A számítógépes grafika alapjai

Benedek Csaba

2019. március 11.

0.1. Vonalrajzoló

Feladat: írjunk 2D vonalrajzolót!

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

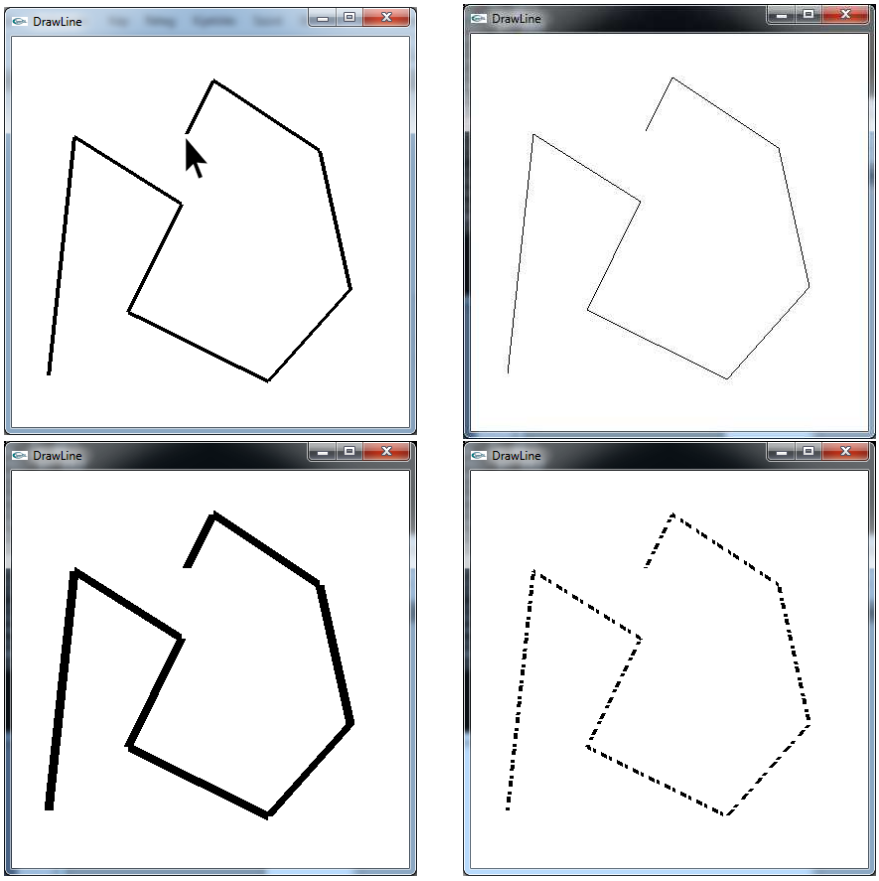
- ,,C" gombbal lehet törölni a képernyőt
- numerikus (1-9) gombokkal a vonal vastagságát lehet állítani
- ,,s" billentyűvel szaggatottá lehet tenni, illetve a szaggatott módot kikapcsolni.

Várt kimeneti képek az láthatók.

0.2. Interaktív pontmozgatás

Egészítsük ki az előbbi vonalrajzolót úgy, hogy lehessen a már meglévő pontok helyzetét utólag interaktívan módosítani.

Ha a felhasználó egy meglévő pont közelében lenyomja a jobb egérgombot, akkor a gomb elengedéséig az egérrel tudja az aktuális pontot mozgatni, mozgás közben pedig interaktívan jelenjen meg a folyamatosan módosuló törött vonal.



1. ábra. A vonalrajzoló program várt kimenetei

1. fejezet

Paraméteres görbék

1.1. Lagrange görbe rajzoló

A korábbi vonalrajzoló programunk kódját felhasználva készítsünk Lagrange megjelenítőt!

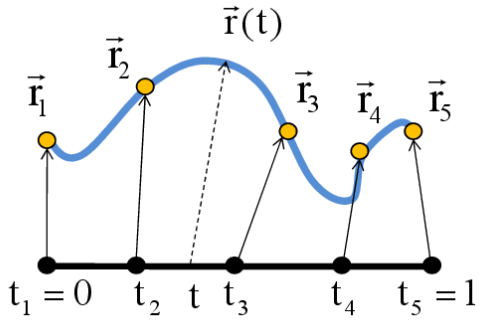
Egér klikkeléssel pontokat lehet felvenni egymásután, melyek piros négyzetként azonnal megjelennek a képernyőn. Ezzel egy időben az aktuális vezérlőpontokra Lagrange görbét illeszt a program, és ezt a görbét megjeleníti. A kontrolpontok a töröttvonal-rajzoló feladathoz hasonlóan legyenek interaktívan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

1.1.1. 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 (1.1 ábra)!

