overrightarrow{N}$ egyenletét:

(4.1.4.1.) $$N = (n_1, n_2, n_3) = \begin{vmatrix} i & j & k \\ c_1 - a_1 & c_2 - a_2 & c_3 - a_3 \\ b_1 - a_1 & b_2 - a_2 & b_3 - a_3 \end{vmatrix}$$

Ezt kifejtve, megkaphatjuk a normálvektor koordinátáit:

$$(4.1.4.2.) \qquad \begin{cases} n_1 = (b_2-a_2)\cdot(c_3-a_3)-(b_3-a_3)\cdot(c_2-a_2) \\ n_2 = (b_3-a_3)\cdot(c_1-a_1)-(b_1-a_1)\cdot(c_3-a_3) \\ n_3 = (b_1-a_1)\cdot(c_2-a_2)-(b_2-a_2)\cdot(c_1-a_1) \end{cases}$$

Azt láthatjuk, hogy $(n_1,n_2,n_3) = (K,L,M)$ mely értékek a (4.1.1.3.) egyenletben már szerepeltek.

4.1.5. *Vektorok szöge, a normálvektor és a szem-tárgy vektor szöge*

Adott két vektor: $\overrightarrow{V}=(v_1,v_2,v_3)$ és $\overrightarrow{W}=(w_1,w_2,w_3)$, ezeknek szögére felírhatjuk: (legyen a szögük: α)

$$(4.1.5.1.) \qquad \cos(\alpha)= \frac{v_1w_1+ v_2w_2+ v_3w_3}{|V|\cdot|W|}$$

ahol: $|V|= \sqrt{v_1^2 + v_2^2 + v_3^2}$ és $|W|= \sqrt{w_1^2 + w_2^2 + w_3^2}$

Így felírhatjuk a normálvektornak a szem-tárgy vektorral alkotott (α) szögét. Ez utóbbi legyen most a $\overrightarrow{P}-\overrightarrow{A}$ vektor, mely a tárgyalt A pontból a szembe mutat. Az A pont ténylegesen reprezentálhatja az ABC háromszöget, hiszen ha A látható (eleget tesz a lokális feltételnek és nem takart), valamint a háromszög nem takart, akkor az egész háromszög látható. A lokális láthatóság kérdését egy poligon esetében tehát annak bármely pontja alapján eldönthetjük.

$$(4.1.5.2.) \qquad \cos(\alpha)= \frac{n_1(p_1-a_1) + n_2(p_2-a_2) + n_3(p_3-a_3)}{|N|\cdot|P-A|}$$

Ennek alapján Ω -t a következőképpen határozhatjuk meg:

(4.1.5.3.)

$$\Omega = \{\alpha|\cos\alpha \geq 0\} = \{\alpha|\ n_1\cdot(p_1-a_1)+n_2\cdot(p_2-a_2)+n_3\cdot(p_3-a_3) \geq 0\}$$

Eszerint egy poligon lokálisan látható, ha normálvektorának és a szem-poligon vektornak a *skaláris szorzata pozitív.*

Ezt a kritériumot használjuk a lokális láthatóság eldöntésére. Módszerünk tehát a következő. A test egy burkoló poligonjának tetszőleges A,B,C csúcspontjai koordinátáival kiszámítjuk a poligon normálvektorát (4.1.4.2.) alapján. Ezután (4.1.5.3.) alapján eldöntjük, hogy a poligon teljesíti-e a lokális láthatóság feltételét. Ha nem, akkor a poligon nem látható, ha igen, akkor - ha a globális láthatósági, később megfogalmazott feltételeket is teljesíti - látható.

Ez a módszer könnyen algoritmizálható. Az (5.6.) részben láthatunk számítógépes, Turbo-Pascal nyelvű eljárásokat, amelyek közül az egyik az adott A,B,C pontok segítségével a normálvektort kiszámítja, a másik egyszerű eljárás a láthatóságot elemzi.

Említettük, hogy a testeket a háttértárban a burkoló poligonok csúcspontjaival tárolhatjuk. Érdemes minden poligonnál a három első csúcsot választani A,B,C-nek, ez meggyorsítja az algoritmust, de egy fontos körülményt meg kell említeni ezzel kapcsolatban. A (4.1.4.2.) egyenletrendszerben az A és B pontokat ha felcseréljük, láthatjuk, hogy éppen ellenkező irányú normálvektorokat kapunk. Ez azt jelenti, hogy a lokális láthatóságot éppen ellenkezően mutatná, azaz a poligon csúcsainak azonos körüljárási sorrendben kell követniük egymást. Még pontosabban ez azt jelenti, hogy a poligon csúcsainak koordinátáit úgy kell felsorolni, hogy ha a testen kívülről nézzük, az *óramutató járásával ellenkező* irányban kövessék egymást. Így a normálvektor is kifelé fog mutatni a testből.

Erre nézzünk egy igen egyszerű példát. Legyen A:(0,0,0); B:(1,0,0) és C:(0,1,0). A testet kívülről, a P:(0,0,10); pontból nézzük, tehát az A,B,C pontok az óramutató járásával ellenkezően követik egymást. Ekkor a normálvektor koordinátái (4.1.4.2.) alapján: N:(0,0,1), azaz a normálvektor felénk mutat. Mivel a (4.1.5.3.) -beli $n_1 \cdot (p_1 - a_1) + n_2 \cdot (p_2 - a_2) + n_3 \cdot (p_3 - a_3) \geq 0$ (jelen esetben = 10), ezért a poligon lokálisan látható. Ha felcserélnénk pl. a B és C pontokat, akkor N: (0,0,-1) normálvektort és (4.1.5.3.)- ben -10 értéket kapunk, tehát akkor a poligon nem látható. Vigyázni kell tehát az említett azonos körüljárási irányra!

4.2. A takarás elmélete

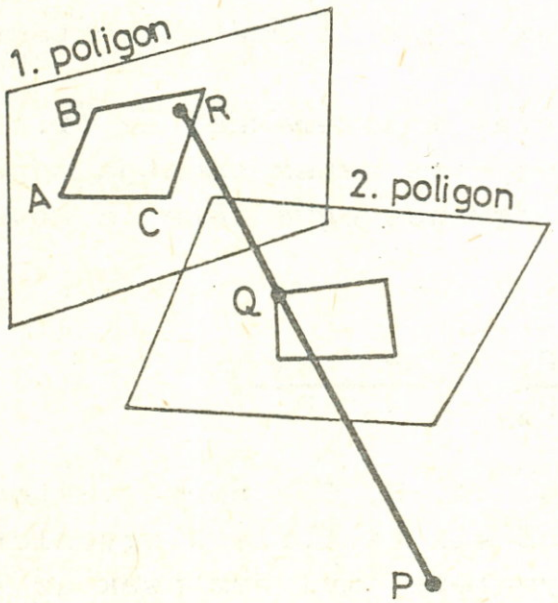
Eddigi vizsgálódásaink során nem használtuk ki azt, hogy a poligonok hány testet közelítenek, mert az alkalmazott képletek a testek számától függetlenek voltak. Ezen az úton haladunk tovább most is. A különbség az, hogy az eddigi vizsgálódásainkban a testek felületét elemeire bontottuk és azokat analizáltuk, most ezeknek a részeknek az egymásra hatását vizsgáljuk. Fokozatosan közeledünk az objektum - sőt, objektumok rendszere - mint egységes egész felé. A perspektív leképezésekre kapott összefüggések még pontokra vonatkoztak, a lokális láthatóság már poligonokra, a takarás pedig objektumokra vonatkozik. Ezért a most megfogalmazott feltételeket globális feltételeknek nevezzük. A takarás nemcsak az objektumok egymásra hatásának egy fontos jelensége, hanem pl. egy konkáv test egyik része is takarhatja egy másik részét. Ez nagyon fontos tényező a látott kép végleges kialakítása szempontjából.

Az előző részben leírt lokális láthatóság elegendő kritérium a takarás megítélésére, ha konvex testről van szó, vagy pedig több konvex, egymást páronként nem takaró objektumról. Ha azonban ez nem teljesül, akkor ez a módszer csak arra jó, hogy a felszínt közelítő poligonhalmazból a nem láthatóakat - ez átlagosan az összes poligon felét jelenti - kiszűrjük. Ez gyors módszer a továbbiakhoz képest, mellyel már 'csak' a maradékot kell figyelembe vennünk. A továbbiakban definiált algoritmus lényege, hogy két, a térben általánosan elhelyezkedő poligonról el tudjuk dönteni, hogy melyik takarja a másikat, majd ennek alapján a két poligont a takarással együtt is ábrázoljuk. A testeket, mint az őket közelítő poligonok halmazát tekintve, e poligonok mindegyikére elvégezzük e műveletet, így mindről tudjuk, melyik takarja a másikat. Ezután a poligonokat sorra ábrázoljuk; először azokat, amelyek nem takarnak egyet sem a poligonok közül, majd ezeket logikailag töröljük és ezt a gondolatmenetet folytatjuk, mindaddig, amíg el nem fogynak a poligonok. Tehát az eljárás során a logikailag nem törölt poligonok közül mindig azt ábrázoljuk, amely nem takar egy poligont sem (a logikailag érvényesek közül). Az ábrázolás módszere olyan, hogy az utoljára ábrázolt poligon mindent eltakar, ami alatta van, tehát ez automatikusan megoldja a vágás problémáját is. A takart poligonokból így tényleg csak annyi fog látszani, amely nincs eltakarva. Ezt a módszert segíti egy poligonfestő algoritmus, amellyel a későbbiekben ismerkedünk meg. E számítástechnikai módszer mellett most is ekzakt analítikus geometriai összefüggéseket nyerünk a vizsgált problémák leírására.

4.2.1. *Két általános helyzetű poligon takarási analízise*

Definiáljuk, mit értünk azon, hogy két poligon takarja egymást. Sokféleképpen közelíthetünk e kérdéshez, de most fogalmazzuk meg úgy ezt a kérdést, hogy a későbbiek során leghasznosabb definícióját adjuk ennek a fogalomnak. Eszerint két, a térben általánosan elhelyezkedő poligon takarja egymást, ha perspektív képeik fedik egymást, azaz létezik nem nullmértékű közös részük. Ez a közös rész is egy poligon, mely az ábrázoláskor már csak az egyik poligonnak lehet része: annak, amely takarja a másikat. Így a két poligon akkor és csak akkor takarja egymást, ha *legalább az egyiknek létezik legalább egy olyan csúcspontja, amelynek perspektív képe a másik poligon perspektív képére (annak belsejébe) esik.* Ezt az utóbbi takarási definíciót fogjuk felhasználni annak eldöntésére, hogy két poligon takarja-e egymást. A takarásnak van egy bizonyos függetlensége a perspektív leképezéstől, éspedig az, hogy a vizsgált objektumokon kívül csak a szem térbeli elhelyezkedésétől függ, de a nézett ponttól független.

A következőkben kritériumot adunk annak eldöntésére, hogy két poligon milyen a takarás szempontjából, hogy viszonyul egymáshoz. Tekintsük az (1.) és a (2.) poligonokat, az (1.) poligon három csúcsa legyen: A:(a_1,a_2,a_3), B:(b_1,b_2,b_3) és C:(c_1,c_2,c_3), a szem pedig: P:(p_1,p_2,p_3). Most egy számítástechnikai módszerrel lépünk tovább; a poligonfestési eljárással. Ennek lényege számunkra most az, hogy a képernyőn bármely poligont egy adott színnel ki tudunk festeni. A következő fejezetben - mely a számítástechnika szemszögéből foglalkozik ezzel a kérdéssel - részletesen leírjuk ezen algoritmus hátterét, ezért ne menjünk bele ennek részleteibe. A képernyőn csak az (1.) poligon legyen ábrázolva, ennek pontjai a képernyőt leíró memóriaterületen más, jól elkülöníthető jellel vannak vannak ábrázolva, mint a poligonon kívüleső pontok, éspedig a



10. ábra

poligon belső pontjai '11','01' vagy '10' értéket kapnak (az ábrázolt színnek megfelelően), a poligonon kívüleső pontok pedig '00' értékűek. Tehát egy pontról egyértelműen eldönthetjük, belső pontja-e egy poligonnak. Felrajzoljuk tehát a képernyőre az (1.) poligon perspektív képét. A (2.) poligon csúcsaira sorra meghatározzuk, hogy perspektív képük (1.) perspektív képére esik-e. Ha olyat találtunk, amely e poligon belsejére esne, már nem is kell folytatnunk az eljárást. Ha nincs ilyen, minden csúcsponton végig kell mennünk, sőt, (1.) és (2.) szerepét felcserélve is meg kell ismételnünk az eljárást. Ez tehát kétféleképpen fejeződhet be. Ha mindkét poligonra igaz, hogy csúcspontjainak perspektív képe közül egy sem esik a másik poligon perspektív képére, akkor a két poligon nem takarja egymást. Ha viszont találunk ilyen csúcspontot, akkor a definíció értelmében a két poligon takarja egymást. Az előző esetben a poligonokkal nincs több gondunk, az utóbbiban azt kell meghatároznunk, melyik takarja a másikat.

Legyen ez esetben a talált csúcspont neve: Q: (q_1,q_2,q_3), melynek képe az (1.) síkján legyen R: (r_1,r_2,r_3). Ez nem más, mint a P és Q pontokon átmenő egyenes és az (1.) poligonon átmenő sík metszéspontja. *Tehát 'R' az (1.), 'Q' a (2.) poligon síkjában van.* Mármost ha $\overline{PQ} > \overline{PR}$, akkor (1.) takarja (2.)-t, ha $\overline{PQ} < \overline{PR}$, akkor (2.) takarja (1.)-et. Ha PQ = PR, akkor (2.)-nek minden pontja (1.) síkjának azonos oldalán van; ez esetben legyen Q: (2.) tetszőleges pontja, R: az (1.) síkjának és a P,Q pontokon átmenő egyenesnek metszéspontja. Ha erre (és még egy másik tetszőleges pontra is) $\overline{PQ} = \overline{PR}$, akkor a két poligon egy síkban van, ha nem, akkor az előző feltételt alkalmazhatjuk a szétválasztásra.

A továbbiakban analítikus geometriai módszerekkel az (1.)-en átmenő sík és a P,Q pontokon átmenő egyenes metszéspontja adja R koordinátáit. A P,Q pontokon átmenő egyenes egyenlete a következő:

(4.2.1.1.) $$\frac{x - p_1}{q_1 - p_1} = \frac{y - p_2}{q_2 - p_2} = \frac{z - p_3}{q_3 - p_3}$$

Az (1.) poligon három csúcspontja: A, B, C, normálvektora: N: (n_1,n_2,n_3). Az A,B,C pontokon átmenő sík (4.1.1.1.) egyenletével ekvivalens a sík normálvektoros egyenlete, melyhez csak az 'A' pontot kell felhasználni:

(4.2.1.2.) $$n_1 \cdot (x-a_1) + n_2 \cdot (y-a_2) + n_3 \cdot (z-a_3) = 0$$

R koordinátáinak meghatározásához vezessük be a következő jelölést:

$$(4.2.1.3.) \qquad D := \frac{n_1\cdot(p_1-a_1) + n_2\cdot(p_2-a_2) + n_3\cdot(p_3-a_3)}{n_1\cdot(q_1-p_1) + n_2\cdot(q_2-p_2) + n_3\cdot(q_3-p_3)}$$

Ha ennek nevezője nulla, akkor a $\overline{PQ}$ egyenes párhuzamos (1.) síkjával. Ez esetben a kettőnek nincs metszéspontja. Ez azonban ellentmond Q konstrukciójának. Q perspektív képe ugyanis rajta van az (1.) poligon perspektív képén; ez csakis akkor lehet, ha a P,Q pontokon átmenő egyenes metszi az (1.) poligont. Ez magából a perspektív transzformáció konstrukciójából következik. Ha ezenfelül (4.2.1.3.) -ben a számláló is nulla, akkor az egyenes benne van a síkban. Ez esetben az (1.) poligon síkja átmegy a P szemponton, azaz a síkot - s így az (1.) poligont - élével látjuk. Mivel a síknak oldalirányú kiterjedése nincs, az (1.) poligont ez esetben nem is kell ábrázolni.

Maradjunk tehát az általános esetnél, amikor (4.2.1.3.) -nak sem a nevezője, sem a számlálója nem nulla. Így 'D' felhasználásával 'R' koordinátái a következők:

$$(4.2.1.4.) \qquad r_i = p_i - (q_i - p_i)\cdot D \qquad \text{ahol: } i=1,2,3$$

A fentebb definiált szétválasztási feltételt fogalmazzuk át számítástechnikailag könnyebben kezelhető formájúra: ha $\overline{PQ}^2 > \overline{PR}^2$, akkor (1.) takarja (2.)-t, ha $\overline{PQ}^2 < \overline{PR}^2$, akkor (2.) takarja (1.)-et. Ez ekvivalens a fentebb megfogalmazott feltétellel, mivel minden szakasz hossza pozitív. Részletesen is felírjuk az itt szereplő mennyiségeket, kifejtve az r_i -értékeket is:

$$\overline{PQ}^2 = (p_1-q_1)^2 + (p_2-q_2)^2 + (p_3-q_3)^2$$

$$\overline{PR}^2 = (p_1-r_1)^2 + (p_2-r_2)^2 + (p_3-r_3)^2 =$$

$$= (q_1-p_1)^2\cdot D^2 + (q_2-p_2)^2\cdot D^2 + (q_3-p_3)^2\cdot D^2 = D^2\cdot\overline{PQ}^2$$

$\overline{PQ}^2 > \overline{PR}^2$ így ekvivalens $1 > D^2$ feltétellel, ez pedig az $1 > |D|$ feltétellel. Összefoglalva: ha $1 > |D|$, akkor (1.) takarja (2.)-t, ha $1 < |D|$, akkor (2.) takarja (1.)-et. $1 = |D|$ esetén a fent leírtak szerint újabb Q ponto(ka)t kell választani. A 'Q' pontot számítástechnikai módszerrel választottuk ki, de minden számítást analítikus geometriai módon végeztünk el, ezért egy ekzakt módszerhez jutottunk.

4.2.2. Tetszőleges sok poligon takarási analízise

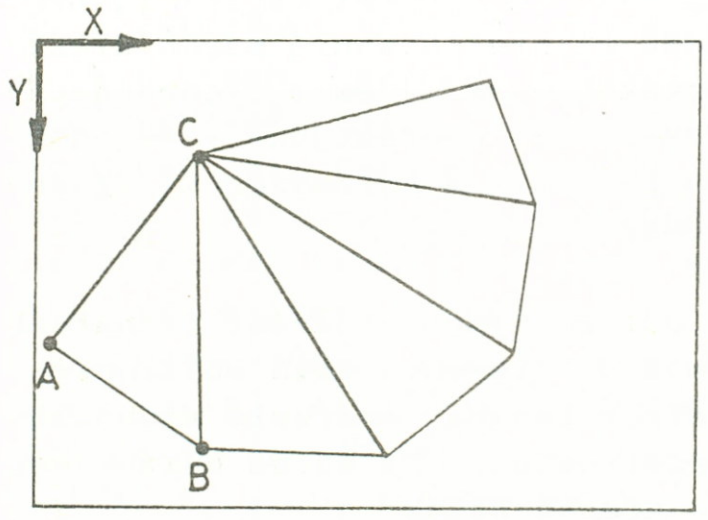
Az előző részben elvégeztük két poligon takarási analízisét. Ha csak ezt a két poligont kell ábrázolnunk, akkor előbb ábrázoljuk a takart poligont, utána a másikat. Mivel poligonfestő algoritmussal dolgozunk, ez a következő poligont attól függetlenül befesti, hogy annak területén eredetileg mi volt. Ezért ez helyettesíti a vágást; nem beszélve arról, hogy lényegesen gyorsabb, mint a vágási algoritmusok.

A (4.2.1.) algoritmussal páronként meghatározhatjuk, mely poligonok takarják egymást. Ez az algoritmus n^2-n -szeri alkalmazását jelenti, ahol n: a lokálisan látható poligonok száma. Így láthatóan lényeges szempont, hogy a poligonok számát kb. felére csökkentettük a (4.1.)-beli módszerrel. A (4.2.1.) algoritmust tehát minden poligon-párra meghívjuk, kezdjük az elsőn, ahhoz sorban hozzárendeljük a többit, de nem cseréljük fel a poligonokat. A (4.2.1.) algoritmust két poligon esetén ugyanis a poligonokat felcserélve is végre kell hajtani, de ez úgyis megtörténik, ha a második poligonhoz érünk az eljárás során. Így minden poligont csak egyszer kell ábrázolni az eljárás során, majd a többivel összevetni, végül pedig - ismét a poligonfestő algoritmussal, háttérszínnel - kifesteni, vagy képernyőt törölni. Közben, ahol takarást találtunk, azt rögzítsük egy referencia-listán. Ez lehet pl. egy file, amelynek egy rekordjára felírjuk a takaró poligonok sorszámát, vagy kezdetüknek rekordszámát. A referencia-lista egy rekordján tehát két szám van, melyeket úgy értelmezzük, hogy az 1. sorszámú poligon takarja a 2. sorszámút.

Így az eljárás befejeztével teljes képünk van a poligonok takarási viszonyairól. Az objektumok ábrázolásakor ezt a referencia-listát használjuk. Kikeressük a lista segítségével azokat a poligonokat, amelyek nem takarnak más poligonokat (ilyen legalább egy van, mivel kell lenni 'legalul' is). A referencia-listáról töröljük mindazon rekordokat, ahol az ábrázolt poligonok szerepeltek. Ugyancsak töröljük a poligonok közül azt, amelyiket már ábrázoltunk. A törlés természetesen logikai törlést jelent. Így haladunk végig, a maradék referencia-listából kiválasztva mindig azokat a poligonakat, amelyek nem takarnak semmit. Ezt tesszük, amíg el nem fogy a teljes referencia-lista. Ezután ha még találunk logikailag nem törölt poligont, az nincs takarva egyetlen más poligontól sem (nem szerepeltek a listán a takartak között). Ezeket megjelenítve az objektumo(ka)t teljességgel ábrázoltuk.

