2. fejezet

Paraméteres görbék

2.1. Lagrange görbe rajzoló

A korábbi vonalrajzoló programunk kódját felhasználva készítsünk Lagrange megjelenítőt!

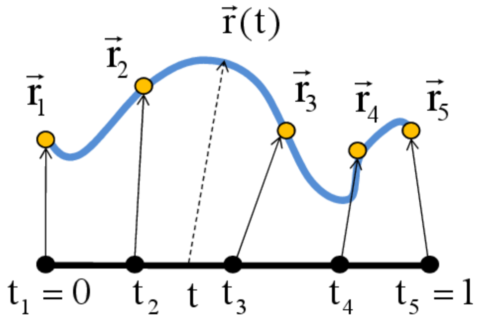
Egér klikkeléssel pontokat lehet felvenni egymásután, melyek piros négyzetként azonnal megjelennek a képernyőn. Ezzel egy időben az aktuális vezérlőpontokra Lagrange görbét illeszt a program, és ezt a görbét megjeleníti. A kontrolpontok a töröttvonal-rajzoló feladathoz hasonlóan legyenek interaktívan mozdíthatók, a görberajzolás folyamatosan kövesse a kontrolpontok változását.

2.1.1. A Lagrange görbe definíciója:

Legyen adott egy n pontot tartalmazó vezérlőpont-vektor $\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_n$ és egy úgynevezett csomópontvektor $(t_1, t_2, \ldots, t_n)$ ahol $\forall i : \vec{r}_i \in \mathbb{R}^2$, $t_i \in \mathbb{R}$ és $t_1 < t_2 < \ldots < t_n$. Az egyszerűség kedvéért legyen az első csomópont $t_1 = 0$, az utolsó $t_n = 1$, a közbe eső csomópontokat pedig helyezzük el a $[0, 1]$ intervallumon egyenletesen, egymástól $\frac{1}{n-1}$ távolságban (2.1 ábra)!

Keressük a következő polinomfüggvényt:

$$\vec{r}(t) = \sum_{i=0}^{n-1} [a_i, b_i] \cdot t^i$$



2.1. ábra. Lagrange interpolációs görbe $n = 5$ kontollponttal. A csomópontvektort a $[0, 1]$ intervallum egyenletes felosztásával származtatjuk

amelyre teljesül, hogy:

$$\vec{r}(t_1) = \vec{r}_1,\ \vec{r}(t_2) = \vec{r}_2, \ldots, \vec{r}(t_n) = \vec{r}_n$$

tehát:

$$\vec{r}(t_j) = [x(t_j), y(t_j)] = \sum_{i=0}^{n-1} [a_i, b_i] \cdot t_j^i = \vec{r}_j : \quad j = 1, \ldots, n$$

A megoldást a következő alakban származtatjuk:

$$\vec{r}(t) = \sum_{i=1}^{n} L_i(t) \cdot \vec{r}_i, \quad \text{ahol} \quad L_i(t) = \frac{\prod_{j \neq i}(t - t_j)}{\prod_{j \neq i}(t_i - t_j)}$$

ahol $L_i(t)$ az i-edik vezérlőponthoz tartozó súlyfüggvény. Belátható, hogy a fenti alak valóban $n-1$-edfokú polinomfüggvényt definiál, ami a csomópontokban a kijelölt vezérlőpontokat interpolálja. Például $n = 3$ esetre a következő a függvény kifejtése:

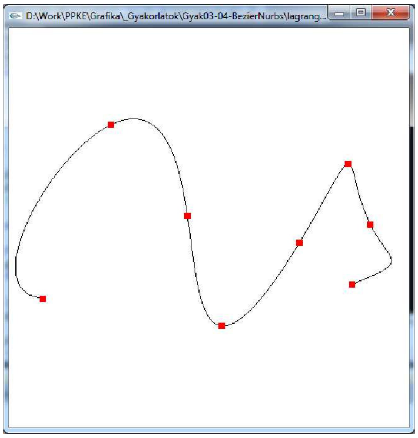
$$\vec{r}(t) = \frac{(t-t_2)(t-t_3)}{(t_1-t_2)(t_1-t_3)} \cdot \vec{r}_1 + \frac{(t-t_1)(t-t_3)}{(t_2-t_1)(t_2-t_3)} \cdot \vec{r}_2 + \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_3-t_2)} \cdot \vec{r}_3$$

A fenti képlet nyilvánvalóan másodfokú polinomot határoz meg, a csomóponti értékek behelyettesítésével pedig a helyes interpolációról is meggyőződhetünk.

