Rastrové algoritmy pro výpočet izočar

Josef Pelikán

KSVI MFF UK Praha,
Malostranské nám. 25,
11800 Praha 1,
e-mail: pelikan@cspguk11

Abstrakt

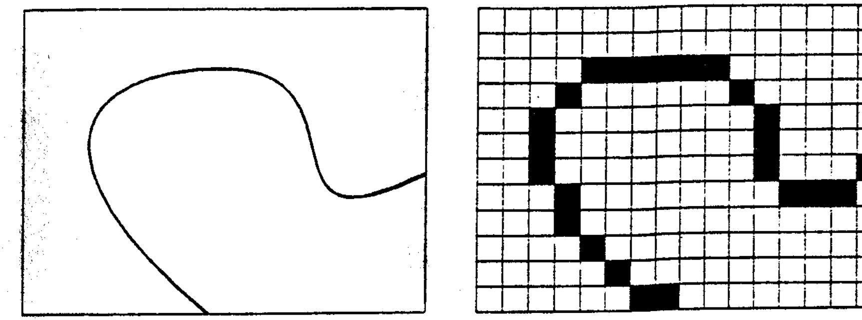
Příspěvek obsahuje popis několika algoritmů pro výpočet izočar v prostředí rastrového výstupního zařízení. Algoritmy byly vyvinuty na základě praktických požadavků při zobrazování naměřených dat v hydrobiologii. Základními požadavky byla univerzálnost použití, velká rychlost výpočtu a velká přesnost. Byly vyzkoušeny dva principy výpočtu: rekurzivní dělení oblasti a trasování jednotlivých izočar, hledání izočar se provádí rekurzivním dělením. Algoritmy dosahují velké přesnosti, chyba je srovnatelná s velikostí rastru. Příspěvek obsahuje kromě popisu jednotlivých algoritmů i diskusi o jejich rychlosti a možnostech dalšího vylepšení.

Úvod

Zobrazování průběhu funkce dvou proměnných pomocí sítě izočar se v praxi často používá, protože je dostatečně přesné a názorné. Cílem tohoto příspěvku je ukázat jeden méně tradiční přístup k řešení problémů při vyhledávání a počítání izočar. Uváděné algoritmy lze po zobecnění použít i k jiným účelům, např. k trasování obrysů dvojrozměrných nebo třírozměrných oblastí, pro jednoduchost a názornost se však v textu budeme držet dvojrozměrného případu.

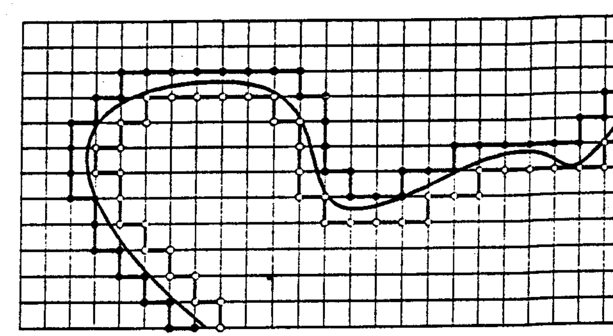
Nejprve zavedeme několik základních pojmů - budeme předpokládat, že máme zobrazovat funkci dvou proměnných $f(x,y)$ na nějaké oblasti O z R^2. **Izočárou s hodnotou *h*** obvykle nazýváme křivku spojující body oblasti O, které mají funkční hodnotu $f(x,y) = h$. Ve většině praktických aplikací se setkáváme se spojitými fukcemi, my však zde nebudeme požadovat spojitost f, a proto musíme zavést pojem **zobecněné izočáry**: bude to křivka obsahující jednak body $[x,y]$, ve kterých je funkce f spojitá a nabývá hodnoty h, a také body nespojitosti funkce f s funkční hodnotou $h' >= h$, v jejichž okolí se vyskytují body s funkční hodnotou menší než h. Co to znamená: představíme-li si funkci f jako závislost výšky terénu na souřadnicích v půdorysu, bude takto definovaná izočára procházet rovněž "svislými stěnami", které obsahují body s výškou h. Zobecněné izočáry budeme v dalším textu též nazývat pro jednoduchost izočarami.

