```
c===========================================================
c     tsrand: Driver routine illustrating use of srand()
c     to "seed" the random number generator, rand(),
c     available on the SGIs.
c
c     Given seed >= 0 and, optionally, number of deviates to
c     generate, outputs
c
c       <i>   <random number>
c
c     i = 1 ... number of deviates on standard output.
c===========================================================
      program         tsrand

      implicit        none

c-----------------------------------------------------------
c     Uniform (on [0.0 .. 1.0]) random number generator.
c-----------------------------------------------------------
      real*8          rand

      integer         iargc,        i4arg
```

```
c-----------------------------------------------------------
c     Command-line arguments:
c
c     seed:     Integer-valued argument to srand() which
c               seeds the rand() random number generator.
c     n:        Number of deviates to generate
c-----------------------------------------------------------
      integer         seed,          n,
     &                               default_n
      parameter     (                default_n = 1 000 )

      integer         i

      if( iargc() .lt. 1 ) go to 900

      seed = i4arg(1,-1)
      if( seed .lt. 0 ) go to 900
      n    = i4arg(2,default_n)

      call srand(seed)
      do i = 1 , n
         write(*,*) i, rand()
      end do

      stop

 900  continue
         write(0,*) 'usage: tsrand <seed> [<n deviates>]'
      stop

      end
```

```
c=============================================================
c     nurand:  Uses SGI-specific uniform-random number
c     generator rand() to generate non-uniformly generated
c     random numbers on the interval [xmin..xmax]. User
c     must supply probability distribution function
c     having a header
c
c        subroutine pdf(x,pofx,maxpofx)
c
c     where 'x' is the input value, 'pofx' is the value
c     of the PDF evaluated at 'x' and 'maxpofx' is the
c     maximum value of the PDF (also a return argument).
c
c     Uses straight-forward algorithm based on area-under-
c     curve (PDF) idea--i.e. generate random point in
c     rectangle [xmin..xmax] x [0..maxpofx], accept point
c     and return x coordinate of point as random number
c     only if random point lies below PDF curve.
c=============================================================
      double precision function nurand(pdf,xmin,xmax)
         implicit      none

         external      pdf
         real*8        rand

         real*8        xmin,         xmax

         real*8        x,            y,
     &                 pofx,         maxpofx
```

```
c-------------------------------------------------------------
c        Loop until a good deviate has been generated.
c        Note that we exit the loop via the 'return'
c        statement---potentially this could be an infinite
c        loop, so for a "production" routine, it might
c        be wise to limit the number of iterations.
c-------------------------------------------------------------
         do while( .true. )
c-------------------------------------------------------------
c           Generate a uniform number in the interval xmin
c           to xmax.
c-------------------------------------------------------------
            x  =  xmin + rand() * (xmax - xmin)
c-------------------------------------------------------------
c           Evaluate PDF at x.
c-------------------------------------------------------------
            call pdf(x,pofx,maxpofx)
c-------------------------------------------------------------
c           Generate another uniform number in the interval
c           0 to maxpofx ...
c-------------------------------------------------------------
            y = rand() * maxpofx
c-------------------------------------------------------------
c           ... and accept the original random number, x,
c           if y < pofx.
c-------------------------------------------------------------
            if( y .lt. pofx ) then
               nurand = x
               return
            end if
         end do

      end
```

```
c=============================================================
c     Sample probability distribution functions.
c=============================================================

