

```

1: ##+#####
2: #
3: # TkMandelbrot -- draws the mandelbrot fractal
4: # based on http://www.students.tut.fi/~warp/Mandelbrot/
5: # by Keith Vetter
6: #
7: # Revisions:
8: # KPV Nov 13, 2002 - initial revision
9: #
10: ##+#####
11: #####
12:
13: set tcl_precision 17
14:
15: set Cwidth 500                ;# Canvas size
16: set Cheight 500
17:
18: set Rmin -2.0                 ;# Left side
19: set Rmax 1.0                 ;# Right side
20: set Imin -1.5                ;# Bottom
21: set Imax [expr {$Imin + ($Rmax - $Rmin) * $Cheight / $Cwidth}]
22:
23: set Rscale [expr {($Rmax - $Rmin) / $Cwidth}]
24: set Iscale [expr {($Imax - $Imin) / $Cheight}]
25: set maxIters 50
26:
27: set S(draw) 0
28: set S(color) red
29: set S(title) "Tk Mandelbrot"
30: set S(version) 1.0
31: lappend S(stack) [list $Rmin $Imax $Rmax $Imin]
32: expr srand([clock clicks])
33:
34: ##+#####
35: #
36: # DoDisplay -- sets up our gui
37: #
38: proc DoDisplay {} {
39:     global Cwidth Cheight
40:
41:     wm title . $::S(title)
42:     frame .bottom -bd 2 -relief ridge
43:     button .redraw -text "Redraw" -command Redraw
44:     set font "[font actual [.redraw cget -font]] -weight bold"
45:     .redraw configure -font $font
46:     catch {image create photo ::img::blank -width 1 -height 1}
47:
48:     button .clear -text Clear -font $font -command Clear
49:     button .zoomin -text "Zoom In" -font $font -command ZoomIn
50:     button .zoomout -text "Zoom Out" -font $font -command ZoomOut

```

```

1: #!/usr/local/bin/tcl
2:
3: source distcl.tcl
4:
5: #retcl create redis          ;# to use local redis server
6: source redis-cloud.tcl      ;# to use cloud redis server
7:
8: package require Tk
9:
10: ##+#####
11: #
12: # TkMandelbrot -- draws the mandelbrot fractal
13: # based on http://www.students.tut.fi/~warp/Mandelbrot/
14: # by Keith Vetter
15: #
16: # Revisions:
17: # KPV Nov 13, 2002 - initial revision
18: #
19: ##+#####
20: #####
21:
22: set colors_version 0
23:
24: set Cwidth 500                ;# Canvas size
25: set Cheight 500
26:
27: set Rmin -2.0                 ;# Left side
28: set Rmax 1.0                 ;# Right side
29: set Imin -1.5                ;# Bottom
30: set Imax [expr {$Imin + ($Rmax - $Rmin) * $Cheight / $Cwidth}]
31:
32: set Rscale [expr {($Rmax - $Rmin) / $Cwidth}]
33: set Iscale [expr {($Imax - $Imin) / $Cheight}]
34: set maxIters 50
35:
36: set S(draw) 0
37: set S(color) red
38: set S(title) "Tk Mandelbrot (distributed)"
39: set S(version) 1.0
40: lappend S(stack) [list $Rmin $Imax $Rmax $Imin]
41: expr srand([clock clicks])
42:
43: ##+#####
44: #
45: # DoDisplay -- sets up our gui
46: #
47: proc DoDisplay {} {
48:     global Cwidth Cheight
49:
50:     wm title . $::S(title)
51:     frame .bottom -bd 2 -relief ridge
52:     button .redraw -text "Redraw" -command Redraw
53:     set font "[font actual [.redraw cget -font]] -weight bold"
54:     .redraw configure -font $font
55:     catch {image create photo ::img::blank -width 1 -height 1}
56:
57:     button .clear -text Clear -font $font -command Clear
58:     button .zoomin -text "Zoom In" -font $font -command ZoomIn
59:     button .zoomout -text "Zoom Out" -font $font -command ZoomOut

