darkredrgb0.26,0.23,0.23 codegreenrgb0,0.6,0 purplergb0.65, 0.12, 0.82

JavaScript keywords=typeof, new, true, false, catch, function, return, null, catch, switch, var, if, in, while, do, else, case, break, keywordstyle=**, ndkeywords=class, export, boolean, throw, implements, import, this, ndkeywordstyle=darkgray, identifierstyle=black, sensitive=false, comment=[l]//, morecomment=[s]/**/, commentstyle=purple,**

basicstyle=, `backgroundcolor=white, breakatwhitespace=false, breaklines=true, captionpos=b, commentstyle=codegreen, frame=tb, keepspaces=true, rulecolor=black, showspaces=false, showstringspaces=false, showtabs=false, tabsize=2,`

Chapter 1

Appendix

1.1 persp.R

[language = R] initialize and create a viewport prepare for drawing perInit = function (plot, newpage = FALSE, dbox = TRUE) info = plot [[1]] is the all the grapical information that transfer into grid [[3]] is the persp call information [[2]] is the plot details eg: x, y, z, xlim, ylim, zlim, col ... create a list that store all information from the persp then pass the information to per for drawing. x is [[2]]; y is [[3]]; z is [[4]] xr is [[5]]; yr is [[6]]; zr is [[7]] col is [[14]]; border is [[15]]; box is [[19]] axes is [[20]], nTicks is [[21]] tickType is [[22]] xlab/ylab/zlab = [[23]]/[[24]]/[[25]] main is in plot[[1]][[4]][[2]][[2]] shade is 0.8, ltheta/lphi = [[16]]/[[17]] expand is [[13]], scale is [[12]] out = list(x = info[[2]], y = info[[3]], z = info[[4]], xr = info[[5]], yr = info[[6]], zr = info[[7]], col = info[[14]], border = info[[15]][1] only allows one color for border , dbox = info[[19]], newpage = newpage, phi = info[[9]], theta = info[[8]], r = info[[10]], d = info[[11]], axes = info[[20]], nTicks = info[[21]], tickType = info[[22]], xlab = info[[23]], ylab = info[[24]], zlab = info[[25]], parameters in 'par' that need added to per lwd = info$lwd, lty = info$lty, col.axis = info$col.axis, col.lab = info$col.lab, cex.lab = info$cex.lab, shade = info[[18]], ltheta = info[[16]], lphi = info[[17]], expand = info[[13]], scale = info[[12]] main = plot[[1]][[4]][[2]][[2]])
if(out$newpage == TRUE$) grid.newpage()out
main call C$_p$ersp = function(plot = NULL, ...)dev.set(recordDev())par = currentPar(NULL)dev.set(pl
information extraction xc = yc = zc = xs = ys = zs = 0 plot = perInit(plot,

newpage = FALSE) xr = plot$xr; yr = plot$yr; zr = plot$zr xlab = plot$xlab; ylab = plot$ylab; zlab = plot$zlab col.axis = plot$col.axis; col.lab = plot$col.lab; col = plot$col; cex.lab = plot$cex.lab nTicks = plot$nTicks; tickType = plot$tickType expand = plot$expand; scale = plot$scale ltheta = plot$ltheta; lphi = plot$lphi main = plot$main; axes = plot$axes dbox = plot$dbox; shade = plot$shade r = plot$r; d = plot$d; phi = plot$phi; theta = plot$theta

xs = LimitCheck(xr)[1] ys = LimitCheck(yr)[1] zs = LimitCheck(zr)[1] xc = LimitCheck(xr)[2] yc = LimitCheck(yr)[2] zc = LimitCheck(zr)[2]

if(scale == FALSE) s = xs if(s ¡ ys) s = ys if (s ¡ zs) s = zs xs = s ys = s zs = s

