



# 2D bezier görbe megjelentése

FÉLÉVES BEADANDÓ SZÁMÍTÓGÉPI GRAFIKA TANTÁRGYBÓL

## Megvalósítandó feladat

Bezier görbe kirajzolása minimum 25 kontrolponttal OpenGL grafikus megjelenítés segítségével.

## Program leírása

A feladat bemutatása GLUT toolkitre épül., mellyel egyszerűen inicializálható adott platformon az OpenGL ablakozó rendszer.

A görbét kirajzolásához a program kényelmi szempont miatt generál egy két véletlenszerű paramétert.

- *numberofpoints* : A kirajzolt görbe kontrollpontjainak a száma.
- *ctrlpoints[PMSIZE][3]*: A kirajzolandó görbe kontrolpontjait tároló mátrix

A változó és a mátrix is a program kezdetén inicializálásra és véletlen értékkel feltöltésre kerül. A véletlen számok generálásának segítségére a következő két függvény került definiálásra:

```
int randomInt(int minValue = 2, int maxValue = (PMSIZE-1)){  
    return (rand() % (maxValue - minValue + 1)) + minValue;  
}
```

Visszatér egy 2 és a program elején definiált makró értéke minden 1 értéktartománybeli egész számmal. Ez fogja megadni nekünk a megjelenítendő görbe kontrolpontjainak a számát, melyet

```
float randomFloat(float minValue = 0.0f, float maxValue = 1.0f){  
    float betweenoandi = static_cast<float>(rand()) /  
        static_cast<float>(RAND_MAX);  
    return (minValue + betweenoandi * (maxValue - minValue))-0.95;  
}
```

Visszatér egy [-0.9;+0.9] értékkal, amellyel a pontmátrix elemeinek értéket adunk.

Itt fontos megemlíteni, hogy a megjelenítés a létrehozott ablak közepét tekinti origónak, s előjelhelyesen az ablak széleit 1.0 értéknek.

```
void generateRandomPoints(){  
    for(int i=0; i<PMSIZE; i++){  
        ctrlpoints[i][0]=randomFloat();  
        ctrlpoints[i][1]=randomFloat();  
    }  
}
```

Ebben az eljárásban a randomFloat(); függvény felhasználásával feltölthük a ctrlpoints mátrix elemeit véletlenszerű pontokkal.

A program egyéb működésbeli eljárásokat is igényel ezek részletezései a következők:

```
void psInit()
{
    srand(time(NULL));
    glClearColor(0.1f, 0.2f, 0.3f, 1.0f);

    for(int i=0; i<PMSIZE; i++){
        ctrlpoints[i][0]=0;
        ctrlpoints[i][1]=0;
    }
}
```

A programrész feladata egy alapvető inicializálás végrehajtása. Elvégezzük a véletlenszámgenerálás beállítását. Beállítjuk a megjelenítendő ablak háttérszínét RGBA leképzéssel, valamint a ctrlpoints mátrix összes pontját nullába álltjuk.

```
void printMessage(){
    printf("Grafika beadando 2020 - 2D bezier gorbe kirajzolasa\n\r");
    printf("Grebely János - CL4WWB - G2BIL");
    for(int k=0; k<51; k++){
        printf("*");
    }
    printf("\n\r * q - kilepes;");
    printf("\n\r * e - Random szamu kontrolpont generalasa;");
    printf("\n\r * r - Random kontrolpontok generalasa;");
    printf("\n\r * a - Kontrolpontok szamanak novelese;");
    printf("\n\r * s- Kontrolpontok szamanak csokkentese;\n\r");
    for(int k=0; k<51; k++){
        printf("*");
    }
}
```

Egy egyszerű tájékoztató fejléc generálása a parancssoros ablakra.

Az grafikus megjelenítés funkciói kívánnak függvényeket, melyeket a főprogramban regisztrálunk majd callback függvényként, s adott esemény bekövetkezésekor ezek meghívódnak.

```
static void pskey(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 'q':
            exit(0);
            break;
        case 'r':
            generateRandomPoints();
            break;
        case 'a':
            if(numberofpoints!=(PMSIZE-1)){
                numberofpoints++;
            }
            break;
        case 's':
            if(numberofpoints!=2){
                numberofpoints--;
            }
            break;
        case 'e':
            numberofpoints=randomInt();
        }
    glutPostRedisplay();
}
```

Ezen függvény az opengl billentyűzet kezelése miatt szükséges. Amikor lenyomunk egy billentyűt, akkor ezen esemény bekövetkezésekor végrehajtódik. A programban definiált események a következők:

q – kilépés a programból

r – pontmátrix random pontokkal való feltöltése

a – megjelenített pontok számának növelése. A pontok számának növekedését egy feltétellel korlátozzuk.

s – megjelenített pontok számának csökkentése. A pontok számának csökkentését egy feltétellel korlátozzuk.

