Oktatási jegyzet

a

Számítógéppel segített tervezés

c. tárgyhoz, 3. éves építészmérnök hallgatók részére.

Összeállította: Csabay Bálint építészmérnök

T a r t a l o m j e g y z é k

1. Általában a CAAD rendszerekről

A számítógéppel segített építészeti tervezőrendszerek (CAAD) egyre szélesebb körben terjednek el, a hagyományos tervezési módszerekkel szemben nyújtott előnyeik miatt. A tárgy keretében a CAAD rendszerek alapismereteivel foglalkozunk, és a CAAD rendszzerekben használt ábrázolási, megjelenítési eljárásokkal, problémákkal.

A CAAD rendszerek legfontosabb előnyei

a hagyományos tervfeldolgozással szemben:

- a tervezés során nélkülözhetetlen módosítások egyszerűbbek
- pontosabb feldolgozást tesznek lehetővé (és követelnek is meg!)
- rengeteg többletinformációt hordoznak
- a térbeli megjelenítés, a modell virtuális megtekintése csökkenti a megvalósításkor felmerülő hibalehetőségeket
- a csoportmunka során adódó tervrészek különválasztását és összehangolását nagyobb megbízhatósággal lehet megvalósítani
- ismételt tervfelhasználás esetén (típustervek) gyorsabb az adaptálás
- egyszerűbb és gyorsabb a dokumentáció szállítása (számítógép hálózatok)
- a tervek archíválása jóval kisebb tárolási helyet igényel.

A CAAD rendszerek néhány hátránya

a hagyományos tervfeldolgozással szemben:

- nagyobb költségű eszközöket igényelnek
- speciálisabb és magasabb szintű ismereteket igényel már a tervfeldolgozás is
- általában lényegesen megnő a papírfelhasználás (a sikertelen vagy közbenső kísérletek, újabb módosítások kinyomtatása)

Az azonban vitathatatlan, hogy már ma sem, de az elkövetkezendő időben bizonyosan nem nélkülözheti egy építész tervezőiroda a számítógépet, és az építészeti tervező programokat.

A CAAD rendszerek használata során meghonosodtak olyan alapfogalmak, amelyek a legtöbb rendszerben valamilyen formában megtalálhatók. Ezek ismerete elengedhetetlen hatékony használatukhoz, ezért a legfontosabbakat az alábbiakban ismertetjük.

A CAAD rendszerek operációs rendszereinél használt alapfogalmak:

Felhasználói felület (*user interface*) a felhasználó és a program kapcsolatát biztosító programkörnyezet, az adatbevitelt- és a megjelenítést lehetővé tevő programrész.

Felhasználóbarát környezet (*user friendly environment*) a felhesználói felület minőségének fokmérője, minél kényelmesebben és magától érthetődőbben kezelhető a program, annál "felhasználóbarátabb".

Parancsorientált rendszer a programmal való párbeszéd (*interakció*) eszköze, a programutasítások végrehajtása adott parancsnevek beírása után történik.

Eseményorientált rendszer a programmal való párbeszéd (*interakció*) eszköze, a programutasítások végrehajtása a felhasználó számára kényelmesebb grafikus eszközökkel vezérelt formában történik, a bekövetkező és az elvárt események kijelzése segíti a felhasználót teendőiben. (pl. kurzor alakja, legördülő /*pull-down*/-tetszőleges helyen megjelenő /*pop-up*/ menük, ikonok, nyomógombok /*button*/ , gördítősávok /*scroll-bar*/, ...)

A CAAD rendszereknél használt alapfogalmak:

2 dimenziós elemek egyenesvonal, kör, körív, ellipszis, szabadon formált görbe, szöveg, sraffozási minta, ...

3 dimenziós elemek síkidomok (poligonok), hasábok, henger...

2,5 dimenzió	2 dimenziós elemek, amelyek rendelkeznek 3-ik dimenzió irányú konstans kiterjedéssel.
"intelligens elemek"	olyan 3 dimenziós elemek, amelyek a geometriai alakjukon kívül még számos más tulajdonsággal rendelkeznek (fizikai tulajdonságok-anyagjellemzők, egymáshoz való kapcsolódások, stb. pl. fal, pillér, nyílászáró, födém, tető, lépcső, ...)
koordináta-rendszerek	abszulút-relatív, derékszögű-polár koordináták – célszerűen használhatók mind kijelzésnél, mind adatbevitelnél.
rasztervonzás	(*snap-grid*) a rajzolni kívánt koordinátapontok vonzása adott raszterháló metszéspontjaiba
fóliatechnika	(*layer*) az elemek csoportosításának eszköze; többnyire automatikusan megadott fóliákhoz rendeli a létrehozott elemeket, de a felhasználó szabadon átteheti azokat más fóliára; fóliákat lehet ki-bekapcsolni (miáltal az azokhoz rendelt elemek eltüntethetők, vagy újra láthatóvá tehetők), átnevezni, fóliacsoportokat létrehozni (és ezzel azokat együtt kezelni), illetve írásvédetté tenni (így azon új elemek létrehozását, az azon lévő meglévő elemek módosítását, törlését megakadályozni)
szintek kezelése	(*story*) az elemek csoportosításának eszköze; egy-egy szint külön is - a más szintekkel együtt is megjeleníthető, az ábrázolni kívánt nézettől, vagy a felhasználó szándékától függően. A szintek elnevezhetőek, magassági értékek rendelhetők hozzájuk, elemek másolhatók egyik szintről a másikra általában a szinthez tartozó magassági értékek értelmes átvételével.

2. Szabadon formált görbék alkalmazása CAAD rendszerekben.

A számítógépppel segített műszaki tervezés során már 1970 táján felmerült az igény, hogy iparművészek által elgondolt görbéket és görbe felületeket számítógéppel lehessen kezelni. Ezek az alakzatok ugyan tetszőleges pontossággal közelíthetők egyenesvonalakkal, illetve síklapú poliéderekkel, de az ilyen ábrázolások terjedelmesek és nehézkesek. Alakleírásra általában véve két módszert alkalmaznak: vagy a görbék, tárgyak mért adataiból indulnak ki (*analitikus alkalmazás*), vagy a tervezési folyamatot támogató és az alakmódosítások egyszerű végrehajthatóságát szem előtt tartó módszereket (*szintetikus alkalmazás*). Ez utóbbiak alkalmasak arra, hogy interaktív módon találja meg a tervező a számára leginkább kívánatos alakot, vagyis egy kiinduló alak módosítgatásaival jusson el a végeredményig. A görbetervezésnél használt módszerek kiterjesztése a felületek ábrázolásánál is használhatóak, ezért részletesebben először csak a a görbe-leíró módszerekkel foglalkozunk.

A görbéket leíró matematikai módszerek kialakításánál sok lényeges szempontot kell figyelembe venni. A tervezők szempontjából legfontosabb ilyen szempontok közül kiemelünk néhányat:

- Fontos szempont az interaktív tervezés maitt a viszonylag egyszerű módosíthatóság, amit többnyire ún. *támpontok* vagy *tartópontok* mozgatásával lehet elérni.

- Nem elhanyagolható az sem, hogy a tartópontok mozgatása csak *lokálisan* hat-e a teljes görbe alakra, vagy *globálisan* a teljes görbére.

- További fontos szempont *a görbék folytonossága*. Általában nem célszerű egyetlen görbével leírni egy adott alakot, sokszor jóval egyszerűbb több görbe illesztésével azt elérni. Az ilymódon összetett görbék csatlakoztatása lehet nulladrendűen folytonos, ez esetben csak illeszkedik egymáshoz két görbe, de megtörik a görbe vonala. Elsőrendűen folytonos akkor lesz, ha az illesztési pontban közös érintője is van a két csatlakozó görbének. Másodrendű folytonosságnál a görbületeknek is azonosaknak kell lenniük.

A szplájnok (*spline*) esetében a tartópontok elmozdítása lokális változást okoz csak a görbében, amelynek előnye az, hogy a görbének csak a módosítani kívánt részéhez közeli egy vagy néhány tartópontjának a helyzetét kell változtatni.

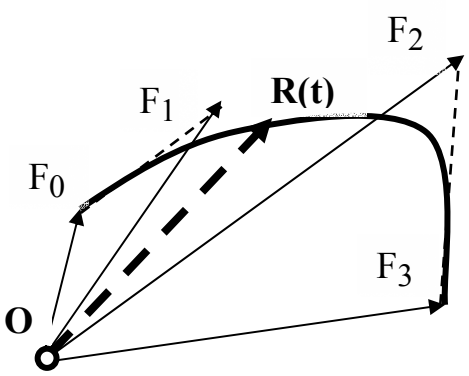
A szplájnokat, mint koncentrált erőkkel terhelt rugalmas rudak megörbült alakját képzelhetjük el. A régi angol hajóépítő és kőfaragó mesterek ténylegesen nehezékek vagy szegek közé feszített rugalmas vesszőket használtak görbék kijelölésére.

A szplájnoknál egyenletesebb görbületet – ezért kellemesebb vonalvezetésű, esztétikusabb görbéket – eredményező görbe-leíró módszert - első sikeres kidolgozója és alkalmazója, P. Bèzier francia autókarosszéria-

tervező mérnök után *Bèzier-görbéknek* nevezik. A Bèzier-görbék egy tartópontjának megváltoztatása ugyan globálisan hat a teljes görbére, azonban több – kevés tartóponttal meghatározott és folytonosan illesztett Bèzier-görbével ez a hátrány csökkenthető.

Matematikai értelemben mind a szplájnokat, mind a Bèzier-görbéket paraméteres függvényekkel írják le, vagyis a görbe pontjainak egy paraméterrel megadott vektor felel meg: **R(t) = [x(t), y(t), z(t)]**

A *t* paraméter egy megadott tartományban (rendszerint 0 és 1 között) változik, miközben az *x, y, z* paraméteres függvények megadják a görbe alakját.



Bèzier-görbék esetén, amennyiben *4* tartópontot (vektort) adunk meg (F_0, F_1, F_2 és F_3 –at), akkor képezhetjük az

$$R(t) = F_0\,(1-t)^3 + F_1\,3t(1-t)^2 + F_2\,3t^2\,(1-t) + F_3\,t^3$$

„súlyozott átlag" vektort, ahol is a súlyozó tényezők rendre $(1-t)^3$, $3t(1-t)^2$, $3t^2(1-t)$ és t^3. Ha a *t* paraméter értéke végigfut 0-tól 1-ig, az *R(t)* vektor végpontja egy görbeszakaszt ír le (az ábrán vastag vonallal jelölve). Láthatjuk a képletből, hogy a *t=0* esetben a Bèzier görbe átmegy az F_0, a *t=1* esetben pedig az F_3 végpontján. Az F_1 és F_2 vektorok végpontjain a Bèzier-görbe nem megy át, de e végpontok mintegy maguk felé „vonzzák" az F_0 és F_3 vektorok végpontjait összekötő egyenesnek a súlyozó tényezők képletei által különböző mértékben befolyásuk alá rendelt szakaszait. További fontos tulajdonsága a görbének, hogy a végpontokon és a velük szomszédos tartópontokon áthaladó egyenesek érintik a görbét a végpontokban. Vagyis, mint ahogy az az ábrán szaggatott vonallal van jelölve, az F_0 pontban érintő az F_0-F_1 pontokon átmenő egyenes, míg az F_3 pontban az F_2-F_3 pontokon átmenő egyenes. Ez a görbe a *t*–ben harmadfokú Bèzier-görbe.
A fenti esetet általánosítva *4* helyett *n+1* tartóponttal, a görbét leíró vektor:

$$R(t) = \sum_{i=0}^{n} F_i \cdot B_{i,n}(t),$$

amely *n=3* esetben valóban a fenti vektoregyenletet eredményezi, ha F_0, F_1, ... F_i tartópontok előre adottak és $B_{i,n}(t)$ a súlyfüggvény (blending function), melynek definíciója:

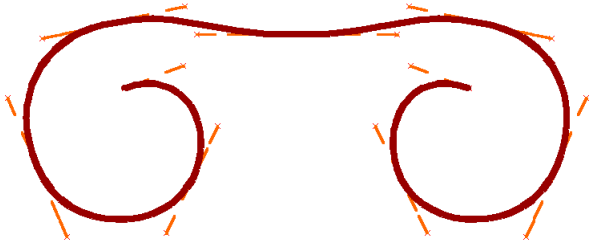
$$B_{i,n}(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i}, \text{ ahol a binomiális együttható} \binom{n}{i} = \frac{n!}{i!\cdot(n-i)!}$$

Az ismertetett vektoregyenlet természetesen felírható az *R(t)* vektor *x(t), y(t)* és *z(t)* paraméteres függvényeire külön-külön is az alábbi skaláregyenletekkel:

$$x(t) = \sum_{i=0}^{n} x_i \cdot B_{i,n}(t), \qquad y(t) = \sum_{i=0}^{n} y_i \cdot B_{i,n}(t), \qquad z(t) = \sum_{i=0}^{n} z_i \cdot B_{i,n}(t),$$

ahol x_i, y_i, z_i a tartópontok (F_i) megfelelő koordinátái.

Látható, hogy a tartópontok számának növelésével egyre bonyolultabb – több tagból álló – lesz a görbét leíró egyenlet, és a görbe fokszáma is növekszik: *n+1* tartópont esetén a görbét leíró polinom fokszáma *n* lesz. Ezért bonyolultabb alakok leírását általában több alacsonyabb fokú Bèzier-görbe egybekapcsolásával szokás megoldani a fentebb említett érintési tulajdonság kihasználásával. Megjegyezzük, hogy a görbe parametrikus volta miatt az érintőleges folytonosság még nem jelenti a *t* paraméter szerinti folytonosságot is, ahhoz hogy az is teljesüljön (vagyis a *t* szerinti derivált is folytonos függvényt adjon) a végpontokba befutó érintőszakaszok aránya 1-et kell adjon, azaz egyforma hosszúaknak kell lenniük. Erre a tulajdonságra a Bèzier-felületek tárgyalásánál még kitérünk.



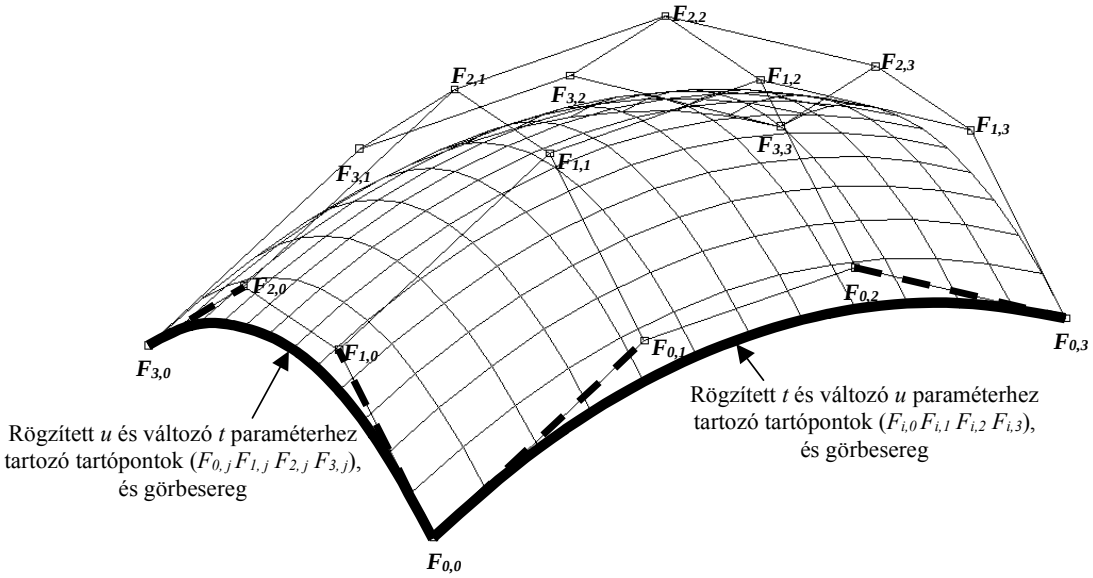
Ion voluta íve több – érintőlegesen csatlakozó - Bèzier-görbével megadva

3. Szabadon formált felületek

Mint már a szabadon formált görbéknél említettük, az ott alkalmazott módszerek kiterjeszthetők felületek esetére is. Így két Bèzier-görbe Descartes-szorzatának előállításával kaphatunk egy Bèzier-felületet. Ez matematikai-értelemben az alábbi vektoregyenletet eredményezi:

$$R(t,u) = \sum_{i=0}^{n} \sum_{j=0}^{m} F_{i,j} \cdot B_{i,n}(t) \cdot B_{j,m}(u),$$

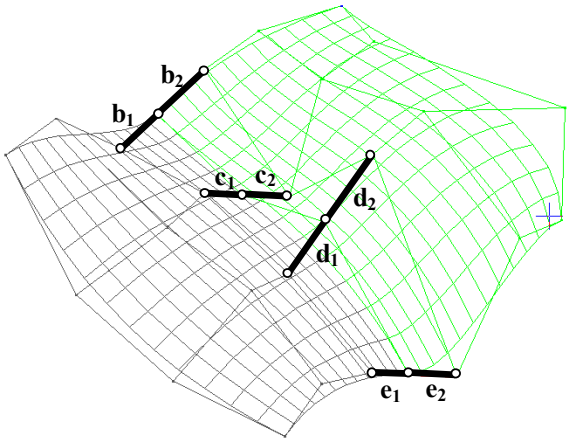
ahol *t* és *u* két paraméter, melyeknek értéke *0* és *1* között változik, $F_{i,j}$ a kétirányú görbesereghez tartozó tartópontok, és $B_{i,n}(t)$ valamint $B_{j,m}(u)$ a fent leírtak szerinti súlyfüggvények külön a *t* és külön az *u* paraméterhez.
A geometriai értelmezéshez az alábbi ábra nyújt segítséget:



Bèzier-felület (n+1)x(m+1) hálóba rendezett tartóponttal

Mint az ábrán látható a két irányban 4-4 tartóponttal megadott görbék által kifeszített felületnek 4x4 azaz 16 tartópontja van. Felületeknél fokozottan érvényes, hogy nagyon bonyolulttá válhat a kezelésük a tartópontok növelésével, ezért több felület illesztésével célszerű bonyolultabb felületeket leírni.