c-------------------------------------------------------------
c     Generates uniform deviates.
c-------------------------------------------------------------
      subroutine puniform(x,pofx,maxpofx)
         implicit      none
         real*8        x,     pofx,    maxpofx

         maxpofx = 1.0d0
         if( 0.0d0 .le. x  .and.  x .le. 1.0d0 ) then
            pofx = 1.0d0
         else
            pofx = 0.0d0
         end if

         return
      end
```

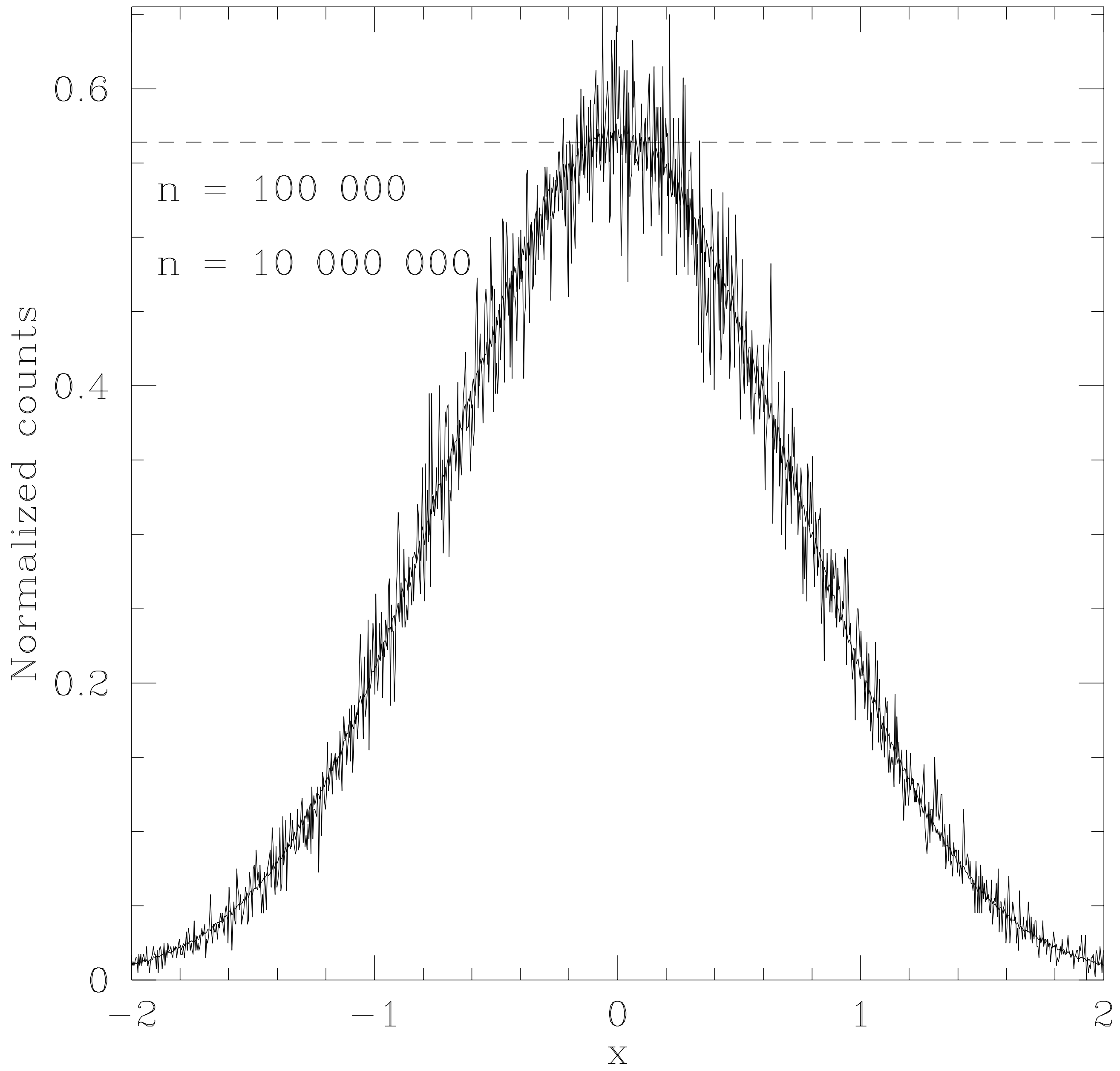
```
c-------------------------------------------------------------
c     Generates gaussian-distributed (unit sigma) deviates.
c-------------------------------------------------------------
      subroutine pgauss(x,pofx,maxpofx)
         implicit      none
         real*8        normalize
c-------------------------------------------------------------
c        Normalization can be any non-zero value,
c        might as well be unity.  "True" normalization
c        is 1 / sqrt(Pi) = 0.5641 8958 3547 7563d0.
c-------------------------------------------------------------
         parameter   ( normalize = 1.0d0 )
         real*8        x,     pofx,    maxpofx

         maxpofx = normalize
         pofx    = normalize * exp(-x**2)

         return
      end
```

```
c===========================================================
c     usage: tnurand <xmin> <xmax> <n> [<nbin> <option>]
c===========================================================
c     tnurand: Driver program for nurand(). This driver
c     generates non-uniformly distributed random-numbers
c     using a user-specified distribution function.  The
c     program is currently set up with two distribution
c     functions (see 'pdfs.f'):
c
c     option = 0            --> uniform
c     option = 1 (default)  --> unit-sigma Gaussian
c
c     The routine calls nurand() to generate n random
c     numbers, then writes binned counts (the interval
c     xmin ... xmax is divided into nbin equal width bins)
c
c       <i>   <count i>
c
c     i = 1 ... nbin, on standard output.
c
c     Note that nurand() uses rand(), so srand() can be
c     called to "seed" nurand().
c===========================================================
      program         tnurand

      implicit        none

c-----------------------------------------------------------
c     External declarations for the user-defined PDFs and
c     declaration of nurand.
c-----------------------------------------------------------
      external        puniform,     pgauss
      real*8          nurand
```

```
      integer         iargc,        i4arg
      real*8          r8arg
      real*8          r8_never
      parameter     ( r8_never = -1.0d-60 )
c-----------------------------------------------------------------------
c     Command-line arguments:
c
c     xmin:   Minimum, maximum values of deviates
c     xmax:
c     n:      Number of deviates to generate
c     nbin:   Number of binning intervals
c     option: Selects probability distribution function
c-----------------------------------------------------------------------
      real*8          xmin,         xmax
      integer         n,            nbin,         option

      integer         max_nbin
      parameter     ( max_nbin = 10 000 )
      real*8          x(max_nbin),  count(max_nbin)

      real*8          dx,           rnum
      integer         i,            j

c-----------------------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------------------
      if( iargc() .lt. 1 ) go to 900

      xmin   = r8arg(1,r8_never)
      if( xmin .eq. r8_never ) go to 900
      xmax   = r8arg(2,r8_never)
      if( xmax .eq. r8_never ) go to 900
      n      = i4arg(3,-1)
      if( n .le. 0 ) go to 900
```

```
      nbin   = min(i4arg(4,1000),max_nbin)
      option = i4arg(5,1)

c-------------------------------------------------------------
c     Set up bins and bin-coordinates (mid-points of bin
c     intervals).
c-------------------------------------------------------------
      dx = (xmax - xmin) / nbin
      do i = 1 , nbin
         count(i) = 0.0d0
         if( i .eq. 1 ) then
            x(1) = xmin + 0.5d0 * dx
         else
            x(i) = x(i-1) + dx
         end if
      end do

c-------------------------------------------------------------
c     Generate and bin random numbers.
