```
    rxm=0.; rym=0.; r2m=0.;             // calculate CM position and Rg
    if( OUTNXT == sum_particles ){      //    if next output time
       OUTNXT += DNOUT;                 // increase for the next output
       for(int i=1; i<=NL; i++ ){
         for(int j=1; j<=NL; j++ ){
           if(world[i][j] > 0){
             rxm+=i;                    // x-mean
             rym+=j;                    // y-mean
           }
         }
       }
       rxm /= sum_particles;            // center of mass position x
       rym /= sum_particles;            // center of mass position y

       for(int i=1; i<=NL; i++ ){
         for(int j=1; j<=NL; j++ ){
           if(world[i][j] > 0){
             rij=(i-rxm)*(i-rxm)
                +(j-rym)*(j-rym);
             r2m+=rij;                 // average of the variance
           }
         }
       }
       r2m = sqrt(r2m/sum_particles);   // compute the radius of gyration

       cout << sum_particles << ' ' << rxm << ' ' << rym << ' '
            << r2m << ' ' << rmax << ' ' << '\n';
    }
  }
  outfile1.close();
  return 0;
}
```

5.7.4 Fraktale durch Iterationen

Eine elegante Möglichkeit Fraktale zu erzeugen ist die von Barnsley vorgeschlagene Methode iterativer Funktionensysteme. Angefangen mit einer Zufallskoordinate werden durch wiederholte, verschachtelte lineare Abbildungen immer weitere Punkte erzeugt, die in ihrer Gesamtheit schließlich ein fraktales Bild ergeben. Wie bei vielen ähnlichen Mehtoden läßt man den Algorithmus zuerst "warmlaufen", bevor man die Punkte zeichnet. Vier mit dieser Methode erzeugte Farne sind in Abb. 5.16 dargestellt. Der Unterschied zwischen den einzelnen Bildern sind nur die eingespeisten Abbildungen – es wurde jeweils derselbe Algorithmus verwendet.

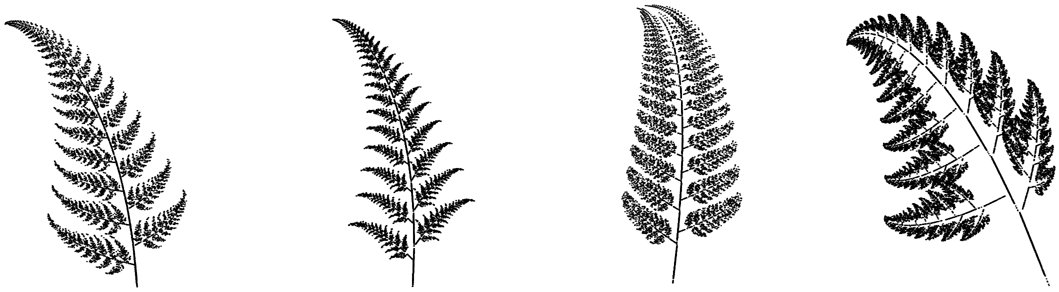

Abbildung 5.16: Vier verschiedene Farne – erzeugt durch verschiedene Iterationen.

Die Abbildungen für Farne sind einigermassen kompliziert. Deshalb wird hier am Beispiel des bekannten Sierpinski Gitters die Erzeugung durch Iteration genauer diskutiert. Der Kern der Iteration besteht aus der linearen Abbildung

$$\mathbf{r}' = \mathcal{D}\mathbf{r} + \mathbf{C} \ , \tag{5.54}$$

wobei $\mathbf{r}$ der Ausgangspunkt (x, y) und $\mathbf{r}$' der seine Abbildung ist. Die Abbildung ist zusammengesetzt aus einem Verschiebungsvektor $\mathbf{C}$ und einer Transformationsmatrix D. Eine Iteration besteht dabei aus einer von `NA` linearen Abbildungen.

