```
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(3591,0,0,0,96,48,0,60,70,6,6,6,6,70,60,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(4104,0,0,124,198,198,198,198,254,198,198,198,198,198,230,96,192,0),
(4103,0,0,126,6,6,6,6,62,6,6,6,6,6,126,24,48,0),
(4103,0,0,0,0,0,0,60,102,102,126,6,6,70,60,24,48,0),
(3591,0,0,24,24,88,120,56,24,28,30,26,24,24,24,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(3591,0,32,60,86,6,6,6,6,6,6,6,6,70,60,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(3591,0,32,60,86,6,14,12,28,24,56,48,112,98,60,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(3590,0,0,6,6,6,102,54,30,14,6,6,6,6,62,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(3590,0,0,0,48,24,0,28,38,6,12,24,48,50,28,0,0,0),
(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),
(3591,0,16,126,104,96,48,48,24,24,12,12,6,6,126,0,0,0),
(3591,0,0,126,96,96,48,48,126,24,12,12,6,6,126,0,0,0),
(3591,0,0,0,48,24,0,60,102,102,102,102,102,102,60,0,0,0),
(3592,0,32,124,214,198,198,198,198,198,198,198,198,198,124,0,0,0),
(3591,0,0,0,48,24,0,62,102,102,102,102,102,102,102,0,0,0),
(3595,0,192,1122,1030,1038,1054,1082,1138,1250,1474,1922,
 1794,1538,1026,0,0,0),
 (3592,0,0,0,48,24,0,254,192,96,48,24,12,6,254,0,0,0),
 (3592,0,0,0,48,48,0,254,192,96,48,24,12,6,254,0,0,0));
procedure ShadowTextXY(x,y : word; color1,color2 : byte;
                       s : string; p : pointer);
{ Wyświetla tekst z cieniem }
begin
 bittextxy(x+1,y+1,color2,s,p);
 bittextxy(x,y,color1,s,p);
end;
begin
  asm
    mov ax,0012h
    int 10h
  end;           { ustawienie trybu 640x480x16 karty VGA }
 bittextxy(10,10,white,'Moduł BitFonts', @my_fonts);
 shadowtextxy(100,100,lightblue,lightcyan,
              'Czy ta czcionka czegoś nie przypomina ?',
               @my_fonts);
 repeat until keypressed;
 asm
   mov ax,0003h
   int 10h
 end;                          { powrót do trybu tekstowego }
end.
```

Wydruk 2.

Turbo Pascal

Krzywe sklejane

Krzysztof Stefański

W wielu zagadnieniach grafiki komputerowej, a także CAD występuje problem narysowania krzywej o dostatecznie gładkim przebiegu, przechodzącej przez zadane punkty (lub w ich pobliżu) i spełniającej ponadto następujące warunki:

- koszt jej obliczania nie powinien być zbyt duży (1);
- powinna mieć przyjemny dla oka, „miękki" wygląd (2).

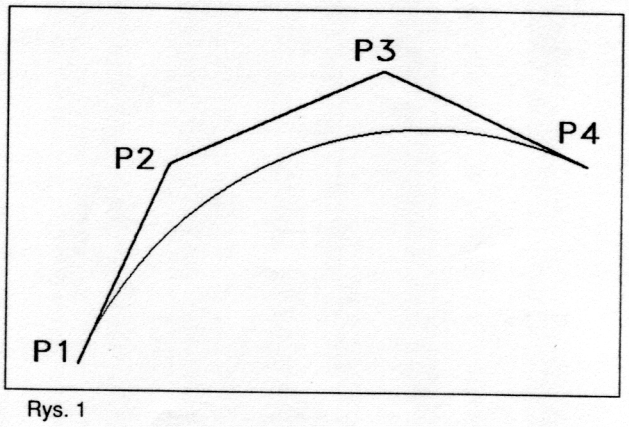
Warunek 1 zazwyczaj łatwo sprawdzić dla konkretnego algorytmu. Warunek 2 ma charakter bardziej psychologiczny, przez co jest trudniejszy do zweryfikowania. Można powiedzieć, że krzywa spełnia warunek 2, jeśli jest zbliżona do krzywej, którą narysowalibyśmy ręcznie, używając krzywików i podobnych narzędzi.