Nalézt přesný matematický popis tvaru izočáry může být velice komplikované i v případě relativně jednoduchých funkcí f. Z hlediska počítačové grafiky však není třeba hledat absolutně přesné řešení, protože se křivka tak jako tak zobrazuje na nepřesném výstupním zařízení; nejčastěji v diskrétních rastrových souřadnicích. Naší snahou bude nalézt algoritmus dávající dobré výsledky při výstupu na rastrové zařízení. Pro zjednodušení budeme používat diskrétní rastrové souřadnice s předpokladem, že **pixel** je čtvercový a má velikost **1x1** (obecnější případy lze na tento snadno převést lineární transformací souřadnic). Ideální izočára se jako každá křivka musí před zobrazením převést do těchto souřadnic a pak se nakreslí jako posloupnost pixelů, jak ukazuje následující obrázek:



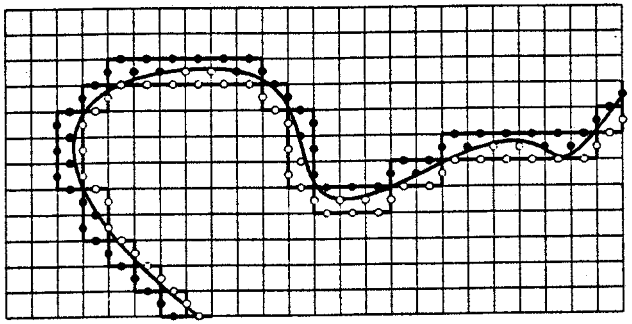
Obr. 1

Budeme-li chápat izočáru jako její rastrovou reprezentaci, můžeme ji popsat definicí zcela jiného charakteru: **izočára** bude množina nebo posloupnost pixelů splňujících nějakou vhodnou podmínku P. Pro začátek můžeme zkusit použít jednoduchou **podmínku** P_1: pixel o souřadnicích $[x,y]$ je součástí izočáry s hodnotou h, jestliže v jeho rozích nabývá funkce f menší i větší hodnoty než h. **Přesně řečeno** - platí nerovnost $min\ \{f(x,y), f(x+1,y), f(x,y+1), f(x+1,y+1)\} < h <= max\ \{f(x,y), f(x+1,y), f(x,y+1), f(x+1,y+1)\}$. Na dalším obrázku je znázorněn průběh ideální izočáry a tlustou čarou jsou ohraničeny pixely, které vyhovují podmínce P_1 (tečkami jsou znázorněny testovací body - plné tečky odpovídají hodnotě $f>=h$ a prázdné tečky hodnotě $f<h$):



Obr. 2

Tvar rastrové reprezentace křivky, který dostaneme použitím podmínky P_1, však neodpovídá našim představám; chtěli bychom z izočáry odstranit "rohové" pixely zdůrazňující zubatý tvar křivky! Zavedeme proto jinou **podmínku** P_2 využívající čtyř testovacích bodů uprostřed stran pixelu. P_2: pixel o souřadnicích $[x,y]$ je součástí izočáry s hodnotou h, jestliže ve středech jeho stran nabývá funkce f hodnoty menší i větší než h. Přesněji řečeno - platí nerovnost $min\ \{f(x+1/2,y), f(x,y+1/2), f(x+1,y+1/2), f(x+1/2, y+1)\} < h <= max\ \{f(x+1/2,y), f(x,y+1/2), f(x+1,y+1/2), f(x+1/2, y+1)\}$. Na obrázku je nakreslena rastrová reprezentace izočáry při použití podmínky P_2:



Obr. 3

Ke kreslení izočar podle výše uvedených definic lze použít několika různých algoritmů; v následujících odstavcích se budeme zabývat dvěma principiálně odlišnými přístupy: rekurzivním dělením rovinné oblasti a inkrementálním trasováním jednotlivých izočar.

Algoritmus 1: rekurzivní dělení roviny

Tento algoritmus lze použít v případech, kdy není třeba získat informaci o průběhu jednotlivých izočar. Algoritmus vyprodukuje pouze rastrový obrázek obsahující izočáry zadaných hodnot, informace o průběhu křivek však chybí a nelze tedy přímo izočáry trasovat (procházet). Algoritmus je založen na myšlence rekurzivního dělení čtvercové oblasti tehdy, když tato oblast může obsahovat část izočáry. Dělení končí v případě, že má zkoumaná oblast velikost jednoho pixelu (a pak rozhodne podmínka **P** o vybarvení tohoto pixelu) nebo tehdy, když oblast pravděpodobně žádnou část izočáry neobsahuje. Při zkoumání čtvercových oblastí roviny použijeme modifikace podmínek P_i na čtverec o straně n. Místo podmínky P_i pro jeden pixel tak budeme užívat podmínku P_i' pro čtverec nxn pixelů. Parametrem algoritmu je **přesnost prohledávání** *c* - číslo udávající rozměr největšího čtverce, který již nebudeme dělit tehdy, jestliže na něm nebude splněna podmínka **P'**.

