Debreceni Egyetem

Informatikai Kar

Felületek a számítógépi grafikában

Diplomamunka

Témavezető:

Dr. Kovács Emőd

tanszékvezető egyetemi docens

Készítette:

Kovács Balázs

programtervező matematikus hallgató

Debrecen

2008

Tartalomjegyzék

1. fejezet

Bevezetés

Témaválasztásomat sok dolog befolyásolta. Valahogy mindig is közel állt hozzám a számítógépes grafika. Az egyetem alatt először a Bevezetés a számítógépi grafikába kurzus alatt találkoztam komolyabban a témával. Ez a kurzus megfelelő alapot nyújtott a Komputergrafika tárgyhoz, mely után végül eldöntöttem, hogy mindenképp az adott témában szeretnék diplomamunkát készíteni.

Ami még befolyásolta a döntésemet, az az, hogy a számítógépi grafika egyre fontosabb tudományág, egyre jobban belefolyik a mindennapi életünkbe. Elég ha csak körbenézünk, manapság hány termék kinézetét tervezik számítógéppel, hány speciális effektet láthatunk a filmekben, és még sorolhatnánk. Sőt, mindenképp meg kell említeni a kizárólag számítógéppel készült termékekről, a mostanság egyre realisztikusabb számítógépes játékokról, valamint a teljesen számítógépes animációval készült egész estés mozikról. Sokszor annyira összefolyik a képzelet és valóság, hogy szinte már nem is lehet a kettőt szétválasztani.

Diplomamunkámmal szeretnék egy leírást adni a számítógépes felületekről, valamint ezek előállításáról. A matematikai alapokat és eszközöket, amik a tárgy megértéséhez szükségesek, tartalmazza a dolgozat.

A dolgozatot négy nagyobb részre tagoltam. Az első fejezetben a szükséges matematikai alapokat vettem sorra, bemutatom a vektorokat, mátrixokat, ezek műveleit, valamint a segítségükkel elvégezhető transzformációkat, amelyek majd leginkább a megjelenítések során lehetnek hasznosak. A következő fejezet a görbék előállítását tartalmazza, amelyek segítségével a tárgyalt felületeket elkészíthetjük. Az ezt követő fejezet a felületeket tárgyalja, itt az előzőekben leírtakra már csak hivatkozok. Az utolsó fejezetben az ábrázoláshoz, megjelenítéshez próbálok hasznos információkat nyújtani, valamint a legelterjedtebb technológiákat bemutatni dióhéjban, inkább kiindulási alapként, mint részletes dokumentációként.

Dolgozatommal leírást szeretnék nyújtani a számítógépi grafika világáról, az itt leírtak alapján

az Olvasó betekintést nyerhet a felületek matematikai előállításába és mivel ehhez elengedhetetlen a hozzájuk tartozó görbék ismerete, így azok genrálásába is. Ahol szükségét érzem, megpróbálom kódrészletekkel, ábrákkal segíteni a megértést és szemléltetni a tárgyaltakat.

Remélem, hogy dolgozatom segítséget nyújt a téma iránt érdeklődőknek, az elméleti tudnivalók terén annyira, mint a gyakorlati megvalósításnál.

2. fejezet

Matematikai alapok

Ebben a fejezetben bemutatok néhány matematikai konstrukciót, amelyek nélkül a grafika tárgyalása nehézkes lenne.

A fejezetben a vektorok és mátrixok elméletéről lesz szó. A témát nem tárgyalom teljes részletességgel, csak annyira, amennyire a továbbiakban szükséges a megértéshez.

2.1. Vektorok

A három dimenziós térben a pontokat, illetve egy görbe, vagy felület pontjait a pontba mutató, origó kezdőpontú három dimenziós (hely) vektorokkal adhatjuk meg:

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

ahol x, y, z a pont koordinátái.

Vektor-műveletek

Vektorok összeadása, kivonása

Legyenek adottak $\boldsymbol{v} = (x_v, y_v, z_v)$ és $\boldsymbol{w} = (x_w, y_w, z_w)$ három dimenziós vektorok. Ekkor a két vektor összegén, illetve különbségén a következőket értjük:

$$\boldsymbol{v} + \boldsymbol{w} = (x_v + x_w, y_v + y_w, y_v + y_w),$$

$$\boldsymbol{v} - \boldsymbol{w} = (x_v - x_w, y_v - y_w, y_v - y_w).$$

Vektorok skalárral történő szorzása

Legyen $\boldsymbol{v} = (x_v, y_v, z_v)$ három dimenziós vektor, $\lambda \in \mathbb{R}$ valós szám. Ekkor

$$\lambda \boldsymbol{v} = (\lambda x_v, \lambda y_v, \lambda z_v)$$

Vektorok hossza

Egy $\boldsymbol{v} = (x_v, y_v, z_v)$ vektor hosszán a következőt értjük:

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

Ez megegyezik a vektor euklideszi normájával.

Egy vektort normalizált vektornak nevezünk, ha $|\boldsymbol{v}| = 1$.

Vektorok skaláris szorzata

Legyenek adottak $\boldsymbol{v} = (x_v, y_v, z_v)$ és $\boldsymbol{w} = (x_w, y_w, z_w)$ három dimenziós vektorok. A két vektor skaláris szorzatán a

$$\boldsymbol{v} \cdot \boldsymbol{w} = |\boldsymbol{v}| \cdot |\boldsymbol{w}| \cdot \cos\phi$$

A skaláris szorzat meghatározható a következő módon is:

$$\boldsymbol{v} \cdot \boldsymbol{w} = x_v x_w + y_v y_w + z_v z_w$$

A skaláris szorzat eredménye egy szám lesz. A fentiekből látszik, hogy a két vektor által bezárt szög meghatározható a következő módon:

$$\phi = \arccos\left(\frac{\boldsymbol{v} \cdot \boldsymbol{w}}{|\boldsymbol{v}| \cdot |\boldsymbol{w}|}\right)$$

Ha $\boldsymbol{v}$ és $\boldsymbol{w}$ egységvektorok, akkor skaláris szorcatuk egyenlő a köztük által bezárt szög cosinusával, ha pedig $\boldsymbol{v}$ és $\boldsymbol{w}$ merőlegesek egymásra, akkor skaláris szorzatuk 0.

Vektorok vektoriális szorzata

Két vektor vektoriális szorzatának az eredménye egy a két vektorra merőleges vektor, melynek hossza $|v||w|\sin\phi$ valamint az $v, w, v \times w$ jobbsodrású rendszert alkot. A vektoriális szorzatot csak a 3 dimenziós térben értelmezzük.

$$\boldsymbol{v} \times \boldsymbol{w} = (y_v z_w - z_v y_w, z_v x_w - x_v z_w, x_v y_w - y_v x_w)$$

2.2. Mátrixok

A számítógépes grafikában a különböző transzformációkat mátrixok szorzásával tudjuk leírni, ezért mindenképpen kell ejteni egy pár szót a mátrixokról is. Egy mátrix általánosan a következő alakban írható fel:

$$A^{N\times M} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix},$$

ahol $a_{ij} \in \mathbb{R}, i, j \in \mathbb{N}$.

Mátrixműveletek

Mátrixok összeadása, kivonása

Két mátrix összegén, illetve különbségén a következőt értjük:

$$A^{N\times M} + B^{N\times M} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1m}+b_{1m} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2m}+b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nm}+b_{nm} \end{pmatrix}$$

$$A^{N\times M} - B^{N\times M} = \begin{pmatrix} a_{11}-b_{11} & a_{12}-b_{12} & \dots & a_{1m}-b_{1m} \\ a_{21}-b_{21} & a_{22}-b_{22} & \dots & a_{2m}-b_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}-b_{n1} & a_{n2}-b_{n2} & \dots & a_{nm}-b_{nm} \end{pmatrix}$$

Mátrixok szorzása

Ha több transzformációt hajtunk végre egymás után, azt elvégezhetjük egyenként is, illetve egyszerre a transzformációs mátrixok szorzatával is. Két mátrix szorzatának meghatározása:

$$A^{N\times M} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}, \quad B^{M\times K} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1k} \\ b_{21} & b_{22} & \dots & b_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{nk} \end{pmatrix},$$

$$C^{N\times K} = A^{N\times M}\cdot B^{M\times K} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ c_{k1} & c_{k2} & \dots & c_{kn} \end{pmatrix}, \quad c_{ij} = \sum_{l=1}^{m} a_{im}b_{mj}.$$

A mátrix-szorzás művelet általában nem felcserélhető, mivel szükséges, hogy az első mátrix oszlopainak száma megegyezzen a második mátrix sorainak számával.

Mátrixok transzponáltja

Egy mátrix transzponáltját úgy állíthatjuk elő, hogy felcseréljük a mátrix sorait és oszlopait, pl.:

$$\boldsymbol{M} = \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}, \quad \boldsymbol{M}^T = \begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix}$$

Általánosan: $\boldsymbol{A}_{ij}^T = \boldsymbol{A}_{ji}, \quad i,j \in \mathbb{N}$.

2.3. Homogén koordináták

A 3 dimenziós térben egy pont koordinátáját három adat segítségével tudjuk meghatározni. Homogén koordináták esetén a tér pontjai rendezett számnégyesekkel lesznek meghatározva, amelyek az arányosság erejéig vannak meghatározva. $(0,0,0,0)$ koordinátájú pont nem létezik.

$$\boldsymbol{x} = (x,y,z) \approx \boldsymbol{x}_h = (xw, yw, zw, w)$$

Szinte minden transzformáció elvégezhető a transzformációs mátrixok szorzásával, kivéve az eltolást (transzláció) [1]. Homogén koordináták esetén viszont ezek a mátrixok 4×4-esek lesznek, így minden transzformáció eklvégezhető mátrix-szorzásként is.

A homogén koordináták használatának előnyei:

– Duális tételek és definíciók használatának lehetősége.

– Projektív geometria használatának lehetősége.

– Végtelen távoli elemek használata (a negyedik koordináta nullának vételével).

2.4. Transzformációk

Mint az előző részben említettem, homogén koordinátákat alkalmazva, a transzformációk felírhatók 4×4-es mátrixok segítségével. A következőkben végigvesszük ezeket a transzformációkat a hozzájuk tartozó mátrixokkal együtt.

Mivel a kétdimenziós esetek a háromdimenziósakból visszavezethetők, azért azokra nem térnék ki külön.

Affin transzformációk

Affinnak nevezünk egy transzformációt, ha párhuzamosságtartó, tehát a párhuzamos egyenesek a transzformáció után is párhuzamosak maradnak, valamint megmarad a szakaszok osztási aránya és az alakzatok területének aránya.

Az általános affin transzormáció felírható

$$\boldsymbol{x}' = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}$$

alakban, $\boldsymbol{A}$ egy 3×3-as mátrix, $\boldsymbol{c}$ tetszőleges, Descartes-koordinátákkal megadott vektor.

Homogén koordinátás megadás esetén

$$\boldsymbol{x}' = \boldsymbol{A'}\boldsymbol{x},$$

ahol az $\boldsymbol{A'}$ speciális alakú

$$\boldsymbol{A'} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & c_1 \\ a_{21} & a_{22} & a_{23} & c_2 \\ a_{31} & a_{32} & a_{33} & c_3 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

ahol a_{ij} az előző definíció $\boldsymbol{A}$ mátrixának elemei, c_i a $\boldsymbol{c}$ vektor koordinátái.

Eltolás

Eltoláskor az alakzatot egy $\boldsymbol{v} = (a, b, c)$ vektorral toljuk el az eredeti pozíciójához képest. Ekkor a transzformáció mátrixa:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ha az eredeti pont (homogén koordináta-rendszerben) $\boldsymbol{x} = (x_1, x_2, x_3, 1)$, és az eltolás mátrixa pedig a fenti, akkor

$$\boldsymbol{x}' = \big(\boldsymbol{T}\boldsymbol{x}\big),$$

azaz a pont koordinátái és az eltolási mátrix szorzata.

Skálázás

A skálázás tulajdonképpen a dimenziónkénti méretarány-változtatást jelenti. A hozzá tartozó transzformációs-mátrix:

$$\boldsymbol{S} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

ahol s_x, s_y, s_z rendre az x, y, z tengelyek menti méretszorzót jelentik.

Forgatás

Mivel háromdimenzióban a forgatás három tengely mentén történhet, így a transzformáció megadásához három különböző mátrixra van szükség.

X-tengely körüli forgatás

$$\boldsymbol{R}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha - & \sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Y-tengely körüli forgatás

$$\boldsymbol{R}_y = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Z-tengely körüli forgatás

$$\boldsymbol{R}_z = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Tükrözés a tengelysíkokra

A forgatáshoz hasonlóan a tükrözés mátrixát is tengelyenként írjuk fel.
Tükrözés az YZ-síkra

$$\boldsymbol{M}_x = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Tükrözés az XZ-síkra

$$\boldsymbol{M}_y = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Tükrözés az XY-síkra

$$\boldsymbol{M}_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Nyírás

Adott egy origón atmenő fix sík $\boldsymbol{n}$ normálvektorával, egy a síkkal párhuzamos $\boldsymbol{t}$ irány, valamint egy $\lambda > 0$ valós szám. A hozzárendelés mátrixa:

$$\boldsymbol{N} = \begin{pmatrix} 1+\lambda t_x n_x & \lambda t_x n_y & \lambda t_x n_z & 0 \\ \lambda t_y n_x & 1+\lambda t_y n_y & \lambda t_y n_z & 0 \\ \lambda t_z n_x & \lambda t_z n_y & 1+\lambda t_z n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

ahol n_i és t_i rendre az $\boldsymbol{n}$ és $\boldsymbol{t}$ vektorok koordinátái.