e – megjelenített pontok számának véletlenértéket adunk.

```

static void psdisplay(void)
{
    int i=o; // inkrementális változó
    glClear(GL_COLOR_BUFFER_BIT);

    //=====
    /*
     * Megjelenítjük a pontmátrix elemeit
     */
    glPointSize(6);
    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    for (i = o; i < numberofpoints; i++){
        if(i==o){
            glColor3f(1.0,0.0,0.0); // kezdőpont - vörös
        }else if(i==numberofpoints-1){
            glColor3f(1.0,1.0,0.0); // végpont - sárga
        }else{
            glColor3f(0.0,1.0,0.0); // köztes pontok - zöld
        }
        glVertex3fv((GLfloat*)&ctrlpoints[i][o]);
    }
    glEnd();
    /*
     * Pontokat összekötjük egy vonallal a szemléltetés miatt
     */
    glLineWidth(0.2);
    glColor3f(0.5, 0.5, 0.5);
    glBegin(GL_LINE_STRIP);

    for (int i = o; i < numberofpoints; i++){
        glVertex3f(ctrlpoints[i][o],ctrlpoints[i][1],0.0);
    }
    glEnd();
    //=====
    /*
     * Bezier görbe leképzése az opengl beépített eljárásával
     */
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, numberofpoints,
(GLfloat*)&ctrlpoints[o][o]);
    glEnable(GL_MAP1_VERTEX_3);
    glColor3f(1.0, 0.0, 1.0);
    glLineWidth(2.0);
    glBegin(GL_LINE_STRIP);
    for (int i = o; i <= (10*numberofpoints); i++){
        glEvalCoord1f((GLfloat) i/(10*numberofpoints));
    }
    glEnd();
    glFinish();
}

```

Ezen függvényben valósítjuk meg a tényleges görbe rajzolást a paraméterek alapján. A folyam első részében megjelenítjük ctrlpoints elemeit vizuálisan. Az első pontot vörössel, köztes pontokat zölddel az utolsót pedig sárgával.

A soron következő egységen a szemléltetés miatt, ezeket a pontokat sorrendben összekötjük egy vonallal.

Az eljárás végén a tényleges görbe rajzolást hajtjuk végre az opengl-be épített interpolációs függvénnyel. A belső függvények átadjuk, hogy 3 paraméteres térben voldozunk, [0;1] értéktartományban, a valós adatok és a közelítő polinom fokszámát majd pedig a pontokat tartalmazó mátrix pointerét. A glEvalCoordif() függvénnyel a rendszer lehetővé teszi, hogy megadott értelmezési tartományon belül n egyenlő részre osztva a paramétertartományban rácspontokat hozzunk létre, majd egyetlen függvényhívással kiszámíthassuk a rácspontokhoz tartozó görbepontokat.

```
int main(int argc, char *argv[])
{
    printMessage();
    glutInit(&argc, argv);
    glutInitWindowSize(1024,768);
    glutInitWindowPosition(250,250);
    glutInitDisplayMode(GLUT_RGB);

    glutCreateWindow("me-grafika-bezier");
    psInit();
    generateRandomPoints();
    glutDisplayFunc(psdisplay);
    glutKeyboardFunc(pskey);

    glutMainLoop();

    return EXIT_SUCCESS;
}
```

A main függvényben a tényleges programmegvalósítás, már a fentebb részletezett eljárások segítségével történik.

Kezdetben a parancssorba generáljuk a fejlécet. Inicializáljuk a megjelenítésért felelős környezetet, beállítjuk az ablakméretet. Meghatározzuk, hogy hol jelenjen meg a képernyőn az ablak, beállítjuk a megjelenítési módot (esetünkben RGB) s végezetül definiáljuk a létrehozott ablak címsorát.

Az általunk létrehozott függvények segítségével elvégezzük a programunk inicializálását, és feltöljük a pontokat tartalmazó mátrixunkat értékekkel.

A glutDisplayFunc(psdisplay) valamint a glutKeyboardFunc(pskey) parancsokkal regisztráljuk az eseménykezelőbe azokat a függvényeket, amelyek az adott feladatok végrehajtását definiáltuk. Végezetül a glutMainLoop(); egy végtelen ciklust hoz létre, az események kezelésére.