c-------------------------------------------------------------
      do i = 1 , n
         if(      option .eq. 0 ) then
            rnum = nurand(puniform,xmin,xmax)
         else if( option .eq. 1 ) then
            rnum = nurand(pgauss,xmin,xmax)
         else
            write(0,*) 'tnurand: Unimplemented option ',
     &                 option
            stop
         end if
         j = min(int((rnum - xmin) / dx) + 1,nbin)
         count(j) = count(j) + 1.0d0
      end do
```

```
c-----------------------------------------------------------
c     Normalize bin counts.
c-----------------------------------------------------------
      do i = 1 , nbin
         count(i) = count(i) / (dx * n)
      end do

c-----------------------------------------------------------
c     Output bin counts.
c-----------------------------------------------------------
      call dvvto('-',x,count,nbin)

      stop

 900  continue
         write(0,*) 'tnurand: <xmin> <xmax> <n> '//
     &              '[<nbin> <option>]'
      stop

      end
```



0.6
0.4
0.2
0
n = 100 000
n = 10 000 000
Normalized counts
−2
−1
0
1
2
x

```
c===========================================================
c     dla: 2d diffusion-limited-aggregation with option
c     for "central bias" to accelerate cluster growth.
c===========================================================
c     usage: dla2d <size> <npart> [<r0> <bias>]
c
c     size:   Number of lattice sites on a side of the
c             arena.
c     npart:  Number of particles to evolve. Each particle
c             is evolved until one of its eight NN is
c             fixed.  It then becomes fixed and a new
c             particle is launched.
c     r0:     Relative launch diameter for new particles
c             (fraction of size).
c     bias:   0 <= bias <= 1.  Amount of bias towards
c             center of arena.  Current default is no
c             bias.
c
c     Program reads initial fixed particle positions
c     (x_i,y_i) from standard input (two numbers per line)
c     and writes final fixed particle positions to standard
c     output in same format.
c
c     The evolution (update) per se is done in-place in
c     the main-program, but separate routines for reading and
c     writing state, generating an initial particle position,
c     and generating a random move have been coded.
c
c     Refer to class notes for further details.
c===========================================================
```

```
      program         dla

      implicit        none

      integer         iargc,        i4arg,
     &                randstep
      real*8          r8arg,        rand
      real*8          r8_never
      parameter     ( r8_never = -1.0d-60 )

      integer         maxsize
      parameter     ( maxsize = 2000 )
      integer         arena(maxsize,maxsize)

      integer         size,         npart
      real*8          r0,           bias
      real*8          default_r0,   default_bias
      parameter     ( default_r0 = 0.75d0,
     &                              default_bias = 0.0d0 )

      real*8          threshold
      integer         nfixed
      integer         xfree,        yfree,        ipart,
     &                nstep,        i,            j,
     &                dxfree,       dyfree
      real*8          theta
      logical         free

      logical         ltrace
      parameter     ( ltrace = .true. )
```

```
c-----------------------------------------------------------
c     Argument parsing.
c-----------------------------------------------------------
      if( iargc() .lt. 2 ) go to 900
      size = i4arg(1,-1)
      if( size .lt. 50  .or.  size .gt. maxsize ) then
         write(0,*) 'dla: Specify size between 50 and ',
     &              maxsize
         stop
      end if
      npart = i4arg(2,-1)
      if( npart .lt. 1 ) go to 900
      r0    = r8arg(3,default_r0)
      bias  = r8arg(4,default_bias)
      threshold = 1.0d0 - bias
      if( ltrace ) then
         write(0,*) 'dla: size   ', size
         write(0,*) 'dla: npart  ', npart
         write(0,*) 'dla: r0     ', r0
         write(0,*) 'dla: bias   ', bias
      end if
c-----------------------------------------------------------
c     Initialize arena and read fixed particle positions
c     from standard input.
c-----------------------------------------------------------
      call getfixed(arena,maxsize,size,nfixed)
      if( nfixed .gt. 0 ) then
         write(0,*) 'dla: Read ', nfixed,
     &              ' particle positions.'
      else
         write(0,*) 'dla: No valid fixed particle ',
     &              'positions read.  Exiting.'
         stop
      end if
```

```
c-------------------------------------------------------------
c     For number of requested particles ...
c-------------------------------------------------------------
      do ipart = 1 , npart
         nstep = 0
c-------------------------------------------------------------
c        Generate random initial position.
c-------------------------------------------------------------
         call initposrand(size,r0,xfree,yfree)
         free = .true.
         do while( free )
c-------------------------------------------------------------
c           Take a random step (+1,0,-1) in both directions.
c-------------------------------------------------------------
            xfree = xfree + randstep()
            yfree = yfree + randstep()
c-------------------------------------------------------------
c           If bias is non zero, take a step towards
c           the origin with probability 'bias'.