VT = diag(1, 4) VT = VT VT = VT VT = VT VT = VT VT = VT VT = VT VT = VT trans = VT

border = plot$border[1]; if(is.null(plot$lwd)) lwd = 1 else lwd = plot$lwd if(is.null(plot$lty)) lty = 1 else lty = plot$lty if(any(!(is.numeric(xr) is.numeric(yr) is.numeric(zr)))) stop("invalid limits") if(

if(!scale) xs = ys = zs = max(xs, ys, zs) colCheck = col2rgb(col, alpha = TRUE)[4,1] == 255 if(is.finite(ltheta) is.finite(lphi) is.finite(shade) colCheck) DoLighting = TRUE else DoLighting = FALSE check the first color act as Fixcols

if (DoLighting) Light = SetUpLight(ltheta, lphi)

create a viewport inside a 'viewport' depth = gotovp(FALSE) lim = PerspWindow(xr, yr, zr, trans, 'r') vp = viewport(0.5, 0.5, 1, 1, default.units = 'npc', xscale = lim[1:2], yscale = lim[3:4]) upViewport(depth)

incrementWindowAlpha() setWindowPlotAlpha(plotAlpha()) setUpUsr(lim)

if (dbox == TRUE) EdgeDone = rep(0, 12) if(axes == TRUE) depth = gotovp(TRUE) PerspAxes(xr, yr, zr, x, y, z xlab, ylab, zlab, xlab, xenc, ylab, yenc, zlab, zenc nTicks, tickType, trans, nTicks, tickType, VT lwd, lty, col.axis, col.lab, cex.lab) lwd, lty, col.axis, col.lab, cex.lab upViewport(depth) else EdgeDone = rep(1, 12) xr = yr = zr = c(0,0)

draw the behind face first return the EdgeDone inorder to not drawing the same Edege two times. depth = gotovp(TRUE) EdgeDone = PerspBox(0, xr, yr, zr, EdgeDone, trans, 1, lwd) upViewport(depth)

depth = gotovp(FALSE) DrawFacets(plot = plot, z = plot$z, x = plot$x, y = plot$y, basicxs = 1/xs, ys = 1/ys, zs = expand/zs, Lightcol = col, colsltheta = ltheta, lphi = lphi, Shade = shade, Light = Light, trans = trans, DoLighting = DoLighting) upViewport(depth)

depth = gotovp(TRUE) EdgeDone = PerspBox(1, xr, yr, zr, EdgeDone, trans, 'dotted', lwd) upViewport(depth)
Shade function LimitCheck = function (lim) not finished yet... s = 0.5 * abs(lim[2] - lim[1]) c = 0.5 * (lim[2] + lim[1]) c(s, c)
XRotate = function (angle) TT = diag(1, 4) rad = angle * pi / 180 c = cos(rad) s = sin(rad) TT[2, 2] = c; TT[3, 2] = -s; TT[3, 3] = c; TT[2, 3] = s; TT
YRotate = function (angle) TT = diag(1, 4) rad = angle * pi / 180 c = cos(rad) s = sin(rad) TT[1, 1] = c; TT[3, 1] = s; TT[3, 3] = c; TT[1, 3] = -s; TT
ZRotate = function (angle) TT = diag(1, 4) rad = angle * pi / 180 c = cos(rad) s = sin(rad) TT[1, 1] = c; TT[2, 1] = -s; TT[2, 2] = c; TT[1, 2] = s; TT
Translate = function(x, y, z) TT = diag(1,4) TT[4, 1] = x TT[4, 2] = y TT[4, 3] = z TT
Scale = function(x, y, z) TT = diag(1,4) TT[1, 1] = x TT[2, 2] = y TT[3, 3] = z TT
Perspective = function(d) TT = diag(1,4) TT[3, 4] = -1 / d TT
SetUpLight = function (theta, phi) u = c(0, -1, 0, 1) VT = diag(1, 4) VT = VT VT = VT Light = u
FacetShade = function(u, v, Shade, Light) nx = u[2] * v[3] - u[3] * v[2] ny = u[3] * v[1] - u[1] * v[3] nz = u[1] * v[2] - u[2] * v[1] sum = sqrt(nx * nx + ny * ny + nz * nz) if (is.finite(sum)) if (sum == 0) sum = 1 elseShade = NA
nx = nx/sum ny = ny/sum nz = nz/sum sum = 0.5 * (nx * Light[1] + ny * Light[2] + nz * Light[3] + 1) sum$^{S}hade$
shadeCol = function(z, x, y, xs, ys, zs, col, ltheta, lphi, Shade, Light) u = v = 0 shade = 0 nx = nrow(z) ny = ncol(z) nx1 = nx - 1 ny1 = ny - 1 cols = 0 ncol = length(col) indx = 0:(length(z)) Light = SetUpLight(ltheta, lphi) for(k in 1:(nx1 * ny1)) nv = 0 i = (indx[k]) j = (indx[k]) icol = (i + j * nx1)
u[1] = xs * (x[i + 2] - x[i + 1]) u[2] = ys * (y[j + 1] - y[j + 2]) u[3] = zs * (z[(i + 1)+ j * nx + 1] - z[i + (j + 1) * nx + 1]) v[1] = xs * (x[i + 2] - x[i + 1]) v[2] = ys * (y[j + 2] - y[j + 1]) v[3] = zs * (z[(i + 1) + (j + 1) * nx + 1] - z[i + j * nx + 1]) icol = (i + j * nx1) shade[k] = FacetShade(u, v, Shade = Shade, Light = Light)
shadedCol = col2rgb(col[icol + 1], alpha = TRUE) if(is.finite(shade[k])) cols[k]