2.1.2. A Lagrange görbe implementációja

Adattagok:

`MyPoint ctrlpoints[MAXPTNUM];` //Ez csak pszeudo kód, a "MyPoint"-ot önállóan kell reprezentálni!



2.2. ábra. A Lagrange interpolációs görbe implementációja

```
int ptnum; //aktuális kontrolpontszám

double knotVector[MAXPTNUM];
```

Együttható számítás:

```
double L( int i, double tt ) {
   double Li = 1.0;
   for (int j = 0; j < ptnum; j++) {
     if (i != j)
       Li *= (tt - knotVector[j]) / (knotVector[i] -knotVector[j]);
   }
   return Li;
}
```

Egyenletes csomóponti (knot) vektor inicializálás:

```
for ( int i=0; i < ptnum; i++) {
   knotVector[i]=(double)i/(double(ptnum-1));
}
```

Görbe adott pontjának számítása t paraméterértéknél:

```
MyPoint CalcLagrangePoint (float t) {
  MyPoint actPT(0,0);
   for(int i = 0; i < ptnum; i++)
     actPT+=ctrlPoint[i]*L(i,t);
   return actPT;
```

}

A várt kimenet a 2.2 ábrán látható.

2.2. Bézier görbe számolt együtthatók alapján

A korábbi vonalrajzoló programunk kódját felhasználva készítsünk Bézier megjelenítőt.

Egér klikkeléssel pontokat lehet felvenni egymásután, melyek piros négyzetként azonnal megjelennek a képernyőn. Ezzel egy időben az aktuális vezértlőpontokra Bézier görbét illeszt a program, és ezt a görbét megjeleníti. A kontrolpontok a töröttvonal-rajzoló feladathoz hasonlóan legyenek interaktívan mozdíthatók, a görberajzolás folyamatosan kövesse a kontrolpontok változását.

2.2.1. A Bézier görbe definíciója

Definiáljuk a görbénket továbbra is súlyfüggvények segítségével, melyeket most $B_i(t)$-vel jelölünk:

$$\vec{r}(t) = \sum_{i=0}^{m} B_i(t) \cdot \vec{r}_i$$

A Lagrange görbe egyik fő hibája a természetellenes hullámossága, amit az $L_i(t)$ súlyfüggvények előforduló negatív értékei okoztak. A hullámosság eltüntethető, ha a következő két feltétel teljesül:

- $B_i(t) \geq 0$ minden i-re és $t \in [0,1]$-re.
- $\sum_{i=0}^{m} B_i(t)$ minden $t \in [0,1]$-re.

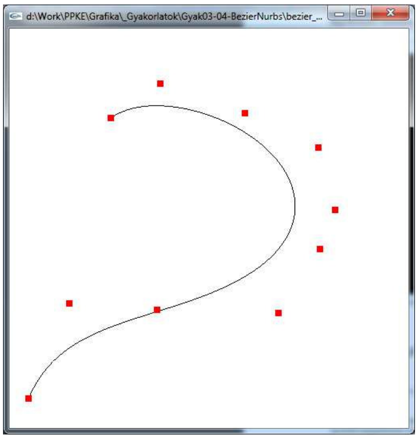
Ekkor a görbe valamennyi pontja a vezérlőpontok konvex burkán belül esik.

A fenti kritériumoknak eleget tesznek a Bernstein polinomok

$$B_i^{(m)}(t) = \binom{m}{i} t^i \cdot (1-t)^{m-i} \quad \text{ahol} \quad \binom{m}{i} = \frac{m!}{i!(m-i)!}$$

mivel a nemnegativitás triviális:

$$B_i^{(m)}(t) \geq 0 \qquad \forall m, i, t$$



2.3. ábra. A Bézier approximációs görbe implementációja

valamint a binomiális tétel alapján:

$$1 = \big(t + (1-t)\big)^m = \sum_{i=0}^{m} \binom{m}{i} t^i \cdot (1-t)^{m-i} = \sum_{i=0}^{m} B_i^{(m)}(t), \quad \forall t \in [0,1]$$

Ráadásul belátható, hogy a görbe az első vezérlőpontból indul, az utolsóba érkezik, mivel:

$$B_0^{(m)}(0) = 1, \qquad B_m^{(m)}(1) = 1$$

a közbülső vezérlőpontokon általában nem megy át a görbe.