Naturalnym kandydatem do rozpatrzenia poruszanego problemu są wielomiany. Dają się łatwo obliczać (np. algorytmem Hornera), ponadto przez dowolne n punktów przechodzi krzywa wielomianowa stopnia najwyżej n-1. Niestety nie spełniają one na ogół warunku 2, poza tym koszt ich obliczania rośnie wraz ze stopniem wielomianu i już dla kilkunastu punktów jest dość znaczny.

Rozsądnym wyjściem z tej sytuacji jest używanie większej liczby wielomianów niskiego stopnia. Każdy z nich odpowiada za przebieg krzywej przez kilka punktów i są odpowiednio „sklejone" ze sobą. Stąd technika ta została nazwana metodą funkcji sklejanych (ang. spline functions). Najszersze zastosowanie znalazły funkcje sklejane trzeciego stopnia (ang. cubic splines), które będą przedmiotem tego artykułu.

Doświadczenia wykazały, że najlepiej zrezygnować z postulatu przechodzenia krzywej przez wszystkie zadane punkty (tzw. punkty kontrolne). Niektóre z nich nie muszą leżeć na krzywej, ale odpowiadają za właściwy kierunek stycznej na końcach krzywej. Umożliwi to później gładkie sklejenie z sąsiednimi krzywymi tego typu.

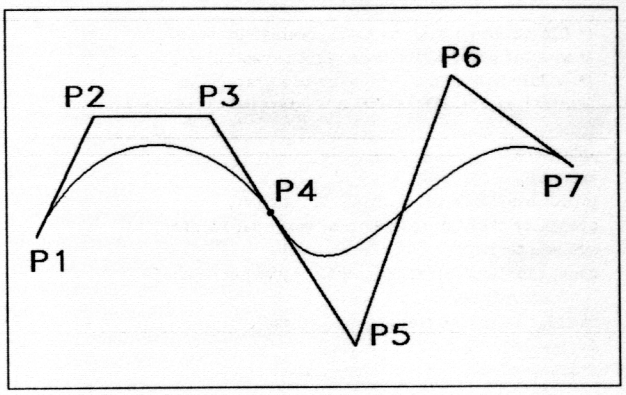
Dla krzywych trzeciego stopnia przyjmuje się cztery punkty kontrolne: P1(x1,y1), P2(x2,y2), P3(x3,y3) oraz P4(x4,y4). Punkty P1 i P4 są końcami krzywej, która jest w punkcie P1 styczna do P1P2, a w punkcie P4 styczna do P3P4 (rys.1).



Rys. 1

W ten sposób można „gładko" skleić tę krzywą z następną, o punktach kontrolnych P4, P5, P6, P7. Wystarczy określić punkt P5 tak, aby punkt P4 należał do odcinka P3P5 (rys. 2).

Możemy także, wyznaczając odpowiednio punkt P5, uzyskać „ostrze" w miejscu sklejenia. Wszystko zależy od naszych potrzeb.



Rys. 2

Ogólna postać równania funkcji sklejanej jest następująca.

Tworzymy dwie funkcje X(t), Y(t), zwracające współrzędne punktu na krzywej zależne od parametru t (0<=t<=1), o równaniach:

```
X(t) = x1*F1(t) + x2*F2(t) + x3*F3(t) + x4*F4(t)
Y(t) = y1*F1(t) + y2*F2(t) + y3*F3(t) + y4*F4(t)
```

gdzie F1(t), F2(t), F3(t), F4(t) (tzw. funkcje łączące) są odpowiednio dobranymi wielomianami trzeciego stopnia. Różne algorytmy wyznaczania funkcji sklejanych różnią się właśnie doborem tych funkcji (każdy z nich ma swoje zalety i wady). Wielu autorów proponuje wielomiany Bernsteina

```
                3
F1(t)=(1-t)
        F2(t)=3*t*(1-t)ý        F3(t)=3*tý*(1-t)
   3 F4(t)=t
```