Vetítések

Ha előállítottunk egy háromdimenziós modellt, akkor azt valahogy meg is szeretnénk jeleníteni. A megjelenítés legtöbbször képernyőn történik, így a háromdimenziós modellből kétdimenziós képet kell készítenünk. Erre szolgálnak a vetítési transzformációk. A vetítéseknél a háromdimenziós képet vetítjük le a kétdimenziós síkra. Ennek következményeként a vetített képnek kevesebb információtartalma lesz.

Párhuzamos vetítés

A koordinátarendszer origójából indítunk egy $\boldsymbol{v}$ vektort, ezzel párhuzamosan fogunk vetíteni az $[x, y]$ síkra, mint képsíkra. Ez azt jelenti, hogy egy pont képe az $[x, y]$ síkra úgy kerül, hogy a $\boldsymbol{v}$ vektor mentén addig mozgatjuk, amíg a z koordinátája nulla nem lesz. Így a pont képe ($\boldsymbol{p}'$) a következőképpen áll elő:

$$\boldsymbol{p}' = \boldsymbol{p} + \lambda \boldsymbol{v}.$$

Mivel $z' = 0$, ezért azt kapjuk, hogy

$$\lambda = -\frac{z}{v_z}.$$

Ezt koordinátánként behelyettesítve, mátrix-alakra hozva a következőt kapjuk a párhuzamos vetítés mátrixára:

$$\boldsymbol{M}_p = \begin{pmatrix} 1 & 0 & -\frac{v_x}{v_z} & 0 \\ 0 & 1 & -\frac{v_y}{v_z} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

Centrális vetítés

A centrális (középpontos) vetítést úgy képzelhetjük el, mintha egy nézőpontból egy sugarat engednénk a térbeli pontra , és az lenne a pont képe a képsíkon, ahol ez a sugár a képsíkot metszi.

Ha a $\boldsymbol{d}$ pont a nézőpont, amely a z-tengelyen található és az $[x, y]$ sík a képsík, akkor a következő összefüggést lehet felírni a vetítés eredményeként kapott $\boldsymbol{p}'$ pontra, ha annak koordinátái rendre x', y', z':

$$x' = x \cdot \frac{d}{d - z}; \quad y' = y \cdot \frac{d}{d - z}; \quad z' = 0,$$

ahol x, y, z az eredeti pont koordinátái.

A vetítést mátrixos alakban megfogalmazva:

$$\boldsymbol{M}_c = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{pmatrix}.$$

Fontos megjegyezni, hogy az $[x, y]$ síkkal párhuzamos, a nézőponton áthaladó sík (eltűnési sík) pontjai nem jelennek meg a vetítés eredményében.

2.5. Matematikai folytonosság

Folytonosságról akkor beszélünk, ha egy függvény egy adott pontban felveszi a határértékét. A definícióból látható, hogy a folytonosság pontbeli tulajdonság. A folytonosságnak a görbék kapcsolásakor lényeges a szerepe, hogy a kapcsolódási pontban milyen folytonosságot követelünk meg.

A folytonosságok definícióit több esetre bonthatjuk, aszerint, hogy két görbe kapcsolásakor milyen szintű és jellegű folytonosságot követelünk meg.

C_0 folytonosság

C_0 folytonosság esetén annyit követelünk meg, hogy a két görbe az illeszkedési pontban ugyanazt az értéket vegye fel. Így lehet törésük, viszont hézagmentesen kapcsolódnak.

C_1 folytonosság

A C_1 folytonosság annyival követel többet a C_0 folytonosságnál, hogy az illeszkedési pontban nem csak hézagmentesen kapcsolódnak a görbék, hanem az első deriváltjuknak is meg kell egyeznie, tehát ugyanaz lesz az érintőjuk.

C_2 folytonosság

Ebben az esetben nem csak az első, hanem a második deriváltak is megegyeznek az illesztési pontban, így az érintő és a görbület is azonos.

G_1 folytonosság

G_1 folytonosság esetén az érintő vektoroknak ugyanabba az irányba kell mutatniuk, viszont a nagyságuk eltérő lehet.

3. fejezet

Görbék

Görbén folytonos vonalat értünk. Egy görbe olyan egyenlettel definiálható, amelyet a görbe pontjai elégítenek ki. A 3D görbéket paraméteres formában is megadhatjuk:

$$x = x(t), \quad y = y(t), \quad z = z(t), \quad t \in [0,1].$$

Az paraméteres egyenletet a következőképpen értelmezhetjük. Ha egy $[0, 1]$ intervallumbeli t értéket behelyettesítünk a $x(t), y(t), z(t)$ egyenletekbe, akkor a görbe egy pontjának koordinátáit kapjuk.

3.1. Paraméteres görbék

A fenti megfogalmazással el is jutottunk a paraméteres görbék meghatározásáig. A paraméteres görbéket egy függvényként tekinthetjük, amely valós számok egy intervallumát képezi le a háromdimenziós térre. Az ábrázolás a definícióból eredően nem jelent problémát, mivel a görbének minden paraméterértékre meg tudjuk határozni egy pontját. Így a paraméter-tartományon végighaladva a görbének tetszőleges sűrűséggel meghatározhatjuk a pontjait (amiket később szakaszokkal összeköthetünk), így tetszőleges finomsággal ábrázolhatjuk az egész görbét.

Példa

Példaként tekintsük először a kört. A kör paraméteres egyenlete:

$$x = a + r\cos t, \quad y = b + r\sin t,$$

itt a t paraméter által fevehető értékek attól függ, hogy a körvonal mekkora részét vesszük, $t \in$ $\in [0, 2\pi]$ esetén megkapjuk a teljes kört. Az egyenletekben a és b a kör középpontjába mutató vektor x és y koordinátái, r pedig a kör sugara.

A második példa legyen egy szintén speciális, viszont már háromdimenziós görbe, a csavarvonal. A csavarvonalat a következő egyenletekkel írhatjuk le:

$$x = r\cos t, \quad y = r\sin t, \quad z = ct.$$

A fenti egyenletben r a csavarvonal sugara, t a paraméter, c pedig a csavarvonal emelkedése.

Kiemelném, hogy ezt a három egyenletet természetesen egy egyenletként is fel lehet írni, csak ezesetben a vektoros írásmódot kell alkalmazni. Utóbbi esetben az egyenlet:

$$\boldsymbol{r}(t) = r\cos(t)\boldsymbol{i} + r\sin(t)\boldsymbol{j} + ct\boldsymbol{k},$$

ahol $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ a koordinátarendszer x, y, z tengelyére illeszkedő egységvektorok.

3.2. Interpoláció

Az interpolációnál arra törekszünk, hogy egy adott ponthalmazra illeszkedő görbét adjunk meg polinom alakban (interpolációs polinom). Bizonyított, hogy egy n pontból álló halmazra illeszthető egy $(n-1)$-ed fokú polinom.

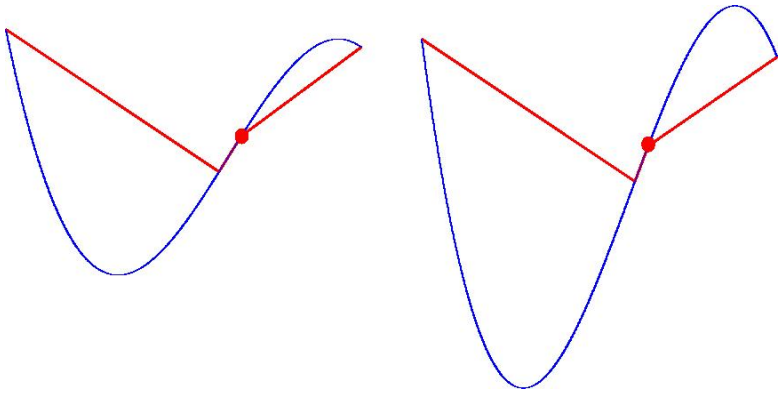
Sajnos az sztenderd interpolációnál adódik egy olyan probléma, hogy minél több pontot adunk meg, annál magasabb fokú lesz az interpolációs polinom. Ezesetben viszont hiába megy át az adott pontokon, közöttük nagyon ugrálni fog a függvény. Bár ezen segíthetünk valamennyit a pontok távolságának megváltoztatásával, de tökéletes eredményt nem lehet vele elérni.

Lagrange-féle interpolációs polinom

A $p_n(x)$ interpolációs polinomot $\mathbb{R}^2$-ben, az $x_0, \dots x_n$ alappontokban a következő alakban lehet megadni:

$$p_n(x) = \sum_{v=0}^{n} \prod_{\substack{\mu=0 \\ \mu \neq v}}^{n} \frac{x - x_\mu}{x_v - x_\mu} y_v.$$

Behelyettesítéssel ellenőrizhető, hogy az összes x_i alappontban a fenti polinom y_i értéket vesz fel. Az ábrán láthatjuk, hogy a jelölt pont minimális megváltoztatása mennyire hat a görbére.



3.1. ábra. Lagrange-interpoláció

3.3. Harmadfokú spline-interpoláció

A spline-interpoláció alapötlete az, hogy a pontok számától függetlenül, ne az összes pontra illeszkedő interpolációs polinomot adjunk meg, hanem közelítsük meg egy másik oldalról a problémát. Összeköthetnénk a pontjainkat egy töröttvonallal is. Ezesetben a görbénk olyan szinten megfelelő lenne, hogy áthalad minden ponton, viszont jobban szeretnénk egy simább, inkább „egy egészként” viselkedő görbét eredményül kapni. A harmadfokú spline-interpoláció alapja az, hogy az egész intervallumot részintervallumokra osztjuk, a részintervallumokban pedig alacsony fokszámú polinomokat használunk. Hogy megfelelő eredményt kapjunk, még hozzá kell venni az előzőekhez, hogy az intervallumhatároknál a polinomok illesztése folyamatos legyen.

A spline-interpolációt a numerikus analízisben is használják, függvények közelítésére.

A S spline-függvényről a következők mondhatók el:

1. S szakaszonként harmadfokú polinom. $S_i(x)$-szel jelöljük $S|_{[x_i,x_i+1]}$, ahol $i = 0, \ldots, n-1$
2. $S(x_i) = y_i$, tehát S áthalad az (x_i, y_i) koordinátájú pontokon
3. $S_i(x+1) = S_{i+1}(x+1)$, tehát folytonos
4. $S_i'(x+1) = S_{i+1}'(x+1)$, S-nek nincsenek törései
5. $S_i''(x+1) = S_{i+1}''(x+1)$, folytonos a görbülete
6. Peremfeltételek:
   - Természetes peremfeltételek: $S''(a) = S''(b) = 0$
   - Hermite-peremfeltételek: $S'(a) = f'(a), \quad S'(b) = f'(b)$

– Periodikus peremfeltételek: $S'(a) = S'(b), \quad S''(a) = S''(b)$,

ahol f a közelítendő függvény, $a = x_0, b = x_n$.

Ha felírjuk a feltételeket, az összes interpolálandó pont esetén, valamint az első és utolsó pont esetén hozzávesszük feltételként a peremfeltételeket, akkor egy n egyenletből álló n ismeretlenes egyenlet-rendszert kapunk, amely viszont speciális abban az értelemben, hogy az egyenlet-rendszerhez tartozó mátrix tridiagonális (csak a főátlóban és a mellette lévő két átlóban szerepelnek nullától különböző elemek). Ezt az egyenlet-rendszert megoldva megkapjuk az interpolációs görbét.

3.4. Hermite-interpoláció

A Hermite-interpoláció a spline-interpoláció egy speciális esete. Úgy illesztünk két pontra egy harmadfokú görbét, hogy adott a két pont, és mindkét pontban a görbe érintője. Ebben az esetben a paraméteres egyenlet:

$$\boldsymbol{p}(t) = (2t^3 - 3t^2 + 1)\boldsymbol{p}_0 + (t^3 - 2t^2 + t)\boldsymbol{m}_0 + (-2t^3 - 3t^2)\boldsymbol{p}_1 + (t^3 - t^2)\boldsymbol{m}_1,$$

ahol $\boldsymbol{p}_0, \boldsymbol{p}_1$ a két végpont, $\boldsymbol{m}_0, \boldsymbol{m}_1$ a meredekség a két végpontban és $t \in [0,1]$ a paraméter.

Ha több pontunk van, mint kettő – és ez a gyakoribb –, akkor a fenti módszert az egymást követő pontpárokra végrehajtva megkapjuk az interpoláló görbét.

3.5. Bézier-approximáció

Az interpolációval ellentétben, ebben a részben approximációról, illetve approximációs polinomokról lesz szó. Míg interpolációnál a kapott görbe átment a megadott kontrollpontokon, addig approximáció esetén, csak megközelíti őket az előre megadott sorrendben.