A görbéknél leírt folytonossági feltétel azonban felületek esetén csak kiegészítéssel érvényes. Két Bèzier-felület nulladrendűen folytonos, ha a szomszédos felületek tartópontjai egybeesnek, hiszen a fentiekből következik, hogy ekkor az adott tartópontok által meghatározott Bèzier-görbék azonosak, tehát valóban illeszkedik a két felület, de természetesen törésvonal alakul ki az illesztés mentén.

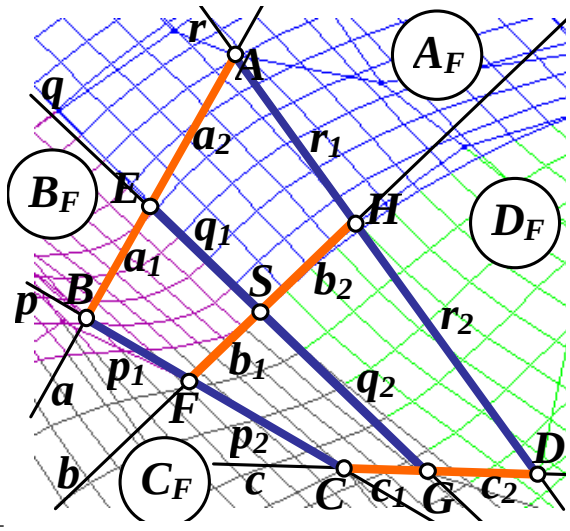


Az elsőrendűen folytonos (amely alatt a *t* és *u* paraméterek szerinti folytonosságot is értjük) csatlakozáshoz nem elegendő, ha két csatlakozó felületnél a határoló görbe tartópontjait a velük szomszédos tartópontsorral összekötő élek (a továbbiakban tartóélek) egy egyenesbe esnek. Ez ugyan szükséges feltétel, de a megkívánt elsőrendű folytonossághoz még az is szükséges, hogy ezen élpárok arányai állandók legyenek.
Tehát a $\frac{b_1}{b_2} = \frac{c_1}{c_2} = \frac{d_1}{d_2} = \frac{e_1}{e_2}$ is szükséges a fenti módon megkövetelt elsőrendű folytonossághoz.

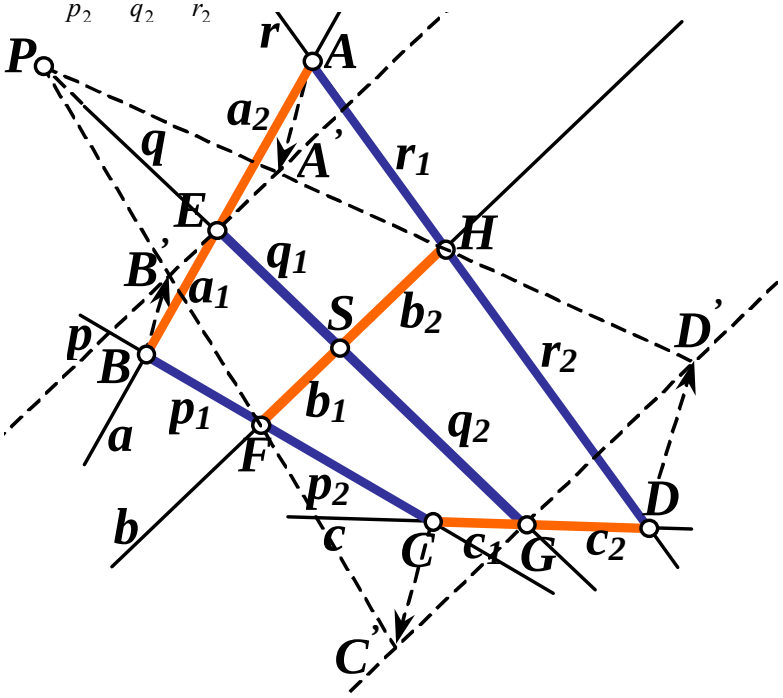
Ellentétben a görbékkel, felületek esetén nem csak egy irányú csatlakozásról kell beszélni, hiszen a felület mindkét oldalirányához csatlakozhatnak további felületelemek. Ez esetben, ha törésvonal nélkül, vagyis érintőlegesen szeretnénk folytatni a felületet, akkor

mindkét irányban teljesülnie kell a fenti elsőrendű folytonossági feltételnek.Ez azonban további megkötéseket jelent négy felületelem sarokcsatlakozásánál.

Az alábbiakban megmutatjuk, hogyan csatlakoztatható egy sarokpontban négy Bèzier-felület kielégítve az elsőrendű folytonosság feltételét. Legyen a mellékelt ábra jelölése szerint A_F, B_F, C_F *és* D_F a négy felület, amelyek az *S* sarokpontban találkoznak és saroktartópontjai egyik irányban az *a*, a *b* és a *c*, másik irányban pedig a *p*, a *q*, és az *r* egyenesekre illeszkednek. Rendre az *a* és *p* egyenesek metszéspontjában a B_F felület *B* tartópontja, az *a* és *q* egyenesek metszéspontjában az A_F és B_F felületek közös *E* tartópontja, az *a* és *r* egyenesek metszéspontjában az A_F felület *A* tartópontja, a *b* és *p* egyenesek metszéspontjában a B_F és C_F felületek közös *F* tartópontja, a *b* és *q* egyenesek metszéspontjában a négy felület közös *S* tartópontja, a *b* és *r* egyenesek metszéspontjában az A_F és D_F felületek közös *H* tartópontja, a *c* és *p* egyenesek metszéspontjában a C_F felület *C* tartópontja, a *c* és *q* egyenesek metszéspontjában a C_F és D_F felületek közös *G* tartópontja, végül a *c* és *r* egyenesek metszéspontjában a D_F felület *D* tartópontja van, továbbá az *a* egyenest osztó a_1, a_2, a *b* egyenest osztó b_1, b_2, és a *c* egyenest osztó c_1, c_2, illetve a *p* egyenest osztó p_1, p_2, a *q* egyenest osztó q_1, q_2, és az *r* egyenest osztó r_1, r_2 szakaszokra teljesül a korábban leírt arány, vagyis: $\frac{a_1}{a_2}=\frac{b_1}{b_2}=\frac{c_1}{c_2}$ valamint $\frac{p_1}{p_2}=\frac{q_1}{q_2}=\frac{r_1}{r_2}$.



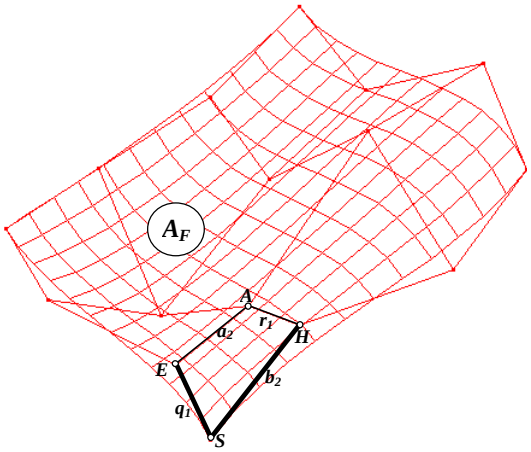
A következőkben be fogjuk látni, hogy a *b* és *q* egyenesek által kifeszített síkhoz mindig választható olyan párhuzamos vetítési irány, amely mellett az *A, B, C* és *D* tartópontoknak erre a síkra vetített *A', B', C'* és *D'* képei olyan trapézt alkotnak, melynek nem párhuzamos oldalaira illesztett egyenesek és az S ponton átmenő egyik tartóegyenes (esetünkben *q*) közös *P* pontban metsződnek, míg a másik *S* ponton átmenő tartóegyenes (esetünkben *b*) pedig a trapéz két párhuzamos oldalával lesz azonos irányú (természetesen speciális esetben a *P* metszéspont lehet a végtelenben is, ekkor trapéz helyett paralelogramma lesz a vetület).



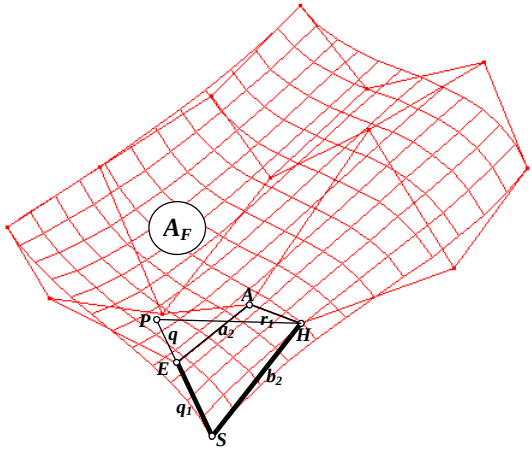
A *q* egyenesen vegyünk fel tetszőlegesen egy *P* pontot, amelyet kössünk össze *F* és *H* pontokkal. Az *E* és *G* pontokon húzzunk párhuzamos egyenest *b*-vel. Az így kialakuló *A'B'C'D'* vetületi trapézban a párhuzamos szelők tételéből következően a *B'E* szakasz hossza (a_1 vetülete, vagyis a_1') úgy aránylik *EA'* szakasz hosszához (a_2 vetülete, vagyis a_2'), mint b_1 aránylik b_2-höz. De mivel $a_1/a_2 = b_1/b_2$, ezért $a_1'/a_2' = a_1/a_2$ is igaz. Ebből pedig következik *BB'E* és *AA'E* háromszögek hasonlósága, vagyis *AA'* és *BB'* párhuzamossága. Ugyanígy belátható az AA' és DD', a BB' és CC', valamint a CC' és DD' párhuzamossága is, vagyis a vetítési irányok párhuzamossága.

A fenti gondolatmenetnek a Bèzier-felületek csatlakozásához szükséges tartópont helyzetek kialakításánál van jelentősége, mivel a tétel fordítva is igaz: tetszőleges trapézból (vagy speciálisan paralelogrammából) a sarokpontok tetszőleges irányú párhuzamos vetítésével – a fenti arányokat betartva – előállíthatók a Bèzier-felületek tartópontjai, amelyekkel biztosítható az összetett felület elsőrendű folytonossága mindkét irány-

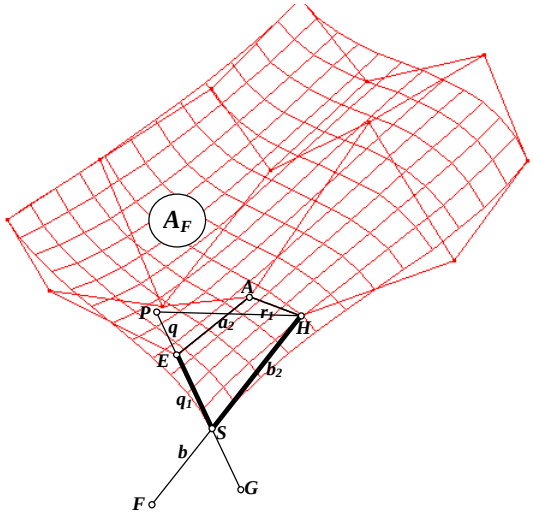
ban. Az alábbi ábrasor bemutatja, hogy egy adott A_F kiinduló Bèzier-felülethez milyen kötöttségek mellett csatlakoztatható további három felület az elsőrendű folytonosság megőrzése mellett:



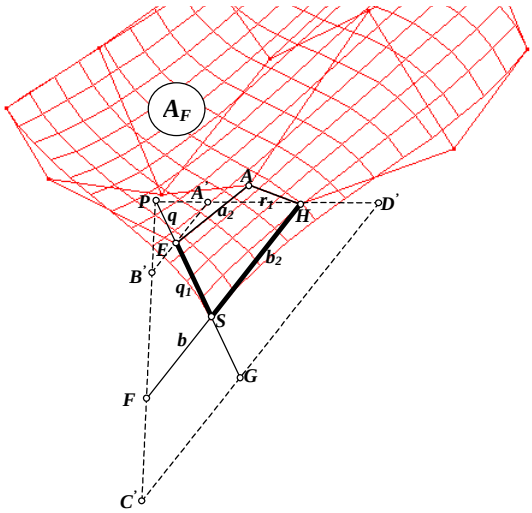
Adott kiindulási A_F Bézier-felület a sarok környéki *S, E, A* és *H* tartópontokkal illetve a_2, b_2, q_1, r_1 tartóélekkel.



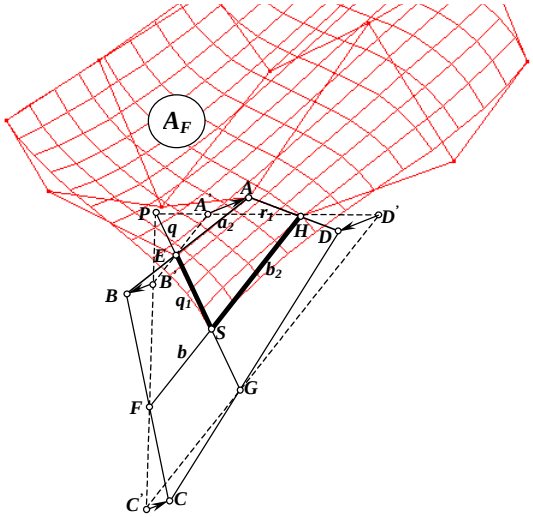
A q_1 tartóélre illeszkedő *q* egyenesen *P* pont felvétele és összekötése *H* tartóponttal.



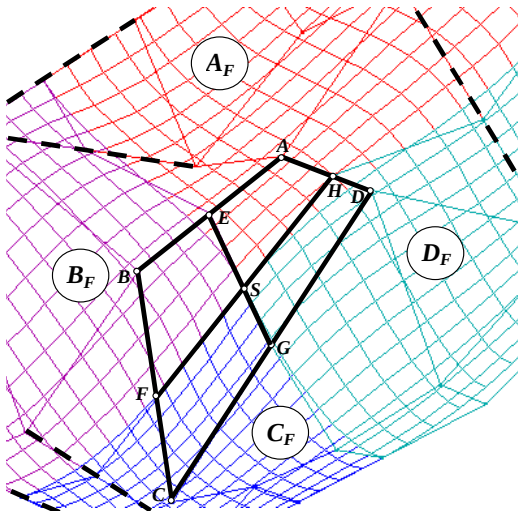
A b_2 tartóélre illeszkedő *b* egyenesen, és a *q* egyenesen a szomszédos felületek *F* és *G* tartópontjainak felvétele.



A *b* egyenessel párhuzamos oldalú és a *P* pontba befutó élű *A'B'C'D'* sarokpontú vetületi trapéz kiszerkesztése.



Az *A'A* vetítési iránnyal párhuzamosan *B', C'* és *D'* rávetítése a megfelelő tartóélekre (*B, C* és *D* tartópontok).



A kiszerkesztett tartópontokkal meghatározott, folytonosan illeszkedő Bèzier-felületek.

Természetesen a csatlakozó felületek többi - az utolsó ábrán szaggatott vonallal jelölt - szomszédos tartóélénél is biztosítani kell az adott iránynak megfelelő arányt.

4. Adatbázisok és alkalmazásuk CAAD rendszerekben

A CAAD rendszerekben használt elemektől elvárjuk, hogy olyan információkat is őrizzenek, amalyek nem csak azok megjelenítéséhez szükségesek, hanem a terv analizálásához, kiértékeléséhez is segítséget nyújtsanak. Pl. amikor megrajzolunk egy falat, akkor egyrészt ismertnek kell lennie a geometriai adatoknak (a fal kontúrjának vetületi koordinátái, magassága, valamely szinthez viszonyított helyzete, stb.), hogy transzformációkat végezhessünk vele, műszaki tervrajzot készíthessünk róla, másrészt a fal egyéb fizikai tulajdonságai (anyaga, határoló felületeinek anyaga, színe, mintázata, térfogatsúlya, stb.), hogy meg tudjuk nézni térbeli hatását adott megvilágítás mellett, vagy hogy meg tudjuk állapítani a súlyát, a felhasznált anyagmennyiséget.

