B G I − F O R T R A N

by Angelo Graziosi

I N T R O D U C T I O N

This document contains a few examples of fortran programs "using" the Borland Graphics Interface (BGI) library.

BGI is a very simple graphics library with which one can write interesting programs with minimum efforts. It allows also to learn the basic principles of Graphics Programming.

We started using this library toward the end of the 1980s. Indeed it was included with the early versions of the Borland Turbo Pascal compiler. Then we continued to develop BGI programs until 2002−2003, mainly using the Borland C++ 2.0 compiler.

Since BGI was, in some way, linked to the DOS world, its usage was progressively abandoned (as the DOS OS) in favor of other systems. But a few years ago we discovered a C++ interface (WinBGIm, using Windows API) to BGI, so we developed a full fortran interface module (f03bgi, not listed here) exploiting the new Fortran 2003 standard.

Recently, we found on the WEB new C ports of BGI: libXbgi and SDL_bgi. So we started to write a new fortran interface (not yet completed) to these libraries, and this document shows what we can do with a minimal coding.

Notice that, while these C/C++ interfaces aim to recompile old BGI programs without or with minimal changes, our goal is to have tools to write new fortran graphics programs: indeed old BGI fortran programs do not exist at all!

Now, a short description of the following examples. Usually they are self−explanatory.

MANDELBROT

"mandelbrot.f90" is a fortran rewriting of the C version one can find in the source distribution of libXbgi and SDL_bgi libraries. Really, it is NOT a simple rewriting, but we have restructured the code. In its initial comment there is the description how it is built. Notice that it uses what we can call "array of function pointers".

An historical note: the first time we wrote a program to generate a Mandelbrot set (in Turbo Pascal, or C? we don’t remember...), using BGI, it took about an hour on our old 286 without math co−processor. After we bought the 80287 co−processor, it took about 10 minutes! That was on VGA 640x480, 16 colors and max. iter = 25. The current mandelbrot.f90 runs with max. iter > 2000 and 256 colors, in a 1600x900 window, taking only few seconds! (On AMD Athlon X2 64)

BIOMORPH

The first time we learned about biomorphs, it was reading an article ((Ri)Creazioni al calcolatore, Le Scienze, italian edition of Scientific American) in the August 1989. We wrote a programs which we present anew here, only that now we have used the same approach we adopted for mandelbrot.f90. Biomprh doesn’t use array of function pointers but only function pointers.

SOLAR_SYSTEM
============

This programs was written, firstly, using GTK−FORTRAN, now we have "translated" it using BGI. It uses SOFAlib, found on the WEB.

DYNAMICS2D
==========

This programs compute and display the trajectory of a point on which is acting a two dimensional field of forces. The particularity is that the force components are read from keyboard. It uses a "functions parser". We have found on the WEB a functions parser written in C++ (http://warp.povusers.org/FunctionParser), and have interfaced it in fortran using the Fortran >= 2003 standard. For a short documentation see the other link on this WEB site.

DOUBLE_PENDULUM−DB
==================

This programs compute and display the motion of a double pendulum. It uses the double buffering technique.

THOMAS_FERMI, LOGISTICS, etc..

BGI, BGIAPP
===========

These are the modules which underlie the above examples.

- bgi.f90 contains the modules interfacing libXbgi.
- bgiapp.f90 implements a few routines that allow to develop fortran BGI programs in World Coordinate System.

To develop the above examples, we have used xbgi−364 (http://libxbgi.sourceforge.net), but this library still contains bugs. We have tried to work around a few of them. We have also added the extension to give an our title to the BGI window.