A Bézier-görbe tulajdonságai

Ha $t \in [0,1]$, akkor a görbe a kontrollpontok konvex burkán belül marad.

A görbe invariáns az affin transzformációkra (eltolás, forgatás, tükrözés, skálázás), így ezeket a transzformációkat nem szükséges az egész görbén végrehajtani, hanem elég csak a kontrollpontokon.

Egyetlen kontrollpont megváltoztatása hatással lesz az egész görbe alakjára.

Mielőtt pontosan megnéznék a Bézier-görbe előállítását, meghatározzunk egy új fogalmat, a Bernstein-polinomot.

$$B_i^n(t) = \binom{n}{k} t^i (1-t)^i.$$

A fenti képletben az $\binom{n}{k}$ a binomiális együtthatót jelöli, azaz $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

A Bézier-görbe tuljadonképpen a kontrollpontok súlyozott átlagából állítható elő, a kontrollpontok számától függ, hogy hanyad fokú Bézier-görbéről beszélünk.

A Bézier-görbe általános felírása:

$$B(t) = \sum_{i=0}^{n} \boldsymbol{p}_i B_i^n(t),$$

ahol $B_i^n(t)$ a Bernstein-féle polinomot jelöli, $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n$, pedig a konrollpontokat.

Elsőfokú Bézier-görbe

A fenti képlet alapján megállapíthatjuk, hogy két kontrollpont esetén a görbe a következőképpen áll elő:

$$B(t) = \sum_{i=0}^{1} \boldsymbol{p}_i B_i^n(t) \Rightarrow B(t) = \boldsymbol{p}_0 B_0^n(t) + \boldsymbol{p}_1 B_1^n(t) \Rightarrow B(t) = \boldsymbol{p}_0 \binom{1}{0}(1-t) + \boldsymbol{p}_1 \binom{1}{1} t \Rightarrow$$

$$B(t) = \boldsymbol{p}_0(1-t) + \boldsymbol{p}_1 t \Rightarrow B(t) = \boldsymbol{p}_0(1-t) + t(\boldsymbol{p}_1 - \boldsymbol{p}_0), t \in [0,1],$$

ami pont megfelel a lineáris interpolációnak, azaz az elsőfokú (lineáris) Bézier-görbe egybeesik a két pontot összekötő szakasszal.

Másodfokú Bézier-görbe

Az előzőekhez hasonlóan lehet levezetni a másodfokú (kvadratikus) Bézier-görbe képletét is, ami a következő:

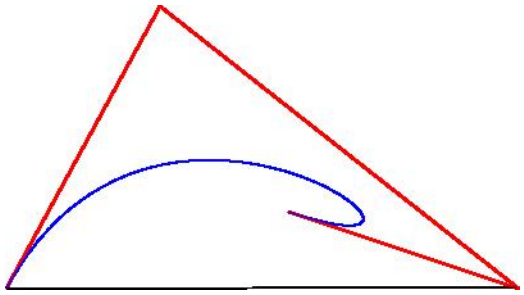
$$B(t) = (1-t)^2 \boldsymbol{p}_0 + 2t(1-t)\boldsymbol{p}_1 + t^2 \boldsymbol{p}_2.$$

Harmadfokú Bézier-görbe

Amit még külön szükséges kiemelni, az a harmadfokú (köbös) Beziér-görbe, mert a harmadfokú görbék több olyan tulajdonsággal is rendelkeznek, ami miatt a grafikában kényelmes az alkalmazásuk (pl. lehet inflexiós pontjuk, ha a négy kontrollpont nincs egy síkban, akkor a görbe térgörbe lesz).

A képletet most is hasonlóan nyerhetjük az általános formulából:

$$B(t) = (1-t)^3\boldsymbol{p}_0 + 3t(1-t)^2\boldsymbol{p}_1 + 3t^2(1-t)\boldsymbol{p}_2 + (1-t)^3\boldsymbol{p}_3.$$



3.2. ábra. A Bézier-görbe a kontrollpontok konvex burkán belül halad

Bézier-görbék előállítása de Casteljau-algoritmussal

Bázisfüggvénynek nevezzük azokat a súlyokat, melyekkel az egyes kontrollpontok részt vesznek a görbe alakításában. Bézier-görbéknél a bázisfüggvény a Bernstein-polinomból adódik. A bázisfüggvényekről elmondhatjuk, hogy összegük 1, ami abból következik, hogy gyakorlatilag a binomiális együtthatókat tartalmazzák. Mivelhogy a bázisfüggvényekre $t = 0$ esetén $B_0^n(0)$ és $t = 1$ esetén $B_n^n(1)$, valamint ebben az esetekben a bázisfüggvény értéke a többi kontrollponton 0, így a Bézier-függvény átmegy az első és utolsó kontrollponton.

A binomiális együtthatók tulajdonságaiból származik, hogy

$$B_{i+1}^m(t) = tB_i^{m-1}(t) + (1-t)B_{i+1}^{m-1}(t)$$

A jobb oldalt alakítva:

$$t\binom{m-1}{i}t^i(1-t)^{m-i-1} + (1-t)\binom{m-1}{i+1}t^{i+1}(1-t)^{m-i-2} =$$
$$\left(\binom{m-1}{i} + \binom{m-1}{i+1}\right)t^{i+1}(1-t)^{m-i-1} =$$
$$\binom{m}{i+1}t^{i+1}(1-t)^{m-i-1} = B_{i+1}^m(t)$$

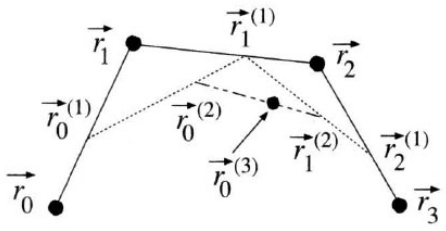
A fentieken alapszik a Bézier-görbék de Casteljau-módszerrel történő ábrázolása. Ha adott n kontrollpont, azokat $n-1$ szakasszal tudjuk összekötni. Minden összekötő szakaszon meghatározzuk a t paraméter-értékhez tartozó osztópontot. Ezek a pontok a kontrollpont-párokhoz tartozó elsőfokú

Bézier-görbe-pontnak felelnek meg. A kapott pontokat kontrollpontként felhasználva, ismét meghatározhatjuk a t paraméterhez tartozó elsőfokú Bézier-görbe-pontokat. Ekkor egy pontra már 3 kontrollpont van hatással (az eredetiek közül), és a most kapott pontok az eredeti kontrollpontokból alkotott (egymás melletti) ponthármasokhoz tartozó másodfokú Bézier-görbére illeszkednek.

Az eljárást addig folytatva, amíg az osztópontokból már nem tudunk új szakaszt, illetve új osztópontot beveztni, megkapjuk az eredeti n kontrolponthoz tartozó, $n-1$ fokszámú Bézier-görbéhez és t paraméterhez tartozó görbe-pontot. A t paramétert futtatva a $[0,1]$ intervallumon pedig az egész görbét.

Az algoritmus:

```
function deCasteljau(i,j)
{
  if i = 0 then
    return P(0,j)
  else
    return (1-u)* deCasteljau(i-1,j) + u* deCasteljau(i-1,j+1)
}
```



3.3. ábra. A de Casteljau-algoritmus ($n = 4$)

A Bézier-görbe tulajdonságai

- Affin invariancia
- Mindig a kontorllpontok által meghatározott konvex burkon belül halad
- Áthalad a végpontokon

3.6. B-spline görbék

Az előzőekben több módszert is tekintettünk, amivel adott pontokhoz görbét tudunk előállítani. A Bézier-görbével sajnos nem lesz tökéletes megoldás. Ennek oka, hogy a görbe erősen approximációs jellegű, illetve lokálisan nem vezérelhető. Ezek sajnos egymásnak ellentmondó követelmények, de a b-spline próbál megfelelő kompromisszumot kötni a két elvárás között.

A b-spline görbe szintén approximáló jellegű, tehát nem feltétlenül halad át a kontrollpontokon, de jobban befolyásolhatjuk a görbe alakját, és kevesebb kontrollpont megadása lesz szükséges.

A b-spline bázisfüggvény

Legyenek t_i skalárok úgy, hogy $t_i \leq t_{i+1},\ t_i \in \mathbb{R},\ i = 1, \ldots, n+k$.

$$N_1^i(t) = \begin{cases} 1, & \text{ha } t_i \leq t < t_{i+1}; \\ 0, & \text{egyébként} \end{cases}$$

$$N_i^k(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_i^{k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1}^{k-1}(t).$$

A fenti rekurzív függvényt normalizált b-spline bázisfüggvénynek, a t_i skalárokat pedig csomóértékeknek nevezzük. Szükséges még hozzátenni, hogy a képeletben $\frac{0}{0}$ is előfordulhat, ezeket 0-nak kell tekinteni. Most, hogy a bázisfüggvények adottak, b-spline felírása hasonlóan történik, mint a Bézier-görbénél, tehát a k-ad rendű b-spline:

$$S(t) = \sum_{i=0}^{n} \boldsymbol{p}_i N_i^k(t), \quad \text{ahol} \quad 2 \leq k \leq n+1$$

$$t_0, \ldots, t_n, t_{n+1}, \ldots, t_{n+k-1}$$

csomópont értékek mellet.

Egy k-ad rendű b-spline bázisfüggvényei $k-1$-ed fokú polinomok, és ezek a polinomok k darab egymás melletti intervallumra vonatkoznak. Első fokú bázisfüggvények esetén a töröttvonalat kapjuk eredményül, és így a görbe lokálisan teljesen vezérelhető, valamint interpolációs jellegű. Magasabb fokú bázisfüggvények esetén a görbe kevésbé lesz lokálisan vezérelhető, illetve elveszíti interpolációs jellegét. Az interpolációs jelleget kierőszakolhatjuk, ha a hozzátartozó csomóértékek távolságát 0-nak választjuk, azaz másodfokú bázis esetén egy, harmadfokú bázis esetén két egymást követő intervallum hosszát kell 0-nak választani.

A b-spline görbe tulajdonságai

- Mint már említettük a b-spline görbe lokálisan változtatható, azaz valamely kontrollpont megváltoztatása nem feltétlenül lesz hatással az teljes görbére.
- A b-spline görbe tartalmazhat egyenes szakaszt, akkor is, ha nem minden kontrollpontja esik egy egyenesre.
- Egy k-ad rendű b-spline görbe bármely pontja legfeljebb k darab kontrollpontjának konvex burkában van. A Bézier-görbével ellentétben a görbe itt konvex burkok uniójában halad, amely része az összes kontrollpont konvex burkának.
- k-ad rendű b-spline esetén, ha egy kontrollpont „k-szoros", azaz $\boldsymbol{p}_j = \cdots = \boldsymbol{p}_{j+k-1}$, akkor a görbe áthalad az adott ponton.
- k-ad rendű b-spline esetén, „k-szoros" csomóérték esetén a görbe áthalad az adott csomóértékhez tartozó kontrollponton.
- Ugyanúgy mint a Bézier-görbénél, ha a görbére szeretnénk affin transzformációt végrehajtani, elegendő ezt csak a kontrollpontokon megtennünk.
- k-ad rendű b-spline esetén, ha $k = n + 1$ és a csomóértékekre: $t_0 = t_1 = \cdots = t_{k-1}, t_k, t_{k+1}, \cdots, t_n = t_{n+1} = \cdots = t_{n+k}$, akkor a görbe interpolálja az első és utolsó konrollpontot, és egybeesik a kontrollpontokhoz tartozó Bézier-görbével.

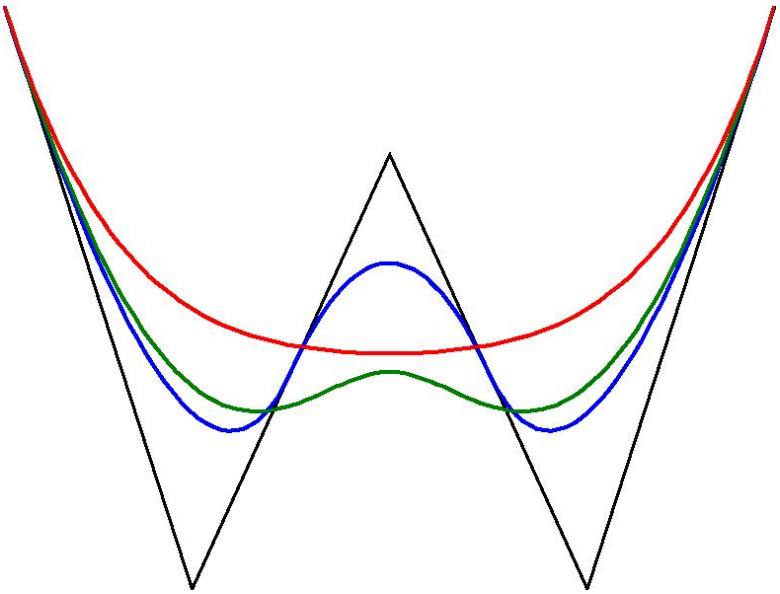
Mint ahogy az ábrán is látszik minél magasabb a b-spline görbe rendje, annál „simábban" viselkedik. 5 kontrollpont esetén a negyedrendű görbe aránylag sima (piros), míg a másodrendű elég változékony képet mutat (kék). Az ábrán szereplő b-spline görbék interpolálják az első és utolsó pontokat.