c-------------------------------------------------------------
            if( bias .ne. 0.0d0 ) then
               dxfree = xfree - (size - 1) / 2
               dyfree = yfree - (size - 1) / 2
               theta  = atan2(1.0d0 * dxfree,1.0d0 * dyfree)
               if( rand() .le. abs(sin(theta)) ) then
                  if( rand() .gt. threshold ) then
                     if( dxfree .gt. 0 ) then
                        xfree = xfree - 1
                     else
                        xfree = xfree + 1
                     end if
                  end if
               end if
               if( rand() .le. abs(cos(theta)) ) then
```

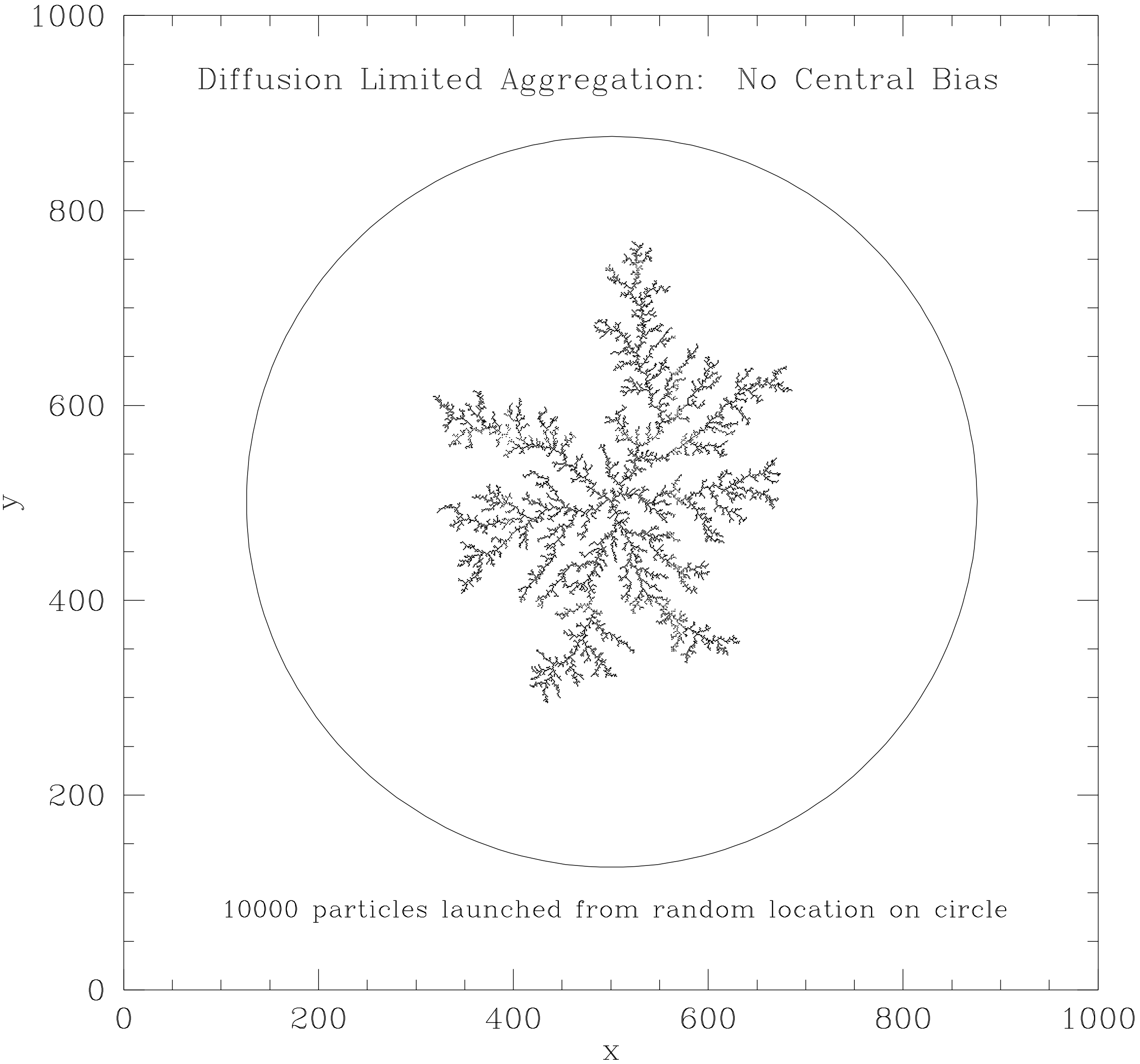
```
               if( rand() .gt. threshold ) then
                  if( dyfree .gt. 0 ) then
                     yfree = yfree - 1
                  else
                     yfree = yfree + 1
                  end if
               end if
            end if
         end if
c------------------------------------------------------------
c        Check if particle is outside arena.
c------------------------------------------------------------
         if( xfree .lt. 1  .or.  xfree .gt. size  .or.
     &       yfree .lt. 1  .or.  yfree .gt. size ) then
c------------------------------------------------------------
c           If it is, reinitialize
c------------------------------------------------------------
            call initposrand(size,r0,xfree,yfree)
         end if
c------------------------------------------------------------
c        Check if particle should be fixed.
c------------------------------------------------------------
         do i = max(xfree-1,1) , min(xfree+1,size)
            do j = max(yfree-1,1) , min(yfree+1,size)
c------------------------------------------------------------
c              If it is, update corresponding arena
c              site and set flag.
c------------------------------------------------------------
               if( arena(i,j) .ne. 0 ) then
                  arena(xfree,yfree) = 1
                  free = .false.