____
This document has been created using EMACS (and some "friends" tools like ps2pdf, pdftk etc..).

```
!
! Fortran Interface to the Xbgi−364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!   cd ~/programming/bgi−fortran/demo
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{bgi.f90,bgiapp.f90} mandelbrot.f90 \
!     −L ~/programming/lib −lXbgi −lX11 −lm −o mandelbrot.out
!
!   ./mandelbrot.out
!

module mandelbrot_lib
  use kind_consts, only: DP
  use bgi, only: CENTER_TEXT, DEFAULT_FONT, GOTHIC_FONT, HORIZ_DIR, &
       KEY_ESC, LEFT_TEXT, TOP_TEXT, WHITE, WM_LBUTTONDOWN, WM_MBUTTONDOWN, &
       WM_MOUSEMOVE, WM_RBUTTONDOWN, WM_WHEELDOWN, WM_WHEELUP, &
       cleardevice, rgb_color, CString, fast_putpixel, getevent, &
       kbhit, mousex, mousey, outtextxy, refresh, setbkcolor, &
       setcolor, setrgbcolor, setrgbpalette, settextjustify, &
       settextstyle, textheight, usleep
  implicit none
  private

  abstract interface
     subroutine colors_palette()
     end subroutine colors_palette
  end interface

  type palette_ptr
     procedure(colors_palette), pointer, nopass :: p => null()
  end type palette_ptr

  integer :: max_iter = 100, max_x, max_y
  type(palette_ptr) :: palette(3)

  public :: input_data, run_app

contains

  subroutine purple_palette()
    integer :: c

    do c = 0, max_iter−1
       call setrgbpalette(c,50+2*c,c,max_iter−c)
    end do

    ! The Mandelbrot set is black
    call setrgbpalette(max_iter,0,0,0)
  end subroutine purple_palette

  subroutine blue_palette()
    integer :: c
```

```
  do c = 0, max_iter−1
     call setrgbpalette(c,0,c,50+2*c)
  end do

  ! The Mandelbrot set is black
  call setrgbpalette(max_iter,0,0,0)
end subroutine blue_palette

subroutine amber_palette()
  integer :: c

  do c = 0, max_iter−1
     call setrgbpalette(c,max_iter−c,50+2*c,c)
  end do

  ! The Mandelbrot set is purple
  call setrgbpalette(max_iter,int(Z'30'),0,int(Z'30'))
end subroutine amber_palette

subroutine input_data()
  use get_data, only: get

  call get('MAX_ITER = ',max_iter)
  write(*,*)

  palette(1)%p => purple_palette
  palette(2)%p => blue_palette
  palette(3)%p => amber_palette
end subroutine input_data

subroutine explain()
  integer, parameter :: MILLI_SECONDS = 1000
  integer :: i, inc, c, k

  ! Don't use a palette
  call setbkcolor(rgb_color(0,0,32))
  call cleardevice()
  call setcolor(rgb_color(255,255,0))

  call settextstyle(GOTHIC_FONT,HORIZ_DIR,1)
  call settextjustify(CENTER_TEXT,CENTER_TEXT)
  c = textheight ('H')

  call outtextxy(max_x/2,max_y/2−3*c, &
       CString("Press '1', '2', or '3' to change the palette;"))
  call outtextxy(max_x/2,max_y/2−2*c, &
       CString('left click to zoom in on a point;'))
  call outtextxy(max_x/2, max_y/2−c, &
       CString('right click to zoom out;'))
  call outtextxy(max_x/2, max_y/2, &
       CString('middle click to restore the initial boundary;'))
  call outtextxy(max_x/2, max_y/2+c, &
       CString("'i' and 'd' to increase/decrease max iterations;"))
  call outtextxy(max_x/2,max_y/2+2*c, &
       CString('ESC to quit the program.'))

  i = 0
  inc = 1
  do while (kbhit() == 0)

     call setcolor(rgb_color(i,0,0))
     call outtextxy (max_x/2,max_y/2+4*c, CString('PRESS A KEY TO BEGIN'))
     i = i+inc

     select case(i)
     case (255)
        inc = −1
     case (0)
        inc = 1
     end select
     k = usleep(1*MILLI_SECONDS)
  end do

  call cleardevice()

  call settextstyle(DEFAULT_FONT,HORIZ_DIR,1)
  call settextjustify(LEFT_TEXT,TOP_TEXT)
```

```
end subroutine explain

subroutine mandelbrot_set(x1,y1,x2,y2)
  real(DP), intent(in) :: x1, y1, x2, y2
  !
  ! We assume that the point (x,y) is repesented by the center of the
  ! pixel box. For example, in the X direction we have that
  !
  !   the center of pixel−box 0 is the point (x1,y)
  !   the center of pixel−box 1 is the point (x1+dx,y)
  !   the center of pixel−box 2 is the point (x1+2*dx,y)
  !   ...
  !   the center of pixel−box max_x is the point (x1+max_x*dx == x2,y)
  !
  ! This means that dx = (x2−x1)/max_x. The same happens in Y direction, and
  ! dy = (y2−y1)/max_y
  !
  integer :: i, j, counter
  real(DP) :: dx, dy, x, y, a, b, tx, d
  logical :: confined

  dx = (x2−x1)/max_x
  dy = (y2−y1)/max_y

  x = x1
  do i = 0, max_x
     y = y1
     do j = 0, max_y

        counter = 0
        a = 0.0_DP
        b = 0.0_DP

        ! Iteration: z(n+1) = z(n)**2 + c; z = a+i*b; c = x+i*y
        confined = .true.
        do while (confined)
           tx = a*a−b*b+x
           b = 2.0_DP*a*b+y
           a = tx
           d = a*a+b*b
           counter = counter+1
           confined = ((d <= 4.0_DP) .and. (counter < max_iter))
        end do

        call setrgbcolor(counter)
        call fast_putpixel(i,j)
        y = y+dy
     end do
     x = x+dx
  end do
end subroutine mandelbrot_set

subroutine run_app
  use bgiapp, only: bgiapp_xmin, bgiapp_xmax, bgiapp_ymin, bgiapp_ymax, &
       bgiapp_width, bgiapp_height

  integer :: current_palette, key = −1000
  real(DP) :: xm, ym, xstep, ystep, x1, y1, x2, y2, &
       xm0, ym0, xstep0, ystep0
  logical :: init, flag, redraw
  character(len = 20) :: buf

  ! Getting the viewing region...
  x1 = bgiapp_xmin()
  x2 = bgiapp_xmax()
  y1 = bgiapp_ymin()
  y2 = bgiapp_ymax()
  max_x = bgiapp_width()
  max_y = bgiapp_height()

  ! Getting DEFAULT for initial boundary
  xm0 = 0.5_DP*(x1+x2)
  ym0 = 0.5_DP*(y1+y2)
  xstep0 = (x2−x1)/2
  ystep0 = (xstep0*max_y)/max_x

  max_x = max_x−1
  max_y = max_y−1
```

```
! Initial boundary
xm = xm0
ym = ym0
xstep = xstep0
ystep = ystep0

init = .true.
flag = .true.
redraw = .true.

call explain()

current_palette = 1
call palette(current_palette)%p()

do while (key /= KEY_ESC)

   x1 = xm−xstep
   y1 = ym−ystep
   x2 = xm+xstep
   y2 = ym+ystep

   if (redraw) then
      call mandelbrot_set(x1,y1,x2,y2)
      call refresh()

      if (flag) then
         call setcolor(WHITE)
         write(buf,*) max_iter
         call outtextxy(0,max_y−20,CString(trim(adjustl(buf))))
         flag = .false.
      end if
      redraw = .false.
   end if

   ! Wait for a key or mouse click
   key = getevent ()

   select case (key)
   case (WM_LBUTTONDOWN, WM_WHEELUP)
      xm = x1+(x2−x1)*mousex()/max_x
      ym = y1+(y2−y1)*mousey()/max_y
      xstep = xstep/2
      ystep = ystep/2
      init = .false.
      redraw = .true.
   case (WM_RBUTTONDOWN, WM_WHEELDOWN)
      xstep = xstep*2
      ystep = ystep*2
      init = .false.
      redraw = .true.
   case (WM_MBUTTONDOWN)
      if (.not. init) then
         xm = xm0
         ym = ym0
         xstep = xstep0
         ystep = ystep0
         redraw = .true.
      end if
   case (ichar('1'))
      if (current_palette /= 1) then
         current_palette = 1
         call palette(current_palette)%p()
         redraw = .true.
      end if
   case (ichar('2'))
      if (current_palette /= 2) then
         current_palette = 2
         call palette(current_palette)%p()
         redraw = .true.
      end if
   case (ichar('3'))
      if (current_palette /= 3) then
         current_palette = 3
         call palette(current_palette)%p()
         redraw = .true.
      end if
```

```
        case (ichar('i'))
           max_iter = max_iter+50
           flag = .true.
           redraw = .true.

           ! Since the current palette depend on MAX_ITER,
           ! you HAVE TO reset it...
           call palette(current_palette)%p()
        case (ichar('d'))
           max_iter = max_iter-50
           flag = .true.
           redraw = .true.

           ! Since the current palette depend on MAX_ITER,
           ! you HAVE TO reset it...
           call palette(current_palette)%p()

        case default
           redraw = .false.

        end select
     end do
  end subroutine run_app
end module mandelbrot_lib

program mandelbrot
  use kind_consts, only: DP
  use general_routines, only: system_time
  use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
  use mandelbrot_lib
  implicit none

  real(DP) :: t0, t1

  call input_data()

  ! We assume the center at (-0.75,0) and size (4.4,2.475)
  call bgiapp_setup(-2.95_DP,1.45_DP,-1.2375_DP,1.2375_DP,1600,900)
  call bgiapp_init('A tribute to Benoit Mandelbrot (1924-2010)')

  write(*,'(A)',advance='NO') 'Please wait, we are working...'

  t0 = system_time()
  call run_app()
  t1 = system_time()

  write(*,*)
  write(*,'(A,F8.3,A)') 'Mandelbrot completed in ',t1-t0,' seconds!'

  call bgiapp_close()
end program mandelbrot
```

```
!
! Fortran Interface to the Xbgi−364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!   cd ~/programming/bgi−fortran/demo
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ~/programming/fparser−fortran/fparser_cd.f90 \
!     ../{bgi.f90,bgiapp.f90} biomorph.f90 \
!     −L ~/programming/lib −lFParser −lstdc++ −lXbgi −lX11 −lm \
!     −o biomorph.out
!
!   ./biomorph.out
!
!
! EXAMPLES
!
!   fcn(z,c) = z^3+c,            c = (0.5,0), (0.44,0), (0.6,−0.2), (−0.6,−0.1)
!   fcn(z,c) = z^z+z^5+c,        c = (0.5,0)
!   fcn(z,c) = sin(z)+z^z+c,     c = (0.5,0)
!   fcn(z,c) = sin(z)+exp(z)+c,  c = (0.5,0)
!   fcn(z,c) = z^5+c,            c = (0.65,0)
!   fcn(z,c) = z^7+z^5+c,        c = (0.8,0)
!
! Notice that we could define the biomorph as fcn(z,c) = f(z) + c, with
!
!   f(z) = z^3, z^z+z^5, sin(z)+z^z... etc.
!

module biomorph_lib
  use kind_consts, only: DP
  use fparser_cd, only: FunctionParser_cd_type, NewParser, Parse, &
       ErrorMsg, GetParseErrorType, DeleteParser, Eval
  use bgi, only: CENTER_TEXT, DEFAULT_FONT, GOTHIC_FONT, HORIZ_DIR, &
       KEY_ESC, LEFT_TEXT, TOP_TEXT, WHITE, WM_LBUTTONDOWN, WM_MBUTTONDOWN, &
       WM_MOUSEMOVE, WM_RBUTTONDOWN, WM_WHEELDOWN, WM_WHEELUP, &
       cleardevice, rgb_color, CString, fast_putpixel, &
       getevent, kbhit, mousex, mousey, &
       outtextxy, refresh, setbkcolor, setcolor, &
       setrgbcolor, setrgbpalette, settextjustify, settextstyle, &
       textheight, usleep
  implicit none
  private

  type(FunctionParser_cd_type) :: fp_biomorph
  integer :: max_iter = 100, num_colors = 0, max_x, max_y
  complex(DP) :: c = (0.5_DP,0.0_DP)

  public :: input_data, run_app

contains

  subroutine input_data()
    use get_data, only: get, MAXLEN
```

```
  character(len = MAXLEN) :: biomorph_buf = 'z^3+c'
  integer :: res

  call get('FCN(z,c) = ',biomorph_buf)
  write(*,*)

  call get('C = ',c)
  write(*,*)

  ! Create the fparser for FCN(z,c)
  call NewParser(fp_biomorph)
  res = Parse(fp_biomorph,biomorph_buf,'z,c')
  if (res >= 0) then
     write(*,*) 'Failure creating fp_biomorph parser...'
     write(*,*)
     write(*,'(A)') 'FCN(z,c) = '//trim(biomorph_buf)
     write(*,'(A)') repeat(' ',res+10)//'^'

     ! Remember : ErrorMsg() is an array of characters...
     write(*,*) ErrorMsg(fp_biomorph)
     write(*,*) 'Error type: ',GetParseErrorType(fp_biomorph)
     write(*,*)
     stop
  end if
end subroutine input_data

subroutine explain()
  integer, parameter :: MILLI_SECONDS = 1000
  integer :: i, inc, c, k

  ! Don't use a palette
  call setbkcolor(rgb_color(0,0,32))
  call cleardevice()
  call setcolor(rgb_color(255,255,0))

  call settextstyle(GOTHIC_FONT,HORIZ_DIR,1)
  call settextjustify(CENTER_TEXT,CENTER_TEXT)
  c = textheight ('H')

  call outtextxy(max_x/2,max_y/2−3*c, &
       CString("Press '1' or '2' to change the palette;"))
  call outtextxy(max_x/2,max_y/2−2*c, &
       CString('left click to zoom in on a point;'))
  call outtextxy(max_x/2, max_y/2−c, &
       CString('right click to zoom out;'))
  call outtextxy(max_x/2, max_y/2, &
       CString('middle click to restore the initial boundary;'))
  call outtextxy(max_x/2, max_y/2+c, &
       CString("'i' and 'd' to increase/decrease max iterations;"))
  call outtextxy(max_x/2,max_y/2+2*c, &
       CString('ESC to quit the program.'))

  i = 0
  inc = 1
  do while (kbhit() == 0)

     call setcolor(rgb_color(i,0,0))
     call outtextxy (max_x/2,max_y/2+4*c, CString('PRESS A KEY TO BEGIN'))
     i = i+inc

     select case(i)
     case (255)
        inc = −1
     case (0)
        inc = 1
     end select
     k = usleep(1*MILLI_SECONDS)
  end do

  call cleardevice()

  call settextstyle(DEFAULT_FONT,HORIZ_DIR,1)
  call settextjustify(LEFT_TEXT,TOP_TEXT)
end subroutine explain

subroutine bgi_palette()
  num_colors = 16
```

```
  call setrgbpalette(0,0,0,128)      ! Blue
  call setrgbpalette(1,0,128,0)      ! Green
  call setrgbpalette(2,0,128,128)    ! Cyan

  call setrgbpalette(3,128,0,0)      ! Red
  call setrgbpalette(4,128,0,128)    ! Magenta
  call setrgbpalette(5,128,128,0)    ! Brown
  call setrgbpalette(6,192,192,192)  ! Light Gray

  call setrgbpalette(7,128,128,128)  ! Dark Gray
  call setrgbpalette(8,0,0,255)      ! Light Blue
  call setrgbpalette(9,0,255,0)      ! Light Green
  call setrgbpalette(10,0,255,255)   ! Light Cyan

  call setrgbpalette(11,255,0,0)     ! Light Red
  call setrgbpalette(12,255,0,255)   ! Light Magenta
  call setrgbpalette(13,255,255,0)   ! Yellow
  call setrgbpalette(14,255,255,255) ! White

  ! Black is on the top
  call setrgbpalette(15,0,0,0)        ! Black
end subroutine bgi_palette

subroutine bw_palette()
  num_colors = 2

  call setrgbpalette(0,255,255,255)  ! White
  call setrgbpalette(1,0,0,0)        ! Black
end subroutine bw_palette

function fcn(z,c) result(ret)
  complex(DP) :: ret
  complex(DP), intent(in) :: z,c
  ret = Eval(fp_biomorph,[z,c])
end function fcn

subroutine biomorph_set(x1,y1,x2,y2)
  real(DP), intent(in) :: x1, y1, x2, y2
  !
  ! We assume that the point (x,y) is repesented by the center of the
  ! pixel box. For example, in the X direction we have that
  !
  !   the center of pixel−box 0 is the point (x1,y)
  !   the center of pixel−box 1 is the point (x1+dx,y)
  !   the center of pixel−box 2 is the point (x1+2*dx,y)
  !   ...
  !   the center of pixel−box max_x is the point (x1+max_x*dx == x2,y)
  !
  ! This means that dx = (x2−x1)/max_x. The same happens in Y direction, and
  ! dy = (y2−y1)/max_y
  !
  real(DP), parameter :: RADIUS = 10.0_DP, QRADIUS = RADIUS*RADIUS
  complex(DP), parameter :: JJ = (0,1)
  integer :: i, j, counter
  real(DP) :: dx, dy, x, y, a = 0, b = 0
  complex(DP) :: z

  dx = (x2−x1)/max_x
  dy = (y2−y1)/max_y

  x = x1
  do i = 0, max_x
     y = y1
     do j = 0, max_y
        z = x+JJ*y
        !
        ! The difference between this loop:
        !
        !   counter = 0
        !   do while (counter < max_iter)
        !
        !     if (condition) exit
        !   end do
        !
        ! and this loop:
        !
        !   do counter = 1, max_iter
        !
```

```
        !     if (condition) exit
        !   end do
        !
        ! is that in the first case, the loop is executed at most MAX_ITER
        ! times with COUNTER <= MAX_ITER after exiting the loop
        ! In the second case, the loop is execute at most MAX_ITER times too,
        ! but counter is max_iter+1 if CONDITION is not satisfied...
        !
        counter = 0
        do while (counter < max_iter)
           z = fcn(z,c)
           a = abs(real(z))
           b = abs(aimag(z))
           counter = counter+1
           !
           ! Notice that ('==>' means 'implies')
           !
           !   (a > RADIUS)  ==>  ((a*a+b*b) > QRADIUS)
           !   (b > RADIUS)  ==>  ((a*a+b*b) > QRADIUS)
           !
           ! so
           !
           !   (a > RADIUS) .or. (b > RADIUS) .or. ((a*a+b*b) > QRADIUS)
           !
           ! is equivalent to
           !
           !   ((a*a+b*b) > QRADIUS)
           !