Při popisu algoritmu pro jednoduchost předpokládejme, že zobrazujeme jedinou izočáru s hodnotou h, že zkoumaná oblast má tvar čtverce s levým horním rohem v bodě $[x_0,y_0]$ a se stranou délky $n = 2^k$ a že funkce f je na tomto čtverci všude definována. Následuje popis **algoritmu** 1a používajícího podmínek P_1 a P_1' a dělení čtverce na čtvrtiny:

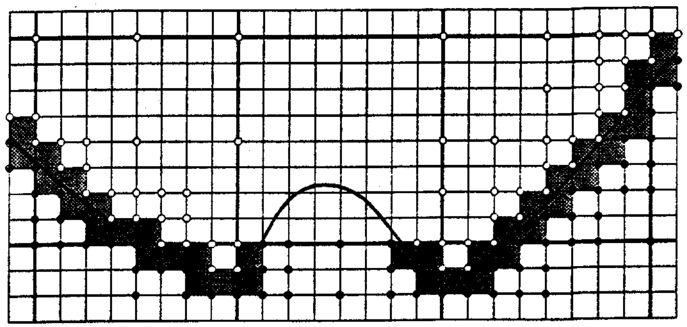
```
Algoritmus 1a;

Vstup: function f ( x, y : real ) : real; { funkce f }
       h : real;                          { hodnota izočáry }
       x0, y0, n : integer;               { čtvercová oblast }
       c : integer;                       { přesnost prohledávání }

  procedure Rozděl ( x, y : integer; strana : integer; { čtverec }
                     r1, r2, r3, r4 : boolean ); { čtyři testované rohy }
   { prozkoumá čtverec s vrcholem v bodě [x,y] a délkou strany 'strana',
     'ri' jsou znaménka funkce f v rozích čtverce }
  var  r5, r6, r7, r8 : boolean; { rohy menších čtverců }

   function Dělit ( r1, r2, r3, r4 : boolean ) : boolean;
   begin
     Dělit := (strana>c) { ještě jsem nedosáhl přesnosti prohledávání }
             or (r1<>r2) or (r1<>r3) or (r1<>r4); { znaménka se liší }
   end;
```



```
  begin { Rozděl }
    if strana=1 then PutPixel(x,y) { pixel patří do izočáry }
               else
     begin { budu dělit }
       strana := strana div 2;
       r5 := f(x+strana,y)>=h;              { střed horní strany }
       r6 := f(x,y+strana)>=h;              { střed levé strany }
       r7 := f(x+strana,y+strana)>=h;       { střed čtverce }
       if Dělit(r1,r5,r6,r7) then Rozděl(x,y,strana,r1,r5,r6,r7);
       r8 := f(x+2*strana,y+strana)>=h;     { střed pravé strany }
       if Dělit(r5,r2,r7,r8) then Rozděl(x+strana,y,strana,r5,r2,r7,r8);
       r5 := f(x+strana,y+2*strana)>=h;     { střed dolní strany }
       if Dělit(r6,r7,r3,r5) then Rozděl(x,y+strana,strana,r6,r7,r3,r5);
       if Dělit(r7,r8,r5,r4) then Rozděl(x+strana,y+strana,strana,r7,r8,r5,r4);
     end;
  end; { Rozděl }

begin { algoritmus 1a }
  Rozděl(x0,y0,n,f(x0,y0)>=h,f(x0+n,y0)>=h,f(x0,y0+n)>=h,f(x0+n,y0+n)>=h);
end;
```

Snadno můžeme algoritmus upravit tak, aby kreslil současně izočáry několika různých hodnot nebo pracoval na obecnější oblasti *O*. Jednoduchou úpravou můžeme též dostat algoritmus, který nekreslí izočáry, ale znázorňuje průběh funkce f vybarvením (oblast, na které je $h_i <= f < h_{i+1}$, se vybarví stejnou barvou i).