               end if
            end do
         end do
```

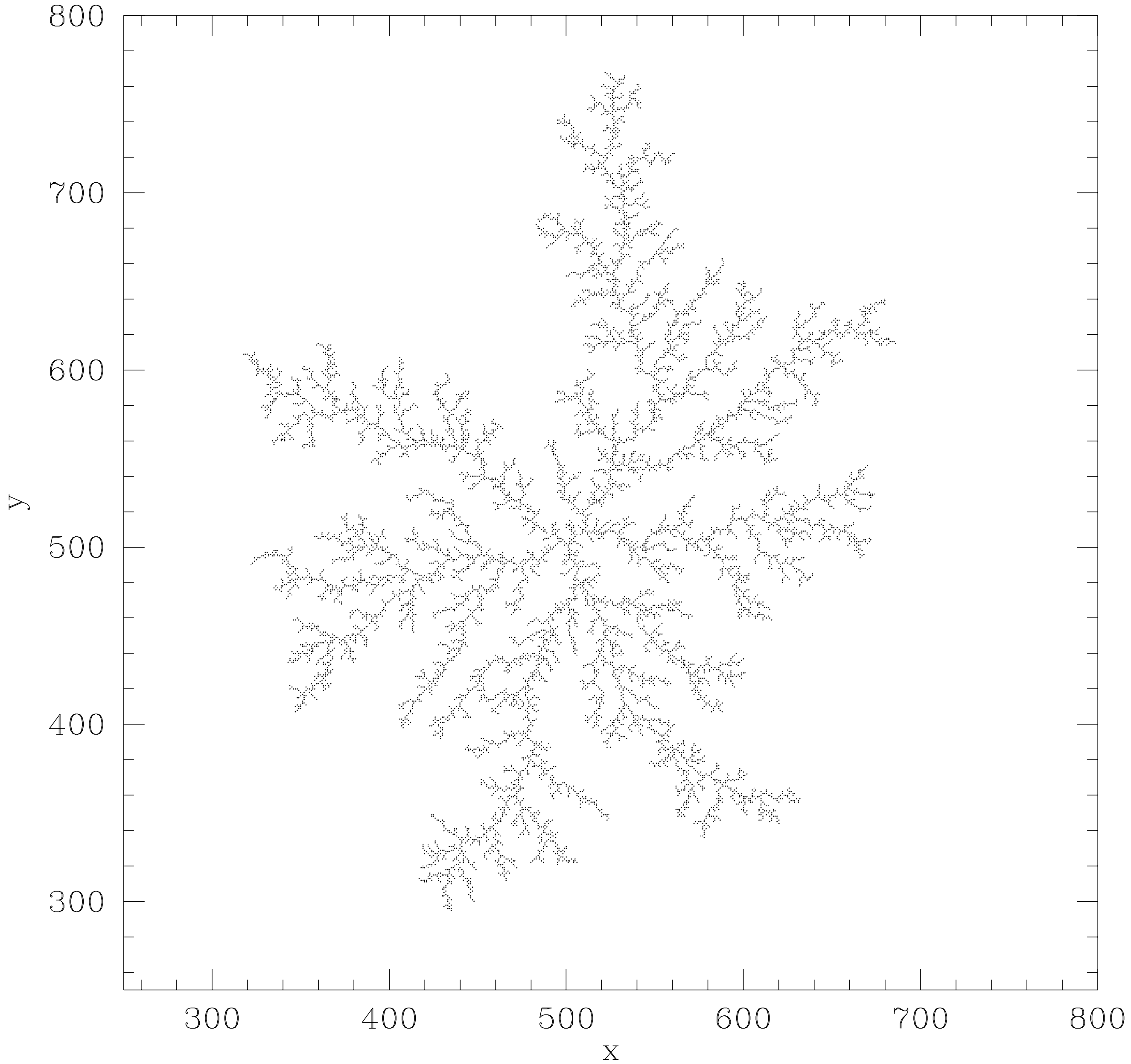
```
            nstep = nstep + 1
         end do
         write(0,*) 'dla: Particle ', ipart, ' fixed after ',
     &              nstep, ' steps'
      end do

c-------------------------------------------------------------
c     Write fixed particle positions to standard output.
c-------------------------------------------------------------
      call putfixed(arena,maxsize,size,nfixed)
      write(0,*) 'dla: Wrote ', nfixed, ' particle positions.'

      stop

900   continue
         write(0,*) 'usage: dla <size> <npart> [<r0> <bias>]'
         write(0,9000) default_r0, default_bias
9000     format(/
     &      '        Current default <r0>: ',f13.2/
     &      '        Current default <bias>: ',1p,e11.4,0p//
     &      '        Program reads initial fixed-particle coordinates '/
     &      '        (integers x,y; 1 <= x,y <= size)  from standard'/
     &      '        input, writes final fixed positions to standard'/
     &      '        output.')
      stop

      end
```

```
c=============================================================
c     Returns -1, 0 or 1 chosen randomly
c=============================================================
      integer function randstep()
         implicit     none
         real*8       rand

         randstep = min(2,int(3.0d0 * rand())) - 1

         return
      end

c=============================================================
c     Initialize arena then read fixed particle positions
c     from standard input.  Ignore particles lying outside
c     of current arena.  Returns number of fixed particles
c     inside arena.