2.2.2. A Bézier görbe „naív” implementációja

Implementáljuk a Bézier görbe számítót a tanult definíció alapján!

```
//MyPoint implementációja: önállóan
MyPoint ctrlpoints[MAXPTNUM];
int ptnum; //aktuális kontrolpontszám
...
float B(int i, float t) {
   GLfloat Bi = 1.0;
   for(int j = 1; j <= i; j++) Bi *= t * (ptnum-j)/j;
     for( ; j < ptnum; j++) Bi *= (1-t);
   return Bi;
}

MyPoint CalcBezierPoint (float t) { //Pszeudo Point
  Point actPT=[0,0];
```

```
  for (int i = 0; i <= ptnum; i++)
    actPT+=ctrlPoint[i]*L(i,t);
  return actPT;
}
```

2.3. Bézier görbe OpenGL implementációja

Hatékonyabb megoldás, beépített OpenGL függvények felhasználásával.

Egydimenziós leképezés:

`void glMap1fd(Glenum target, TYPE u1, TYPE u2, GLInt stride, GLInt order, const TYPE * points)`

- target: mit reprezentálnak a kontrollpontok: modelltérbeli pontot (`GL_MAP1_VERTEX_3`) vagy színt (`GL_MAP1_COLOR_4`) stb
- u1, u2: paramétertartomány (nálunk [0,1])
- stride: nálunk a pontok dimenziója (itt 3)
- order: görbe rendje (kpontok száma+1)
- points: kontrollpontokat tartalmazó tömb

A parancs kiadása előtt engedélyezni kell a leképzés opciót:
`glEnable(GL_MAP1_VERTEX_3);`

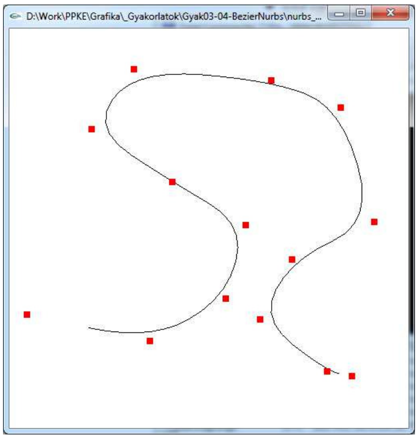
Kontroll pontok definiálása

`GLfloat ctrlpoints[MAXPTNUM][3];`

- `ctrlpoints[i][j]` az i-edik kontrolpont j-edik koordinátája
- 3D pontokkal dolgozik, a pont koordinátái rendre: [x,y,z], 2D-ben z=0-t használjunk

`void glEvalCoord1fd(TYPE u);`

az u paraméterértéknél kiértékeli a görbét, azaz meghatározza az aktuális pontot és esetünkben a `glVertex*()` parancsot is automatikusan végrehajtja rá (tehát azonnal meg is jeleníti)



2.4. ábra. A NURBS görbe implementációja

2.4. NURBS görberajzoló

A korábbi vonalrajzoló programunk kódját felhasználva készítsünk NURBS megjelenítőt.

Egér klikkeléssel pontokat lehet felvenni egymásután, melyek piros négyzetként azonnal megjelennek a képernyőn. Ezzel egy időben az aktuális kontrolpontokra NURBS görbét illeszt a program, és ezt a görbét megjeleníti. A kontrolpontok a töröttvonal-rajzoló feladathoz hasonlóan legyenek interaktívan mozdíthatók, a görberajzolás folyamatosan kövesse a kontrolpontok változását.