           ! but if (a > RADIUS), we don't need to evaluate (b > RADIUS) and
           ! ((a*a+b*b) > QRADIUS); if (b > RADIUS), we don't need to evaluate
           ! ((a*a+b*b) > QRADIUS). In short, the long condition
           !
           !   if ((a > RADIUS) .or. (b > RADIUS) .or. ((a*a+b*b) > QRADIUS))
           !
           ! is a little faster than the short
           !
           !   if ((a*a+b*b) > QRADIUS))
           !
           if ((a > RADIUS) .or. (b > RADIUS) .or. ((a*a+b*b) > QRADIUS)) &
                exit
        end do

        if ((a < RADIUS) .or. (b < RADIUS)) then
           ! On the top there is BLACK... see setup_rgb_palette()
           call setrgbcolor(num_colors-1)
        else
           call setrgbcolor(mod(counter-1,num_colors-1))
        end if

        call fast_putpixel(i,j)

        y = y+dy
     end do
     x = x+dx
  end do
end subroutine biomorph_set

subroutine run_app
  use bgiapp, only: bgiapp_xmin, bgiapp_xmax, bgiapp_ymin, bgiapp_ymax, &
       bgiapp_width, bgiapp_height

  integer :: current_palette, key = -1000
  real(DP) :: xm, ym, xstep, ystep, x1, y1, x2, y2, &
       xm0, ym0, xstep0, ystep0
  logical :: init, flag, redraw
  character(len = 20) :: buf

  ! Getting the viewing region...
  x1 = bgiapp_xmin()
  x2 = bgiapp_xmax()
  y1 = bgiapp_ymin()
  y2 = bgiapp_ymax()
  max_x = bgiapp_width()
  max_y = bgiapp_height()

  ! Getting DEFAULT for initial boundary
  xm0 = 0.5_DP*(x1+x2)
  ym0 = 0.5_DP*(y1+y2)
```

```
xstep0 = (x2−x1)/2
ystep0 = (xstep0*max_y)/max_x

max_x = max_x−1
max_y = max_y−1

! Initial boundary
xm = xm0
ym = ym0
xstep = xstep0
ystep = ystep0

init = .true.
flag = .true.
redraw = .true.

call explain()

current_palette = 1
call bgi_palette()

do while (key /= KEY_ESC)

   x1 = xm−xstep
   y1 = ym−ystep
   x2 = xm+xstep
   y2 = ym+ystep

   if (redraw) then
      call biomorph_set(x1,y1,x2,y2)
      call refresh()

      if (flag) then
         call setcolor(WHITE)
         write(buf,*) max_iter
         call outtextxy(0,max_y−20,CString(trim(adjustl(buf))))
         flag = .false.
      end if
      redraw = .false.
   end if

   ! Wait for a key or mouse click
   key = getevent()

   select case (key)
   case (WM_LBUTTONDOWN, WM_WHEELUP)
      xm = x1+(x2−x1)*mousex()/max_x
      ym = y1+(y2−y1)*mousey()/max_y
      xstep = xstep/2
      ystep = ystep/2
      init = .false.
      redraw = .true.
   case (WM_RBUTTONDOWN, WM_WHEELDOWN)
      xstep = xstep*2
      ystep = ystep*2
      init = .false.
      redraw = .true.
   case (WM_MBUTTONDOWN)
      if (.not. init) then
         xm = xm0
         ym = ym0
         xstep = xstep0
         ystep = ystep0
         redraw = .true.
      end if
   case (ichar('1'))
      if (current_palette /= 1) then
         current_palette = 1
         call bgi_palette()
         redraw = .true.
      end if
   case (ichar('2'))
      if (current_palette /= 2) then
         current_palette = 2
         call bw_palette()
         redraw = .true.
      end if
   case (ichar('i'))
```

```
            max_iter = max_iter+50
            flag = .true.
            redraw = .true.
         case (ichar('d'))
            max_iter = max_iter−50
            flag = .true.
            redraw = .true.

         case default
            redraw = .false.

         end select
      end do
      call DeleteParser(fp_biomorph)
   end subroutine run_app
end module biomorph_lib

program biomorph
   use kind_consts, only: DP
   use general_routines, only: system_time
   use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
   use biomorph_lib
   implicit none
   real(DP) :: t0,t1

   call input_data()
   call bgiapp_setup(−4.0_DP,4.0_DP,−2.25_DP,2.25_DP,1600,900)
   call bgiapp_init("A tribute to Clifford Pickover's Biomorphs")

   write(*,'(A)',advance='NO') 'Please wait, we are working...'

   t0 = system_time()
   call run_app()
   t1 = system_time()

   write(*,*)
   write(*,'(A,F8.3,A)') 'Biomorph completed in ',t1−t0,' seconds!'

   call bgiapp_close()
end program biomorph
```

```
!
! Fortran Interface to the Xbgi−364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://www.iausofa.org/2013_1202_F/sofa_f−20131202_b.tar.gz
!   bsdtar −xvof sofa_f−20131202_b.tar.gz
!   mv sofa/20131202_b/f77 SOFAlib
!   apack SOFAlib−20131202_b−src.tar.xz SOFAlib
!   touch −r sofa_f−20131202_b.tar.gz SOFAlib−20131202_b−src.tar.xz
!   rm −rf sofa*
!   cd SOFAlib/src
!   make FC=gfortran[−mp−4.9] INSTALL_DIR=$HOME/work/SOFAlib
!   make FC=gfortran[−mp−4.9] INSTALL_DIR=$HOME/work/SOFAlib test
!   mv ~/work/SOFAlib/lib/libsofa.a ~/programming/lib/libSOFA.a
!   make clean
!   rm libsofa.a
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!   cd ~/programming/bgi−fortran/apps
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{bgi.f90,bgiapp.f90} solar_system.f90 \
!     −L ~/programming/lib −lSOFA −lXbgi −lX11 −lm \
!     −o solar_system.out
!
!   ./solar_system.out
!

module solar_system_lib
  use kind_consts, only: DP
  use get_data, only: get
  implicit none
  private

  real(DP) :: jd0, jd_tot = 36500.0_DP, jd_stp = 1.0_DP

  public :: input_data, run_app

contains

  function get_gregorian_date() result(jul)
    real(DP) :: jul
    integer :: year = 2000, month = 1, day = 1, ho = 12, mi = 0, se = 0, ierr
    real(DP) :: djm0, djm, day_frac

    write (*,*) 'IAU PLAN94 Planet Coordinates (See IAU SOFA Documentation)'
    write (*,*) 'Input Gregorian Date'
    call get('  YEAR    :',year)
    call get('  MONTH   :',month)
    call get('  DAY     :',day)
    write (*,*) 'Input Time of Day'

    call get('  HOURS      :',ho)
    call get('  Minutes    :',mi)
```

```
    call get('  SECONDS    :',se)
    day_frac = (ho+(mi/60.0_DP)+(se/3600.0_DP))/24.0_DP

    call iau_CAL2JD(year,month,day,djm0,djm,ierr)
    if (ierr == 0) then
       jul = djm0+djm+day_frac
    else
       write(*,*) 'An error occurred! Exiting...'
       stop
    end if
  end function get_gregorian_date

  subroutine input_data()
    ! Starting Gregorian date in JD
    write(*,*) 'STARTING TIME:'
    write(*,*)

    jd0 = get_gregorian_date()
    write(*,*)

    call get('Time intervall (JD) :',jd_tot)
    write(*,*)

    call get('Time step (JD) :',jd_stp)
    write(*,*)
  end subroutine input_data

  subroutine run_app
    use bgi, only: YELLOW
    use bgiapp, only: bgiapp_dot
    integer, parameter :: MAXP = 9
    integer :: body_color(MAXP) = 0
    integer  :: ierr, k         ! error flag, planet id
    real(DP) :: x, y            ! coordinates in the plane
    ! Julian time at which positions are computed; end intervall;
    ! positions and velocities
    real(DP) :: jd, jd1, pos(3,2)
    logical :: first

    ! Setup of the colors...
    do k = 1, MAXP
       body_color(k) = k
    end do

    first = .true.
    jd1 = jd0+jd_tot
    jd = jd0
    do while (jd <= jd1)

       !write(*,*) 'Current JD: ', jd
       if (first) then
          first = .false.
          x = 0
          y = 0

          ! The SUN!!!
          call bgiapp_dot(x,y,YELLOW)
       end if

       do k = 1, MAXP
          call iau_PLAN94 (0.0_DP,jd,k,pos,ierr)
          if (ierr == 0) then
             x = pos(1,1)
             y = pos(2,1)

             ! Planet k...
             call bgiapp_dot(x,y,body_color(k))
          end if
       end do
       jd = jd+jd_stp
    end do
  end subroutine run_app
end module solar_system_lib

program solar_system
  use kind_consts, only: DP
  use general_routines, only: system_time
  use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
```

```
  use solar_system_lib
  implicit none
  real(DP) :: t0,t1

  call input_data()
  call bgiapp_setup(−40.0_DP,40.0_DP,−40.0_DP,40.0_DP)
  call bgiapp_init('An example of SOFA anf BGI−Fortran usage')

  write(*,'(A)',advance='NO') 'Please wait, we are working...'

  t0 = system_time()
  call run_app()
  t1 = system_time()

  write(*,*)
  write(*,'(A,F8.3,A)') 'Solar system completed in ',t1−t0,' seconds!'

  call bgiapp_close()
end program solar_system
```

```
!
! Fortran Interface to the Xbgi−364p/WinBGIm−6.0p Libraries
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD XBGI (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!
! HOW TO BUILD WinBGIm−6.0 (MSYS2/MINGW64 shell)
!
!   cd ~/work/WinBGIm−6.0
!   make
!   mv libbgi.a ~/programming/lib/mingw64/libWinBGIm6.0.a
!
!   make clean
!
!   cd all−tests
!   g++ −O3 −Wall −mwindows −I .. test−bgidemo0.cxx \
!     −L ~/programming/lib/mingw64 −lWinBGIm6.0 \
!     −lgdi32 −lcomdlg32 −luuid −loleaut32 −lole32 −o test−bgidemo0
!
! HOW TO BUILD THE APP
!
!   cd ~/programming/bgi−fortran/apps
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall $BLD_OPTS −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ~/programming/fparser−fortran/fparser_dp.f90 \
!     ../{bgi.f90,bgiapp.f90} dynamics2d.f90 \
!     −L ~/programming/lib/$PLATFORM $LIBS −o dynamics2d$EXE
!
!   ./dynamics2d$EXE
!
!   where:
!
!     BLD_OPTS =
!     PLATFORM =
!     LIBS = −lFParser −lstdc++ −lXbgi −lX11 −lm
!     EXE = .out
!
!   for the build on GNU/Linux
!
!     $BLD_OPTS = −static [−mwindows]
!     $PLATFORM = mingw64
!     $LIBS = −lFParser −lWinBGIm6.0 \
!             −lgdi32 −lcomdlg32 −luuid −loleaut32 −lole32 −lstdc++
!     EXE =
!
!   for the build on MSYS2/MINGW64
!
!
! EXAMPLES
!
!   ax = '−2*x', ay = '−3*y', t in [0,100], h = 0.0005, P0(2,4), V0(−2,0)
!
!   ax = '−((x+2)/hypot(x+2,y)^3 + (x−2)/hypot(x−2,y)^3)',
!   ay = '−(y/hypot(x+2,y)^3 + y/hypot(x−2,y)^3)',
!   t in [0,100], h = 0.005, P0(1,4), V0(−0.25,0)
```

```
!

module dynamics2d_lib
  use kind_consts, only: DP
  use fparser_dp, only: FunctionParser_type, NewParser, Parse, &
       ErrorMsg, GetParseErrorType, DeleteParser, Eval
  implicit none
  private

  type(FunctionParser_type) :: fp_ax, fp_ay
  integer, parameter :: NEQ = 4
  real(DP) :: t0 = 0.0_DP, t1 = 25.0_DP, h = 0.05_DP, h2, h6

  !
  ! The first implementation was with these meaning
  !
  !   y(1) = x, y(2) = vx, y(3) = y, y(4) = vy
  !
  ! i.e. the system to be integrated was
  !
  !   y'(1) = y(2)
  !   y'(2) = ax(y(1:3:2))
  !   y'(3) = y(4)
  !   y'(4) = ay(y(1:3:2))
  !
  ! Now we use
  !
  !   y(1) = x, y(2) = y, y(3) = vx, y(4) = vy
  !
  ! i.e. the system to be integrated IS
  !
  !   y'(1) = y(3)
  !   y'(2) = y(4)
  !   y'(3) = ax(y(1:2))
  !   y'(4) = ay(y(1:2))
  !

  ! w(:,:) work space to compute K1, K2, K3, K4. Notice that the method uses
  ! w(:,3) and NOT w(:,4)!
  !
  real(DP) :: y0(NEQ) = [ 1.0_DP, 4.0_DP, -2.0_DP, 0.0_DP ], w(NEQ,3)

  public :: input_data, run_app

contains

  subroutine input_data()
    use get_data, only: get, MAXLEN
    character(len = MAXLEN) :: ax_buf = '-x', ay_buf = 'y'
    integer :: res

    call get('AX(x,y) = ',ax_buf)
    call get('AY(x,y) = ',ay_buf)
    write(*,*)

    call get('T0 = ',t0)
    call get('T1 = ',t1)
    call get('H = ',h)
    h2 = 0.5_DP*h
    h6 = h/6.0_DP
    write(*,*)

    call get('X0 = ',y0(1))
    call get('Y0 = ',y0(2))
    write(*,*)

    call get('VX0 = ',y0(3))
    call get('VY0 = ',y0(4))
    write(*,*)

    ! Create the fparser for AX(x,y)
    call NewParser(fp_ax)
    res = Parse(fp_ax,ax_buf,'x,y')

    if (res >= 0) then
       write(*,*) 'Failure creating fp_ax parser...'
       write(*,*)
       write(*,'(A)') 'AX(x,y) = '//trim(ax_buf)
```

```
      write(*,'(A)') repeat(' ',res+10)//'^'

      ! Remember : ErrorMsg() is an array of characters...
      write(*,*) ErrorMsg(fp_ax)
      write(*,*) 'Error type: ',GetParseErrorType(fp_ax)
      write(*,*)
      stop
    end if

    ! Create the fparser for AY(x,y)
    call NewParser(fp_ay)
    res = Parse(fp_ay,ay_buf,'x,y')
    if (res >= 0) then
      write(*,*) 'Failure creating fp_ay parser...'
      write(*,*)
      write(*,'(A)') 'AY(x,y) = '//trim(ay_buf)
      write(*,'(A)') repeat(' ',res+10)//'^'

      ! Remember : ErrorMsg() is an array of characters...
      write(*,*) ErrorMsg(fp_ay)
      write(*,*) 'Error type: ',GetParseErrorType(fp_ay)
      write(*,*)
      stop
    end if
  end subroutine input_data

  subroutine sub(x,y,f)
    real(DP), intent(in) :: x, y(:)
    real(DP), intent(out) :: f(:)
    f(1) = y(3)
    f(2) = y(4)

    f(3) = Eval(fp_ax,y(1:2))
    f(4) = Eval(fp_ay,y(1:2))
  end subroutine sub

  subroutine rk4step(x,y)
    real(DP), intent(inout) :: x, y(:)
    real(DP) :: xh, xh2
    !
    !  THIS SUBROUTINE REPLACES X BY X+H AND ADVANCES THE SOLUTION OF THE
    !  SYSTEM OF DIFFERENTIAL EQUATIONS DY/DX=F(X,Y) FROM Y(X) TO Y(X+H)
    !  USING A FIFTH-ORDER RUNGE-KUTTA METHOD.
    !
    !  SUB IS THE NAME OF A SUBROUTINE SUB(X,Y,F) WHICH SETS THE VECTOR F
    !  TO THE DERIVATIVE AT X OF THE VECTOR Y.
    !
    !  W IS A WORKING-SPACE ARRAY, TREATED AS CONSISTING OF THREE CONSEC-
    !  UTIVE WORKING VECTORS OF LENGTH NEQ.
    !
    !  Adapted from CERNLIB drkstp.F:
    !
    !    http://cernlib.sourcearchive.com/documentation/2005.05.09.dfsg/
    !    drkstp_8F_source.html
    !
    xh = x+h
    xh2 = x+h2

    ! Computing w(:,1) = K1
    call sub(x,y,w(:,1))

    ! Computing w(:,2) = y+H*K1/2
    w(:,2) = y(:)+h2*w(:,1)

    ! Computing w(:,3) = K2
    call sub(xh2,w(:,2),w(:,3))

    ! Computing w(:,1) = K1+2*K2
    w(:,1) = w(:,1)+2.0_DP*w(:,3)

    ! Computing w(:,2) = y+H*K2/2
    w(:,2) = y(:)+h2*w(:,3)

    ! Computing w(:,3) = K3
    call sub(xh2,w(:,2),w(:,3))

    ! Computing w(:,1) = (K1+2*K2)+2*K3
    w(:,1) = w(:,1)+2.0_DP*w(:,3)
```

```
    ! Computing w(:,2) = y+H*K3
    w(:,2) = y(:)+h*w(:,3)

    ! Computing w(:,3) = K4
    call sub(xh,w(:,2),w(:,3))

    ! Advance the solution Y(t+h) = Y(t) + H*[(K1+2*K2+2*K3)+K4]/6
    y(:)=y(:)+h6*(w(:,1)+w(:,3))

    x = xh
  end subroutine rk4step

  subroutine run_app()
    use bgi, only: YELLOW
    use bgiapp, only: bgiapp_dot
    real(DP) :: t, y(NEQ)

    t = t0
    y = y0
    do while (t < t1)

       call bgiapp_dot(y(1),y(2),YELLOW)

       ! We take a RK step
       call rk4step(t,y)
    end do

    call DeleteParser(fp_ax)
    call DeleteParser(fp_ay)
  end subroutine run_app
end module dynamics2d_lib

program dynamics2d
  use kind_consts, only: DP
  use general_routines, only: system_time
  use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
  use dynamics2d_lib
  implicit none

  real(DP) :: t0, t1

  call input_data()
  call bgiapp_setup(−5.0_DP,5.0_DP,−5.0_DP,5.0_DP)
  call bgiapp_init('Dynamics in 2D')

  write(*,'(A)',advance='NO') 'Please wait, we are working...'

  t0 = system_time()
  call run_app()
  t1 = system_time()

  write(*,*)
  write(*,'(A,F8.3,A)') 'Completed in ',t1−t0,' seconds!'

  call bgiapp_close()
end program dynamics2d
```

```
!
! Fortran Interface to the Xbgi-364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi-364.tar.gz
!   tar -xf xbgi-364.tar.gz
!   cd xbgi-364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!   cd ~/programming/bgi-fortran/apps
!
!   rm -rf {*.mod,~/programming/modules/*} && \
!   gfortran -O3 -Wall -J ~/programming/modules \
!     ~/programming/basic-modules/basic_mods.f90 \
!     ../{bgi.f90,bgiapp.f90} double_pendulum-DB.f90 \
!     -L ~/programming/lib -lXbgi -lX11 -lm -o double_pendulum-DB.out
!
!   ./double_pendulum-DB.out
!
!
! EXAMPLES
!
!   t in [0,25], h = 0.0005, m1 = 1.5, m2 = 1, l1 = 1, l2 = 0.5,
!   th1 = th2 = 90, omg1 = omg2 = 0
!

module double_pendulum_lib
  use kind_consts, only: DP
  implicit none
  private

  ! Units for input data:
  !
  !   length in meters
  !   time   in second
  !   mass   in kg
  !   angle  in deg
  !   omg    in deg/sec
  !
  integer, parameter :: NEQ = 4
  real(DP), parameter :: KGRAV = 9.81_DP, &
       Z3 = 1.0_DP/3, Z43 = 4*Z3