c=============================================================
      subroutine getfixed(arena,maxsize,size,nfixed)
         implicit     none

         integer      maxsize,      size,       nfixed
         integer      arena(maxsize,maxsize)

         integer      x,            y,          rc,
     &                i,            j

         do j = 1 , size
            do i = 1 , size
               arena(i,j) = 0
            end do
         end do
```

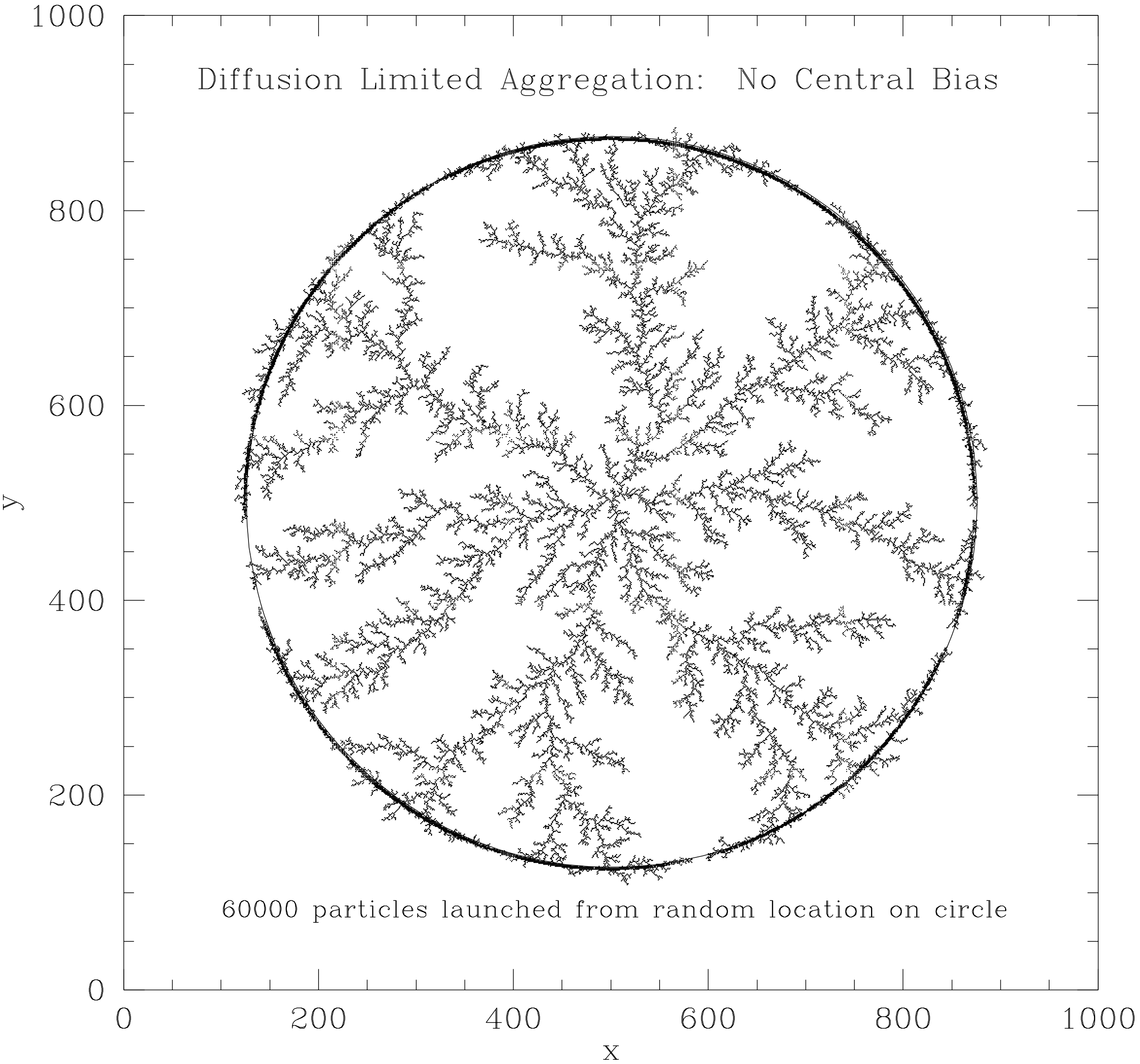
```
      nfixed = 0
 100  continue
         read(*,*,end=200,iostat=rc)  x,  y
         if( rc .eq. 0  ) then
            if( 1 .le. x  .and.  x .le. size   .and.
     &          1 .le. y  .and.  y .le. size ) then
               arena(x,y) = 1
               nfixed = nfixed + 1
            end if
         end if
      go to 100

 200  continue
      return

   end

c=============================================================
c     Writes fixed particle positions to standard output.
c     Returns number of fixed particles.
c=============================================================