4.3. *Poligonfestés*

A poligonfestési algoritmusok célja az, hogy egy tetszőleges síkidomot pl. a számítógép képernyőjén be tudjunk festeni egy megadott színűre. A feladat más szóval azt jelenti, hogy pontosan meg kell tudnunk határozni mindazon pontokat, amelyek a síkidom belsejébe esnek. Ezeket - és csakis ezeket - színezzük be a megadott színűre. Egy általános síkidomot poligonnal tetszőleges mértékben közelíthetünk. Definiálunk tehát egy olyan algoritmust, amellyel egy tetszőleges poligont be tudunk festeni. Legyen a poligon oldalainak száma: n. Jelöljük ki egy tetszőleges csúcspontját, majd ezt kössük össze az összes többivel. Így pontosan n-2 háromszögre vágtuk fel a poligont, mint ahogy egy példán a 11. ábrán láthatjuk. Ilyenformán a feladatot visszavezettük a tetszőleges háromszög festésére, mivel az említett általános poligon befestése n-2 háromszög befestését jelenti. Legyen a háromszög csúcspontja: A:(a_1,a_2), B:(b_1,b_2) és C:(c_1,c_2). A pontok koordinátáit most tekintsük a képernyő-koordináták szerint, mivel a módszerünk elsősorban a képernyőn valósítható meg könnyen. Tegyük fel, hogy teljesül $b_2 \le a_2 \le c_2$, ha nem, akkor a háromszög csúcspontjait átnevezzük. Felírjuk a háromszög oldalain átmenő három egyenes egyenletét:



11. ábra.

$\overline{AB}$ szakasz esetén: $(y-a_2)\cdot(b_1-a_1)=(x-a_1)\cdot(b_2-a_2)$

$\overline{AC}$ szakasz esetén: $(y-a_2)\cdot(c_1-a_1)=(x-a_1)\cdot(c_2-a_2)$

$\overline{BC}$ szakasz esetén: $(y-b_2)\cdot(c_1-b_1)=(x-b_1)\cdot(c_2-b_2)$

A módszerünk lényege a következő: a háromszöget vízszintes csíkokra vágjuk a képernyő sorai szerint. Minden csíknak a háromszög belsejébe eső részét befestjük. Mivel a csíkok szélessége éppen egy képernyő-sor szélességének felel meg, ezért így a háromszög minden belső pontját befestettük.

Így az y: b_2-től a_2-ig, majd a_2-től c_2-ig terjedő értékekre kiszámítjuk a szakaszok megfelelő x-koordinátáit is. Így minden, a háromszögbe eső szakaszt definiáltunk. Ezért a fenti egyenletekből fejezzük ki x -et. A megkülönböztetés miatt indexeljük is az x -értékeket:

$\overline{AB}$ szakasz esetén: $$x_1 = \frac{(y-a_2)\cdot(b_1-a_1)}{b_2-a_2} + a_1$$

$\overline{AC}$ szakasz esetén: $$x_2 = \frac{(y-a_2)\cdot(c_1-a_1)}{c_2-a_2} + a_1$$

$\overline{BC}$ szakasz esetén: $$x_3 = \frac{(y-b_2)\cdot(c_1-b_1)}{c_2-b_2} + b_1$$

Kiszámítjuk ennek alapján $y = b_2$ -tól a_2 -ig terjedő értékeire az x_1 és x_3 koordinátákat, így az (x_1,y)-től (x_3,y)-ig terjedő szakaszokat befestjük a megadott színre. Hasonló módon kiszámítjuk $y = a_2$ -től c_2-ig terjedő értékeire az x_2 és x_3 koordinátákat, így az $\{(x_2,y);(x_3,y)\}$ szakaszokat befestjük a megadott színre. A kapott szakaszok teljesen befestik a háromszöget. Ugyanígy járunk el a poligon összes háromszögével.

Egy ezzel a módszerrel írt poligonfestő eljárást láthatunk az (5.5.) részben. A módszer alkalmas plottereken való poligonfestésre is. Ha ritkán húzzuk meg az algoritmusban szereplő szakaszokat, akkor a poligont csupán becsíkoztuk. Ha sűrűn húzzuk meg őket, akkor itt is elérhető a poligonfestő hatás.

4.4. Tónusozás

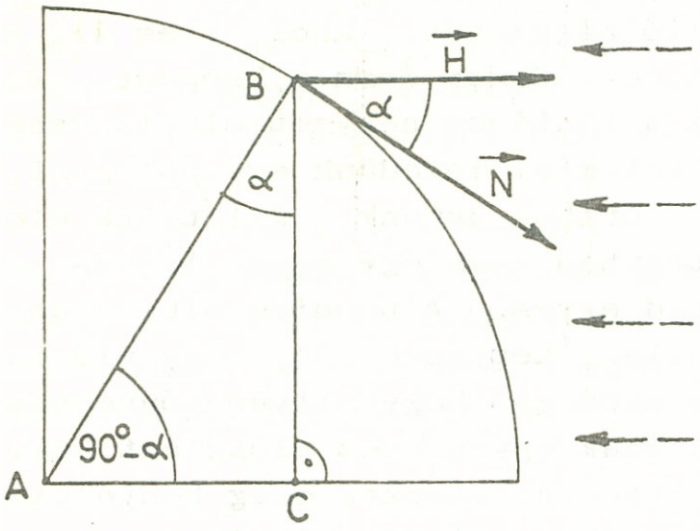
Az alakzatokat a valóságban is különböző színekben és a színek erőssége szerint különböző tónusokban látjuk. Ha a számítógépen a valóság hű mását szeretnénk szimulálni, elengedhetetlen tényező a tónusok alkalmazása. Így a valóságot árnyaltabban tudjuk ábrázolni. Mennél többféle tónust alkalmazunk, annál hitelesebb a kép, amelyet készítünk. Ennek értelmében adott a $t_0, t_1, \ldots t_n$ tónussorozat, a következő módon: t_0: fehér, t_n: egy adott színből a lehető legerőteljesebb, a többi pedig e kettő közötti egyenletes átmenet. Az IBM személyi számítógépeken alapkiépítésben a színek kis száma miatt mindez csak keveréssel érhető el, már viszonylag kis 'n' érték esetén is. Sokkal jobb a helyzet, ha némely rajzgépet tekintjük. A Versatec plotteren pl. igen sokféle tónus előállítható, akár megtervezve, akár véletlenszám-generátor segítségével. A tónust ez esetben a következő módon készítjük el: tekintsünk egy 16x16 -os mátrixot, amelynek minden eleme 0 vagy 1-es. Ezt a mátrixot $16^2=256$ féleképpen tölthetjük ki. Rendeljünk minden kitöltéshez egy-egy tónust, oly módon, hogy a mátrixhoz hozzárendelünk egy 16x16 képpontból álló négyzetet. Ahol 1-es áll a mátrixban, ott fekete, ahol '0' áll, ott fehér lesz a képpont. Az említett raster-plotter esetében ez a 16x16-os négyzet oldala csak 2 milliméter, tehát, ha ilyen négyzetekkel lefedünk egy poligont, akkor valamilyen tónushoz vagy textúrához jutunk. Állítsunk elő 256 tónust, úgy, hogy az első mátrixban csak egy egyes legyen, a többi nulla, az utolsóban pedig mind egyes. A többire általában: az i-edikben i db egyes, a többi nulla. Legyen n=256, így elő is állt a $t_1, \ldots t_n$ tónussorozat. Nem mindegy, hogy milyen tónusokat alkalmazunk, ugyanis a fekete pontoknak egyenletes eloszlást kell biztosítani, ez egyes i-értékekre egyszerű, másokra elég bonyolult feladat. Ha magunk konstruáljuk meg, akkor figyelni kell az egyesek egymástól való távolságát és figyelembe kell venni azt, hogy a minta ismétlődni fog, ezért a határeloszlásokra is gondoljunk. Ez 256 esetben igen hosszadalmas, ezért célszerű a feladathoz véletlenszám-generátort alkalmazni, amely a mátrixba az i-edik esetben véletlenszerűen elhelyez i db egyest. Ha meggeneráljuk mind a 256 tónust, akkor lesznek benne egyenletes és kevésbé folyamatos tónusok. Ez utóbbiakat ismét generáljuk meg, mindaddig, amíg el nem érjük a kívánt minőséget. Végül 256 tónushoz jutunk. Ez már olyan mennyiség, hogy ezekkel szinte a folyamatos átmenet hatását tudjuk érzékeltetni. A módszer IBM személyi számítógépeken is alkalmazható, de a kis felbontás miatt a tónusok nem érvényesülhetnek megfelelően.

4.5. *Fény-árnyék hatások*

Az előző részben a tónusok előállítását írtuk le, most pedig az alkalmazásáról ejtsünk néhány szót. Leginkább a fény-árnyék hatásokat érzékeltethetjük segítségükkel. Azonban a fény és az árnyék - az alább leírt algoritmusok szerint - különböző algoritmusokkal valósítható meg. Ezért tehát külön tárgyaljuk őket.

4.5.1. *A fény és a fényerősség*

Tekintsünk egy egységnyi területet, pl. egy méter oldalú négyzetet. Erre jusson egységnyi fénymennyiség, ha a fénysugárral merőlegesen helyezkedik el. A kérdés az, hogy ha nem merőlegesen helyezkedik el, akkor hányad részét kapja ennek a fénymennyiségnek ? Ez a beesési szög (α) függvénye. Mint azt a 12. ábrán is láthatjuk, a fénymennyiségnek ez esetben $\cos(\alpha)$-szorosát kapja, mivel az ábrán $\overline{BC} = \cos(\alpha)$



12. ábra

Az ábrán az egységnyi oldalú négyzetet ábrázoljuk oldalnézetben, azaz tekintjük a vetületét. Ennek végpontjai: A és B. B-nek a fénysugárra vett merőleges vetülete: C A beesési szög pedig nem más, mint a négyzet normálvektorának (N) és a fénysugár irányú (H) vektornak egymással bezárt szöge. A 'H' vektor iránya a négyzettől a fény felé mutasson, mert a későbbi számítások során ezt az irányt rögzítjük. Hasonló a helyzet, mint amikor a lokális láthatóságot leírtuk. Kézzelfogható ez a hasonlóság, ha elgondoljuk, hogy az alakzatnak az a része kap fényt - eltekintve az árnyéktól - amely a fényforrásból látható. Most az a célunk, hogy definiáljuk, mennyi fényt kap a test akkor, ha nincs árnyékban. Minden poligonra - mely a test felszínét közelíti - ez a kérdés eldönthető, így tehát az egész testre is. Tekintsünk egy - a test felszínét közelítő - poligont. Legyen ennek az egyik csúcsa: A. Ha a fényforrás elég távol van a poligontól, akkor annak minden részét közel azonos mértékben világítja meg. Ezért egy poligon esetén egy fényvektorral számolunk.

Legyen a fényvektor: H (h_1,h_2,h_3), a poligon normálvektora: N (n_1,n_2,n_3). Ekkor az általuk bezárt α szögre a következő összefüggést mondhatjuk el (4.1.5.1.) alapján:

$$(4.5.1.1.) \qquad \cos(\alpha)= \frac{n_1h_1+ n_2h_2+ n_3h_3}{|N|\cdot|H|}$$

ahol: $|N|= \sqrt{n_1^2 + n_2^2 + n_3^2}$ és $|H|= \sqrt{h_1^2 + h_2^2 + h_3^2}$

Ha a fényforrás elég távol van, akkor az egész testre egységesen $\overrightarrow{H}$ vektorral számolhatunk, ha közelebb van és koordinátái: F: (f_1,f_2,f_3), akkor a $\overrightarrow{H}= \overrightarrow{F}- \overrightarrow{A}$ összefüggéssel is számolhatunk. Ennek alapján tehát kiszámíthatjuk a beesési szög cosinusát. Éppen erre van szükségünk a továbbiakban. Ennek alapján a következőket mondhatjuk el. Az egyik megállapításunkhoz akkor juthatunk el, ha a lokális láthatóság megfogalmazásakor leírt gondolatmenetet követjük most is. Eszerint a poligon a test megvilágított oldalán van, ha $\cos(\alpha)\geq 0$, azaz a megvilágítási szögtartomány:

$$(4.5.1.2.) \quad \Psi = \{\alpha|\ \cos(\alpha) \geq 0\} = \{\ \alpha|\ n_1h_1+ n_2h_2+ n_3h_3\geq 0\ \}$$

Ha $\alpha < 0$, akkor a poligon a test árnyékos oldalán található. A másik információ, hogy $\cos(\alpha)$ az előzőek alapján a megvilágítás erősségét jelenti. Így $\alpha = 90°$ (azaz $\cos(\alpha)= 1$) mellett a poligon a legtöbb fényt kapja, azaz ekkor a legerősebb tónussal ábrázoljuk, amely - maradva az előző részben a tónusokról leírtaknál - t_n, hasonlóképpen $\alpha = 0°$ (azaz $\cos(\alpha)= 0$) esetén: t_1. A többi tónust $\cos(\alpha)$ értékétől függően egyenletesen osszuk szét, a következő összefüggés alapján, α beesési szög esetén a poligon az i-edik tónust kapja. Ez jellemzi az adott poligon *fényerősségét*:

$$(4.5.1.3.) \quad i := n\cdot(1-|\cos(\alpha)|)$$

Tehát a poligon az i-edik tónust kapja. Ez az összefüggés teljesíti mindazokat az elvárásokat, amelyeket vele szemben támasztottunk. Cos(α) alapján, egyenletesen változtatja a tónust, így a valóságot szimuláló kép élethűségét nagymértékben megnövelhetjük. Mindez csak akkor igaz, ha a poligon nincs árnyékban, azaz a test fényes oldalán van és más objektumok, illetve a test esetleges konkáv részeinek árnyékoló hatása nem érvényesül rajta. Ez utóbbi kérdést pedig a következő részben elemezzük.

4.5.2. *Az árnyék*

A fény hatásainak vizsgálatát a lokális láthatósággal hoztuk kapcsolatba, hiszen mindkét kérdést hasonlóképpen vizsgáltuk. Az árnyékolás kérdéskörét pedig a (4.2) részben leírt teljes takarási mechanizmussal hozhatjuk rokonságba. Az árnyékolás algoritmusa lényegesen bonyolultabb, mint a fényerősség vizsgálatáé, ezért számítógépes megvalósítása esetén a program futása is lassabb. Így célszerű a módszereket egymás kiegészítésével használni. A (4.5.1) részben leírt módszer az árnyékolás szempontjából egy igen fontos tényezőt hordoz magában. Az objektum felszínét közelítő poligonok (nagy átlagban) kb. feléről kimutatja, hogy a test árnyékos felén helyezkednek el. Árnyékolás szempontjából tehát csak a többi poligont kell megvizsgálnunk, amelyek a test fényes felén helyezkednek el. Ezek azonban nem biztos, hogy meg is vannak világítva, hiszen a test esetleges konkáv részei és más objektumok árnyékoló hatása is érvényesülhet rajtuk. Tehát megvizsgáljuk azokat a poligonokat, amelyekről az előző részben megállapítottuk, hogy a poligon fény felőli részén találhatóak.

Tekintsük a (4.2.) algoritmust. Vegyük észre, hogy az árnyék a takarással rokon fogalom, hiszen a test azon részei vannak árnyékban, amelyek a fény forrása felől nézve takart állapotban vannak. A (4.2.) algoritmust alkalmazzuk úgy, hogy a szem helyére a fényforrást tegyük. Így az algoritmussal eldönthetjük két poligonról, hogy melyik takarja a másikat. Módosítsuk az algoritmust, hiszen itt ez kevés; pontosan meg kell határoznunk, milyen mértékben érvényesül az egyik poligon árnyékoló hatása a másikra.

Tegyük fel, hogy eldöntöttük, két poligon közül az egyik árnyékolja a másikat, pl. (1.) árnyékolja (2.)-t. Ekkor az árnyékolt poligonon pontosan meg tudjuk határozni a másik sokszög árnyákát. A (4.2.)-ben leírt módszer alkalmas arra, hogy megállapítsuk, az árnyékoló poligonnak mely csúcsai esnek az árnyékolt poligon területére, sőt, az említett árnyékpontoknak meg is tudjuk határozni a koordinátáit is. Ugyanis ez esetben is egyenes és sík metszéspontját kell kiszámítani, mely megtehető a (4.2.1.4.) egyenrendszer alapján. Ha a fény párhuzamos a síkkal, azt az esetet is árnyékosnak kell tekinteni. A (4.2.) módszerrel azt is kiszámíthatjuk, hogy a (2.) poligonnak mely csúcspontjait árnyékolja az (1.). A (2.) poligon egy tetszőleges csúcsát ugyanis a fényforrással összekötve a (4.2.) módszer kimutatja, hogy metszi-e a kapott egyenes (1.) területét.

Így (2.) minden csúcsáról eldönthető, hogy (1.) árnyékolja-e vagy sem. (2.) árnyékolt csúcspontjai és (1.)-nek azon csúcsai, amelyek (2.) területére esnek, az árnyékolt (2.) poligon területén egy újabb poligont jelölnek ki. Ennek minden csúcspontja kiszámítható a koordinátája a módszerrel. A (4.2.) módszernek ezen módosítását úgy alkalmazzuk, hogy ha két poligont találunk, amelyek árnyékolják egymást, akkor a módszert és megfordítását egyszerre alkalmazzuk rajta. Így tehát az árnyéknak, mint poligonnak meg tudjuk határozni a csúcsai koordinátáját. Tudjuk, hogy ez az árnyék-poligon (2.) síkjában van.

Az így kapott poligonokat számítógépes megvalósítás esetén raktározzuk el egy file-ba, csúcspontjainak koordinátái segítségével. Egy rekord egy csúcs három koordinátájának feleljen meg. Tüntessük fel a poligont lezáró rekordban, hogy ez a poligon mely poligon síkjába esik. Esetünkben tehát a (2.) poligon sorszámát tüntessük fel. Amikor az ábrázolás során egy poligonhoz érünk, annak megjelenítése után az árnyék-poligonok listáján megnézzük, hogy tartozik-e hozzá egy vagy több rávetülő poligon-árnyék. Ha igen, akkor ábrázoljuk azt is - alkalmazva azon is egyszerűen a perspektív leképezést - az árnyéknak definiált színben. Célszerű ugyanis az árnyékolt részeknek egységes színt adni, a lehető legsötétebb színt és tónust, azaz t_n-et.

Minden poligon-párra meghatároztuk az árnyék hatását, így a teljes objektum - sőt: tetszőleges számú objektum - árnyékolását meg tudjuk oldani ezzel a módszerrel. Ha azt akarjuk ábrázolni, hogy az alakzat egy síkon áll, akkor a síkot, mint egy nagyméretű poligont, vegyük fel a többi sokszög közé. Ezen is tudjuk így érzékeltetni az árnyékokat.

Most már árnyaltan tudjuk ábrázolni a testet. Meg tudjuk jeleníteni árnyékát, folyamatos átmenetet tudunk képezni a világos és sötét részei között. Ha elég nagy felbontással dolgozunk, el tudjuk érni a fénykép minőségét. Valós testek esetén célszerű az alakzat részeire feltüntetni, hogy a valóságban milyen színűek. Ez pedig már a színes fénykép minőségének elérését jelenti !

4.6. Transzformációk az alakzaton és perspektív képén

Az eddig leírt matematikai modell azt a célt szolgálta, hogy a valóságot minél jobban tudjuk ábrázolni és számítógépen szimulálni. Most - mintegy a modell kiegészítéseként - vizsgáljuk meg azt a kérdést, hogy az alakzatot és a perspektív transzformációkkal kapott képét milyen transzformációknak vethetjük alá. Ezek a transzformációk nemcsak a valóság lehető legtökéletesebb ábrázolásában játszanak szerepet, hanem a számítógépes videotechnika alapjait is képezik. Így a statikus ábrázoláson túlmutatva, ezeket a transzformációkat - többek közt - az objektumok dinamikus ábrázolására is felhasználhatjuk, azaz valós idejű mozgást, forgást tudunk bemutatni segítségükkel.

Az objektumok a három dimenziós tér valamely részhalmazában találhatók meg, ezért rájuk a három dimenziós tér transzformációi vonatkoznak. Az alakzatok képét perspektív transzformációval egy síkra képeztük le. Többféle leképezés is lehetséges, amelyet perspektívnak nevezhetünk, pl. a képsík helyétől vagy a vetítés típusától függően. Az így keletkező képeket is transzformálhatjuk, rájuk a sík kétdimenziós transzformációi vonatkoznak, amelyek az alkalmazott vetítés típusától függetlenek. Általánosan is igaz, hogy a most bemutatott transzformációk az alakzattól függetlenül vizsgálhatók. Először az alakzat képének kétdimenziós transzformációit vizsgáljuk meg, majd pedig magán a három dimenziós testen végrehajtható háromdimenziós transzformációkat. Ezek a tér, illetve a sík egy-egy pontjára vonatkoznak. Egy poligon transzformációjához elegendő a csúcspontjait transzformálni, mivel ezeket összekötve hozzájutunk a transzformált poligonhoz. Hasonlóképpen egy poliéder transzformációját a felszínét alkotó poligonok transzformációira vezethetjük vissza.

Az alakzatokat a felszínüket közelítő poligonokkal írtuk le. Ezért a leírt módszerrel két és három dimenziós alakzatokat tudunk transzformálni. A következőkben tehát a két és három dimenziós terek pontjaira vonatkozó transzformációkat részletezzük. Ezek általában 3x3 -as, illetve 4x4 -es mátrixokkal írhatók le. A mátrixműveleteket ismerteknek tételezzük fel, ezért azok bemutatása most nem célunk.