  real(DP) :: t0 = 0.0_DP, t1 = 25.0_DP, h = 0.0005_DP, &
       m1 = 0.8_DP, m2 = 1.2_DP, &        ! Kg
       rho1 = 10.49_DP, rho2 = 19.3_DP, & ! Ag, Au: in g/cm**3
       l1 = 0.5_DP, l2 = 1.1_DP, &
       h2, h6, &
       msum, msum1, r1, r2

  !
  ! We adopt the equation found in
  !
  !   http://www.myphysicslab.com/dbl_pendulum.html
  !
  ! i.e.
  !
  !   y(1) = th1, y(2) = th2, y(3) = omg1, y(4) = omg2
  !
  ! i.e. the system to be integrated IS
  !
  !   y'(1) = y(3)
```

```
  !   y'(2) = y(4)
  !   y'(3) = a1(y(1:2))
  !   y'(4) = a2(y(1:2))
  !
  ! w(:,:) work space to compute K1, K2, K3, K4. Notice that the method uses
  ! w(:,3) and NOT w(:,4)!
  !
  real(DP) :: y0(NEQ) = [ 60.0_DP, 100.0_DP, 0.0_DP, 0.0_DP ], w(NEQ,3)

  public :: input_data, run_app

contains

  subroutine input_data()
    use math_consts, only: DEG2RAD, PI
    use get_data, only: get

    call get('T0 (s) = ',t0)
    call get('T1 (s) = ',t1)
    call get('H (s) = ',h)
    h2 = 0.5_DP*h
    h6 = h/6.0_DP
    write(*,*)

    call get('M1 (kg) = ',m1)
    call get('M2 (kg) = ',m2)
    msum = m1+m2
    msum1 = msum+m1
    write(*,*)

    call get('RHO1 (g/cm**3) = ',rho1)
    call get('RHO2 (g/cm**3) = ',rho2)
    ! mass in Kg, mass*1000 in g, rho in g/cm**3, radius in cm,
    ! radius/100 in m
    r1 = (((m1*1000/rho1)/(Z43*PI))**Z3)/100
    r2 = (((m2*1000/rho2)/(Z43*PI))**Z3)/100
    ! print *, r1, r2
    write(*,*)

    call get('L1 (m) = ',l1)
    call get('L2 (m) = ',l2)
    write(*,*)

    call get('TH1 (deg) = ',y0(1))
    call get('TH2 (deg) = ',y0(2))
    write(*,*)

    call get('OMG1 (deg/s) = ',y0(3))
    call get('OMG2 (deg/s) = ',y0(4))
    write(*,*)

    ! Converting ALL angles in radians
    y0 = y0*DEG2RAD
  end subroutine input_data

  subroutine sub(x,y,f)
    real(DP), intent(in) :: x, y(:)
    real(DP), intent(out) :: f(:)
    ! y(1) = th1, y(2) = th2, y(3) = omg1, y(4) = omg2
    real(DP), save :: a, b, c, d, e, g

    ! a = th1-th2
    a = y(1)-y(2)

    ! b = (omg1**2) * l1
    b = l1*y(3)**2

    ! c = (omg2**2) * l2
    c = l2*y(4)**2

    ! d = 2*sin(th1-th2)
    d = 2.0_DP*sin(a)

    ! e = cos(th1-th2)
    e = cos(a)

    ! g = msum1-m2*cos(2*th1-2*th2)
```

```
  g = msum1−m2*cos(2.0_DP*a)

  ! We do not need th1−th2 any more. So a = sin(th1−2*th2))
  a = sin(a−y(2))

  ! Now computing the field
  f(1) = y(3)
  f(2) = y(4)
  f(3) = (−KGRAV*(msum1*sin(y(1))+m2*a)−m2*d*(c+b*e))/(l1*g)
  f(4) = (d*(msum*(b+KGRAV*cos(y(1)))+c*m2*e))/(l2*g)
end subroutine sub

subroutine rk4step(x,y)
  real(DP), intent(inout) :: x, y(:)
  real(DP) :: xh, xh2
  !
  !  THIS SUBROUTINE REPLACES X BY X+H AND ADVANCES THE SOLUTION OF THE
  !  SYSTEM OF DIFFERENTIAL EQUATIONS DY/DX=F(X,Y) FROM Y(X) TO Y(X+H)
  !  USING A FIFTH−ORDER RUNGE−KUTTA METHOD.
  !
  !  SUB IS THE NAME OF A SUBROUTINE SUB(X,Y,F) WHICH SETS THE VECTOR F
  !  TO THE DERIVATIVE AT X OF THE VECTOR Y.
  !
  !  W IS A WORKING−SPACE ARRAY, TREATED AS CONSISTING OF THREE CONSEC−
  !  UTIVE WORKING VECTORS OF LENGTH NEQ.
  !
  !  Adapted from CERNLIB drkstp.F:
  !
  !    http://cernlib.sourcearchive.com/documentation/2005.05.09.dfsg/
  !    drkstp_8F_source.html
  !
  xh = x+h
  xh2 = x+h2

  ! Computing w(:,1) = K1
  call sub(x,y,w(:,1))

  ! Computing w(:,2) = y+H*K1/2
  w(:,2) = y(:)+h2*w(:,1)

  ! Computing w(:,3) = K2
  call sub(xh2,w(:,2),w(:,3))

  ! Computing w(:,1) = K1+2*K2
  w(:,1) = w(:,1)+2.0_DP*w(:,3)

  ! Computing w(:,2) = y+H*K2/2
  w(:,2) = y(:)+h2*w(:,3)

  ! Computing w(:,3) = K3
  call sub(xh2,w(:,2),w(:,3))

  ! Computing w(:,1) = (K1+2*K2)+2*K3
  w(:,1) = w(:,1)+2.0_DP*w(:,3)

  ! Computing w(:,2) = y+H*K3
  w(:,2) = y(:)+h*w(:,3)

  ! Computing w(:,3) = K4
  call sub(xh,w(:,2),w(:,3))

  ! Advance the solution Y(t+h) = Y(t) + H*[(K1+2*K2+2*K3)+K4]/6
  y(:)=y(:)+h6*(w(:,1)+w(:,3))

  x = xh
end subroutine rk4step

subroutine run_app()
  use bgi, only: BROWN, clearviewport, getvisualpage, RED, &
       setactivepage, setcolor, setfillstyle, SOLID_FILL, swapbuffers, &
       YELLOW, WHITE
  use bgiapp, only: bgiapp_fillellipse, bgiapp_line
  real(DP) :: t, y(NEQ), x1, y1, x2, y2

  ! By default, the current visual and active page is 0 (zero),
  ! so we select the off screen page for drawing
  call setactivepage(1)
```

```
    t = t0
    y = y0
    do while (t < t1)
       ! The current active (off screen) page becomes the visual page
       ! and the current visual page becomes the off screen page, i.e.
       ! what is drawn on the off screen is outputted on the screen visible
       call swapbuffers()

       ! We clear the off screen for the next drawing
       call clearviewport()

       ! First pendulum: conversion from generalized to cartesian coordinates
       x1 = l1*sin(y(1))
       y1 = −l1*cos(y(1))

       ! Second pendulum: conversion from generalized to cartesian coordinates
       x2 = x1+l2*sin(y(2))
       y2 = y1−l2*cos(y(2))

       ! Draw arms positions on the off screen
       call setcolor(BROWN)
       call bgiapp_line(0.0_DP,0.0_DP,x1,y1)
       call bgiapp_line(x1,y1,x2,y2)

       ! Draw the origin on the off screen
       call setcolor(RED)
       call setfillstyle(SOLID_FILL,RED)
       call bgiapp_fillellipse(0.0_DP,0.0_DP,0.02_DP,0.02_DP)

       ! Draw the position of first pendulum on the off screen
       call setcolor(WHITE)
       call setfillstyle(SOLID_FILL,WHITE)
       call bgiapp_fillellipse(x1,y1,r1,r1)

       ! Draw the position of second pendulum on the off screen
       call setcolor(YELLOW)
       call setfillstyle(SOLID_FILL,YELLOW)
       call bgiapp_fillellipse(x2,y2,r2,r2)

       ! We take a RK step
       call rk4step(t,y)
    end do

    ! Making active page the same as visual page
    call setactivepage(getvisualpage())
  end subroutine run_app
end module double_pendulum_lib

program double_pendulum
  use kind_consts, only: DP
  use general_routines, only: system_time
  use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
  use double_pendulum_lib
  implicit none

  real(DP) :: t0, t1

  call input_data()
  call bgiapp_setup(−2.0_DP,2.0_DP,−2.0_DP,2.0_DP)
  call bgiapp_init('Double Pendulum')

  write(*,'(A)',advance='NO') 'Please wait, we are working...'

  t0 = system_time()
  call run_app()
  t1 = system_time()

  write(*,*)
  write(*,'(A,F8.3,A)') 'Completed in ',t1−t0,' seconds!'

  call bgiapp_close()
end program double_pendulum
```

```
!
! Fortran Interface to the Xbgi−364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!   cd ~/programming/bgi−fortran/apps
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ~/programming/ode−modules/ode_integrators.f90 \
!     ../{bgi.f90,bgiapp.f90} thomas_fermi.f90 \
!     −L ~/programming/lib −lXbgi −lX11 −lm −o thomas_fermi.out
!
!   ./thomas_fermi.out
!
!
! DESCRIPTION
!   BEST SLOPE for decreasing function u(x) (>= 0) satisying the
!   Thomas−Fermi equation and u(0) = 1 condition.
!   With GBS method and MU_ERR = 5.0E−9, we find
!
!     u'(0) = −MU = −1.5880710214376457 +− 2.9802322387695339E−009
!
!   (in X[0,69.491249999966996]) to be compared with
!
!     u'(0) = −1.588071022611375312718684508
!
!   as reported in
!
!     P.Amore et al., Accurate calculation of the solutions to the Thomas−Fermi,
!     http://arxiv.org/pdf/1205.1704v2.pdf (http://arxiv.org/abs/1205.1704)
!

module thomas_fermi_lib
  use kind_consts, only: DP
  implicit none
  private

  integer :: id_method = 2
  integer, parameter :: NEQ = 2, NC_MAX = 500
  real(DP), parameter :: Z0 = 0, Z1 = 1, Z4 = 4, &
       Q2 = Z1/2, Q43 = Z4/3
  integer :: nc_majo = 100
  real(DP) :: h = 0.00005_DP, eps = 1.0E−12_DP, mu_err = 5.0E−9_DP

  !
  ! To avoid the singularity at the origin, we use this transformation
  !
  !   u(x) = (1+(4/3) * x**(3/2))*y(x)
  !
  ! and the Thomas−Fermi eq. for u(x),
  !
  !   u''(x) = u(x) ** (3/2) / sqrt(x)
  !
  ! becomes, for y(x),
  !
```

```
  !   (1+(4/3) * x**(3/2))*y''(x) + 4*sqrt(x)*y'(x)
  !
  !     + (y(x)/sqrt(x))*(1−sqrt(y(x))*(1+(4/3) * x**(3/2))**(3/2)) = 0
  !
  ! Notice, the boundary conditions for neutral atoms satisfied by u(x),
  !
  !   u(0) = 1,  u(+inf) = 0
  !
  ! are the same for y(x), y(0) = 1,  y(+inf) = 0
  !
  ! Notice also, that u'(0) = y'(0).
  !

  public :: input_data, run_app

contains

  subroutine input_data()
    use get_data, only: get
    real(DP), parameter :: MACHEPS = epsilon(1.0_DP)

    write(*,*) 'Choose the method:'
    write(*,*) '  1 : RK4'
    write(*,*) '  2 : GBS'
    write(*,*) '  3 : RKM'
    call get('ID_METHOD =',id_method)
    ! For GBS or RKM step, the default initial H step can be greather..
    if (id_method == 2 .or. id_method == 3) h = 0.005_DP
    write(*,*)

    call get('MU_ERR = ',mu_err)
    call get('H = ',h)
    if (id_method == 2 .or. id_method == 3) then
       write(*,*)
       call get('EPS = ',eps)
       if (eps < 1000*MACHEPS) then
          write(*,*) 'EPS TOO SMALL! Exiting... '
          stop
       end if
    end if
    write(*,*)

    call get('NC_MAJO =',nc_majo)
    if (nc_majo < 3) nc_majo = 10
    if (nc_majo > NC_MAX) nc_majo = NC_MAX
    write(*,*)
  end subroutine input_data

  subroutine sub(x,y,f)
    real(DP), intent(in) :: x, y(:)
    real(DP), intent(out) :: f(:)
    real(DP), save :: a, b

    ! Now computing the field
    f(1) = y(2)

    if (x == Z0) then
       f(2) = Z0
    else
       a = sqrt(x)
       b = Z1+Q43*x*a

       ! We use abs(y(1)) as argument of sqrt() to avoid troubles when y(1) < 0
       f(2) = (y(1)*(b*sqrt(abs(y(1))*b)−Z1)−Z4*x*y(2))/(a*b)
    end if
  end subroutine sub

  subroutine majo_result()
    real(DP), parameter :: Z3 = 3, Z73 = 73, &
         R73 = sqrt(Z73), &
         A1 = 9−R73, A2 = (6497−755*R73)/152, &
         Q3_16 = Z3/16, Q3 = Z1/Z3, R3_16 = Q3_16 ** Q3
    !
    ! A simple implementation of the Majorana method as described in
    !
    !   S. Esposito, Majorana solution of the Thomas−Fermi equation,
    !   Am. J. Phys. 70, 852 (2002).
    !
```

```
   integer, save :: m, n, &
        mm2, mm1, nm1, &
        m1, m3_2, m6, m7, m8_2, &
        n1, n4_2, n7
   real(DP), save :: sum_val, a(0:NC_MAX), tt(NC_MAX−2)

   write(*,*)
   write(*,'(A)',advance='NO') 'Computing MU with Majorana method...'

   a(0:2) = [Z1, A1, A2]
   ! print *
   ! print *, a(0)
   ! print *, a(1)
   ! print *, a(2)

   do m = 3, nc_majo
      mm1 = m−1
      mm2 = m−2
      m1 = m+1
      m3_2 = 2*(m+3)
      m6 = m+6
      m7 = m+7
      m8_2 = 2*(m+8)

      n = mm2
      nm1 = n−1
      n1 = n+1
      n4_2 = 2*(n+4)
      n7 = n+7
      tt(n) = n1*a(n1)−n4_2*a(n)+n7*a(nm1)

      sum_val = Z0
      do n = 1, mm2
         sum_val = sum_val+a(m−n)*tt(n)
      end do

      ! Partial value
      a(m) = sum_val+a(mm1)*(m7−m3_2*A1)+a(mm2)*m6*A1

      ! Final value
      a(m) = a(m)/(m8_2−m1*A1)
      !print *, a(m)
   end do

   ! The MU value as computed with Majorana method
   sum_val = R3_16*sum(a(:nc_majo))

   write(*,*)
   write(*,*) 'MU(MAJO) = ', sum_val, &
        'with N = ', nc_majo+1, 'coefficients...'
end subroutine majo_result

subroutine run_app()
   use bgi, only: YELLOW
   use bgiapp, only: bgiapp_dot
   use ode_integrators, only: rk4step, deqgbs, deqrkm

   ! For RK4 w(NEQ,3) would be sufficient...
   ! For RKM w(NEQ,6) would be sufficient...
   ! For GBS we need w(NEQ,36)...
   !
   ! We assume mu in (1.5,1.6) and an initial guess mu = 1.6
   !
   real(DP) :: x, xz, y1_old, y(NEQ), w(NEQ,36), h0, &
        mu = 1.6_DP, delta_mu = 1.6_DP−1.5_DP

   do
      h0 = h
      x = Z0
      y = [ Z1, −mu ]

      ! Just a little greater, so that the follwing loop is executed
      ! at least one time
      y1_old = y(1)+0.1_DP

      do while (y(1) >= Z0 .and. y(1) < y1_old)

         call bgiapp_dot(x,y(1),YELLOW)
```

```
        ! We take an ode integrator step
        y1_old = y(1)
        if (id_method == 1) then
           call rk4step(NEQ,h,x,y,w,sub)
        else
           h = h0
           xz = x+h
           if (id_method == 2) then
              call deqgbs(NEQ,x,xz,y,h,eps,w,sub)
           else
              call deqrkm(NEQ,x,xz,y,h,eps,w,sub)
           end if
           x = xz
        end if
        !print *, x,y(1)
     end do

     write(*,*) 'MU = ', mu, 'X = ', x

     if (abs(delta_mu) < mu_err) exit

     delta_mu = sign(abs(Q2*delta_mu),y(1))
     mu = mu+delta_mu
  end do

  write(*,*)
  write(*,*) 'MU = ', mu, 'DELTA_MU = ', delta_mu
  write(*,*)  'X = ', x, 'y(X) = ', y(1)

  call majo_result()
end subroutine run_app

! subroutine majo_result()
!   real(DP), parameter :: Z3 = 3, Z73 = 73, &
!        R73 = sqrt(Z73), &
!        A1 = 9−R73, A2 = (6497−755*R73)/152, &
!        Q3_16 = Z3/16, Q3 = Z1/Z3, R3_16 = Q3_16 ** Q3
!   !
!   ! A simple implementation of the Majorana method as described in
!   !
!   !   S. Esposito, Majorana solution of the Thomas−Fermi equation,
!   !   Am. J. Phys. 70, 852 (2002).
!   !
!   integer, save :: m, n, &
!        mm2, mm1, nm1, &
!        m1, m3_2, m6, m7, m8_2, &
!        n1, n4_2, n7
!   real(DP), save :: sum_val, a(0:NC_MAX)

!   write(*,*)
!   write(*,'(A)',advance='NO') 'Computing MU with Majorana method...'

!   a(0:2) = [Z1, A1, A2]
!   ! print *
!   ! print *, a(0)
!   ! print *, a(1)
!   ! print *, a(2)

!   do m = 3, nc_majo
!      mm1 = m−1
!      mm2 = m−2
!      m1 = m+1
!      m3_2 = 2*(m+3)
!      m6 = m+6
!      m7 = m+7
!      m8_2 = 2*(m+8)

!      sum_val = Z0
!      do n = 1, mm2
!         nm1 = n−1
!         n1 = n+1
!         n4_2 = 2*(n+4)
!         n7 = n+7

!         sum_val = sum_val+(a(m−n)*(n1*a(n1)−n4_2*a(n)+n7*a(nm1)))
!      end do
```

```
  !      ! Partial value
  !      a(m) = sum_val+a(mm1)*(m7−m3_2*A1)+a(mm2)*m6*A1

  !      ! Final value
  !      a(m) = a(m)/(m8_2−m1*A1)
  !      !print *, a(m)
  !   end do

  !   ! The MU value as computed with Majorana method
  !   sum_val = R3_16*sum(a(:nc_majo))

  !   write(*,*)
  !   write(*,*) 'MU(MAJO) = ', sum_val, &
  !        'with N = ', nc_majo+1, 'coefficients...'
  ! end subroutine majo_result
end module thomas_fermi_lib

program thomas_fermi
  use kind_consts, only: DP
  use general_routines, only: system_time
  use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
  use thomas_fermi_lib
  implicit none
  real(DP) :: t0, t1

  call input_data()
  call bgiapp_setup(0.0_DP,100.0_DP,−0.1_DP,1.1_DP,1000,500)
  call bgiapp_init('Thomas−Fermi Functions')

  write(*,*) 'Please wait, we are working...'

  t0 = system_time()
  call run_app()
  t1 = system_time()

  write(*,*)
  write(*,'(A,F8.3,A)') 'Completed in ',t1−t0,' seconds!'

  call bgiapp_close()
end program thomas_fermi
```

```
!
! Fortran Interface to the Xbgi−364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!   cd ~/programming/bgi−fortran/apps
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{bgi.f90,bgiapp.f90} logistics.f90 \
!     −L ~/programming/lib −lXbgi −lX11 −lm −o logistics.out
!
!   ./logistics.out
!
!
! DESCRIPTION
!   We iterate the logistic equation,
!
!     p(t+1) = r * p(t) * (1−p(t))
!
!   max_iter times to say we have reached the convergence, then we plot the
!   next npoint iterations. All this with r parameter varying in the interval
!   [r_min,r_max], at steps of r_stp
!
! A SAMPLE OF INPUT PARAMETERS
!   p0 = 0.7, r in [2.9,3.9], r_stp = 0.001, max_iter = 1000, npoints = 100
!   p0 = 0.7, r in [0.0,4.0], r_stp = 0.001, max_iter = 1000, npoints = 100
!   in a window 1000 x 268 pixels
!

module logistics_lib
  use kind_consts, only: DP
  implicit none
  private

  integer :: max_iter = 1000, npoints = 100
  real(DP) :: p0 = 0.7_DP, r_min = 2.9_DP, r_max = 3.9_DP, &