Für das Sierpinki Gitter braucht man die drei Abbildungen:

$$\mathcal{D}_1 = \begin{pmatrix} 0.50194129421 & 0 \\ 0 & 0.50194129421 \end{pmatrix} \text{ und } \mathbf{C}_1 = \begin{pmatrix} 152 \\ 7 \end{pmatrix} , \tag{5.55}$$

$$\mathcal{D}_2 = \begin{pmatrix} 0.50184091097 & 0 \\ 0 & 0.50184091097 \end{pmatrix} \text{ und } \mathbf{C}_2 = \begin{pmatrix} 36 \\ 191 \end{pmatrix} \tag{5.56}$$

und

$$\mathcal{D}_3 = \begin{pmatrix} 0.49697157843 & 0 \\ 0 & 0.49697157843 \end{pmatrix} \text{ und } \mathbf{C}_3 = \begin{pmatrix} 265 \\ 192 \end{pmatrix} . \tag{5.57}$$

Im wesentlichen sind das also drei Streckungen gekoppelt mit drei Verschiebungen. Das aus dieser Abbildung nach 40,000 Iterationen (bei 500 Aufwärm-Iterationen) entstehende Fraktal ist in Abbildung 5.17 (links) dargestellt. Die Abbildungen werden nach zwei Prinzipien ausgewählt. Einmal, mit 10-prozentiger Wahrscheinlichkeit, wählt man einfach zufällig eine der `NA` Abbildungen, oder man benutzt, mit 90-prozentiger Wahrscheinlichkeit, die linearen Abbildungen gewichtet nach der Determinante von $\mathcal{D}$.

Verwendet man anstelle der drei Abbildungen nur die ersten beiden, so erhält man lediglich eine Gerade. Die drei Verschiebungsvektoren geben im wesentlichen die Dreiecksstruktur vor, während die Transformationsmatrizen lediglich um den Faktor zwei verkürzen. Hängt man die vierte Abbildung:

$$\mathcal{D}_4 = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix} \text{ und } \mathbf{C}_4 = \begin{pmatrix} 100 \\ 230 \end{pmatrix} \tag{5.58}$$

an die drei anderen, so ergibt sich die Projektion des dreidimensionalen Sierpinski Gitters, wie in Abb. 5.17 (rechts) dargestellt.