Mindezeket az információkat adatbázisokban tárolják. Az adatbázis valamely azonosan kezelt elemek sorozatát jelenti, ahol az elemek különböző jellegű adatokkal vannak leírva. Egy elemet az azt leíró adatokkal rekordnak neveznek, a rekordok adatait mezőknek. Pl. a fal rekordja lehet az azonosítójele, a geometriai adatai (alapsík, kontúrpontok az alapsíkon, magasság), az anyagjellemzői (anyagnév, térfogatsúly, hőellenállás, hangellenállás, ...), határoló felületeinek jellemzői (szín, textúra, fényelnyelés, tükröződés, fénytörés, ...). Amennyiben így határozunk meg egy falat, akkor minden falat ezekkel az adatokkal kell leírni, vagyis e rekordok összesége a falak adatbázisa. Ugyanílyen módon adottak további adatbázisok az azonos tulajdonságú elemekhez. (pl. anyagok, fóliák, szintek, stb. adatbázisa). Az adatbázisok egyes mezői között kapcsolatot lehet teremteni, ezáltal egyrészt elkerülhető az adatok többszöri tárolása, másrészt dinamikussá tehető az elemek közötti kapcsolat. Az íly módon kapcsolt adatbázisokat relációs adatbázisoknak nevezik (így a falak és a szintek azonosító jele között kapcsolatot teremtve összerendelhető pl. a 001 azonosító jelű fal a 03. azonosító jelű szinttel, vagyis az adott fal a 03. szintre kerül; ha kapcsolat van a szintek alsó síkja és a falak alsó síkja között, akkor egy adott fal adott szinttel való összerendelése a fal alsó sík változását is jelenti).

Az egyes rekordokat meghatározó mezők adatainak leírását adatszerkezetnek nevezik. Egy rekord adatszerkezetének megtervezése meghatározza, hogy mennyire lesz később alkalmas az adott elem leírására, anélkül hogy az egész adatbázis szerkezetét módosítani kelljen. A mezők adatai lehetnek számok, szövegek, hivatkozási címek más adatbázis mezőjére (vagy file-ra), vagy összetettebb adatok (pl. számpárok, számhármasok – koordináták, bonyolultabb struktúrák).

Az adatbázisokkal szemben fontos követelmény, hogy egységesen és ellenőrzött módon lehessen új adatokat felvinni, a már meglévő adatokat módosítani, vagy lekérni.

azonosító (sorszám)	név (szöveg)	viszonyítási sík (szám-négyes)	ref. vonal (lista-cím)	vastagság (szám)	a ref. vonalhoz viszonyított helyzet (karakter {J|B|K})
001	***38-as kism. téglafal***	***(A1,B1,C1,D1)***	***X1. vonallánc***	***0.38***	***J***
002	***vasbeton fal***	***(A2,B2,C2,D2)***	***X2. vonallánc***	***0.40***	***B***
. . .	. . .	. . .	. . .	. . .	. . .

alsósík magasság (szám)	felsősík magasság (szám)	térfogatsúly (szám)	jobb oldali felület anyaga (lista-cím)	bal oldali felület anyaga (lista-cím)
0.00	***2.70***	***1800.00***	***fehér meszelés***	***tégla1 textúra***
-0.80	***0.00***	***2400.00***	***nyersbeton textúra***	***nyersbeton textúra***
. . .	. . .	. . .	. . .	. . .

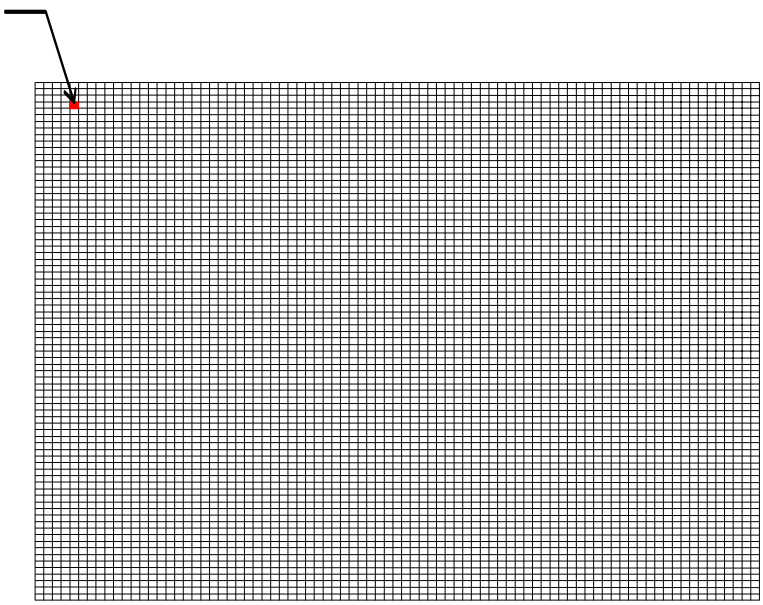
él felületek anyaga (lista-cím)	hővezetési tényező (szám)	jobb oldali felület fénytörése (szám)	...	él felületek fénytörése (szám)
fehér meszelés	***0.78***	***1***	...	***1***
nyersbeton textúra	***1.55***	***1***	...	***1***
. . .	. . .	. . .	...	. . .

Példa fal adatbázisra CAAD rendszerben. Ha mondjuk a viszonyítási sík oszlopot összerendeljük a szint adatbázis azonosító oszlopával, akkor a viszonyítási sík paraméterei helyére a megfelelő szint-rekord azonosítója kerülhet. Természetesen a szint adatbázis megfelelő rekordjában meg kell legyen adva a sík egyenlete, így ebben az adatbázisban nem szükséges, sőt nem is célszerű megadni a redundancia elkerülése miatt. Ettől fogva elegendő csak másik szint azonosítójára kicserélni az aktuálisat ahhoz, hogy a fal átkerüljön a kívánt szintre a magassági adataival együtt.

5. Képfeldolgozás, rendering-eljárás

Alapfogalmak.

Egy számítógéppel készített, ill. azzal megjelenített kép nem más, mint **pixel**[1]-sorok és oszlopok rendszere.



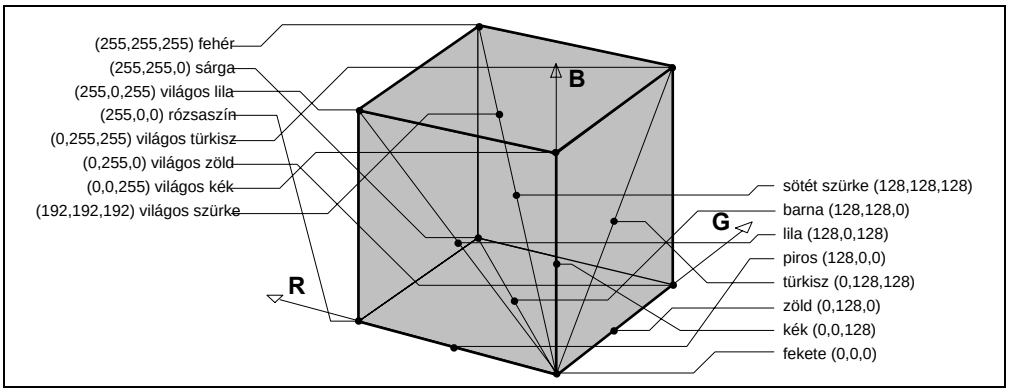
Minden pixel tartalmaz egy értéket, ami egy színnek felel meg.

A számítógépbe beépített grafikus kártyától valamint a monitortól függően különböző számú szín jeleníthető meg pixelenként (pl. 16, 256, ... , ≈16,8 millió – ami az átlagos emberi szem színérzékelésének felel meg, ezért *true colour*-nak, valódi / tényleges színnek is neveznek).

A színértékek tárolására szolgáló memóriaszükséglet pixelenként: 16 szín esetén 1/2 byte (4 bit), 256 szín esetén 1 byte (8 bit), míg ≈16,8 millió szín esetén 3 byte (24 bit). És általában a memória igény $n/8$ byte (n bit), ha 2^n különböző színt akarunk megjeleníteni.

A színeket a számítógép színösszetevőkből számítja ki. Ezek a számítógép alapszínei: **R** (red - piros), **G** (green - zöld) és **B** (blue - kék), amelyeket - a színfelbontástól függően - szintén számértékek jellemeznek. Az egyik legelterjedtebb színrendszer az RGB rendszer, amelynek neve az alapszínek kezdőbetűiből származik (a számítógépi feldolgozásoknál elterjedt színrendszerek általában átszámíthatók egymásra, tehát az egyik rendszerben meghatározott szín kölcsönösen megfeleltethető a másik rendszer ugyanazon színének).

Az RGB rendszer színábrázolását az alábbi ábra szemlélteti:



RGB rendszert ábrázoló színtest

Ez a színtest ≈16,8 millió szín számértékkel való jellemzésére alkalmas, mivel a három alapszín mindegyike 256 különböző értéket vehet fel (0-tól 255-ig), vagyis mindhárom színértéket egy-egy byte-on (8-8 biten) tárolhatjuk. A tárolási méret, adatmozgatási sebesség kevesebb szín alkalmazásánál kedvezőbb, szükséges volt kidolgozni az ennél kevesebb szín megjelenítését biztosító tárolási formákat is.

[1] pixel: picture element - képpont.

A 256 szín ábrázolását oly módon oldották meg, hogy a 16,8 millió színből valamely 256 színt egy ún. palettán leírják a fenti RGB értékekkel, és az egyes pixelek értéke nem maguk a színek, hanem e paletta sorszáma. A megjelenített szín pedig a palettán ezen a sorszámon található színérték. Így egyidejűleg csak 256 szín jeleníthető meg, bár a ≈16,8 millió bármelyik 256 színe. Sok esetben 256 szín használata is megfelelő minőséget eredményez.

Bizonyos esetekben elgendő 16 szín is képek megjelenítéséhez. Itt a színértékek leírására 1-1 bit szolgál (a három alapszínre 3 bit), továbbá egy ún. fényerősség bit szabályozza, hogy az adott szín világosabb vagy sötétebb árnyalata jelenjen-e meg. Ezen a módon az 1. ábrán bemutatott színtestben elhelyezett 16 szín ábrázolható.

A rendering-eljárás nem más, mint egy olyan módszer, amely egy 3 dimenziós modell vetületi képén kiszámítja az egyes pixelek színértékét majd az így előállított képet megjeleníti. A kapott eredmény egy olyan raszterképet eredményez, amelyik a múlt század végi pointillista festők képeire emlékeztet, csak a számítógép raszterhálója akár jóval sűrűbb, vagyis egy képpont mérete sokkal kisebb is lehet, mint mondjuk Signac, Pissarro, vagy Seurat festményein.




A kiszámított színértékek függenek a modell egyes felületeinek az alapszínétől, a fényforrás alapszínétől, a felület jellemzőitől (pl. szórt fényre való reagálás, tükröződés), tükröződő felület esetén a környezet színeitől, átlátszó vagy áttetsző felületek esetén a mögötte lévő felületek színeitől, stb.

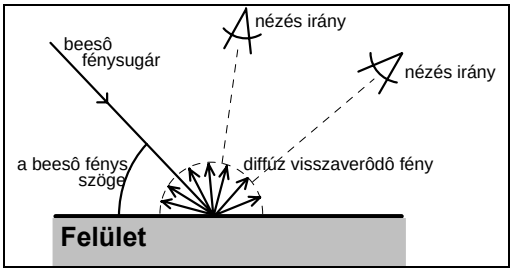
Ha ezeket a jellemzőket a felület valóságos fizikai jellemzőiből számítaná a rendering-eljárás, az nagyon hosszadalmas és bonyolult számítást eredményezne, nem beszélve a kezelhetetlenül sok paraméterről. Ezért (tapasztalati úton) olyan anyagjellemzőket dolgoztak ki, amelyekkel a valóságot elég jól lehet modellezni, ugyanakkor a számítások jelentősen leegyszerűsíthetők.

A továbbiakban a rendering-eljárások anyagjellemzőiről lesz szó. Különböző CAAD rendszereknél eltérő meghatározások, jelölések lehetnek, de az alapelvek a legtöbb rendering-eljárásnál hasonló.

- **Alapszín**: A felületeknek, ill. fényforrásoknak a fentiekben leírt RGB értékekkel megadott színe.
- **Környezeti fényvisszaverődési együttható**, Ka: Ez az együttható a 0..1 intervallumban írja le annak a fokát, hogy a környezeti szórt fényre a vizsgált felület hogyan reagál. Ez az úgynevezett ambiens, vagy

környezeti szórt fény olyan fény, ami nem egy meghatározott pontból vagy adott irányból érkezik, hanem a tárgyat egyenletesen éri. 0 érték mellett ez a szórt fény semmi szerepet nem játszik, 1-es értéknél pedig teljes egészében figyelembe vesszük. Az ambiens fény erőssége természetesen nem a felület tulajdonsága, hanem az egész modellre nézve kell globálisan megadni.

- **Szórt fényvisszaverődési együttható, K_d:** A diffúz (szórt) fényvisszaverődés azt jelenti, hogy a felületre beeső fénysugár minden irányban egyenletesen verődik vissza (szóródik), tehát a beeső fénysugár nem egy kitüntetett irányba verődik vissza (ld. a mellékelt ábrát). Ez azt jelenti, hogy a világosság, ami a diffúz visszaverődésből adódik, a szemlélő nézőpontjától független, azt kizárólag a felület (színe és geometriája) és a fényforrás (annak helyzete, színe és intenzitása) határozza meg. Lényegében a diffúz rész határozza meg a felület világosságát.



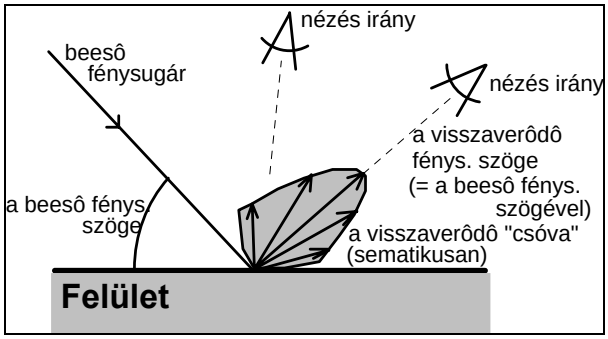
Szétszórt fényvisszaverődés

- **Tükröződő fényvisszaverődési együttható, K_s:**
  A tükröződő fényvisszaverődés együtthatója azt adja meg, hogy az optika törvényei szerint (*beesési szög=visszaverődési szög*) milyen mértékben verődik vissza a fény egy meghatározott irányban. Ebből következik, hogy a felületen a türöződő fényvisszaverődés okozta világosság-eloszlás függ a nézőponttól: A felületnek az a része tűnik majd a legvilágosabbnak, ahol a nézőpont, a visszaverődő fénysugárba esik. Szigorúan matematikai értelemben ez a feltétel csak egy tetszőlegesen kicsiny felületrészre igaz, de ez ellentmond a megfigyelt tényeknek. A tapasztalatok azt igazolják inkább, hogy bár a visszaverődő fény a visszaverődési szög irányában a legnagyobb intenzitású, de a beeső fénysugár és a felület síkja közötti nagyobbik szögtartományban is van hatása, természetesen a visszaverődési szögtől bármely irányban távolodva csökkenő intenzitással. Ezt a változó intenzitású "csóvát" szemlélteti a tükröződő fényvisszaverődést bemutató ábra (ld. 3. ábra).

E három együttható (K_a, K_d és K_s) értékeinek összege hozzávetőlegesen 1-et kell adjanak, bár ettől az "ökölszabálytól" esetenként el lehet térni.

Az utolsó paraméter az intenzitás csökkenésének mértékét adja meg:

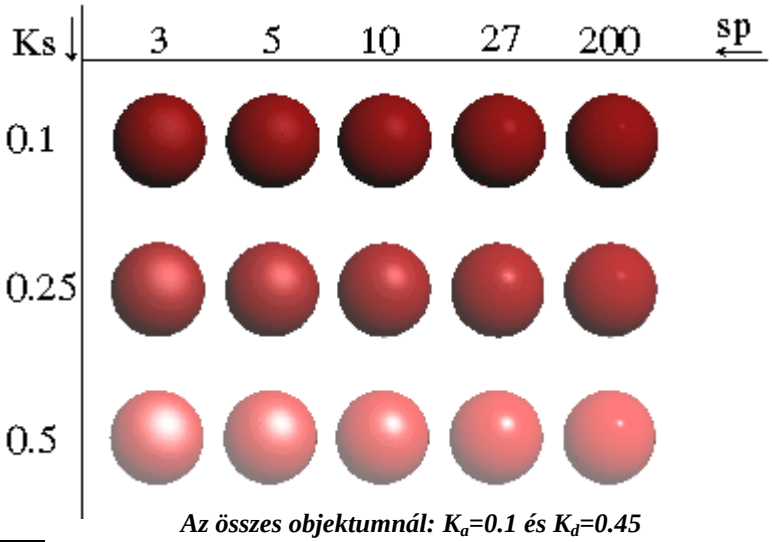
- **Tükröződő fényvisszaverődési kitevő, *sp*:** Amint már az előzőekben leírtuk, a tükröződő fényvisszaverődés világosságértéke ott a legnagyobb, ahol az optika feltételei (*beesési szög=visszaverődési szög*) a leginkább teljesülnek. A tükröződő fényvisszaverődés kitevője azt méri, hogy milyen mértékű a tükröződés intenzitása abban a térirányban, amerre a fenti feltétel nem "optimálisan" teljesül, azaz amerre a beesési szög nem *pontosan* egyezik a visszaverődés szögével. Minél nagyobb *sp* értéke, annál inkább egy adott térirányba összpontosul a visszaverődés, vagyis annál "karcsúbb" lesz az ábra szerinti "csóva".