       r_stp = 0.001_DP

  public :: input_data, run_app

contains

  subroutine input_data()
    use get_data, only: get
    !
    ! YOU CANNOT CALL BGI ROUTINES HERE!
    !
    call get('MAX_ITER = ',max_iter)
    call get('NPOINTS = ',npoints)
    write(*,*)

    call get('P0 = ',p0)
    write(*,*)

    call get('R_MIN = ',r_min)
```

```
    call get('R_MAX = ',r_max)
    call get('R_STP = ',r_stp)
    write(*,*)
  end subroutine input_data

  subroutine run_app
    use bgi, only: YELLOW
    use bgiapp, only: bgiapp_dot
    integer  :: k, l, n_rstp
    real(DP) :: r, p

    ! Number of r steps, rounding up
    n_rstp = 1+int((r_max−r_min)/r_stp)

    r = r_min
    do l = 1, n_rstp
       p = p0
       do k = 1, max_iter
          p = r*p*(1.0_DP−p)
       end do

       ! Now we assume having reached the "convergence", i.e. a fix,
       ! oscillating or chaotic limit. So we can plot at most npoints points
       do k = 1, npoints
          p = r*p*(1.0_DP−p)

          ! Drawing point k...
          call bgiapp_dot(r,p,YELLOW)
       end do

       r = r+r_stp
    end do
  end subroutine run_app
end module logistics_lib

program logistics
  use kind_consts, only: DP
  use general_routines, only: system_time
  use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
  use logistics_lib
  implicit none
  real(DP) :: t0,t1

  call input_data()
  call bgiapp_setup(2.85_DP,3.95_DP,−0.05_DP,1.05_DP)
  call bgiapp_init('Logistics Equation Iterations, P(t) vs R')

  write(*,'(A)',advance='NO') 'Please wait, we are working...'

  t0 = system_time()
  call run_app()
  t1 = system_time()

  write(*,*)
  write(*,'(A,F8.3,A)') 'Completed in ',t1−t0,' seconds!'

  call bgiapp_close()
end program logistics
```

```
!
! Fortran Interface to the Xbgi−364p/WinBGIm−6.0 Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD XBGI (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!
! HOW TO BUILD WinBGIm−6.0 (MSYS2/MINGW64 shell)
!
!   cd ~/work/WinBGIm−6.0
!   make
!   mv libbgi.a ~/programming/lib/mingw64/libWinBGIm6.0.a
!
!   make clean
!
!   cd all−tests
!   g++ −O3 −Wall −mwindows −I .. test−bgidemo0.cxx \
!     −L ~/programming/lib/mingw64 −lWinBGIm6.0 \
!     −lgdi32 −lcomdlg32 −luuid −loleaut32 −lole32 −o test−bgidemo0
!
! HOW TO BUILD THE APP
!
!   cd ~/programming/bgi−fortran/apps
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall $BLD_OPTS −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{bgi.f90,bgiapp.f90} radio_decay.f90 \
!     −L ~/programming/lib/$PLATFORM $LIBS −o radio_decay$EXE
!
!   ./radio_decay$EXE
!
!   where:
!
!     BLD_OPTS =
!     PLATFORM =
!     LIBS = −lXbgi −lX11 −lm
!     EXE = .out
!
!   for the build on GNU/Linux
!
!     $BLD_OPTS = −static [−mwindows]
!     $PLATFORM = mingw64
!     $LIBS = −lWinBGIm6.0 −lgdi32 −lcomdlg32 −luuid −loleaut32 −lole32 \
!             −lstdc++
!     EXE =
!
!   for the build on MSYS2/MINGW64
!
!
! NOTES
!
!   An idea from
!
!     Dean Karlen ""Physics 75.502/487 − Computational Physics −
!     Fall/Winter 1998/99""
!
```

```
module radio_decay_lib
  use kind_consts, only: DP
  implicit none
  private
  integer :: num_nuclei = 100
  real(DP) :: alpha = 0.01_DP, t_step = 1, total_time = 300
  real(DP) :: t_min, t_max, n_min, n_max

  public :: input_data, run_app, t_min, t_max, n_min, n_max

contains

  subroutine input_data()
    use get_data, only: get
    call get('NUM_NUCLEI = ',num_nuclei)
    write(*,*)
    call get('ALPHA = ',alpha)
    write(*,*)
    call get('T_STEP = ',t_step)
    write(*,*)
    call get('TOTAL_TIME =',total_time)
    write(*,*)

    ! We take a margin of about 5% over the interval
    t_min = 0.05_DP*total_time
    t_max = 0 + (total_time+t_min)
    t_min = 0 − t_min

    n_min = 0.05_DP*num_nuclei
    n_max = 0 + (num_nuclei+n_min)
    n_min = 0 − n_min
  end subroutine input_data

  subroutine run_app()
    use bgi, only: LIGHTRED, setcolor, YELLOW
    use bgiapp, only: bgiapp_line
    integer  :: i, n_parents, n
    real(DP) :: r, p, t1, t, n1_exp, n_exp, n1_the, n_the

    ! Initialization...
    p = alpha*t_step
    n_parents = num_nuclei
    n = n_parents

    t = 0

    ! Initializing "prev" variables, i.e. variables at "previous" time
    t1 = t
    n1_exp = n_parents
    n1_the = n_parents

    ! LOOP over time..
    do while (t < total_time)
       ! LOOP over each remaining parent nucleus
       do i = 1, n
          call random_number(r)

          ! Decide if the nucleus decays..
          ! If it decays, reduce the number of parents by 1
          if (r < p) n_parents = n_parents−1
       end do

       ! Update time to current
       t = t+t_step

       ! The "experimental" result at current time
       n_exp = n_parents

       ! The "expected" result at current time
       n_the = num_nuclei*exp(−alpha*t)

       ! PLOT N vs. t : "experimental"...
       call setcolor(YELLOW)
       call bgiapp_line(t1,n1_exp,t,n_exp)

       ! ..."expected" or "theoretical"
       call setcolor(LIGHTRED)
       call bgiapp_line(t1,n1_the,t,n_the)
```

```
        ! Update current number of nuclei and the plotting "positions"..
        n = n_parents
        t1 = t
        n1_exp = n_exp
        n1_the = n_the
      end do
    end subroutine run_app
end module radio_decay_lib

program radio_decay
    use kind_consts, only: DP
    use general_routines, only: system_time
    use randoms, only: init_random_seed
    use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
    use radio_decay_lib
    implicit none
    real(DP) :: t0, t1

    call init_random_seed()

    call input_data()
    call bgiapp_setup(t_min,t_max,n_min,n_max,900,900)
    call bgiapp_init('Simulating Radioactive Decay')

    write(*,*) 'Please wait, we are working...'

    t0 = system_time()
    call run_app()
    t1 = system_time()

    write(*,*)
    write(*,'(A,F8.3,A)') 'Completed in ',t1−t0,' seconds!'

    call bgiapp_close()
end program radio_decay
```

```
!
! Fortran Interface to the Xbgi−364p/WinBGIm−6.0p Libraries
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD XBGI (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!
! HOW TO BUILD WinBGIm−6.0 (MSYS2/MINGW64 shell)
!
!   cd ~/work/WinBGIm−6.0
!   make
!   mv libbgi.a ~/programming/lib/mingw64/libWinBGIm6.0.a
!
!   make clean
!
!   cd all−tests
!   g++ −O3 −Wall −mwindows −I .. test−bgidemo0.cxx \
!     −L ~/programming/lib/mingw64 −lWinBGIm6.0 \
!     −lgdi32 −lcomdlg32 −luuid −loleaut32 −lole32 −o test−bgidemo0
!
! HOW TO BUILD THE APP
!
!   cd ~/programming/bgi−fortran/apps
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall $BLD_OPTS −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{bgi.f90,bgiapp.f90} joule_expansion.f90 \
!     −L ~/programming/lib/$PLATFORM $LIBS −o joule_expansion$EXE
!
!   ./joule_expansion$EXE
!
!   where:
!
!     BLD_OPTS =
!     PLATFORM =
!     LIBS = −lXbgi −lX11 −lm
!     EXE = .out
!
!   for the build on GNU/Linux
!
!     $BLD_OPTS = −static [−mwindows]
!     $PLATFORM = mingw64
!     $LIBS = −lWinBGIm6.0 −lgdi32 −lcomdlg32 −luuid −loleaut32 −lole32 \
!             −lstdc++
!     EXE =
!
!   for the build on MSYS2/MINGW64
!

module joule_expansion_lib
  use kind_consts, only: DP
  implicit none
  private

  integer :: natoms = 1000           ! Number of atoms
  real(DP) :: x0 = 0.25_DP, l = 3, h ! boxes offset, size, half size
```

```
  public :: input_data, run_app

contains

  subroutine input_data()
    use get_data, only: get

    call get('NATOMS = ',natoms)
    call get('X0 = ',x0)
    call get('L = ',l)
    write(*,*)

    h = l*0.5_DP
  end subroutine input_data

  subroutine draw_gasbox(n,a,b)
    use bgi, only: CENTER_TEXT, RED, setcolor, settextjustify, TOP_TEXT, &
         YELLOW, WHITE
    use bgiapp, only: bgiapp_box, bgiapp_dot, bgiapp_text
    integer, intent(in) :: n
    real(DP), intent(in) :: a(:), b(:) ! The top−left and bottom−right corners
    integer :: i
    real(DP) :: x1, x2, y1, y2, x, y, u(2)
    character(len = 20) :: buf

    x1 = a(1)
    x2 = b(1)
    y1 = b(2)
    y2 = a(2)
    write(buf,*) n

    call setcolor(RED)
    call bgiapp_box(x1,x2,y1,y2)

    call settextjustify(CENTER_TEXT,TOP_TEXT)
    call setcolor(WHITE)
    call bgiapp_text((x1+x2)/2,y1−h/4,trim(adjustl(buf)))

    do i = 1, n
       call random_number(u)

       x = x1+u(1)*(x2−x1)
       y = y1+u(2)*(y2−y1)

       call bgiapp_dot(x,y,YELLOW)
    end do
  end subroutine draw_gasbox

  subroutine run_app()
    use bgi, only: clearviewport, delay, getvisualpage, quit, setactivepage, &
         swapbuffers
    real(DP), dimension(2) :: a1, b1, c1, d1, a2, b2, c2, d2, dx, dy
    real(DP) :: u, p
    integer :: n1, n2

    ! Steps to "build" the boxes
    dx = [ l, 0.0_DP ]
    dy = [ 0.0_DP, l ]

    ! Boxes initialization
    ! first...
    a1 = [ −x0, −h ]
    b1 = a1−dx
    c1 = b1+dy
    d1 = c1+dx

    ! second...
    a2 = [ x0, −h ]
    b2 = a2+dx
    c2 = b2+dy
    d2 = c2−dx

    ! Initialization of the number of atoms in boxes
    n1 = natoms
    n2 = 0

    ! By default, the current visual and active page is 0 (zero),
    ! so we select the off screen page for drawing
```

```
    call setactivepage(1)

    ! Draw gas boxes on the off screen
    call draw_gasbox(n1,c1,a1)
    call draw_gasbox(n2,d2,b2)

    ! Main loop
    do while (.not.quit())
       ! The current active (off screen) page becomes the visual page
       ! and the current visual page becomes the off screen page, i.e.
       ! what is drawn on the off screen is outputted on the screen visible
       call swapbuffers()

       ! We clear the off screen forq the next drawing
       call clearviewport()

       ! Computing the expansion...
       p = (n1+0.0_DP)/natoms
       call random_number(u)

       if (u < p) then
          if (n1 > 0) n1 = n1−1
       else
          if (n1 < natoms) n1 = n1+1
       end if
       n2 = natoms−n1

       ! Draw gas expansion boxes on the off screen
       call draw_gasbox(n1,c1,a1)
       call draw_gasbox(n2,d2,b2)

       call delay(1)
    end do

    ! Making active page the same as visual page
    call setactivepage(getvisualpage())
  end subroutine run_app
end module joule_expansion_lib

program joule_expansion
  use kind_consts, only: DP
  use general_routines, only: system_time
  use randoms, only: init_random_seed
  use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
  use joule_expansion_lib
  implicit none

  real(DP) :: t0, t1

  call init_random_seed()

  call input_data()
  call bgiapp_setup(−4.0_DP,4.0_DP,−3.0_DP,3.0_DP,800,600)
  call bgiapp_init('Joule Expansion')

  write(*,'(A)',advance='NO') 'Please wait, we are working...'

  t0 = system_time()
  call run_app()
  t1 = system_time()

  write(*,*)
  write(*,'(A,F9.3,A)') 'Completed in ',t1−t0,' seconds!'

  call bgiapp_close()
end program joule_expansion
```

```
!
! Fortran Interface to the Xbgi−364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!   cd ~/programming/bgi−fortran/apps
!
!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall −J ~/programming/modules \
!     ~/programming/basic−modules/basic_mods.f90 \
!     ../{bgi.f90,bgiapp.f90} balls_sim.f90 \
!     −L ~/programming/lib −lXbgi −lX11 −lm −o balls_sim.out
!
!   ./balls_sim.out
!

module balls_sim_lib
  use kind_consts, only: DP
  implicit none
  private

  type ball_type
     real(DP) :: mass = 0.0_DP, &
          density = 0.0_DP, &
          radius = 0.0_DP
     real(DP), dimension(2) :: frc = 0.0_DP, &
          acc = 0.0_DP, &
          vel = 0.0_DP, &
          pos = 0.0_DP
  end type ball_type

  integer :: nballs = 12
  real(DP) :: density = 0.01_DP, stiffnes = 5.0E5_DP
  real(DP) :: m0 = 400.0_DP, m1 = 8000.0_DP, &
       tstep = 1.0_DP/512  ! 1.953125E−03 = 0.000000001_2
  real(DP) :: box_xmin, box_xmax, box_ymin, box_ymax
  type(ball_type), allocatable :: ball(:)

  public :: balls_on, balls_off, input_data, run_app, setup_balls

contains

  subroutine balls_on()
    integer :: ierr

    allocate(ball(nballs),stat=ierr)
    if (ierr /= 0) then
       write(*,*) '*** FATAL ERROR ***'
       write(*,*) 'BALL: Allocation request denied'
       stop
    end if
  end subroutine balls_on

  subroutine balls_off()
    integer :: ierr

    if (allocated(ball)) deallocate(ball,stat=ierr)
```

```
  if (ierr /= 0) then
     write(*,*) '*** FATAL ERROR ***'
     write(*,*) 'BALL: Deallocation request denied'
     stop
  end if
end subroutine balls_off

subroutine input_data()
  use get_data, only: get

  call get('NBALLS = ',nballs)
  call get('DENSITY = ',density)
  call get('STIFFNES = ',stiffnes)
  call get('TSTEP = ',tstep)
  write(*,*)

  call get('M0 = ',m0)
  call get('M1 = ',m1)
end subroutine input_data

subroutine setup_balls()
  use math_consts, only: PI
  use randoms, only: init_random_seed
  use bgi, only: setrgbpalette
  use bgiapp, only: bgiapp_xmin, bgiapp_xmax, bgiapp_ymin, bgiapp_ymax
  real(DP), parameter :: Z3 = 1.0_DP/3, Z43PI = 4*Z3*PI
  real(DP) :: u(9)
  integer  :: i

  ! Getting the box boundaries...
  box_xmin = bgiapp_xmin()
  box_xmax = bgiapp_xmax()
  box_ymin = bgiapp_ymin()
  box_ymax = bgiapp_ymax()

  ! Use timer to generate random numbers
  call init_random_seed()

  ! Set startup conditions of elastic balls
  do i = 1, nballs
     call random_number(u)

     ! The RGB color for i−ball
     call setrgbpalette(i,int(64+u(1)*192),int(64+u(2)*192),int(64+u(3)*192))

     ball(i)%mass = m0+(i−1)*(m1−m0)/(nballs−1)
     ball(i)%density = density

     ball(i)%radius = ((ball(i)%mass/ball(i)%density)/Z43PI)**Z3

     ball(i)%pos = [ (1.0_DP−u(4))*(box_xmin+ball(i)%radius) &
          +u(4)*(box_xmax−ball(i)%radius), &
          (1.0_DP−u(5))*(box_ymin+ball(i)%radius) &
          +u(5)*(box_ymax−ball(i)%radius) ]

     ball(i)%vel = 200*[ u(6)−u(7), u(8)−u(9) ]
  end do
end subroutine setup_balls

subroutine draw_ball(p,r,col)
  use bgi, only: setrgbcolor
  use bgiapp, only: bgiapp_circle
  real(DP), intent(in) :: p(:), r
  integer, intent(in) :: col

  call setrgbcolor(col)
  call bgiapp_circle(p(1),p(2),r)
  call bgiapp_circle(p(1),p(2),r−0.5_DP)
  call bgiapp_circle(p(1),p(2),r−1.0_DP)
end subroutine draw_ball

subroutine run_app()
  use bgi, only: clearviewport, delay, getvisualpage, quit, setactivepage, &
       swapbuffers
  real(DP) :: force(2), ball_distance, dist_min, dst(2)
  integer :: i, j

  ! By default, the current visual and active page is 0 (zero),
```

```
! so we select the off screen page for drawing
call setactivepage(1)

! Draw elastic balls on the off screen page
do i = 1, nballs
   call draw_ball(ball(i)%pos,ball(i)%radius,i)
end do

! Main loop
do while (.not.quit())
   ! The current active (off screen) page becomes the visual page
   ! and the current visual page becomes the off screen page, i.e.
   ! what is drawn on the off screen is outputted on the screen visible
   call swapbuffers()

   ! We clear the off screen for the next drawing
   call clearviewport()

   ! Test all elastic balls against each other.
   ! Calculate forces if they touch.
   do i = 1, nballs−1
      do j = i+1, nballs
         ! Distance between elastic balls (Pythagoras' theorem)
         dst = ball(j)%pos−ball(i)%pos
         ball_distance = norm2(dst)
         dist_min = ball(i)%radius+ball(j)%radius

         if (ball_distance < dist_min) then
            ! Cosine and sine to the angle between ball i and j
            ! (trigonometry): here 'force' is a unit vector!
            force = dst/ball_distance

            ! Spring force (Hooke's law of elasticity)
            ! Here 'force' is the total force of 'i' on 'j'
            ! (All capital letters are vectors)