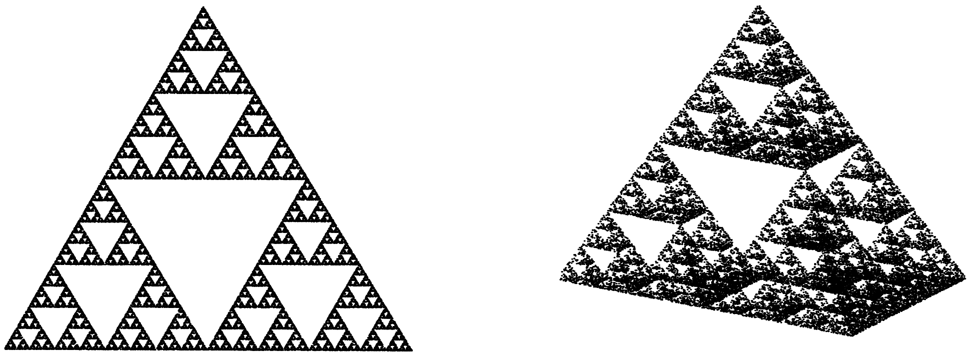

Abbildung 5.17: Das Sierpinski Gitter, erzeugt durch iterative Abbildung (links), und sein großer Bruder der durch vier hintereinandergeschaltete lineare Abbildungen erzeugt wurde.

```
#include<iostream>
#include<fstream>
using std::cout;
using std::cin;
using std::ofstream;
using std::ifstream;
#include"rng2.h"

static const int NDMAX=100;                      // max. number of ...
static double D[NDMAX][2][2], C[NDMAX][2];       // linear transformation
static double P[NDMAX];                          // det. weights

double determinants( int NA );
int iteration( int NA, double &x, double &y );

int main()
{
   char cinfile[20];
```

```
   int NA, n1=1, n2=100;
   double sumdets=0.0;
   double x, y;

   cout << "# Inputfile ? ";
   cin >> cinfile;
   cout << " ... read from " << cinfile << '\n';
   cout << "# Iterations ? ";
   cin >> n2; cout << n2 << '\n';

   ifstream detfile(cinfile);
   detfile >> NA;
   cout << "# " << NA << " determinants ... \n";
   for( int i=0; i<NA; i++ ){
     cout << "# det " << i << ' ';
     for( int j=0; j<2; j++ ){
       for( int k=0; k<2; k++ ){
         detfile >> D[i][j][k];
         cout << D[i][j][k] << ' ';
       }
     }
     for( int j=0; j<2; j++ ){
       detfile >> C[i][j];
       cout << C[i][j] << ' ';
     }
     cout << '\n';
   }
   detfile.close();                           // end of reading data

   rng0(12345678);
   rng(); rng(); rng(); rng();                // warm up random numbers

   sumdets = determinants( NA );              // initialize determinant weights

   x = rng()/dp * 1000;
   y = rng()/dp * 1000;
   for( int i=0; i<500; i++ )                 // start with 500 warm-up steps
      iteration( NA, x, y );

   for( int i=n1; i<=n2; i++ ){               // perform selected iterations
      if( iteration( NA, x, y ) < 0 )
         break;
      cout << x << ' ' << y << '\n';          // output of r'
   }
}
```

```
double determinants( int NA )
{
  double sumdets = 0.0;
  for( int i=0; i<NA; i++ ){
    P[i] =  D[i][0][0] * D[i][1][1]         // calculate determinants
          - D[i][0][1] * D[i][1][0];
    if( P[i] <= 0.0 ) P[i]=-P[i];
    sumdets += P[i];
  }
  for( int i=0; i<NA; i++ ){
    P[i] /= sumdets;                        // normalize
  }
  return sumdets;
}

int iteration( int NA, double &x, double &y )
{
  int ierr = 0, iabb = 0;
  double sum = 0.0, z;

  if( rng()/dp > 0.1 ){                       // do this 90%
    z = rng()/dp;
    for( int i = 0; i < NA; i++ ){
      sum += P[i];
      if( sum >= z ){
        iabb = i;                             // select D_i according
        break;                                // to the weight of its
      }                                       // determinant
    }
  }
  else{                                       // do this 10%
    iabb = int( rng()/dp*NA );                // random D_i
  }
  z = D[iabb][0][0] * x + D[iabb][0][1] * y + C[iabb][0];
  y = D[iabb][1][0] * x + D[iabb][1][1] * y + C[iabb][1];
  x = z;

  return ierr;
}
```

5.7.5 Random Midpoint Displacement

Ein schöner Algorithmus, den man verwenden kann um fraktale Landschaften zu erzeugen, ist das sog. “random midpoint displacement” (RMD). Die Idee ist daß man ausgehend von einer Einheitszelle (Dreieck oder Quadrat) alle Verbindungslinien halbiert (in beiden Fällen) und diese Punkte dann zufällig vertikal versetzt. Im Fall des Quadrats muß man außerdem noch den Punkt im Zentrum versetzen, damit man vier Quadrate, diesmal halber Gr”ße, erhält. Diese Operation führt man nun für alle “Tochter”-Quadrate iterativ beliebig oft durch. Mit jeder Iteration steigt der Aufwand um den Faktor 4, und man hat nach n Iterationen bereits ein Gitter der Auflösung 2^n. Bei $n = 10$ sind sowohl dem Speicherplatz des Computers als auch der Auflösung am Bildschirm natürliche Grenzen gesetzt.

In Abb. 5.18 sind die ersten sechs Iterationen einer bestimmten Zufallszahlenserie abgebildet. Diese Bilder wurden mit dem `gnuplot`-script

```
set term post enhanced color
set hidden3d
set contour
set out ’rmd6.ps
splot ’f.6’ u 1:2:($3/150) t ’Iteration 6’ w l lt 1
```

