Appendix C: Cross-Sectioning Functions

```r
get_midpoint <- function( f ) min( f ) + 0.5 * ( max( f ) - min( f ) )

curve_axes <- function( mdls, idx ){
  # Since the model is a thin slice, one axis will have much smaller
  # extents than the other two, which we will use
  # for our 2D curve coordinates, X and Y.
  # For now, assume that the largest extent is X
  # and the second-largest extent is Y
  extents <- mdls[[ idx ]]$extents()

  # Treat the slice as a two-dimensional curve
  # with coordinates along curve_x_axis and curve_y_axis
  # and curve_z_axis being razor thin.

  curve_x_axis <- which.max( extents ) %>% names() # 'x', 'y', or 'z'
  curve_z_axis <- which.min( extents ) %>% names() # 'x', 'y', or 'z'
  curve_y_axis <- setdiff( axes, c( curve_x_axis, curve_z_axis ))

  list( cx = curve_x_axis, cy = curve_y_axis, cz = curve_z_axis )
}

get_curve <- function( x, idx, fun='min', ax = axs ){
  # The wall of the vessel might be thick, and the inner
  # profile could differ from the outer profile
  # fun='min' gets one side; fun='max' gets the other side
  get_curve_points(
      model_matrix = models[[ idx ]]$get()
    , as_width  = ax$cx
    , as_height = ax$cz
    , as_curve  = ax$cy
    , tol = 5 * STRIPE_WIDTH
    , height = models[[ idx ]]$calculate( median )[ ax$cz ]
    , width  = x
    , fun = fun
  )
}

# Calculate the slope of each segment defined by
# each consecutive pair of perimeter points
local_slope <- function( i, data ){
```

```
  names( data ) <- c( 'x', 'y' )
  delta_y <- data[ i, 'y' ] - data[ i - 1, 'y' ]
  delta_x <- data[ i, 'x' ] - data[ i - 1, 'x' ]
  round( delta_y / delta_x, 3 )
}

find_center <- function( data, slopes, a = 20, b = 80 ){
  # Given two points, A and B, on an arc,
  # find the intersection of rays AO and BO

  N <- nrow( data ) # Number of points available on arc
  index_A <- round( a/100 * N, 0 ) # offset of point A
  index_B <- round( b/100 * N, 0 ) # offset of point B
  ray_AO  <- get_perpendicular2D( m = slopes[ index_A ], P = data[ index_A, ] )
  ray_BO  <- get_perpendicular2D( m = slopes[ index_B ], P = data[ index_B, ] )
  get_intersection2D( ray_AO, ray_BO )
}

sketch_radius <- function( xyz, fixed_coordinate, ctr, data, i ){
  NUM_POINTS <- 2

  # Decide the X, Y, and Z coordinates to be plotted
  items <- list()
  items[[ 1 ]] <- rep( fixed_coordinate, NUM_POINTS )
  items[[ 2 ]] <- seq( ctr[ 'x' ], data[ i, 'x' ], length.out = NUM_POINTS )
  items[[ 3 ]] <- seq( ctr[ 'y' ], data[ i, 'y' ], length.out = NUM_POINTS )

  # Decide which axis is X, Y, and Z for this object
  segx <- unlist( items[ xyz[[ 1 ]] ] )
  segy <- unlist( items[ xyz[[ 2 ]] ] )
  segz <- unlist( items[ xyz[[ 3 ]] ] )
  segments3d( x = segx, y = segy, z = segz )
}

# receive a series of X coordinates as data[ i ]
# convert each X coordinate into a Y coordinate on
# the perimeter of a circle having center ctr and
# radius r

sketch_arc <- function( xyz, fixed_coordinate, ctr, arc_data, i, r ){
  NUM_POINTS <- 10                # Number of arc points

  # Decide the X, Y, and Z coordinates to be plotted
  items <- list()
```

```
  # The first item is the Z coordinate
  # The first items is easy -- it doesn't change, since
  # we are making a 2D plot

  # Here, we need to calculate Y, given X, r, a, and b
  a <- ctr[ 'x' ]
  b <- ctr[ 'y' ]
  x <- arc_data[ i, 'x' ]
  yplus  <- b + sqrt( (r + ( x - a ) ) * ( r - ( x - a ) ))
  yminus <- b - sqrt( (r + ( x - a ) ) * ( r - ( x - a ) ))
  y <- c( yplus, yminus )
  x <- c( x    , x      )
  items[[ 1 ]] <- rep( fixed_coordinate, NUM_POINTS )