            !
            !   F(i −> j) = −k * S = −k*(Bd−Dm) = −k*(|Bd|−|Dm|)*U
            !   U = Bd/|Bd|
            force = −stiffnes*(ball_distance−dist_min)*force

            ! F(i) = F(i)+F(j,i) = F(i)−F(i,j), F(j) = F(j)+F(i,j)
            ! being F(i,j) the force of 'i' on 'j'
            ball(i)%frc = ball(i)%frc−force
            ball(j)%frc = ball(j)%frc+force
         end if
      end do
   end do

   !  Update acceleration, velocity, and position of elastic balls
   !  (using the Euler−Cromer 1st order integration algorithm)
   do i = 1, nballs
      ! Accelerate balls (acceleration = force / mass)
      ball(i)%acc = ball(i)%frc/ball(i)%mass

      ! Reset force vector
      ball(i)%frc = 0.0_DP

      ! Update velocity
      ! delta velocity = acceleration * delta time
      ! new velocity = old velocity + delta velocity
      ball(i)%vel = ball(i)%vel+ball(i)%acc*tstep

      ! Update position
      ! delta position = velocity * delta time
      ! new position = old position + delta position
      ball(i)%pos = ball(i)%pos+ball(i)%vel*tstep
   end do

   ! Keep elastic balls within screen boundaries
   do i = 1, nballs
      ! Right
      if (ball(i)%pos(1) > box_xmax−ball(i)%radius) then
         ball(i)%vel(1) = −ball(i)%vel(1)
         ball(i)%pos(1) = box_xmax−ball(i)%radius
      end if

      ! Left
      if (ball(i)%pos(1) < box_xmin+ball(i)%radius) then
```

```
            ball(i)%vel(1) = −ball(i)%vel(1)
            ball(i)%pos(1) = box_xmin+ball(i)%radius
         end if

         ! Top
         if (ball(i)%pos(2) > box_ymax−ball(i)%radius) then
            ball(i)%vel(2) = −ball(i)%vel(2)
            ball(i)%pos(2) = box_ymax−ball(i)%radius
         end if

         ! Bottom
         if (ball(i)%pos(2) < box_ymin+ball(i)%radius) then
            ball(i)%vel(2) = −ball(i)%vel(2)
            ball(i)%pos(2) = box_ymin+ball(i)%radius
         end if
      end do

      ! Draw elastic balls update positions on the off screen
      do i = 1, nballs
         call draw_ball(ball(i)%pos,ball(i)%radius,i)
      end do
      call delay(1)
   end do

   ! Making active page the same as visual page
   call setactivepage(getvisualpage())
  end subroutine run_app
end module balls_sim_lib

program balls_sim
  use kind_consts, only: DP
  use general_routines, only: system_time
  use bgiapp, only: bgiapp_setup, bgiapp_init, bgiapp_close
  use balls_sim_lib
  implicit none

  real(DP) :: t0, t1

  call input_data()
  call balls_on()

  call bgiapp_setup(−400.0_DP,400.0_DP,−300.0_DP,300.0_DP,800,600)
  call bgiapp_init('Bouncing Balls Simulation')
  call setup_balls()

  write(*,'(A)',advance='NO') 'Please wait, we are working...'

  t0 = system_time()
  call run_app()
  t1 = system_time()

  write(*,*)
  write(*,'(A,F9.3,A)') 'Completed in ',t1−t0,' seconds!'

  call bgiapp_close()
  call balls_off()
end program balls_sim
```

```
!
! Fortran Interface to the Xbgi−364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! HOW TO BUILD (GNU/Linux Mint)
!
!   cd ~/work
!   wget http://libxbgi.sourceforge.net/xbgi−364.tar.gz
!   tar −xf xbgi−364.tar.gz
!   cd xbgi−364/src
!   make
!   make demo
!
!   ./demo
!   make clean
!   cd test
!   make
!   ./mandelbrot
!
!   cd ..
!   mv libXbgi.a ~/programming/lib
!   cd ~/programming/bgi−fortran/demo

!   rm −rf {*.mod,~/programming/modules/*} && \
!   gfortran −O3 −Wall −J ~/programming/modules \
!     ../bgi.f90 hopalong.f90 \
!     −L ~/programming/lib −lXbgi −lX11 −lm −o hopalong.out
!
!   ./hopalong.out
!

program hopalong
  use bgi, only: BLACK, X11, X11_1280x1024, YELLOW, &
       cleardevice, closegraph, rgb_color, CString, detectgraph, &
       getmaxx, getmaxy, initgraph, kbhit, outtextxy, fast_putpixel, &
       refresh, setbkcolor, setcolor
  implicit none

  integer :: gd = X11, gm = X11_1280x1024, counter
  real :: j, k, x, y, xx, xp, yp, xoffs, yoffs, u(3)
  logical :: stop_app

  call init_random_seed()

  !call detect_graph(gd,gm)
  call initgraph(gd,gm,CString(''))

  call setbkcolor(BLACK)
  call cleardevice()
  call setcolor(YELLOW)
  call outtextxy(0,0,CString('Press a key to exit...'))

  xoffs = getmaxx() / 2.
  yoffs = getmaxy() / 3.

  call random_number(u(1:2))

  j = u(1)*100.
  k = u(2)*100.

  x = 0.
  y = 0.
  xx = 0.
  xp = 0.
  yp = 0.

  ! Random RGB
  call random_number(u(1:3))
  call setcolor(rgb_color(int(u(1)*256.),int(u(2)*256.),int(u(3)*256.)))

  counter = 0
  stop_app = .false.
  do while (.not. stop_app)
```

```
      xx = sqrt(abs(k*x−1.))
      xx = y−sign(xx,x)
      y = j−x
      x = xx
      xp = 2*x+xoffs
      yp = 2*y+yoffs
      call fast_putpixel(int(xp),int(yp))
      counter = counter+1
      if (counter == 50000) then
         counter = 0

         ! Random RGB
         call random_number(u(1:3))
         call setcolor(rgb_color(int(u(1)*256.),int(u(2)*256.),int(u(3)*256.)))

         call refresh ()
         if (kbhit() /= 0) stop_app = .true.
      end if
   end do

   call closegraph()

contains

   subroutine init_random_seed()
     integer ::  i = 0,n,clock
     integer, dimension(:), allocatable :: seed
     call random_seed(size = n)
     allocate(seed(n))
     clock = time()
     !call system_clock(count = clock)
     seed = clock+37*(/ (i−1, i = 1,n) /)
     call random_seed(put = seed)
     deallocate(seed)
   end subroutine init_random_seed
end program hopalong
```

```
!
! Fortran Interface to the Xbgi-364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! This is the 'bgi' module.
!

module bgi
  use, intrinsic :: iso_c_binding, only: c_associated, C_BOOL, C_CHAR, &
       C_FUNPTR, c_f_pointer, C_INT, c_loc, C_NULL_CHAR, C_PTR, &
       C_SIGNED_CHAR, C_SIZE_T
  implicit none
  private

  ! =================
  !   BGI CONSTANTS
  ! =================

  integer, parameter, public :: MAXBUF = 256

  ! BGI colors
  integer(C_INT), parameter, public :: BLACK = 0
  integer(C_INT), parameter, public :: BLUE = 1
  integer(C_INT), parameter, public :: GREEN = 2
  integer(C_INT), parameter, public :: CYAN = 3
  integer(C_INT), parameter, public :: RED = 4
  integer(C_INT), parameter, public :: MAGENTA = 5
  integer(C_INT), parameter, public :: BROWN = 6
  integer(C_INT), parameter, public :: LIGHTGRAY = 7
  integer(C_INT), parameter, public :: DARKGRAY = 8
  integer(C_INT), parameter, public :: LIGHTBLUE = 9
  integer(C_INT), parameter, public :: LIGHTGREEN = 10
  integer(C_INT), parameter, public :: LIGHTCYAN = 11
  integer(C_INT), parameter, public :: LIGHTRED = 12
  integer(C_INT), parameter, public :: LIGHTMAGENTA = 13
  integer(C_INT), parameter, public :: YELLOW = 14
  integer(C_INT), parameter, public :: WHITE = 15

  integer(C_INT), parameter, public :: CGA_LIGHTGREEN = 1
  integer(C_INT), parameter, public :: CGA_LIGHTRED = 2
  integer(C_INT), parameter, public :: CGA_YELLOW = 3

  integer(C_INT), parameter, public :: CGA_LIGHTCYAN = 1
  integer(C_INT), parameter, public :: CGA_LIGHTMAGENTA = 2
  integer(C_INT), parameter, public :: CGA_WHITE = 3

  integer(C_INT), parameter, public :: CGA_GREEN = 1
  integer(C_INT), parameter, public :: CGA_RED = 2
  integer(C_INT), parameter, public :: CGA_BROWN = 3

  integer(C_INT), parameter, public :: CGA_CYAN = 1
  integer(C_INT), parameter, public :: CGA_MAGENTA = 2
  integer(C_INT), parameter, public :: CGA_LIGHTGRAY = 3

  integer(C_INT), parameter, public :: EGA_BLACK = 0
  integer(C_INT), parameter, public :: EGA_BLUE = 1
  integer(C_INT), parameter, public :: EGA_GREEN = 2
  integer(C_INT), parameter, public :: EGA_CYAN = 3
  integer(C_INT), parameter, public :: EGA_RED = 4
  integer(C_INT), parameter, public :: EGA_MAGENTA = 5
  integer(C_INT), parameter, public :: EGA_LIGHTGRAY = 7
  integer(C_INT), parameter, public :: EGA_BROWN = 20
  integer(C_INT), parameter, public :: EGA_DARKGRAY = 56
  integer(C_INT), parameter, public :: EGA_LIGHTBLUE = 57
  integer(C_INT), parameter, public :: EGA_LIGHTGREEN = 58
  integer(C_INT), parameter, public :: EGA_LIGHTCYAN = 59
  integer(C_INT), parameter, public :: EGA_LIGHTRED = 60
  integer(C_INT), parameter, public :: EGA_LIGHTMAGENTA = 61
  integer(C_INT), parameter, public :: EGA_YELLOW = 62
  integer(C_INT), parameter, public :: EGA_WHITE = 63

  integer(C_INT), parameter, public :: MAXCOLORS = 15
  integer(C_INT), parameter, public :: MAXRGBCOLORS = 4096
```

```
integer(C_INT), parameter, public :: EMPTY_FILL = 0
integer(C_INT), parameter, public :: SOLID_FILL = 1
integer(C_INT), parameter, public :: LINE_FILL = 2
integer(C_INT), parameter, public :: LTSLASH_FILL = 3
integer(C_INT), parameter, public :: SLASH_FILL = 4
integer(C_INT), parameter, public :: BKSLASH_FILL = 5
integer(C_INT), parameter, public :: LTBKSLASH_FILL = 6
integer(C_INT), parameter, public :: HATCH_FILL = 7
integer(C_INT), parameter, public :: XHATCH_FILL = 8
integer(C_INT), parameter, public :: INTERLEAVE_FILL = 9
integer(C_INT), parameter, public :: WIDE_DOT_FILL = 10
integer(C_INT), parameter, public :: CLOSE_DOT_FILL = 11
integer(C_INT), parameter, public :: USER_FILL = 12

! Line styles
integer(C_INT), parameter, public :: SOLID_LINE = 0
integer(C_INT), parameter, public :: DOTTED_LINE = 1
integer(C_INT), parameter, public :: CENTER_LINE = 2
integer(C_INT), parameter, public :: DASHED_LINE = 3
integer(C_INT), parameter, public :: USERBIT_LINE = 4

integer(C_INT), parameter, public :: NORM_WIDTH = 1
integer(C_INT), parameter, public :: THICK_WIDTH = 3

integer(C_INT), parameter, public :: DOTTEDLINE_LENGTH = 2
integer(C_INT), parameter, public :: CENTRELINE_LENGTH = 4
integer(C_INT), parameter, public :: DASHEDLINE_LENGTH = 2
integer(C_INT), parameter, public :: USERBITLINE_LENGTH = 16 ! GG

! Fonts
integer(C_INT), parameter, public :: DEFAULT_FONT = 0
integer(C_INT), parameter, public :: TRIPLEX_FONT = 1
integer(C_INT), parameter, public :: SMALL_FONT = 2
integer(C_INT), parameter, public :: SANSSERIF_FONT = 3
integer(C_INT), parameter, public :: GOTHIC_FONT = 4
integer(C_INT), parameter, public :: BIG_FONT = 5
integer(C_INT), parameter, public :: SCRIPT_FONT = 6
integer(C_INT), parameter, public :: SIMPLEX_FONT = 7
integer(C_INT), parameter, public :: TRIPLEX_SCR_FONT = 8
integer(C_INT), parameter, public :: COMPLEX_FONT = 9
integer(C_INT), parameter, public :: EUROPEAN_FONT = 10
integer(C_INT), parameter, public :: BOLD_FONT = 11

! Direction constants
integer(C_INT), parameter, public :: HORIZ_DIR = 0
integer(C_INT), parameter, public :: VERT_DIR = 1

! Justifications
integer(C_INT), parameter, public :: LEFT_TEXT = 0
integer(C_INT), parameter, public :: CENTER_TEXT = 1
integer(C_INT), parameter, public :: RIGHT_TEXT = 2
integer(C_INT), parameter, public :: BOTTOM_TEXT = 0
integer(C_INT), parameter, public :: TOP_TEXT = 2

! Writing modes
integer(C_INT), parameter, public :: COPY_PUT = 0
integer(C_INT), parameter, public :: XOR_PUT = 1
integer(C_INT), parameter, public :: OR_PUT = 2
integer(C_INT), parameter, public :: AND_PUT = 3
integer(C_INT), parameter, public :: NOT_PUT = 4

! Pages
integer(C_INT), parameter, public :: MAX_PAGES = 4

! Graphics errors
integer(C_INT), parameter, public :: grOk = 0
integer(C_INT), parameter, public :: grNoInitGraph = -1
integer(C_INT), parameter, public :: grNotDetected = -2
integer(C_INT), parameter, public :: grFileNotFound = -3
integer(C_INT), parameter, public :: grInvalidDriver = -4
integer(C_INT), parameter, public :: grNoLoadMem = -5
integer(C_INT), parameter, public :: grNoScanMem = -6
integer(C_INT), parameter, public :: grNoFloodMem = -7
integer(C_INT), parameter, public :: grFontNotFound = -8
integer(C_INT), parameter, public :: grNoFontMem = -9
integer(C_INT), parameter, public :: grInvalidMode = -10
integer(C_INT), parameter, public :: grError = -11
```

```
integer(C_INT), parameter, public :: grIOerror = −12
integer(C_INT), parameter, public :: grInvalidFont = −13
integer(C_INT), parameter, public :: grInvalidFontNum = −14
integer(C_INT), parameter, public :: grInvalidDeviceNum = −15
integer(C_INT), parameter, public :: grInvalidVersion = −18

! Graphics drivers constants, includes X11 which is particular to XBGI.
integer(C_INT), parameter, public :: DETECT = 0
integer(C_INT), parameter, public :: CGA = 1
integer(C_INT), parameter, public :: MCGA = 2
integer(C_INT), parameter, public :: EGA = 3
integer(C_INT), parameter, public :: EGA64 = 4
integer(C_INT), parameter, public :: EGAMONO = 5
integer(C_INT), parameter, public :: IBM8514 = 6
integer(C_INT), parameter, public :: HERCMONO = 7
integer(C_INT), parameter, public :: ATT400 = 8
integer(C_INT), parameter, public :: VGA = 9
integer(C_INT), parameter, public :: PC3270 = 10
integer(C_INT), parameter, public :: X11 = 11

! Graphics modes constants.
integer(C_INT), parameter, public :: CGAC0 = 0
integer(C_INT), parameter, public :: CGAC1 = 1
integer(C_INT), parameter, public :: CGAC2 = 2
integer(C_INT), parameter, public :: CGAC3 = 3
integer(C_INT), parameter, public :: CGAHI = 4

integer(C_INT), parameter, public :: MCGAC0 = 0
integer(C_INT), parameter, public :: MCGAC1 = 1
integer(C_INT), parameter, public :: MCGAC2 = 2
integer(C_INT), parameter, public :: MCGAC3 = 3
integer(C_INT), parameter, public :: MCGAMED = 4
integer(C_INT), parameter, public :: MCGAHI = 5

integer(C_INT), parameter, public :: EGALO = 0
integer(C_INT), parameter, public :: EGAHI = 1

integer(C_INT), parameter, public :: EGA64LO = 0
integer(C_INT), parameter, public :: EGA64HI = 1

integer(C_INT), parameter, public :: EGAMONOHI= 3

integer(C_INT), parameter, public :: HERCMONOHI = 0

integer(C_INT), parameter, public :: ATT400C0 = 0
integer(C_INT), parameter, public :: ATT400C1 = 1
integer(C_INT), parameter, public :: ATT400C2 = 2
integer(C_INT), parameter, public :: ATT400C3 = 3
integer(C_INT), parameter, public :: ATT400MED = 4
integer(C_INT), parameter, public :: ATT400HI = 5

integer(C_INT), parameter, public :: VGALO = 0
integer(C_INT), parameter, public :: VGAMED = 1
integer(C_INT), parameter, public :: VGAHI = 2

integer(C_INT), parameter, public :: PC3270HI = 0

integer(C_INT), parameter, public :: IBM8514LO = 0
integer(C_INT), parameter, public :: IBM8514HI = 1

integer(C_INT), parameter, public :: X11_CGALO = 0
integer(C_INT), parameter, public :: X11_CGAHI = 1
integer(C_INT), parameter, public :: X11_EGA = 2
integer(C_INT), parameter, public :: X11_VGA = 3
integer(C_INT), parameter, public :: X11_640x480 = 3
integer(C_INT), parameter, public :: X11_HERC = 4
integer(C_INT), parameter, public :: X11_PC3270 = 5
integer(C_INT), parameter, public :: X11_SVGALO = 6
integer(C_INT), parameter, public :: X11_800x600 = 6
integer(C_INT), parameter, public :: X11_SVGAMED1 = 7
integer(C_INT), parameter, public :: X11_1024x768 = 7
integer(C_INT), parameter, public :: X11_SVGAMED2 = 8
integer(C_INT), parameter, public :: X11_1152x900 = 8
integer(C_INT), parameter, public :: X11_SVGAHI = 9
integer(C_INT), parameter, public :: X11_1280x1024 = 9
integer(C_INT), parameter, public :: X11_WXGA = 10
integer(C_INT), parameter, public :: X11_1366x768 = 10
integer(C_INT), parameter, public :: X11_USER = 11
```

```
integer(C_INT), parameter, public :: X11_FULLSCREEN = 12

! Key codes
integer(C_INT), parameter, public :: KEY_HOME = 80
integer(C_INT), parameter, public :: KEY_LEFT = 81
integer(C_INT), parameter, public :: KEY_UP = 82
integer(C_INT), parameter, public :: KEY_RIGHT = 83
integer(C_INT), parameter, public :: KEY_DOWN = 84
integer(C_INT), parameter, public :: KEY_PGUP = 85
integer(C_INT), parameter, public :: KEY_PGDN = 86
integer(C_INT), parameter, public :: KEY_END = 87
integer(C_INT), parameter, public :: KEY_INSERT = 99
