```
##+#########################################################################
#
# TkMandelbrot -- draws the mandelbrot fractal
# based on http://www.students.tut.fi/~warp/Mandelbrot/
# by Keith Vetter
#
# Revisions:
# KPV Nov 13, 2002 - initial revision
#
##+#########################################################################
###########################################################################

set tcl_precision 17

set Cwidth 500                                  ;# Canvas size
set Cheight 500

set Rmin -2.0                                   ;# Left side
set Rmax 1.0                                    ;# Right side
set Imin -1.5                                   ;# Bottom
set Imax [expr {$Imin + ($Rmax - $Rmin) * $Cheight /$Cwidth}]

set Rscale [expr {($Rmax - $Rmin) / $Cwidth}]
set Iscale [expr {($Imax - $Imin) / $Cheight}]
set maxIters 50

set S(draw) 0
set S(color) red
set S(title) "Tk Mandelbrot"
set S(version) 1.0
lappend S(stack) [list $Rmin $Imax $Rmax $Imin]
expr srand([clock clicks])

##+#########################################################################
#
# DoDisplay -- sets up our gui
#
proc DoDisplay {} {
   global Cwidth Cheight

   wm title . $::S(title)
   frame .bottom -bd 2 -relief ridge
   button .redraw -text "Redraw" -command Redraw
   set font "[font actual [.redraw cget -font]] -weight bold"
   .redraw configure -font $font
   catch {image create photo ::img::blank -width 1 -height 1}

   button .clear -text Clear -font $font -command Clear
   button .zoomin -text "Zoom In" -font $font -command ZoomIn
   button .zoomout -text "Zoom Out" -font $font -command ZoomOut
```

```
#!/usr/local/bin/tcl

source distcl.tcl

#retcl create redis     ;# to use local redis server
source redis-cloud.tcl  ;# to use cloud redis server

package require Tk

##+#########################################################################
#
# TkMandelbrot -- draws the mandelbrot fractal
# based on http://www.students.tut.fi/~warp/Mandelbrot/
# by Keith Vetter
#
# Revisions:
# KPV Nov 13, 2002 - initial revision
#
##+#########################################################################
###########################################################################

set colors_version 0

set Cwidth 500                                  ;# Canvas size
set Cheight 500

set Rmin -2.0                                   ;# Left side
set Rmax 1.0                                    ;# Right side
set Imin -1.5                                   ;# Bottom
set Imax [expr {$Imin + ($Rmax - $Rmin) * $Cheight /$Cwidth}]

set Rscale [expr {($Rmax - $Rmin) / $Cwidth}]
set Iscale [expr {($Imax - $Imin) / $Cheight}]
set maxIters 50

set S(draw) 0
set S(color) red
set S(title) "Tk Mandelbrot (distributed)"
set S(version) 1.0
lappend S(stack) [list $Rmin $Imax $Rmax $Imin]
expr srand([clock clicks])

##+#########################################################################
#
# DoDisplay -- sets up our gui
#
proc DoDisplay {} {
   global Cwidth Cheight

   wm title . $::S(title)
   frame .bottom -bd 2 -relief ridge
   button .redraw -text "Redraw" -command Redraw
   set font "[font actual [.redraw cget -font]] -weight bold"
   .redraw configure -font $font
   catch {image create photo ::img::blank -width 1 -height 1}

   button .clear -text Clear -font $font -command Clear
   button .zoomin -text "Zoom In" -font $font -command ZoomIn
   button .zoomout -text "Zoom Out" -font $font -command ZoomOut
```

```
    button .color -text "Select Color" -font $font -command {ChangeColor 0}
    button .random -text "Random Colors" -font $font -command {ChangeColor 1}