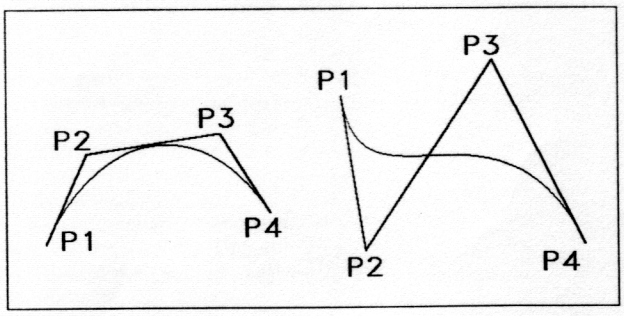
których zaletą jest to, że przy odpowiednim doborze punktów kontrolnych możemy uzyskać funkcje sklejane, mające drugą pochodną ciągłą (krzywe Béziera). Do celów grafiki komputerowej przyjmiemy inny układ funkcji łączących, wprowadzony przez H. Timmera (Douglas Aircraft Co.):

```
F1(t)=(1-2*t)*(1-t)ý
F2(t)=4*t*(1-t)ý
F3(t)=4*tý*(1-t)
F4(t)=(2*t-1)*tý
```

Charakterystyczną własnością tego układu funkcji są równości:

```
X(1/2)=(x2+x3)/2
Y(1/2)=(y2+y3)/2
```

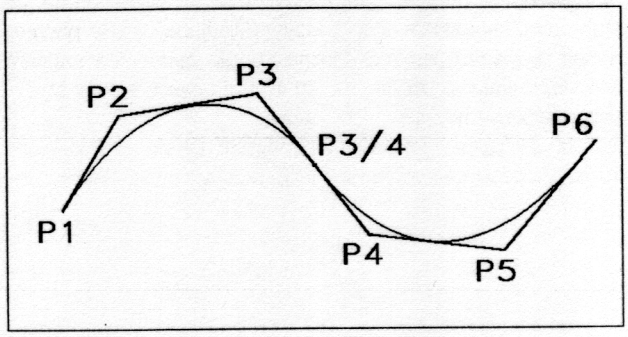
z których wynika, że krzywa przecina w połowie odcinek P2P3 lub jest do niego w tym punkcie styczna (rys. 3).



Rys. 3

Pozwoli nam to na wygodne tworzenie bardziej skomplikowanych krzywych. Jedynym ograniczeniem przedstawionej dalej metody jest parzystość liczby punktów kontrolnych.

Zasadę sklejania krzywych najlepiej zilustruje przykład krzywej z sześcioma punktami kontrolnymi P1,.., P6 (rys. 4).



Rys. 4

Krzywa ta składa się z dwóch fragmentów. Pierwszy ma punkty kontrolne P1, P2, P3 i P3/4 (środek odcinka P3P4), punkty kontrolne drugiego to P3/4, P4, P5, P6. Fragmenty te są sklejone w punkcie P3/4. Ponieważ każdy z nich jest styczny w tym punkcie do odcinka P3P4, więc sklejenie jest gładkie. Po dodaniu do tego układu dalszych dwóch punktów P7 i P8 krzywa składałaby się już z trzech części o punktach kontrolnych odpowiednio P1, P2, P3, P3/4; P3/4, P4, P5, P5/6 oraz P5/6, P6, P7, P8.

Z opisanych własności algorytmu Timmera wynika, że dla danego ciągu punktów kontrolnych P1, P2,..., Pn (n parzyste) krzywa sklejana o tych punktach kontrolnych przecina się lub jest styczna do łamanej P1P2...Pn w środkach wszystkich jej odcinków z wyjątkiem pierwszego i ostatniego. Ponadto, przyjmując Pn=P1, otrzymamy krzywą zamkniętą. Jeśli dodatkowo przeprowadzimy punkty kontrolne P2 i P(n-1) tak, by P1 (=Pn) należał do odcinka P2P(n-1), to w P1 krzywa zamknie się też gładko, bez ostrza.