Tükröződő fényvisszaverődés

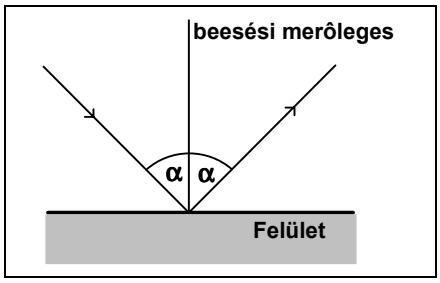
Mivel a korábban említett okok miatt ezek az együtthatók (K_a, K_d, K_s és *sp*) fizikai értelembe véve *nem* anyagjellemzők, ezért megadásukhoz próbálkoznunk kell. Az egyéni kísérletekhez támpontul szolgál a következő ábra.

A visszaverődési együtthatók variációi:



Az összes objektumnál: K_a=0.1 és K_d=0.45

A felület anyagparaméterei mellett fontosak a felületek geometriai tulajdonságai is. A felületek a normálisukkal jellemezhetők. Ez sík felület esetében egyszerűen a felületre merőleges vektor, és a sík minden pontjában azonos irányú, amint az alábbi ábráról leolvasható.



Beesési és visszaverődési szög

A modellek sokszor nem csak síkfelületekkel rendelkeznek. Emiatt a normálisokat nem csak sík felületre kell értelmezzük, hanem íves határoló felületekkel rendelkező testekre is (gömb, henger, tórusz, szabadon-formált felületek, forgástestek stb.). Ezeknek a felületeknek a normálisai általában nem állandó irányúak, mint a síkfelületek normálisai, hanem függenek a vizsgált felületi ponttól. Egy görbült felület minden pontjához egy ún. pontnormális rendelhető hozzá. Ez a pontnormális határozza meg, hogy melyik a test külső oldala a vizsgált pontban. Pl. gömb esetében a felület egy pontnormálisát a sugárnak a gömbfelületen túli meghosszabítása eredményezi. Általánosságban egy tetszőleges felület pontnormálisa az adott felületet az adott pontban érintő sík normálisa.

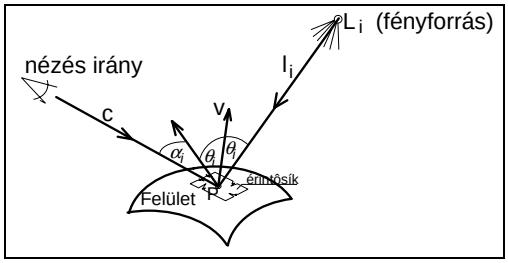
Az eddigi alapismereteket felhasználva a felület szín-számítása matematikai formában is leírható. A felület teljes fényintenzitása a felületi pontot a nézőponttal összekötő vektortól (c), a megfigyelt felületi pont normálvektorától (v), és a felület anyagjellemzőitől függ, továbbá az összes fényforrástól (L_i, i=1...n), ahol l_i az i.-ik fényforrásból érkező fénysugár felületi pontba érkező vektora.

Ekkor a felület egy P pontjában a teljes fényintenzitás:

$$I_P = K_a^F \cdot I^F + \sum_{i=1}^{n} \left(K_d^F \cdot I_i \cdot \cos\theta_i + K_s^F \cdot I_i \cdot \cos^{sp}\alpha_i \right)$$

ahol $\cos\theta_i$ = a felületnormális és l_i fénysugár-vektor között bezárt szög (beesési szög), és $\cos\alpha_i$ = a felületi pont nézési irányának az eltérési szöge a fényvisszaverődés irányától, továbbá:

I_i=az i.-ik fényforrástól és a felülettől függő RGB színérték,
I^F=a felület RGB színértéke,
K_a^F=a felület környezeti fényvisszaverési együtthatója,
K_d^F=a felület szórt fényvisszaverési együtthatója,
K_s^F=a felület tükröződő fényvisszaverési együtthatója
és
sp=a felület tükröződő fényvisszaverési kitevője.



A felület P pontjának vizsgálata a fényintenzitás számításhoz

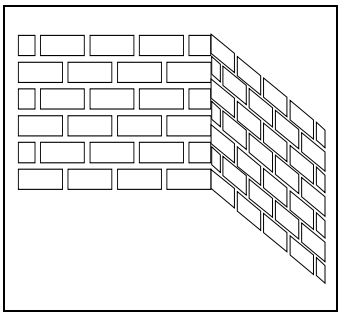
A fényforrások I_i intenzitása (színértéke) a valóságban csak a nagyon távoli – párhuzamos sugarakkal rendelkező fényforrások esetében tekinthető állandónak, egy izzólámpa esetében az intenzitás csökken. Ezt a rendering-eljárások is figyelembe veszik pontszerű fényforrások esetében az intenzitás csökkenéssel (attenuation - csillapodás).

A fényforrások csoportosítása:

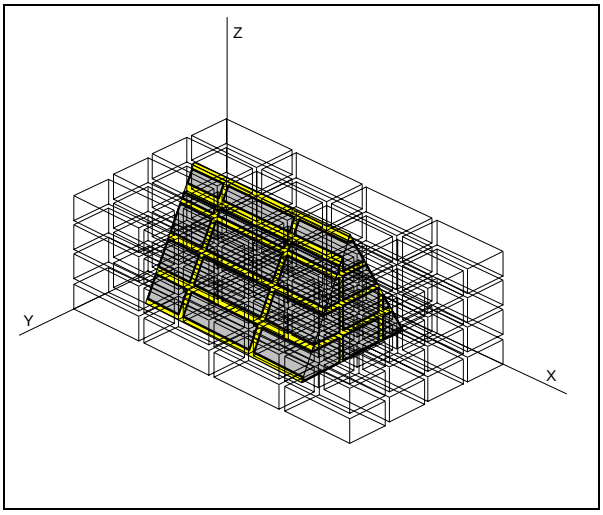
- **Párhuzamos** megvilágítás. Ennél a megvilágítási módnál a fénysugarak párhuzamosak és a fény intenzitása nem változik (ez modellezi pl. a napfényt).
- **Pontszerű** megvilágítás. Ennél a megvilágítási módnál a fénysugarak egy pontból indulnak ki és a fény intenzitása a távolság függvényében csökken (ez modellezi pl. az izzólámpát).
- **Irányított pontszerű** megvilágítás. Ez a megvilágítási mód az előzőhöz hasonlít, de a fénysugarak nem minden irányban haladnak, csak egy megadott fénykúpon belül, és a fényintenzitás nem csak a távolság függvényében, hanem a fénykúp tengelyétől a kúppalást felé is csökkenhet (ez modellezi pl. spotlámpát, reflektort).

Mint az eddigiekből kitűnt, a felületek színét a felülethez rendelt RGB színérték alapján a nézőpont, a megvilágítás és a felületi jellemzők alapján lehet meghatározni. Ezzel azonban nem lehetne egy anyagszerű felületet ábrázolni, hiszen egy felületen belül eltérő színek lehetségesek a felület textúrájától függően. Pl. fa, márvány vagy téglafelületek nem jellemezhetők egy homogén színnel. Az anyagszerkezet modellezésére a rendering-eljárások alapvetően kétféle módszert alkalmaznak:

- **picture mapping:** Egy előre elkészített és a számítógépben tárolt képet az adott felületre vetítő eljárás. Előnye, hogy egyszerű a felületekre vetítendő képet létrehozni, hiszen pl. faerezet, téglamintázat, stb. esetén egy megfelelő kép-letapogató eszközzel (scanner) közvetlenül létrehozható a textúra képe. Hátránya viszont, hogy a szomszédos felületek határán a mintaátmenet nem lesz folyamatos.

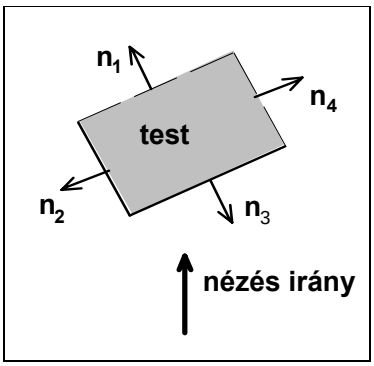


- **procedurale texture:** Ebben az esetben egy eljárás kiszámítja a háromdimenziós tér minden pontjában a színértéket. Az e térbe helyezett felületekre rávetülnek a felületek által kimetszett színértékek, így a szomszédos felületek határán is folyamatos lesz a mintaátmenet. A szomszédos ábra egy téglafalak és fugáik által leírt térbe helyezett prizmatestet mutat, amely a felületeivel kimetszi a téglamintázatot a fugákkal.



A képkiszámítási algoritmusok gyakran használnak a számítások gyorsítására még egy eljárást, amit itt az alapfogalmak között kell megemlítsünk, az ún. "backface culling" eljárást.

Backface culling-ként azt az eljárást jelölik, amelynek során a kép előállításakor eltávolítják azoknak a felületeknek az adatait, amelyeknek a normálisai a nézési iránytól távolodó irányba mutatnak. Ha az X-Y síkban felveszünk egy metszetet egy testen keresztül, akkor láthatjuk hogy a testnek azon felületei, amelyek normálisai a backface culling-feltételeket kielégítik, az előttük lévő felületektől amúgyis takartak, így a láthatóság szerinti képen nem jelennek meg.



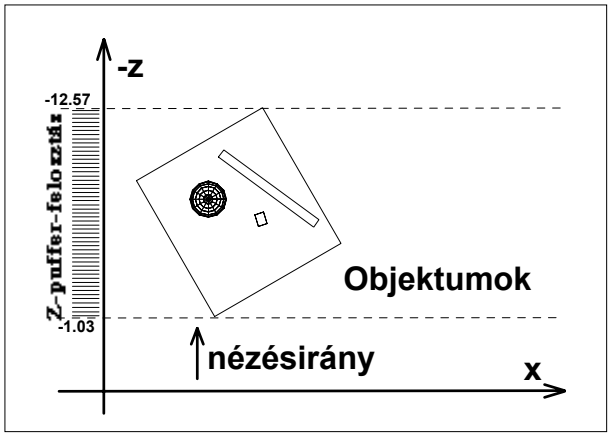
Backface culling (hátsó-sík kiválasztás)

Z-Puffer, vagy mélység-puffer algoritmus.

A számítógépi kép előállítására szolgáló modell lényegében sok kisebb-nagyobb síkfelületből áll, miután az íves felületek is burkoló síkfelületekkel közelíthetők.

A z-puffer eljárás az alapja számos képelőállító módszernek, így többi között a Flat-Shading, Gouraud-Shading és Phong-Shading eljárásoknak is. Ezért érdemes a z-puffer eljárás munkamenetével megismerkedni.

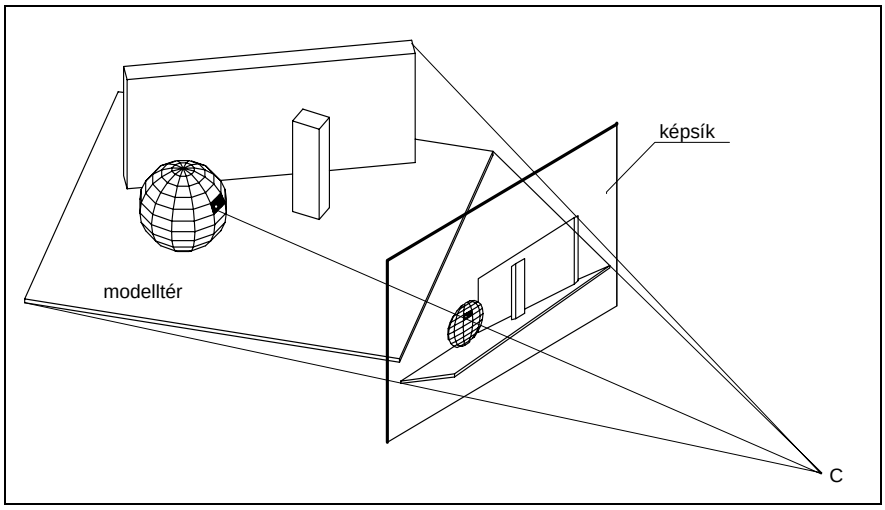
A z-puffer működéséhez tekintsünk meg egy modellt felülnézetben. A képernyő, amin a modellt (a későbbi kiszámított képet) ábrázoljuk, álljon a mellékelt ábrának megfelelően a papír síkjára mint alapsíkra merőlegesen. A modell z irányban egy meghatározott értéktartományt fed le - a példánkban -1.03-tól -12.57-ig[2]. Most ezt a tartományt, amit a modell a *z* irányban lefed, diszkrét, egészszám értékekre felosztjuk. A hátsó határ (esetünkben: -12.57) feleljen meg a lehetséges legkisebb értéknek, (rendszerint 0-nak), az első határ pedig a lehető legnagyobbnak, és a *z*-értékekhez e két - legkisebb és legnagyobb határ közötti – értékeket rendeljük.



Egy z-puffer felépítés

Hogy a legnagyobb érték milyen konkrét számértéket vehet fel, az az úgynevezett z-puffer mélységétől függ, amit bit egységekben mérnek. A használatos z-puffer mélységérték 16 és 24 bit között van, ami 65536 -tól ≈16,8 millió diszkrét értéknek felel meg.[3]

Maga a z-puffer nem más, mint egy képernyőfelbontás méretű táblázat, tehát minden egyes képpont egy z-puffer értéket kap. A képfelépítés kezdetekor a rendering-eljárás törli a z-puffert, minden számértéket a leghátsó határértékre állít be, azaz a z-puffer "celláit" 0 értékkel tölti fel. Ezután a rendering-program nekilát a modell felületeinek tetszőleges sorrendben való feldolgozásához. Ehhez először is átszámítja a felületeket a kiválasztott (párhuzamos, vagy perspektívikus) vetítés és lépték alapján a képernyőn való ábrázoláshoz. Leegyszerűsítve, ez a számítás eredményezi a felület sarokpontjainak a képernyőkoordinátáit.



A modell képernyővetülete

[2] Az X-Y-tengelyek esetünkben a képernyő síkjára illeszkednek.

[3] amennyiben a Z-puffer 16 bites, 65536 diszkrét értéket vehet fel.

Ezután a rendering-eljárás feldolgozza az összes - a sarokkoordináták által körülírt - képernyő pontot. Ehhez először végrehajtja a z-puffer tesztet: meghatározza a pixelek z irányú mélységét, és azt átszámítja z-puffer értékre. Az így kapott eredményt összehasonlítja azzal az értékkel, amely már egy korábban megtalált pixelnek a z-puffer cellájában szerepel. Ha az eredmény nagyobb, mint az ott lévő érték, az azt jelenti, hogy az aktuális felület pixele közelebb van a szemlélőhöz - így a pixel megjeleníthető - és a z mélységét mint új értéket a z-pufferbe beviszi. Ha az eredmény kisebb mint a már ott eltárolt érték, akkor az aktuális felület vizsgált pontja nem látható, mivel egy másik felület takarja.

A továbbiakban három különböző eljárást tárgyalunk, amelyek mind a z-puffer algoritmuson alapulnak. Maga a z-puffer eljárás azt mutatja meg, hogy egy felület egy adott pontja látható-e vagy sem. Ami még hiányzik, az a megjelenítendő pixelek színe. A további eljárásoknál a felület mindíg sík felületet jelent, akkor is ha íves felületekről van szó, mivel - felületmodellező rendszereknél - az íves felületeket is síklapokkal közelítik a rendering-eljárások. Az igényesebb módszerek azonban képesek a síklapközelítésű íves felületeket valóban íves felületként ábrázolni.

Flat- (Quick) Shading eljárás

A legegyszerűbb felületkitöltő eljárás a Flat-Shading eljárás (szokásos ezt Quick-Shading eljárásnak is nevezni). Ez a módszer a modell felületeinek csak a középpontjában határozza meg a korábbi matematikai képlet szerint az adott felület színértékét, és ezzel a színnel tölti ki a teljes síkfelület összes pixelét. Az eredményül kapott kép hasonlatos egy takartvonalas képhez, ahol a felületek még színesek is.

Flat-Shading modell

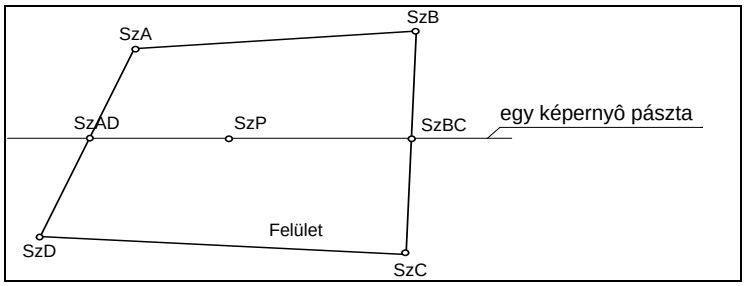
Előny: Az eljárás gyorsan eredményez kész képet, mivel csupán néhány bonyolultabb fényszámítást kell elvégezni.

Hátrány: Az így kapott kép a számítógépi képábrázolás legalacsonyabb minőségi foka. A testeket határoló, közelítő felületlapok egymástól határozottan elkülönülnek (ld. az ábrán a harang felületét).

Gouraud-Shading

A Gouraud-árnyékolási módszer először a felületlapok sarkainak színértékét határozza meg. A síklap-árnyékolással ellentétben a felület pixeljeit most nem azonos színekkel tölti fel, hanem súlyozott arányban a sarokpontok távolságától függően interpolálja a színértékeket minden egyes felületi pixelnél.