Nurbs görbe rajzoló a következő elemeket és lépéseket tartalmazza:

Adattagok:

```
GLUnurbsObj *theNurb; //NURBS objektum
#define ORDER 3 //NURBS rendje - ellentétben Bézierrel, ez tőlünk függő
szabad partaméter, nálunk legyen konst 3!
GLfloat ctrlpoints[MAXPTNUM][3]; //kontrollpontok
GLfloat knots[MAXPTNUM+ORDER]; //kiértékelés paraméterértékeit (ti-ket)
tartalmazó vektor
```

Létrehozás, inicializáció:

```
theNurb=gluNewNurbsRenderer();
gluNurbsProperty(theNurb,GLU_SAMPLIN_TOLERANCE, 25.0);
gluNurbsProperty(theNurb, GLU_DISPLAY_MODE, GLU_FILL);
```

Knot inicializáció:
`ptnum=` "aktuális kontroll pontok száma" // knots vektor hasznos része mindig ptnum+ORDER elemű

Töltsük fel a knots vektor első ptnum+ORDER elemét: osszuk fel a [0 1] intervallumot egyenletesen ptnum+ORDER részre, és `knots[i]` legyen `i/ (ptnum+ORDER)`;

Teljes görbe rajzolása:

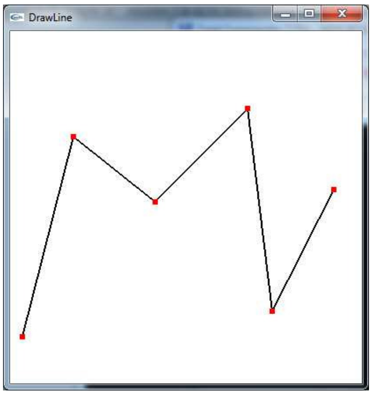
```
gluBeginCurve(theNurb);
  gluNurbsCurve(theNurb,ptnum+ORDER, knots,3, &ctrlpoints[0][0],
                ORDER,GL_MAP1_VERTEX_3);
gluEndCurve(theNurb)
```

2.5. Implementáljuk a Catmull-Clark algoritmust

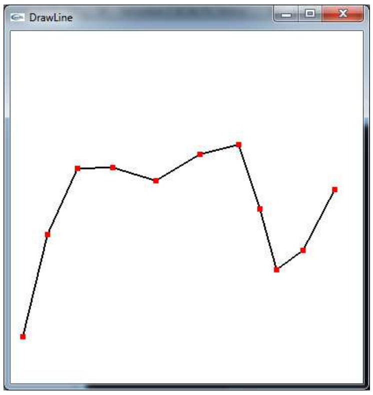
Használjuk a korábbi törött vonal rajzoló programunkat, majd a kapott vonalat simítsuk Catmull algoritmussal! Implementáljuk a simító inverz műveletét is!

Az egér bal klikkelésével lehet hozzávenni pontokat a törött vonalhoz, amely egy-egy új szakaszt eredményez, majd a teljes vonal azonnal megjelenik a képernyőn.

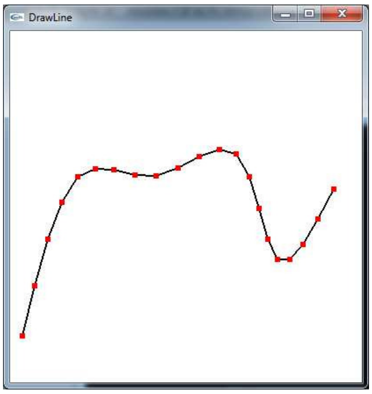
Ezután "j" gomb leütésével simíts egyet a vonalon, és "n" leütésével csökkentsd a simítást



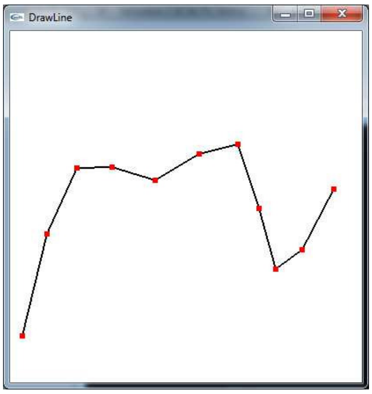
(a) Kezdeti törtöttvonal



(b) j lenyomásával



(c) j lenyomásával



(d) n lenyomásával

2.5. ábra. Catmull-Clark simítás és inverze