  items[[ 2 ]] <- x              # The second item is the X coordinate
  items[[ 3 ]] <- y              # The third item is the Y coordinate

  # Decide which axis is X, Y, and Z for this object
  segx <- unlist( items[ xyz[[ 1 ]] ] )
  segy <- unlist( items[ xyz[[ 2 ]] ] )
  segz <- unlist( items[ xyz[[ 3 ]] ] )

  # As a side-effect, plot the point
  points3d( x = segx, y = segy, z = segz, color = 'red', size = 0.5 )

  # Return the coordinates X, Y
  list( x = x, y = y )
}

illustrate_slice_centers <- function( models, model_index, file_name ){
  N <- 20 # Number of arc points to use

  smooth_curve <- get_xy( models = models, model_index = model_index, N = N )

  # Calculate the slope of each segment defined by each consecutive pair of perimeter points
  slopes <- map_dbl( 2:N, ~local_slope( .x, smooth_curve ))

  # Estimate the center, based on several perimeter points
  centers <- map2_df( 15:25, 65:75, ~find_center( data = smooth_curve, slopes=slopes, .x, .y ))
  est_ctr <- apply( centers, 2, mean)
  est_radius <- mean( map_dbl( 1:nrow( smooth_curve ), ~euclidean_distance( est_ctr, smooth_curve[ .x, 

  sliced_at <- models[[ model_index ]]$calculate( range )[ , model_index ] %>% mean()
  items <- list()
```

```
  items[[ 1 ]] <- sliced_at
  items[[ 2 ]] <- est_ctr[ 'x' ]
  items[[ 3 ]] <- est_ctr[ 'y' ]

  xyz <- orient( model_index )
  origin_x <- items[ xyz[[ 1 ]] ]
  origin_y <- items[ xyz[[ 2 ]] ]
  origin_z <- items[ xyz[[ 3 ]] ]

  show_slice( mdls = models, idx = model_index )
  show_center( origin_x, origin_y, origin_z, est_radius, flavor='point' )
  show_radii(
      idx = model_index
    , ctr = est_ctr
    , crv = smooth_curve
    , fixed_coordinate = sliced_at
  )
  arc_points <- show_arc(
      idx = model_index
    , ctr = est_ctr
    , crv = smooth_curve
    , fixed_coordinate = sliced_at
    , r   = est_radius
  )
  rgl.snapshot( filename = paste0( FIGURES_PATH, file_name, '.png' ))
}

orient <- function( model_index ){
  switch( model_index
    , x <- 1; y <- 2; z <- 3  # 1
    , x <- 2; y <- 1; z <- 3  # 2
    , x <- 3; y <- 2; z <- 1  # 3
  )
  list( x = x, y = y, z = z )
}

show_radii <- function( idx, ctr, crv, fixed_coordinate ){
  xyz <- orient( idx )

  c( 20, 80 ) %>%
    walk(
      ~sketch_radius(
          xyz  = xyz
        , fixed_coordinate = fixed_coordinate
```

```
        , ctr  = ctr
        , data = crv
        , i    = .x * 0.01 * nrow( crv )
      )
    )
}

show_arc <- function( idx, ctr, crv, fixed_coordinate, r ){
  xyz <- orient( idx )
  seq( 0, 100, 10 ) %>% # send sketch_arc() a series of X coordinates as data[ i ]
    map(
      ~sketch_arc(
          xyz  = xyz
        , fixed_coordinate = fixed_coordinate
        , ctr  = ctr
        , arc_data = crv
        , i    = .x * 0.01 * nrow( crv )
        , r    = r
      )
    ) -> arc_points
  arc_points
}

show_center <- function( x, y, z, est_radius, flavor='point' ){
  if( flavor %in% 'point' ){
    # Show center on the slice
    rgl.points(
        x     = x
      , y     = y
      , z     = z
      , r     = est_radius
      , color = 'red'
      , size  = 10
    )
  } else {  # 'sphere'
    # Show center on the slice
    rgl.spheres(
        x     = x
      , y     = y
      , z     = z
      , r     = est_radius
      , color = 'red'
      , alpha = 0.1               # 0: transparent ... 1: opaque
    )
```

```
  }
}

get_xy <- function( models=models, model_index=model_index, N=N ){

    # Draw a ray, perpendicular to the perimeter at points A and B
    # This should be two radii ( if they are directed inward )