= rgb(shade[k] * shadedCol[1], shade[k] * shadedCol[2], shade[k] * shadedCol[3], maxColorValue = 255) else cols[k] = rgb(1,1,1,0) list(cols = cols, shade = shade)

shade end... PerspBox = function(front = 1, x, y, z, EdgeDone, VT, lty, lwd = lwd) u0 = u1 = u2 = u3 = 0 v0 = v1 = v2 = v3 = 0 for (f in 1:6) p0 = Face[f, 1] p1 = Face[f, 2] p2 = Face[f, 3] p3 = Face[f, 4]

u0[1] = x[Vertex[p0, 1]] u0[2] = y[Vertex[p0, 2]] u0[3] = z[Vertex[p0, 3]] u0[4] = 1 u1[1] = x[Vertex[p1, 1]] u1[2] = y[Vertex[p1, 2]] u1[3] = z[Vertex[p1, 3]] u1[4] = 1 u2[1] = x[Vertex[p2, 1]] u2[2] = y[Vertex[p2, 2]] u2[3] = z[Vertex[p2, 3]] u2[4] = 1 u3[1] = x[Vertex[p3, 1]] u3[2] = y[Vertex[p3, 2]] u3[3] = z[Vertex[p3, 3]] u3[4] = 1

v0 = TransVector(u0, VT) v1 = TransVector(u1, VT) v2 = TransVector(u2, VT) v3 = TransVector(u3, VT)

v0 = v0/v0[4] v1 = v1/v1[4] v2 = v2/v2[4] v3 = v3/v3[4]

d = v1 - v0 e = v2 - v1

nearby = (d[1]*e[2] - d[2]*e[1]) ¡ 0

draw the face line by line rather than polygon if ((front nearby) —— (!front !nearby)) if (!EdgeDone[Edge[f, 1]]) grid.lines(c(v0[1], v1[1]), c(v0[2], v1[2]), default.units = 'native', gp = gpar(lty = lty, lwd = lwd)) EdgeDone[Edge[f, 1]] = EdgeDone[Edge[f, 1]] + 1 if (!EdgeDone[Edge[f, 2]]) grid.lines(c(v1[1], v2[1]), c(v1[2], v2[2]), default.units = 'native', gp = gpar(lty = lty, lwd = lwd)) EdgeDone[Edge[f, 2]] = EdgeDone[Edge[f, 2]] + 1 if (!EdgeDone[Edge[f, 3]]) grid.lines(c(v2[1], v3[1]), c(v2[2], v3[2]), default.units = 'native', gp = gpar(lty = lty, lwd = lwd)) EdgeDone[Edge[f, 3]] = EdgeDone[Edge[f, 3]] + 1 if (!EdgeDone[Edge[f, 4]]) grid.lines(c(v3[1], v0[1]), c(v3[2], v0[2]), default.units = 'native', gp = gpar(lty = lty, lwd = lwd)) EdgeDone[Edge[f, 4]] = EdgeDone[Edge[f, 4]] + 1 EdgeDone