B-spline görbe előállítás Cox-de Boor algoritmussal

Eljutottunk oda, hogy polinom alakban meg tudjuk adni a b-spline görbét, de ez a számítás eléggé rosszul kondicionált, ezért megadunk lineáris műveleteket tartalmazó algoritmust, amely a Bézier-görbénél bemutatott de Casteljau-algoritmus egy általánosítása.

A definícióból

$$S(t) = \sum_{i=0}^{n} \boldsymbol{p}_i N_i^k(t) = \sum_{i=0}^{n} \frac{t - t_i}{t_{i+k-1} - t_i} \boldsymbol{p}_i N_i^{k-1}(t) + \sum_{i=0}^{n} \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} \boldsymbol{p}_i N_{i+1}^{k-1}(t)$$



3.4. ábra. Különböző rendű b-spline görbék

A második tagon indextranszformációt hajtunk végre, $i = i - 1$, ekkor

$$S(t) = \sum_{i=0}^{n+1} \frac{\boldsymbol{p}_i(t - t_i) + \boldsymbol{p}_{i-1}(t_{i+k-1} - t)}{t_{i+k-1} - t_i} N_i^{k-1}(t) =: \sum_{i=0}^{n} \boldsymbol{p}_i^{[1]} N_i^{k-1}(t),$$

ahol $\boldsymbol{d}_{-1} = 0$ és $\boldsymbol{d}_{n+1} = 0$. Az újraindexelést folytatva

$$S(t) = \sum_{i=0}^{n+j} \boldsymbol{p}_i^{[j]}(t) N_i^{k-j}(t),\ j = 1, \ldots, k-1.$$

Ha

$$\boldsymbol{p}_j^{[0]}(t_s) = \boldsymbol{p}_j\ (i = 0, \ldots, n),$$

akkor

$$\boldsymbol{p}_i^{[j]}(t) = (1 - \lambda)\boldsymbol{p}_{i-1}^{[j-1]}(t) + \lambda \boldsymbol{p}_i^{[j-1]}(t),\ j > 0$$

konvex lineáris kombináció, ahol

$$\lambda = \frac{t - t_i}{t_{i+k-j} - t_i}.$$

Ha az algoritmust folytatjuk addig, hogy $j = k - 1$, akkor megkapjuk az N_r^1 bázisfüggvényt, azaz $t \in [t_r, t_{r+1}]$ esetén a függvényértéket

$$S(t) = \boldsymbol{p}_r^{k-1}(t).$$

Egyenletes és nem egyenletes B-spline

Eddig a nem egyenletes b-spline görbéről (NUBS – Non-Uniform B-Spline) volt szó, mert nem követeltük meg a csomóértékek egyenletességét. Amennyiben megköveteljük az egyenletességet, akkor a rekurzív definíciót egyszerűbb alakban is fel lehet írni:

$$N_i^k(t) = \frac{t-i}{k-1} N_i^{k-1}(t) + \frac{i+k-t}{k-1} N_{i+1}^{k-1}(t).$$

Az egyenletesség megkövetelése abban áll, hogy a csomóvektor tagjaira igaz a következő:

$$t_{i+1} - t_i = c,$$

ahol a c konstans.

A végezetül még bemutatom a b-spline alapfüggvény kódját, melyet a b-spline és NURBS görbék előállításánál is használtam:

```
double N(int i, int k, double t) {
double Eq1, Eq, Den1, Den2;
if (k == 1) {
  if (this.knotVector[i] <= t && t < this.knotVector[i+1]) {
   return 1;
  } else {
   return 0;
  }
}
Den1 = this.knotVector[i+k-1] - this.knotVector[i];
if (dt > 0.000001) {
  Eq1 = this.B(i, k-1, t) / dt;
} else {
Eq1 = 0;
  }
  Den2 = this.knotVector[i+k] - this.knotVector[i+1];
  if (dt > 0.000001) {
   Eq2 = this.B(i+1, k-1, t) / dt;
  } else {
```

```
   Eq2 = 0;
  }
  return ((t - this.knotVector[i]) * Eq1 +
   (this.knotVector[i+k] - t) * Eq2);
}
```

3.7. NURBS görbék

A NURBS (Non-Uniform Rational B-Spline) görbéket a b-spline továbbfejlesztésével hozták létre. Az $i.$ kontrollpont hatását a b-spline bázisfüggvény aktuális paraméter melletti értéke és a kontrollpont saját súlytényezőjének a szorzata adja meg. Tehát annyi újdonság van a b-spline-hoz képest, hogy bevezetünk egy w_i súlyfüggvényt a kontrollpontokhoz. A NURBS bázisfüggvényét a következő módon írhatjuk fel:

$$R_i^k(t) = \frac{N_i^k(t)w_i}{\sum_{j=0}^{n} N_j^k(t)w_i}.$$

Ennek segítségével a NURBS görbék felírása:

$$C(t) = \sum_{i=0}^{n} \boldsymbol{p}_i R_i^k(t),$$

ami a k-ad fokú NURBS görbe definíciója, ahol $\boldsymbol{p}_i,\ (i = 0, \ldots, n)$ a kontrollpontok, $(t_0, \ldots, t_n)$ a csomóértékekből alkotott csomóvektor és $w_i,\ (i = 0, \ldots, n)$ a súlyok.

Mivel a B-spline bázisfüggvények polinomok, ezért a NURBS bázisfüggvények két polinom hányadosaként (racionális függvény) írhatók fel, innen jön a névben szereplő „R” betű.

NURBS előállítása b-spline-ból homogén koordinátákkal

A b-spline-ban szereplő kontrollpontokat írjuk fel oszlopvektorként, homogén koordinátákkal, a negyedik koordináta legyen 1:

$$\boldsymbol{p}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

Ha megszorozzuk a koordinátákat egy konstanssal (homogén koordináták esetén), akkor a pont nem változtatja meg a pozícióját. Szorozzuk meg a $\boldsymbol{p}_i$ koordinátáit w_i súllyal

$$\boldsymbol{p}_i^w = \begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i z_i \\ w_i \end{bmatrix}$$

Tehát $\boldsymbol{p}_i$ és $\boldsymbol{p}_i^w$ ugyanazt a pontot jelölik. Helyettesítsük be most ezt a b-spline egyenletébe:

$$C^w(t) = \sum_{i=0}^{n} N_i^k(t)\boldsymbol{p}_i^w = \sum_{i=0}^{n} N_i^k(t) \begin{bmatrix} w_i x_i \\ w_i y_i \\ w_i z_i \\ w_i \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} N_i^k(t) w_i x_i \\ \sum_{i=0}^{n} N_i^k(t) w_i y_i \\ \sum_{i=0}^{n} N_i^k(t) w_i z_i \\ \sum_{i=0}^{n} N_i^k(t) w_i \end{bmatrix}$$

A fenti egyenlet tehát a b-spline egyenlete homogén-koordinátás alakban. Most térjünk vissza Descartes-féle koordinátákra, vagyis osszuk be a koordinátákat a negyedikkel:

$$C(t) = \begin{bmatrix} \frac{\sum_{i=0}^{n} N_i^k(t) w_i x_i}{\sum_{i=0}^{n} N_i^k(t) w_i} \\ \frac{\sum_{i=0}^{n} N_i^k(t) w_i y_i}{\sum_{i=0}^{n} N_i^k(t) w_i} \\ \frac{\sum_{i=0}^{n} N_i^k(t) w_i z_i}{\sum_{i=0}^{n} N_i^k(t) w_i} \\ 1 \end{bmatrix} = \sum_{i=0}^{n} \frac{N_i^k(t) w_i}{\sum_{j=0}^{n} N_j^k(t) w_j} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

A kapott egyenletet átrendezve

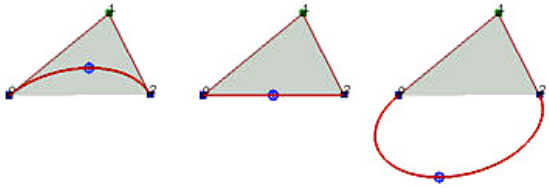
$$C(t) = \frac{1}{\sum_{i=0}^{n} N_i^k(t) w_i} \sum_{i=0}^{n} N_i^k(t) w_i \boldsymbol{p}_i$$

Fontos megjegyeznünk, hogy a w_i súlyok alapesetben pozitívak, viszont ettől eltérő eset is előfordulhat. Pl. ha a w_i súly 0, akkor a $\boldsymbol{p}_i$ kontrollpont együtthatója 0, így ez a pont nem lesz hatással $C(t)$-re, semmilyen t esetén.

A NURBS görbe tulajdonságai

- $C(t)$ NURBS görbe egy szakaszos görbe, melynek minden komponense k-ad fokú racionális görbe (racionális B-spline görbe).
- Az összekapcsolt NURBS görbék áthaladnak az első és utolsó kontrollponton.
- Ha $w_i = c,\ \forall i$, ahol c egy nem nulla konstans, akkor $R_i^k(t) = N_i^k(t)$, azaz a NURBS és B-spline bázisfüggvények megegyznek. Ebből is látszik tehát, hogy a B-spline egy speciális NURBS.

– Erős konvex burok tulajdonság: a görbe a kontrollpontjainak erős konvek burkában fut. Továbbá, ha $t \in [t_i, t_{i+1})$ csomóérték-tartománynak, akkor $C(t)$ a $\boldsymbol{p}_{i-k}, \ldots, \boldsymbol{p}_i$ kontrollpontok konvex burkában marad, ha a súlyok nemnegatívak. Ha negatív súlyok is előfordulhatnak, akkor a konvex-burok tulajdonságai a görbének nem lesznek érvényesek.

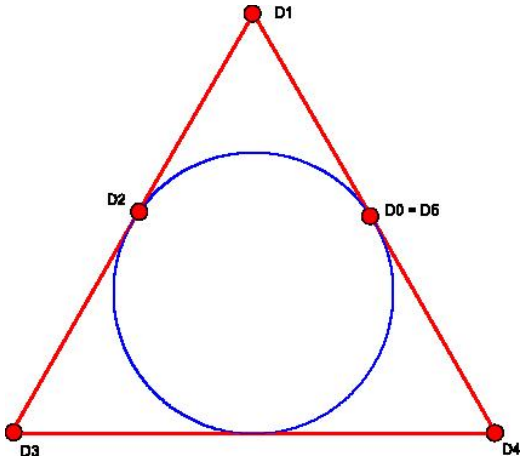


3.5. ábra. NURBS görbék, ha a súly pozitív, nulla vagy negatív

– A görbe lokálisan változtatató, azaz megváltoztatva a $\boldsymbol{p}_i$ pont helyzetét csak a görbe $[t_i, \ldots, t_{i+k+1})$ tartományban lévő szakaszára lesz hatással.

– A NURBS görbére teljesül a projektív invariancia tulajdonság, azaz nem csak affin, hanem projektív transzformációk esetén is elegendő a kontrollpontokon végrehajtani az adott transzformációt.

– NURBS görbével előállíthatók a másodrendű görbék, míg Bézier, vagy B-spline görbékkel csak közelíteni lehet őket.



3.6. ábra. Kör előállítása NURBS-görbével

Az ábrán a kör előállítását mutatom be. A kontrollpontok az adottak, a súlyok rendre: 1, 1/2, 1, 1/4, 1/4, 1; a csomóértékek pedig rendre: 0, 0, 1/3, 1/3, 2/3, 1, 1. Ez az előállítási mód majd a forgásfelületek előállításánál fog szerepet játszani.

4. fejezet

Felületek

Ezennel el is érkeztünk a dolgozat fő témájához, a felületek előállításának tárgyalásához. Az előző fejezetben a görbék konstruálását taglaltuk, ahol a paraméter-tartomány egy dimenziós volt, és a paraméter-tartománybeli elemekhez rendeltünk három dimenziós pontokat, így állt elő a görbe. A felületeknél hasonló lesz a paraméteres előállítás menete, a fő változás, hogy a paraméter-tartomány ezesetben két dimenziós lesz.

Mielőtt azonban belevágnánk, szükséges megemlíteni, hogy a felületeket nem csak paraméteres egyenlet segítségével állíthatjuk elő, hanem implicit egyenletekkel is. Például az R sugarú, (x_0, y_0, z_0) középpontú gömbfelület implicit egyenlete:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - R^2 = 0$$

Ezt a formát akkor szerencsés használni, ha egy pontról el akarjuk dönteni hogy az adott felületre illeszkedik-e vagy sem. Ilyenkor csak be kell helyettesíteni a pont koordinátáit az egyenletbe és ha az egyenlőség igaz, akkor illeszkedik a pont, ellenkező esetben pedig nem.