Moduł SPLINE.PAS przedstawiony na wydruku 1 pozwala na wykorzystywanie funkcji sklejanych w programach w Turbo Pascalu pracujących w trybie graficznym. Składa się on z trzech procedur. Jedna z nich, InitTable, znajduje się w części prywatnej modułu i służy do wypełnienia tablic funkcji łączących (jest wykonywana w części implementacyjnej modułu). Pozwala to uniknąć wielokrotnego obliczania tych samych współczynników, przyspieszając działanie pozostałych procedur Timmer i DrawCubic.

Procedura Timmer rysuje pojedynczą krzywą trzeciego stopnia z zadanymi czterema punktami kontrolnymi (parametrami procedury). Należą one do typu rekordowego PointType, zdefiniowanego w module standardowym Graph. Składa się on z dwóch pól x,y typu integer.

Do rysowania krzywych sklejanych z większą liczbą punktów kontrolnych służy procedura DrawCubic. Wywołuje się ją dokładnie tak samo, jak procedurę standardową DrawPoly używaną do rysowania łamanych. Ma ona dwa parametry. Pierwszy to liczba punktów kontrolnych, drugi jest nazwą tablicy zawierającej te punkty jako dane typu PointType. Pierwszy parametr musi być liczbą parzystą równą co najmniej 4.

W razie podania nieparzystej liczby punktów kontrolnych ostatni jest ignorowany. Liczba punktów kontrolnych może być mniejsza od rozmiaru tablicy (można go w programie zadeklarować „na zapas"), ale w żadnym wypadku nie może go przekraczać.

152

W części implementacyjnej (niejawnej) modułu zadeklarowana jest stała ile. Określa ona, ile punktów pojedynczej funkcji będzie obliczanych w programie. Można zastąpić jej wartość (przyjętą jako 100) inną liczbą. Zwiększenie stałej zwiększy precyzję rysunku, spowolni jednak wykonanie procedury i zwiększy ilość pamięci zajętej przez tablice funkcji łączących (24 bajty na punkt). Nadmierne zmniejszenie tej stałej może spowodować, że krzywa sklejana będzie przypominać wyglądem łamaną. Można dobrać optymalną wartość doświadczalnie w zależności od rodzaju rysowanych obiektów oraz karty graficznej i monitora.

Na wydruku 2 znajduje się program DEMO.PAS demonstrujący działanie modułu SPLINE oraz umożliwiający obejrzenie krzywych sklejanych o zadanych punktach kontrolnych. Program pozwala na wprowadzenie punktów kontrolnych w pozycji kursora (białego punktu) przez naciśnięcie klawisza Enter. Sterowanie położeniem kursora odbywa się za pomocą klawiszy Home, End, PgUp i PgDn (szybkie przesuwanie) oraz klawiszy kursora (precyzyjne sterowanie). Wprowadzanie punktów kończy naciśnięcie klawisza Esc. Na ekranie widzimy wtedy punkty kontrolne połączone w łamaną oraz krzywą sklejaną.

O jakości krzywych sklejanych świadczy między innymi dokładność, z jaką mogą one przybliżać typowe, często spotykane w grafice krzywe. W szczególności można zadać pytanie, jaką część okręgu można w zadowalający sposób przybliżyć jedną funkcją trzeciego stopnia. Opisany wyżej algorytm daje bardzo dobre przybliżenie dla ćwiartki okręgu. Aby się o tym przekonać, wystarczy wyświetlić odpowiednio dobraną krzywą trzeciego stopnia o parametrach podanych na wydruku 3 oraz okrąg (w różnych kolorach).

Artykuł ten nie wyczerpuje wszystkich zagadnień związanych z funkcjami sklejanymi. Wiele z nich, zwłaszcza różne inne algorytmy, uogólnienie na przypadek funkcji dwóch zmiennych (powierzchnie sklejane), analiza dokładności przybliżeń funkcjami sklejanymi oraz dalsze zastosowania w grafice komputerowej można znaleźć w podanej literaturze.

□

Literatura:

[1] R.L. Burden, J.D. Faires „Numerical Analysis", Prindle, Weber & Schmidt, Boston 1985.

[2] M. Jankowski „Elementy grafiki komputerowej", WN-T, Warszawa 1990.