Egy A, B, C, D sarokpontokkal leírt felület sarokpontjaiban kiszámított színértékek: SzA, SzB, SzC és SzD. A felület határoló oldalain lineáris interpolációval kiszámított színértékek: SzAD és SzBC az SzA és SzD, ill. SzB és SzC színértékek között. Ekkor egy tetszőleges felületi pont színértéke (SzP) SzAD és SzBC színértékek lineáris interpolációja.



Színérték interpoláció

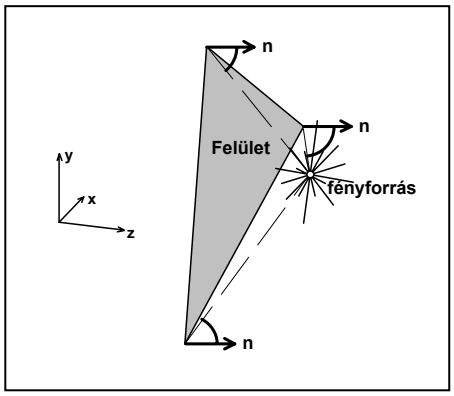
Miután az egymással határos lapok a sarokpontokon "osztoznak", ezért a két szomszédos lap közös élein az interpoláció azonos színértéket eredményez. Így a képről eltűnnek az élek, mint színtörési vonalak és az íves testfelületek valóban ívesnek is fognak látszani.

Gouraud-Shading modell

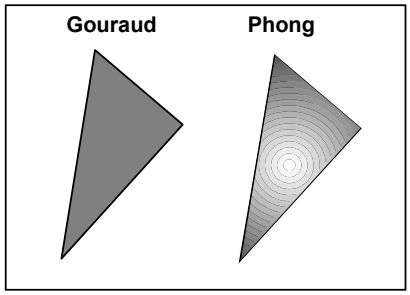
Előny: Ez az eljárás olyan képeket állít elő, ahol a gömb, és más hasonló testek íves felületei megfelelő képet adnak (v.ö. a Flat-Shading modell és a Gouraud-Shading modell ábráin a harang felületét).

Hátrány: A Gouraud-árnyékolási algoritmusból bizonyos geometriai elrendezés mellett az alábbi probléma adódik:

A Gouraud-Shading által helytelenül kezelt eset:



Ha egy pontszerű fényforrás olyan felület felett van, amely felületnek a kiterjedési méretei - a fényforrás és a felület közötti távolsághoz viszonyítva - közel azonosak (vagy nagyobbak), akkor az a fényfolt, amit tulajdonképpen a fényforrásnak a felületen eredményeznie kellene, nem jelenik meg (ld. az alábbi ábrán - Gouraud). Ez közvetlenül az interpoláció kiindulási adataiból adódik.



A Gouraud-Shading és Phong-Shading eljárás különbsége.

A síklap sarokpontjai adott szöget zárnak be a fénysugárral és ettől a szögtől függően vesznek fel egy-egy színértéket. Ha ezután interpolálunk, akkor természetes, hogy a felület egyik pixele sem lesz világosabb, mint a sarokpontok bármelyike, - a kívánt fényfolt tehát nem fog előállni.

Phong-Shading

Az ún. Phong-féle árnyékolási mószer megkísérli a Gouraud-féle árnyékolás fenti problémáját mind nagy síkfelületek[4], mind pedig íves felületek esetén általánosan megoldani.

Ehhez - a Gouraud-árnyékolástól eltérően - nem a színértéket interpolálja a sarokpontok színértékeiből, hanem minden egyes felületponthoz a pontnormálist interpolálja a sarokpontok pontnormálisaIból. Az így kapott pontnormálisok alapján számíthatók ki a korábbi részben megadott egyenletek szerint a pixel színértékek. Amint az könnyen belátható, a rendering-eljárás eredménye sík felületeknél még közeli pont-fényforrás esetén is helyes lesz (ld. előző ábra - Phong).

Példa modellünk most csak kis részleteiben különbözik az előző Gouraud-féle árnyékolással készült képtől.

Phong-Shading modell

Előny: Ez az eljárás olyan képeket hoz létre, amelyek nem csak gömb, vagy hasonló felületi formák esetében, de "durvább" geometriáknál, és több irányú fényforrás[5] mellett is nagyon reális képet eredményez.

[4] A fényforrás távolságához viszonyítva nagy síkfelületekről van szó.

[5] Pontszerű- és reflektor-fényforrások esetén.

Hátrány: A matematikai számolási igény ennél az eljárásnál sokkal nagyobb, mint az előző esetben. Ennél fogva a Phong-képekhez több számolási idő szükséges, mint a Gouraud-képekhez, - de ez a mai számítógépek növekvő számítási teljesítményei mellett mind jelentéktelenebb hátrány.

Raytracing (fénysugárkövetés)

Az eddig ismertetett rendering-eljárások (z-puffer eljárások) több olyan lényeges elvárásnak nem felelnek meg, amely egy fotó-realisztikus képpel kapcsolatban elengedhetetlen. Eddig nem volt szó a vetett-árnyékok, az átlátszó-áttetsző felületek és a tükröződések megjelenítéséről. Különösen ez utóbbi két hatás elérése az ún. raytracing-eljárással valósítható meg.

A képernyő egy pixelén keresztül, mint kis "lyukon" át nézve a mögötte lévő modellt, a pixel területén a modell egy színpontját látjuk (természetesen ez csak a szemünk felbontó képességéhez viszonyítva elegendően kicsi pixel terület esetében igaz). Ez a szín - abban az esetben, ha a modelltérben ábrázolt tárgynak egyetlen felülete sem látszik - a háttér színe, ill. ha ez a látósugár egy vagy több felületet metsz, akkor - a legközelebbi felülettel kezdve – megkezdődik a fénysugár követése.

Itt azonnal több lehetőség is felmerülhet:

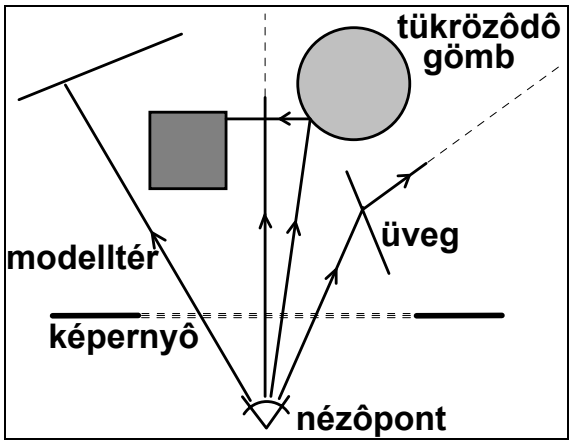
- Ha a "látott" felület matt színű, akkor e felület színértékét kell kiszámítani és megjeleníteni.
- Ha a szóban forgó felület tükröződő, akkor a vetítősugarat a visszaverődési irányban tovább követjük vagy a következő felületig, vagy a háttérbe. Következő felület esetén az egész folytatódik, végül a pixelen át az esetleg többszörösen visszaverődő fénysugár "végállomásán" talált színértéket jelenítjük meg, természetesen a tükröződések során "gyengülő" színértékkel.
- Átlátszó vagy áttetsző felületek esetén a fénysugarat a fénytöréssel módosított útvonalon követjük tovább, végül az így megtalált színérték és az átlátszó felület metszett pontbeli színértéke a felület átlátszóságának mértékétől függően súlyozva adják meg az adott pixel színét.

Természetesen elképzelhető a fenti esetek kombinációja is, pl. lehet egy felület egyidejűleg átlátszó és tükröződő is. Ekkor a fénysugarat ketté osztva követjük, egyrészt a visszavert, másrészt a felületen áthaladó (esetleg megtört) fénysugarat, és a két "végállomás" színértékén arányosan osztozik a pixel színe.

Itt említjük meg a vetett árnyékok számítási módszerét, bár ez nem kötött a raytracing-eljáráshoz, a fent említett z-puffer eljárásokon alapuló módszerek is figyelembe vehetik.

Arról van ugyanis szó, hogy a színérték kiszámításakor a korábban leírt matematikai képlet figyelembe veheti az összes fényforrást (L_i). Ha azonban csak azokat a fényforrásokat vesszük figyelembe, amelyek a kiszámítandó felületi pontból "látszanak", tehát egyetlen másik felület által sem takartak, akkor a módszer máris alkalmas a vetett árnyék megjelenítésére.

A raytracing programok működését szemléltő ábra:



A fénysugárkövetés sematikus folyamata

A korábbi modellünket a raytracing-módszerrel kiszámítva a mellékelt ábra szerinti képet kapjuk.

Előny: Ezzel a módszerrel előállított képek a tükröződések és árnyékolások révén a legmesszebbmenőkig valósághűen hatnak.

Hátrány: A matematikai számításokból adódó időráfordítás a sugárkövetésnél sokszorosan nagyobb, mint a z-puffer algoritmusok esetén. A raytracing kétségkívül az eddig szóba került eljárások közül a leghosszadalmasabb számolást igényli.

Raytracing (fénysugárkövetési) modell

Radiosity

A raytracing-eljárásnál léteznek a valóságot még hűségesebben visszaadó képelőállító módszerek is. Ha szemügyre vesszük alaposan a környezetünket, különösen a belső tér fény-árnyék határait, megállapíthatjuk hogy rendszerint az árnyékkontúrok nem élesek, hanem inkább elmosódottak (így pl. egy belső tér valamely sarkában). A raytracing-eljárás ellenben mindíg éles kontúrokat és vetett-árnyékokat eredményez. A visszavert fényszóródás az a jelenség, amely az elmosódott árnyékokat, és a súrolt fényt eredményezi. Ugyanis a felületek a rájuk eső fény egy részét visszaverik, és így mint másodlagos fényforrások megvilágítják a környezetüket. A felület anyaga, színe meghatározza, a felületet elhagyó fény intenzitását. Az ezt figyelembe vevő eljárás a radiosity. A radiosity egy rekurzív folyamat, mivel a felületekről visszaverődő szórt fény az általa "megvilágított" felületekről ismét visszaverődve visszahat az eredeti felületre. Mind a raytracing, mind pedig a radiosity rendkívül számításigényes eljárás, azonban ez utóbbi nagyságrendekkel nagyobb számítási időt igényel. Az a tulajdonsága azonban, hogy független a nézőponttól, lehetővé teszi animáció esetén, legalábbis ha a modelltér megvilágítása nem változik, hogy csak egyszer kelljen a számítást elvégezni, míg a sugárkövetést minden képkockán végre kell hajtani.

Radiosity eljárással készített kép.

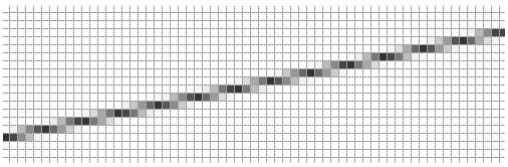
(Forrás: CADvilág c. folyóirat, 2000 február-március 4. évfolyam 1. szám.)

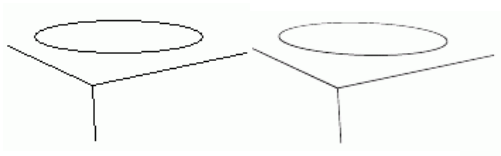
Anti-aliasing

Kitérünk még egy olyan hibajelenségre, amely a pixelgrafikus megjelenítők sajátosságából adódik. Vonalas ábráknál az egyenesek, vagy görbe vonalak, illetve a felületek találkozási kontúrvonalai a raszteres képpontmegjelenítés következtében lépcsőzöttséget mutatnak. Természetesen a képraszter irányaitól való különböző hajlásszögben eltérő vonalak, illetve a jobb-rosszabb képfelbontás esetén ez az ábrázolási hiba eltérő mértékben jelentkezik, de mindenképpen zavaró lehet. A lépcsőzöttségen kívül a raszterfelbontó eszköz lépésközénél kisebb, keskenyebb részek akár el is tűnhetnek, animációk esetén a képmozgás következtében ugrálás léphet fel. Ezeket a szépséghibákat nevezik aliasing-nek, E hibákat lehet javítani a rasztermegjelenítő eszközök fizikai tulajdonságainak javításával is (nagyobb felbontóképesség, animációknál több kép másodpercenként), de szoftveres megoldásokat is fejlesztettek ki a probléma kezelésére. Az azonos felbontás mellett, a szem számára szebb és teljesebb képet előállító módszereket nevezik anti-aliasing-nek.

Raszterképernyőn ábrázolt vonalak általában nem esnek pontosan a rasztervonalra. A vonalon lévő és a környező képpontok színének megváltoztatásával (a vonalszín és a környezetszín közötti átmeneti színekkel) simábban futó vonal hatása érhető el. A mellékelt ábrán látható, hogy a fekete színű vonal környezetében lévő pixelek fokozatosan a fehér háttér színébe szürkülő árnyalatukkal csökkentik a vonal lépcsőzöttségéből

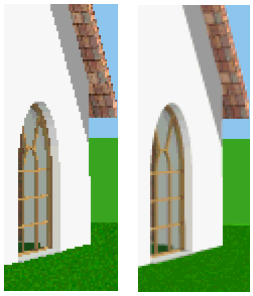
adódó kellemetlen hatást. A következő két ábra egyenes és görbe vonalak esetében mutatja meg, hogy milyen különbség érhető el az anti-aliasing használatával (bal oldalon az anti-aliasing nélkül, mellette az azzal készített kép). A kinagyított jobb oldali képen a vonalak mentén jól kivehetők a vonal- és háttérszín közötti átmenetet biztosító különböző szürkeárnyalatú pixelek.

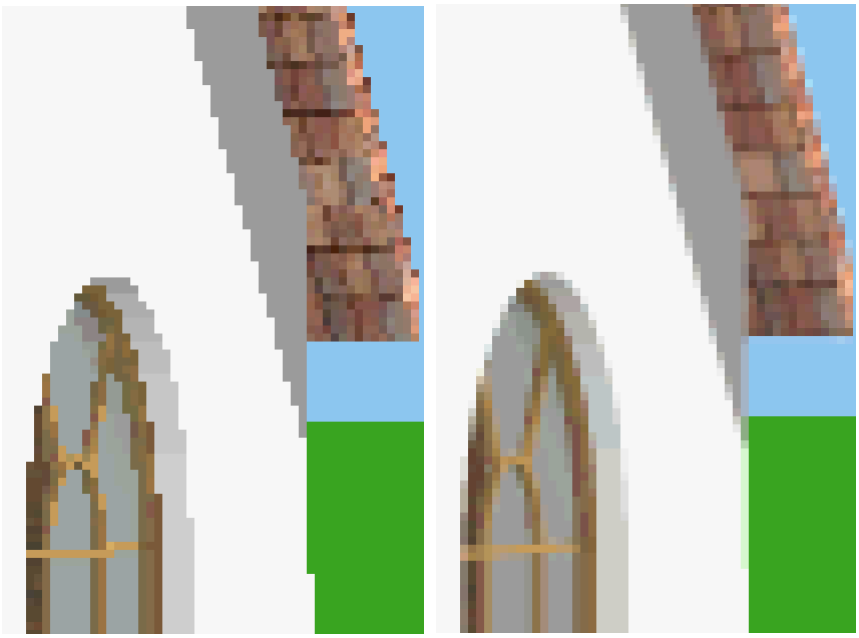




A fenti megoldás csak vektoros ábrák esetén segít, pixelenként előállított képeknél – mint pl. a raytracing

eljárással létrehozott képeknél – nem, ugyanis azoknál megszűnnek az egyes képpontok közötti összefüggések. Az ilyen esetekben alkalmazott eljárások lényege, hogy az egyes képpontok színét a környező képpontok színéhez közelítik különböző átlagolási, súlyozási szempontok szerint, aminek következtében az éles kontúrok lágyulnak, kissé elmosódnak. Az alábbi ábrák egy épületrészleten mutatják be az anti-aliasing eljárás hatását. A normál méretű képeken is érzékelhető, de még inkább kinagyítva látszik jól az anti-aliasing nélkül és az azzal készült kép közötti különbség.