4.1. Paraméteres egyenlet

Abban az esetben viszont, amikor nekünk kell előállítani a felület pontjait, szerencsésebb az paraméteres egyenletet használni, ilyenkor az adott paraméter-értékeket behelyettesíve, az ahhoz tartozó felületpontot kapjuk eredményül. Az előbbi példánál maradva – R sugarú, (x_0, y_0, z_0) középpontú gömbfelület –, a következő paraméteres egyenletekkel írható fel:

$$x = x_0 + R \cdot \cos(2\pi)u \cdot \sin(\pi)v; \quad y = y_0 + R \cdot \sin(2\pi)u \cdot \sin(\pi)v; \quad z = z_0 + R \cdot \cos(\pi)v,$$

vagy vektorosan

$$\boldsymbol{r}(u,v) = \begin{pmatrix} x_0 + R \cdot \cos 2\pi u \cdot \sin \pi v \\ y_0 + R \cdot \sin 2\pi u \cdot \sin \pi v \\ z_0 + R \cdot \cos \pi v \end{pmatrix},$$

ahol u, v a paraméterek, azaz a görbékhez hasonlóan a paraméteres előállítás a felületeknél is skalár-vektor függvény, amley a kétdimenziós skalár-térből képez a háromdimenziós vektorok terébe.

Tenzori szorzat

Az

$$s(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} \boldsymbol{P}_{i,j} F_i(u) G_j(v)$$

felületet tenzori szorzattal előállított felületnek nevezzük, ahol $F_i(u)$ és $G_j(v)$ különböző alapfüggvények, például Bézier-felület esetén a Bernstein polinom, de lehetnek akár Lagrange, vagy racionális b-spline alapfüggvények is.

4.2. Coons-folt

Adott négy egymást páronként metsző térgörbe paraméteres egyenlete (térbeli négyszög). A Coons-folt az a felület lesz, melyet ez a négy görbe határol.

Nézzük meg, hogy hogyan is történik pontosan a felület előállítása. Legyenek adottak a következő görbék:

$$\boldsymbol{a}_1(u),\ \boldsymbol{a}_2(u),\ \boldsymbol{b}_1(v),\ \boldsymbol{b}_2(v),\ \text{ahol} \quad u, v \in [0,1].$$

Mivel a Coons-folt határai ezek a határológörbék lesznek, ezért az $\boldsymbol{r}(u,v)$ felületről már elmondhatjuk, hogy

$$\begin{aligned} \boldsymbol{r}(u,0) &= \boldsymbol{a}_1(u) \\ \boldsymbol{r}(u,1) &= \boldsymbol{a}_2(u) \\ \boldsymbol{r}(0,v) &= \boldsymbol{b}_1(v) \\ \boldsymbol{r}(1,v) &= \boldsymbol{b}_2(v). \end{aligned}$$

A Coons-foltot három felületből állítjuk elő, ebből kettő az $\boldsymbol{a}_1$ és $\boldsymbol{a}_2$, illetve $\boldsymbol{b}_2$ és $\boldsymbol{b}_2$ által alkotott vonalfelületek, a harmadik pedig egy nyeregfelület.

A vonalfelületek előállítása a következő módon történik: a határológörbék azonos paraméterértékhez tartozó pontjait kötjük össze egyenes szakaszokkal (ezek lesznek a paramétervonalak). Ezen felület kiszámítása

$$\boldsymbol{r}_a(u,v) = (1-v)\boldsymbol{a}_1(u) + v\boldsymbol{a}_2(u) = (1-v)\boldsymbol{r}(u,0) + v\boldsymbol{r}(u,1).$$

A másik két határológörbével hasonlóan felírhatjuk vonalfelületünket:

$$\boldsymbol{r}_b(u,v) = (1-u)\boldsymbol{b}_1(v) + u\boldsymbol{b}_2(v) = (1-u)\boldsymbol{r}(0,v) + u\boldsymbol{r}(1,v).$$

A fenti két felület a határológörbék pontjainak lineáris kombinációjából álltak elő. A négy görbe által meghatározott nyeregfelületet pedig a következő módon adhatjuk meg mátrix-alakban:

$$\boldsymbol{r}_{ab}(u,v) = \begin{pmatrix} 1-u & u \end{pmatrix} \begin{pmatrix} \boldsymbol{r}(0,0) & \boldsymbol{r}(0,1) \\ \boldsymbol{r}(1,0) & \boldsymbol{r}(1,1) \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}$$

kibontva

$$\boldsymbol{r}_{ab}(u,v) = (1-u)(1-v)\boldsymbol{r}(0,0) + (1-u)v\boldsymbol{r}(0,1) + u(1-v)\boldsymbol{r}(1,0) + uv\boldsymbol{r}(1,1).$$

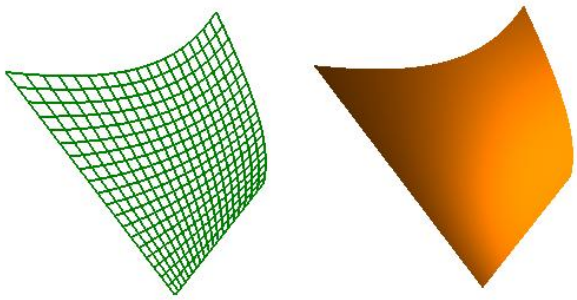
A három felületből az $\boldsymbol{r}(u,v) = \boldsymbol{r}_a(u,v) + \boldsymbol{r}_b(u,v) - \boldsymbol{r}_{ab}(u,v)$ képlet alapján kajuk meg a Coons-foltot, azaz

$$\begin{aligned} \boldsymbol{r}(u,v) = & \\ & (1-v)\boldsymbol{r}(u,0) + v\boldsymbol{r}(u,1) + (1-u)\boldsymbol{r}(0,v) + u\boldsymbol{r}(1,v) - \\ & - \Big((1-u)(1-v)\boldsymbol{r}(0,0) + (1-u)v\boldsymbol{r}(0,1) + u(1-v)\boldsymbol{r}(1,0) + uv\boldsymbol{r}(1,1)\Big), \end{aligned}$$

mátrix alakban pedig

$$\begin{aligned} \boldsymbol{r}(u,v) = \begin{pmatrix} 1-u & u \end{pmatrix} \begin{pmatrix} \boldsymbol{r}(0,v) \\ \boldsymbol{r}(1,v) \end{pmatrix} + \begin{pmatrix} \boldsymbol{r}(u,0) & \boldsymbol{r}(u,1) \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix} - \\ - \begin{pmatrix} 1-u & u \end{pmatrix} \begin{pmatrix} \boldsymbol{r}(0,0) & \boldsymbol{r}(0,1) \\ \boldsymbol{r}(1,0) & \boldsymbol{r}(1,1) \end{pmatrix} \begin{pmatrix} 1-v \\ v \end{pmatrix}. \end{aligned}$$

Ezek alapján, ha meg vannak adva a határológörbék, és azok megfelenek a feltételeknek, akkor elő tudjuk állítani a hozzá tartozó Coons-foltot. Megjegyzem, hogy az egymással „szemközti” görbéknek,



4.1. ábra. Coons-folt

ugyanazzal a paraméter-intervallummal kell rendelkezniük, viszont nem feltétlenül kell egybeesnie a $[0,1]$ intervallummal, ha nem esik egybe, akkor a fenti képletben meg kell változtatni a megfelelő határokat is.

Az ábrán látható Coons-folt esetén két (egymás melletti) határológörbe egyenes, a másik kettő pedig parabola.

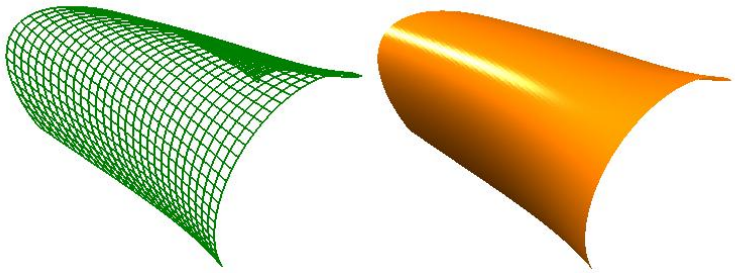
4.3. Bikubikus felületek

A bikubikus interpoláció tulajdonképpen a harmadfokú interpoláció továbbgondoloása. A harmadfokú görbéket általánosan a $\boldsymbol{r}(t) = \sum_{i=0}^{3} \boldsymbol{p}_i B_i(t)$ összefüggésel tudjuk leírni, ahol $\boldsymbol{p}_i$ az interpolációs, vagy kontrollpont, $B_i(t)$ bedig a bázisfüggvény értéke a t paraméterérték mellett. Harmadfokú görbéknél a $B_i(t)$ t-ben harmadfokú függvény. Ha ki akarjuk terjeszteni két dimenziósra a paramétertartományt, azt a következő módon tehetjük meg:

$$\boldsymbol{r}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} A_{ij}(u,v)\boldsymbol{p},$$

kibontva

$$\begin{aligned}\boldsymbol{r}(u,v) = {} & a_{00} + a_{10}u + a_{01}v + a_{20}u^2 + a_{11}uv + a_{02}v^2 + \\ & + a_{21}u^2v + a_{12}uv^2 + a_{22}u^2v^2 + a_{30}u^3 + a_{03}v^3 + a_{31}u^3v + \\ & + a_{13}uv^3 + a_{32}u^3v^2 + a_{23}u^2v^3 + a_{33}u^3v^3.\end{aligned}$$



4.2. ábra. Bikubikus felület

4.4. Bézier-felület

A Bézier-felületeket a szokásos módon határozzuk meg. Legyenek adottak a kontrollpontok az egyik dimenzió mentén $n+1$ darab, a másik dimenzió mentén pedig $m+1$. Ezt a kontrollhálónak, vagy Bézier-hálónak nevezzük. Egy adott kontrollpontra hivatkozzunk $\boldsymbol{P}_{i,j}$ alakban, ahol i, j rendre a sor- és oszlopindexeket jelölik. Ekkor az $s(u,v)$ Bézier-felület:

$$s(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(u) B_j^n(v) \boldsymbol{P}_{i,j},$$

ahol B_i^m, B_j^n rendre az $n.$ és $m.$ fokú Beziér-görbék, $i.$ és $j.$ alapfüggvényei az u és v paraméter-értékek mellett. Részletesen:

$$B_i^n(t) = \binom{n}{k} t^i (1-t)^i.$$

A kontrollpontok mátrixba rendezésével a Bézier-felület mátrixos alakját kapjuk.

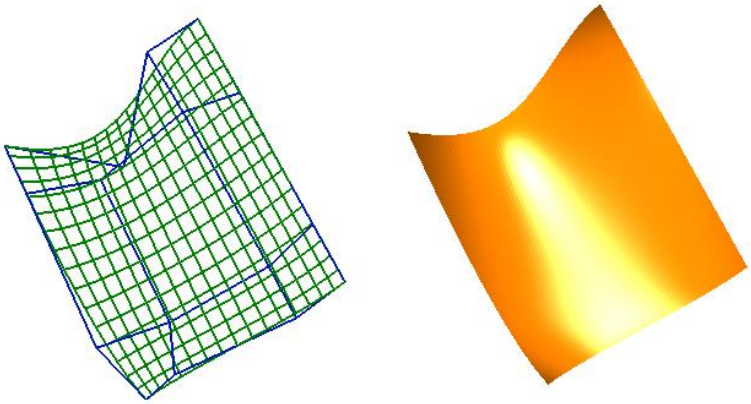
$$s(u,v) = [B_{m,0}(u), B_{m,1}(u), \cdots, B_{m,m}(u)] \begin{bmatrix} \boldsymbol{P}_{0,0} & \boldsymbol{P}_{0,1} & \cdots & \boldsymbol{P}_{0,m} \\ \boldsymbol{P}_{1,0} & \boldsymbol{P}_{1,1} & \cdots & \boldsymbol{P}_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{P}_{n,0} & \boldsymbol{P}_{n,1} & \cdots & \boldsymbol{P}_{n,m} \end{bmatrix} \begin{bmatrix} B_{n,0}(v) \\ B_{n,1}(v) \\ \vdots \\ B_{n,n}(v) \end{bmatrix}.$$

A Bézier-felület tulajdonságai

– A felület interpolálja a sarkok pontjait.

– A felület a kontrollháló által meghatározott konvex burokban halad.

– Igaz lesz az affin invariancia, azaz, ha affin transzformációt szeretnénk a görbére elkalmazni, akkor azt elegendő megtenni a kontrollpontokon.