dPolygon = function(x, y, z, col, trans)

the total number of polygon that we need to draw nx = length(x) ny = length(y) total = nx * ny stops = (nx - 1) * (ny - 1)

set the temp value for x,y,z prepare for subsetting xTmp = rep(x, length(y)) yTmp = rep(y,each = nx) zTmp = as.numeric(z)

the drawing order is along x-axis, and then along y-axis then create a vector like a 4Xn matrix, i.e the first column contain all the first points for every polygons the second column contain all the second points for every polygons

and so on pBreak = c(1:total, 1 + 1:total, 1 + nx + 1:total, nx + 1:total) xBreak = xTmp[pBreak] yBreak = yTmp[pBreak] zBreak = zTmp[pBreak] draw the box if required the vectors now has four paths, every paths contain the information of every points of every polygon now we need to change the order of this vector, so that the first four index should be the order for drawing the first points, not the first four points for the first four polygon points subsetting plot.index = rep(c(1, 1 + total, 1 + 2 * total, 1 + 3 * total), total) + rep(0:(total - 1), each = 4) sequence for 'problem's polygons index, e.g along x-axis, there are n-1 polygons, n is the number of points in x direction we don't want to draw the nth polygon, hence we deleted those polygon dp = rep((4 * seq(nx,total,nx)), each = 4) - (3:0) final subsetting xCoor = xBreak[plot.index][-dp][1 : (4 * stops)] yCoor = yBreak[plot.index][-dp][1 : (4 * stops)] zCoor = zBreak[plot.index][-dp][1 : (4 * stops)] vectorize the cols colRep = rep$_l en(col, length(xCoor))$ use the first corner of every polygon to determind the order for drawing corn.id = 4* 1:(length(xCoor)/4) xc = xCoor[corn.id] yc = yCoor[corn.id] method for using the zdepth for changing the drawing order for every polygon orderTemp = cbind(xc, yc, 0, 1) zdepth = orderTemp[, 4] the zdepth of a set of 4 points of each polygon a = order(zdepth, decreasing = TRUE) oo = rep(1:4, length(a)) + rep(a - 1, each = 4) * 4 xyCoor = trans3d(xCoor[oo], yCoor[oo], zCoor[oo], trans) colRep = colRep[a] record the total number of polygon pMax = length(xyCoor$x$)/4$pout = list(xyCoor = xyCoor, pMax = pMax, colRep = colRep, polygonOrder = a)pout$ DrawFacets = function(plot, z, x, y, xs, ys, zs, col, ltheta, lphi, Shade, Light, trans, DoLighting) pout = dPolygon(x, y, z, col, trans) xyCoor = pout$xyCoorpMax =$ *pout*pMax; colRep = pout$colReppolygonOrder =$ *pout*polygonOrder polygons = cbind(xyCoor$x, xyCoor$y) polygon.id = rep(1:pMax, each = 4) col = plotcol if (DoLighting == TRUE) col[is.na(col)] = rgb(1, 1, 1) if(is.finite(Shade) Shade ¡= 0) Shade = 1 shadding = shadeCol(z, x, y, x, y, z xs, ys, zs, xs, ys, zs col, col, ncol ltheta, lphi, Shade, Light = Light) ltheta, lphi, Shade(not shade) shadedCol = shadding[[1]] clean if any NA's Z-value shade = shadding[[2]][polygonOrder] misshade = !is.finite(shade) misindex = rep(misshade, each = 4) polygonOrder = poly-