4.6.1. *Transzformációk két dimenzióban*

A kétdimenziós transzformációk jelentős részének közös tulajdonsága, hogy egy egy 3x3 -as mátrix segítségével egységesen jellemezhetjük őket. Jelöljünk általánosan t-vel egy kétdimenziós transzformációt, mely lehet eltolás, nagyítás, forgatás, stb. Ekkor a sík egy tetszőleges $R:(r_1,r_2)$ pontját 't' az $R_t:(r_1^t,r_2^t)$ pontba viszi át: $R_t = t \cdot R$ alapján. 't' általános alakja:

(i) $$t = \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

A mátrix harmadik oszlopa két egyesből és egy nullából áll, minden transzformáció esetén, ez gyorsítja az alkalmazást.

Ahhoz, hogy a transzformációkat egységesen, ebben az alakban kezelhessük, a pontok koordinátáit egészítsük ki egy fiktív, harmadik koordinátával, amelyik azonosan: 1. Így a transzformációk általános alakja a következő:

(ii) $$(r_1^t,r_2^t,1) = (r_1,r_2,1)\begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

A transzformációkat ebben a szerkezetben tárgyaljuk. Előnye, hogy két transzformáció kompozícióját is könnyen meg tudjuk vele fogalmazni. Ha pl. egymás után alkalmazunk egy 't_e' eltolást és egy 't_f' elforgatást, akkor a két transzformáció kompzíciója: $t = t_e \cdot t_f$, azaz a két mátrix szorzata, mely szintén (i) alakú mátrix lesz. A transzformációk sorozatának kompozíciója is (i) alakú mátrix lesz, azaz a 3. oszlopát két nulla és két egyes fogja alkotni.

Összetett transzformációkat előbb elemi transzformációk sorozatára bontunk, ezek mátrixait definiáljuk, majd összeszorozzuk őket. Ez lesz az összetett transzformáció mátrixa. Így pl. egy Q pont körüli α szögű elforgatás három transzformáció kompozíciójára bontható: 1: eltolás Q-ba, 2: elforgatás α szöggel, 3: eltolás, vissza az origóba. Tekintsük át részletesen az elemi transzformációkat.

4.6.1.1. *Eltolás*

Az adott 'R' pontot x irányban e_1-vel, y irányban e_2-vel eltoljuk. Másként fogalmazva, a következő eltolási vektort alkalmazzuk: $\overrightarrow{E}:(e_1,e_2)$. Az *eltolási mátrix* a következő:

$$t_e = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ e_1 & e_2 & 1 \end{bmatrix}$$

Az (r_1,r_2) pont (e_1,e_2)-vel való eltolás után az (r_1^t,r_2^t) pontba kerül. Ennek koordinátáira:

$$(r_1^t,r_2^t,1) = (r_1,r_2,1)\cdot\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ e_1 & e_2 & 1 \end{bmatrix}$$

4.6.1.2. *Nagyítás, kicsinyítés*

Ezzel a transzformációval a képet x irányban l_1, y irányban l_2 -szeresére nagyítjuk (ha l_1, $l_2 > 1$), illetve kicsinyítjük (ha l_1, $l_2 < 1$). Egy tetszőleges R pont koordinátái tehát l_1, illetve l_2-szeresére változnak. A *nagyítási mátrix:*

$$t_n = \begin{bmatrix} l_1 & 0 & 0 \\ 0 & l_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ezt kifejezzük a tetszőleges (r_1,r_2) pontra, amely így az (r_1^t,r_2^t) pontba kerül:

$$(r_1^t,r_2^t,1) = (r_1,r_2.1)\cdot\begin{bmatrix} l_1 & 0 & 0 \\ 0 & l_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.6.1.3. *Elforgatás*

Forgassuk el az R pontot α szöggel az origó körül, az óramutató járásának megfelelő irányban. Az *elforgatási mátrix*:

$$t_n = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ezt kifejezzük a tetszőleges (r_1,r_2) pontra, amely így az (r_1^t,r_2^t) pontba kerül:

$$(r_1^t,r_2^t,1) = (r_1,r_2,1)\cdot\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.6.1.4. *Középpontos tükrözés*

Ez a transzformáció tulajdonképpen speciális esete az adott pont körül adott szöggel való elforgatásnak, mivel a középpontos tükrözés nem más, mint a tükrözés középpontja körüli 180° -os elforgatás. Fontossága miatt viszont kiemeljük, s így egyúttal egy példát is adunk az elemi transzformációk kompozíciójára.

Legyen a tükrözés középpontja $K:(k_1,k_2)$, legyen $R:(r_1,r_2)$ egy tetszőleges pont. Ekkor a teendőnk a következő: eltolás az $E:(-k_1,-k_2)$ vektorral, azaz a tükrözés K középpontját az origóba átvisszük. Ezután elforgatás következik α=180 fokkal, majd eltolás az $E:(e_1,e_2)$ vektorral, azaz a középpontot visszahozzuk K-ba. Induljunk ki általánosan az adott pont körüli α szögű elforgatásból és alkalmazzuk ennek speciális esetére, a középpontos tükrözésre:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -k_1 & -k_2 & 1 \end{bmatrix}\cdot\begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 1 \\ 0 & 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ k_1 & k_2 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -k_1 & -k_2 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ k_1 & k_2 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 2k_1 & 2k_2 & 1 \end{bmatrix}$$

A kapott mátrix tehát nem más, mint a $K:(k_1,k_2)$ középpontú tükrözés mátrixa. Ezeknek a transzformációknak közös jellemzőjük, hogy függetlenek a transzformált ponttól, azaz mindegyik pontra egységesen alkalmazhatóak. Ez a számítások lényeges gyorsítását jelenti, amelyeket még inkább gyorsít az a körülmény, hogy itt valójában csak 3x2 -es mátrixokkal kell számolni.

4.6.1.5. *Ablakozás és vágás*

Ezekre a transzformációkra több esetben is szükség lehet, pl. akkor, ha a képernyőn egy grafikus ablakot készítünk és ebben akarunk rajzolni. Így azokat a síkidomokat, amelyek ebből az ablakból 'kilógnak', vágni kell. Általánosan vágás alatt egy adott síkidom területén kívül eső részek levágását értjük. Ilyen értelemben a vágás a poligonfestéssel rokon művelet, mivel ott egy síkidom belső pontjait színeztük ki, a vágás pedig a külső pontok elhagyását jelenti, azaz csak a belső pontok maradhatnak meg.

Legyen most a kijelölt síkidom egy téglalap (a képernyő ablaka, akár maga a teljes képenyő). Az ábrázolt poligonoknak csak az ezen belüli részét szeretnénk ábrázolni. Minden ábrázolt pontról egyértelműen eldönthető, hogy benne van-e az ablakban, egyszerűen megvizsgáljuk, hogy az első és utolsó sor, illetve oszlop közé esik-e. A módszer lassúsága miatt érdemes inkább a sokszögeket határoló egyenesszakaszokról eldönteni: mely részük esik az ablak területére. Így egy újabb poligont kapunk és csak azt kell ábrázolnunk. Ezért elegendő csupán a szakaszok vágását definiálni. E feladat klasszikus megoldása a Cohen-Sutherland algoritmus. Ennek lényege: a szakasz mindkét végpontjához négy bitet rendelünk, melyek '1' értéket kapnak (rendre) a következőképpen. 1.bit: a pont a képernyő bal oldalától balra van; 2.bit: a jobb oldalától jobbra; 3. bit: az alsó sora alatt; 4. bit: a felső sora felett van.

Ezt szemlélteti a következő táblázat is, amelynek értelmezése a következő. A síkot kilenc részre osztottuk. Minden részhez egy bit-négyest rendelünk, pontosabban, a pont annak a résznek a a bit-négyesét veszi fel, amelyikben található.

1001	0001	0101
1000	Ablak: 0000	0100
1010	0010	0110

Képezzük a szakasz két végponjához ily módon rendelt bit-négyes logikai szorzatát, mely újra egy bit-négyes lesz. Átgondolhatjuk, hogy ha ennek értéke nem nulla, akkor a teljes szakasz a téglalapon kívül van, ha mindkét bit-négyes értéke zérus, akkor a teljes szakasz a téglalapon belül van, egyébként csak egy része esik belülre. Csak ez esetben kell vágni, mégpedig az ablak és a szakasz metszéspontja(i)nak megkeresésével. Ez egyszerű feladat, mivel az ablak azon széle, amelyik metszi az adott szakaszt, ki is jelöli a metszéspont egyik koordinátáját. A másik koordináta így egyszerűen kiszámítható, pl. a szakasz egyenletéből, az ismert koordináta behelyettesítésével.

Ez a transzformáció nem kapcsolódik a mátrixokhoz, nem illeszthető be a többi - eddig tárgyalt - transzformáció közé. Ebből is látható, hogy a kétdimenziós transzformációk köre igen sokrétű, kimeríthetetlen lehetőségeket rejt magában. Speciális feladatokra ezért egyedi transzformációkat is előállíthatunk. A fent definiált alapvető transzformációk is segítséget nyújthatnak ezek összeállításában. Változatos, érdekes számítógépes grafikát készíthetünk ily módon.

4.6.2. *Transzformációk három dimenzióban*

A három dimenziós transzformációk magára az objektumra vonatkoznak. Ezek már nem mindig a valóság pontos ábrázolásának eszközei, hanem - éppúgy, mint a kétdimenziós transzformációk - túlmutatnak azon: a valóság transzformált ábrázolásmódját segítik elő. A kétdimenziós transzformációk egyfajta általánosításaként is felfoghatjuk őket, ezért hasonló szerkezetben tárgyaljuk őket.

A három dimenziós transzformációk jelentős részének is egy mátrixot feleltethetünk meg. Ez alól vannak kivételek is, amikor pl. nincs szabályszerűség abban, hogyan definiáljuk a transzformációt. A 'szabályos' három dimenziós transzformációkhoz egy 4x4 -es mátrixot rendelhetünk. Jelöljünk általánosan t-vel most egy három dimenziós transzformációt. Ekkor 't' a következőképpen írható le: $t: R:(r_1,r_2,r_3) \rightarrow R_t:(r_1^t,r_2^t,r_3^t)$, azaz a tér tetszőleges R pontját R_t-be viszi át. 't' általános alakja a következő:

(i)
$$\begin{bmatrix} t_{11} & t_{12} & t_{13} & 0 \\ t_{21} & t_{22} & t_{23} & 0 \\ t_{31} & t_{32} & t_{33} & 0 \\ t_{41} & t_{42} & t_{43} & 1 \end{bmatrix}$$

A mátrix negyedik oszlopa három egyes és egy nulla, minden alább bemutatott transzformáció esetén. Ez gyorsítja a gépi megvalósításokat.

Ahhoz, hogy a transzformációkat egységesen, ebben az alakban kezelhessük, a pontok koordinátáit egészítsük ki egy fiktív, negyedik koordinátával, amelyik azonosan: 1. Így a transzformációk általános alakja a következő:

(ii)
$$(r_1^t,r_1^t,r_1^t,1) = (r_1,r_1,r_1,1)\cdot\begin{bmatrix} t_{11} & t_{12} & t_{13} & 0 \\ t_{21} & t_{22} & t_{23} & 0 \\ t_{31} & t_{32} & t_{33} & 0 \\ t_{41} & t_{42} & t_{43} & 1 \end{bmatrix}$$

A transzformációkat ebben a szerkezetben tárgyaljuk. Előnye, hogy transzformációk kompozícióját is könnyen megfogalmazhatjuk. Ha pl. egymás után alkalmazunk egy 't_e' eltolást és egy 't_f' elforgatást, akkor a két transzformációs mátrix szorzata: $t = t_e \cdot t_f$ is (i) alakú mátrix lesz, éppúgy, mint az ilyen mátrixok sorozatának szorzata is.

Összetett transzformációkat előbb elemi transzformációk sorozatára bontunk, ezek mátrixait definiáljuk, majd összeszorozzuk őket. Ez lesz az összetett transzformáció mátrixa. Tekintsük át vázlatosan az elemi transzformációkat. Alkalmazzuk az eddigi jelöléseket is. A továbbiakban csupán a transzformációs mátrixok bemutatására szorítkozunk, melyek az $(R_t,1) = (R,1)\ t$ szerkezetben alkalmazhatók. Ez a szerkezet a 3. részben bemutatott matematikai modelltől eltérő. Ebből is látható, hogy az ilyen típusú problémák megoldása során többféle lehetőség közül is választhatunk. A perspektív transzformáció definiálását is részben az alábbi, elemi transzformációkra vezettük vissza, de más megközelítésben.

4.6.2.1. *Eltolás*

Az R pontot R_t pontba toljuk el, az $E:(e_1,e_2,e_3)$ eltolási vektorral:

$$(r_1^t,r_1^t,r_1^t,1) = (r_1,r_1,r_1,1)\cdot\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ e_1 & e_2 & e_3 & 1 \end{bmatrix}$$

4.6.2.2. *Nagyítás, kicsinyítés*

Az R pont képe nagyítás, illetve kicsinyítés után: R_t Az x,y,z irányokban az $L:(l_1,l_2,l_3)$ nagyítási (kicsinyítési) vektort alkalmazzuk:

$$(r_1^t,r_1^t,r_1^t,1) = (r_1,r_1,r_1,1)\cdot\begin{bmatrix} l_1 & 0 & 0 & 0 \\ 0 & l_2 & 0 & 0 \\ 0 & 0 & l_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.6.2.3. *Elforgatás*

Három dimenzióban három elemi forgatási transzformációt különböztetünk meg, aszerint, hogy melyik tengely körül történik az elforgatás. Így megkülönböztethetünk t_x, t_y és t_z elforgatási transzformációkat. Két fontos közös tulajdonságukat emeljük ki. Az egyik, hogy azt a koordinátát változatlanul hagyják, amely tengely körül az elforgatás történik. A másik, hogy az alábbi transzformációs mátrixok α nagyságú, az óramutató járásával egyező körüljárási irányra vonatkoznak. Ez a körüljárási irány az elforgatási tengely egy - az elforgatási tengelyre vonatkozóan pozitív koordinátájú - pontból nézve értendő.

x tengely körüli elforgatás:

$$(r_1^t, r_1^t, r_1^t, 1) = (r_1, r_1, r_1, 1) \cdot \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

y tengely körüli elforgatás:

$$(r_1^t, r_1^t, r_1^t, 1) = (r_1, r_1, r_1, 1) \cdot \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

z tengely körüli elforgatás:

$$(r_1^t, r_1^t, r_1^t, 1) = (r_1, r_1, r_1, 1) \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.7. Mozgó, forgó alakzatok megjelenítése. A számítógépes videotechnika matematikai alapjai.

A videtechnika számítógépes megvalósításáról a későbbiekben még szó lesz, de elöljáróban említsünk meg annyit, hogy sok hasonlósági tényező van a hagyományos filmekkel. Itt is képkockából áll egy film. Tekintsünk el most az ezt megvalósító számítástechnikai módszerektől. Ezt a kérdést a 7. fejezetben vizsgáljuk. Csak azt a kérdést vessük elemzés alá, hogy mi legyen ezeken a képkockákon. A képkockákat gyorsan egymás után vetítve a mozgás hatását tudjuk elérni.

A (4.6.) részben definiált transzformációk képezik a videotechnika matematikai alapjait, ezért most néhány gondolatot fogalmazunk meg arról, hogy mi módon tudjuk felhasználni ezeket a képleteket. Az objektumokról alkotott képet a szem, az általa nézett pont és maga az objektum határozza meg, ha eltekintünk a közöttük lévő közegtől. A test mozgását a szemhez képest kell bemutatni, amely ez esetben mint statikus objektum szerepel. A test hozzá képesti elmozdulását tudjuk érzékeltetni. Így viszont az egész testet vagy az ábrázolt képet kell transzformálni. Ez igen sok műveletet jelent, ezért érdemes a szemen végrehajtani az említett transzformáció inverzét. Ez igen előnyös, mivel csak egyetlen pont koordinátáit kell transzformálni és ez sokkal gyorsabb feladat, mintha az objektum minden pontját transzformálnánk. Az inverz transzformáció a már leírt formában egyszerűen megvalósítható.

Részletesebben ez a következőket jelenti. Ha a testet kockánkénti 'α' szögelfordulással akarjuk forgatni egy tengely körül, akkor ezzel ekvivalens a szem '-α' kockánkénti szögelfordulása az említett tengely körül. A test (e_1,e_2,e_3) eltolását, azaz haladását a két képkocka közt a szemnek $(-e_1,-e_2,-e_3)$ eltolásával ábrázolhatjuk. Így a szemhez képest haladó testet láthatunk a képernyőn. Ugyanígy közeledést és távolodást is érzékeltethetünk. Ennek másik módja a nagyítási, kicsinyítési tényezők növelése, illetve csökkentése. A keringést a forgáshoz hasonlóan érzékeltethetjük, a különbség csupán annyi, hogy ez esetben a tengely, amely körül forgatunk, ne haladjon át a testen. Mindezeket a tényezőket együttesen is alkalmazhatjuk, azaz képezhetjük az inverz transzformációk mátrixainak szorzatát is, az előző részben vázolt matematikai modell alapján. Szemléletes, sokszínű filmeket készíthetünk így különféle testekről.

5. HÁROM DIMENZIÓS GRAFIKA SZÁMÍTÓGÉPEN

5.1. *Néhány gondolat a grafikus perifériákról.*

Ebben a részben az eddig biológiai és matematikai szemszögből vizsgált kérdéseket megvizsgáljuk a számítógépes megvalósítás szempontjából. Mindezek előtt azonban nem érdektelen összefoglalni mindazon - a hardware, software és az operációs rendszer - által nyújtott segédeszközöket, amelyek munkánkat támogatják. Kezdjük a hardware -rel.

A grafika alapvető célja a szemléltetés. Két nagy csoportra bonthatjuk a számítógéphez csatolható grafikus megjelenítő perifériákat, rajzgépekre (plotterekre) és grafikus képernyőkre. A téma nagy terjedelme miatt nem célunk most ezek részletes bemutatása, értékelése és összehasonlítása, csupán vázlatosan ismertetjük a fő irányvonalakat. Kezdjük a rajzgépekkel. A rajzgépek - ma már többé-kevésbé - elavult változataiban egy mozgatható karra egy cserélhető tollat szereltek, amely a programok által meghatározott úton mozgott a papír felett. Később ezt továbbfejlesztették, oly módon, hogy a software mindig felkérte a megfelelő színű tollat az operátortól; majd pedig eleve több, különböző színű tollat szereltek a gépbe. A papírt tekercsbe fűzték, így már nemcsak a toll, hanem a papír is tudott mozogni. Ez előnyös volt, mert a rajzolás sebessége megnőtt. Ez utóbbi rajzgépre példa a Calcomp típusú, a kezdetlegesebbre pedig egyes régi HP -plotterek. Mivel az ezirányú fejlesztésnek már nem látták több értelmét, teljesen új elveket kellett kialakítani, így jöttek létre többek közt a raszter -plotterek, melyekben - mint ismeretes - igen sok tű összehangolt munkájával jönnek létre a rajzok. Ezekből a tűkből áramlik a festékanyag a papírra. A tűk egy vagy több sorban egymás mellé vannak felfűzve. A felbontást a tűk közti távolság nagysága adja. A fejlettebb raszter -plotterekben 8-10, vagy több tű is van milliméterenként, azaz egy méteres szélességű rajzhoz sokszor 10000 tű is szükséges, tehát ezek igen bonyolult szerkezetek. Igen nagy előnyük az előző típusú plotterekkel szemben a gyorsaságuk mellett, hogy árnyalt tónusokat is ábrázolhatunk segítségükkel. A papír szintén tekercsen van, ahonnan a tűk alá kerül. Ezeknek két állapotuk van, felemelt és letett, ennek megfelelően írnak vagy sem a papírra. Ilyenek például az egyes Versatec-

plotterek, amelyeknél pl. egy FORTRAN-hoz illeszkedő, részben célnyelvű software segítségével egy mágnesszalagra 'rajzolhatunk', azaz a leendő rajzot a software logikailag mintegy 1/6 mm -es csíkokra vágja, majd - egy csíkot egy rekordnak ábrázolva - mágnesszalagra fűzi. Erről a szalagról egy rekordot beolvasva a plotter egy tűnyi távolsággal előre mozgatja a rajzot. Mivel a tűk a mágnesszalagon lévő rekordnak (azaz csíknak) megfelelően állnak be, ezért a rajz ilyen lépések sorozatával kialakul. A fejlettebb típusokon a rajz még színezhető is.

A grafikus megjelenítő berendezések egy másik nagy csoportja a képernyők. Nem minden képernyő üzemeltethető grafikus módban, csak azok, amelyeknél a hozzá tartozó számítógép tartalmaz grafikus kártyát is. Ezek igen sokfélék lehetnek, megszabják, hogy a képernyőn hány színt használhatunk egyszerre, illetve meghatározzák a képernyő felbontását is. A nálunk leginkább használt IBM és Commodore gépeken normál üzemmódban 320x200, dupla felbontás esetén 640x200 képpont a képernyő felbontása. Normál üzemmódban egyszerre max. 4, dupla felbontásnál csak egy szín és egy háttérszín lehet a képernyőn. Ezek a kártyák 16 Mbyte -osak.

Ezzel a felbontással nem is lehet igazán jó minőségű képeket előállítani, csak ha grafikus bővítőkártyákat használunk. Ilyenek pl. az IBM EGA kártyák, melyeknek felbontása nagyobb. Több üzemmódjuk közt van olyan, amelyben a felbontásuk elérheti a 720x350-et is; valamint van olyan üzemmódjuk, melyben 64 szín mellett a felbontásuk: 640x350. Vannak ennél még nagyobb felbontást lehetővé tevő grafikus bővítőkártyák is. Ezek között igen nagy a választék, akad köztük 1 Mbyte-os is, amelynek a felbontása 1024x1024, 16 szín ábrázolható rajta egyidejűleg, 16 különböző árnyalatban. Ezen egy byte felel meg egy képpontnak, mert annak 256 ($=16^2$) különböző értéke van. Mivel ezek ma még nehezen terjednek, a standard IBM grafikus kártyák pedig szinte minden IBM, vagy azzal kompatibilis számítógépben megtalálhatóak, ezért foglalkozzunk részletesebben az IBM gépek grafikus hátterével. Sok egyéb tényezőtől is függ a kép minősége, pl. a gép műszaki állapotától, de ezek bemutatása most nem célunk.