    button .about -image ::img::blank -command About -highlightthickness 0

    frame .flbl
    label .lbl -bd 2 -relief ridge -textvariable S(msg)
    canvas .c -width $Cwidth -height $Cheight -bd 2 -relief ridge -bg gray50 \
        -highlightthickness 0
    .c xview moveto 0 ; .c yview moveto 0
    image create photo ::img::myImage -width $Cwidth -height $Cheight
    .c create image 0 0 -image ::img::myImage -anchor nw -tag image

    ToggleButtons 0
    pack .bottom -side right -fill y -ipadx 10 -ipady 5

    set row -1
    grid rowconfigure .bottom [incr row] -minsize 5
    grid .zoomin  -in .bottom -sticky ew -pady 2 -row [incr row]
    grid .zoomout -in .bottom -sticky ew -pady 2 -row [incr row]
    grid rowconfigure .bottom [incr row] -minsize 20
    grid .redraw  -in .bottom -sticky ew -pady 2 -row [incr row]
    grid .clear   -in .bottom -sticky ew -pady 2 -row [incr row]
    grid rowconfigure .bottom [incr row] -minsize 20
    grid rowconfigure .bottom [incr row] -weight 1
    grid .color   -in .bottom -sticky ew -pady 2 -row [incr row]
    grid .random  -in .bottom -sticky ew -pady 2 -row [incr row]
    grid rowconfigure .bottom [incr row] -minsize 5

    pack .flbl -side bottom -fill x
    pack .lbl -in .flbl -side bottom -fill x
    pack .c -fill both -expand 1

    bind .c <Button-1> [list DoBox 0 %x %y]
    bind .c <B1-Motion> [list DoBox 1 %x %y]
    bind all <Alt-c> {console show}
    update
    pack propagate .flbl 0                      ;# Don't let it grow
    place .about -in .bottom -relx 1 -rely 1 -anchor se
}
##+#########################################################################
#
# ToggleButtons -- changes button state if we're drawing
#
proc ToggleButtons {drawing} {
    global S

    array set state {0 disabled 1 normal}

    if {$drawing} {
        foreach w {.zoomin .zoomout .clear .color .random} {
            $w config -state disabled
        }
        .redraw config -text "Stop Drawing"
        return
    }
    foreach w {.clear .color .random} {
        $w config -state normal
    }
    .zoomout config -state $state([expr {[llength $S(stack)] > 1}])
```

```
    button .color -text "Select Color" -font $font -command {ChangeColor 0}
    button .random -text "Random Colors" -font $font -command {ChangeColor 1}
    button .about -image ::img::blank -command About -highlightthickness 0

    frame .flbl
    label .lbl -bd 2 -relief ridge -textvariable S(msg)
    canvas .c -width $Cwidth -height $Cheight -bd 2 -relief ridge -bg gray50 \
        -highlightthickness 0
    .c xview moveto 0 ; .c yview moveto 0
    image create photo ::img::myImage -width $Cwidth -height $Cheight
    .c create image 0 0 -image ::img::myImage -anchor nw -tag image

    ToggleButtons 0
    pack .bottom -side right -fill y -ipadx 10 -ipady 5

    set row -1
    grid rowconfigure .bottom [incr row] -minsize 5
    grid .zoomin  -in .bottom -sticky ew -pady 2 -row [incr row]
    grid .zoomout -in .bottom -sticky ew -pady 2 -row [incr row]
    grid rowconfigure .bottom [incr row] -minsize 20
    grid .redraw  -in .bottom -sticky ew -pady 2 -row [incr row]
    grid .clear   -in .bottom -sticky ew -pady 2 -row [incr row]
    grid rowconfigure .bottom [incr row] -minsize 20
    grid rowconfigure .bottom [incr row] -weight 1
    grid .color   -in .bottom -sticky ew -pady 2 -row [incr row]
    grid .random  -in .bottom -sticky ew -pady 2 -row [incr row]
    grid rowconfigure .bottom [incr row] -minsize 5

    pack .flbl -side bottom -fill x
    pack .lbl -in .flbl -side bottom -fill x
    pack .c -fill both -expand 1

    bind .c <Button-1> [list DoBox 0 %x %y]