4.3. ábra. A Bézier-felület

4.5. B-spline felület

A b-spline felületet úgy képzelhetjük el, mintha egy b-spline görbét mozgatnánk úgy, hogy a görbe kontrollpontjai egyenként, egy-egy b-spline görbe mentén mozoknak. Ha ebből indulunk ki, akkor a következő módon adhatjunk meg a b-spline felület definícióját. Az i. kontrollpont mozogjon az a_i görbe mentén, a mozgó b-spline görbe pedig legyen a. Ekkor a-ra a következő írható fel:

$$a(u) = \sum_{i=0}^{n} a_i N_i^{k_1}(u),$$

ahol az a_i szintén b-spline görbét takar

$$a_i(v) = \sum_{j=0}^{m} \boldsymbol{P}_{i,j} N_j^{k_2}(v),$$

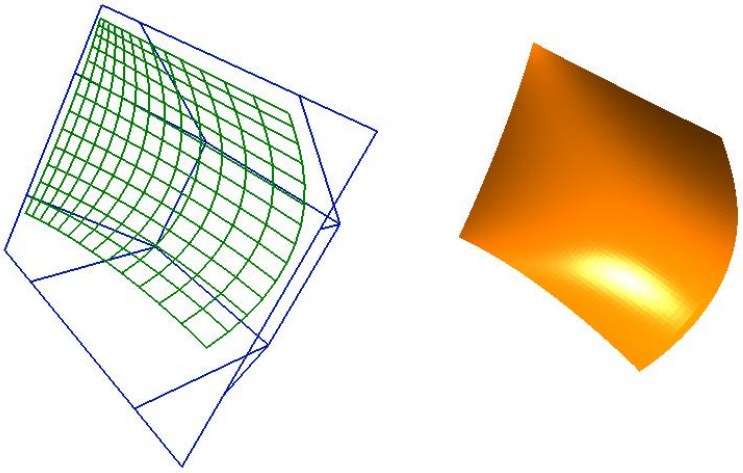
ahol $\boldsymbol{P}_{i,j}$ az i, j indexű kontrollpont koordinátáit takarja. A második képletet behelyettesítve az elsőbe megkapjuk a b-spline felület paraméteres egyenletét:

$$s(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} b_i j N_i^{k_1}(u) N_j^{k_2}(v)$$

Emlékeztetőül felírjuk a b-spline alapfüggvény képletét:

$$N_1^i(t) = \begin{cases} 1, & \text{ha } t_i \leq t < t_{i+1}; \\ 0, & \text{egyébként} \end{cases}$$

$$N_i^k(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_i^{k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1}^{k-1}(t).$$



4.4. ábra. A b-spline felület

A b-spline felület örökli a görbe tulajdonságait, tehát az így kapott felületről elmondhatjuk a következőket:

– A b-spline felület lokálisan változtatható.

– A felület belül marad a kontrollpontok úgynevezett erős konvex burkában.

– A felületre igaz az affin invariancia tulajdonság.

– Hasonlóan, mint a görbéknél, a Bézier felületet tekinthetjük speciális b-spline felületnek. Ezen speciális eset akkor áll elő, ha $n = k_1, m = k_2$ és a csomóvektor a következő alakú: $T = = [0, \ldots, 0, 1, \ldots, 1]$.

4.6. NURBS felület

A NURBS felület széles körben elterjedt, ennek oka, hogy nagyon finoman lehet befolyásolni az alakját, akár a konrollpontok áthelyezésével, akár a kontrollpontokhoz tartozó súlyok módosításával. A NURBS görbe lokálisan vezérelhető, ez a tulajdonság a felületre is öröklődik és így egy konrollpont

megváltoztatása nem lesz hatással az egész felületre, hanem csak egy részére, amely a felület fokszámaitól függ. A súlyok pedig „mágnesként” hatnak a kontrollpontokban, az adott súly növelésével vonzza a felületet, csökkentésével pedig taszítja az adott kontrollpontban. Például egy harmadfokú NURBS egy kontrollpont változtatása esetén a változás csak 16 tartományra terjed ki. Amennyiben mégsem lehet elég finoman befolyásolni, alakítani a felületet, fokszámnöveléssel növelhetjük a konrollpontok és csomóértékek számát, ezzel még finomabbá téve a felület alakíthatóságát. A vezérlőpontok egyenkénti áthelyezésénél sokkal szemléletesebb módszer a felületszobrászat, amely a vezérlőpontokat nem közvetlenül, hanem egy természetes formaalakító művelet beiktatásával változtatja.

A NURBS felületek megadásához szükségünk lesz az előző fejezetben tárgyalt NURBS görbe egyenletére, ami a következő

$$R_i^k(t) = \frac{N_i^k(t) w_i}{\sum_{j=0}^{n} N_j^k(t) w_i}.$$

Az u irányban p-ed fokú és a v irányban q-ad fokú NURBS felületet a következő racionális függvénnyel adjuk meg:

$$s(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u) N_j^q(v) w_{i,j} \boldsymbol{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_i^p(u) N_j^q(v) w_{i,j}},$$

ahol $\boldsymbol{P}_{i,j}$ a kontrollháló pontjait jelöli, $w_{i,j}$ a súlyokat, $N_i^p(u)$ és $N_j^q(v)$ pedig a nem racionális b-spline alapfüggvény a következő csomóértékekkel:

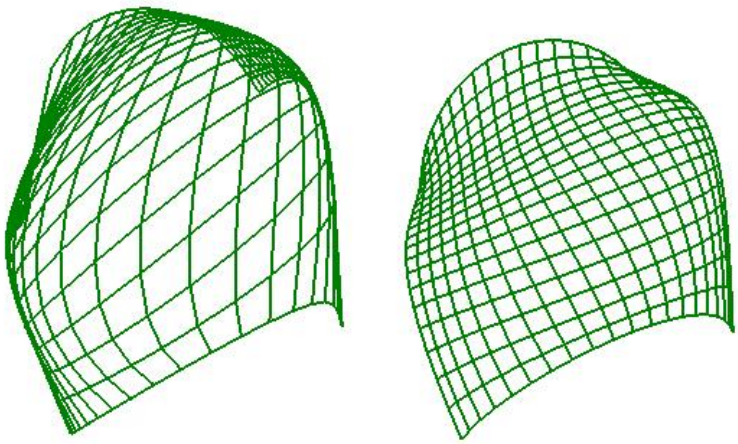
$$U = (\underbrace{0,\ldots,0}_{p+1}, u_{p+1},\ldots,u_{r-p-1},\underbrace{1,\ldots,1}_{p+1}) V = (\underbrace{0,\ldots,0}_{q+1}, v_{q+1},\ldots,v_{s-q-1},\underbrace{1,\ldots,1}_{q+1}),$$

ahol $r = n + p + 1$ és $s = m + q + 1$.

Forgásfelület előállítása

Hatalmas előnyként említendő, hogy NURBS felületként előállíthatók a másodrendű felületek és a forgásfelületek is. Tegyük fel, hogy egy forgásfelület meghatározó görbéje:

$$\boldsymbol{m}(v) = \sum_{i=0}^{n} \boldsymbol{d}_i \tilde{w}_i \frac{N_i^k(v)}{\sum_{r=0}^{n} \tilde{w}_r N_r^k(v)}, \quad v_1 = \cdots = v_{k-1}, v_k, \ldots, v_n, v_{n+1} = \cdots = v_{n+k-1}.$$



4.5. ábra. A NURBS felület súlyainak megváltoztaása

Ezt a görbét forgatjuk egy egyenes tengely körül. A forgás során a $\boldsymbol{d}_i$ pontok a forgástengelyre merőleges síkú körvonalak mentén fognak mozogni, amely szintén leírható NURBS görbével. A $\boldsymbol{d}_i$ által leírt kör racionális b-spline reprezentációja legyen:

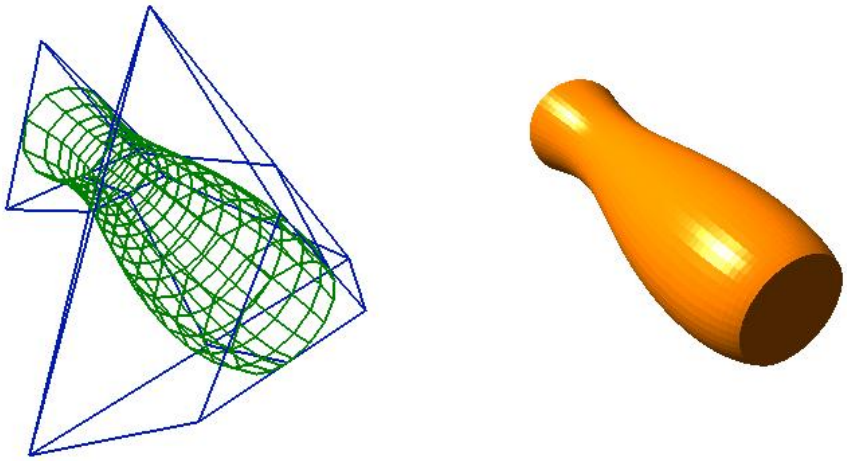
$$\boldsymbol{d}_i(u) = \sum_{j=0}^{5} \boldsymbol{p}_{ij} \hat{w}_j \frac{N_j^3(u)}{\sum_{r=0}^{5} \hat{w}_r N_r^3(u)}.$$

A fenti képletben a kontrollpontok és csomóértékek a NURBS görbénél bemutatott kör-előállításnak felenek meg. Eredményként előáll a forgásfelület NURBS reprezentációja:

$$s(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{5} \boldsymbol{p}_{ij} \tilde{w}_i \hat{w}_j \frac{N_i^k(v)}{\sum_{r=0}^{n} \tilde{w}_r N_r^k(v)} \frac{N_j^3(u)}{\sum_{r=0}^{5} \hat{w}_r N_r^3(u)},$$

ahol a $w_{ij} = \tilde{w}_i \hat{w}_j$ helyettesítést alkalmazva:

$$s(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{5} \boldsymbol{p}_{ij} w_{ij} \frac{N_i^k(v)}{\sum_{r=0}^{n} \tilde{w}_r N_r^k(v)} \frac{N_j^3(u)}{\sum_{r=0}^{5} \hat{w}_r N_r^3(u)}.$$



4.6. ábra. Forgásfelület előállítása NURBS-zel

5. fejezet

Felületek ábrázolása

Ebben a fejezetben a felületek ábrázolásával foglalkozunk. Az előző fejezetekben megismerkedtünk a görbék és a felületek matematikai egyenletek segítségével történő előállításával. Eljutottunk tehát addig, hogy adott paraméter érték(ek) esetén meg tudjuk határozni a görbe, illetve felület adott paraméter(ek)hez tartozó pontját. Most egy kicsit gyakorlati szempontból is körüljárjuk a problémát, azaz arra adunk módszereket, hogy egy adott felületet hogyan lehet megjeleníteni a képernyőn. A két utolsó részfejezetben két, a témához szorosan kapcsolódó technológiába nyerhetünk betekintést, a DirectX-be és az OpenGL-be.

Mindenek előtt azt a problémát kell megoldani, hogy egy háromdimenziós felületet hogyan jelenítsünk meg a kétdimenziós képernyőn. Erre a problémára nyújt megoldást a 2. fejezetben tárgyalt párhuzamos és centrális vetítés. Általában ez még nem jelent teljes körű megoldást, legtöbbször alkalmaznunk kell még forgatásokat és eltolásokat, ezeket szintén tárgyaltuk a második fejezetben. Ezen transzformációk elvégzése után a modellünket levetítettük a vetítési síkra, melynek mérete feltétlenül egyezik meg a képernyőnk felbontásával, így még alkalmaznunk kell egy bizonyos WindowToViewport transzformációt. Ez a következőképpen működik:

$$x_q = (x_p - Window.left)(View.width/Window.width) + View.left$$

$$y_q = (y_p - Window.top)(View.height/Window.height) + View.top$$

Ahol a Window a megjelenítendő ablakot jelöli, a View a képernyőnk ablaka, a tulajdonságaik pedig: left (bal oldal koordinátája), top (a felső rész koordinátája), width (szélesség), height (magasság). Ezek alapján már meg tudunk jeleníteni egy háromdimenziós pontot a képernyőn, a felületek ábrázolásához viszont ez még kevés lesz.

A háromdimenziós objektumok végtelen sok határolóponttal rendelkeznek, amelyeket nekünk ábrázolni kellene. Mivel a memória véges, ezért ez egy lehetetlen feladat. A felületek ábrázolásának

legegyszerűbb, valamint legkevésbé számolás igényes módja a drótvázmodell. Ennél az ábrázolási módszernél gyakorlatilag egy sematikus ábrát adunk az ábrázolandó felületről. Előállítása nagyon egyszerű, annyit kell mindössze tennünk, hogy mindkét paraméter mentén végighaladunk a paramétertartományon, az adott rácspontokban kiszámoljuk a paraméter-értékekhez tartozó felületi pontot, és ezeket szakaszokkal kötjük össze.

```
for (i=0; i<1; i+=h)
  for (j=v; j<1; j+=v)
    line(s(i, j-v), s(i,j));
```

A fenti kódban `line` utasítás a paraméterben megadott két pont között húz egy vonalat. Ezzel a kóddal az u irányú paramétervonalakat ábrázolhatjuk, általában azonban mindkét irányt szokták ábrázolni. A másik irány ábrázolása hasonló kóddal történhet:

```
for (j=0; j<1; j+=v)
  for (i=h; i<1; i+=h)
    line(s(i-h, j), s(i,j));
```

5.1. Poligonok

Ha megnézzük az ábrázolt drótvázmodelles felületünket, testünket, láthatjuk, hogy a kapott kép közel sem valószerű. Szebb eredmény érhetünk el, ha a megjelenítendő testet, felületet lapokból rakjuk össze. A lapok olyan sokszögeket jelentek, amelyek közelítik a felületet. Minél több lappal közelítjük a felületet, annál pontosabb lesz a közelítés és természetesen annál többet is kell számolni. Felületek ábrázolásakor a háromszög-lapokkal dolgozunk, mert három (nem egy egyenesre eső) pontra mindig illeszkedik egy sík, illetve az általános, háromdimenziós négyszög már képes a megcsavarodásra, azaz arra, egy adott nézőpontból a lap mindkét oldala látható. A háromszög-lapok eléréséhez elegendő a felület előállításához szükséges paraméter-téglalapot felosztani háromszögekre, és az ottani felület-pontokat összekötve háromszög-lapokat kapunk. A paraméter-téglalap felosztása: ha u irányban N, v irányban pedig M részre szeretnék osztani a paramétertartományt, akkor

$$[u_i, v_j] = \left[u_{min} + (u_{max} - u_{min})\frac{i}{N}, v_{min} + (v_{max} - v_{min})\frac{j}{M}\right]$$

és a háromszögek $s(u_i, v_j), s(u_{i+1}, v_j), s(u_i, v_{j+1})$, valamint $s(u_{i+1}, v_j), s(u_{i+1}, v_{j+1}), s(u_i, v_{j+1})$ lesznek. Ezt nevezik a paraméteres felület tesszelációjának. Nyilván ha a paraméterekre: $u, v \in [0, 1]$, akkor a fenti képlet így egyszerűsödik:

$$[u_i, v_j] = \left[\frac{i}{N}, \frac{j}{M}\right]$$

Ezzel eljutottunk oda, hogy a felületünk háromszög alakú sík lapokból áll. Ezeket ha egy színnel kitöltve ábrázoljuk a képernyőn, akkor csak egy nagy egyszínű foltot kapunk, nem tudunk különbséget tenni a lapok között, ezért valamilyen tulajdonság alapján eltérő színt kellene adni a lapoknak, valahogy valósághűbbé kellene tenni az ábrázolást.