```

```

51: button .color -text "Select Color" -font $font -command {ChangeColor 0}
52: button .random -text "Random Colors" -font $font -command {ChangeColor 1}
53: button .about -image ::img::blank -command About -highlightthickness 0
54:
55: frame .flbl
56: label .lbl -bd 2 -relief ridge -textvariable S(msg)
57: canvas .c -width $Cwidth -height $Cheight -bd 2 -relief ridge -bg gray50 \
58:   -highlightthickness 0
59: .c xview moveto 0 ; .c yview moveto 0
60: image create photo ::img::myImage -width $Cwidth -height $Cheight
61: .c create image 0 0 -image ::img::myImage -anchor nw -tag image
62:
63: ToggleButtons 0
64: pack .bottom -side right -fill y -ipadx 10 -ipady 5
65:
66: set row -1
67: grid rowconfigure .bottom [incr row] -minsize 5
68: grid .zoomin -in .bottom -sticky ew -pady 2 -row [incr row]
69: grid .zoomout -in .bottom -sticky ew -pady 2 -row [incr row]
70: grid rowconfigure .bottom [incr row] -minsize 20
71: grid .redraw -in .bottom -sticky ew -pady 2 -row [incr row]
72: grid .clear -in .bottom -sticky ew -pady 2 -row [incr row]
73: grid rowconfigure .bottom [incr row] -minsize 20
74: grid rowconfigure .bottom [incr row] -weight 1
75: grid .color -in .bottom -sticky ew -pady 2 -row [incr row]
76: grid .random -in .bottom -sticky ew -pady 2 -row [incr row]
77: grid rowconfigure .bottom [incr row] -minsize 5
78:
79: pack .flbl -side bottom -fill x
80: pack .lbl -in .flbl -side bottom -fill x
81: pack .c -fill both -expand 1
82:
83: bind .c <Button-1> [list DoBox 0 %x %y]
84: bind .c <B1-Motion> [list DoBox 1 %x %y]
85: bind all <Alt-c> {console show}
86: update
87: pack propagate .flbl 0 ;# Don't let it grow
88: place .about -in .bottom -relx 1 -rely 1 -anchor se
89: }
90: ##+#####
91: #
92: # ToggleButtons -- changes button state if we're drawing
93: #
94: proc ToggleButtons {drawing} {
95:   global S
96:
97:   array set state {0 disabled 1 normal}
98:
99:   if {$drawing} {
100:     foreach w {.zoomin .zoomout .clear .color .random} {
101:       $w config -state disabled
102:     }
103:     .redraw config -text "Stop Drawing"
104:     return
105:   }
106:   foreach w {.clear .color .random} {
107:     $w config -state normal
108:   }
109:   .zoomout config -state $state([expr {[llength $S(stack)] > 1}])