5.2. *A képernyőt meghatározó memóriaterület szerkezete*

A számítógépek felépítésének általános gyakorlata, hogy a képernyőt leíró adatterületnek ki van jelölve a memória egy része. Ez a terület kódolt formában pontosan azt írja le, ami a képernyőn látható. A direkt képernyőelérés, az említett memóriaterület minél pontosabb ismerete nagyon fontos, mert közelebb visz a grafikai problémák gyors, pontos megoldásához. Az IBM PC, XT, AT gépeken igen gyakran alkalmazott nyelv a TURBO-PASCAL, ezért a továbbiakban ennek a nyelvnek a terminológiáit használjuk. Az IBM PC -ken a képernyőt leíró memóriaterület általában a $b800:$0 abszolút címen kezdődik és hossza normál vagy dupla felbontású grafikus üzemmód, azaz GraphMode, GraphColorMode vagy HiRes esetén 16 K, azaz 16384 byte, szöveges üzemmód (TextMode) esetén 4 Kbyte.

Szöveges üzemmód esetén a képernyőt a következő tömbbel írhatjuk le:

```
f1: array[1..25,1..160] of char absolute $b800:$0;
```

vagy:

```
f2: array[1..25,1...160] of byte absolute $b800:$0;
```

Az f1 és f2 tömbök 4000 byte hosszúak, az első dimenziójuk jelenti a képernyőn a sorokat, a második az oszlopokat oly módon, hogy egy oszlopnak két elem felel meg. Az n-edik oszlopnak a 2n-edik és a 2n-1 -edik elem. A 2n-1 -edik tartalmazza magát a megjelenített karaktert, a 2n -edik pedig annak színét és alapszínét. 16 szín lehetséges, de mivel mindegyik villoghat is, ezért 32 féle különböző fajta színt adhatunk egy karakternek, valamint 8 féle alapszínt is, tehát összesen 32x8=256 féle színkombináció rendelhető minden karakterhez. Egy byte éppen 256 féle értéket vehet fel, tehát ez az adott sor n-edik oszlopa esetén a 2n -edik elem lesz. Az említettekre példa az alább mellékelt Példa1 nevű, Turbo-Pascal nyelvű program, ahol a már említett f1 és f2 tömböket ráde-finiáljuk a memóriának a képernyőt leíró területére. Ezután figyeljük meg a programnak a 'TextMode-analízis' -részét. Az f1 tömb segítségével az 'a,b,c'-karaktereket felírjuk a képernyő első és utolsó sorának 1.,2. és 80. (utolsó) oszlopába, pl. a képernyő bal alsó sarka tudvalevően a 25. sorban, a 80. oszlopban van, ez az eddigiek szerint az f1[25,159] elem. Ezeknek az f2 tömb segítségével különböző színt és alapszínt adunk. Tekintsük most a Példa1 programot. A program 2. része a grafikus mód elemzése, ennek értelmezése pedig a program szövege után található.

```
program  Pelda1;
{$c-}
var
   i,j: integer;
   c1:  char;
   b:   byte;
  {Rádefiniálunk a képernyő-területre:}
   e1:  array[1..100,1..80]  of byte  absolute  $b800:$0;
   e2:  array[1..100,1..80]  of byte  absolute  $ba00:$0;
   f1:  array[1..25,1..160]  of char  absolute  $b800:$0;
   f2:  array[1..25,1..160]  of byte  absolute  $b800:$0;

begin
  { 1.  Textmode-analízis:  }
  { f1 tömbbel a,b,c-karaktereket írunk a képernyő
    1. és  utolsó sorába.  f2 tömb  színezi ezeket
    a karaktereket:}
   ClrScr;
   TextMode;

   f2[1,2]:=3;          f1[1,1]:='a';
   f2[1,4]:=10;         f1[1,3]:='b';
   f2[1,160]:=20;       f1[1,159]:='c';
   f2[25,2]:=37;        f1[25,1]:='a';
   f2[25,4]:=45;        f1[25,3]:='b';
   f2[25,160]:=216;     f1[25,159]:='c';
   Read(kbd,c1);                          { Várakoztató utasítás }

  {2.  GraphColorMode-analízis:  }
   GraphColorMode;          { Itt állhat GraphMode vagy HiRes is }
   Palette(1);

  {az első 5 páratlan sort meghúzza:}
   b:=255;
   for i:=1  to 5 do for j:=1 to 80 do e1[i,j]:=b;

  {az utolsó 10 páros sor 2. felét meghúzza:}
   b:=127;
   for i:=90 to 100 do for j:=41 to 80 do e2[i,j]:=b;
   read(kbd,c1);                          { Várakoztató utasítás }
end.
```

Most nézzük a képernyőt leíró memóriaterület felépítését grafikus üzemmódok esetén. A Példa1 -program ismét segíti elemzésünket. Az e1 és az e2 tömbök együttesen írják le a képernyő memóriaterületét. A képernyő, a grafikus üzemmódokban szintén az $b800:$0 abszolút címen kezdődik és 16 Kbyte hosszú területet foglal el. Ezen belül két, jól elkülöníthető, 8 Kbyte, azaz 8192 byte hosszúságú részre osztható. Az első területre az e1, a másodikra az e2 tömböket definiáltuk a Példa1 -programban. A képernyőt ugyanis normál grafikus üzemmódban 200 sor, 320 oszlop alkotja, pontosabban: 0-199 -ig számozzuk a sorokat felülről lefelé és 0-319 -ig számozzuk az oszlopokat balról jobbra. Ez azt jelenti, hogy a képernyőt 64000 képpont alkotja, mindkét fél képernyőterületet 32000. Ez a következő módon valósul meg.

Induljunk ki a legkisebb egységből, a képpontból. Egy képpontot normál grafikus üzemmódban két biten, dupla felbontás esetén egy biten ábrázolunk:

- Normál felbontás:
  A képernyőn egyszerre összesen négy szín lehet, az alapszín és három másik. Minden képpont színe e négy szín valamelyike kell, hogy legyen. Ezt a színt a képpontot leíró bitpár értéke határozza meg, ha ez 00, akkor a képpont alapszínű, 01,10,11 pedig a másik három színnek felel meg. Így egy byte -on négy szomszédos képpontot ábrázolhatunk, mindegyiket négyféle színben. A képernyő egy sorát 320 képpont alkotja, ezt 80 szomszédos byte tartalmazza.

- Dupla felbontás:
  A képernyőn minden pont két állapotban lehet összesen, alapszínű vagy egy másik. Az alapszín a legtöbb gépen egyszerűen nem létezik, vagyis sötét, de egyes AT -ken HiRes -ben is színes a háttér. Minden gépen tehát egy szín és egy háttérszín lehet összesen egyszerre, HiRes -ben. Egy képpontot HiRes -ben egy bit ír le, ha ennek értéke: 1, akkor a pont színes, ha 0, akkor háttérszínű. Így egy byte -on nyolc képpontot ábrázolunk, mindegyiket kétféle színben. A képernyő egy sorát 640 képpont alkotja, ezt 80 szomszédos byte tartalmazza.

A két felbontás egymással kompatibilis, a következő értelemben. Tegyük fel, adott egy grafika a képernyőn, normál felbontásban. Kimentjük a képernyő-területet valahova, HiRes -re váltunk,

majd visszatöltjük az említett területet. A fentiekből is következik, hogy a grafika megőrzi jellegét, nem történik vele más, csak négyszínűből két színűvé válik. A képpontokat ugyanis eddig két bit írta le, ez a két bit most már két képpontot fog jelenteni. Ha értékük 00 volt, akkor a pont háttérszínű volt, s az is marad a helyette álló két pont. Ha 01 vagy 10 volt, akkor egy világos és egy háttérszínű pont áll majd az egy képpont helyén. Ez azt jelenti, hogy az ilyen színű területek HiRes -ben is színesek, de gyengébb tónusúak lesznek, mint az eredetileg 11-gyel leírt képpontok. Azok helyén ugyanis két fényes képpont áll majd. A leírt váltás visszafelé, HiRes -ből normál felbontásúra is igaz, értelemszerűen. Egy ilyen 'váltást' nem lehet közvetlenül elvégezni, mert akkor a képernyő tartalmát a rendszer törli, ezért van szükség közben egy mentésre és visszatöltésre.

Megállapítottuk, hogy a képernyő egy sorát minden esetben 80 byte írja le. Így a grafikus üzemmódokban összesen 16000 byte írja le a képernyőt. Mivel 16 Kbyte = 16394 byte - ennyi a képernyő memóriaterülete - mi történik a felmaradó 394 byte-tal ? Erre keressük a választ.

Mint már említettük, a képernyő memóriaterülete két 8 Kbyte-os részből áll. Az első a $b800:$0 címen kezdődik és a képernyő páratlan sorait tartalmazza, de a sorok általánosan használt, 0-val kezdődő számozása szerint a párosakat, még pontosabban a 0,2,4,6,...196,198 sorokat. A másik terület a $ba00:$0 címen kezdődik és a képernyő páros sorait tartalmazza, azaz - hasonlóan az előzőekhez - az általános használatú, 0 kezdetű sorszámozás szerint a páratlanokat, még pontosabban az 1,3,5,7,...197,199 sorokat. Mindkét terület 8000 értékes byte-tal kezdődik és 192 redundáns, a képernyő szempontjából lényegtelen byte-tal fejeződik be. Ilymódon a Példa1 nevű programban az e1 és az e2 tömbök együttesen leírják a képernyő területét. Első indexük a sor számára utal. Ha 'e1' első indexe: n, akkor a 2n-2 -edik sorban, ha 'e2' első indexe: n, akkor a 2n-1 -edik sorban van az a képpont, amelyre a tömb eleme vonatkozik. Amennyiben bőven áll rendelkezésre memória, sokszor célszerű a következő tömbbel leírni a képernyő területét:

```
e: array[16192] of byte absolute $b800:$0;
```

Itt az egész területet egységesen kezeljük. Ez a tömb tartalmazza a két fél képernyőterületet és a köztük lévő 192 byte-os redundáns területet is.

5.3. *A Turbo-Pascal nyelv grafikus elemei*

A Turbo-Pascal nyelv igen alkalmas az IBM személyi számítógépeken grafikus programok fejlesztésére. Foglaljuk össze a grafikus ábrázoláshoz legszükségesebb eljárásokat e nyelven belül. Mint már említettük, három grafikus üzemmódot ismerünk, ezek a következők:

GraphColorMode	320x200 felbontású színes grafika
GraphMode	320x200 felbontású fekete-fehér grafika
HiRes	640x200 felbontású fekete + egyszínű grafika

Az első kettőben a képernyőre 40x25, HiRes-ben 80x25 karaktert írhatunk, hasonlóan a szöveges üzemmódhoz. Általánosan érvényes, hogy a (0,0) koordinátájú pont a képernyő bal felső sarka, az X -koordináta jobbra, az Y -koordináta lefelé növekszik. Értelmezési tartományuk a következő: Y: (0-199); X: (0-319), HiResben X: (0-639). A feketét is beleértve összesen 16 színnel gazdálkodhatunk, mely színekre egy 0-15 -ig terjedő egész számmal, vagy magának a színnek az angol nevével is hivatkozhatunk, a következő táblázat szerint:

0: Black	(fekete)	8. DarkGray	(sötétszürke)
1: Blue	(kék)	9. LightBlue	(világoskék)
2: Green	(zöld)	10. LightGreen	(világoszöld)
3: Cyan	(enciánkék)	11. LightCyan	(világosencián)
4: Red	(vörös)	12. LightRed	(világosvörös)
5: Magenta	(lila)	13. LightMagenta	(világoslila)
6: Brown	(barna)	14. Yellow	(sárga)
7: LightGray	(világos szürke)	15. White	(fehér)

A fenti megfeleltetések állandóan érvényesek, tehát ahol a 0-15 közül valamelyik számot használjuk pl. a szín kijelölésére, ott írhatjuk helyette a táblázatban lévő, neki megfelelő nevet, mint szövegkonstansot. Így van ez pl. a HiResColor(n) procedura esetén is, amellyel HiRes esetén a színt definiáljuk. Normál grafikus üzemmódnál négy szín lehet egyszerre a képernyőn, egy háttér és három másik, sőt, még az is meg van szabva, hogy melyik szín melyekkel állhat együtt. Ennek megfelelően négy paletta áll rendelkezésünkre, melyek közül a Palette(n) procedúrával választhatunk, ahol $n \in \{0,1,2,3\}$.

A Plot(X,Y,n) procedúrával a képernyő (X,Y) -koordinátájú pontjának adunk 'n' szín-értéket, mely az elmondottak szerint 0 esetén a háttérszín, egyébként az aktív paletta többi három színéből a megfelelő. Ha GraphMode -ban vagyunk és színes a monitorunk, akkor a háttérszín mellett csak piros, kék és világosszürke, vagy háttér, világoskék -és piros, fehér színösszeállítással dolgozhatunk. A képernyő háttérszínét a GraphBackGround(n) eljárással módosíthatjuk. A Draw(x1,y1,x2,y2,n) procedúra a képernyőn egyenest húz az (x1,y1) koordinátájú pontból az (x2,y2) pontba.

A GraphWindow(x1,y1,x2,y2) procedúra leszűkíti a képernyőt egy aktív ablakra, ahol az ablaknak (x1,y1) a bal felső, (x2,y2) a jobb alsó sarkai. Ezután ennek az ablaknak a bal felső sarka lesz a (0,0) koordinátájú pont.

Néhány szót még a Turbo-Pascal grafikus kiterjesztéséről. A Borland Intézet, amely a Turbo-Pascal -t készítette, néhány - a grafikus programokat támogató - rutint is mellékelt a rendszerhez. Említsünk meg ezek közül néhányat, melyek hasznosak lehetnek munkánk során. Ezek a grafikus rutinok a 'graph.bin' -könyvtárban találhatók, melyeket a fordítóprogram akkor vesz figyelembe, ha az {$i graph.p} utasítást alkalmazzuk, amely behívja az említett rutinokat.

A Circle(x,y,r,s) -rutin az (x,y) középpont körül 'r' sugarú kört rajzol 's' -színnel. Az Arc(x,y,alfa,r,s) -rutin az (x,y) koordinátáról indulva 'alfa' -szögű, 'r' -sugarú körívet rajzol, 's' színnel. A GetPic(buffer,x1,y1,x2,y2) -rutin a képernyő egy - az (x1,y1) bal felső és az (x2,y2) jobb alsó sarkával kijelölt - területét eltárolja a buffer nevű változóban, ennek inverze pedig a PutPic(buffer,x1,y1,x2,y2) -rutin, mely a képernyőre visszatölti az említett mező tartalmát. A buffernek elegendő helyet kell lefoglalni, mert különben a később deklarált változókat helyhiány miatt felülírja. Kivétel, ha ez az utolsó változó, a deklarálások mellett pedig marad elegendő memóriaterület, mint az alábbi program esetén. Ennek neve: Példa2. Feladata: szemlélteti a grafikus kiterjesztésről elmondottakat. Előbb a Circle -rutin alkalmazásával egy egyszerű alakzatot rajzol, majd ezt a képernyőn mozgatja, több irányban, a 'nyilak' -funkcióbillentyűk segítségével. Enter hatására a program befejeződik. Csak addig tudjuk az alakzatot mozgatni, ameddig a közepe el nem éri a képernyő szélét. A funkcióbillentyűk kezelését a program Fbill -rutinja végzi. Az {$i-} direktíva hatása: <Enter> nélkül is beolvashatunk egy karaktert.

```
program Pelda2;
{$i-}
{$i graph.p}

var
   c:char;
   i,j:integer;
   buffer:integer;

label
   bal,jobb,le,fel,valaszt,vege;

{-}

function Fbill:integer;
var
   evar:byte;
begin
   evar:=0;
   while (evar=0) do begin
      Read(kbd,c);
      evar:=1;
      case Integer(c) of
         13:  evar:=1;      {enter}
         27:  Read(kbd,c);
         else evar:=0;
      end;
   end;
   Fbill:=Integer(c);
end;

{-}

begin
   GraphColorMode;
   GraphBackGround(blue);
   for i:=1 to 90 do Circle(160,100,i,1);
   for i:=1 to 60 do Circle(160,100,i,2);
   for i:=1 to 30 do Circle(160,100,i,3);
   GetPic(buffer,0,0,319,199);    { Az előállított ábra tárolása }
   i:=0;
   j:=199;
```

```
 bal:                                    { Az alakzat balra mozog }
   if i>-160 then begin
      i:=i-1;
      PutPic(buffer,i,j);
      if not KeyPressed then goto bal;
   end;
   goto valaszt;

 jobb:                                   { Az alakzat jobbra mozog }
   if i<160 then begin
      i:=i+1;
      PutPic(buffer,i,j);
      if not KeyPressed then goto jobb;
   end;
   goto valaszt;

 le:                                     { Az alakzat lefelé mozog }
   if j<300 then begin
      j:=j+1;
      PutPic(buffer,i,j);
      if not KeyPressed then goto le;
   end;
   goto valaszt;

 fel:                                    { Az alakzat felfelé mozog }
   if j>100 then begin
      j:=j-1;
      PutPic(buffer,i,j);
      if not KeyPressed then goto fel;
   end;


 valaszt:        { A billentyűhöz tartozó funkció szétválasztása }
   case fbill of
        77:goto jobb;
        75:goto bal;
        72:goto fel;
        80:goto le;
        13:goto vege;
        else goto valaszt;
   end;
 vege:ClrScr;
end.
```

5.4. Kopírozás

A kopírozásnak nagy jelentősége lehet akkor, ha több, három dimenziós objektumot akarunk együtt ábrázolni. Tekintsünk ugyanis csupán két testet. Ha takarják egymást, akkor egy triviális, de az ábrázolás szempontjából lényeges körülményt kell figyelembe venni. Az esetek túlnyomó többségében teljes takarás van a testek között, azaz egyértelmű, hogy csak a tőlünk távolabbi testen érvényesül a takarási hatás. Ez alól kivétel például az, ha az egyik egy bonyolult konkáv test, amely körülveszi a másikat. Most tekintsünk el ettől az esettől, mivel annak csak teljesen általánosan oldhatjuk meg az ábrázolását. Tekintsük az esetek túlnyomó többségét, amikor egyértelmű, hogy az egyik test takart, a másik látható. Ez esetben az azonos nézőpont mellett külön ábrázolhatjuk a testeket, majd a kapott képernyőket elmentjük és végül összekopírozzuk őket. Ez az eljárás is tökéletes képet ad a testekről. Előnye, hogy egy igen gyors algoritmus adható rá. Ha más módszerrel oldanánk meg ezt a feladatot, az ehhez képest nagyon lassú lenne, ugyanis akkor külön kellene figyelni mindkét testen az őket burkoló poligonok jelentős részét és azokon vizsgálni, hogy melyik takarja egymást. Látható, hogy a kopírozás ennél lényegesen egyszerűbb.

A kopírozás esetén csupán azt kell tisztázni, melyik test van hozzánk közelebb, ezután már hívhatjuk is az alább leírt algoritmust. A kopírozáshoz szinte mindent fel kell használnunk a képernyő felépítésében eddig leírtakból, tehát ez az algoritmus az eddigieknek mintegy szintézise lesz, sőt talán még tisztábban mutatja meg a képernyő tulajdonságait.

Tekintsük a képernyőt az egyszerűség kedvéért egy 16192 byte nagyságú tömbnek. Így a csupán a két fél képernyőt és a kettő közt álló redundáns területtel foglalkozunk. A 192 byte-os redundáns terület az egészhez képest olyan kevés és a módszer olyan gyors, hogy mindez nem is érezhető, viszont az algoritmus még egyszerűbbé válik ezáltal. Tehát a képernyő most mint egy 16192 byte-os egész van előttünk. Így elegendő konstruálni egy olyan modult, amely egy byte-ra elvégzi a kopírozást. Ezt kell hívni 16192-ször és akkor az egész képernyőt tudjuk kopírozni. Így működik az alább leírt, 'Scr_Kopiroz' nevű modul, amely, mint a neve is mutatja, képernyőket lehet vele összekopírozni. Ennek 'Byte_Kopiroz' nevű modulja byte-okra végzi el ugyanezt. Első látásra nem nyilvánvaló, hogyan működik, ezért nézzük részletesebben.