Obdobného principu lze využít při konstrukci **algoritmu 1b** pracujícího s jinou definicí izočáry pomocí podmínky P_2. Testovací body pak budou ležet ve středech stran čtvercové oblasti a čtverec se v případě potřeby bude dělit na **devět** menších. Nakreslené křivky budou mít lepší tvar, ale hlavní nevýhoda vyplývající z nepřesného vyhledávání izočáry odstraněna nebude. Tato nevýhoda spočívá v možnosti vynechání části izočáry tam, kde se vyskytují jemné detaily (zákruty) velikosti maximálně *c*. Následující obrázek ukazuje příklad takové nepřesnosti při použití algoritmu **1a**:



Obr. 4

Jestliže můžeme explicitní znalostí funkce f vyloučit možnost tvarových detailů izočar řádově menších než *c*, pracují algoritmy **1a** i **1b** uspokojivě.

Nyní se krátce zastavíme u **efektivity** algoritmů rekurzivního dělení. Budeme předpokládat, že časově nejnáročnější operací je vyvolání funkce f, a proto budeme efektivitu algoritmu měřit relativně jako "počet vyvolání funkce f na jeden pixel délky izočáry". Uvedeme zde pouze výsledky, podrobný postup výpočtu efektivity by byl příliš obsáhlý. Při kreslení jediné málo členité izočáry algoritmem **1a** se dosahuje časové složitosti **5**

až 7 (volání *f* / pixel). V průměru tedy vychází složitost 6 a při současném kreslení několika izočar lze očekávat mírné zrychlení (např. při průměrné vzdálenosti izočar 8 pixelů se střední složitost zmenší na 5.4). Při kreslení jedné izočáry algoritmem **1b** se dosahuje průměrné časové složitosti **8.5**, při kreslení více izočar s průměrnou vzdáleností 9 pixelů se složitost zmenší na **8.0**.

Ve srovnání s naivním algoritmem ověřujícím podmínku P pro každý pixel oblasti jsou algoritmy rekurzivního dělení mnohem rychlejší, jsou však pomalejší než algoritmy trasování izočar popisované v následujícím odstavci.

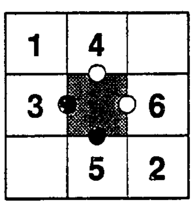
Algoritmus 2: trasování izočar

Algoritmy trasování izočáry jsou založeny na jiném principu: procházejí postupně izočáru a generují ji jako posloupnost pixelů, z nichž každé dva následující spolu v rovině sousedí. Tyto algoritmy je vhodné použít v případě, že je třeba získat informaci o průběhu křivky (např. při kreslení na plotru).

Základní myšlenka algoritmu: předpokládejme, že máme pixel o souřadnicích $[x_0,y_0]$, který patří do izočáry (je na něm splněna podmínka P). Pak musí existovat alespoň dva ze sousedních osmi pixelů, na nichž podmínka P rovněž platí. Vybereme si jeden z nich, přejdeme do něj a celou úvahu budeme opakovat s tím, že se nikdy nebudeme vracet do pixelů v nichž jsme již byli. Trasování končí tehdy, když se vrátíme do výchozího pixelu $[x_0,y_0]$ (izočára se uzavřela) nebo když opustíme oblast *O*. Zajímavý zůstává jen způsob, jak efektivně vybírat sousední pixely vyhovující podmínkce P - cílem je, aby trasovací algoritmus testoval co nejméně "nepravých" sousedů - aby se co nejméně mýlil. Z toho samozřejmě plyne i minimální počet volání funkce *f*.

Mechanismů jak vybírat potenciální následníky je mnoho, použijeme-li čistě kombinatorický přístup, můžeme např. pro jednotlivé kombinace hodnot v testovacích bodech vytvořit tabulku, ve které budou uloženy ve vhodném pořadí potenciální následníci. Tato metoda je použita v následujícím popisu trasovacího algoritmu. Jiný postup by mohl být založen na přibližném výpočtu diferenciálu funkce *f* v aktuálním bodě $[x,y]$ a podle výsledku by se zjistilo, ve kterém směru bude pravděpodobně izočára pokračovat.