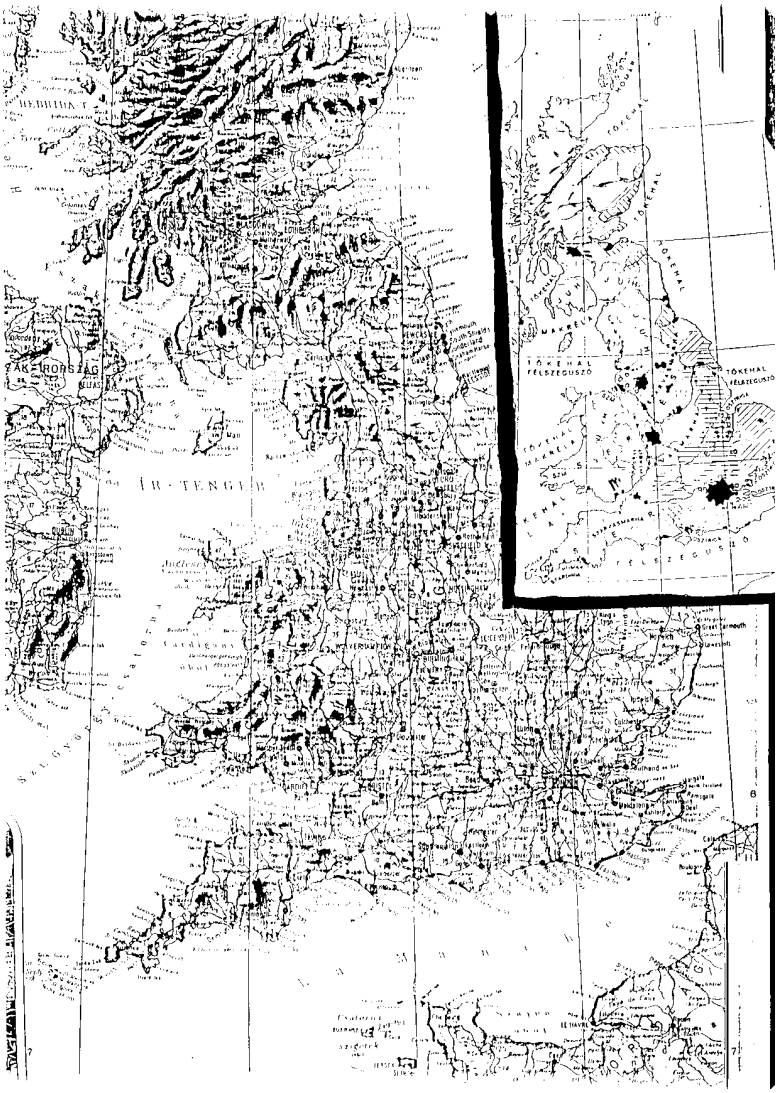


6. Fraktálok és építészeti alkalmazásuk

A múlt század végi Angliában felvetődött annak kérdése, hogy vajon pontosan milyen hosszú a szigetország partvidékének hossza. A problémát elsősorban nem az okozta, hogy az apály-dagály miatti tengerszint ingadozás megnehezíti a szárazföld határolóvonalának pontos megmérését, hiszen a szélsőértékek ismeretében már lehetett volna egy átlagos hosszt számítani, hanem az hogy milyen pontosságú legyen a mérés.

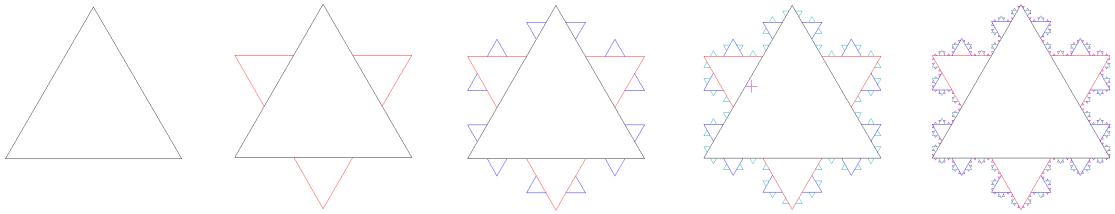
Természetszerűen ha különböző léptékű terveken végezzük el a mérést (amint az a mellékelt részletesebb ill. kevésbé részletes térképről látható), más-más eredményt kapunk, ugyanígy ha a valóságban egy méterrúddal mérünk vagy milliméter hosszúságú egyenesszakaszokkal közelítjük a tényleges hosszt, egyre nagyobb értékeket kapunk. Ebben nincs semmi váratlan, az azonban már meglepőnek tűnhet, hogy az egyre pontosabb eredmények minden határon túl nőnek, nem konvergálnak egy adott értékhez.

Helge von Koch 1870 - 1924
svéd matematikus



Ezt a problémát szemlélteti H. von Koch 1904-ben készített mesterséges sziget modellje, amelyet róla Koch-szigetnek neveznek:

Rajzoljunk első menetben egy egységnyi oldalú szabályos háromszöget. Következő lépesben osszuk fel a háromszög oldalait 3 egyenlő részre, és minden oldalon rajzoljunk az osztópontokra illeszkedő további szabályos háromszögeket. Az így kialakuló oldalakat is harmadoljuk és minden oldalon ismét rajzoljunk az osztópontokra illeszkedő további szabályos háromszögeket, majd mindezt folytatva egyre kisebb és kisebb háromszögeket. Az alábbi ábrasor az első 5 lépést mutatja:



0. lépés 1. lépés 2. lépés 3. lépés 4. lépés ...

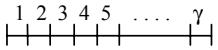
Írjuk fel a külső kontúr által alkotott síkidom területét és kerületét lépésenként:

lépés	Terület	Kerület
0.	$1/2 \cdot 3^{1/2}/2 = 3^{1/2}/4$	3
1.	$3^{1/2}/4 \cdot (1+1/9 \cdot 3) = 3^{1/2}/4 \cdot (1+1/3) = 3^{1/2}/3$	$3 \cdot 4/3$
2.	$3^{1/2}/3+3^{1/2}/4 \cdot 1/3 \cdot (4/9) = 3^{1/2}/3 \cdot (1+1/4 \cdot (4/9))$	$3 \cdot (4/3)^2$
3.	$3^{1/2}/3 \cdot (1+1/4 \cdot (4/9+(4/9)^2))$	$3 \cdot (4/3)^3$
4.	$3^{1/2}/3 \cdot (1+1/4 \cdot (4/9+(4/9)^2+(4/9)^3))$	$3 \cdot (4/3)^4$
...	...	...
n.	$3^{1/2}/3 \cdot (1+1/4 \cdot (4/9+(4/9)^2+(4/9)^3+...+(4/9)^{n-1}))$	$3 \cdot (4/3)^n$
...	...	...

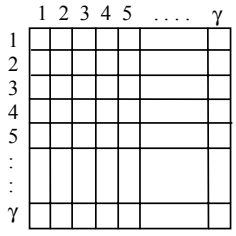
Ha *n* tart a végtelenhez, akkor a $(4/9+(4/9)^2+(4/9)^3+...+(4/9)^{n-1})$ egy végtelen mértani sorozat összege, ahol az összeg = $a_1/(1-q)$, ($a_1 = 4/9$; $q = 4/9$), *vagyis* $4/9/(1-4/9) = 4/5$. Ezt visszaírva a területszámítás képletébe $3^{1/2}/3 \cdot (1+1/4 \cdot 4/5) = 2/5 \cdot 3^{1/2}$ -at kapunk a területre, tehát a terület felülről korlátos (ez szemléletesen is belátható, ha pl. a kiindulási szabályos háromszög köré egy a kört írunk, a végső kontúr által határolt terület ezen a körön belül marad).

A kerület esetében azonban, ha *n* tart a végtelenhez, nem található felső korlát, minden határon túl növekszik. Ez többek között azért is probléma, mert ez alapján a fejezet elején felvetett kérdésre, hogy tudniillik milyen hosszú Anglia partvidéke, azt kell mondjuk, hogy végtelen. Ebből azonban az is adódik, hogy a hosszmérték nem alkalmas két ilyen alakzat összehasonlítására, hiszen eszerint végtelen hosszú földünk legkisebb szigetének és a legnagyobb kontinensének a kerülete is.

Erre a problémára a megoldás a mértékegység helyes megválasztásában keresendő. Az senkinek nem jutna eszébe, hogy két síkidom közül az egyiknek a kerületét a másiknak a területével hasonlítsa össze, mivel a hossz és a terület egymással nem összevethető két mértékegység. Ebből következik, hogy a fent felvetett probléma kezelésére talán más mértékegységet kellene választani. Az alábbi gondolatmenet erre próbál megoldást mutatni:



Vegyünk először egy egység hosszúságú szakaszt, és osszuk fel γrészre. Ekkor az elemi rész mérete $r = 1/\gamma$, az elemi részek száma pedig $N = \gamma$ lesz.



Ha egy egység-négyzet mindkét oldalhosszát a fentiek szerint felosztjuk γ részre, akkor az elemi rész mérete $r^2 = 1/\gamma^2$, míg az elemi részek száma $N = \gamma^2$ lesz.

Az első esetben vonalról, vagyis 1 dimenziós alakzatról, míg a második esetben síkidomról vagyis 2 dimenziós alakzatról volt szó. Másképpen fogalmazva a dimenzió mérőszáma először $D = 1$, másodszor $D = 2$ volt.

Általánosítva: $N = (1/r)^D$, ahol *N* az elemi részek száma, *r* a mérete és *D* a dimenzió mérőszáma.

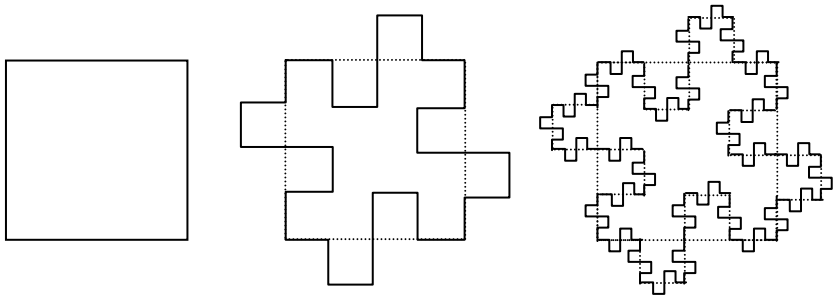
Nézzük meg a korábbi Koch-szigat egy oldalát, és kövessük annál is a fenti gondolatmenetet:

- Az elemi részek száma: $N=4 \cdot 4 \cdot 4 \cdots = 4^n$
- Az elemi részek mérete: $r=1/3 \cdot 1/3 \cdot 1/3 \cdots = (1/3)^n$
- Ekkor a fenti $N = (1/r)^D$ egyenlet alapján $4^n=(1/(1/3)^n)^D$, azaz $4^n=(3^n)^D$, amiből $\log(4^n)=D \cdot \log(3^n)$, vagyis $n \cdot \log(4)=n \cdot D \cdot \log(3)$.
- Egyszerűsítés és átrendezés után: $D = \log(4)/\log(3)$, ami ≈ 1.2618.

Elfogadva a korábbi gondolatmenetet, a dimenzió mérőszáma (*D*) itt nem egészszámra adódik, vagyis ez esetben nem egész, hanem tört dimenziós terekről, más szóval *fraktálokról* beszélhetünk.

Egy, talán még egyszerűbben követhető példával, egy négyzet alakzat kiindulású Koch szigettel is illusztráljuk a fraktálokat:

A kiinduló négyzet oldalait osszuk 4 azonos szakaszra, és az így kapott ¼ oldalhosszúságú kis négyzetekkel - az ábra szerinti módon - hol beharapva, hol kiugorva hozzuk létre a következő lépésbeli alakzatot. Az így kapott geometriai alakzat-sorozatnak vizsgáljuk a kerületeit, majd a területeit.



0. lépés 1. lépés 2. lépés

A kerület lépésenként:

lépés	db	elemi méret	méret	
0	*4*	*1*	$4 \cdot 1 = 4$	
1	$4 \cdot 8$	$1 / 4$	$4 \cdot 8 \cdot 1 / 4 = 8$	
2	$4 \cdot 8 \cdot 8$	$(1 / 4) \cdot (1 / 4)$	$4 \cdot 8^2 \cdot (1 / 4)^2 = 16$	
3	$4 \cdot 8 \cdot 8 \cdot 8$	$(1 / 4) \cdot (1 / 4) (1 / 4)$	$4 \cdot 8^3 \cdot (1 / 4)^3 = 32$	
...	...	...	...	
n	$4 \cdot 8^n$	$(1 / 4)^n$	$4 \cdot 8^n \cdot (1 / 4)^n = 4 \cdot 2^n$	(ha a lépésszám tart a végtelenhez, akkor a méret is tart a végtelenhez)

A terület lépésenként:

lépés	db	elemi méret	méret	
0	*1*	$(1)^2$	*1*	
1	$4 \cdot 4$	$(1 / 4)^2$	$(4^2) \cdot (1 / 4)^2 = 1$	
2	$4 \cdot 4 \cdot 4 \cdot 4$	$(1 / 4^2)^2$	$(4^2)^2 \cdot (1 / 4^2)^2 = 1$	
3	$(4^2)^3$	$(1 / 4^3)^2$	$(4^2)^3 \cdot (1 / 4^3)^2 = 1$	
...	...	...	...	
n	$(4^2)^n$	$(1 / 4^n)^2$	$(4^2)^n \cdot (1 / 4^n)^2 = 1$	(a lépésszámtól függetlenül a méret = 1)

Mint látjuk, a kerület n lépés után $4 \cdot 2^n$ lesz, ahol ha *n* minden határon túl növekszik, akkor a kerület is tart a végtelenhez. Ugyanakkor a terület (egyébként szemlélet útján is könnyen belátható, hogy a lépések számától függetlenül) mindíg 1.

Vegyük észre, hogy a terület számsorozata azért eredményezte a helyes eredményt, mert az elemi méreteknél az oldalhosszúságot a számunkra természetesnek tartott négyzetre emeléssel kapott elemi területekkel számoltuk. Tegyük ugyanezt a kerület sorozatánál is, vagyis emeljük az elemi méreteket valamilyen *D* hatványra úgy, hogy a kiindulási 4 érték állandó maradjon. Ekkor az n-ik lépés után a méretünk:

$4 \cdot 8^n \cdot (1/4^n)^D = 4$, amiből egyszerűsítés és átrendezés után: $8^n = 4^{nD}$.

A két oldal logaritmusát véve, és átrendezve: $D = lg\ 8 / lg\ 4 = lg\ 2^3 / lg\ 2^2 = 3 \cdot lg\ 2 / 2 \cdot lg\ 2 = 3 / 2 = 1,5$

Vagyis a fenti Koch-sziget kontúrvonalát $D = 1,5$ dimenziószámú fraktállal írhatjuk le.

Visszatérve a kiinduló problémánkhoz, amiatt nem hasonlítható össze két sziget kontúrvonalának a hossza, mert azok nem 1 dimenziós alakzatok. Hasonlóképpen belátható például egy szivacsot határoló felületekről vagy az emberi test bőrfelületéről és a természetben előforduló számtalan más felületről is, hogy azok nem 2 dimenziósak, hanem ugyancsak fraktálok 2 és 3 dimenzió közötti mérőszámmal.

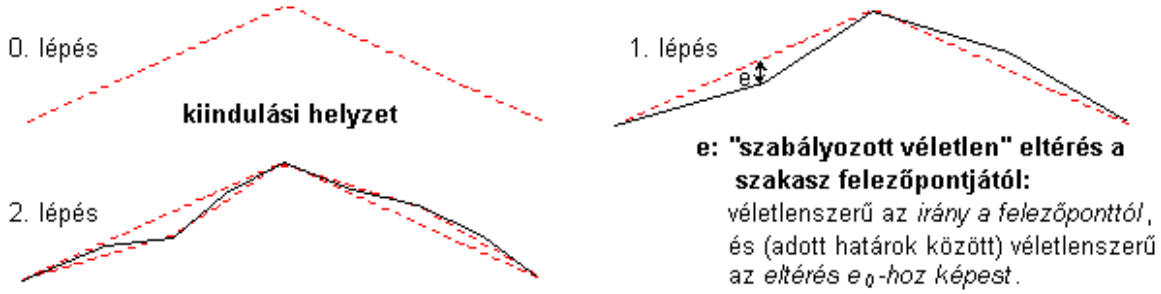
Benoit B. Mandelbrot 1924 - lengyel matematikus

Itt kell megemlítenünk Benoit B. Mandelbrot Varsóban született matematikus nevét, akinek elévülhetetlen a szerepe a fraktálok kutatásában, ill. azok különböző matematikai, geometriai területeken való alkalmazásában.

Az eddig ismertettekből a fraktálok két fontos tulajdonsága szűrhető le, mégpedig hogy a *fraktálok tört dimenzióval mérhetők*, és hogy ö*nhasonlók* (pl. a Koch-sziget egy adott lépésénél annak egy tetszőleges részletét vizsgálva feltűnik, hogy az olyan, mintha az egy korábbi lépésnél előálló részlet kinagyítása lenne).

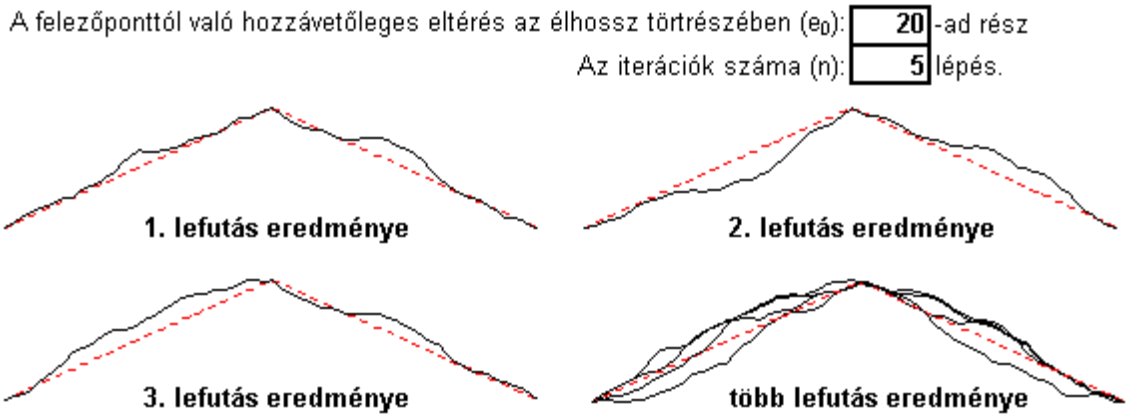
A fraktálok egyebek mellett felhasználhatók az építészeti környezet leírásánál. A korábban ismertetett Koch-szigat szabályos alakzatot eredményezett, de ha az algoritmusban szerepet kap a véletlen, akkor véletlen fraktálokról beszélünk. A véletlen fraktálokkal a természetet meglehetősen jól modellezhetjük. Egy fodrozódó vízfelület, vagy egy fa geometriai leírása és modellezése - amennyiben valósághűen kívánjuk ezt megtenni - igen bonyolult feladat. Az alábbi példa arra mutat rá, hogy egy alkalmasan megválasztott kiinduló alakzat és egy megfelelő algoritmus is képes leírni olyan természeti képződményeket, amelyeket egyébként csak nagyon nagyszámú közelítő felületelemmel írhatnánk le.