```

```

60: button .color -text "Select Color" -font $font -command {ChangeColor 0}
61: button .random -text "Random Colors" -font $font -command {ChangeColor 1}
62: button .about -image ::img::blank -command About -highlightthickness 0
63:
64: frame .flbl
65: label .lbl -bd 2 -relief ridge -textvariable S(msg)
66: canvas .c -width $Cwidth -height $Cheight -bd 2 -relief ridge -bg gray50 \
67:   -highlightthickness 0
68: .c xview moveto 0 ; .c yview moveto 0
69: image create photo ::img::myImage -width $Cwidth -height $Cheight
70: .c create image 0 0 -image ::img::myImage -anchor nw -tag image
71:
72: ToggleButtons 0
73: pack .bottom -side right -fill y -ipadx 10 -ipady 5
74:
75: set row -1
76: grid rowconfigure .bottom [incr row] -minsize 5
77: grid .zoomin -in .bottom -sticky ew -pady 2 -row [incr row]
78: grid .zoomout -in .bottom -sticky ew -pady 2 -row [incr row]
79: grid rowconfigure .bottom [incr row] -minsize 20
80: grid .redraw -in .bottom -sticky ew -pady 2 -row [incr row]
81: grid .clear -in .bottom -sticky ew -pady 2 -row [incr row]
82: grid rowconfigure .bottom [incr row] -minsize 20
83: grid rowconfigure .bottom [incr row] -weight 1
84: grid .color -in .bottom -sticky ew -pady 2 -row [incr row]
85: grid .random -in .bottom -sticky ew -pady 2 -row [incr row]
86: grid rowconfigure .bottom [incr row] -minsize 5
87:
88: pack .flbl -side bottom -fill x
89: pack .lbl -in .flbl -side bottom -fill x
90: pack .c -fill both -expand 1
91:
92: bind .c <Button-1> [list DoBox 0 %x %y]
93: bind .c <B1-Motion> [list DoBox 1 %x %y]
94: bind all <Alt-c> {console show}
95: update
96: pack propagate .flbl 0 ;# Don't let it grow
97: place .about -in .bottom -relx 1 -rely 1 -anchor se
98: }
99: ##+#####
100: #
101: # ToggleButtons -- changes button state if we're drawing
102: #
103: proc ToggleButtons {drawing} {
104:   global S
105:
106:   array set state {0 disabled 1 normal}
107:
108:   if {$drawing} {
109:     foreach w {.zoomin .zoomout .clear .color .random} {
110:       $w config -state disabled
111:     }
112:     .redraw config -text "Stop Drawing"
113:     return
114:   }
115:   foreach w {.clear .color .random} {
116:     $w config -state normal
117:   }
118:   .zoomout config -state $state([expr {[llength $S(stack)] > 1}])

```

```

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

```

```

119: .zoomin config -state $state([expr {[llength [.c find withtag box]] > 1})
120: .redraw config -text "Redraw"
121: }
122: ##+#####
123: #
124: # Render -- Renders the mandelbrot set
125: #
126: proc Render {} {
127:     global Cwidth Cheight Rmin Rmax Imin Imax maxIters Rscale Iscale
128:     global S
129:
130:     set sTime [clock click -milliseconds]
131:     ToggleButtons 1
132:     set S(draw) 1
133:
134:     if {[wininfo ismapped .c]} {                ;# Recompute scaling factors
135:         set Cheight [wininfo height .c]
136:         set Cwidth [wininfo width .c]
137:         set Rscale [expr {($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:
154:         distcl::prefetch redis mand slice $Cheight $Rmin $Rscale $Imax $Iscale $S
155:         start $stop $col_ver
156:     }
157:     # Get and display all the slices
158:     for {set start 0} {$start < $Cwidth} {incr start $step} {
159:         set stop [expr {min($start+$step, $Cwidth)}]
160:         set slice [distcl::get redis mand slice $Cheight $Rmin $Rscale $Imax $Is
161:             cale $start $stop $col_ver]
162:         ::img::myImage put $slice -format gif -to $start 0 $stop $Cheight
163:         update
164:         if {$S(draw) == 0} break