Následuje popis **algoritmu 2b** využívajícího definice izočáry pomocí podmínky P_2. Algoritmus dostane jako vstup souřadnice pixelu $[x_0,y_0]$, na kterém je P_2 splněna, a produkuje posloupnost pixelů tvořících izočáru. Při vybírání následníků pomáhá tabulka **TAB**, která pro každou kombinaci testovacích hodnot obsahuje pořadí prohledávání sousedů. Příklad jedné položky tabulky **TAB** (opět jsou testovací body s hodnotou *f>=h* označeny plnou tečkou a body s hodnotou *f<h* prázdnou tečkou):



Obr. 5


```
Algoritmus 2b;

Vstup: function f ( x, y : real ) : real;  { funkce f }
       h : real;                           { hodnota izočáry }
       x0, y0 : integer;                   { počáteční pixel }

type Směry = ( S, SZ, Z, JZ, J, JV, V, SV, zpátky, nic ); { směry postupu }

const TAB : array [ boolean, boolean, boolean, boolean ] of
              { kombinace hodnot v testovacích bodech: true = f>=h }
            array [ 1 .. 8 ] of Směry = ...
              { pořadí prohledávání }

var  x, y : integer; { souřadnice aktuálního pixelu }
     PosledníSměr : Směry;

  function DalšíPixel : Směry;
    { vrací 'zpátky' jestliže se izočára vrátila do bodu [x0,y0] a
      'nic' jestliže opustila oblast O; mění souřadnice pixelu [x,y] }
  var  Možnosti : array [ 1 .. 8 ] of Směry; { které směry budu zkoušet }
       n : 1 .. 8;
       Nahoře : boolean; { výsledek testu v polovině horní strany pixelu }
  begin
    DalšíPixel := zpátky;
    Možnosti := TAB[f(x+1/2,y)>=h,f(x,y+1/2)>=h,f(x+1,y+1/2)>=h,f(x+1/2,y+1)>=h];
    for n := 1 to 8 do { zkouším maximálně osm sousedů }
      if (Možnosti[n]<>nic) and <ještě jsem v tom pixelu nebyl> then
        begin { zkouším jednoho kandidáta }
          Nováx := <posunutí ve směru Možnosti[n]>;
          Nováy := <posunutí ve směru Možnosti[n]>;
          Nahoře := f(Nováx+1/2,Nováy)>=h;
          if <pixel [Nováx,Nováy] je v oblasti O> and
             ( (Nahoře<>(f(Nováx,Nováy+1/2)>=h) or
               (Nahoře<>(f(Nováx+1,Nováy+1/2)>=h) or
               (Nahoře<>(f(Nováx+1/2,Nováy+1)>=h) ) then
            if (Nováx=x0) and (Nováy=y0) then exit  { uzavřel jsem izočáru }
                                         else
              begin { postupuji na další pixel }
                x := Nováx; y := Nováy;
                DalšíPixel := Možnosti[n];
                exit;
              end;
        end;
    DalšíPixel := nic;
  end; { DalšíPixel }

begin { Algoritmus 2b }
  x := x0; y := y0;
  <ZačátekIzočáry(x,y)>;
  PosledníSměr := DalšíPixel;   { ve kterém směru jsem vykročil }
  if PosledníSměr<>nic then
    begin
      repeat
        PosledníSměr := DalšíPixel;
        if PosledníSměr<=SV then <Posunutí(PosledníSměr)>;
      until PosledníSměr>SV;
      if PosledníSměr<>zpátky then
        begin { trasuji druhým směrem z počátku }
          x := x0; y := y0;
          <ZačátekDruhéVětveIzočáry(x,y)>;
          repeat
            PosledníSměr := DalšíPixel;
            if PosledníSměr<=SV then <Posunutí(PosledníSměr)>;
          until PosledníSměr>SV;
        end;
    end;
  <KonecIzočáry>;
end; { Algoritmus 2b }
```

Podobně by vypadal **algoritmus 2a** založený na podmínce P_1 a využívající čtyřech testovacích bodů v rozích pixelu, uvádět ho zde nebudeme. Pokud budeme porovnávat kvalitu výstupu algoritmů **2a** a **2b**, zjistíme, že varianta **2b** produkuje požadovaný tvar rastrové reprezentace izočar (viz obrázek 3), kdežto varianta **2a** počítá výstup obsahující přebytečné "rohové" pixely (viz obrázek 2). Protože se však izočáry počítají postupně, lze snadno tuto závadu algoritmu **2a** odstranit zařazením jednoduchého "filtru" za jeho výstup. Tento filtr musí každou dvojici kolmých směrů (např. .., S, Z, ..) nahradit odpovídajícím šikmým směrem (.., SZ, ..). Při použití takového filtru již dává algoritmus **2a** uspokojivé výsledky.