[3] J. Lansdown „Grafika komputerowa", WN-T, Warszawa 1990.

[4] G. Mullineux „CAD: Computational Concepts and Methods", Macmillan Publ. Co, New York 1986.

[5] J. Stoer, R. Bulirsch „Wstęp do analizy numerycznej", PWN, Warszawa 1987.

```
{SPLINE.PAS}
(*******************************************************)
(*          FUNKCJE SKLEJANE TRZECIEGO STOPNIA         *)
(*             Krzysztof  Stefański 02/1994            *)
(* 60-959 Poznań 2, skr.poczt. 307 tel.(0-61)47-12-94  *)
(*******************************************************)
(* Podstawową procedurą modułu jest DrawCubic.         *)
(* Składnia identyczna z procedurą DrawPoly z modułu   *)
(* Graph: pierwszy parametr - liczba punktów (parzysta *)
(* i równa co najmniej 4), drugi parametr - tablica z  *)
(* punktami. Punkty przedstawione jako rekordy typu    *)
(* PointType z modułu Graph. Pierwszy parametr nie     *)
(* może przekroczyć rozmiarów tablicy (procedura tego  *)
(* nie sprawdza). Jeśli pierwszy parametr jest liczbą  *)
(* nieparzystą, ostatni punkt jest ignorowany.         *)
```

```
(* Dla czterech punktów można zamiast DrawCubic       *)
(* wywołać procedurę Timmer. Przed wywołaniem procedur *)
(* modułu system musi być w trybie graficznym.        *)
(*******************************************************)
unit spline;
interface
uses graph;
procedure Timmer(p1,p2,p3,p4: PointType);
procedure DrawCubic(NumPoints: word; var Points);
implementation
const ile=100; {liczba odcinków krzywej sklejanej}
      odwr=1/ile;
var f1,f2,f3,f4: array[1..ile] of real;
    {tablice funkcji łączących}
procedure InitTable; {oblicza funkcje łączące}
var k: word;
    t,tt,t1,t1t1,t12: real;
begin
  for k:=1 to ile do
  begin
    t:=odwr*k;
    tt:=Sqr(t);
    t1:=1-t;
    t1t1:=Sqr(t1);
    t12:=1-2*t;
    f1[k]:=t12*t1t1;
    f2[k]:=4*t*t1t1;
    f3[k]:=4*tt*t1;
    f4[k]:=-t12*tt
  end
end; {InitTable}
procedure Timmer(p1,p2,p3,p4: PointType);
var k: word;
begin
 MoveTo(p1.x,p1.y);
  for k:=1 to ile do
 LineTo(Round(f1[k]*p1.x+f2[k]*p2.x+f3[k]*p3.x+f4[k]*p4.x),
       Round(f1[k]*p1.y+f2[k]*p2.y+f3[k]*p3.y+f4[k]*p4.y))
end; {Timmer}
procedure DrawCubic(NumPoints: word; var Points);
var p: pointer;
    k,segment,ofset: word;
    tablica: array[1..4] of PointType;
begin
 p:=@Points;
 Move(p^,tablica,16);
 if (NumPoints=4) or (NumPoints=5) then
  Timmer(tablica[1],tablica[2],tablica[3],tablica[4])
 else
 if NumPoints>5 then
 begin
  segment:=Seg(Points);
  ofset:=Ofs(Points);
  {środek odcinka P3P4:}
 tablica[4].x:=(tablica[3].x+tablica[4].x) shr 1;
 tablica[4].y:=(tablica[3].y+tablica[4].y) shr 1;
 Timmer(tablica[1],tablica[2],tablica[3],tablica[4]);
  for k:=1 to (NumPoints-6) div 2 do
  begin
   {przesunięcie wskaźnika o 2 elementy tablicy:}
    Inc(ofset,8);
   p:=Ptr(segment,ofset);
   Move(p^,tablica,16);
   {środki skrajnych odcinków:}
  tablica[1].x:=(tablica[1].x+tablica[2].x) shr 1;
  tablica[1].y:=(tablica[1].y+tablica[2].y) shr 1;
  tablica[4].x:=(tablica[3].x+tablica[4].x) shr 1;
  tablica[4].y:=(tablica[3].y+tablica[4].y) shr 1;
 Timmer(tablica[1],tablica[2],tablica[3],tablica[4])
  end;
  Inc(ofset,8);
 p:=Ptr(segment,ofset);
 Move(p^,tablica,16);
 tablica[1].x:=(tablica[1].x+tablica[2].x) shr 1;
 tablica[1].y:=(tablica[1].y+tablica[2].y) shr 1;
 Timmer(tablica[1],tablica[2],tablica[3],tablica[4])
 end
end; {DrawCubic}
```

```
begin
  InitTable
end.
```