    bind .c <B1-Motion> [list DoBox 1 %x %y]
    bind all <Alt-c> {console show}
    update
    pack propagate .flbl 0                      ;# Don't let it grow
    place .about -in .bottom -relx 1 -rely 1 -anchor se
}
##+#########################################################################
#
# ToggleButtons -- changes button state if we're drawing
#
proc ToggleButtons {drawing} {
    global S

    array set state {0 disabled 1 normal}

    if {$drawing} {
        foreach w {.zoomin .zoomout .clear .color .random} {
            $w config -state disabled
        }
        .redraw config -text "Stop Drawing"
        return
    }
    foreach w {.clear .color .random} {
        $w config -state normal
    }
    .zoomout config -state $state([expr {[llength $S(stack)] > 1}])
```

```
110:     .zoomin  config -state $state([expr {[llength [.c find withtag box]] > 1}])
111:     .redraw config -text "Redraw"
112:  }
113:  ##+##########################################################################
114:  #
115:  # Render -- Renders the mandelbrot set
116:  #
117:  proc Render {} {
118:     global Cwidth Cheight Rmin Rmax Imin Imax maxIters Rscale Iscale
119:     global S
120:
121:     set sTime [clock click -milliseconds]
122:     ToggleButtons 1
123:     set S(draw) 1
124:
125:     if {[winfo ismapped .c]} {                  ;# Recompute scaling factors
126:         set Cheight [winfo height .c]
127:         set Cwidth [winfo width .c]
128:         set Rscale [expr {($Rmax - $Rmin) / $Cwidth}]
129:         set Iscale [expr {($Imax - $Imin) / $Cheight}]
130:     }
131:     Clear
132:     ::img::myImage config -width $Cwidth -height $Cheight
133:
134:     set step 4                                  ;# Do interlaced drawing
135:     for {set start 0} {$start < $step} {incr start} {
136:         for {set x $start} {$x < $Cwidth} {incr x $step} {
137:             set c_re [expr {$Rmin + $x * $Rscale}]
138:             set data ""
139:             for {set y 0} {$y < $Cheight} {incr y} {
140:                 set c_im [expr {$Imax - $y * $Iscale}]
141:
142:                 set z_re $c_re
143:                 set z_im $c_im
144:                 for {set n 0} {$n < $maxIters} {incr n} {
145:                     set z_re2 [expr {$z_re * $z_re}] ;# Have we escaped yet???
146:                     set z_im2 [expr {$z_im * $z_im}]
147:                     if {($z_re2 + $z_im2) > 4} {
148:                         break
149:                     }
150:                     set z_im [expr {2 * $z_re * $z_im + $c_im}]
151:                     set z_re [expr {$z_re2 - $z_im2 + $c_re}]
152:                 }
153:                 lappend data $::colors($n)
154:             }
155:             ::img::myImage put $data -to $x 0
156:             update
157:             if {$S(draw) == 0} break
158:         }
159:         if {$S(draw) == 0} break
```

```
119:     .zoomin  config -state $state([expr {[llength [.c find withtag box]] > 1}])
120:     .redraw config -text "Redraw"
121:  }
122:  ##+##########################################################################
123:  #
124:  # Render -- Renders the mandelbrot set
125:  #
126:  proc Render {} {
127:     global Cwidth Cheight Rmin Rmax Imin Imax maxIters Rscale Iscale
128:     global S
129:
130:     set sTime [clock click -milliseconds]
131:     ToggleButtons 1
132:     set S(draw) 1
133:
134:     if {[winfo ismapped .c]} {                  ;# Recompute scaling factors
135:         set Cheight [winfo height .c]
136:         set Cwidth [winfo width .c]
137:         set Rscale [expr {($Rmax - $Rmin) / $Cwidth}]
138:         set Iscale [expr {($Imax - $Imin) / $Cheight}]