Mivel két bit ír le (normál üzemmód esetén) egy képpontot, egy byte-on - négy bitpár segítségével - négy képpontot írhatunk le. Ahhoz, hogy a legkisebb, közvetlenül elérhető egységnél, a byte-nál kisebb egységhez, a bitpárhoz is hozzá tudjunk férni egy trükk segítségével; definiáljuk azokat a kettes számrendszerbeli számokat, amelyek csupa egyesből állnak és csak az egyik bitpárjuk nulla. Tekintsük azokat is, amelyek ezek tükörképeként, minden bitjük nulla, kivéve az egyik bitpárt. Itt bitpár alatt két olyan szomszédos bitet értünk, amely egy képpontot ír le, tehát az (1,2); (3,4); (5,6); (7,8); biteket. Az említett kettes számrendszerbeli számokat tegyük bele az A és a C tömbökbe, amelyek típusa: array[1..4] of byte. Ezt szemlélteti a következő táblázat is:

A -tömb:

index:	Értéke: (számrendszer)	
	kettes:	tízes:
1	00111111	63
2	11001111	207
3	11110011	243
4	11111100	252

C-tömb:

index:	Értéke: (számrendszer)	
	kettes:	tízes:
1	11000000	192
2	00110000	48
3	00001100	12
4	00000011	3

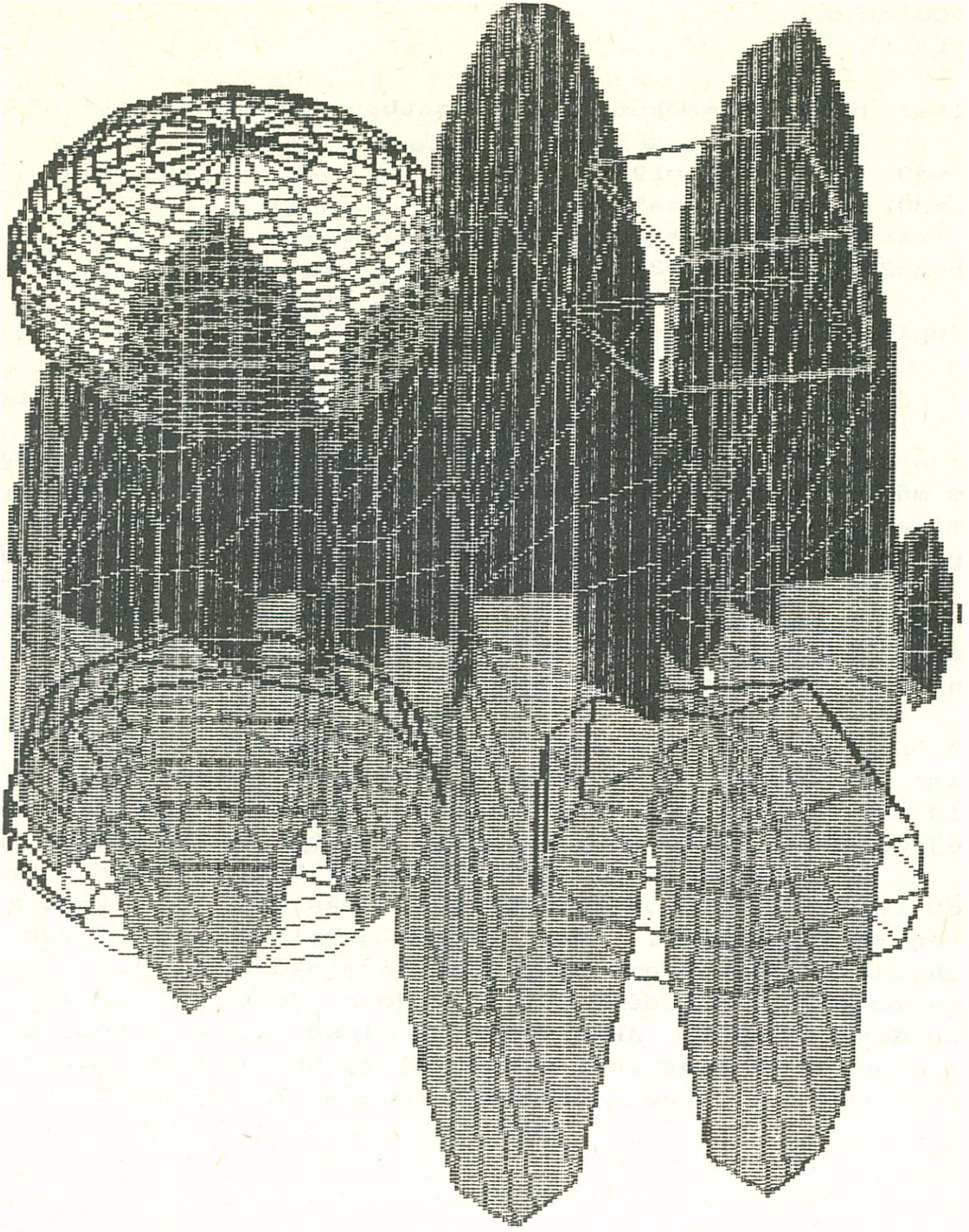
Vegyük észre, hogy az A tömb i-edik elemének kettes számrendszerbeli alakjában az i-edik bitpár: 00, a többi egyes, a C tömb i-edik elemének kettes számrendszerbeli alakjában az i-edik bitpár: 11, a többi nulla. Nevezzük b1-nek és b2-nek a két, egymásra kopírozandó byte-ot. Legyen b1 felül. Kopírozásra csak akkor van szükség, ha egyik byte sem nulla, egyébként az eredmény byte - legyen a modulban a neve: 'ki' - felveszi a nem nulla byte értékét. (Ha mindkettő nulla, természetesen az eredmény is nulla). Ezt a szétválasztást fejezi ki az alábbi program 'Scr_Kopiroz' rutinjának a kommentben 'esetek szétválasztása' néven megjelölt rész. A 'Byte_Kopiroz' rutin a[n]=a[n] or b1 -feltétele akkor teljesül, ha b1-nek az n-edik bitpárja: '00'. Ekkor a 'ki' -byte n-edik bitpárja a b2 byte n-edik bitpárját veszi fel, ha pedig a feltétel nem teljesül, b1 -ét fogja felvenni. c[n] and b2 értéke mindenütt nulla, csak az i-edik bitpárja olyan, mint b2-nek. Hasonlóan igaz ez c[n] and b1 esetén is. A megfelelő bitpárokat tehát kiválasztjuk, majd összefűzzük őket a 'ki' nevű byte -ba.

```
Program Kopiroz;
{$c-}
type
     scr      =     array[1..16192] of byte;
var
     srek:          scr  absolute  $b800:$0;
     s1rek, s2rek:scr;
     i,j:           integer;
     b:             byte;
     a,c:           array[1..4] of byte;

{-}

procedure Byte_Kopiroz(var b1,b2,ki:byte);
var
  n:integer;
begin
  ki:=0;
  for n:=1 to 4  do
       if (a[n]=a[n] or b1)
       then ki:=ki or (c[n] and b2)
       else ki:=ki or (c[n] and b1);
  { A 'ki' -byte összefűzi a megfelelő bitpárokat }
end;

{-}

procedure Scr_Kopiroz(var s1rek,s2rek,srek:scr);
begin
  b:=0;
  for i:=1 to 16192 do begin              {esetek szétválasztása:}
       if s1rek[i]=s2rek[i] then srek[i]:=s1rek[i]
       else begin                         {ha a két byte nem egyenlő:}
                                          {   ha valamelyik nulla:    }
            if s1rek[i]=b then srek[i]:=s2rek[i];
            if s2rek[i]=b then srek[i]:=s1rek[i]
            else if s1rek[i]<>b then
                                          {    ha egyik sem nulla:    }
            Byte_Kopiroz(s1rek[i],s2rek[i],srek[i]);
       end;
  end;
end;
```

```
begin
  GraphColorMode;
  Palette(1);

  { Feltöltjük  a tömböket  a táblázatban  szereplő értékekkel: }

  a[1]:=63;       c[1]:=192;
  a[2]:=207;      c[2]:=48;
  a[3]:=243;      c[3]:=12;
  a[4]:=252;      c[4]:=3;

  Scr_Kopiroz(s1rek,s2rek,srek);
end.
```

Ez a program példa a közvetlen képernyőkezelésre, a bitmanipulációs műveletekre. A (7.3) részben még visszatérünk erre a kérdésre, mivel a kopírozás igen sokoldalúan hasznosítható, például a számítógépes videotechnikában lehet alkalmazni. Ezzel és sok más hasonló módszerrel igen változatos trükkfilmeket lehet készíteni. A kopírozó programmal előállított képekre nézzünk egy példát is, a 13. ábrán látható, több testet ábrázoló rajz is ezzel a programmal nyerte el végleges formáját.

Külön figyelmet érdemel az scr = array[1..16192] of byte típus, amely nem mást, mint magát a képernyőt jelenti. Több lehetőség is van a képernyő definiálására, mint már ezt jeleztük az előző fejezetekben. Azt, hogy melyiket választjuk, a feladat szabja meg. Ha pl. a memóriával takarékoskodni kell, akkor a két fél képernyőt leíró változatot alkalmazzuk, más esetekben viszont kifejezetten megéri a fenti 'scr' típust alkalmaznunk. Ilyen pl. a videotechnika is, ahol ez a legjobb képernyődefiniálás, mivel a keletkező mozgófilm minősége így a legjobb. Jobb, mintha a két fél képernyőt definiálnánk. Akkor ugyanis kissé ugrál a kép, itt viszont az ehhez a felbontáshoz képest legjobb minőségű mozgófilmet tudjuk konstruálni. Erről bővebben még a 7. fejezetben olvashatunk.



13. ábra: Térbeli testek (számítógépes grafika)

5.5. *Poligonfestés*

A poligonfestés igen lényeges feladat a három dimenziós grafikában. Ugyanis mindeddig poligonokkal közelítettük a test felszínét. Egy ilyen poligon pontjai - mivel egy síkban vannak - hasonlóan viselkednek a fény-árnyék hatások, a tónusok szempontjából. Ezért egy poligon pontjait célszerű egy színnel vagy tónussal kitölteni. Ebből látható a poligonfestő algoritmusok jelentősége. Nélkülük olyan tömör testet nem tudunk ábrázolni, amelyet poligonokkal közelítettünk.

A poligonokkal való közelítés nem az egyetlen módszer a testek ábrázolására. A szabályos, matematikai testeket nem is kell közelítenünk, hiszen azok felszínét már eleve poligonok alkotják. Tulajdonképpen a valós testeket is ilyen poliéderekkel közelítettük. Ez a módszer azért is jó, mert viszonylag gyors és a valóságot elég jól tudjuk általa közelíteni, mivel a pontosságot elvileg korlátlanul növelhetjük. A gyakorlatban azonban nem poligonokat, hanem pontokat látunk, melyek végül egységes képpé állnak össze. A valóságot a szemünk - mint már leírtuk - kb. 50 millió ponttal közelíti. Tulajdonképpen elegendő lenne a testeket a számítógépes analízis esetén is pontokkal közelíteni. Annyi ponttal, amennyi elegendő ahhoz, hogy a képernyő minden pontjára legalább egy pontot le tudjunk képezni. Így a kapott kép is folyamatos lenne. Ennek a megoldásnak is megvannak az előnyei és a hátrányai is. Előnye pl. hogy ebben az esetben nincs szükség a poligonfestésre, de igen nagy hátránya, hogy szabályos testek esetén is igen nagy tárterületet igényelne. Ezáltal - a fellépő sok számítás miatt - egyszerű testek ábrázolásánál igencsak megnövekedne a felhasznált gépidő mennyisége. Egyéb gondok is vannak ezzel a megadási móddal. Az adatbázis felvitele nehézkes és nehéz megtalálni azt a pontsűrűségi mértéket, amely még éppen elég ahhoz, hogy a kapott kép folyamatos legyen, de ne jusson egy képpontra az adatbázis több pontja is, mert ez utóbbi csak a redundanciát növeli.

Ennek ellenére érdemes néhány gondolat erejéig elidőzni még e lehetőség mellett. Ez az eset tulajdonképpen nem is áll olyan messze a poligonokkal való közelítéstől. Egy pontot ugyanis tekinthetünk elfajult poligonnak is, amelynek csupán egy csúcsa van. Ennek is van normálvektora, amely nem más, mint a testet az adott pontban érintő síkra az adott pontban állított, a testből kifelé mutató merőleges vektor.

Így az adatbázisban a pont koordinátái után a normálvektor koordinátái következnek. Ez elegendő a pont leírására. Ha megfigyeljük viszont a poligonos adatbázis szerkezetét, láthatjuk, hogy a két eset teljesen ugyanaz. A különbség csupán annyi, hogy ott a normálvektor koordinátái előtt több pont - a poligon csúcsainak - koordinátái álltak, itt csak egy pont koordinátái állnak előtte. Az adatbázis szervezése szempontjából ez teljesen ugyanaz, mivel általánosságban néhány rekordot egy normálvektort leíró rekord követ, majd egy lezáró rekord. Ilyen egységekből épül fel az adatbázis. Ezt azt jelenti, hogy egy file-on belül is vegyesen alkalmazhatjuk a poligonokat és a pontokat leíró rekordokat, mivel a kezelésük egységes. Így a test természete szabhatja meg, hogy melyik megadást válasszuk. A ponttal történő megadás esetén a normálvektor alapján állapíthatjuk meg, hogy a pont látható-e, fényben vagy árnyékban van. Ez szabja meg a színét és tónusát is.

A poligonokkal való leírás esetén viszont elengedhetetlenül szükséges valamilyen poligonfestő algoritmust készíteni. Ennek matematikai elméletéről a (4.3) részben már volt szó. Egy konvex poligonokat festő algoritmust valósít meg a következő példánkban a 'Festn' -nevű modul. Egy általános poligont a 't' tömbben eltárolunk a csúcspontjainak képernyő-koordinátái alapján. A modul a poligonnak elméletben meghúzza az egy csúcsából kiinduló átlóit. Ezáltal a poligont háromszögekre szeletelte fel. Minden ilyen háromszögre meghívja a 'Fest3' nevű modult, amely egy háromszöget fest be adott színűre.

Ez utóbbi úgy működik, hogy a háromszöget felvágja vízszintes csíkokra, a képernyő sorai szerint. Megvizsgálja, hol metszi az adott sor a háromszög oldalát. Ez két pontot jelent (kivéve, ha éppen egy csúcspontban vagyunk); ezeket összeköti a megadott színnel, ily módon beszínezi az egész háromszöget. Az alábbi program egyúttal egy példát is ad, amelyet ki is próbálhatunk a gyakorlatban is. A 't' tömbbe eltároljuk a képernyőn lévő négy pont koordinátáit. A négy pont - mint csúcspont - által megadott négyszöget a Festn eljárás a 2-vel jelölt színűre kifesti, a Huzn eljárás pedig a négyszög kerületét a 3-mal jelölt színűvel húzza meg. Midkét eljárás második paramétere a tömbben lévő pontok száma. Ez esetünkben maximum 10 lehet, de - a memória szabta korlátig - a tömb méretét - s így az ábrázolandó poligon maximális oldalszámát is - növelhetjük, a 'tomb' -típus dimenziójának emelése által.

```
Program Poligonfestes;
type
   tomb = array[1..10,1..2]of integer;
var
   t    : tomb;
   c1   : char;

procedure Fest3(var a1,a2,b1,b2,c1,c2,szin:integer);
var        {Háromszög-festő algoritmus:}
   p1,p2,q1,q2,r1,r2,osz1,osz2,e1,e2,cons1,cons2:real;
   sor:integer;

begin      {a csúcsok koordinátáit a  p1,p2,q1,q2,r1,r2  változókba
           tesszük, úgy, hogy p2 ≤ q2 ≤ r2}
   if (a2<=b2)and(a2<=c2) then begin
      p1:=a1;       p2:=a2;
      if b2<=c2  then begin
         q1:=b1;    q2:=b2;    r1:=c1;    r2:=c2;
      end
      else begin
         q1:=c1;    q2:=c2;    r1:=b1;    r2:=b2;
      end;
   end;

   if (b2<=a2)and(b2<=c2) then begin
      p1:=b1;       p2:=b2;
      if a2<=c2 then begin
         q1:=a1;    q2:=a2;    r1:=c1;    r2:=c2;
      end
      else begin
         q1:=c1;    q2:=c2;    r1:=a1;    r2:=a2;
      end;
   end;

   if (c2<=a2)and(c2<=b2) then begin
      p1:=c1;       p2:=c2;
      if a2<=b2 then begin
         q1:=a1;    q2:=a2;    r1:=b1;    r2:=b2;
      end
      else begin
         q1:=b1;    q2:=b2;    r1:=a1;    r2:=a2;
      end;
   end;
```

```
   {csíkokra szeletelünk, azokat megrajzoljuk:}
   if p2<>q2 then begin
      e1:=(p1-q1)/(p2-q2);
      e2:=(p1-r1)/(p2-r2);
      osz1:=p1;
      osz2:=p1;
      for sor:=Round(p2) to Round(q2) do begin
          Draw(Round(osz1),sor,Round(osz2),sor,szin);
          osz1:=osz1+e1;
          osz2:=osz2+e2;
      end;
   end
   else Draw(Round(p1),Round(p2),Round(q1),Round(q2),szin);

   if q2<>r2 then begin
      e1:=(q1-r1)/(q2-r2);
      e2:=(p1-r1)/(p2-r2);
      osz1:=q1;
      osz2:=p1+e2*(q2-p2);
      for sor:=Round(q2) to Round(r2) do begin
          Draw(Round(osz1),sor,Round(osz2),sor,szin);
          osz1:=osz1+e1;
          osz2:=osz2+e2;
      end;
   end
   else Draw(Round(q1),Round(q2),Round(r1),Round(r2),szin);
end;

{--}

procedure Festn(t:tomb; n,szin:integer);
{poligonfestő algoritmus:}
var
   i:integer;
begin
   for i:=2 to n-1 do
   Fest3(t[1,1],t[1,2],t[i,1],t[i,2],t[i+1,1],t[i+1,2],szin);
end;

{--}
```

5.5. *Poligonfestés*

A poligonfestés igen lényeges feladat a három dimenziós grafikában. Ugyanis mindeddig poligonokkal közelítettük a test felszínét. Egy ilyen poligon pontjai - mivel egy síkban vannak - hasonlóan viselkednek a fény-árnyék hatások, a tónusok szempontjából. Ezért egy poligon pontjait célszerű egy színnel vagy tónussal kitölteni. Ebből látható a poligonfestő algoritmusok jelentősége. Nélkülük olyan tömör testet nem tudunk ábrázolni, amelyet poligonokkal közelítettünk.

A poligonokkal való közelítés nem az egyetlen módszer a testek ábrázolására. A szabályos, matematikai testeket nem is kell közelítenünk, hiszen azok felszínét már eleve poligonok alkotják. Tulajdonképpen a valós testeket is ilyen poliéderekkel közelítettük. Ez a módszer azért is jó, mert viszonylag gyors és a valóságot elég jól tudjuk általa közelíteni, mivel a pontosságot elvileg korlátlanul növelhetjük. A gyakorlatban azonban nem poligonokat, hanem pontokat látunk, melyek végül egységes képpé állnak össze. A valóságot a szemünk - mint már leírtuk - kb. 50 millió ponttal közelíti. Tulajdonképpen elegendő lenne a testeket a számítógépes analízis esetén is pontokkal közelíteni. Annyi ponttal, amennyi elegendő ahhoz, hogy a képernyő minden pontjára legalább egy pontot le tudjunk képezni. Így a kapott kép is folyamatos lenne. Ennek a megoldásnak is megvannak az előnyei és a hátrányai is. Előnye pl. hogy ebben az esetben nincs szükség a poligonfestésre, de igen nagy hátránya, hogy szabályos testek esetén is igen nagy tárterületet igényelne. Ezáltal - a fellépő sok számítás miatt - egyszerű testek ábrázolásánál igencsak megnövekedne a felhasznált gépidő mennyisége. Egyéb gondok is vannak ezzel a megadási móddal. Az adatbázis felvitele nehézkes és nehéz megtalálni azt a pontsűrűségi mértéket, amely még éppen elég ahhoz, hogy a kapott kép folyamatos legyen, de ne jusson egy képpontra az adatbázis több pontja is, mert ez utóbbi csak a redundanciát növeli.

Ennek ellenére érdemes néhány gondolat erejéig elidőzni még e lehetőség mellett. Ez az eset tulajdonképpen nem is áll olyan messze a poligonokkal való közelítéstől. Egy pontot ugyanis tekinthetünk elfajult poligonnak is, amelynek csupán egy csúcsa van. Ennek is van normálvektora, amely nem más, mint a testet az adott pontban érintő síkra az adott pontban állított, a testből kifelé mutató merőleges vektor.

Így az adatbázisban a pont koordinátái után a normálvektor koordinátái következnek. Ez elegendő a pont leírására. Ha megfigyeljük viszont a poligonos adatbázis szerkezetét, láthatjuk, hogy a két eset teljesen ugyanaz. A különbség csupán annyi, hogy ott a normálvektor koordinátái előtt több pont - a poligon csúcsainak - koordinátái álltak, itt csak egy pont koordinátái állnak előtte. Az adatbázis szervezése szempontjából ez teljesen ugyanaz, mivel általánosságban néhány rekordot egy normálvektort leíró rekord követ, majd egy lezáró rekord. Ilyen egységekből épül fel az adatbázis. Ezt azt jelenti, hogy egy file-on belül is vegyesen alkalmazhatjuk a poligonokat és a pontokat leíró rekordokat, mivel a kezelésük egységes. Így a test természete szabhatja meg, hogy melyik megadást válasszuk. A ponttal történő megadás esetén a normálvektor alapján állapíthatjuk meg, hogy a pont látható-e, fényben vagy árnyékban van. Ez szabja meg a színét és tónusát is.

A poligonokkal való leírás esetén viszont elengedhetetlenül szükséges valamilyen poligonfestő algoritmust készíteni. Ennek matematikai elméletéről a (4.3) részben már volt szó. Egy konvex poligonokat festő algoritmust valósít meg a következő példánkban a 'Festn' -nevű modul. Egy általános poligont a 't' tömbben eltárolunk a csúcspontjainak képernyő-koordinátái alapján. A modul a poligonnak elméletben meghúzza az egy csúcsából kiinduló átlóit. Ezáltal a poligont háromszögekre szeletelte fel. Minden ilyen háromszögre meghívja a 'Fest3' nevű modult, amely egy háromszöget fest be adott színűre.