```

```

160:     }
161:     set S(draw) 0
162:     ToggleButtons 0
163:     set sTime [expr {[[clock click -milliseconds] - $sTime) / 1000}]
164:     INFO "Time: [Duration $sTime]"
165: }
166: ##+#####
167: #
168: # gradient -- adjusts a color to be "closer" to either white or black
169: # see https://wiki.tcl-lang.org/2847
170: #
171: proc gradient {rgb factor} {
172:     foreach {r g b} [winfo rgb . $rgb] {break}
173:
174:     # Figure out color depth and number of bytes to use in the final result.
175:     set max 255; set len 2
176:     if {($r > 255) || ($g > 255) || ($b > 255)} {set max 65535; set len 4}
177:
178:     # Compute new red value by incrementing the existing value by a
179:     # value that gets it closer to either 0 (black) or $max (white)
180:     set range [expr {$factor >= 0.0 ? $max - $r : $r}]
181:     set increment [expr {int($range * $factor)}]
182:     incr r $increment
183:
184:     # Compute a new green value in a similar fashion
185:     set range [expr {$factor >= 0.0 ? $max - $g : $g}]
186:     set increment [expr {int($range * $factor)}]
187:     incr g $increment
188:
189:     # Compute a new blue value in a similar fashion
190:     set range [expr {$factor >= 0.0 ? $max - $b : $b}]
191:     set increment [expr {int($range * $factor)}]
192:     incr b $increment
193:
194:     ### Format the new rgb string
195:     set rgb [format "%.${len}X%.${len}X%.${len}X" \
196:             [expr {($r>$max)?$max:((($r<0)?0:$r)}] \
197:             [expr {($g>$max)?$max:((($g<0)?0:$g)}] \
198:             [expr {($b>$max)?$max:((($b<0)?0:$b)}]
199:     return $rgb
200: }
201: ##+#####
202: #
203: # GradientColors
204: #
205: # Get maxIters number of colors in a gradient from black to white of
206: # color RGB.
207: #
208: proc GradientColors {{rgb red} {min -.5} {max .75}} {
209:     global S colors maxIters
210:
211:     set S(color) $rgb
212:     for {set i 0} {$i <= $maxIters} {incr i} {
213:         set grad [expr {$min + 1.0* $i * ($max - $min) / $maxIters}]
214:         set colors($i) [gradient $rgb $grad]
215:     }
216:     set colors($maxIters) black
217: }
218: ##+#####

```

```

162:     }
163:     set S(draw) 0
164:     ToggleButtons 0
165:     set sTime [expr {[[clock click -milliseconds] - $sTime) / 1000}]
166:     INFO "Time: [Duration $sTime]"
167: }
168: ##+#####
169: #
170: # gradient -- adjusts a color to be "closer" to either white or black
171: # see https://wiki.tcl-lang.org/2847
172: #
173: proc gradient {rgb factor} {
174:     foreach {r g b} [winfo rgb . $rgb] {break}
175:
176:     # Figure out color depth and number of bytes to use in the final result.
177:     set max 255; set len 2
178:     if {($r > 255) || ($g > 255) || ($b > 255)} {set max 65535; set len 4}
179:
180:     # Compute new red value by incrementing the existing value by a
181:     # value that gets it closer to either 0 (black) or $max (white)
182:     set range [expr {$factor >= 0.0 ? $max - $r : $r}]
183:     set increment [expr {int($range * $factor)}]
184:     incr r $increment
185:
186:     # Compute a new green value in a similar fashion
187:     set range [expr {$factor >= 0.0 ? $max - $g : $g}]
188:     set increment [expr {int($range * $factor)}]
189:     incr g $increment
190:
191:     # Compute a new blue value in a similar fashion
192:     set range [expr {$factor >= 0.0 ? $max - $b : $b}]
193:     set increment [expr {int($range * $factor)}]
194:     incr b $increment
195:
196:     ### Format the new rgb string
197:     set rgb [format "%.${len}X%.${len}X%.${len}X" \
198:             [expr {($r>$max)?$max:((($r<0)?0:$r)}] \
199:             [expr {($g>$max)?$max:((($g<0)?0:$g)}] \
200:             [expr {($b>$max)?$max:((($b<0)?0:$b)}]
201:     return $rgb
202: }
203: ##+#####
204: #
205: # GradientColors
206: #
207: # Get maxIters number of colors in a gradient from black to white of
208: # color RGB.
209: #
210: proc GradientColors {{rgb red} {min -.5} {max .75}} {
211:     global S colors maxIters
212:
213:     set S(color) $rgb
214:     for {set i 0} {$i <= $maxIters} {incr i} {
215:         set grad [expr {$min + 1.0* $i * ($max - $min) / $maxIters}]
216:         set colors($i) [gradient $rgb $grad]
217:     }
218:     set colors($maxIters) black
219: }
220: ##+#####