139:     }
140:     Clear
141:     ::img::myImage config -width $Cwidth -height $Cheight
142:
143:     # Make color map available to compute servers
144:     redis -sync set mand_colors [array get ::colors]
145:
146:     # Compute servers will generate vertical slices as gifs
147:     set step 10                                 ;# width of a slice
148:     set col_ver [pid]:$::colors_version
149:
150:     # Prefetch all the slices so the compute servers can get busy
151:     for {set start 0} {$start < $Cwidth} {incr start $step} {
152:         set stop [expr {min($start+$step, $Cwidth)}]
153:         distcl::prefetch redis mand slice $Cheight $Rmin $Rscale $Imax $Iscale $
    start $stop $col_ver
154:     }
155:     # Get and display all the slices
156:     for {set start 0} {$start < $Cwidth} {incr start $step} {
157:         set stop [expr {min($start+$step, $Cwidth)}]
158:         set slice [distcl::get redis mand slice $Cheight $Rmin $Rscale $Imax $Is
    cale $start $stop $col_ver]
159:         ::img::myImage put $slice -format gif -to $start 0 $stop $Cheight
160:         update
161:         if {$S(draw) == 0} break
```

```
    }
    set S(draw) 0
    ToggleButtons 0
    set sTime [expr {([clock click -milliseconds] - $sTime) / 1000}]
    INFO "Time: [Duration $sTime]"
}
##+##########################################################################
#
# gradient -- adjusts a color to be "closer" to either white or black
# see https://wiki.tcl-lang.org/2847
#
proc gradient {rgb factor} {
    foreach {r g b} [winfo rgb . $rgb] {break}

    # Figure out color depth and number of bytes to use in the final result.
    set max 255; set len 2
    if {($r > 255) || ($g > 255) || ($b > 255)} {set max 65535; set len 4}

    # Compute new red value by incrementing the existing value by a
    # value that gets it closer to either 0 (black) or $max (white)
    set range [expr {$factor >= 0.0 ? $max - $r : $r}]
    set increment [expr {int($range * $factor)}]
    incr r $increment

    # Compute a new green value in a similar fashion
    set range [expr {$factor >= 0.0 ? $max - $g : $g}]
    set increment [expr {int($range * $factor)}]
    incr g $increment

    # Compute a new blue value in a similar fashion
    set range [expr {$factor >= 0.0 ? $max - $b : $b}]
    set increment [expr {int($range * $factor)}]
    incr b $increment

    ### Format the new rgb string
    set rgb [format "#%.${len}X%.${len}X%.${len}X" \
                [expr {($r>$max)?$max:(($r<0)?0:$r)}] \
                [expr {($g>$max)?$max:(($g<0)?0:$g)}] \
                [expr {($b>$max)?$max:(($b<0)?0:$b)}]]
    return $rgb
}
##+##########################################################################
#
# GradientColors
#
# Get maxIters number of colors in a gradient from black to white of
# color RGB.
#
proc GradientColors {{rgb red} {min -.5} {max .75}} {
    global S colors maxIters

    set S(color) $rgb
    for {set i 0} {$i <= $maxIters} {incr i} {
        set grad [expr {$min + 1.0* $i * ($max - $min) / $maxIters}]
        set colors($i) [gradient $rgb $grad]
    }
    set colors($maxIters) black
}
##+##########################################################################
```

```
    }
    set S(draw) 0
    ToggleButtons 0
    set sTime [expr {([clock click -milliseconds] - $sTime) / 1000}]
    INFO "Time: [Duration $sTime]"
}
##+##########################################################################
#
# gradient -- adjusts a color to be "closer" to either white or black
# see https://wiki.tcl-lang.org/2847
#
proc gradient {rgb factor} {
    foreach {r g b} [winfo rgb . $rgb] {break}