integer(C_INT), parameter, public :: KEY_DELETE = −1
integer(C_INT), parameter, public :: KEY_F1 = −66
integer(C_INT), parameter, public :: KEY_F2 = −65
integer(C_INT), parameter, public :: KEY_F3 = −64
integer(C_INT), parameter, public :: KEY_F4 = −63
integer(C_INT), parameter, public :: KEY_F5 = −62
integer(C_INT), parameter, public :: KEY_F6 = −61
integer(C_INT), parameter, public :: KEY_F7 = −60
integer(C_INT), parameter, public :: KEY_F8 = −59
integer(C_INT), parameter, public :: KEY_F9 = −58
integer(C_INT), parameter, public :: KEY_F10 = −57
integer(C_INT), parameter, public :: KEY_F11 = −56
integer(C_INT), parameter, public :: KEY_F12 = −55
integer(C_INT), parameter, public :: KEY_LEFT_CTRL = −29
integer(C_INT), parameter, public :: KEY_RIGHT_CTRL = −28
integer(C_INT), parameter, public :: KEY_LEFT_SHIFT = −31
integer(C_INT), parameter, public :: KEY_RIGHT_SHIFT = −30
integer(C_INT), parameter, public :: KEY_LEFT_ALT = −23
integer(C_INT), parameter, public :: KEY_LEFT_WIN = −21
integer(C_INT), parameter, public :: KEY_RIGHT_WIN = −20
integer(C_INT), parameter, public :: KEY_ALT_GR = 3
integer(C_INT), parameter, public :: KEY_TAB = 8
integer(C_INT), parameter, public :: KEY_BS = 9
integer(C_INT), parameter, public :: KEY_RET = 13
integer(C_INT), parameter, public :: KEY_PAUSE = 19
integer(C_INT), parameter, public :: KEY_SCR_LOCK = 20
integer(C_INT), parameter, public :: KEY_ESC = 27

! Mouse constants
integer(C_INT), parameter, public :: WM_LBUTTONDOWN = 1 ! left button
integer(C_INT), parameter, public :: WM_MBUTTONDOWN = 2 ! middle button
integer(C_INT), parameter, public :: WM_RBUTTONDOWN = 3 ! right button
integer(C_INT), parameter, public :: WM_WHEELUP = 4     ! wheel up
integer(C_INT), parameter, public :: WM_WHEELDOWN = 5   ! wheel down
integer(C_INT), parameter, public :: WM_MOUSEMOVE = 6   ! motion

! =============
!   BGI TYPES
! =============

! This type records information about the last call to arc. It is used
! by getarccoords to get the location of the endpoints of the arc
type, bind(c) :: arccoordstype
   integer(C_INT) :: x, y           ! Center point of the arc
   integer(C_INT) :: xstart, ystart  ! The starting position of the arc
   integer(C_INT) :: xend, yend      ! The ending position of the arc
end type arccoordstype

! This type defines the fill style for the current window.  Pattern is
! one of the system patterns such as SOLID_FILL.  Color is the color to
! fill with
type, bind(c) :: fillsettingstype
   integer(C_INT) :: pattern  ! Current fill pattern
   integer(C_INT) :: color    ! Current fill color
end type fillsettingstype

! This type records information about the current line style.
! linestyle is one of the line styles such as SOLID_LINE, upattern is a
! 16−bit pattern for user defined lines, and thickness is the width of the
! line in pixels
type, bind(c) :: linesettingstype
   integer(C_INT) :: linestyle  ! Current line style
   integer(C_INT) :: upattern   ! 16−bit user line pattern (unsigned!)
   integer(C_INT) :: thickness  ! Width of the line in pixels
end type linesettingstype
```

```
! This type records information about the text settings
type, bind(c) :: textsettingstype
   integer(C_INT) :: font       ! The font in use
   integer(C_INT) :: direction  ! Text direction
   integer(C_INT) :: charsize   ! Character size
   integer(C_INT) :: horiz      ! Horizontal text justification
   integer(C_INT) :: vert       ! Vertical text justification
end type textsettingstype

! This type records information about the viewport
type, bind(c) :: viewporttype
   ! Viewport bounding box
   integer(C_INT) :: left, top, right, bottom

   ! Whether to clip image to viewport
   integer(C_INT) :: clip
end type viewporttype

! This type records information about the palette
type, bind(c) :: palettetype
   integer(C_SIGNED_CHAR) :: size
   integer(C_SIGNED_CHAR) :: colors(0:MAXCOLORS)
end type palettetype

! This type records information about the (bitmap)image
type :: imagetype
   ! Pointer to the data
   type(C_PTR) :: image_ptr

   integer :: width
   integer :: height
end type imagetype

! =================
!   BGI INTERFACE
! =================

interface

   ! Drawing routines...
   subroutine arc(x,y,stangle,endangle,radius) bind(c)
     import :: C_INT
     integer(C_INT), value :: x, y, stangle, endangle, radius
   end subroutine arc

   subroutine bar(left,top,right,bottom) bind(c)
     import :: C_INT
     integer(C_INT), value :: left, top, right, bottom
   end subroutine bar

   subroutine bar3d(left,top,right,bottom,depth,topflag) bind(c)
     import :: C_INT
     integer(C_INT), value :: left, top, right, bottom, depth, topflag
   end subroutine bar3d

   subroutine circle(x,y,radius) bind(c)
     import :: C_INT
     integer(C_INT), value :: x, y, radius
   end subroutine circle

   subroutine cleardevice() bind(c)
   end subroutine cleardevice

   subroutine clearviewport() bind(c)
   end subroutine clearviewport

   subroutine c_drawpoly(numpoints,polypoints) bind(c, name='drawpoly')
     import :: C_INT
     integer(C_INT), value :: numpoints

     ! polypoints should be an array of 2*numpoints elements
     integer(C_INT), intent(in) :: polypoints(*)
   end subroutine c_drawpoly

   subroutine ellipse(x,y,stangle,endangle,xradius,yradius) bind(c)
     import :: C_INT
     integer(C_INT), value :: x, y, stangle, endangle, xradius, yradius
   end subroutine ellipse
```

```
subroutine fillellipse(x,y,xradius,yradius) bind(c)
  import :: C_INT
  integer(C_INT), value :: x, y, xradius, yradius
end subroutine fillellipse

subroutine c_fillpoly(numpoints,polypoints) bind(c, name='fillpoly')
  import :: C_INT
  integer(C_INT), value :: numpoints

  ! polypoints should be an array of 2*numpoints elements
  integer(C_INT), intent(in) :: polypoints(*)
end subroutine c_fillpoly

subroutine floodfill(x,y,border) bind(c)
  import :: C_INT
  integer(C_INT), value :: x, y, border
end subroutine floodfill

subroutine line(x1,y1,x2,y2) bind(c)
  import :: C_INT
  integer(C_INT), value :: x1, y1, x2, y2
end subroutine line

subroutine linerel(dx,dy) bind(c)
  import :: C_INT
  integer(C_INT), value :: dx, dy
end subroutine linerel

subroutine lineto(x,y) bind(c)
  import :: C_INT
  integer(C_INT), value :: x, y
end subroutine lineto

subroutine pieslice(x,y,stangle,endangle,radius) bind(c)
  import :: C_INT
  integer(C_INT), value :: x, y, stangle, endangle, radius
end subroutine pieslice

subroutine fast_putpixel(x,y) bind(c, name='_putpixel')
  import :: C_INT
  integer(C_INT), value :: x, y
end subroutine fast_putpixel

subroutine putpixel(x,y,color) bind(c)
  import :: C_INT
  integer(C_INT), value :: x, y, color
end subroutine putpixel

subroutine rectangle(left,top,right,bottom) bind(c)
  import :: C_INT
  integer(C_INT), value :: left, top, right, bottom
end subroutine rectangle

subroutine sector(x,y,stangle,endangle,xradius,yradius) bind(c)
  import :: C_INT
  integer(C_INT), value :: x, y, stangle, endangle, xradius, yradius
end subroutine sector

! Miscellaneous routines..
function converttorgb(v) bind(c)
  import :: C_INT
  integer(C_INT) :: converttorgb
  integer(C_INT), value :: v
end function converttorgb

subroutine delay(millisec) bind(c)
  import :: C_INT
  integer(C_INT), value :: millisec
end subroutine delay

function event() bind(c)
  import :: C_INT
  integer(C_INT) :: event
end function event

subroutine getarccoords(arccoords) bind(c)
  import :: arccoordstype
```

```
  type(arccoordstype), intent(out) :: arccoords
end subroutine getarccoords

function getbkcolor() bind(c)
  import :: C_INT
  integer(C_INT) :: getbkcolor
end function getbkcolor

function getcolor() bind(c)
  import :: C_INT
  integer(C_INT) :: getcolor
end function getcolor

function getevent() bind(c)
  import :: C_INT
  integer(C_INT) :: getevent
end function getevent

subroutine c_getfillpattern(cstr) bind(c, name='getfillpattern')
  import :: C_PTR
  type(C_PTR), value :: cstr
end subroutine c_getfillpattern

subroutine getfillsettings(fillinfo) bind(c)
  import :: fillsettingstype
  type(fillsettingstype), intent(out) :: fillinfo
end subroutine getfillsettings

subroutine getlinesettings(lineinfo) bind(c)
  import :: linesettingstype
  type(linesettingstype), intent(out) :: lineinfo
end subroutine getlinesettings

function getmaxcolor() bind(c)
  import :: C_INT
  integer(C_INT) :: getmaxcolor
end function getmaxcolor

function getmaxheight() bind(c)
  import :: C_INT
  integer(C_INT) :: getmaxheight
end function getmaxheight

function getmaxwidth() bind(c)
  import :: C_INT
  integer(C_INT) :: getmaxwidth
end function getmaxwidth

function getmaxx() bind(c)
  import :: C_INT
  integer(C_INT) :: getmaxx
end function getmaxx

function getmaxy() bind(c)
  import :: C_INT
  integer(C_INT) :: getmaxy
end function getmaxy

function getpixel(x,y) bind(c)
  import :: C_INT
  integer(C_INT), value :: x, y
  integer(C_INT) :: getpixel
end function getpixel

subroutine getviewsettings(viewport) bind(c)
  import :: viewporttype
  type(viewporttype), intent(out) :: viewport
end subroutine getviewsettings

function getwindowheight() bind(c)
  import :: C_INT
  integer(C_INT) :: getwindowheight
end function getwindowheight

function getwindowwidth() bind(c)
  import :: C_INT
  integer(C_INT) :: getwindowwidth
end function getwindowwidth
```

```
function getx() bind(c)
  import :: C_INT
  integer(C_INT) :: getx
end function getx

function gety() bind(c)
  import :: C_INT
  integer(C_INT) :: gety
end function gety

subroutine moverel(dx,dy) bind(c)
  import :: C_INT
  integer(C_INT), value :: dx, dy
end subroutine moverel

subroutine moveto(x,y) bind(c)
  import :: C_INT
  integer(C_INT), value :: x, y
end subroutine moveto

subroutine refresh() bind(c)
end subroutine refresh

subroutine setbkcolor(color) bind(c)
  import :: C_INT
  integer(C_INT), value :: color
end subroutine setbkcolor

subroutine setbkrgbcolor(index) bind(c)
  import :: C_INT
  integer(C_INT), value :: index
end subroutine setbkrgbcolor

subroutine setcolor(color) bind(c)
  import :: C_INT
  integer(C_INT), value :: color
end subroutine setcolor

subroutine setrgbcolor(index) bind(c)
  import :: C_INT
  integer(C_INT), value :: index
end subroutine setrgbcolor

subroutine setfillpattern(upattern,color) bind(c)
  import :: C_INT, C_CHAR
  character(C_CHAR), intent(in) :: upattern(*)
  integer(C_INT), value :: color
end subroutine setfillpattern

subroutine setfillstyle(pattern,color) bind(c)
  import :: C_INT
  integer(C_INT), value :: pattern, color
end subroutine setfillstyle

!void setlinestyle(int linestyle, unsigned upattern, int thickness);
subroutine setlinestyle(linestyle,upattern,thickness) bind(c)
  import :: C_INT
  integer(C_INT), value :: linestyle, upattern, thickness
end subroutine setlinestyle

subroutine setviewport(left,top,right,bottom,clip) bind(c)
  import :: C_INT
  integer(C_INT), value :: left, top, right, bottom, clip
end subroutine setviewport

subroutine setwritemode(mode) bind(c)
  import :: C_INT
  integer(C_INT), value :: mode
end subroutine setwritemode

function usleep(useconds) bind(c)
  import :: C_INT
  integer(C_INT), value :: useconds
  integer(C_INT)  :: usleep
end function usleep

! Window Creation / Graphics Manipulation routines...
```

```
subroutine closebgi() bind(c)
end subroutine closebgi

subroutine closegraph() bind(c)
end subroutine closegraph

subroutine detectgraph(graphdriver,graphmode) bind(c)
  import :: C_INT
  integer(C_INT), intent(out) :: graphdriver, graphmode
end subroutine detectgraph

subroutine getaspectratio(xasp,yasp) bind(c)
  import :: C_INT
  integer(C_INT), intent(out) :: xasp, yasp
end subroutine getaspectratio

function c_getdrivername() bind(c, name='getdrivername')
  import :: C_PTR
  type(C_PTR) :: c_getdrivername
end function c_getdrivername

function getgraphmode() bind(c)
  import :: C_INT
  integer(C_INT) :: getgraphmode
end function getgraphmode

function getmaxmode() bind(c)
  import :: C_INT
  integer(C_INT) :: getmaxmode
end function getmaxmode

function c_getmodename(mode_number) bind(c, name='getmodename')
  import :: C_INT, C_PTR
  type(C_PTR) :: c_getmodename
  integer(C_INT), value :: mode_number
end function c_getmodename

subroutine getmoderange(graphdriver,lomode,himode) bind(c)
  import :: C_INT
  integer(C_INT), value :: graphdriver
  integer(C_INT), intent(out) :: lomode, himode
end subroutine getmoderange

subroutine graphdefaults() bind(c)
end subroutine graphdefaults

function c_grapherrormsg(error_code) bind(c, name='grapherrormsg')
  import :: C_INT, C_PTR
  type(C_PTR) :: c_grapherrormsg
  integer(C_INT), value :: error_code
end function c_grapherrormsg

function graphresult() bind(c)
  import :: C_INT
  integer(C_INT) :: graphresult
end function graphresult

! Being libXbgi written in ANSI C, we cannot have default (optional)
! parameters
subroutine initgraph(graphdriver,graphmode,pathtodriver) bind(c)
  import :: C_INT, C_CHAR
  integer(C_INT), intent(inout) :: graphdriver, graphmode
  character(C_CHAR), intent(in) :: pathtodriver(*)
end subroutine initgraph

subroutine initwindow(width,height) bind(c)
  import :: C_INT
  integer(C_INT), value :: width, height
end subroutine initwindow

subroutine openbgi(width,height,title) bind(c)
  import:: C_INT, C_CHAR
  integer(C_INT), value :: width, height
  character(C_CHAR), intent(in) :: title(*)
end subroutine openbgi

! Not available in Xbgi
function installuserdriver(name,detect) bind(c)
```

```
  import :: C_INT, C_CHAR, C_FUNPTR
  character(C_CHAR), intent(in) :: name(*)
  type(C_FUNPTR), value :: detect
  integer(C_INT) :: installuserdriver
end function installuserdriver

! Not implemented in Xbgi
function installuserfont(name)  bind(c)
  import :: C_INT, C_CHAR
  character(C_CHAR), intent(in) :: name(*)
  integer(C_INT) :: installuserfont
end function installuserfont

! Not implemented in Xbgi
function registerbgidriver(driver) bind(c)
  import :: C_PTR, C_INT
  type(C_PTR), value :: driver
  integer(C_INT) :: registerbgidriver
end function registerbgidriver

! Not implemented in Xbgi
function registerbgifont(font) bind(c)
  import :: C_PTR, C_INT
  type(C_PTR), value :: font
  integer(C_INT) :: registerbgifont
end function registerbgifont

! This routine only clears the device in Xbgi, so you should not use it
subroutine restorecrtmode() bind(c)
end subroutine restorecrtmode

subroutine setaspectratio(xasp,yasp) bind(c)
  import :: C_INT
  integer(C_INT), value :: xasp, yasp
end subroutine setaspectratio

! It uses "unsigned"
function setgraphbufsize(bufsize) bind(c)
  import :: C_INT
  integer(C_INT), value :: bufsize
  integer(C_INT) :: setgraphbufsize
end function setgraphbufsize

subroutine setgraphmode(mode) bind(c)
  import :: C_INT
  integer(C_INT), value :: mode
end subroutine setgraphmode

! User interation routines...
function getch() bind(c)
  import :: C_INT
  integer(C_INT) :: getch
end function getch

function kbhit() bind(c)
  import :: C_INT
  integer(C_INT) :: kbhit
end function kbhit

function xkbhit() bind(c)
  import :: C_INT
  integer(C_INT) :: xkbhit
end function xkbhit

! Double buffering support routines...
function getactivepage() bind(c)
  import :: C_INT
  integer(C_INT) :: getactivepage
end function getactivepage

function getvisualpage() bind(c)
  import :: C_INT
  integer(C_INT) :: getvisualpage
end function getvisualpage

subroutine setactivepage(page) bind(c)
  import :: C_INT
  integer(C_INT), value :: page
```

```
end subroutine setactivepage

subroutine setvisualpage(page) bind(c)
  import :: C_INT
  integer(C_INT), value :: page