## Teljes forráskód

```
1.  /*
2.   * Grafika beadandó 2020 - 2D bezier görbe kirajzolása
3.   *
4.   * Írta Grebely János / CL4WWB / - G2BIL
5.   *
6.   * A program célja 2D bezier görbe kirajzolása.
7.   * Maximum pontok száma PMSIZE makróban definiálható.
8.   *
9.   * A program alapvető interakcióra képes a lentebb írtak alapján
10.  * Kilépés : q
11.  * Random számú kontrolpont generálása : e
12.  * Random kontrolpontok generálása : r
13.  * Kontrolpontok számának növelése : a
14.  * Kontrolpontok számának csökkentése : s
15.  */
16.
17. #ifdef __APPLE__
18. #include <GLUT/glut.h>
19. #else
20. #include <GL/glut.h>
21. #endif
22.
23. #include <stdlib.h>
24. #include <stdio.h>
25. #include <time.h>
26.
27. #define PMSIZE 30
28.
29. int numberofpoints=2;
30. GLfloat ctrlpoints[PMSIZE][3];
31.
32. float randomFloat(float minValue = 0.0f, float maxValue = 1.0f){
33.     float betweenoandi = static_cast<float>(rand()) / static_cast<float>(RAND_MAX);
34.     return (minValue + betweenoandi * (maxValue - minValue))-0.95;
35. }
36.
37. int randomInt(int minValue = 2, int maxValue = (PMSIZE-1)){
38.     return (rand() % (maxValue - minValue + 1)) + minValue;
39. }
40.
41. void generateRandomPoints(){
42.     for(int i=0; i<PMSIZE; i++){
43.         ctrlpoints[i][0]=randomFloat();
44.         ctrlpoints[i][1]=randomFloat();
45.     }
46. }
47. void psInit()
48. {
```

```

49. srand(time(NULL));
50. glClearColor(0.1f, 0.2f, 0.3f, 1.0f);
51.
52. for(int i=0; i<PMSIZE; i++){
53.     ctrlpoints[i][0]=o;
54.     ctrlpoints[i][1]=o;
55. }
56. }
57. static void psdisplay(void)
58. {
59.     int i=o; // inkrementális változó
60.     glClear(GL_COLOR_BUFFER_BIT );
61.
62. //=====
63. /*
64. * Megjelenítjük a pontmátrix elemeit
65. */
66. glPointSize(6);
67. glColor3f(0.0, 1.0, 0.0);
68. glBegin(GL_POINTS);
69. for (i = o; i < numberofpoints; i++){
70.
71.     if(i==o){
72.         glColor3f(1.0,0.0,0.0); // kezdőpont - vörös
73.     }else if(i==numberofpoints-1){
74.         glColor3f(1.0,1.0,0.0); // végpont - sárga
75.     }else{
76.         glColor3f(0.0,1.0,0.0); // köztes pontok - zöld
77.     }
78.     glVertex3fv((GLfloat*)&ctrlpoints[i][0]);
79. }
80. glEnd();
81. /*
82. * Pontokat összekötjük egy vonallal a szemléltetés miatt
83. */
84. glLineWidth(0.2);
85. glColor3f(0.5, 0.5, 0.5);
86. glBegin(GL_LINE_STRIP);
87.
88. for (int i = o; i < numberofpoints; i++){
89.     glVertex3f(ctrlpoints[i][0],ctrlpoints[i][1],0.0);
90. }
91. glEnd();
92. //=====
93. /*
94. * Bezier görbe leképzése az opengl beépített eljárásával
95. */
96. glMapIf(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, numberofpoints, (GLfloat*)&ctrlpoints[0][0]);
97. glEnable(GL_MAP1_VERTEX_3);
98. glColor3f(1.0, 0.0, 1.0);

```

```

99.    glLineWidth(2.0);
100.   glBegin(GL_LINE_STRIP);
101.   for (int i = 0; i <= (10*numberofpoints); i++){
102.     glEvalCoord1f((GLfloat) i/(10*numberofpoints));
103.   }
104.   glEnd();
105.   glFinish();
106. }
107.
108.
109. static void pskey(unsigned char key, int x, int y)
110. {
111.   switch (key)
112.   {
113.     case 'q':{
114.       exit(0);
115.       break;
116.     }
117.     case 'r':{
118.       generateRandomPoints();
119.       break;
120.     }
121.     case 'a':{
122.       if(numberofpoints!=(PMSIZE-1)){
123.         numberofpoints++;
124.       }
125.       break;
126.     }
127.     case 's':{
128.       if(numberofpoints!=2){
129.         numberofpoints--;
130.       }
131.       break;
132.     }
133.     case 'e':{
134.       numberofpoints=randomInt();
135.     }
136.   }
137.   glutPostRedisplay();
138. }
139. void printMessage(){
140.   printf("Grafika beadando 2020 - 2D bezier gorbe kirajzolasa \n\r");
141.   printf("Grebely Janos - CL4WWB - G2BIL\n\r");
142.   for(int k=0; k<51; k++){
143.     printf("*");
144.   }
145.   printf("\n\r * q - kilepes;");
146.   printf("\n\r * e - Random szamu kontrolpont generalasa;");
147.   printf("\n\r * r - Random kontrolpontok generalasa;");
148.   printf("\n\r * a - Kontrolpontok szamanak novelese;");

```

```
149. printf("\n\r * s- Kontrolpontok szamanak csokkentese;\n\r");
150. for(int k=0; k<51; k++){
151.     printf("*");
152. }
153. }
154. int main(int argc, char *argv[])
155. {
156.     printMessage();
157.     glutInit(&argc, argv);
158.     glutInitWindowSize(1024,768);
159.     glutInitWindowPosition(250,250);
160.     glutInitDisplayMode(GLUT_RGB);
161.
162.     glutCreateWindow("me-grafika-bezier");
163.     psInit();
164.     generateRandomPoints();
165.     glutDisplayFunc(psdisplay);
166.     glutKeyboardFunc(pskey);
167.
168.     glutMainLoop();
169.
170.     return EXIT_SUCCESS;
171. }
```