    # Figure out color depth and number of bytes to use in the final result.
    set max 255; set len 2
    if {($r > 255) || ($g > 255) || ($b > 255)} {set max 65535; set len 4}

    # Compute new red value by incrementing the existing value by a
    # value that gets it closer to either 0 (black) or $max (white)
    set range [expr {$factor >= 0.0 ? $max - $r : $r}]
    set increment [expr {int($range * $factor)}]
    incr r $increment

    # Compute a new green value in a similar fashion
    set range [expr {$factor >= 0.0 ? $max - $g : $g}]
    set increment [expr {int($range * $factor)}]
    incr g $increment

    # Compute a new blue value in a similar fashion
    set range [expr {$factor >= 0.0 ? $max - $b : $b}]
    set increment [expr {int($range * $factor)}]
    incr b $increment

    ### Format the new rgb string
    set rgb [format "#%.${len}X%.${len}X%.${len}X" \
                [expr {($r>$max)?$max:(($r<0)?0:$r)}] \
                [expr {($g>$max)?$max:(($g<0)?0:$g)}] \
                [expr {($b>$max)?$max:(($b<0)?0:$b)}]]
    return $rgb
}
##+##########################################################################
#
# GradientColors
#
# Get maxIters number of colors in a gradient from black to white of
# color RGB.
#
proc GradientColors {{rgb red} {min -.5} {max .75}} {
    global S colors maxIters

    set S(color) $rgb
    for {set i 0} {$i <= $maxIters} {incr i} {
        set grad [expr {$min + 1.0* $i * ($max - $min) / $maxIters}]
        set colors($i) [gradient $rgb $grad]
    }
    set colors($maxIters) black
}
##+##########################################################################
```

```
#
# RandomColors -- picks colors randomly
#
proc RandomColors {} {
   global colors maxIters
   for {set i 0} {$i <= $maxIters} {incr i} {
       set colors($i) [format "\#%04X%04X%04X" [expr {int(rand() * 0xFFFF)}] \
              [expr {int(rand() * 0xFFFF)}] [expr {int(rand() * 0xFFFF)}]]
   }
   set colors($maxIters) black
}
##+########################################################################
#
# ChangeColor -- puts in a new color scheme
#
proc ChangeColor {random} {
   global S maxIters
   if {$random} {
       RandomColors
       INFO "Selecting new colors randomly -- press Redraw to see"
   } else {
       set color [tk_chooseColor -initialcolor $S(color) -parent . \
                      -title "Tk Mandelbrot Color"]
       if {$color == ""} return
       INFO "Setting new color $color -- press Redraw to see"
       GradientColors $color
   }

}
##+########################################################################
#
# Canvas2Z -- converts from canvas to mandelbrot coordinates
#
proc Canvas2Z {x y} {
   global Rmin Imax Rscale Iscale

   set re [expr {$Rmin + $Rscale * $x}]
   #set im [expr {$Imin + $Iscale * $y}]
   set im [expr {$Imax - $Iscale * $y}]

   return [list $re $im]
}
##+########################################################################
#
# DoBox -- handles mousing to create the zoom box
#
proc DoBox {what x y} {
   global B

   .c delete box
   if {$what == 0} {                           ;# Button down
       .zoomin config -state disabled          ;# No box, no button
       set B(x0) [.c canvasx $x]
       set B(y0) [.c canvasx $y]
   } else {                                    ;# Button motion
       set B(x1) [.c canvasx $x]
       set B(y1) [.c canvasx $y]

       .c create rect $B(x0) $B(y0) $B(x1) $B(y1) -outline white -tag box \
```

```
#
# RandomColors -- picks colors randomly
#
proc RandomColors {} {
   global colors maxIters
   for {set i 0} {$i <= $maxIters} {incr i} {
       set colors($i) [format "\#%04X%04X%04X" [expr {int(rand() * 0xFFFF)}] \
              [expr {int(rand() * 0xFFFF)}] [expr {int(rand() * 0xFFFF)}]]