gonOrder[!misshade] polygons = polygons[!misindex,] polygon.id = polygon.id[!misindex]
cols = shadedCol[polygonOrder] else cols = rep$_len(col, length(polygons[,1]))[polygonOrder]$
xrange = range(polygons[,1], na.rm = TRUE) yrange = range(polygons[,2],
na.rm = TRUE)
grid.polygon(polygons[,1], polygons[,2], id = polygon.id, default.units = 'native', gp = gpar(col = plot$border, fill = cols, lty = plot$lty, lwd = plot$lwd))
TransVector = function(u, T) u
lowest = function (y1, y2, y3, y4) (y1 ¡= y2) (y1 ¡= y3) (y1 ¡= y4)
labelAngle = function(x1, y1, x2, y2)
dx = abs(x2 - x1) if (x2 ¿ x1) dy = y2 - y1 else dy = y1 - y2
if (dx == 0) if(dy ¿ 0) angle = 90 else angle = 270 else angle = 180/pi
* atan2(dy, dx) angle
PerspAxis = function(x, y, z, axis, axisType, nTicks, tickType, label, VT, lwd
= 1, lty, col.axis = 1, col.lab = 1, cex.lab = 1)
don't know how to use numeric on the switch... axisType = as.character(axisType)
tickType = as.character(tickType) u1 = u2 = u3 = c(0.,0.,0.,0.) tickLength =
.03
switch(axisType, '1' = min = x[1]; max = x[2]; range = x, '2' = min = y[1];
max = y[2]; range = y, '3' = min = z[1]; max = z[2]; range = z)
d$_frac = 0.1 * (max - min) nint = nTicks - 1$
if(!nint)nint = nint + 1 i = nint
ticks = axisTicks(c(min, max), FALSE, nint = nint) min = ticks[1] max =
ticks[length(ticks)] nint = length(ticks) - 1
but maybe not this one... haven't test yet... while((min ¡ range[1] - d$_frac||range[2]+$
$d_frac < max)i < 20)nint = i + 1ticks = axisTicks(c(min, max), FALSE)range = range(ticks)nint = le$
axp seems working... axp = 0 axp[1] = min axp[2] = max axp[3] = nint
Do the following calculations for both ticktypes Vertex is a 8*3 matrix; i.e. the
vertex of a box AxisStart is a vector of length 8 axis is a output u1, u2 are the
vectors in 3-d the range of x,y,z switch (axisType, '1' = u1[1] = min u1[2] =
y[Vertex[AxisStart[axis], 2]] u1[3] = z[Vertex[AxisStart[axis], 3]] , '2' = u1[1] =
x[Vertex[AxisStart[axis], 1]] u1[2] = min u1[3] = z[Vertex[AxisStart[axis], 3]] , '3'
= u1[1] = x[Vertex[AxisStart[axis], 1]] u1[2] = y[Vertex[AxisStart[axis], 2]] u1[3]
= min) u1[1] = u1[1] + tickLength*(x[2]-x[1])*TickVector[axis, 1] u1[2] = u1[2]
+ tickLength*(y[2]-y[1])*TickVector[axis, 2] u1[3] = u1[3] + tickLength*(z[2]-
z[1])*TickVector[axis, 3] u1[4] = 1

axisType, 1 = 'draw x-axis' 2 = 'draw y-axis' 3 = 'draw z-axis' switch (axisType, '1' = u2[1] = max u2[2] = u1[2] u2[3] = u1[3] , '2' = u2[1] = u1[1] u2[2] = max u2[3] = u1[3] , '3' = u2[1] = u1[1] u2[2] = u1[2] u2[3] = max) u2[4] = 1