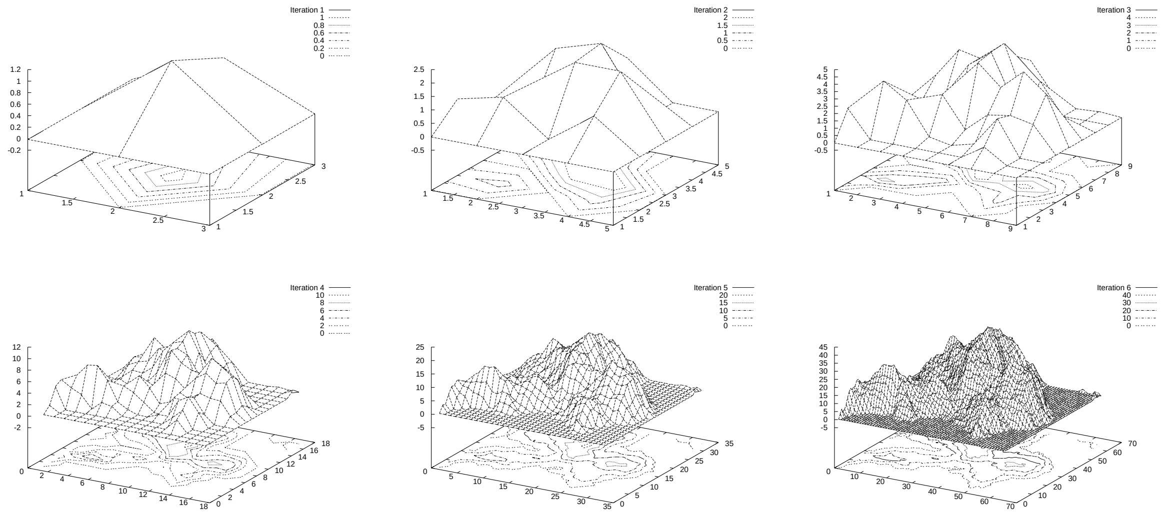

Abbildung 5.18: Random Midpoint displacement verschiedener Iterationstiefe.

Zum Abschluß dieses Kapitels (und zur Abschreckung) wird in diesem Abschnitt das Beispielprogramm in der Programmiersprache FORTRAN vorgestellt.

```
c
      program RMD
c
      implicit integer(h-k,m-n)
      implicit real(a-g,o-z)
      implicit logical(l)
c
      parameter(nmax=9,n2=2**nmax)
      dimension a(n2+1,n2+1)
c
      write(*,'($,a)') 'Iterationen?        '
      read(*,*) it
      write(*,'($,a)') 'Negativparameter?   '
      read(*,*) pn
      write(*,'($,a)') 'Streckung?          '
      read(*,*) dd
      write(*,'($,a)') 'SEED1?               '
      read(*,*) NSEED1
c
      call srand(NSEED1)
c
      w=-.005
c
      ni2=2**it
      ns=ni2/2
c
      a(1,1)=w
      a(1,ni2+1)=w
      a(ni2+1,1)=w
      a(ni2+1,ni2+1)=w
      amax=w
c
1     continue
      l1=.false.
      l2=.false.
      do 10 i=1,ni2+1,ns
         do 11 j=1,ni2+1,ns
            if(l1.or.l2)then
               a(i,j)=0.
               if(l1.and.l2)then
                  a(i,j)=(a(i+ns,j+ns)+a(i-ns,j+ns)+
     1                    a(i+ns,j-ns)+a(i-ns,j-ns))/4.+
     2                    (rand()-pn)*ns
                  if(a(i,j).gt.amax) amax=a(i,j)
               else if(l1.and.(.not.l2)) then
```

```
                if(i.ne.1) a(i,j)=a(i,j)+a(i-ns,j)
                if(i.ne.ni2+1) a(i,j)=a(i,j)+a(i+ns,j)
                a(i,j)=a(i,j)/2.+(rand()-pn)*ns
                if(a(i,j).gt.amax) amax=a(i,j)
             else
                if(j.ne.1) a(i,j)=a(i,j)+a(i,j-ns)
                if(j.ne.ni2+1) a(i,j)=a(i,j)+a(i,j+ns)
                a(i,j)=a(i,j)/2.+(rand()-pn)*ns
                if(a(i,j).gt.amax) amax=a(i,j)
             endif
          endif
          if(l2) then
             l2=.false.
          else
             l2=.true.
          endif
11       continue
        if(l1) then
           l1=.false.
        else
           l1=.true.
        endif
        l2=.false.
10    continue
      ns=ns/2
      if(ns.ge.1) goto 1
c
c
      open(99,file='f')
c     write(99,*) ni2
      do 20 j=1,ni2+1
         do 21 i=1,ni2+1
            if(a(i,j)/amax.le.w) a(i,j)=w
            write(99,*) i,j,int(a(i,j)*1000*dd)
21       continue
         write(99,'(a,$)') char(10)
20    continue
      close(99)
      end
```