   }
   set colors($maxIters) black
}
##+########################################################################
#
# ChangeColor -- puts in a new color scheme
#
proc ChangeColor {random} {
   global S maxIters
   if {$random} {
       RandomColors
       INFO "Selecting new colors randomly -- press Redraw to see"
   } else {
       set color [tk_chooseColor -initialcolor $S(color) -parent . \
                      -title "Tk Mandelbrot Color"]
       if {$color == ""} return
       INFO "Setting new color $color -- press Redraw to see"
       GradientColors $color
   }
   incr ::colors_version
}
##+########################################################################
#
# Canvas2Z -- converts from canvas to mandelbrot coordinates
#
proc Canvas2Z {x y} {
   global Rmin Imax Rscale Iscale

   set re [expr {$Rmin + $Rscale * $x}]
   #set im [expr {$Imin + $Iscale * $y}]
   set im [expr {$Imax - $Iscale * $y}]

   return [list $re $im]
}
##+########################################################################
#
# DoBox -- handles mousing to create the zoom box
#
proc DoBox {what x y} {
   global B

   .c delete box
   if {$what == 0} {                           ;# Button down
       .zoomin config -state disabled          ;# No box, no button
       set B(x0) [.c canvasx $x]
       set B(y0) [.c canvasx $y]
   } else {                                    ;# Button motion
       set B(x1) [.c canvasx $x]
       set B(y1) [.c canvasx $y]

       .c create rect $B(x0) $B(y0) $B(x1) $B(y1) -outline white -tag box \
```

```
277:            -dash 1
278:        .zoomin config -state normal           ;# Have box, have button
279:    }
280: }
281: ##+#################################################################
282: #
283: # Redraw -- starts or stops drawing of the fractal
284: #
285: proc Redraw {} {
286:    global S
287:
288:    if {$S(draw)} {
289:        INFO "stopping"
290:        set S(draw) 0
291:        return
292:    }
293:    INFO "redrawing..."
294:    Render
295: }
296: ##+#################################################################
297: #
298: # ZoomIn -- zooms in the display to the box on the screen
299: #
300: proc ZoomIn {} {
301:    global S Rmin Rmax Imin Imax
302:
303:    INFO "zooming in..."
304:    if {[.c find withtag box] != ""} {
305:        foreach {x0 y0 x1 y1} [.c bbox box] break
306:        .c delete box
307:
308:        foreach {Rmin2 Imax2} [Canvas2Z $x0 $y0] break
309:        foreach {Rmax2 Imin2} [Canvas2Z $x1 $y1] break
310:
311:        foreach {Rmin Rmax Imin Imax} \
312:            [list $Rmin2 $Rmax2 $Imin2 $Imax2] break
313:    }
314:    lappend S(stack) [list $Rmin $Imax $Rmax $Imin]
315:    after 1 Render
316: }
317: ##+#################################################################
318: #
319: # ZoomOut -- pops coordinates off stack and renders them
320: #
321: proc ZoomOut {} {
322:    global S Rmin Rmax Imin Imax
323:
324:    if {[llength $S(stack)] < 2} return
325:    INFO "zooming out..."
326:    set a [lindex $S(stack) end-1]
327:    set S(stack) [lrange $S(stack) 0 end-1]     ;# Leave current at the end
328:
329:    foreach {Rmin Imax Rmax Imin} $a break
330:    after 1 Render
331: }
332: proc INFO {msg} {
333:    set ::S(msg) $msg
334: }
335: proc About {} {
```

```
280:            -dash 1
281:        .zoomin config -state normal           ;# Have box, have button
282:    }
283: }
284: ##+#################################################################
285: #
286: # Redraw -- starts or stops drawing of the fractal
287: #
288: proc Redraw {} {
289:    global S
290:
291:    if {$S(draw)} {
292:        INFO "stopping"
293:        set S(draw) 0
294:        return
295:    }
296:    INFO "redrawing..."