5.2. Árnyalások

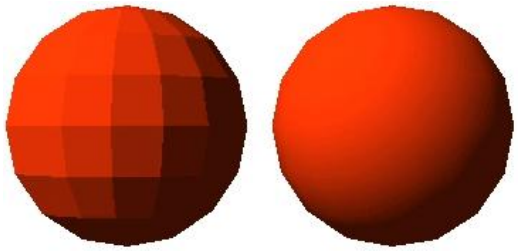
Ezekkel a felületekkel legtöbbször a valós – illetve egy valószerű – világot szeretnénk megjeleníteni. Világunkban az objektumok színnel rendelkeznek, az objektum anyagának a színétől, tulajdonságaitól és a fényektől (megvilágítástól) függően látjuk valamilyen színűnek a tárgyat. Éppen ezért a „virtuális” világunkban is ezek alapján határozzuk meg a színinformációkat. Az egyszerűség kedvéért a következőkben úgy tekintjük, mintha csak egy fényforrásunk lenne és a tér egy adott pontjában meg tudjuk határozni a fény beesési szögét.

Árnyalásokkal a megjelenítendő lapok színét befolyásolhatjuk. Különösen akkor látványos a használatuk, ha egy objektumot kevés lap segítségével szeretnénk megjeleníteni. Persze a felbontás növelésével tovább javíthatók az eredmények.

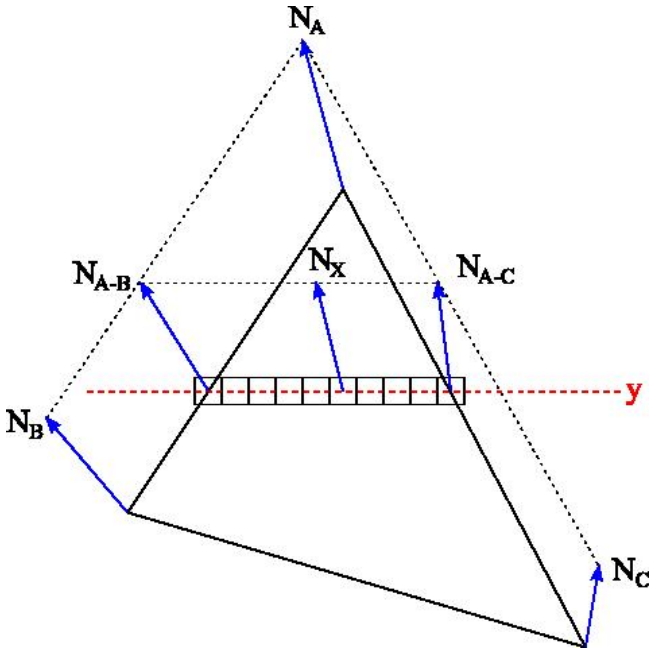
A legegyszerűbb módszer, ha vesszük a lap felületi normális vektorát, vagyis ennek a vektornak és a fényforrásnak a szögét, és ez alapján adjuk meg a lap színének intenzitását. Ahol ez a bezárt szög kisebb, ott kisebb intenzitást, ahol pedig nagyobb, ott nagyobbat. A laphoz tartozó minden képpont a kiszámolt intenzitás-értékkel fog szerepelni. Ezt a módszert nevezzük flat shading-nek, vagy konstans-árnyalásnak.

Bonyolultabb módszer, viszont látványos eredmény érhető el a Gouraud-féle árnyalással. Nevét Henri Gouraud-ról kapta, aki 1971-ben fejlesztette ki ezt a módszert. Ezzel a technikával változó intenzitást adhatunk a lapknak. Az intenzitást minden pixelben meghatározzuk. A csúcspontokban kiszámoljuk az intenzitást, a lapon belül pedig ezek lineáris kombinációjából határozzuk meg az adott pont intenzitását.

Még szebb eredmény érhető el a Phong-féle árnyalás alkalmazásával, amely nagyban hasonlít Gouraud módszeréhez, csak nem az intenzitást interpolálja, hanem a felületi normálvektort, és ez alapján (és a fénysugár szöge alapján) határozza meg minden egyes pontban az intenzitás-értékeket.



5.1. ábra. Különbség a konstans-árnyalás és Gouraud-árnyalás között



5.2. ábra. A Phong-árnyalás

5.3. Láthatóság

A fentiek alapján már elő tudjuk állítani a felülethez tartozó lapokat, árnyalni is tudjuk azokat, de ha egy adott pontból nézünk a felületre, akkor még nem biztos, hogy minden lapot látni fogunk. Ahhoz hogy helyesen jelenítsünk meg egy felületet, még ezt is hozzá kell venni, hogy a nézőpontból egy adott képernyőpixel helyén melyik lapot kell megjeleníteni.

Hátsó lapok eltávolítása

Általában egy felület esetén nem látjuk az összes lapot. Zárt felület esetén főleg nem. Ez a módszer arra hivatott, hogy amelyik lapot nem láthatjuk, azzal ne is foglalkozzunk és így spóroljunk az

erőforrásainkkal. A hátsó lapok eltávolításának alapja, hogy vesszük az adott (sík) laphoz tartozó felületi normális vektor és a nézőpontpól a lapba indított vektor által bezárt szöget. Ha ez a szög nem hegyesszög, akkor az adott lapnak a hátsó oldalát látjuk. Zárt testek ábrázolásánál nem láthatjuk egy lap hátsó oldalát, így ezekkel a lapokkal nem szükséges foglalkoznunk.

Z-puffer

Ha több objektumot szeretnénk ábrázolni a képernyőn, akkor lehet hasznos ez a módszer. Azon alapszik, hogy az objektumokat nem egyből a képernyőre, hanem egy – a képernyő méreteivel megegyező méretű – pufferba tesszük, és a puffer minden elemében tároljuk az adott pixelhez tartozó z-értéket. A puffer egy adott értéke csak akkor változhat, ha van egy lapunk, amely leképezve pontosan arra a képernyőpontra illeszkedik, és a pufferben eddig nem volt érték, vagy volt, de ez nagyobb volt mint amelyet most kellene beírni. A pontos színértéket csak abban az esetben határozzuk meg, ha a puffer érték változott. Így elkerülhetjük, hogy feleslegesen kelljen kiszámolni a nem megjelenítendő színértékeket.
Az algoritmus lépései:

– Az létrehozzuk a z-puffert (a képernyő minden pixeléhez), és minden elemét beállítjuk a maximális mélységre (amelynél távolabbi pontokat már nem veszünk figyelembe).

– Minden objektum minden határolófelületét pixelekké konvertálva, azoknak minden elemére:

  - Meghatározzuk a pixel mélységét (a kamera függőleges síkjától való távolságát)
  - Ha ez az érték kisebb, mint amit a mélységpufferban aktuálisan tárolunk, akkor a pixel mélységét az aktuális mélységre állítjuk és meghatározzuk a pont színét, amit a pixelhez rendelünk.

Sugárkövetés

Vegyük a képernyő pontjait, és mindegyikbe indítsunk a nézőpontból egy sugarat. Az adott képernyőpontban csak azt az objektumot kell ábrázolni, amelyiket ez a sugár először elmetszi. Ha sugár az adott pontban nem találkozik objektummal, akkor abban a képernyőpontban nem kell ábrázolni semmit.

Az alogritmus lépései:

– A képernyő minden pixeléhez meghatározzuk a nézőpontból a pixelen áthaladó félegyenest. Párhuzamos vetítésnél a képernyő síkjára merőleges félegyenesekkel dolgozunk.

– Ezekkel elmetsszük az ábrázolandó objektumokat.

– Minden pixelben meghatározzuk a nézőponthoz legközelebbi metszéspontot, az ehhez tartozó objektumot, majd ennek meghatározzuk a színét az adott pontban, végül kirajzoljuk a képernyőre.

A fenti két művelet elég időigényes, viszont sok objektum esetén többet nyerünk ezekkel vizsgálatokkal, mintha minden objektumot külön-külön ábrázolnánk.

5.4. OpenGL

A dolgozat témája nem az OpenGL, csak egy betekintést szeretnék nyújtani, mivel a témához szerintem szorosan kapcsolódik. Az előző fejezetekben megnéztük a görbék és felületek matematikai előállításának módszereit. Most egy picit belepillantunk a gyakorlati oldalba is. Az OpenGL a GLUT (OpenGL Utility Toolkit) függvénykönyvtárral kiegészítve leveszi a vállunkról a számítások fáradságos feladatát.

Az OpenGL-ről

Az OpenGL a Silicon Graphics által készített Graphics Library (GL) továbbfejlesztett, illetve szabványosított változata. A rendszer tulajdonképpen egy függvénykönyvtár, amely eredetileg a C nyelvhez készült, de napjainkban sok más programnyelvhez is elérhető.

Az OpenGL egy hardver-független interfész, amelyet sok hardver-platformra implementáltak, és főként a 3D-s képszintézist támogatja. Mivel az egyes grafikus kártyák eltérőek, ezért az OpenGL a kártya által nem támogatott funkciókat szoftveresen hajtja végre, amiből a felhasználó csak sebességbeli különbséget vehet észre.

Az OpenGL platformtól és alkalmazástól független rendszer, a pixelműveleteket igénylő képfeldolgozási feladatoktól az animációkon kersztül az igényes három dimenziós felületek megjelenítéséig sok dolgot támogat. A platformfüggetlenség megőrzése érdekében grafikus adatbevitelt, vagy ablakkezelést nem építettek a rendszerbe, ezeket kiegészítő függvénykönyvtárakkal lehet megoldani.

Az OpenGL szintaxisa

A C nyelvi változatot alapul véve mutatjuk be a parancsok szintaxisát. Egy parancs általánosan a következő módon épül fel:

```
glVertex{234}{sifd}[v]();
```

A fenti példában egy csúcspontot hozunk létre, utána megadjuk, hogy hány paraméter lesz, azoknak milyen a típusa, és a végén, hogy tömbként adjuk-e meg. A típusok a következők lehetnek: `b, s, i, f, d, ub, us, ui,` rendre 8, 16, 32 bites előjeles egész, 32, 64 bites lebegőpontos, 8, 16, 32 bites előjeles egész. Ez alapján egy helyes hívás:

```
glVertex3f(1.0,2.0,3.0);
```

Az OpenGL-ben vannak definiált konstansok, ezek neve nagybetűs, és GL-taggal kezdődik, valamint ha több szóból áll, akkor a szavak _ jellel kapcsolandónk össze. Pl.: `GL_TRIANGLE_STRIP`.

Állapotváltozók: a rendszerben számos globális paraméter, úgynevezett állapot- vagy globális változó van, melyek kurrens értéke szerint hajtja végre a rendszer a parancsokat. Ezeket az őket beállító paranccsal állíthatjuk be (pl. `glColor();`), vagy ha olyan jellegű a változó, hogy csak két értéke lehet, akkor a `glEnable();` paranccsal tudjuk engedélyezni, a `glDisable();` paranccsal letiltani, illetve a `glGet();` paranccsal lekérdezni az aktuális értéket. Az állapotváltozók rövidebbé teszik a kódot, valamint gyorsabbá a program végrehajtását.