   subroutine putfixed(arena,maxsize,size,nfixed)
      implicit     none

      integer      maxsize,      size,       nfixed
      integer      arena(maxsize,maxsize)

      integer      i,            j

      nfixed = 0
      do j = 1 , size
         do i = 1 , size
            if( arena(i,j) .ne. 0 ) then
               nfixed = nfixed + 1
```

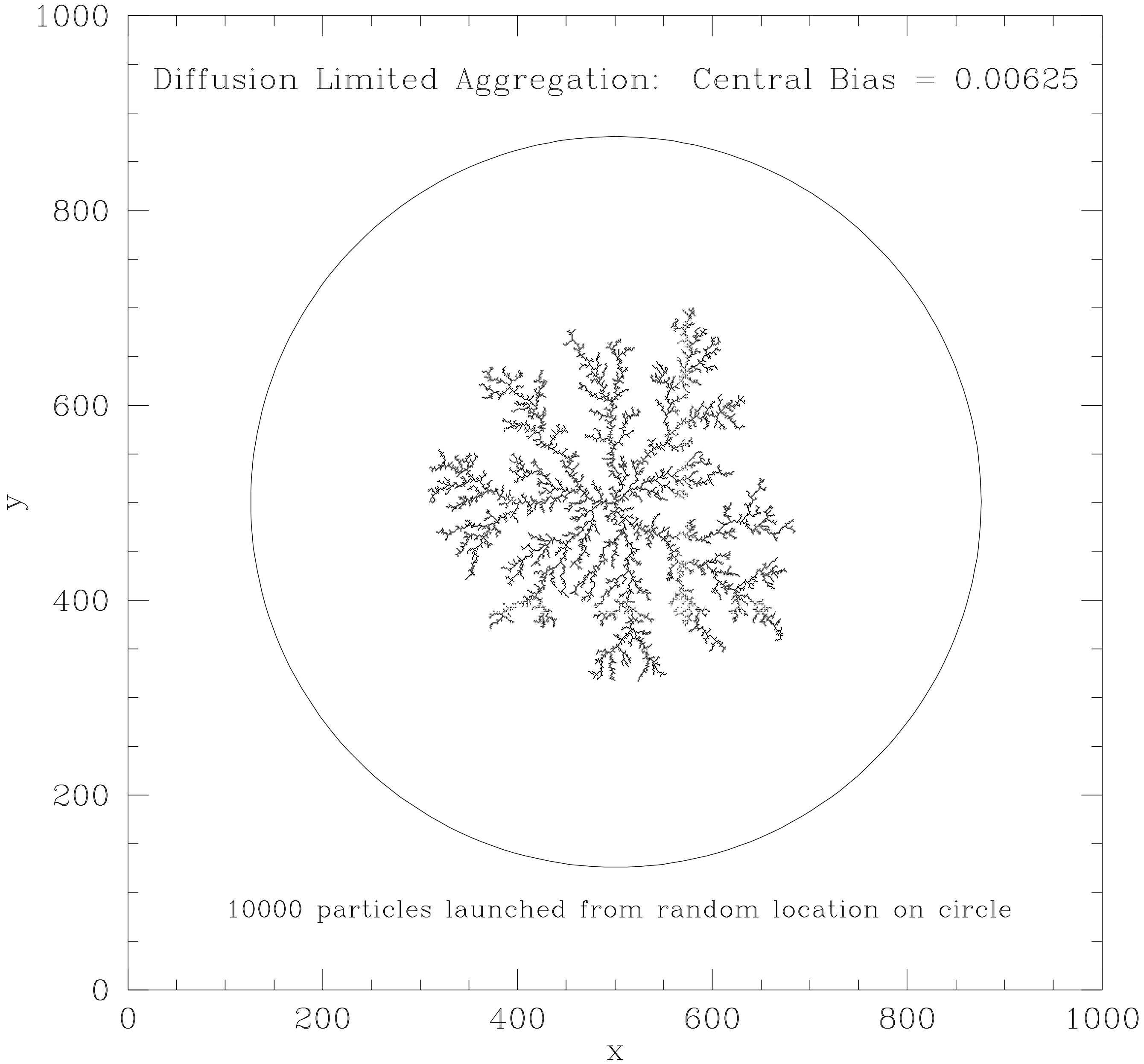
```
               write(*,*) i, j
            end if
         end do
      end do

      return
   end

c=================================================================
c     Generates initial particle position 0.5 * r0 * size
c     from arena center, randomly positioned in angle.
c=================================================================
      subroutine initposrand(size,r0,xfree,yfree)
         implicit     none

         real*8       rand
         integer      size,    xfree,     yfree
         real*8       r0
         real*8       r,       theta
c-----------------------------------------------------------------
c        Generate a random angle from 0 to 2 Pi.
c-----------------------------------------------------------------