297:    Render
298: }
299: ##+#################################################################
300: #
301: # ZoomIn -- zooms in the display to the box on the screen
302: #
303: proc ZoomIn {} {
304:    global S Rmin Rmax Imin Imax
305:
306:    INFO "zooming in..."
307:    if {[.c find withtag box] != ""} {
308:        foreach {x0 y0 x1 y1} [.c bbox box] break
309:        .c delete box
310:
311:        foreach {Rmin2 Imax2} [Canvas2Z $x0 $y0] break
312:        foreach {Rmax2 Imin2} [Canvas2Z $x1 $y1] break
313:
314:        foreach {Rmin Rmax Imin Imax} \
315:            [list $Rmin2 $Rmax2 $Imin2 $Imax2] break
316:    }
317:    lappend S(stack) [list $Rmin $Imax $Rmax $Imin]
318:    after 1 Render
319: }
320: ##+#################################################################
321: #
322: # ZoomOut -- pops coordinates off stack and renders them
323: #
324: proc ZoomOut {} {
325:    global S Rmin Rmax Imin Imax
326:
327:    if {[llength $S(stack)] < 2} return
328:    INFO "zooming out..."
329:    set a [lindex $S(stack) end-1]
330:    set S(stack) [lrange $S(stack) 0 end-1]     ;# Leave current at the end
331:
332:    foreach {Rmin Imax Rmax Imin} $a break
333:    after 1 Render
334: }
335: proc INFO {msg} {
336:    set ::S(msg) $msg
337: }
338: proc About {} {
```

```
336:     tk_messageBox -icon info -parent . -title "About $::S(title)" \
337:         -message "$::S(title)\n\nby Keith Vetter\nNovember, 2002"
338:  }
339:  ##+##########################################################################
340:  #
341:  # Duration - Prints out seconds in a nice format
342:  # https://wiki.tcl-lang.org/789
343:  #
344:  proc Duration { int_time } {
345:     if {$int_time == 0} {return "0 secs"}
346:     set timeList [list]
347:     foreach div {86400 3600 60 1} mod {0 24 60 60} name {day hr min sec} {
348:         set n [expr {$int_time / $div}]
349:         if {$mod > 0} {set n [expr {$n % $mod}]}
350:         if {$n > 1} {
351:             lappend timeList "$n ${name}s"
352:         } elseif {$n == 1} {
353:             lappend timeList "$n $name"
354:         }
355:     }
356:     return [join $timeList]
357:  }
358:  proc Clear {} {
359:     .c delete box
360:     ::img::myImage blank
361:  }
362:  ###############################################################
363:  ###############################################################
364:  ###############################################################
365:
366:  DoDisplay
367:  RandomColors
368:  INFO "Welcome to Tk Mandelbrot"
369:  Render
```

```
339:     tk_messageBox -icon info -parent . -title "About $::S(title)" \
340:         -message "$::S(title)\n\nby Keith Vetter\nNovember, 2002"
341:  }
342:  ##+##########################################################################
343:  #
344:  # Duration - Prints out seconds in a nice format
345:  # https://wiki.tcl-lang.org/789
346:  #
347:  proc Duration { int_time } {
348:     if {$int_time == 0} {return "0 secs"}
349:     set timeList [list]
350:     foreach div {86400 3600 60 1} mod {0 24 60 60} name {day hr min sec} {
351:         set n [expr {$int_time / $div}]
352:         if {$mod > 0} {set n [expr {$n % $mod}]}
353:         if {$n > 1} {
354:             lappend timeList "$n ${name}s"
355:         } elseif {$n == 1} {
356:             lappend timeList "$n $name"
357:         }
358:     }
359:     return [join $timeList]
360:  }
361:  proc Clear {} {
362:     .c delete box
363:     ::img::myImage blank
364:  }
365:  ###############################################################
366:  ###############################################################
367:  ###############################################################
368:
369:  DoDisplay
370:  GradientColors skyblue
371:  INFO "Welcome to Tk Mandelbrot"
372:  Render
```