end subroutine setvisualpage

! Image routines...
function imagesize(left,top,right,bottom) bind(c)
  import :: C_INT
  integer(C_INT) :: imagesize
  integer(C_INT), value :: left, top, right, bottom
end function imagesize

subroutine getimage(left,top,right,bottom,bitmap) bind(c)
  import :: C_INT, C_PTR
  integer(C_INT), value :: left, top, right, bottom
  type(C_PTR), value :: bitmap
end subroutine getimage

subroutine putimage(left,top,ptr,op) bind(c)
  import :: C_INT, C_PTR
  integer(C_INT), value :: left, top
  type(C_PTR), value :: ptr
  integer(C_INT), value :: op
end subroutine putimage

! Text routines...
subroutine gettextsettings(texttypeinfo) bind(c)
  import :: textsettingstype
  type(textsettingstype), intent(out) :: texttypeinfo
end subroutine gettextsettings

subroutine outtext(textstring) bind(c)
  import :: C_CHAR
  character(C_CHAR), intent(in) :: textstring(*)
end subroutine outtext

subroutine outtextxy(x,y,textstring) bind(c)
  import :: C_INT, C_CHAR
  integer(C_INT), value :: x, y
  character(C_CHAR), intent(in) :: textstring(*)
end subroutine outtextxy

subroutine settextjustify(horiz,vert) bind(c)
  import :: C_INT
  integer(C_INT), value :: horiz, vert
end subroutine settextjustify

subroutine settextstyle(font,direction,charsize) bind(c)
  import :: C_INT
  integer(C_INT), value :: font, direction, charsize
end subroutine settextstyle

subroutine setusercharsize(multx,divx,multy,divy) bind(c)
  import :: C_INT
  integer(C_INT), value :: multx, divx, multy, divy
end subroutine setusercharsize

function textheight(textstring) bind(c)
  import :: C_INT, C_CHAR
  character(C_CHAR), intent(in) :: textstring(*)
  integer(C_INT) :: textheight
end function textheight

function textwidth(textstring) bind(c)
  import :: C_INT, C_CHAR
  character(C_CHAR), intent(in) :: textstring(*)
  integer(C_INT) :: textwidth
end function textwidth

! Mouse routines...
subroutine clearmouseclick(kind) bind(c)
  import :: C_INT
  integer(C_INT), value :: kind
end subroutine clearmouseclick

subroutine getmouseclick(kind,x,y) bind(c)
```

```
  import :: C_INT
  integer(C_INT), value :: kind
  integer(C_INT), intent(out) :: x, y
end subroutine getmouseclick

function ismouseclick(kind) bind(c)
  import :: C_INT, C_BOOL
  integer(C_INT), value :: kind
  logical(C_BOOL) :: ismouseclick
end function ismouseclick

function mousex() bind(c)
  import :: C_INT
  integer(C_INT) :: mousex
end function mousex

function mousey() bind(c)
  import :: C_INT
  integer(C_INT) :: mousey
end function mousey

! Palette routines...
function c_getdefaultpalette() bind(c, name='getdefaultpalette')
  import :: C_PTR
  type(C_PTR) :: c_getdefaultpalette
end function c_getdefaultpalette

subroutine getpalette(palette) bind(c)
  import :: palettetype
  type(palettetype), intent(out) :: palette
end subroutine getpalette

function getpalettesize() bind(c)
  import :: C_INT
  integer(C_INT) :: getpalettesize
end function getpalettesize

subroutine setallpalette(palette) bind(c)
  import :: palettetype
  type(palettetype), intent(in) :: palette
end subroutine setallpalette

subroutine setpalette(colornum,color) bind(c)
  import :: C_INT
  integer(C_INT), value :: colornum, color
end subroutine setpalette

subroutine setrgbpalette(colornum,red,green,blue) bind(c)
  import :: C_INT
  integer(C_INT), value :: colornum, red, green, blue
end subroutine setrgbpalette

! RGB COLOR routines...
function rgb_color(r,g,b) bind(c, name='COLOR')
  import :: C_INT
  integer(C_INT) :: rgb_color
  integer(C_INT), value :: r, g, b
end function rgb_color

! C routines...
!  'strlen' from Tobias Burnus,
!  http://gcc.gnu.org/ml/fortran/2010-02/msg00029.html
function c_strlen(str) bind(c, name='strlen')
  import :: C_PTR, C_SIZE_T
  type (C_PTR), value :: str
  integer(C_SIZE_T)  :: c_strlen
end function c_strlen

function c_malloc(memsize) bind(c, name='malloc')
  import :: C_PTR, C_INT
  integer(C_INT), value :: memsize
  type(C_PTR) :: c_malloc
end function c_malloc

subroutine c_free(p) bind(c, name='free')
  import :: C_PTR
  type(C_PTR), value :: p
end subroutine c_free
```

```
  end interface

  ! BGI types...
  public :: arccoordstype, fillsettingstype, linesettingstype, &
       textsettingstype, viewporttype, palettetype, imagetype

  ! Drawing routines...
  public :: arc, bar, bar3d, circle, cleardevice, clearviewport, drawpoly, &
       ellipse, fillellipse, fillpoly, floodfill, line, linerel, lineto, &
       pieslice, fast_putpixel, putpixel, rectangle, sector

  ! Miscellaneous routines..
  public :: converttorgb, delay, event, getarccoords, getbkcolor, getcolor, &
       getevent, getfillpattern, getfillsettings, getlinesettings, &
       getmaxcolor, getmaxheight, getmaxwidth, getmaxx, getmaxy, getpixel, &
       getviewsettings, getwindowheight, getwindowwidth, getx, gety, moverel, &
       moveto, refresh, setbkcolor, setbkrgbcolor, setcolor, setrgbcolor, &
       setfillpattern, setfillstyle, setlinestyle, setviewport, setwritemode, &
       usleep

  ! Window Creation / Graphics Manipulation routines...
  public :: closebgi, closegraph, detectgraph, getaspectratio, getdrivername, &
       getgraphmode, getmaxmode, getmodename, getmoderange, graphdefaults, &
       grapherrormsg, graphresult, initgraph, initwindow, openbgi, &
       installuserdriver, installuserfont, registerbgidriver, registerbgifont, &
       restorecrtmode, setaspectratio, setgraphbufsize, setgraphmode

  ! User interation routines...
  public :: getch, kbhit, xkbhit

  ! Double buffering support routines...
  public :: getactivepage, getvisualpage, setactivepage, setvisualpage, &
       swapbuffers

  ! Image routines...
  public :: imagesize, getimage, putimage, allocateimage, freeimage, &
       copyimage, pasteimage

  ! Text routines...
  public :: gettextsettings, outtext, outtextxy, settextjustify, settextstyle, &
       setusercharsize, textheight, textwidth

  ! Mouse routines...
  public :: clearmouseclick, getmouseclick, ismouseclick, mousex, mousey

  ! Palette routines...
  public :: getdefaultpalette, getpalette, getpalettesize, setallpalette, &
       setpalette, setrgbpalette

  ! RGB COLOR routines...
  public :: rgb_color, red_value, green_value, blue_value

  ! Utility routines...
  public :: CString, quit

contains

  subroutine c_f_stringconvert(cstring,str)
    type(c_ptr), intent(in) :: cstring
    character(len=*), intent(out) :: str
    character(C_CHAR), dimension(:), pointer :: farray
    integer :: i

    call c_f_pointer(cstring,farray,[c_strlen(cstring)])

    str = repeat(' ', len(str))
    do i = 1, min(len(str), int(c_strlen(cstring)))
       str(i:i) = farray(i)
    end do
  end subroutine c_f_stringconvert

  function getdefaultpalette() result(fptr)
    type(palettetype), pointer :: fptr
    type(C_PTR) :: cptr

    fptr => null()
```

```
  cptr = c_getdefaultpalette()
  call c_f_pointer(cptr,fptr)
end function getdefaultpalette

subroutine getdrivername(str)
  character(*), intent(out) :: str

  call c_f_stringconvert(c_getdrivername(),str)
end subroutine getdrivername

subroutine getmodename(mode_number,str)
  integer, intent(in) :: mode_number
  character(*), intent(out) :: str

  call c_f_stringconvert(c_getmodename(mode_number),str)
end subroutine getmodename

subroutine grapherrormsg(error_code,str)
  integer, intent(in) :: error_code
  character(*), intent(out) :: str

  call c_f_stringconvert(c_grapherrormsg(error_code),str)
end subroutine grapherrormsg

subroutine getfillpattern(pattern)
  character, intent(out), target :: pattern(8)

  call c_getfillpattern(c_loc(pattern))
end subroutine getfillpattern

subroutine drawpoly(numpoints,points)
  integer(C_INT), intent(in) :: numpoints
  integer, intent(in), dimension(numpoints,2) :: points
  integer(C_INT), dimension(numpoints*2), target :: oned

  oned = reshape(transpose(points), (/ 2*numpoints /))
  call c_drawpoly(numpoints,oned)
end subroutine drawpoly

subroutine fillpoly(numpoints,points)
  integer(C_INT), intent(in) :: numpoints
  integer, intent(in), dimension(numpoints,2) :: points
  integer(C_INT), dimension(numpoints*2), target :: oned

  oned = reshape(transpose(points), (/ 2*numpoints /))
  call c_fillpoly(numpoints,oned)
end subroutine fillpoly

subroutine allocateimage(img,width,height)
  type(imagetype), intent(out) :: img
  integer, intent(in) :: width, height
  integer :: memsize

  memsize = imagesize(0,0,width,height)

  img%width = width
  img%height = height
  img%image_ptr = c_malloc(memsize)

  if (.not. c_associated(img%image_ptr)) then
     if (graphresult() == grOk) call closegraph()
     write(*,*) 'ALLOCATEIMAGE: Allocation request denied'
     write(*,*) 'Error: not enough heap space.'
     stop
  end if
end subroutine allocateimage

subroutine freeimage(img)
  type(imagetype), intent(inout) :: img

  img%width = 0
  img%height = 0

  if (c_associated(img%image_ptr)) then
     call c_free(img%image_ptr)
  else
     if (graphresult() == grOk) call closegraph()
     write(*,*) 'FREEIMAGE: Deallocation request denied'
```

```
      write(*,*) 'Error: not associated pointer.'
      stop
    end if
  end subroutine freeimage

  subroutine copyimage(left,top,img)
    integer, intent(in) :: left, top
    type(imagetype), intent(inout) :: img

    call getimage(left,top,left+img%width,top+img%height,img%image_ptr)
  end subroutine copyimage

  subroutine pasteimage(left,top,img,op)
    integer, intent(in) :: left, top
    type(imagetype), intent(in) :: img
    integer, intent(in) :: op

    call putimage(left,top,img%image_ptr,op)
  end subroutine pasteimage

  function CString(string) result(array)
    character(len=*), intent(in) :: string
    character(kind=C_CHAR), dimension(len(string)+1) :: array
    integer :: i

    do i = 1, len(string)
       array(i) = string(i:i)
    end do
    array(len(string)+1) = C_NULL_CHAR
  end function CString

  function quit()
    logical :: quit
    character :: qchar

    quit = .false.
    if (kbhit() /= 0) then
       qchar = char(getch())
       quit = (qchar == 'Q' .or. qchar == 'q')
    end if
  end function quit

  function red_value(v)
    integer :: red_value
    integer, intent(in) :: v

    ! we need shift right
    red_value = (iand(ishft((v),-16),int(Z'FF')))
  end function red_value

  function green_value(v)
    integer :: green_value
    integer, intent(in) :: v

    ! we need to shift right
    green_value = (iand(ishft((v),-8),int(Z'FF')))
  end function green_value

  function blue_value(v)
    integer :: blue_value
    integer, intent(in) :: v

    blue_value = (iand((v),int(Z'FF')))
  end function blue_value

  subroutine swapbuffers()
    integer :: oldv, olda

    oldv = getvisualpage()
    olda = getactivepage()

    call setvisualpage(olda)
    call setactivepage(oldv)
  end subroutine swapbuffers
end module bgi
```

```
!
! Fortran Interface to the Xbgi−364p Library
! by Angelo Graziosi  (firstname.lastnameATalice.it)
! Copyright Angelo Graziosi
!
! It is distributed in the hope that it will be useful,
! but WITHOUT ANY WARRANTY; without even the implied warranty of
! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
!
! This is the 'bgiapp' module.
!

module bgiapp
  use kind_consts, only: DP
  use bgi, only: cleardevice, closebgi, CString, ellipse, getch, LIGHTGREEN, &
       line, fast_putpixel, putpixel, fillellipse, LEFT_TEXT, openbgi, &
       outtext, outtextxy, rectangle, rgb_color, setactivepage, setbkcolor, &
       setcolor, settextjustify, setvisualpage, TOP_TEXT, usleep, YELLOW
  implicit none
  private

  integer :: width = 600, height = 600
  real(DP) :: x_min = −1.0_DP, x_max = 1.0_DP, &
       y_min = −1.0_DP, y_max = 1.0_DP, &
       scx = 0.0_DP, scy = 0.0_DP

  interface bgiapp_text
     module procedure outtext1,outtext3
  end interface bgiapp_text

  public :: bgiapp_close, bgiapp_init, bgiapp_setup, &
       bgiapp_xmin, bgiapp_xmax, bgiapp_ymin, bgiapp_ymax, &
       bgiapp_width, bgiapp_height, bgiapp_box, bgiapp_circle, bgiapp_dot, &
       bgiapp_fast_dot, bgiapp_fillellipse, bgiapp_line, bgiapp_text

contains

  subroutine bgiapp_init(title)
    character(len = *), intent(in) :: title

    call openbgi(width,height,CString(title))

    call setbkcolor(rgb_color(0,0,40))
    call cleardevice()
  end subroutine bgiapp_init

  subroutine bgiapp_close()
    integer, parameter :: SECONDS = 1000000
    integer :: k

    call settextjustify(LEFT_TEXT,TOP_TEXT)
    call setcolor(YELLOW)
    call outtextxy(0,0,CString('Press a key to exit...'))
    k = getch()

    call setcolor(LIGHTGREEN);
    call outtextxy (0,20,CString('Ok, leaving in 5 seconds...'))

    k = usleep (5*SECONDS)

    call closebgi()
    print *, 'All done'
  end subroutine bgiapp_close

  subroutine bgiapp_setup(x1,x2,y1,y2,wh,ht)
    use get_data, only: get
    real(DP), intent(in), optional :: x1, x2, y1, y2
    integer, intent(in), optional :: wh, ht

    ! Initializing with defaults values...
    if (present(x1)) x_min = x1
    if (present(x2)) x_max = x2
    if (present(y1)) y_min = y1
    if (present(y2)) y_max = y2
    if (present(wh)) width = wh
    if (present(ht)) height = ht
    call get('WIDTH  (pixels) =',width)
    call get('HEIGHT (pixels) =',height)
```

```
  write(*,*)
  call get('XMIN =',x_min)
  call get('XMAX =',x_max)
  write(*,*)
  call get('YMIN =',y_min)
  call get('YMAX =',y_max)
  write(*,*)
  scx = (width/(x_max−x_min))   ! x scale
  scy = (height/(y_max−y_min))  ! y scale
end subroutine bgiapp_setup

function xs(x) result(ret)
  integer :: ret
  real(DP), intent(in) :: x
  ret = 0+nint((x−x_min)*scx)
end function xs

function ys(y) result(ret)
  integer :: ret
  real(DP), intent(in) :: y
  ret = 0+nint((y_max−y)*scy)
end function ys

subroutine outtext1(text)
  character(len=*), intent(in) :: text
  call outtext(CString(text))
end subroutine outtext1

subroutine outtext3(x,y,text)
  real(DP), intent(in) :: x, y
  character(len=*), intent(in) :: text
  call outtextxy(xs(x),ys(y),CString(text))
end subroutine outtext3

 function bgiapp_xmin() result(r)
  real(DP) :: r
  r = x_min
end function bgiapp_xmin

function bgiapp_xmax() result(r)
  real(DP) :: r
  r = x_max
end function bgiapp_xmax

function bgiapp_ymin() result(r)
  real(DP) :: r
  r = y_min
end function bgiapp_ymin

function bgiapp_ymax() result(r)
  real(DP) :: r
  r = y_max
end function bgiapp_ymax

function bgiapp_width() result(r)
  integer :: r
  r = width
end function bgiapp_width

function bgiapp_height() result(r)
  integer :: r
  r = height
end function bgiapp_height

subroutine bgiapp_box(x1,x2,y1,y2)
  real(DP), intent(in) :: x1, x2, y1, y2
  call rectangle(xs(x1),ys(y1),xs(x2),ys(y2))
end subroutine bgiapp_box

subroutine bgiapp_circle(x,y,r)
  real(DP), intent(in) :: x, y, r
  call ellipse(xs(x),ys(y),0,360, &
       abs(xs(r)−xs(0.0_DP)),abs(ys(r)−ys(0.0_DP)))
end subroutine bgiapp_circle

subroutine bgiapp_dot(x,y,color)
  real(DP), intent(in) :: x, y
  integer, intent(in) :: color
```

```
    call putpixel(xs(x),ys(y),color)
  end subroutine bgiapp_dot

  subroutine bgiapp_fast_dot(x,y)
    real(DP), intent(in) :: x, y
    call fast_putpixel(xs(x),ys(y))
  end subroutine bgiapp_fast_dot

  subroutine bgiapp_fillellipse(x,y,a,b)
    real(DP), intent(in) :: x, y, a, b
    call fillellipse(xs(x),ys(y), &
         abs(xs(a)−xs(0.0_DP)),abs(ys(b)−ys(0.0_DP)))
  end subroutine bgiapp_fillellipse

  subroutine bgiapp_line(x1,y1,x2,y2)
    real(DP), intent(in) :: x1, y1, x2, y2
    call line(xs(x1),ys(y1),xs(x2),ys(y2))
  end subroutine bgiapp_line
end module bgiapp
```