Ez utóbbi úgy működik, hogy a háromszöget felvágja vízszintes csíkokra, a képernyő sorai szerint. Megvizsgálja, hol metszi az adott sor a háromszög oldalát. Ez két pontot jelent (kivéve, ha éppen egy csúcspontban vagyunk); ezeket összeköti a megadott színnel, ily módon beszínezi az egész háromszöget. Az alábbi program egyúttal egy példát is ad, amelyet ki is próbálhatunk a gyakorlatban is. A 't' tömbbe eltároljuk a képernyőn lévő négy pont koordinátáit. A négy pont - mint csúcspont - által megadott négyszöget a Festn eljárás a 2-vel jelölt színűre kifesti, a Huzn eljárás pedig a négyszög kerületét a 3-mal jelölt színűvel húzza meg. Midkét eljárás második paramétere a tömbben lévő pontok száma. Ez esetünkben maximum 10 lehet, de - a memória szabta korlátig - a tömb méretét - s így az ábrázolandó poligon maximális oldalszámát is - növelhetjük, a 'tomb' -típus dimenziójának emelése által.

```
TONGEN:PROC OPTIONS(MAIN);
DCL T(256) FIXED(7) DEC;
     /* ebben lesznek az 1-256 egész számok, véletlen szerinti
        sorrendben rendezve */
DCL C4      CHAR(4);
DCL C16     CHAR(16);
DCL C64     CHAR(80);
     /* c256 hexadecimális alakja, 1 hexadecimális számjegy 4 bi-
        náris számjegynek felel meg */
DCL C256    CHAR(256);
     /*  256 bináris számjegy, azaz mindegyike: 0 vagy 1 */
DCL ICK     PIC'999' DEF C64 POS(73);
     /*  ciklusváltozó, de egyúttal az eredmények könyvtárbeli
         tárolását segíti (ld. később) */
DCL HEX(16) CHAR(4) DEF C64;
     /*  c64 -re rádefiniálunk 16 db, 4 számjegyű hexadecimális
         számot */
DCL TR(16) CHAR(16) DEF C256;
     /*  c256 -ra rádefiniálunk 16 db, 16 számjegyű bináris
         számot */
DCL TT(256) CHAR(1) DEF C256;
     /*  c256-ra rádefiniálunk 256, bináris számjegyet */
DCL F FILE OUTPUT RECORD ENV(F(80) CONSECUTIVE);
     /*  az OS operációs rendszer szerinti, szabályos könyvtári
         formátumú file, egy rekordján c64 egy értékét tároljuk,
         azaz egy tónus leírását */
SUBSTR(C64,73,8)='00000000';
     /*  a könyvtárazást elősegítő művelet */
OPEN FILE(F);
DO ICK=1 TO 256;
   DO I=1 TO 256;
      TT(I)='0';
   END;
     /*  c256 minden jegyének '0' kezdeti értéket adunk */
   CALL VREND(T);       /*  véletlen rendezés */
   DO I=1 TO ICK;
      TT(T(I))='1';
   END;
     /*  ahol T(1)-T(256) közül 1-es; azt az indexét TT -nek,
         azaz annyiadik jegyét c256-nak 1-essé változtatjuk, ez
         16 db 16 jegyű bináris számot jelent */
```

```
   DO J=1 TO 16;
      C16=TR(J);
      CALL AT(C16,C4);
      HEX(J)=C4;
   END;
     /* sorra hexadecimálissá konvertáltuk a 16 bináris számot */
   WRITE FILE(F) FROM(C64);
     /*  az eredményt az 'f' file-on eltároljuk  */
END;
CLOSE FILE(F);
/* ------ */
VREND: PROC(T);
     /*  az 1-256 számok véletlen rendezése  */
     /*  ezeket a t-tömb tartalmazza  */
 DCL T(256),T1(256) FIXED(7) DEC;
 DCL PS PIC'999V.9999';
 DCL PS1 PIC'999' DEF PS;
 DO I=1 TO 256;
    T1(I)=I;
 END;
     /*  a T1-tömbben vannak a számok 1-256 -ig  */
 J=255;
 L=0;
CIM:
 IF L>=256 THEN GOTO KI;
 PS=J*VG+1;
     /*  a 'vg' -véletlenszám-generátorral  a még meglevő  'j' db
         1-256  közti  egészből kiválaszt  véletlenszerűen egyet,
         mivel 'PS' egész része 'PS1':  */
 I=PS1;
 L=L+1;
 T(L)=T1(I);
     /*  a kiválasztott számot 'T' következő elemébe teszi  */
 DO K=I TO J+1;
    T1(K)=T1(K+1);
 END;
     /*  T1-ből a most kiválasztottat elhagyja,  csak 'j' db szám
         marad, a T(1)-T(J) elemekben  */
 J:=J-1;
 GOTO CIM;
KI:
END VREND;
/* ------ */
```

```
VG:PROC;
     /*   véletlenszámgenerátor, a századmásodpercek alapján   */
 DCL S STATIC;
 DCL P9 PIC'99V.(6)9' STATIC;
 DCL IA PIC'999' DEF P9 POS(4);
 DCL IB PIC'999' DEF P9 POS(7);
     /* a tizedespont utáni 1.,2.,3. illetve 4.,5.,6. számjegy */
 DCL TIME BUILTIN;           /*  idő-változó  */
 IF S<>1.111 THEN DO;
    S=1.111;
    P9=3.141592;
 END;
     /*   kezdőérték  P9-nek,  melyet  csak  először  kap meg   */
 IA=IA+SUBSTR(TIME,7,1);                /*  az idő 7. számjegye   */
 IB=IB+SUBSTR(TIME,8,1);                /*  az idő 8. számjegye   */
                                  /*  azaz századmásodpercek   */
 P9=MIN(IA,IB)/MAX(MAX(IA,IB),1);
     /*   0-val való osztás ne történjen   */
 RETURN(P9);
END VG;
/* ------ */

AT:PROC(C16,C4);
/*  bináris-hexadecimális konverzió  */
 DCL C16 CHAR(16);
 DCL C4,V4 CHAR(4);
 DCL V1 CHAR(1);
 DCL C1(4) CHAR(4) DEF C16;          /*  4 db  bináris  szám   */
 DCL C2(4) CHAR(1) DEF C4;       /*  4 hexadecimális számjegy   */
 DO IAT=1 TO 4;
    V4=C1(IAT);
    IF V4='0000' THEN V1='0';
    IF V4='0001' THEN V1='1';
    IF V4='0010' THEN V1='2';
    IF V4='0011' THEN V1='3';
    IF V4='0100' THEN V1='4';
    IF V4='0101' THEN V1='5';
    IF V4='0110' THEN V1='6';
    IF V4='0111' THEN V1='7';
    IF V4='1000' THEN V1='8';
    IF V4='1001' THEN V1='9';
    IF V4='1010' THEN V1='A';
    IF V4='1011' THEN V1='B';
```

```
      IF V4='1100' THEN V1='C';
      IF V4='1101' THEN V1='D';
      IF V4='1110' THEN V1='E';
      IF V4='1111' THEN V1='F';
      C2(IAT)=V1;
       /*  a soron következő  4 bináris számjegyből 1 hexadecimális
           számjegyet konvertáltunk  */
  END;
 END AT;
 /* ------ */
END TONGEN;
```

A következő, Fortran nyelvű program az előbb kialakult tónusokat szemlélteti. Egy nagy alakú négyzetrács elemeibe helyezi a tónusokat. Részletesebben a kommentekben.

```
      DIMENSION IPAT(4096),IPATA(16),XX(1000),YY(1000)
      IL=0
      LL=0
      N=256
C         n db rekord van
      INP=1
C         inp: a rajzolási vastagság, itt 1-es, a legkisebb
      J=0
      DO 20 I=1,N
C         minden rekordot beolvasunk, az egy rekordon lévő 16 hexa
C         számot az  IPATA tömb 16 elemébe,  majd ki is írjuk ezt:
      READ(15,61)(IPATA(K),K=1,16)
61    FORMAT(16Z4)
      WRITE(6,65)(IPATA(K),K=1,16)
65    FORMAT(10X,16Z4)
C         áttesszük a 16 hexa számot IPAT következő 16 elemébe
      DO 201 K=1,16
201   IPAT(J+K)=IPATA(K)
20    J=J+16
C         inicializáljuk a plottert, 1. rajzolási vastagsággal:
      CALL PLOTS(0,0,0)
      CALL NEWPEN(INP)
      L=0
```

```
      DO   21   J=1,16
      DO   210  I=1,16
C          egy (majdnem négyzet) téglalapot  4 csúcspontjával kije-
C          löljük, minden téglalapot  egy-egy  tónussal  ki  fogunk
C          tölteni
      XX(1)=40*I
      XX(2)=XX(1)+35
      XX(3)=XX(2)
      XX(4)=XX(1)
      YY(1)=40*J
      YY(2)=YY(1)
      YY(3)=YY(2)+40
      YY(4)=YY(3)
C          a téglalap két oldalát meghúzzuk:
      CALL PLOT(XX(1),YY(1),3)
      CALL PLOT(XX(2),YY(2),2)
      CALL PLOT(XX(3),YY(3),2)
      L=L+1
      KREC=16*L-15
C          IPAT tömb  KREC -edik  elemétől 16 elem adja a tónust, a
C          TONE -modulnak negatív előjellel adjuk meg az elemszámot
      CALL TONE(0.,0.,IPAT(KREC),-16)
C          az  XX és YY  tömbökben található  a téglalapok aktuális
C          koordinátái,  az aktuális  téglalapba  az aktuális tónus
C          kerül
      CALL TONE(XX,YY,4,1)
210   CONTINUE
21    CONTINUE
      CALL PLOT(0.,0.,999)
C          a plotter működését befejezzük
      STOP
      END
```

5.7. *A perspektív leképezés Turbo-Pascal megvalósítása*

A (3.6.) részben összefoglaltuk a perspektív leképezések algoritmusát. Most ennek alapján elkészítjük a tárgyközpontú perspektív leképezés algoritmusának Turbo-Pascal megvalósítását. Először a (3.4.5.) rész (vi) és (vii) egyenletei alapján meghatározott c_x, c_y, c_z együtthatókat definiáljuk, alkalmazva az l_x, l_y, h_x transzformációs együtthatókat. Ezután a (3.4.5.)-beli (x) alapján meghatározzuk a transzformációs együtthatókat, alkalmazva még a (3.5.4.) műveletcsökkentési eljárást is. Az említett együtthatóknak megfelelő változókat rendre a következőképpen nevezzük: cx,cy, cz,lx,ly,hx, x0,x1,x2, y0,y1,y2,y3, z0,z1,z2,z3. Szándékosan nem alkalmazunk rájuk tömböket, az egyszerűség miatt. A szem, illetve a nézett pont, valamint a szem-tárgy vektor koordinátáira ezután már logikus az következő jelcsoport: p1,p2,p3, s1,s2,s3, v1,v2,v3. Minden említett tényezőt valósnak deklarálhatunk. Nézzük az algoritmus azon részét, amely a transzformációs együtthatókat előállítja. Ezt csak egyszer kell definiálni egy leképezés esetén, tehát itt kell elérnünk, hogy az a rész, amelyben minden pontra műveletet végzünk, a lehető legegyszerűbb legyen, mert ez döntően befolyásolja az algoritmus gyorsaságát. Tegyük fel, hogy néhány változó már kapott kezdeti értéket, akár a programban, akár interaktívan, ezek a szem és a nézett pont koordinátái: p1, p2, p3, s1, s2, s3. Minden mást az algoritmus alapján a program számol ki:

```
      { a szem-tárgy vektor koordinátáinak definiálása: }
v1:=p1-s1;                v2:=p2-s2;          v3:=p3-s3;
d :=v1*v1+v2*v2;          { d, e: segédváltozók }
e :=d+v3*v3;
if d>0 then begin
   cy:=1/sqrt(d);         { cx, cy, cz: segédváltozók }
   cz:=1/sqrt(e);
   cx:=cy/cz;
      { nagyítási, felbontási együtthatók: lx, ly, hx }
   cx:=cx*hx*lx;
   cy:=cy*ly;
      { transzformációs együtthatók definiálása: }
   x0:=(s1*v2-s2*v1)*cx;
   x1:=-v2*cx;
   x2:=v1*cx;
```

```
        y0:=((s1*v1+s2*v2)*v3-s3*d)*cy;
        y1:=-v1*v3*cy;
        y2:=-v2*v3*cy;
        y3:=d*cy;

        z0:=(v1*p1+v2*p2+v3*p3)*cz;
        z1:=-v1*cz;
        z2:=-v2*cz;
        z3:=-v3*cz;
    end
    else begin
        x0:=-p1*p3*hi*lx;
        x1:=p3*hi*lx;
        x2:=0;

        y0:=-p2*p3*ly;
        y1:=0;
        y2:=p3*ly;
        y3:=0;

        z0:=s3+p3;
        z1:=0;
        z2:=0;
        z3:=-1;
    end;
          { műveletcsökkentés: }
    if x2<>0 then begin
        x0:=x0/x2;              x1:=x1/x2;
        y0:=y0/x2;              y1:=y1/x2;
        y2:=y2/x2;              y3:=y3/x2;
        z0:=z0/x2;              z1:=z1/x2;
        z2:=z2/x2;              z3:=z3/x2;
    end;
```

A műveletcsökkentés eredményeképpen minden pont transzformálásánál egy szorzást megtakarítunk, ugyanis $x_2 \neq 0$ esetén a kijelölt műveletcsökkentés után $x_2 \equiv 1$, azaz nem kell vele szoroznunk ($x_2 \cdot q_2$ helyett csak q_2 áll). Még jobb a helyzet akkor, ha $x_2=0$, mivel ekkor két művelettel, egy szorzással és egy osztással is csökkenthetjük transzformált pontonként a műveletek számát. Ekkor a (3.6.5.) egyenletrendszerben lévő $x_2 \cdot q_2 + x_1 \cdot q_1 + x_o$ helyett csupán $x_1 \cdot q_1 + x_o$ is elegendő. Ezért a következő programrészben $x_2=0$ kérdésnek megfelelően választjuk ketté a pontonkénti transzformációkat leíró részt.

A következő programrészlet bemutatja a csúcspontonként elvégzendő perspektív transzformáció megvalósítását. Ebben a programrészletben szerepel néhány olyan változó, amelynek már egy előző szakaszban értéket adtunk. A színekre vonatkozó változók tulajdonképpen a szín számát tartalmazzák. A programrészletben levő változók jelentése a következő:

takar: Ha értéke nulla, nincs takarás, azaz a takart élek is láthatók, a látható élektől megkülönböztetett színnel. Ekkor a test átlátszó, üvegszerű. Ha értéke nem nulla, akkor csak a látható poligonok lesznek megrajzolva, így a test tömör lesz.

hx: hx = 1, akkor szimpla, hx = 2, akkor dupla felbontás

pal: paletta száma

hszin: a képernyő háttérszíne

tszin: a takart poligonok éleinek színe, ha azok láthatók; illetve a látható poligonok éleit kihúzó szín, ha a takart élek nem láthatók

lszin: a látható élek színe

dszin: HiRes esetén a rajz színe

szv: színváltozó, mindig felveszi az aktuális szín értékét

x0,x1,x2,y0,y1,y2,y3,z0,z1,z2,z3: az előzőekben már ismertetett transzformációs együtthatóknak megfeleltetett változók.

c: részeredményeket tároló segédváltozó

poli: Olyan mező, amely tartalmazza az ábrázolandó testet, alább részletezett formában. Tegyük fel, hogy a test a felszín-file-ban volt, melynek szerkezetét a 6. fejezet ismerteti. A 'poli'-mező szerkezete ehhez nagyon hasonló, ezért ebbe könnyen be tudjuk tölteni a file -ból az adatokat. Tegyük fel, hogy ez az egyszerű betöltési művelet már egy korábbi programszakaszban meg is történt, azaz a test már a 'poli' mezőn található. Mivel a test a memóriában van, már nem kell I/O -műveleteket alkalmazni, tehát gyorsabb lesz az algoritmus. Most nézzük a 'poli' nevű mező szerkezetét. Deklarálása egyszerűen a következő lehet:

```
poli: array [1..npoli] of real;
```

Itt 'npoli' egy egész konstans, amellyel a 'poli' -mező maximális méretét határozzuk meg. A 'poli'-mezővel kapcsolatban deklarálunk még néhány más változót is, a következő módon (jelentésüket a deklaráció után részletezzük):

```
type
  rek          = record
                   n, feny    : integer;
                   n1, n2, n3 : integer;
                   q : array [1..ntomb, 1..3] of integer;
                 end;
  rekpointer = ^rek;
var
  pr : rekpointer;
```

Ez a deklaráció egy poligont ír le, az alábbi módon:

pr: Címváltozó; ennek tartalma az a cím, amelyen az adott poligon a tárban kezdődik. A 'poli' -mezőn egymás után vannak ezek a poligonok, 'pr' kezdeti értéke 'poli' címe. A következő poligon címét az előző poligon 'n' értékéből számíthatjuk ki.

n: Abszolút értéke éppen annyi, mint a poligon csúcsainak száma. Ha n>0, akkor a poligon látható, ha n<0, akkor takart.

feny: Ha feny=1, akkor a poligon a test fényes oldalán van, egyébként az árnyékos oldalán.

n1,n2,n3: a poligon normálvektorának koordinátái

q: A poligon csúcspontjainak koordinátáit tartalmazó tömb. Egy csúcs (az i-edik, x, y, z irányú) koordinátái: q[i,1], q[i,2], q[i,3].

ntomb: Konstans, egy poligon csúcsainak maximális száma. Az adott poligon csúcsainak tényleges száma azonban csak: Abs(n); ez nem nagyobb, mint 'ntomb' értéke. Így a következő poligon címe: Addr(q[Abs(n)+1,1]). Így a poligonok sorban követik egymást a 'poli' -mezőn, amely tulajdonképpen csak a helyet jelöli ki számukra a memóriában.

Eddig a 'poli' -mezővel kapcsolatos változókat részleteztük, most nézzük a többit.

nn: a poligonok száma

i: az aktuális poligon száma

t: Tömb, mely az aktuális poligon csúcsainak perspektív transzformált koordinátáit tartalmazza. Így deklarációja a következő:

```
type
  tomb = array[1..10,1..2] of integer;
var
  t : tomb;
```

Ez összhangban áll az (5.5.) részbeli 't' tömbbel, ebbe a tömbbe kerülnek a poligon csúcsainak képernyő-koordinátái. A tömböt a poligonfestő 'Festn', illetve a poligon-kihúzó 'Huzn' eljárásoknak adjuk át.

j: segédváltozó, mely jelzi, hogy az adott poligon melyik csúcsánál járunk.

Festn: Poligonfestő eljárás, amelyet az (5.5.) részben írtunk le. Most a következőképpen hívjuk meg: Festn(t,n,szv), ahol 't' tartalmazza a fentiek szerint a poligon koordinátáit, 'n' a poligon csúcsainak száma (n ez esetben mindig pozitív, mert a megfestett poligon a test látható oldalán van), 'szv': ilyen színű lesz a poligon.

Huzn: Poligonkihúzó eljárás, amelyet az (5.5.) részben írtunk le. Most a következőképpen hívjuk meg: Huzn(t,n,szv), vagy Huzn(t,-n,szv). Akkor is szükség van rá, ha a poligon területét és kerületét más színűre festjük. Itt 't' tartalmazza a fentiek szerint a poligon koordinátáit, 'n' a poligon csúcsainak száma (n ez esetben lehet pozitív, vagy negatív, aszerint, hogy a poligon látható, vagy takart. 'szv': ilyen színnel lesz a poligon kerülete kihúzva.

tx,ty: Eltolási tényezők; a képernyőn ennyivel toljuk jobbra, illetve felfelé a rajzot.