Keressük a következő polinomfüggvényt:

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



1.1. 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:

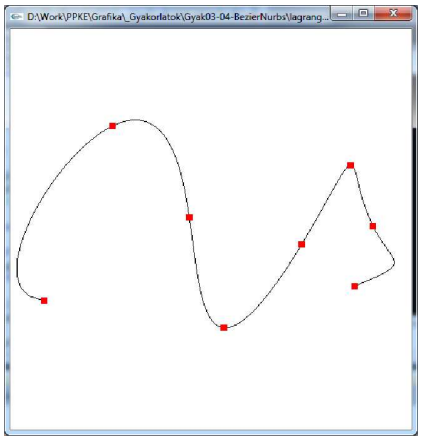
$$\vec{r}(t) = \frac{(t - t_2)(t - t_3)}{(t_1 - t_2)(t_1 - t_3)} \cdot \vec{r}_1 + \frac{(t - t_1)(t - t_3)}{(t_2 - t_1)(t_2 - t_3)} \cdot \vec{r}_2 + \frac{(t - t_1)(t - t_2)}{(t_3 - t_1)(t_3 - t_2)} \cdot \vec{r}_3$$

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

1.1.2. A Lagrange görbe implementációja

Adattagok:

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



1.2. 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;
}
```

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 1.2 ábrán látható.

1.2. 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.

1.2.1. 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:

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

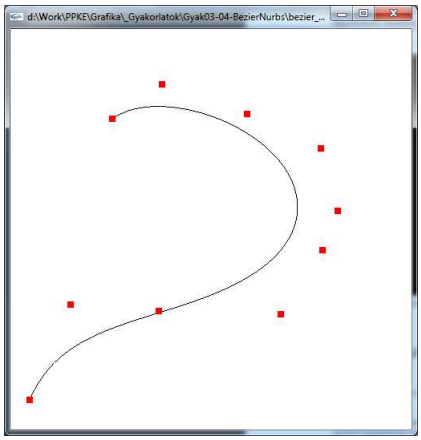
valamint a binomiális tétel alapján:

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

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.

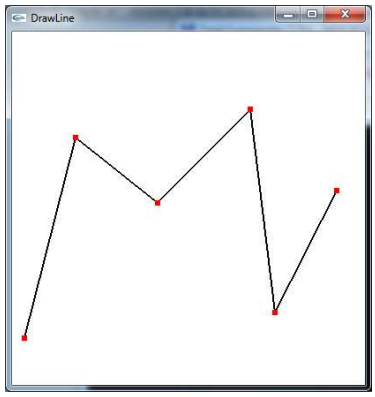


1.3. ábra. A Bézier approximációs görbe implementációja

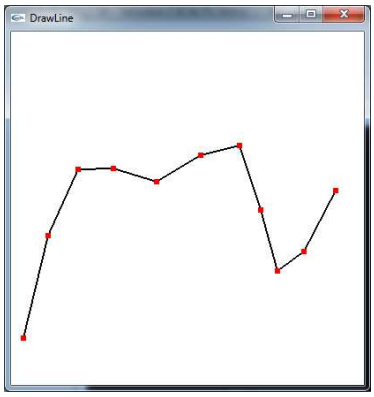
1.2.2. A Bézier görbe ,,naív" implementációja

Implementáljuk a Bézier görbe számítót a tanult definíció alapján!

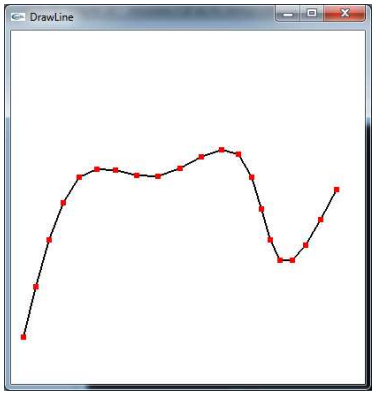
```
//MyPoint implementációja:  önállóan
MyPoint ctrlpoints[MAXPTNUM];
int ptnum; //aktuális kontrolpontszám
...
float B(int i, float t) {
   GLfloat Bi = 1.0;
   for(int j = 1; j <= i; j++) Bi *= t * (ptnum-j)/j;
     for( ; j < ptnum; j++) Bi *= (1-t);
   return Bi;
}

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;
}
```



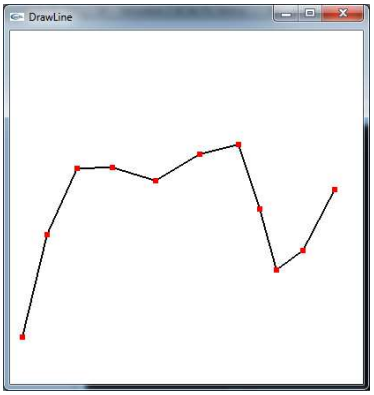
(a) Kezdeti törtöttvonal



(b) j lenyomásával



(c) j lenyomásával



(d) n lenyomásával

1.4. ábra. Catmull-Clark simítás és inverze

1.3. 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