         theta = rand() * 8.0d0 * atan(1.0d0)
         r     = 0.5d0 * r0 * (size - 1)
         xfree = 0.5d0 * (size - 1)  + r * cos(theta)
         yfree = 0.5d0 * (size - 1)  + r * sin(theta)
         return
      end
```

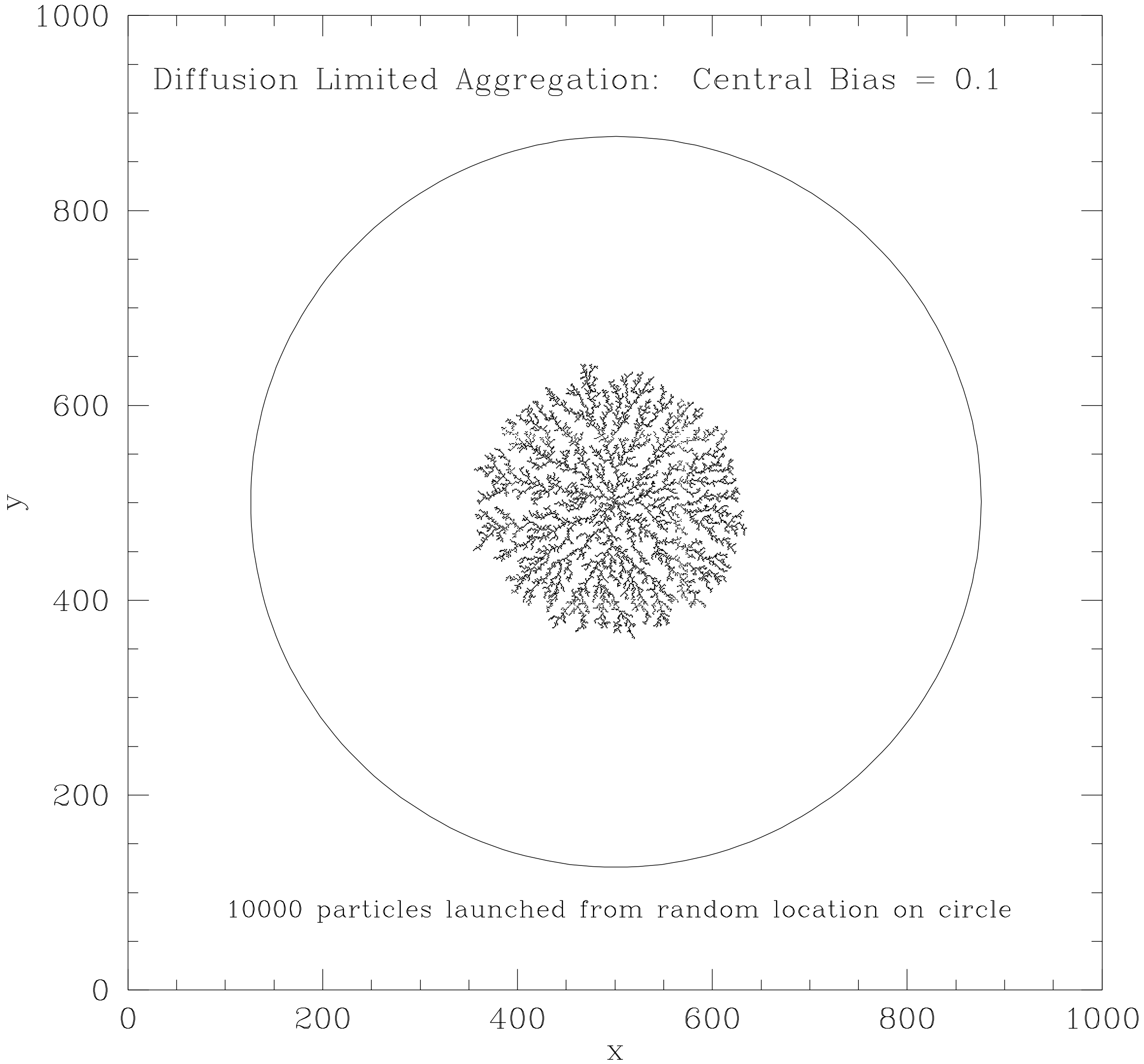

Diffusion Limited Aggregation: No Central Bias
10000 particles launched from random location on circle
1000
800
600
400
200
0
y
0
200
400
600
800
1000
x

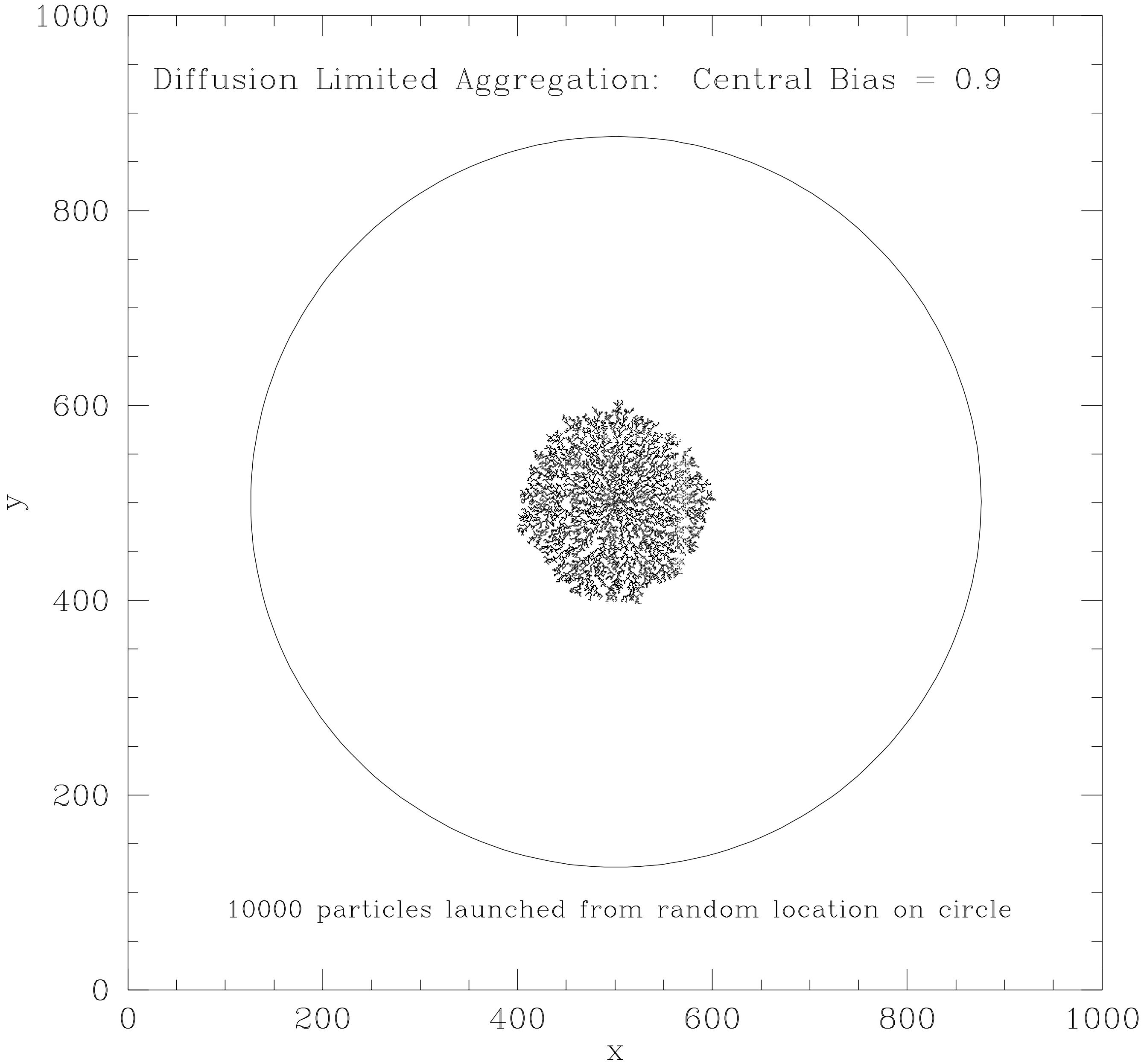


800
700
600
500
400
300
y
300
400
500
600
700
800
x


Diffusion Limited Aggregation: No Central Bias
60000 particles launched from random location on circle
1000
800
600
400
200
0
y
0
200
400
600
800
1000
x


Diffusion Limited Aggregation: Central Bias = 0.00625
10000 particles launched from random location on circle
y
x
0
200
400
600
800
1000


Diffusion Limited Aggregation: Central Bias = 0.1
10000 particles launched from random location on circle
x
y
0
200
400
600
800
1000


Diffusion Limited Aggregation: Central Bias = 0.9
10000 particles launched from random location on circle
x
y
0
200
400
600
800
1000