switch(tickType, '1' = u3[1] = u1[1] + tickLength*(x[2]-x[1])*TickVector[axis, 1] u3[2] = u1[2] + tickLength*(y[2]-y[1])*TickVector[axis, 2] u3[3] = u1[3] + tickLength*(z[2]-z[1])*TickVector[axis, 3] , '2' = u3[1] = u1[1] + 2.5*tickLength*(x[2]-x[1])*TickVector[axis, 1] u3[2] = u1[2] + 2.5*tickLength*(y[2]-y[1])*TickVector[axis, 2] u3[3] = u1[3] + 2.5*tickLength*(z[2]-z[1])*TickVector[axis, 3])

u3 is the the labels at the center of each axes switch(axisType, '1' = u3[1] = (min + max)/2 , '2' = u3[2] = (min + max)/2 , '3' = u3[3] = (min + max)/2) u3[4] = 1

transform the 3-d into 2-d v1 = TransVector(u1, VT) v2 = TransVector(u2, VT) v3 = TransVector(u3, VT)

v1 = v1/v1[4] v2 = v2/v2[4] v3 = v3/v3[4]

label at center of each axes srt = labelAngle(v1[1], v1[2], v2[1], v2[2]) text(v3[1], v3[2], label, 0.5, srt = srt) grid.text(label = label, x = v3[1], y = v3[2], just = "centre", rot = srt, default.units = "native", vp = 'clipoff', gp = gpar(col = col.lab, lwd = lwd, cex = cex.lab))

tickType is not working.. when = '2' switch(tickType, '1' = arrow = arrow(angle = 10, length = unit(0.1, "in"), ends = "last", type = "open") drawing the tick..

grid.lines(x = c(v1[1], v2[1]), y = c(v1[2], v2[2]), default.units = "native", arrow = arrow, vp = 'clipoff', gp = gpar(col = 1, lwd = lwd , lty = lty)) , '2' seems working '2' = at = axisTicks(range, FALSE, axp, nint = nint) lab = format(at, trim = TRUE) for(i in 1:length(at)) switch(axisType, '1' = u1[1] = at[i] u1[2] = y[Vertex[AxisStart[axis], 2]] u1[3] = z[Vertex[AxisStart[axis], 3]] , '2' = u1[1] = x[Vertex[AxisStart[axis], 1]] u1[2] = at[i] u1[3] = z[Vertex[AxisStart[axis], 3]] , '3' = u1[1] = x[Vertex[AxisStart[axis], 1]] u1[2] = y[Vertex[AxisStart[axis], 2]] u1[3] = at[i])

tickLength = 0.03

u1[4] = 1 u2[1] = u1[1] + tickLength*(x[2]-x[1])*TickVector[axis, 1] u2[2] = u1[2] + tickLength*(y[2]-y[1])*TickVector[axis, 2] u2[3] = u1[3] + tickLength*(z[2]-z[1])*TickVector[axis, 3] u2[4] = 1 u3[1] = u2[1] + tickLength*(x[2]-x[1])*TickVector[axis, 1] u3[2] = u2[2] + tickLength*(y[2]-y[1])*TickVector[axis, 2] u3[3] = u2[3] +