    # ??? What if this is a complete circle, instead of just an arc?
    # Pick two points, A, and B, along the perimeter

    # For the current model, determine which axis is razor-thin,
    # and which axis has the largest extent.
    axs       <- curve_axes( mdls = models, idx = model_index )
    max_x     <- models[[ model_index ]]$calculate( f = max )[ axs$cx ]
    min_x     <- models[[ model_index ]]$calculate( f = min )[ axs$cx ]

    # Create data for a 2D curve representing this model's slice
    curve_x   <- seq( min_x, max_x, length.out=N )
    curve_y   <- map_dbl( curve_x, ~get_curve( x = .x, idx = model_index, fun='min', ax = axs ))
    curve_xy  <- data.frame( x=curve_x, y=curve_y )
    curve_xy[ is.finite( curve_xy$y ), ]
    data.frame( x=curve_xy$x, y=predict( loess( y ~ x, curve_xy, span=0.50 )) )
  }

show_slice <- function( mdls, idx ) switch(
    idx
  ,  mdls[[ idx ]]$show( LEFT_VIEW  ) # 1
  ,  mdls[[ idx ]]$show( TOP_VIEW   ) # 2
  ,  mdls[[ idx ]]$show( FRONT_VIEW ) # 3
)

plot_limits <- function( f ) sort(
  c(
      apply( f, 2, max ) %>% max()
    , apply( f, 2, min ) %>% min()
  )
)

slice_advice <- function( mdls, ax=X_AXIS ){
  best  <- best_slice( mdls[[ ax ]]$get(), ax )
  cat(
    sprintf(
      '\nThe tallest cross-section is at %s = %f'
```

```
      , toupper( axes[ ax ] )
      , round( best, 1 )
    )
  )
  mdls[[ ax ]]$get_band( ax = ax, ctr = best, thickness =  1.8 * STRIPE_WIDTH )
}

slice_the_slice <- function( mdls, ax=X_AXIS ){
  model_data        <- mdls[[ ax ]]$get()
  histogram_buckets <- model_data[ , ax ] %>% hist( plot = FALSE )
  best_mid          <- as.list( histogram_buckets )[[ 'mids' ]][ which.max( histogram_buckets$counts ) ]
  as.data.frame( get_band( model_data, ax, best_mid ))
}

micro_slice <- function( base_name, mdls, ax ){
  slice_advice( mdls, ax )
  show_multi_view( paste0( base_name, axes[ ax ] ), mdls[[ ax ]] )
  thin_slice <- slice_the_slice( mdls, ax )
  lims <- plot_limits( thin_slice)
  # Show the resulting profile
  fn <- paste0( FIGURES_PATH, base_name, axes[ ax ], 'thin.png' )
  cat( sprintf( '\nPlotting %s\n', fn ))
  keep_axes <- axes[ -ax ]
  # _____ Begin Plotting _____
  png( fn )
    print({
      ggplot( thin_slice, aes_string( x = keep_axes[ 1 ], y = keep_axes[ 2 ] )) +
        geom_point()   +
        xlim( lims ) +
        ylim( lims )
    })
  dev.off()
 # _____ End Plotting _____
  thin_slice
}
```