Efektivita trasovacích algoritmů je obecně vyšší než efektivita algoritmů rekurzivního dělení. Kdybychom počítali ideální časovou složitost algoritmu **2b** (za předpokladu, že vyhledávací strategie se nikdy nesplete a nebude se počítat ani jeden zbytečný testovací bod), vyjde hodnota 2.9 volání *f* na jeden pixel délky izočáry. Pro algoritmus **2a** vychází obdobná ideální složitost 2.4. Praktická měření ukázala, že algoritmus **2b** s vyhledávací tabulkou **TAB** dosahuje průměrné složitosti 3.4 a algoritmus **2a** dokonce téměř ideální složitosti 2.5. Velká rychlost je tedy podstatnou výhodou algoritmů pro trasování izočar; k řešení zbývá ještě problém, kterým se budeme zabývat v následujícím odstavci.

Algoritmus 3: vyhledávání izočar

Algoritmy popsané v předchozím odstavci potřebují zadat jako vstupní parametr bod $[x_0,y_0]$, který je součástí izočáry a ze kterého začínají s trasováním. Algoritmus vyhledávající tyto počáteční body by měl splňovat dva přirozené požadavky: měl by najít co nejvíc izočar vyskytujících se v zadané oblasti O (nejraději všechny) a neměl by zadat dva body ze stejné izočáry. Triviální algoritmus testující všechny pixely oblasti O není prakticky použitelný pro svou velkou časovou složitost. Při znalosti vlastností konkrétní funkce *f* můžeme někdy použít vhodnou speciální metodu, nás teď bude spíše zajímat **univerzální algoritmus** použitelný v obecném případě. Uspokojivé výsledky dává ve většině případů modifikace **algoritmu 1a** nebo **1b**, která nepočítá celé izočáry, ale pouze hledá počáteční body $[x_0,y_0]$.

Uvedeme si nyní **algoritmus 3a**, který vyhledává počáteční pixely pro trasovací algoritmus **2a**. Několikanásobné zadávání pixelů jedné izočáry je znemožněno tím, že se pro každou **buňku** velikosti *c* (viz algoritmus 1a) uchovává seznam hodnot izočar, které již v této buňce byly trasovány. Vyhledávací algoritmus již v této buňce pixely zaznamenaných izočar nehledá. Opět budeme pro jednoduchost předpokládat, že oblast O je čtverec s vrcholem $[x_0,y_0]$ a stranou délky $n=2^k$.

```
Algoritmus 3a;

Vstup: function f ( x, y : real ) : real; { funkce f }
       h : array [ 1 .. Max1 ] of real;   { hodnoty hledaných izočar }
       Počet : integer;                   { počet hodnot v poli 'h' }
       x0, y0, n : integer;               { čtvercová oblast }
       c : integer;                       { přesnost prohledávání }

var  Buňky : array [ 0 .. Max2, 0 .. Max2 ] of
             set of 1 .. Max1;
       { pro každou buňku seznam netrasovaných izočar -
         - do tohoto pole musí zapisovat procedura trasující izočáry! }

  function Hodnota ( f : real ) : integer;  { neimplementována }
    { vrací číslo 'i' intervalu, do kterého padne f:
      h[i] <= f < h[i+1] }
```