Vegyünk fel két szakaszt, amely egy hegygerinc lábainak két szélső pontját köti össze a csúcspontjával. A szakaszok felezőpontjait véletlenszerűen felfelé vagy lefelé mozdítsuk el adott korlátok között ugyancsak véletlenszerűen meghatározott távolsággal. Ugyanezt ismételjük minden újonnan létrejövő szakasszal, tetszőleges számú lépésen keresztül.



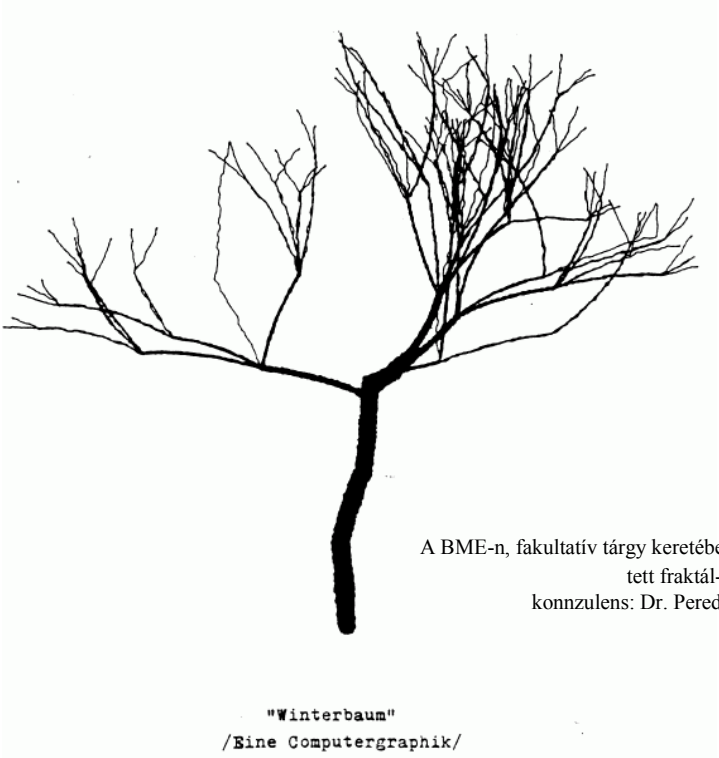
Az algoritmus első két lépése

Az algoritmus kiindulási adatait és néhány lehetséges eredményét mutatja az alábbi ábrasor, amely egy véletlen fraktálokkal előállított hegyvonulat sziluettjét mutatja be.



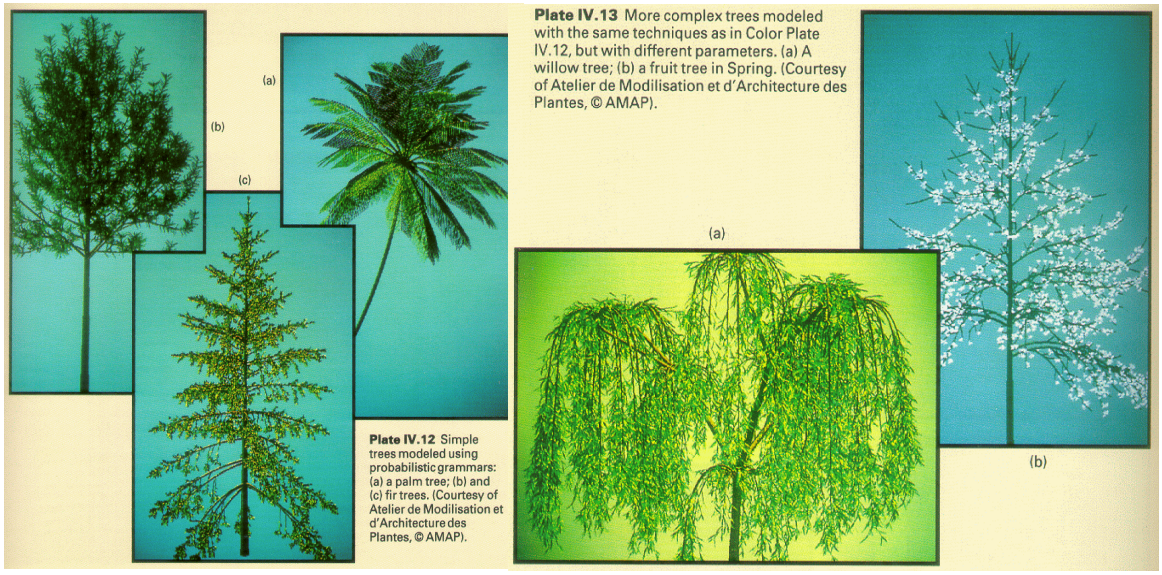
Azonos kiinduló paraméterek melletti különböző futási eredmények

Hasonlóképpen a mellékelt ábrán bemutatott fa rajzolata is véletlen fraktálokkal készült. A képet fakultatív tárgy keretében készítette egy építészhallgató.



A BME-n, fakultatív tárgy keretében készí-
tett fraktál-fa képe.
konnzulens: Dr. Peredy József

Végezetül bemutatunk néhány szép példát, amelyekben ugyancsak véletlen fraktál-felületek használtak az é-
píté-szeti környezet bemutatására. A képek James Foley, Adries van Dam, Steven Feiner és John Hughles szerzők Computer Graphics - Principles and Practice c. könyvéből valók. Az első képsorok különböző fajtájú fák fraktál előállítását példázza, a szabályosság és véletlen megfelelő megválasztásával.



Az alábbi példák a természeti környezet – erdők, hegyek, tengerpart – fraktálokkal előállított modelljeit mutatják be.

Plate IV.11 "Vol Libre Ridge": Fractal mountains generated with the Fournier–Fussell–Carpenter algorithm. (Courtesy of Loren Carpenter.)

Plate IV.19 A beach at sunset. (Courtesy of Bill Reeves, Pixar, and Alain Fournier, University of Toronto.)

Plate IV.1 An image generated with an iterated function system. The function system contains fewer than 120 affine maps. (Courtesy of Michael Barnsley, Arnaud Jacquin, François Malassenet, Laurie Reuter, and Alan Sloan.)

7. CAAD rendszerekhez kapcsolódó programnyelvek

A CAAD rendszerek fejlesztői többnyire igyekeznek minden építészeti tervezéshez szükséges szerkesztést, modellépítést támogatni, azonban gyakran előadódnak olyan egyedi feladatok, amelyeknél gyorsabban célt lehet érni bizonyosfokú programozási ismeretekkel.

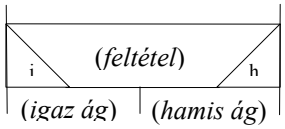
Ezt segítendő a CAAD rendszerek általában rendelkeznek valamilyen programozási lehetőséggel, amely nemcsak a programozók számára, de a többi felhasználóinak is lehetővé teszi a CAAD rendszer hatékonyabb kihasználását. Például az AutoCAD®-ben 'LISP', 'C' vagy az újabb verziókban 'Visual Basic' programnyelven lehet fejlesztéseket illetve kiegészítéseket írni. A tárgy keretében oktatott ArchiCAD® rendszer saját – Basic szerű – programozási felületet biztosít e célra, a *Geometric Description Language*-t, vagy rövidített nevén a *GDL*-t.

A programnyelvek célja elsősorban az, hogy paraméterezhető tárgyakat (nyílászárókat, bútorokat, lámpákat, helyiség-pecsét formátumokat, stb.) lehessen használatukkal létrehozni, amelyek azután a rendszer szerkesztő-modellező felületét kezelve - a már meglávőkhöz hasonló módon - kényelmesen használhatók.

Általánosságban elmondható, hogy egy programnyelv használatához algoritmikus gondolkodásmód szükséges, vagyis a program által végrehajtandó folyamatot át kell gondolni, megtervezni és a program számára értelmezhető formában leírni. Magára az algoritmus készítésére vannak különféle technikai eszközök, ezek közöl most a struktogramot, mint algoritmus leíró formát ismertetjük röviden.
Egy folyamat leggyakrabban egymást követő cselekmények – utasítások sorozata, amit ***szekvenciának*** nevezünk. Ezt a struktogram egymás alá rajzolt téglalapokkal jelöli, a téglalapokba beírva az éppen soronkövetkező teendőt, vagy végrehajtandó utasítást. Ilyen utasítás bármi lehet a legegyszerűbb alaputasítástól,

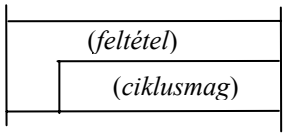
(akár „üres” utasítás, amikor nem kell semmit tenni – erre a SKIP utasítást szokás használni), az egészen bonyolult összetett utasításig, amelyeket persze egy újabb algoritmusban ki kell fejteni. Így egy struktúrált problémaleírást kapunk (innen a struktogram elnevezés), amelyből azután egy adott program saját utasításkészletének ismeretében megírható a konkrét program. A téglalapokba írt utasítások közül körülkrítve jelöljük az összetett utasításokat, amelyek további kifejtésre szorulnak.
Azonban csupán egymást követő utasításokkal – a már említett szekvenciákkal - nem írható le a folyamatok jelentős része. Igen gyakran ***elágazásokat*** kell alkalmazni, amikor adott feltételtől függően más-más a teendő, ill. ***ciklussal*** valósítható meg, ha mindaddig kell ismétlődően egy vagy több utasítást végrehajtani, amíg egy adott feltétel ezt megköveteli. Az ellágazások és ciklusok megvalósítására minden programnyelvben kell legyen megfelelő utasítás, legfeljebb azok „kényelmi szolgáltatásaiban” lehet különbség (pl. hogy többirányú elágazás is lehetséges-e, vagy ezt csak több egymásutáni kétirányú elágazással lehet megoldani, ill. a ciklusok esetében csak egy számlálóval biztosítható-e a ciklusok ismétlődési száma, vagy tetszőleges feltétellel szabályozható ez, stb.).

Az elágazások jelölése a struktogramban:



ahol a feltétel teljesülése esetén az igaz ág, nem teljesülése esetén a hamis ág utasításait kell figyelembe venni.

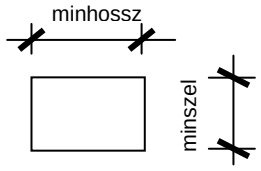
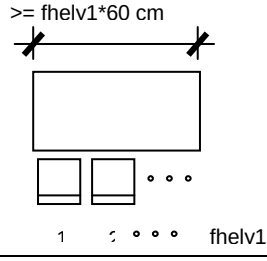
A ciklusok jelölése a struktogramban:

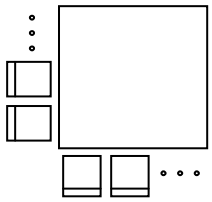
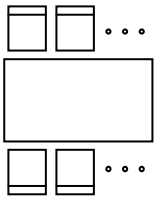
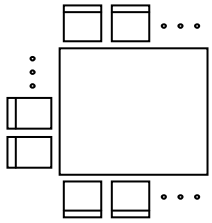
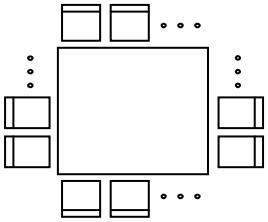


ahol a feltétel teljesülése esetén a ciklusmag utasításit kell figyelembe venni, egyébként lehet csak továbblépni. Ebből következik, ha nem kívánunk végtelen sokáig tartó ciklust, akkor a ciklusmag utasításai között kell legyen olyan, amely a feltételt hamisra váltja.
Mint említettük az utasítások lehetnek összetettek, amelyek egy másik algoritmusban kerülnek kifejtésre, de ezekkel az utasításokkal együtt közölhetjük azt is, hogy milyan paramátereket vegyen figyelembe az utasítást kifejtő leírás. Ennek szokásos formája az utasítás után zárójelek közé vesszőkkel (esetleg pontosvesszőkkel) elválasztott paraméterlista.
Fontos megjegyezni, hogy az algoritmusok folyamatokat írnak le, melyek során adott változó értékei megváltozhatnak. Ha egy változónak értéket adunk, azt az ***a* := *b*** (olvasd: *a* legyen egyenlő *b*-vel) formában tesszük, megkülönböztetve az egyenlőség vizsgálatától, amit pl. elágazások vagy ciklusok feltételeinél használhatunk, mint pl. ***a* = *b*** (olvasd: *a* egyenlő *b*-vel?).

A fentiek illusztrálására készítsünk struktogramot, amely az alábbi feladat algoritmusát írja le:

Feladat: ***Készítendő egy étkezőhely asztallal és székekkel az alábbi elrendezések szerint.***

az *elrendez* nevű változó értékei:	**a lehetséges elrendezések ábrái**
nincs szék:	
székek 1oldalt:	

az *elrendez* nevű változó értékei:	a lehetséges elrendezések ábrái
székek 2oldalt:	
székek szemben:	
székek 3oldalt:	
székek 4oldalt:	

Az algoritmus kezdeti, vagy bemenő paraméterei (amelyeket különbözőképpen megadva különböző eredményeket kapunk):

fhely: az összes ülőhely száma.

fhely1: az 1. sor (az ábrán vízszintesen az asztaltól lefelé elhelyezkedő sor) ülőhelyeinek száma.

elrendez: a fenti ábrák szerint a következő értékeket veheti fel:
"*nincs szék*", "*székek 1oldalt*", "*székek 2oldalt*", "*székek szemben*", "*székek 3oldalt*", "*székek 4oldalt*".

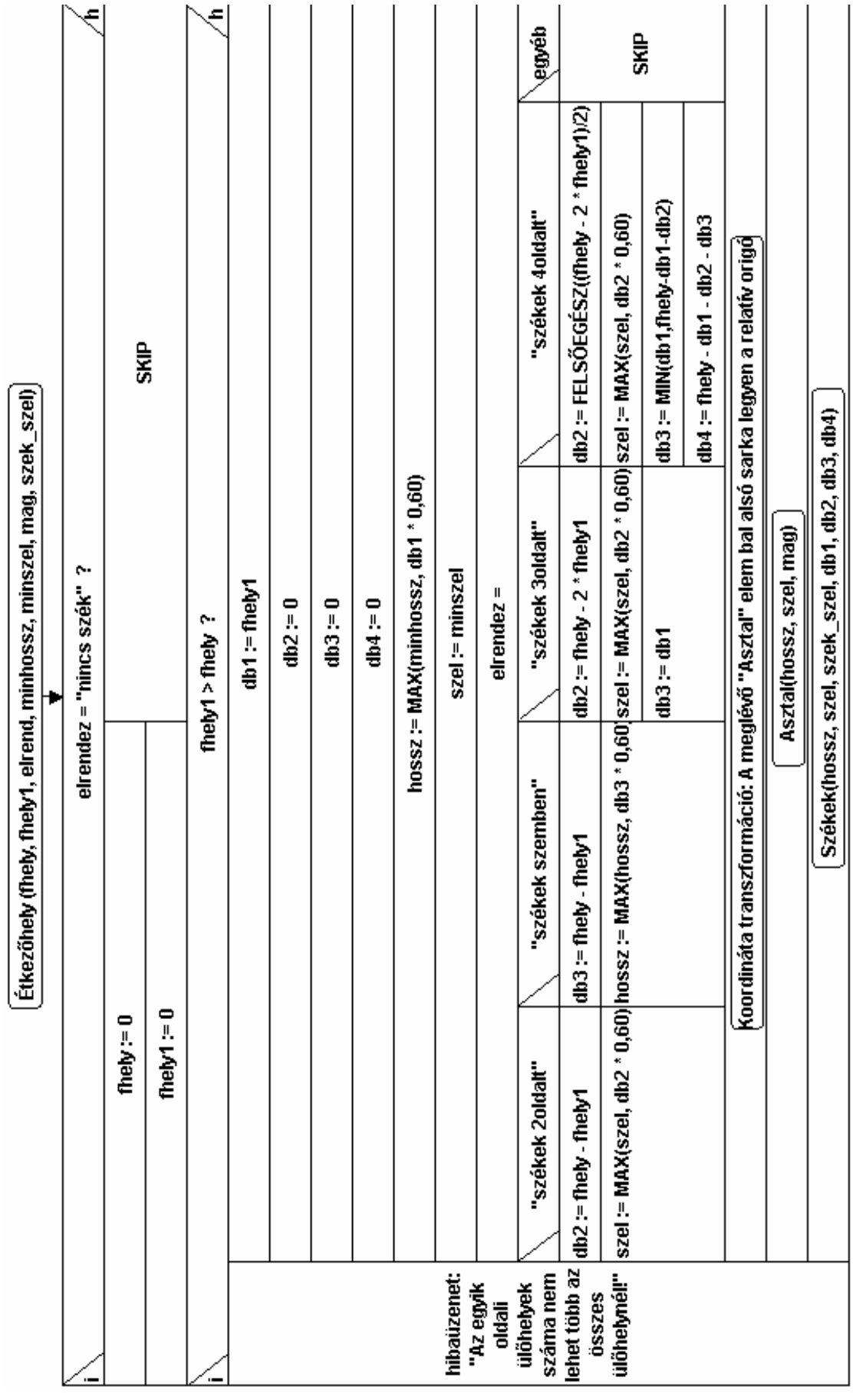
minhossz: az asztal legkisebb megengedett hosszúsági mérete.
(program teszteléskor adjuk meg pl. 80 cm-re)

minszel: az asztal legkisebb megengedett szélességi mérete.
(program teszteléskor adjuk meg pl. 80 cm-re)

mag: az asztallap felső síkjának magassága.
(program teszteléskor adjuk meg pl. 72 cm-re)

szek_szel: az ülőhelyek szélességi mérete.
(program teszteléskor adjuk meg pl. 44 cm-re)