Rajzolás

Az OpenGL alapból támogatja a Bézier-görbék és -felületek előállítását és megjelenítését. A görbék és felületek előállításához úgynevezett kiértékelőket (evaluators) használ a rendszer. Görbék előállításához először egy egydimenziós kiértékelést kell definiálnunk a `glMap1()` parancs segítségével. Paraméterként meg kell adni a Bézier-görbe rendjét, paramétertartományát, kontrollpontjait, valamint azt, hogy a kontrollpontok mit reprezentálnak, pl. a modelltérbeli pontot vagy színt. Ezután engedélyeznünk kell a megfelelő objektum kiértékelését a `glEnable()` paranccsal. Végül már csak le kell kérdezni adott paraméter-érték mellett az értéket a `glEvalCoord1()` paranccsal, amely egy vertexet hoz létre. A `glEvalCoord1()` a görbe egyetlen pontját határozza meg, viszont a `glMapGrid1()` függvénnyel létre tudjuk hozni a rácspontokat, a `glMeshEval1()` segítségével pedig az összes rácspontban kiértékeljük a függvényt.

A felületek megjelenítése nagyon hasonlít a görbékéhez, csak mindennek a kétdimenziós változatát kell használnunk, azaz létrehozunk kétdimenziós kiértékelést (`glMap2()`) és az adott paraméter-értékek mellett a `glEvalCoord2()` függvénnyel kapjuk meg a felület pontját, valamint ha egy rács mentén vagyunk kíváncsiak a felület-pontokra, akkor először fel kell osztanunk a paramétertartományt a `glMapGrid2()` függvénnyel és utána az megadott paraméterek mellett a `glEvalMesh2()` fügvénnyel kapjuk meg a felület pontjait. Utóbbinál első paraméterként megadhatjuk a módot, amely lehet `GL_POINT`, `GL_LINE`, `GL_FILL`, melyeknél rendre ponthálóval, poligonhálóval, vagy kitöltött poligonhálóval jeleníti meg a felületet. Fontos még megemlíteni, hogy ha a `GL_AUTO_NORMAL` engedélyezett, akkor a felületi normálist is meghatározza a rendszer.

A NURBS görbéket, illetve felületeket az OpenGL nem támogatja, viszont a GLUT (OpenGL Utility Toolkit) nevű könyvtár-kiegészítés igen. Nézzük meg, hogy a GLUT segítségével hogyan jeleníthetünk meg NURBS felületeket. NURBS objektumstruktúrát a `gluNewNurbsRenderer()` paranccsal tudunk létrehozni és az általa visszaadott címmel tudunk az objektumra hivatkozni, amelyet majd ha végeztünk, a `gluDeleteNurbsRenderer()` hívásával tudunk megszüntetni. A `gluNurbsProperty()` segítségével az objektum megjelenítésére vonatkozó beállításokat változtathatjuk meg. Ilyen beállítások közé tartozik például a `GLU_DISPLAY_MODE`, melynek értékei `GLU_FILL`, `GLU_OUTLINE_POLYGON` vagy `GLU_OUTLINE_PATCH` lehetnek. A teljesség igénye nélkül megadhatjuk itt még a mintavételezési módszert, a közelítés pontosságát, az osztáspontok számát, stb.

Egy adott felület megadását a `gluBeginSurface()` és a `gluEndSurface()` között adhatjuk meg, ahol paraméterként a felület azonosítóját kell átadnunk. A kettő között a `gluNurbsSurface()` hívásával adhajuk meg a felületet, melynek paraméterei közt meg kell adnunk a felület azonosítóját, az u és v irányú csomóértékek számát, és az őket tartalmazó tömböt, hogy az u és v irányú, egymást követő kontorllpont értékek között hány `GLfloat` érték található (milyen messze vannak egymástól a kontrollpontok tömbjében), természetesen a kontrollpontok tömbjét, a felület u- és v-beli rendjét, valamint a felület típusát, amely lehet nem racionális, illetve racionális b-spline, továbbá textúrakoordináták, vagy normálisok.

Amennyiben grafikával szeretnénk folglalkozni, az OpenGL nagyon jó választásnak tűnik, platformfüggetlensége, elterjedtsége miatt.

5.5. DirectX

A DirectX a Microsoft által Windows platformra készített API. Első verziója 1995-ben a DirectX 1.0 volt, jelenleg a legújabb verziója a 2008-ban megjelent DirectX 10.1. (Technikai okok miatt a DirectX 9.0 verziójával foglalkozunk.) A DirectX 8.1-es verziótól kezdve része a Windowsnak. Számos programozási nyelv támogatja, például a C++, C#, Delphi, Visual Basic, stb. Alkalmazása széles körben elterjedt, dinamikusan fejlődő technológia, melyet főként a játékprogramozásban használnak. A DirectX a multimédia programozást támogatja, ennek megfelelően több részből áll: DirectDraw, DirectX Graphics, DirectMusic, DirectSound, DirectInput, DirectSetup.

A dolgozat témájához csak a DirectDraw és a DirectX Graphics (mely magában foglalja a DirectDraw és a korábbi Direct3D funkcionalitását) kapcsolódik. A DirectDraw a két dimenziós megjelenítést támogatja, a DirectX Graphics pedig a háromdimenziós képszintézist. A D3DX (Direct3D Extensions) segédkönyvtár jelentősen legegyszerűsíti a DirectX Graphics programozását.

A DirectX API objektumai az ActiveX technológiát meglapozó COM (Component Object Model) objektumok, melyek funkcionalitását egy vagy több interfészen keresztül lehet elérni. A DirectX elfedi előlünk a harvereszközök sajátosságiait, így nem alakulhatnak ki kompatibilitási problémák. Ezt úgy valósítja meg, hogy a DirectX API-k egy egységes interfészen keresztül kommunikálnak a hardverrel, melynek neve HAL (Hardware Abstraction Layer). Azon funkciókat, melyeket a grafikus kártya nem valósít meg a REF (Reference Device) eszköz szoftveresen utánozza.

DirectX-ben az OpenGL-hez hasonlóan állapotokon keresztül tudjuk befolyásolni az objektumok megjelenítését. Az állapotokak három nagy csoportját különböztetjük meg, renderelést befolyásoló állapotok (render states), mintavételezési állapotok (sampler states) és textúra csatorna-keverést érintő állapotok (testure stage states). Ezeket a hozzájuk tartozó API hívásokkal tudjuk beállítani, pl. `IDirect3DDevice::SetRenderState()`.

A modellezés DirectX-ben is vertexek segítségével lehetséges. Mivel egy vertex nem csak a térbeli pozíciójával rendelkezik mint attribútummal, ezért lehetőségünk van az úgynevezett rugalmas vertexformátum (FVF, Flexible Vertex Format) használatára, amelyben magunk adhatjuk meg, hogy vertexeink milyen tulajdonságokkal rendelkeznek.Ilyen tulajdonságok lehetnek például a pontméret, pozíció, szín, stb. A létrehozott vertexe bekerülnek az úgynevezett vertex-pufferba, és innen lesznek kiolvasva megjelenítés előtt, a saját vertex-formátumunknak megfelelően.

A DirectX nyújt néhány beépített geometria primitívet: pont, különálló vonalak, folytonos szakaszok, különálló háromszögek, háromszögek kapcsolódó oldallal, háromszögek közös csúccsal. Ezek segítségével rakhatjuk össze objektumainkat. Az előző fejezetben leírtak alapján, a tárgyalt felületeket például kapcsolódó oldallal rendelkező háromszögekből rakhatjuk össze.

Amennyiben bonyolult modellel rendelkezünk, körülményes lenne csúcsokkal megadni azokat. Szerencsére számos 3D modellező program létezik és természetesen sok file-formátum is, melyeket azok használnak (pl.: MD3, 3D Studio ASC, POV-Ray, stb.). A Microsoft is rendelkezik ilyen formátummal, melyet X File-nak hívnak. Természetesen a DirectX rendelkezik azokkal az eszközökkel melyek kezelik ezt a formátumot, amelyeket az XFile osztály tartalmaz.

Összességében elmondhatjuk a DirectX-ről, hogy egy komplex technológiát ad a fejlesztő kezébe, amellyel nem csak grafikai, hanem egyéb multimédiás feladatok megvalósítása is egyszerűbbé válik.

6. fejezet

Melléklet

6.1. 1. melléklet

```
float N(float u, int i, int k, const float* Knots)
{
  if(k==1)
  {
    if( Knots[i] <= u && u <= Knots[i+1] ) {
      return 1.0f;
    }
    return 0.0f;
  }
  float Den1 = Knots[i+k-1] - Knots[i];
  float Den2 = Knots[i+k] - Knots[i+1];
  float Eq1=0,Eq2=0;
  if(Den1>0) {
    Eq1 = ((u-Knots[i]) / Den1) * N(u,i,k-1,Knots);
  }
  if(Den2>0) {
    Eq2 = (Knots[i+k]-u) / Den2 * N(u,i+1,k-1,Knots);
  }
  return Eq1+Eq2;
}
```

```
Point Calculate(float u, float v)
{
  Point result = {0, 0, 0};
  float d = 0.0;
  for (int i = 0; i != num_cvs_u; i++)
  {
    for (int j = 0; j != num_cvs_v; j++)
    {
      float temp = N(u, i, order_u, knots_u) *
       N(v, j, order_v, knots_v) * weights[i][j];
      result.x += temp * Points[i][j].x;
      result.y += temp * Points[i][j].y;
      result.z += temp * Points[i][j].z;

      d += temp;
    }
  }

  result.x /= d;
  result.y /= d;
  result.z /= d;
  return result;
}
```

A fenti kóddal NURBS felületet genrálhatunk. Input-ként meg kell adni:

– A kontrollpontok `Point` típusú kétdimenziós tömbjét (`Points`)

– A kontrollpontok számát u és v paraméterirányban (`num_cvs_u`, `num_cvs_v`)

– A csomóértékek vektorait (`knots_u`, `knots_v`)

– A felület fokát (`degree_u`, `degree_v`)

– A súlyok – a kontrollpontok tömbjével megegyező méretű – tömbjét (`weights`)

7. fejezet

Összegzés

Remélem munkámmal átfogó betekintést nyújthattam a téma iránt érdeklődőknek. Az itt leírtakkal – véleményem szerint – az érdeklődő használható dokumentációt kap a felületmodellezéshez. A téma nagyon sokrétű, az utolsó fejezetbe próbáltam az általam fontosabbnak vélt módszerekről és technológiákról említést tenni. Ezen fejezetet kiindulási alapnak szántam, inkább irányvonalként, hogy merre érdemes indulni az előző fejezetek elovasása, megértése után.

Gondolkoztam rajta, hogy az egyes felületek előtt közvetlen lehetett volna venni a görbéket, de ez a megoldás ésszerűbbnek tűnt, ha esetleg valaki csak a görbékre kíváncsi, vagy azokat ismeri, csak nem tudja, hogyan lesznek belőlük felületek, könyebben el tud igazodni így a dokumentumban. A másik ok, amiért ez a tagolást választottam, hogy számomra így követhetőbbnek tűnt, mi a különbség az egyes fajták között és hogyan lehet visszavezetni egyiket a másikra.

A diplomamunka elkészítése során, ahol szükségét éreztem, ábrákkal, illetve kódrészletekkel próbáltam segíteni a könnyebb megértést, szemléltetni a képletekkel vagy szövegesen leírt információkat.

A dolgozat „Görbék” című fejezetéhez készült ábrákat saját programmal készítettem. Köztes formtumként SVG fileformátumot használtam, egyszerű (XML-alapú) leprogramozása, és vektorgrafikus mivolta miatt.

A felületekhez tartozó ábrák szintén saját programmal készültek, C nyelvi környezetben. Az ábrák létrehozásakor felhasználtam az OpenGL lehetőségeit, a beépített transzformációkat, a kamera elhelyezést, a megvilágítási modellt. A felületek előállításához nem vettem igénybe az OpenGL beépített függvényeit, ezeket külön leprogramoztam, ezzel mintegy kipróbálva a dolgozatban leírt módszereket.

A megírás során próbáltam következetes lenni, ahol szügségét éreztem, visszautalást vagy ismétlést elhelyezni a szövegben.

A dolgozat végén felsorolt irodalomjegyzékben az Olvasó még jobban elmélyedhet témában és az itt nem szereplő témakörökben.

Végezetül szeretnék köszönetet mondani témavezetőmnek, Dr. Kovács Emődnek, a dolgozat megírásához nyújtott segítségéért, valamint az általa rendelkezésemre bocsájtott anyagokért.

Irodalomjegyzék

[1] Nyisztor Károly: *Garfika és játékprogramozás DirectX-szel*, SZAK kiadó, 2005

[2] Dr. Szirmay-Kalos László, Antal György, Csonka Ferenc: *Háromdimenziós grafika, animáció és játékfejlesztés*, ComputerBooks, 2004

[3] Schwarcz Tibor: *Bevezetés a számítógépi grafikába*, (egyetemi jegyzet)

[4] Dr. Kovács Emőd, Hernyák Zoltán, Radványi Tibor, Király Roland: *A C# programozási nyelv a felsőoktatásban*, (egyetemi jegyzet)

[5] Juhász Imre: *OpenGL*, (egyetemi jegyzet)

[6] Dr. Kovács Emőd: *Komputergrafika*, (órai jegyzet)

[7] Dr. C.-K. Shene: *CS3621 Introduction to Computing with Geometry Notes*, (`http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/notes.html`)