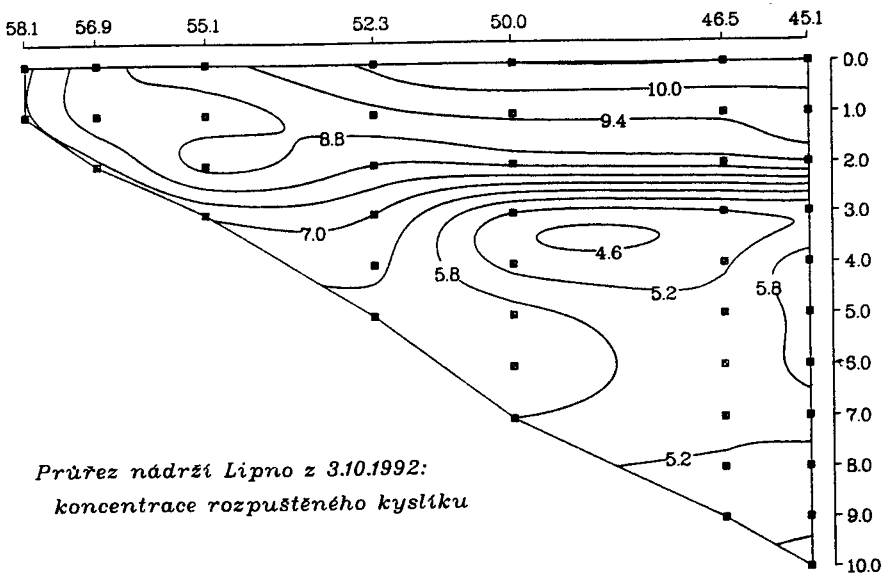


```
  procedure Rozděl ( x, y : integer; strana : integer; { čtverec }
                     r1, r2, r3, r4 : integer ); { čtyři testované rohy }
   { prozkoumá čtverec s rohem v bodě [x,y] a délkou strany 'strana',
     'ri' jsou intervaly funkce f v rozích čtverce }
  var  r5, r6, r7, r8 : integer; { rohy menších čtverců }
       i : integer;              { hodnota trasované izočáry }

   function Dělit ( r1, r2, r3, r4 : integer;
                    x, y : integer ) : boolean;
   var  Rmin, Rmax : integer; { minimální a maximální 'ri' }
   begin
     Rmin := min(r1,r2,r3,r4);
     Rmax := max(r1,r2,r3,r4);
     Dělit := (strana>c) { ještě jsem nedosáhl přesnosti prohledávání }
              or ( (Rmax<>Rmin) { hodnoty v rozích se liší }
                   and ([Rmin+1..Rmax]*Buňky[(x-x0) div c,(y-y0) div c]<>[]) );
                   { v aktuální buňce zbylo něco ke trasování }
   end;

  begin { Rozděl }
   if strana=1 then
     begin { budu něco trasovat }
       for i := min(r1,r2,r3,r4)+1 to max(r1,r2,r3,r4) do
         if i in Buňky[(x-x0) div c,(y-y0) div c] then
           <trasuj izočáru s hodnotou h[i] z bodu [x,y]>
     end       else
     begin
       strana := strana div 2;
       r5 := Hodnota(f(x+strana,y));            { střed horní strany }
       r6 := Hodnota(f(x,y+strana));            { střed levé strany }
       r7 := Hodnota(f(x+strana,y+strana));     { střed čtverce }
       if Dělit(r1,r5,r6,r7,x,y) then
         Rozděl(x,y,strana,r1,r5,r6,r7);
       r8 := Hodnota(f(x+2*strana,y+strana));   { střed pravé strany }
       if Dělit(r5,r2,r7,r8,x+strana,y) then
         Rozděl(x+strana,y,strana,r5,r2,r7,r8);
       r5 := Hodnota(f(x+strana,y+2*strana));   { střed dolní strany }
       if Dělit(r6,r7,r3,r5,x,y+strana) then
         Rozděl(x,y+strana,strana,r6,r7,r3,r5);
       if Dělit(r7,r8,r5,r4,x+strana,y+strana) then
         Rozděl(x+strana,y+strana,strana,r7,r8,r5,r4);
     end;
  end; { Rozděl }

begin { algoritmus 3a }
  <naplň celé pole 'Buňky' hodnotami [1..Max1]>
  Rozděl(x0,y0,n,Hodnota(f(x0,y0)),Hodnota(f(x0+n,y0)),
                 Hodnota(f(x0,y0+n)),Hodnota(f(x0+n,y0+n)));
end;
```