A fenti feladat algoritmusát leíró struktogram egy lehetséges megoldása az alábbiak szerint nézhet ki:

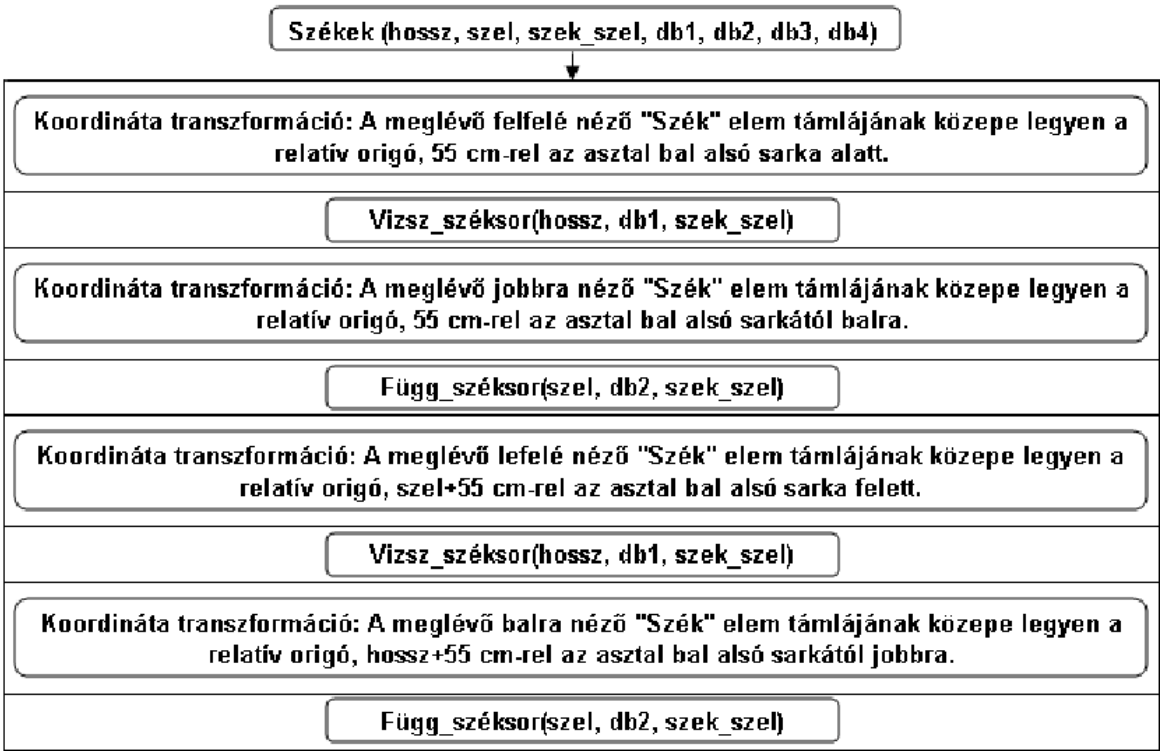


A struktogramot az alábbiak szerint kell „olvasni”, vagyis értelmezni:
Az algoritmus elején általában célszerű vizsgálni a bemenő paraméterek értékeit. Jelen esetben az első teendő megvizsgálni, hogy az *elrendez* nevű változó értéke „*nincs szék*”-e, mert ha az, akkor bármi is volt a *fhely* -nek és *fhely1* -nek megadva, azok 0 értékűek kell legyenek, egyébként marad az értékük annyi, amennyi a kezdeti beállított értékük volt.
A következő teendő megvizsgálni, hogy *fhely1* értéke nagyobb-e, mint *fhely*. Ha így van, az nyilván hibás érték, nem lehet az első sorban több szék, mint az asztal körüli összes székek száma. Ez esetben egy hibaüzenet kiírásával a végrehajtás befejeződik.
Ha rendben vannak a bemenő paraméterek, akkor folytatódik az algoritmus. Felveszünk 4 új változót, (*db1*, *db2*, *db3* és *db4*) melyek az asztal körüli alul, baloldalt, felül, ill. jobboldalt elhelyezendő székek darabszámának tárolására fognak szolgálni. Kezdetben a *db1* értékéül a bemenő *fhely1* -ben megadott számot adjuk át, hiszan az éppen ezt tartalmazta, míg a többi legyen egyelőre 0, majd a végrehajtás során az elrendezéstől függően kapják meg a végleges számértéküket.
Ezenkívül további két változó (*hossz* és *szel*) fogja tartalmazni az asztal vízszintes irányú, vagyis hossszúsági, ill. a függőleges irányú, vagyis szélességi méretét. A *hossz* értéke a kezdeti paraméterként megadott *minhossz* és a *db1**60 értékek közül a nagyobbik lesz (minden vízszintes irányú székre 60 cm-t számolva). Ezt adja értékül a **MAX(*minhossz*,db1*60)** függvény. A *szel* értéke egyelőre *minszel* lesz, később az elrendezéstől függően kapja meg végleges értékét.
Most jutottunk el oda, hogy megvizsgáljuk a különböző elrendezési eseteket, vagyis az *elrendez* nevű változó különböző értékei szerint elágazik a végrehajtás:

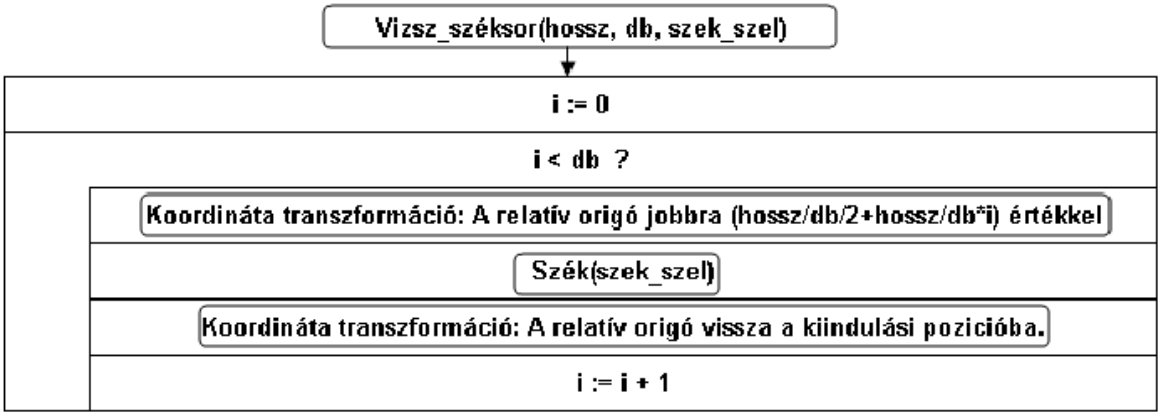
- Ha 2 oldalt vannak a székek, akkor nyilván függőleges irányban kell elhelyezni a vízszintesen még el nem helyezett székeket, vagyis az összes székek számából le kell vonni az alul elhelyezett székek számát. Ugyanakkor ezen székek száma megadja az asztal szélességi méretét is, ami hasonlóan a *hossz* számításához az eddigi *szel* és a *db2**60 értékek közül lesz a nagyobbik (a *szel* ezidáig megegyezett a kezdeti paraméterként megadott *minszel* értékkel, ezt néhány sorral feljebb biztosítottuk).
- Ha szemben helyezkednek el a székek, akkor nyilván felül kell elhelyezni az alul még el nem helyezett székeket, vagyis az összes székek számából le kell vonni az alul elhelyezett székek számát. Ugyanakkor ezen székek száma módosíthatja az asztal hosszúsági méretét is, ha több szék kerül felülre, mint alul van (ezt ugyancsak a **MAX(...)** függvénnyel tudjuk megadni).
- Ha 3 oldalt vannak a székek - feltételezve, hogy szemben azonos számú széket helyezünk el (ezt biztosítja ennek az ágnak az utolsó utasítása, a **db3:=db1** értékadás) - ekkor nyilván függőleges irányban kell elhelyezni a vízszintesen még el nem helyezett székeket, vagyis az összes székek számából le kell vonni az alul elhelyezett székek számának kétszeresét. Az asztal szélességi mérete itt is a 2 oldali székelrendezéshez hasonlóan adódik, továbbá ne feledkezzünk meg a *db3* értékének átadni *db1* értékét, biztosítva a szemben lévő székek azonos számát.
- Ha 4 oldalt vannak a székek – természetesen itt is feltételezve, hogy szemben azonos számú széket helyezünk el - ekkor nyilván függőleges irányban bal oldalt kell elhelyezni a vízszintesen még el nem helyezett székek felét (pontosabban a kapott székszám értéket felfelé kell kerekíteni egész számra). Ezt a **FELSŐEGÉSZ((fhely – 2 * fhely1)/2)** függvény számítja ki, vagyis az összes székek számából le kell vonni az alul elhelyezett székek számának kétszeresét, a kapott értéket 2-vel osztva és felfelé kerekíve adódik *db2* értéke. Az asztal szélességi mérete itt is a 2 oldali ill. 3 oldali székelrendezéshez hasonlóan adódik, hiszen a negyedik oldalon már nem lehet több szék, mint a második oldalon, mert e két függőleges oldalon elhelyezendő székek felét felfelé kerekítettük a második oldal szék számának kiszámításánál. Miután nem feltétlenül egyenletes lesz a székek száma a szemben lévő oldalakon, hiszen töredék székeket nem helyezünk el, továbbá a szemben azonos számú szék elhelyezésének ellentmondhat az, ha az első sorban már több széket helyeztünk el, mint az összes szék fele, ezért a harmadik oldal székeinek száma csak legfeljebb annyi lehet, mint az első oldalé. Ezt a **MIN(db1, fhely-db1-db2)** függvény biztosítja (ha az összes székek számából levonjuk a már két oldal elhelyezett székek számát és ez kevesebb az első oldali székek számánál, akkor már csak ennyit tudunk elhelyezni, sőt ez esetben hiába adtuk meg, hogy 4 oldalt kívánjuk elrendezni a székeket, a negyedik oldalra már nem marad szék). A negyedik oldalon természetesen már csak a maradék székek helyezhetők el. Az elágazás utolsó ágán (*egyéb*) nem kell semmit tenni, hiszen a "*nincs szék*", ill. a "*székek 1 oldalt*" esetekben a korábbi szék számok és asztalméretek értékein nem kell változtatni.

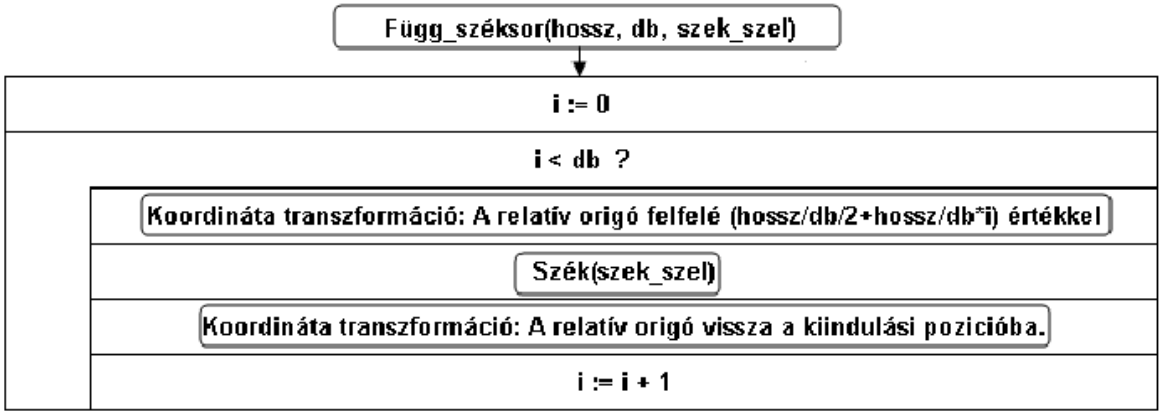
Bármelyik ágon is haladtunk végig, az asztal méretei, ill. az elhelyezendő székek száma körben megkapta a megfelelő értéket, így elkészíthető az asztal és a székek rajza.
Az asztal kirajzolása előtt válasszunk olyan koordinátarendszert, amelyben az elhelyezendő asztalunk további transzformáció nélkül elhelyezhető (célszerű az adott CAAD rendszerben meglévő asztal alapértelmezése szerinti koordinátarendszert választani, de persze mi is készíthetünk olyan asztalt, amely így fog elhelyezkedni)
A struktogram Asztal(hossz, szel, mag) utasítása egy olyan összetett utasítás, amelyet egy külön struktogramban kifejtve tudunk leírni, vagy ha a CAAD rendszerben van egy megfelelő ilyen objektum, vagyis a

programnyelv érti ezt az utasítást, akkor nem szükséges tovább kifejteni. Ez általában is igaz, hogy addig kell részletezni az algoritmust, ameddig az adott programrendszer számára érthető alaputasításokig nem jutunk. Pl. a fenti **MIN, MAX, FELSŐÉRTÉK** függvények is elvárhatóan léteznek a használt programnyelv függvényei között. Azonban ha mégsem, akkor kifejthetők egy további struktogramban, ahol le kell írni mit jelent pl. az **a := MIN(x,y)** függvény, vagyis két (esetleg több) érték közöl a kisebb. Ezt egy egyszerű elágazással tudjuk leírni: Ha **x < y**, akkor **a := x**, egyébként **a := y**.
A Székek(hossz, szel, szek_szel, db1, db2, db3, db4) összetett utasítás egy adott *hossz* és *szel* méretű asztal oldalai mentén rendre *db1*, *db2*, *db3* és *db4* számú és *szek_szel* szélességű széket elhelyezve rajzol ki. Ez az utasítás már feltehetően nincs a programnyelv alaputasításai között, így további struktogramban kell kifejteni.
Az alábbi struktogram ezt írja le:



A fenti struktogram 4 pár azaz 8 utasításból áll, amely párok egy-egy sor széket helyeznek le. Magát a széksor-elhelyezést mindig megelőzi egy utasítás, amely a koordinátarendszert úgy veszi fel, hogy a széksor azonos módon legyen elhelyezhető.
Itt már szerepet kapott a konkrét megvalósítás, vagyis az ArchiCAD® GDL programnyelv sajátosságainak ismerete is, mivel az abban használt koordináta transzformáció miatt célszerű volt külön kezelni a függőleges és vízszintes széksorok elhelyezését. Azonban általánosan megírható lenne csak széksor elhelyezéssel is az algoritmus.
A további struktogramok a Vízsz_széksor(hossz, db, szek_szel) ill. a Függ_széksor(hossz, db, szek_szel) utasításokat, vagyis az adott méretű asztal mellett elhelyezendő, adott számú és szélességű szék elhelyezését részletezik:





Mindkét fenti struktogram egy ciklust ír le, ahol az *i* nevű (úgynevezett ciklusváltozó) értéke kezdetben 0, majd a ciklus feltétel megvizsgálja, hogy ez az érték kisebb-e, mint *db*, vagyis az adott száksorban elhelyezendő székek száma. Ha nem kell egyetlen széket sem elhelyezni, vagyis *db=0*, akkor a ciklusfeltétel hamis lesz (hiszen 0 < 0 nem igaz), és a ciklusmag egyetlen egyszer sem hajtódik végre. Ha volt elhelyezendő szék, vagyis *db>0*, akkor a ciklusfeltétel nyilván igaz, és végrehajtódik a ciklusmag. A ciklusmagban a koordinátarendszert az *i* ciklusváltozó aktuális értékétől függő helyen vesszük fel, elhelyezünk egy széket, majd a kiindulási koordinátarendszer pozíciót visszaállítjuk. Végül *i* értékét eggyel megnöveljük. (Itt fontos a korábban leírt értékadás és egyenlőség-vizsgálat közötti különbség, hiszen az **i=i+1** matematikai értelemben nem teljesülhet, ha ez egyenletet jelentene. Itt természetesen a **:=** jelet használjuk, ami azt jelenti, hogy *i* értéke legyen az eddigi értékénél eggyel nagyobb. És ez biztosítja, hogy nem lesz végtelen ciklus, hiszen a ciklusmag minden végrehajtásakor *i* értéke növekszik, előbb-utóbb elérve *db* értékét és ezzel a ciklusfeltételt hamisra váltva.

A fenti példafeladatot átültetve pl. az ArchiCAD® GDL programnyelvére egy a kiinduló feltételeknek megfelelő étkezőgarnitúrát készíthetünk néhány paraméter beállításával. Természetesen létrehozhatunk ilyen étkezőhelyet grafikus szerkesztéssel, programozás nélkül is a CAAD rendszerben, de ekkor az asztalt is, és a székeket egyenként, ill. geometriai transzformációkkal (eltolás, forgatás, sokszorozás, tükrözés, stb.) kell elhelyezzük.

készült (javított, módosított változat): Budapesti Műszaki és Gazdaságtudományi Egyetem

2003 október