Wydruk 1.

```
{DEMO.PAS}
program CubicSplineDemo;
uses graph,crt,spline;
const ilepunktow=100; {max. rozmiar tablicy z punktami}
var karta,tryb: integer;
    k: word;
    koniec: boolean;
    p: PointType;
   tablica: array[1..ilepunktow] of PointType;
    znak: char;
begin
 karta:=Detect;
 InitGraph(karta,tryb,'');
 k:=0;
 koniec:=False;
 MoveTo(GetMaxX div 2,GetMaxY div 2);
 PutPixel(GetMaxX div 2,GetMaxY div 2,MaxColors);
 repeat
   znak:=ReadKey;
   case znak of
     #0:          {klawisz specjalny}
     begin
       znak:=ReadKey;
       case znak of
         {poruszanie kursorem:}
         #72: begin                    {Up}
                PutPixel(GetX,GetY,0);
                MoveTo(GetX,GetY-1);
                PutPixel(GetX,GetY,MaxColors);
              end;
         #75: begin                    {Left}
                PutPixel(GetX,GetY,0);
                MoveTo(GetX-1,GetY);
                PutPixel(GetX,GetY,MaxColors);
              end;
         #77: begin                    {Right}
                PutPixel(GetX,GetY,0);
                MoveTo(GetX+1,GetY);
                PutPixel(GetX,GetY,MaxColors);
              end;
         #80: begin                    {Down}
                PutPixel(GetX,GetY,0);
                MoveTo(GetX,GetY+1);
                PutPixel(GetX,GetY,MaxColors);
              end;
         {szybkie poruszanie kursorem:}
         #73: begin                    {PgUp}
                PutPixel(GetX,GetY,0);
                MoveTo(GetX,GetY-10);
                PutPixel(GetX,GetY,MaxColors);
              end;
         #71: begin                    {Home}
                PutPixel(GetX,GetY,0);
                MoveTo(GetX-10,GetY);
                PutPixel(GetX,GetY,MaxColors);
              end;
         #79: begin                    {End}
                PutPixel(GetX,GetY,0);
                MoveTo(GetX+10,GetY);
                PutPixel(GetX,GetY,MaxColors);
              end;
         #81: begin                    {PgDown}
                PutPixel(GetX,GetY,0);
                MoveTo(GetX,GetY+10);
                PutPixel(GetX,GetY,MaxColors);
              end;
       end;
     end;
     #27: koniec:=True;          {Esc - koniec}
     #13: begin                  {Enter - zaznaczenie}
            Inc(k);
            if k>1 then
          Line(GetX,GetY,tablica[k-1].x,tablica[k-1].y);
            tablica[k].x:=GetX;
            tablica[k].y:=GetY;
            Circle(GetX,GetY,4);
          end;
   end; {case}
 until koniec or (k=ilepunktow);
 ClearDevice;
 SetColor(LightRed);
 DrawCubic(k,tablica);
 SetColor(Yellow);
 DrawPoly(k,tablica);
 while KeyPressed do
 znak:=ReadKey;
 repeat until KeyPressed;
 CloseGraph;
end.
```

Wydruk 2.

```
const r=400;  {promień}
     p=0.41421356;  {sqrt(2)-1}
     dane: array [1..4,1..2] of integer=
    ((0,r),(Round(r*p),r),(r,Round(r*p)),(r,0));
.............
  SetColor(...);
  DrawCubic(4,dane);
  SetColor(...);
  Circle(0,0,r);
.............
```

Wydruk 3.