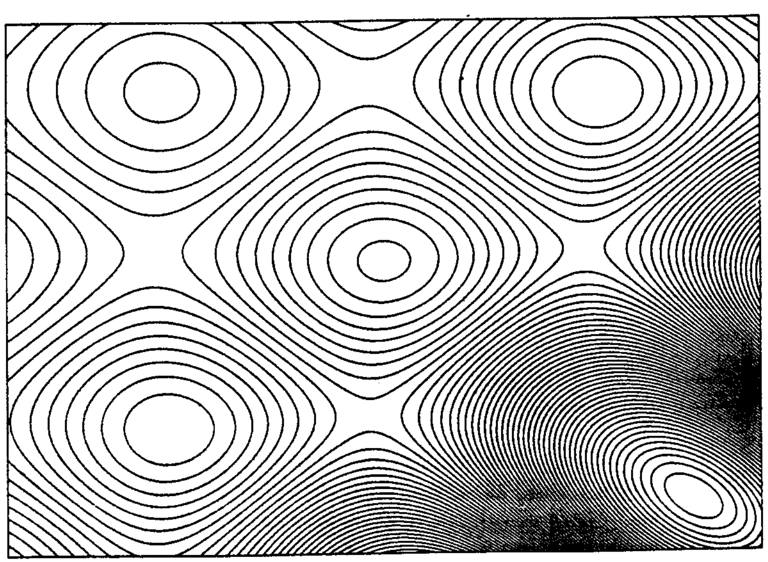
Z popisu algoritmu **3a** plyne, že nemohou být nalezeny pouze takové izočáry, které ohraničující oblast, do níž nepadne ani jeden rohový bod buňky (bod čtvercové sítě se stranou *c*). K opakovanému zadání stejné izočáry nemůže dojít. Při vhodné volbě parametru *c* je přesnost hledání izočar dostatečná.

Praktické výsledky

Popisované algoritmy byly implementovány v systému **IZO** pro počítače řady IBM PC, který se připravuje pro použití v hydrobiologii. V současné verzi se tam využívají algoritmy **2a**, **3a** a algoritmus **1a** upravený na vybarvování ploch, varianty algoritmů **2b** a **3b** byly zavrženy pro menší efektivitu. Na místě funkce *f* se používá parametrizovaná verze bikubické B-spline aproximace. V běžných podmínkách (při rozlišení řádově 500x400 na obrazovce a 2000x1500 na laserové tiskárně) počítají tyto algoritmy dostatečně rychle. Na obrázku je ukázka jedné z testovacích funkcí a ukázka finálního výstupu z programu **IZO**:



Průřez nádrží Lipno z 3.10.1992: koncentrace rozpuštěného kyslíku



Závěr

Byly popsány dva odlišné principy hledání a výpočtu izočar v diskrétních souřadnicích. Obecné vlastnosti všech algoritmů jsou: velká přesnost (chyba nemůže být větší než jeden pixel), obecnost (nepožaduje se spojitost zobrazované funkce *f*), jednoduchost a velká rychlost (nepoužívají se výpočty derivací ani žádné iterační numerické postupy). Nejrychlejší popisovaný algoritmus dosahuje v běžných podmínkách relativní časové složitosti **2.5** (výpočtu hodnoty funkce *f* na jeden pixel délky izočáry).

Díky vlastnostem uvedených algoritmů existuje mnoho prostoru pro jejich další úpravy a vylepšování. Můžeme zde uvést alespoň některé myšlenky: **A)** v případě polynomiální funkce *f* by stálo za pokus použít vhodného diferenčního schématu k výpočtu hodnot *f* zejména při použití trasovacích algoritmů. Pokud to podmínky dovolují, je velmi vhodné použít k výpočtu funkce *f* celočíselnou aritmetiku. **B)** Jednoduše lze počítat a kreslit vyhlazené izočáry - stačí pouze zvětšit rozlišení, spočítat izočáry v subpixelových souřadnicích a pomocí vhodného filtru kresbu převést do původního rastrového systému. Časová složitost algoritmů pro výpočet izočar se zvětšuje pouze lineárně se zvětšením rozlišení. **C)** Někdy by mohlo být lepší pracovat s úspornější vektorovou reprezentací izočar. V takovém případě by se zřejmě dal sestrojit algoritmus aproximující nějakou analytickou metodou rastrově spočítané izočáry. Vektorově reprezentované izočáry by se mohly libovolně zmenšovat a zvětšovat. **D)** Při nutnosti změny rozlišení v rastrové reprezentaci lze použít algoritmus, který využívá výsledků v původním rozlišení a dopočítává pouze některé hodnoty - není třeba vůbec izočáry vyhledávat...