Ezzel leírtuk a változók rendeltetését. További információkat a kommentekben találhatunk még. Ez a programrész össze is foglalja a látás analíziséről leírt elméleti vizsgálataink jelentős részét. Nézzük ezután a programrészletet: (a kommenteket most dőlt betűkkel írjuk, hogy jobban elkülönüljenek a program szövegétől)

```
{ üzemmódok, színek beállítása: }
if hx=1 then begin
   GraphColorMode;
   Palette(pal);
   GraphBackGround(hszin);
end
else begin
   HiRes;
   HiResColor(dszin);
   szv:=1;
end;
```

```
if x2<>0 then begin
   {ez esetben a műveletcsökkentés miatt - mint fentebb leír-
   tuk - a (3.6.5.) egyenletrendszerben lévő x2q2+x1q1+xo he-
   lyett csupán q2+x1q1+xo is elegendő. }

   if takar=0 then begin
      {nincs takarás, a test üvegszerű}
      szv:=tszin;                          {aktuális szín: tszin}
      pr:=Addr(poli);                      {az 1. poligon címe}
      if  hx=2  then szv:=1;               {HiRes -nél a szín}
      for i:=1 to nn do  begin
         {minden poligonon végigmegyünk:}
         with pr^ do begin
            if n<0 then begin
               {takart poligon, csak ezeket rajzoljuk meg}
               for j:=1 to Abs(n) do begin
                  {végigmegyünk a poligon élein:}
                  c:=z3*q[j,3]+z2*q[j,2]+z1*q[j,1]+z0;
                  {Trunc: egészrész-függvény}
                  t[j,1]:=Trunc((q[j,2]+x1*q[j,1]+x0)/c)+tx;
                  t[j,2]:=
                  Trunc((y3*q[j,3]+y2*q[j,2]+y1*q[j,1]+y0)/c)+ty;
                  {a képernyő y-koordinátája lefelé növekszik, így
                  a kép fejjel lefelé áll,  ezért vissza kell for-
                  dítani:}
                  t[j,2]:=200-t[j,2];
               end;
               {kihúzzuk a poligon kerületét;  mivel a poligon ta-
               kart, ezért: n<0 }
               Huzn(t,-n,szv);
            end;
            pr:=Addr(q[Abs(n)+1,1]);
            {a poligon láthatóságától függetlenül, a következő po-
            ligon címére ugrottunk}
         end;
      end;
   end;
end;

{Eddig - ha üvegszerűre akartuk rajzolni a testet, azaz takar=0
esetben -  csak a takart poligonokat rajzoltuk meg.}
```

```
   {Most a látható poligonok következnek:}
   if lszin<>0 then begin
      {látható poligonok rajzolása, ha színük nem olyan, mint a
      háttérszín}
      szv:=lszin;                          {aktuális szín: lszin}
      pr:=Addr(poli);                      {az 1. poligon címe}
      for i:=1 to nn do begin
      {minden poligonon végigmegyünk:}
         with pr^ do begin
            if n>=0 then begin
               {látható poligon, csak ezeket rajzoljuk meg}
               if feny=1 then szv:=lszin else szv:=aszin;
               {ha a fényes oldalon van, színe: lszin, ha az
               árnyékoson, akkor színe: aszin}
               if hx=2 then szv:=1;    {HiRes -nél a szín}
               for j:=1 to n do begin
                  {végigmegyünk a poligon élein:}
                  c:=z3*q[j,3]+z2*q[j,2]+z1*q[j,1]+z0;
                  t[j,1]:=Trunc((q[j,2]+x1*q[j,1]+x0)/c+tx);
                  t[j,2]:=
                  Trunc((y3*q[j,3]+y2*q[j,2]+y1*q[j,1]+y0)/c)+ty;
                  {a képernyő y-koordinátája lefelé növekszik, így
                  a kép fejjel lefelé áll, ezért vissza kell for-
                  dítani:}
                  t[j,2]:=200-t[j,2];
               end;
               if takar<>0 then begin
                  {ha tömör a test, akkor kifestjük és éleit ki-
                  éleit kihúzzuk, ha üvegszerű, akkor csak az éle-
                  ket húzzuk ki}
                  Festn(t,n,szv);
                  Huzn(t,n,tszin);
               end
               else Huzn(t,n,szv);
            end;
            pr:=Addr(q[Abs(n)+1,1]);
         end;
      end;
   end;
else ... end;   {következne: x2=0 eset, de ez az előzőektől csak
abban különbözik, hogy q[j,2]+x1*q[j,1]+x0 helyett x1*q[j,1]+x0
kerül mindenütt, mert x2*q[j,2]=0 }
```

5.8. *Takarás, fény-árnyék hatások*

A (4.1.) részben megadtuk annak kritériumát, hogy az objektum felszínének mely része látható egy adott pontból és melyik nem. Ebben kulcsszerepe volt a normálvektoroknak. Eszerint egy poligon - amely része a felszín közelítésének - akkor látható, ha a szem-tárgy vektor és a poligon normálvektora -90° és 90° közé eső szöget zár be, azaz a skaláris szorzatuk pozitív. Ezt a lokális láthatósági kritériumot fogalmazzuk most meg Turbo-Pascal nyelven. Először tekintsünk egy olyan függvényeljárást, amely egy poligon három különböző, tetszőleges pontja alapján megadja a normálvektor koordinátáit. Legyen a három pont: A, B, C, a normálvektor: N. Mindegyiknek három koordinátája van. A pontok koordinátáit valós változóban tároljuk, hogy a részletszámítások közben ne lépjen fel túlcsordulás. A normálvektor koordinátáit egész számokként kapjuk, ezért konverzióra és újabb túlcsordulás -vizsgálatra is szükség van. Ezekkel a rutinfeladatokkal most nem foglalkozunk; az algoritmusnak csak az érdemi részét írjuk le. A függvényeljárás a normálvektor hosszát is visszaadja:

```
type
  rtomb = array[1..3] of real;
  itomb = array[1..3] of integer;

function Normal(var a,b,c: rtomb; var n: itomb): integer;
var
  r : real;
begin
  n[1]:= Trunc((b[2]-a[2])*(c[3]-a[3])
              -(b[3]-a[3])*(c[2]-a[2]));
  n[2]:= Trunc((b[3]-a[3])*(c[1]-a[1])
              -(b[1]-a[1])*(c[3]-a[3]));
  n[3]:= Trunc((b[1]-a[1])*(c[2]-a[2])
              -(b[2]-a[2])*(c[1]-a[1]));
  r:=Sqrt(n1*n1+n2*n2+n3*n3);
  if  (r>-32768) and (r<32767) then Normal:=Trunc(r)
      else Normal:=0;
      { Normal:=0 akkor is, ha túlcsordulna }
end;
```

A következő függvényeljárás két tetszőleges vektor szögének cosinusát számolja ki. Ez is szükséges lesz a láthatóság eldöntéséhez. A vektorok koordinátái a 'v' és a 'w' tömbökben találhatóak, melyeknek típusa az eddigiekkel összhangban 'itomb'.

```
function Vektor_cos(var v,w: itomb):real;
var
  av,aw,s: real;
begin
  {v,w vektorok abszolút értékei, azaz hosszúságuk:}
  av:=Sqrt(v[1]*v[1]+v[2]*v[2]+v[3]*v[3]);
  aw:=Sqrt(w[1]*w[1]+w[2]*w[2]+w[3]*w[3]);
  {skaláris szorzatuk:}
  s:=v[1]*w[1]+v[2]*w[2]+v[3]*w[3];
  {v,w által bezárt szög cosinusa:}
  Vektor_cos:=Trunc(s/(av*aw));
end;
```

Legyenek a szem, a vizsgált poligon normálvektora, valamint a poligon egy tetszőleges pontjának koordinátái rendre a 'p', 'n' és az 'a' tömbökben. Helyezzük el a $\overrightarrow{P-A}$ vektor koordinátáit a 'ps' tömbben. Megállapítjuk, látható-e a poligon:

```
ps[1]:= p[1]-a[1];
ps[2]:= p[2]-a[2];
ps[3]:= p[3]-a[3];
if Vektor_cos(n,ps)>0 then  {látható}
                      else  {takart};
```

Ha 'p' helyett a 'h' fényforrás-koordinátákkal hívjuk meg ezt az eljárást, akkor alkalmassá válik arra is, hogy eldönthessük, a poligon a test fényes, vagy árnyékos oldalán van-e:

```
if Vektor_cos(h,ps)>0  then {fényes}  else {árnyékos};
```

Ezt a vizsgálatot természetesen csak a látható poligonok etén kell elvégezni; nekünk mindegy, hogy a nem látható poligonok a test világos vagy sötét oldalán vannak. Mivel a Vektor_cos függvényeljárás két vektor cosinusát számítja ki, ezért ez az eljárás többre is alkalmas, mint a láthatóság vagy a fény-árnyék eldöntésére. A cosinus függvény értéke 0-tól 1-ig terjedhet. Esetünkben a poligon akkor kapja a legtöbb fényt, ha merőleges a fénysugárra.

Általában: annál több fényt kap, mennél kisebb szöget zár be a fénysugárra merőleges síkkal, azaz mennál kisebb szöget zár be a normálvektor a fénysugárral.

Ezt a tényt úgy is kifejezhetjük, hogy ha $\cos(\alpha) \rightarrow 0$, akkor a poligon világosodik, ha $\cos(\alpha) \rightarrow 1$, akkor a poligon sötétedik, ahol α: a poligon normálvektorának a fénysugárral bezárt szöge. Ez a megállapítás alkalmas a poligon tónusának kijelölésére. Így az egész testre vonatkoztatva lágy átmenet adható a test világos és sötét részei között, ha elegendő számú tónust generálunk. Ezáltal elérhető a fényképszerű, a valóságot jól szemléltető hatás.

Tegyük fel, hogy a tónusok száma: k, ahol az első a teljesen világos, a k-adik a teljesen sötét, a többi pedig a kettő közti fokozatos átmenet. A (2.4.) részben felírt egyenlet az alapja a (4.5.1.3.) összefüggésnek, amely a poligon fényerősségét definiálja, oly módon, hogy megadja a hozzá tartozó tónus számát. Ezt a következő programrészlettel írhatjuk le (r: real segédváltozó; i: integer, a poligon az i-edik tónust fogja kapni):

```
r:= Vektor_cos(h,ps);
if r>0  then i:= n*Trunc(1-r);  {a poligon fényes, az i-edik
                                  tónust kapja}
else {a poligon árnyékos};
```

Ezzel a látás mechanizmusának számítógépes szemléltetését befejeztük. A leírt - és más elveken alapuló - algoritmusok természetesen több irányban fejleszthetők. Mivel ezen algoritmusok némelyike igen sok számítást igényel, fontos célkitűzés lehet a gyorsításuk, mert minden jelentéktelennek látszó módosítással komoly gépidő-mennyiséget takaríthatunk meg a számítógépes grafika területén.

6. GRAFIKAI ADATBÁZIS SZERVEZÉSE, HÁROM DIMENZIÓS TESTEK TÁROLÁSÁRA

Ebben a részben a három dimenziós testek tárolására alkalmas adatbázis szerkezetével foglalkozunk. A javasolt szerkezet nem az egyetlen módja a testek tárolásának, mivel - különböző szempontok alapján - sokféle adatbázist felépíthetünk. A cél az, hogy a program és az általa használt adatbázis egységes rendszert alkossanak, vagyis az adatbázist úgy kell tervezni, hogy a leginkább segítse a feladat hatékony megoldását. Ezért most egy olyan adatbázist definiálunk, mely három dimenziós ábrázoláshoz eddig felépített rendszerünkhöz illeszkedik. Az eddigiekben már több ízben érintettük ezt a témát. A feladat támasztotta igények felmerülésekor már meg is fogalmaztunk néhány kérdést, amely egy ilyen adatbázis felépítéséhez közelebb visz. Részleteiben így már elég sokat el is mondtunk erről a kérdésről, most összefoglaljuk mindazt, amit eddig kialakítottunk a térbeli objektumok adatbázisáról.

Mivel egy ilyen adatbázis három dimenziós testeket ír le, ezért azt mondhatjuk, hogy a valóság egy igen elvont formában történő ábrázolását valósítja meg. Mivel nem minden testet tudunk leírni képletekkel, ezért szükség van olyan beviteli módszerre, amelynek eredményeképpen a valós alakzatok térbeli, absztrakt képét elő tudjuk állítani. Ezt valósítják meg a három dimenziós beviteli eszközök, melyeknek lényege: a test felületének pontjait valamilyen rendszer szerint feldolgozzák, megmérik a három dimenziós koordinátáikat, majd a kapott értékeket a gép ábrázolási módja szerinti formára hozva, egy file-ba gyűjtik. Ma már több ilyen eszköz létezik. Ilyen például a három dimenziós akusztikus tablet, amelyben a pontok három dimenziós koordinátáját három mikrofon méri akusztikus jelek időkülönbségei alapján. Más, hasonló célú eszköz még a Lincoln-pálca, illetve Burton-féle doboz. Ezek az eszközök digitalizálják a valós testek felületét. A felszínt poligonokra bontva, illetve valamilyen sűrűségben pontonként is megadhatjuk. A poligonok csak közelítik a felületet, bizonyos hibával, de maga a beviteli eszköz sem teljesen pontos. Mérések alapján dolgozik, ez pedig elkerülhetetlenül hibával jár. A pontosságot precízebb eszközzel, illetve a felbontás növelésével javíthatjuk. Az adatok egy file-ba kerülnek. Ez a file még 'nyers', azaz nem tartalmaz még azadatokra vonatkozó kiegészítő információkat, melyek a számításokat majd meggyorsíthatják. Ehhez szükséges egy konvertáló program, amely ebből a 'nyers' formából előállítja az adatbázisban előírt formátumot.

Ha nem létező, de képlettel vagy valamilyen matematikai módszerrel definiálunk egy alakzatot, azt is célszerű először egy, az előzőhöz hasonló, 'nyers' file-ba tenni. Ennek szerkezete igen egyszerű, egy rekordját három koordináta alkotja. Ezek állhatnak önmagukban, ha csak egy pontot ábrázolnak, de csoportjai poligonokat is alkothatnak. Ez esetben a poligon kezdetét és végét jelölni kell. Ebből állítjuk elő konvertálással az adatbázist. Az adatbázist a következő file-ok alkotják.

A legfontosabb file az alakzat felszínének absztrakt formáját tartalmazza. E felszín file szerkezete a következő. Egy koordináta értékét egy valós vagy egész számmal ábrázoljuk. A Turbo-Pascal nyelv ezeket 6, illetve 2 byte-on ábrázolja, ezért a valós számokkal történő ábrázolás nagy hátránya, hogy háromszor akkora területet igényel velük az adatbázis. Előnye viszont a nagy pontosság, másrészt az a körülmény, hogy egész számokkal e nyelvben csak az [-32768,...32767] intervallumon belül dolgozhatunk. Ez esetben vigyáznunk kell a túlcsordulásra. Általában egy adott test esetén érdemes mérlegelnünk, hogy egészekkel vagy valósakkal ábrázoljuk. A valós eset nagyobb pontossága miatt elvileg jobb minőségű képet kaphatnánk, de ez a hagyományos képernyők kis felbontása miatt nem érvényesülhet. Legtöbbször nem is lenne különbség a két eset között. Így célszerűbb az egész számokkal történő ábrázolás. Nagy felbontású képernyők esetén viszont ezt érdemes meggondolni.

Három ilyen koordináta adja a test felszínének egy pontját. Ez alkotja egyben a file egy rekordját. Egy poligont a csúcspontjaival ábrázolunk. Felsoroljuk a csúcspontokat oly módon, hogy ha a testen kívülről nézzük őket, akkor az óramutató járásával ellenkező irányban kövessék egymást. Tehát minden csúcspont egy rekordnak felel meg. A csúcspont-rekordok után két speciális rekord következik. Az egyik az adott poligonnak egy - a testből kifelé mutató - normálvektora, a másik pedig egy lezáró rekord. Ezt is három szám alkotja, melyek közül az első a gépi számábrázolás maximuma körüli értékű, a másik kettő pedig speciális szám. Ezek a takarást, ill. a fény-árnyék hatásokat leíró - később leírt file - azon elemére mutatnak, amelynél az adott poligonra való hivatkozás kezdődik. Ennek hiányában e számok értéke nulla. Ez is megkönnyíti a file-okon végzett műveleteket. Így egy poligont, amely n csúcsú, n + 2 rekordon ábrázolunk. Ha nem poligonokkal, hanem pontokkal dolgozunk, akkor egy pont tekinthető egy csúcsú poligonnak. Az eddigiek alkalmazhatók ebben ez esetben is, azaz a pont koordináta-rekordját a testből kifelé mutató, a testet az adott pontban

érintő sík normálvektorának koordinátáit tartalmazó rekord követi, végül pedig a lezáró rekord zárja a pontot leíró három rekordot. Ez azonban igen nagy helyet foglal, ezért a csak pontokat ábrázoló file esetén elegendő két rekord is egy pont definiálására. Vegyes file esetén - amely pontokat és poligonokat is tartalmaz - elegendő egy speciális record, amely jelzi, hogy poligon rekord-csoportok után pontokat leíró rekord-csoportok következnek. Így ezeknek is elegendő pontonként két rekord, de amikor ismét poligon rekord-csoportok következnek, azt speciális rekorddal jelezni kell, hogy programból követhessük, milyen típusú rekordoknál tartunk éppen. A file-ot az ilyen rekord-csoportok összesssége alkotja.

A másik file a (4.2.2.) részben említett referencia-listát tartalmazza. Erre konkáv testek, illetve egymást takaró testek esetén van csak szükség, ha nem adható meg egyértelműen, hogy melyik takarja a másikat (pl. bonyolult formájú testek esetén). Az említett file egy rekordja két számot tartalmaz, amelyet úgy kell értelmeznünk, hogy az 1. sorszámú poligon takarja a 2. sorszámú poligont. Ezek a sorszámok a testfelszínt leíró file azon rekordjainak számai, ahol az adott poligonok kezdődnek. A referencia-lista rekordjait logikailag törölhetjük, ezt pl. úgy érhetjük el, hogy a bennük szereplő rekord -sorszámok előjelét negatívra változtatjuk. Ez a file az objektum(ok) egy nézőpontból való ábrázolásával áll elő, tehát csak ideiglenes jellegű, mivel a szem, illetve a nézett pont megváltoztatásával ez a file is megváltozik.

A harmadik file az árnyék-poligonok file-ja. Erről a (4.5.2) részben esett szó. Egy poligont itt is a csúcspontjaival ábrázoljuk. Egy rekord egy csúcs három dimenziós koordinátáit tartalmazza. A poligont hasonlóan ábrázoljuk, mint a felszínt leíró file esetén. A különbség, hogy itt nem kell a normálvektort tartalmazó rekord, csak a lezáró rekord. Ennek második száma annak a poligonnak a sorszáma, amelyen az a poligon kezdődik a felszín-file-ban, melynek síkjában van. Ez a file is ideiglenes. Ugyanaz mondható el róla ebből a szempontból, mint az előző file-ra. Tehát az említett file-ok alkotják az adatbázist.

7. VIDEOTECHNIKA SZÁMÍTÓGÉPEN

7.1. *A digitális számítógépes film felépítése és tulajdonságai*

A mozgó, számítógépes film igen hasznos eszköz az elkészült grafikák tárolására, de amellett, hogy archíválási funkciót is betölt, sokkal lényegesebb feladata a mozgás bemutatása. Egy mozgó grafika mindig lényegesen több információt hordoz magában, mint a statikus, sokkal plasztikusabb az így kapott, mozgó kép, mint az álló. Az alakzatoknak olyan térbeliséget kölcsönöz, amelyet más eszközzel nem is lehetne ábrázolni. Ilyenformán a számítógépes grafika egyik 'csúcsának' is lehet tekinteni a három dimenziós filmeket. Egy ilyen filmet programmal tudunk lejátszani. Digitális jellege és más jellemzői miatt sok tulajdonságban eltér a hagyományos filmekétől. Ezenkívül az az elvárásunk is a számítógépes filmtől - pontosabban az őket lejátszó programtól - , hogy a hagyományos filmekkel szemben rendelkezzen bizonyos előnyökkel. Jogos igény pl. - a gép korlátai által megszabott határokig - a színek lehetőség szerinti összeállítása, változtathatósága, a film vetítési sebességének szabályozása, egyes képkockák kimerevíthetősége, valamint, hogy a filmet oda-vissza is lehessen játszani.

A számítógépes film alapja egy file, melynek egy rekordja a film egy kockája. Ezeket olvassuk be egymás után a képernyő területére. Az állóképek gyors egymásutánisága - mint a hagyományos filmek esetén is - mozgó hatást kelt. Másodpercenként 6-7 kocka is elegendő e hatás kiváltásához. Ez abból adódik, hogy az emberi szem kb. 1/6 másodpercig őrzi a látott képet. Így - ha elegendő a számítógép adatátviteli sebessége - a folyamatoas hatást itt is el tudjuk érni. Nem mindegy azonban, milyen adat -és programszerkezetet definiálunk ehhez a feladathoz.

Mint azt már az 5. fejezetben is láthattuk, a képernyő területét többféleképpen is leírhatjuk. A direkt címzésre legmegfelelőbb az e1 és e2 tömbökkel való leírás. Ennek egyszerűsített változata:

```
f1: array[1..8000] of byte absolute $b800:$0;
f2: array[1..8000] of byte absolute $ba00:$0;
```

Az ebben a szerkezetben készített filmnek előnye, hogy a képernyő területét a legpontosabban leírja, nincsenek redundáns memóriaterületek. Azonban igen nagy hátrány, hogy az e szerkezet melletti betöltés eredményeképpen a film vetítésekor hullámzás figyelhető meg, ezért a film vetítésére a leghelyesebb a következő képernyő-szerkezetet megadni:

```
srek: array[1..16192] of byte absolute $b800:$0;
```

Ez a képernyő-szerkezet csak 192 byte redundáns byte-ot tartalmaz, ez azonban a lefoglalt adatmezők méretéhez képest igen kevés, nem növeli jelentős mértékben a helyfoglalást. Nagy előnye viszont, hogy a teljes képernyőt, mint egységes egészt tudjuk így kezelni. Ebben a szerkezetben érhetjük el a legjobb vizuális hatást, a filmek minősége így lesz a lehető legjobb. Ennek a szerkezetnek adjuk az 'SCR' -elnevezést, mely legyen egyrészt az ilyen szerkezetű adatmező típusának neve, másrészt pedig a filmet tartalmazó file -ok kiterjesztésének egységes neve is. Ilyenformán a képernyő és a vele megegyező típusú adatmező, valamint a filmet tartalmazó képernyő-file deklarálása a következő:

```
type
   scr   = array[1..16192] of byte;
var
   srek  : scr absolute $b800:$0;
   puffer: scr;
   sfile : file of scr;
```

Tehát legyen a 'SCR' a képernyőt tartalmazó file típusának neve. Amennyiben az egész film befér a memóriába, érdemes ott tárolni, mivel onnan lényegesen gyorsabb az elérése, mint bármely háttértárról. Hosszú filmek viszont csak a háttértárban férnek el, egy mérét felett semmiképpen sem tölthetjük bele őket a memóriába. Nem mindegy viszont, hogy milyen háttértárról olvassuk be a filmet, mivel pl. winchester-ről lényegesen gyorsabb, mint floppy drive-ról. Másik fontos körülmény az, hogy a háttértárról beolvasva a film csíkozna, rossz minőségű lenne, éppen a lassú átvitel miatt. Szinte folyamatosan látnánk, hogy cserélődik ki a következő kocka tartalmával az előző. Ezért ez esetben közbe kell iktatnunk egy puffert, amelybe először beolvassuk a képernyő egy rekordját, - ez alatt az előző képkocka látható a képernyőn - majd pedig a pufferből olvassuk a képernyő területére. Így megszűnik a csíkos és gyors elérésű háttértár esetén jó minőségű filmet láthatunk.

7.2. Mozgó, forgó testek megjelenítése valós időben

Az alább bemutatott program a fentebb megfogalmazott elvek szerint működő interaktív filmvetítő rutin. A vetítést funkcióbillentyűkkel szabályozhatjuk. A program lényege: sorra vetíti a filmeket, mindegyiket ciklikusan, azaz néhányszor újra kezdi, majd automatikusan áttér a következőre. Ha minden filmen végigment, akkor az elsőnél újra kezdi a vetítést. Ebben a 'Vetit' nevű modul játszik kulcsszerepet, azaz ez képes egy ilyen film levetítésére. Bemenő paraméterei: a film neve kiterjesztés nélkül, mert a kötelező SCR- kiterjesztést a program hozzáolvassa. A második bemenő paraméter azt szabja meg, hogy milyen hosszú ideig vetítse a filmet, a harmadik pedig két film egymásba való átmenetét szabályozza. Négy lehetséges értéke van: 0,1,2,3. 0 esetén nincs átmenet, 1 esetén: addig, amíg a következő filmet beolvassa, az előzőt vetíti lassan, 2 hatása: a beolvasás alatt a következő filmet vetíti lassan, 3 esetén pedig kockánként felváltva mind a kettőt.