```

```

219: #
220: # RandomColors -- picks colors randomly
221: #
222: proc RandomColors {} {
223:     global colors maxIters
224:     for {set i 0} {$i <= $maxIters} {incr i} {
225:         set colors($i) [format "\#%04X%04X%04X" [expr {int(rand() * 0xFFFF)}] \
226:             [expr {int(rand() * 0xFFFF)}] [expr {int(rand() * 0xFFFF)}]]
227:     }
228:     set colors($maxIters) black
229: }
230: ##+#####
231: #
232: # ChangeColor -- puts in a new color scheme
233: #
234: proc ChangeColor {random} {
235:     global S maxIters
236:     if {$random} {
237:         RandomColors
238:         INFO "Selecting new colors randomly -- press Redraw to see"
239:     } else {
240:         set color [tk_chooseColor -initialcolor $$color -parent . \
241:             -title "Tk Mandelbrot Color"]
242:         if {$color == ""} return
243:         INFO "Setting new color $color -- press Redraw to see"
244:         GradientColors $color
245:     }
246: }
247: ##+#####
248: #
249: # Canvas2Z -- converts from canvas to mandelbrot coordinates
250: #
251: proc Canvas2Z {x y} {
252:     global Rmin Imax Rscale Iscale
253:
254:     set re [expr {$Rmin + $Rscale * $x}]
255:     #set im [expr {$Imin + $Iscale * $y}]
256:     set im [expr {$Imax - $Iscale * $y}]
257:
258:     return [list $re $im]
259: }
260: ##+#####
261: #
262: # DoBox -- handles mousing to create the zoom box
263: #
264: proc DoBox {what x y} {
265:     global B
266:
267:     .c delete box
268:     if {$what == 0} {
269:         .zoomin config -state disabled           ;# Button down
270:         set B(x0) [.c canvasx $x]                ;# No box, no button
271:         set B(y0) [.c canvasx $y]
272:     } else {
273:         ;# Button motion
274:         set B(x1) [.c canvasx $x]
275:         set B(y1) [.c canvasx $y]
276:     }
277:     .c create rect $B(x0) $B(y0) $B(x1) $B(y1) -outline white -tag box \

```

```

221: #
222: # RandomColors -- picks colors randomly
223: #
224: proc RandomColors {} {
225:     global colors maxIters
226:     for {set i 0} {$i <= $maxIters} {incr i} {
227:         set colors($i) [format "\#%04X%04X%04X" [expr {int(rand() * 0xFFFF)}] \
228:             [expr {int(rand() * 0xFFFF)}] [expr {int(rand() * 0xFFFF)}]]
229:     }
230:     set colors($maxIters) black
231: }
232: ##+#####
233: #
234: # ChangeColor -- puts in a new color scheme
235: #
236: proc ChangeColor {random} {
237:     global S maxIters
238:     if {$random} {
239:         RandomColors
240:         INFO "Selecting new colors randomly -- press Redraw to see"
241:     } else {
242:         set color [tk_chooseColor -initialcolor $$color -parent . \
243:             -title "Tk Mandelbrot Color"]
244:         if {$color == ""} return
245:         INFO "Setting new color $color -- press Redraw to see"
246:         GradientColors $color
247:     }
248:     incr ::colors_version
249: }
250: ##+#####
251: #
252: # Canvas2Z -- converts from canvas to mandelbrot coordinates
253: #
254: proc Canvas2Z {x y} {
255:     global Rmin Imax Rscale Iscale
256:
257:     set re [expr {$Rmin + $Rscale * $x}]
258:     #set im [expr {$Imin + $Iscale * $y}]
259:     set im [expr {$Imax - $Iscale * $y}]
260:
261:     return [list $re $im]
262: }
263: ##+#####
264: #
265: # DoBox -- handles mousing to create the zoom box
266: #
267: proc DoBox {what x y} {
268:     global B
269:
270:     .c delete box
271:     if {$what == 0} {
272:         .zoomin config -state disabled           ;# Button down
273:         set B(x0) [.c canvasx $x]                ;# No box, no button
274:         set B(y0) [.c canvasx $y]
275:     } else {
276:         ;# Button motion
277:         set B(x1) [.c canvasx $x]
278:         set B(y1) [.c canvasx $y]
279:     }
280:     .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
```