A program minden - az előző részben megfogalmazott - vetítési elvet figyelembe vesz, annak megfelelően működik. Így a képkockák beolvasásakor alkalmazott puffer az Stomb[1]. Az 'Stomb' egyébként a képkockák tömbje; ha a film 5 kockánál rövidebb, akkor a tömb feltöltődik a filmmel és a forgatás innen történik. Ha ennél hosszabb, akkor a háttértárról. A funkcióbillentyűk hatását itt nem részletezzük, mivel a program kommentjében ez megtalálható. A billentyűk kezelését a már ismert 'Fbill' nevű függvényeljárás látja el. A billentyűket kódjuk alapján szétválasztja, majd a 'Vizsgal' és a 'Vetit' függvényeljárások ennek alapján rendelnek különböző funkciót hozzá. A start-stop funkciót az 'Esc' is ellátja, ez is kiderül a program szövegéből. A gyorsítás-lassítás műveletében az 'i' változó kulcsszerepet játszik, mivel ennek értéke az az időtartam, amelyre két kocka között a program leállítja a filmet. A változók neve általában jelzi is a funkciójukat, így pl. a 'szin', 'pal' (palatta száma), 'hatter' (háttérszín), 'uzem' (ha értéke 0, HiRes-ben, ha 1, GraphColorMode-ban vagyunk), 'elore' (ha értéke 1, a filmet előre, ha 0, hátra vetíti), 'pnr' (jelzi, hogy hányadik filmkockán tartunk), 'size' (a filmet tartalmazó file mérete, azaz a képkockák száma a filmben). Srek: a képernyőt, Sfile pedig a filmet tartalmazó, ('scr' kiterjesztésű) file-ot jelenti. Külön kiemeljük az F8 és F9 billentyűk szerepét. F8 hatására egy kiemelt színösszeállítás jön létre, amellyel egyes testek tömörré válnak. F9 színjátszóvá, azaz kockánként más színűvé teszi a filmet, ez csak lassú vetítésnél lehet hatásos.

```
Program Vetit;
type
      szoveg = string[8];
      scr    = array[1..16192] of byte;
label
      olv,vege;

{-**********************-}

function Vetit(st:szoveg; long,villog:integer):integer;
 { Ez az eljárás egy filmet ciklikusan vetít }
 var
    i,pal,szin,hatter,uzem,elore,j,pnr,size,l:integer;
    srek:scr absolute $b800:$0;
    stomb:array[1..4] of scr;
    sfile:file of scr;
 label
    ind,ind1,hatra,hatra1,ki,vege;

{-------}
{ 'Vetit' belső eljárásai: 'Fbill','Szinjatszo','Vizsgal' }

 function Fbill:integer;
 {Ez az eljárás a funkcióbillentyűk kódját adja vissza}
 var
    c1:char;
 begin
    Read(kbd,c1);
    if Integer(c1)=27 then begin
       Read(kbd,c1);
       Fbill:=Integer(c1);
    end
    else Fbill:=0;
 end;

{-------}

 procedure Szinjatszo;   {A szín vagy a paletta értékét csökkenti}
 begin
    if uzem=1 then begin
       if pal<2    then pal:=4 else pal:=pal-1;
       Palette(pal);
    end
```

```
    else begin
       if szin<1   then szin:=15 else szin:=szin-1;
       HiresColor(szin);
    end;
end;

{-------}

 function Vizsgal:integer;
 { A billentyűkhöz funkciókat rendel }
 label
    vege,vissz;
 begin
    j:=Fbill;
    case j of
      60:begin                          {start-stop: F2}
        vissz:
           j:=Fbill;
           if (j=60)or(j=59)or(j=68) then  goto vege;
           goto vissz;
         end;

      61:if i>=10 then i:=i-10;         {gyorsít: F3}
      96:i:=0;                          {leggyorsabb: ctF3}
      62:i:=i+10;                       {lassít: F4}
      97:i:=i+100;                      {jobban lassít: ctF4}

      63:begin                          {paletta, szín: F5}
         if uzem=1 then begin
            if pal>=4   then pal:=1 else pal:=pal+1;
            Palette(pal);
         end
         else begin
            if szin>=15 then szin:=1 else szin:=szin+1;
            HiresColor(szin);
         end;
      end;
      98:Szinjatszo;                    {paletta, szín vissza: ctrlF5}

      64:if uzem=1 then begin           {hatterszín: F6}
            if hatter>=15 then hatter:=0 else hatter:=hatter+1;
            GraphBackGround(hatter);
      end;
```

```
      99:if uzem=1 then begin       {háttérszín vissza: ctF6}
            if hatter<=0 then hatter:=15 else hatter:=hatter-1;
            GraphBackGround(hatter);
      end;

      65:begin                              {üzemmód-váltás: F7}
         if uzem=1      then begin
            uzem:=0;
            HiRes;
            HiresColor(szin);
         end
         else begin
            uzem:=1;
            GraphColorMode;
            GraphBackGround(hatter);
            Palette(pal);
         end;
      end;

      66:begin
          {F8: kiemelt szinösszeállítás}
          {Az objektum látszólag tömör lesz}
         uzem:=1;  pal:=2; hatter:=10;
         Graphcolormode;
         GraphBackGround(hatter);
         Palette(pal);
      end;

       {75,72,77,80,68 kódú billentyűk:}
       {nyilak, F10: a vetítés irányát megváltoztatják}
      75,72:elore:=0;
      77,80:elore:=1;
      68:if elore=1 then elore:=0 else elore:=1;
    end;

   vege:vizsgal:=j;
 end;

{ 'Vizsgal' belső eljárásainak vége }
{-----}
```

```
begin
   { a film-file  megnyitása, ellenőrzése }
   Assign(sfile,st+'.scr');
   {$i-}Reset(sfile);{$i+}
   if IoResult<>0 then begin
      Gotoxy(1,1);    Write(st,':A file hianyzik!');
      Delay(2500);    goto vege;
   end;
   size:=FileSize(sfile);
   if size<1 then begin
      Gotoxy(1,1);    Write(st,':A file serult!');
      Delay(2500);    goto vege;
   end;

   { változók inicializálása: }
   hatter:=blue;      pal:=1;
   szin:=1;           uzem:=1;
   elore:=1;
   i:=0;              j:=0;

   { rövid filmek esete: }
   if size<=4 then   begin              {size: a filmkockák száma}
      { a film betöltése az 'stomb' -be: }
      for pnr:=0 to size-1 do begin
                 {ha 'villog' értéke:
                  0: nincs átmenet, 1: régi, 2: új, 3: mindkettő}
          if (villog=1)or (villog=3) then srek:=stomb[pnr+1];
                                                          {régi}
          Seek(sfile,pnr);
          Read(sfile,stomb[pnr+1]);
          if villog>1 then srek:=stomb[pnr+1];                {új}
      end;

      { rövid filmek vetítése: }
     ind:
      for l:=1 to long do begin
          if elore=1 then for pnr:=1 to size do begin
             if KeyPressed then goto ki;
             srek:=stomb[pnr];
             Delay(i);
             if j=67 then Szinjatszo;        {F9}
          end
```

```
        else begin
           { vetítés visszafelé: }
           pnr:=size;
          hatra:
           if KeyPressed then goto ki;
           srek:=stomb[pnr];
           Delay(i);
           if j=67 then Szinjatszo;          {F9}
           pnr:=pnr-1;
           if pnr>0 then goto hatra;
        end;
    end;
    goto vege;
  ki:
    j:=Vizsgal;
    if (j<>59)and(j<>79)and(j<>83) then goto ind;
                                    {F1, End, Del: eljárás vége}
end

{ hosszú filmek esete: }
else begin
    size:=size-1;
    l:=0;
  ind1:
    if l>long then goto vege;
    if elore=1 then for pnr:=0 to size do begin
       { vetítés előre: }
       if KeyPressed then begin
          j:=Vizsgal;
          l:=0;
          if (j=59)or(j=79)or(j=83) then goto vege;
                                    {F1, End, Del: eljárás vége}
          if (j=75)or(j=72)or(j=68) then goto ind1;
                                                {nyilak, F10}
       end;
       if j=67 then Szinjatszo;                 {F9}
       Seek(sfile,pnr);
       Read(sfile,stomb[1]);
       srek:=stomb[1];
       Delay(i);
    end
```

```
        else begin
           { vetítés visszafelé: }
           pnr:=size;
          hatra1:
           if KeyPressed then begin
              j:=Vizsgal;
              l:=0;
              if (j=59)or(j=79)or(j=83) then goto vege;
                                                    {F1, End, Del}
              if (j=77)or(j=80)or(j=68) then goto ind1;
                                                    {nyilak, F10}
           end;
           if j=67 then Szinjatszo;                 {F9}
           Seek(sfile,pnr);
           Read(sfile,stomb[1]);
           srek:=stomb[1];
           Delay(i);
           pnr:=pnr-1;
           if pnr>=0 then goto hatra1;
        end;
        l:=l+1;
        goto ind1;
     end;
    vege:
     Close(sfile);
     Play:=j;
 end;

{---  'Vetit' vége ---}

begin
    GraphColorMode;
    GraphBackGround(1);
    Palette(3);
    { ciklusban vetítjük a filmeket: }
 olv:
    if Vetit('film1',30,2)=79  then goto vege;
    if Vetit('film2',30,2)=79  then goto vege;
    if Vetit('film1',30,2)=79  then goto vege;
    goto olv;
vege:
    TextMode;
end.
```

7.3. *Számítógépes filmek kopírozása*

Az (5.4.) részben már szerepelt egy kopírozást megvalósító modul. Ennek egy továbbfejlesztett változata az a program, amely számítógépes filmeket tud kopírozni. Ez igen hasznos lehetőség, mert sok trükkfilmet is elő tudunk állítani ezzel és ehhez hasonló más programokkal. A térábrázolásban is van jelentősége, mert ha azt tudjuk, hogy két mozgó test közül az egyik mindig eltakarja a másikat, akkor kopírozással még jobb térhatású filmeket készíthetünk, mint az alapfilmek, amelyekből kiindulunk. A film kopírozása is igen gyors művelet.

Az alább leírt program tartalmazza az (5.4.) részben leírt 'Byte_Kopiroz' és 'Scr_Kopiroz' modulokat. Scr típusú képernyőt használ. Három file-lal dolgozik, ezek: s1file, s2file, s3file. Az s1file és s2file képeit montírozza egybe az s3file-on, melynek hossza akkora lesz, mint a két film közül a rövidebbik. Mindezt a 'Film_Kopiroz' -modul hajtja végre. Ennek elején input-output ellenőrzéseket végez, majd inicializálja az (5.4.) részben leírt 'a' és 'c' tömböket. Mivel ezek csak egyszer kapnak értéket a program folyamán - és ez meg is marad - ez még inkább gyorsítja ezt a programot. Ezután ciklusban beolvassa a filmkockákat, elvégzi a kopírozást, majd a file-ok lezárásra kerülnek.

```
Program Kopiroz;
{$c-}
type
     scr      = array[1..16192] of byte;
     szoveg   = string[12];

procedure Film_Kopiroz(st1,st2,st3:szoveg);
var
     srek:  scr  absolute  $b800:$0;
     s1rek, s2rek:          scr;
     s1file,s2file,s3file: file of scr;
     size,size2,i,j:        integer;
     b:     byte;
     a,c:   array[1..4] of byte;
label
     vege;
```

```
procedure Byte_Kopiroz(var b1,b2,ki:byte);
var
  n:integer;
begin
  ki:=0;
  for n:=1 to 4  do
  if (a[n]=a[n] or b1)
     then ki:=ki or (c[n] and b2)
     else ki:=ki or (c[n] and b1);
end;

{-}

procedure Scr_Kopiroz(var s1rek,s2rek,srek:scr);
begin
  b:=0;
  for i:=1 to 16192 do begin
      if s1rek[i]=s2rek[i] then srek[i]:=s1rek[i]
      else begin
           if s1rek[i]=b then srek[i]:=s2rek[i];
           if s2rek[i]=b then srek[i]:=s1rek[i]
           else if s1rek[i]<>b then
           Byte_Kopiroz(s1rek[i],s2rek[i],srek[i]);
      end;
  end;
end;

{-}

begin                    { input-output ellenőrzés az 1. file-on: }
  assign(s1file,st1+'.scr');
  {$i-}reset(s1file);{$i+}
  if ioresult<>0    then begin
     writeln('Nincs 1. kepernyo-file!');
     delay(2000);
     goto vege;
  end;
  if filesize(s1file)<1 then begin
     writeln('Serult az 1. kepernyo-file!');
     delay(2000);
     goto vege;
  end;
```

```
  { input-output ellenőrzés a 2. file-on: }
  assign(s2file,st2+'.scr');
  {$i-}reset(s2file);{$i+}
  if ioresult<>0    then begin
     writeln('Nincs 2. kepernyo-file!');
     delay(2000);
     goto vege;
  end;
  if filesize(s2file)<1 then begin
     writeln('Serult a 2. kepernyo-file!');
     delay(2000);
     goto vege;
  end;

  { A 3. file definiálása: }
  assign(s3file,st3+'.scr');
  rewrite(s3file);
  size:=filesize(s1file);
  size2:=filesize(s2file);
  if size>size2 then size:=size2;

  { A tömbök feltöltése: }
  a[1]:=63;       c[1]:=192;
  a[2]:=207;      c[2]:=48;
  a[3]:=243;      c[3]:=12;
  a[4]:=252;      c[4]:=3;

  { Kopírozás: }
  for j:=0 to size-1 do begin
      gotoxy(1,1);        write(j+1);
      seek(s1file,j);     read(s1file,s1rek);
      seek(s2file,j);     read(s2file,s2rek);
      Scr_Kopiroz(s1rek,s2rek,srek);
      write(s3file,srek);
      read(kbd,c1);
  end;

 vege:
  close(s1file);
  close(s2file);
  close(s3file);
end;
```

```
begin
   GraphColorMode;
   Palette(1);
   Film_Kopiroz('film1','film2','film3');
end.
```

A kopírozó programmal készült a 13. ábra is. A bemutatott programok a számítógépes videotechnika alapját képező feladatok közül is talán a legfontosabbak. Ezeken kívül természetesen sokféle programot írhatunk, amely a filmekkel kapcsolatos. Változatos, mozgó grafikát készíthetünk ily módon. Az ilyen számítógépes, animációs filmek az IBM számítógépek kis felbontóképessége miatt még nem igazán jó minőségűek. Az egyre nagyobb felbontású grafikus kártyák kifejlesztése a számítógépeket fejlesztő ipar egyik állandó feladata. Sorra jelennek meg a jobbnál jobb grafikus kártyák, melyekkel egyre inkább a valóságot megközelítő grafikákat készíthetünk. A leírt módszerek ettől függetlenek. Az ilyen filmeknek sokoldalú alkalmazási területe van, pl. a reklámgrafika. Itt azonban fontos, hogy az ábrák mellett szöveg is megjelenhessen. Ezért célszerű a grafikus rendszereket kibővíteni olyan rutinokkal is, amelyek segítségével képesek vagyunk írni is ezekre a filmekre.

8. BEFEJEZÉS

A számítógépes grafikának - a sokoldalú fejlesztéseknek köszönhetően - ma már hatalmas elméleti háttere van. Egész könyvtárat töltene meg e tudományág eddig elért eredményeinek bemutatása, ezért e könyv keretein belül a csak a témánk szempontjából legjelentősebbnek vélt területekkel foglalkoztunk. A speciális esetekre vonatkozó, egyes részproblémákat feldolgozó, gyors algoritmusokkal szemben előnyben részesítettük a problémákat általánosabban leíró - de sokszor emiatt időigényesebb - algoritmusokat. Ezért a térlátás elmélete és számítógépes szimulációja átfogó képét szerettük volna kialakítani és egységes rendszerbe foglalni.

A számítógépes grafika napjainkban már igen szerteágazó tudomány. Az elmélet azonban e tudományágban - mint sok másban is - lényegesen meghaladja a gyakorlatot. A számítógépekre kifejlesztett ötletes algoritmusok a jövendő alkalmazások reményteljes ígéretét hordozzák magukban. Sajnos, ezen alkalmazások kiteljesedésére valószínűleg még sokat kell várnunk. A jövő útja mindenképpen e tudományág sokrétű alkalmazása. Eközben az elmélet is megalapozottabbá válik, valamint tökéletesednek a hardware eszközök is. Így pl. a közeljövőben az egyre jobb felbontású képernyők megjelenése, majd pedig - remélhetőleg - a számítógépes grafika és a holográfia összekapcsolása várható. Már ilyen irányú kísérletekről és biztató eredményekről is egyre többet hallunk.

A következő irodalomjegyzékben néhány olyan művet sorolunk fel, amelyek témáinkhoz kapcsolódnak. Ezen belül is három témakörre csoportosítottuk őket. Ezek: a szem felépítése; az e könyvben szereplő matematikai részek - főleg a mátrixszámítás, az analitikus geometria és a vektorszámítás - ismeretanyaga, valamint a számítógépes grafika területei. Mindezeket ajánljuk azoknak az Olvasóknak a figyelmébe, akiket valamely felsorolt terület különösen érdekel.

9. IRODALOMJEGYZÉK

9.1. *A szem felépítése, anatómiája*

1. Szentágothai János, - Réthelyi Miklós: Funkcionális anatómia. Az ember anatómiájának fejlődéstana és tájanatómiája. Medicina, 1985.

2. Kiss Ferenc - Szentágothai János: Az ember anatómiájának atlasza, 3. kötet: Idegtan. Értan. Érzékszervek. Medicina, 1982.

3. Handbook of Physiology, Section 1: Neurophysiology, volume 1 Washington, D. C. 1959
Glenn A. Fry: The image-forming mechanism of the eye.

4. Troland, L. T. : Principles of Psychophysiology. New York, 1930.

5. Dr. Weinstein Pál : Szemünk világa Medicina, 1958

9.2. *Mátrixszámítás, analitikus geometria, vektorszámítás*

1. I. N. Bronstein - K.A. Szemengyajev : Matematikai zsebkönyv Műszaki könyvkiadó, Budapest, 1987.

2. Ole Osterby - Zahari Zlatev : Direct methods for sparse matrices. Berlin, 1983.

3. G. S. Pandit - S. P. Gupta : Struktural analysis : A matrix approach. New-Delhi, 1983.

4. Victor Pan : How to multiply matrices faster Berlin, 1984.

5. Philip J. Davis : Circulant matrices 1979.

6. Jánossy Lajos - Tasnádi Péter : Vektorszámítás. 1. kötet: Vektor- és tenzoralgebra
Tankönykiadó, 1980.

7. Kendall E. Atkinson : An introduction to numerical analysis
New York, 1978.

8. Szendrei János : Algebra és számelmélet
Tankönyvkiadó, 1978.

9. Gáspár László : Mátrixaritmetikai gyakorlatok
Tankönyvkiadó, 1978.

10. Gene H. Golub - Charles F. Van Loan : Matrix computations
Oxford, 1986.

9.3. *Számítógépes grafika*

1. W. M. Newmann - R. F. Sproull : Interaktív számítógépes grafika
Műszaki könyvkiadó, 1985

2. Revitzky János : A számítógépes grafika felületkitöltő algoritmusai
SZTAKI, 1985.

3. Komáromi Imre : Számítógépes grafika

4. Sutherland, I. E. : Three-dimensional Data Input by Tablet
1974.

5. Sutherland, I.E. & G.W. Hodgman : Reentrant Polygon Clipping
1974.

6. Turbo Pascal, version 3.0, Borland International
1985.

LAPOZGATÓ SOROZAT

A Műszaki Könyvkiadó 1987 végén kezdte el a Lapozgató sorozat kiadását. A sorozatnak az a célja, hogy az IBM PC XT/AT-felhasználók olyan magyar nyelvű segédleteket kapjanak, amelyeket napi munkájuk során – egy-egy szoftver alkalmazásakor – szinte a számítógép tartozékaként, emlékeztetőként használhassanak. A kötetek az eredeti dokumentációra épülve megadják az utasítások, eljárások, függvények pontos szintaxisát, rövid leírását, és mindezt megjegyzésekkel, példákkal egészítik ki. Az utasítások funkció, ill. ábécé szerinti csoportosítása, valamint a könyvek spirálozott, zsebkönyv méretű kivitele a gyors információkeresést – és ami a fő, a gyors megtalálást – segíti.

A sorozat kötetei:

Herneczki Katalin: PC-DOS 3.20 (megj. 1987. december, ára: 120 Ft).
A legismertebb, legjobban elterjedt operációs rendszerrel foglalkozik.

•

Herneczki István: MACRO Assembler (terv. megj. 1988. február, ára: 150 Ft).
Az Intel 8088-as processzor regisztereit, címzési módjait, utasításkészletét, pszeudoutasításait írja le.

•

Szolek András: dBASE III PLUS (terv. megj. 1988. március, ára: 150 Ft).
Az Ashton-Tate által forgalmazott dBASE III PLUS relációs adatbáziskezelő rendszert foglalja össze.

•

Donát János: WordStar (terv. megj. 1988. április, ára: 120 Ft).
Az IBM PC számos szövegszerkesztője közül az egyik legelterjedtebb a WordStar, amely kedvelt segédeszköz levelek, cikkek, kézikönyvek készítése során.

•

Szolek András: Turbo Pascal (előkészületben).
Az egyik legnépszerűbb és legjobban használható programozási nyelvet, az amerikai Borland szoftverház termékét tárgyalja.

•

dr. Barakonyi Károly: FRAMEWORK II (előkészületben)
A FRAMEWORK II integrált programcsomag szövegszerkesztésre, táblázat- és adatbázis-kezelésre, grafikus feladatok megoldására, telekommunikációra egyaránt használható. Az összefüggő feladatokat funkcionálisan és vizuálisan is összekapcsolja.

Ára: 149,– Ft