tickLength*(z[2]-z[1])*TickVector[axis, 3] u3[4] = 1 v1 = TransVector(u1, VT) v2 = TransVector(u2, VT) v3 = TransVector(u3, VT)
v1 = v1/v1[4] v2 = v2/v2[4] v3 = v3/v3[4]
Draw tick line grid.lines(x = c(v1[1], v2[1]), y = c(v1[2], v2[2]), default.units = "native", vp = 'clipoff', gp = gpar(col = col.axis, lwd = lwd, lty = lty))
Draw tick label grid.text(label = lab[i], x = v3[1], y = v3[2], just = "centre", default.units = "native", vp = 'clipoff', gp = gpar(col = col.axis, adj = 1, pos = 0.5, cex = 1)))
PerspAxes = function(x, y, z, xlab, ylab, zlab, nTicks, tickType, VT, parameters in par lwd = 1, lty = 1, col.axis = 1, col.lab = 1, cex.lab = 1) xAxis = yAxis = zAxis = 0 -Wall u0 = u1 = u2 = u3 = 0
u0[1] = x[1]; u0[2] = y[1]; u0[3] = z[1]; u0[4] = 1 u1[1] = x[2]; u1[2] = y[1]; u1[3] = z[1]; u1[4] = 1 u2[1] = x[1]; u2[2] = y[2]; u2[3] = z[1]; u2[4] = 1 u3[1] = x[2]; u3[2] = y[2]; u3[3] = z[1]; u3[4] = 1
v0 = TransVector(u0, VT) v1 = TransVector(u1, VT) v2 = TransVector(u2, VT) v3 = TransVector(u3, VT)
v0 = v0/v0[4] v1 = v1/v1[4] v2 = v2/v2[4] v3 = v3/v3[4]
if (lowest(v0[2], v1[2], v2[2], v3[2])) xAxis = 1 yAxis = 2 else if (lowest(v1[2], v0[2], v2[2], v3[2])) xAxis = 1 yAxis = 4 else if (lowest(v2[2], v1[2], v0[2], v3[2])) xAxis = 3 yAxis = 2 else if (lowest(v3[2], v1[2], v2[2], v0[2])) xAxis = 3 yAxis = 4 else warning("Axis orientation not calculated") drawing x and y axes PerspAxis(x, y, z, xAxis, '1', nTicks, tickType, xlab, VT, lwd = lwd, lty = lty, col.axis = col.axis, col.lab = col.lab, cex.lab = cex.lab)
PerspAxis(x, y, z, yAxis, '2', nTicks, tickType, ylab, VT, lwd = lwd, lty = lty, col.axis = col.axis, col.lab = col.lab, cex.lab = cex.lab)
Figure out which Z axis to draw if (lowest(v0[1], v1[1], v2[1], v3[1])) zAxis = 5 else if (lowest(v1[1], v0[1], v2[1], v3[1])) zAxis = 6 else if (lowest(v2[1], v1[1], v0[1], v3[1])) zAxis = 7 else if (lowest(v3[1], v1[1], v2[1], v0[1])) zAxis = 8 else warning("Axis orientation not calculated")
drawing the z-axis PerspAxis(x, y, z, zAxis, '3', nTicks, tickType, zlab, VT, lwd = lwd, lty = lty, col.axis = col.axis, col.lab = col.lab, cex.lab = cex.lab)
PerspWindow = function(xlim, ylim, zlim, VT, style) xmax = xmin = ymax = ymin = u = 0 u[4] = 1 for (i in 1:2) u[1] = xlim[i] for (j in 1:2) u[2] = ylim[j] for (k in 1:2) u[3] = zlim[k] v = TransVector(u, VT) xx = v[1] / v[4] yy = v[2] / v[4] if (xx ¿ xmax) xmax = xx if (xx ¡ xmin) xmin = xx if (yy ¿ ymax) ymax

= yy if (yy ¡ ymin) ymin = yy pin1 = convertX(unit(1.0, 'npc'), 'inches', valueOnly = TRUE) pin2 = convertY(unit(1.0, 'npc'), 'inches', valueOnly = TRUE) xdelta = abs(xmax - xmin) ydelta = abs(ymax - ymin) xscale = pin1 / xdelta yscale = pin2 / ydelta scale = if(xscale ¡ yscale) xscale else yscale xadd = .5 * (pin1 / scale - xdelta); yadd = .5 * (pin2 / scale - ydelta); GScale in C xrange = GScale(xmin - xadd, xmax + xadd, style) yrange = GScale(ymin - yadd, ymax + yadd, style) c(xrange, yrange)

GScale = function(min, max, style) switch(style, 'r' = temp = 0.04 * (max - min) min = min - temp max = max + temp , 'i' =) c(min, max)

global variables. TickVector = matrix(ncol = 3, byrow = TRUE, data = c(0, -1, -1, -1, 0, -1, 0, 1, -1, 1, 0, -1, -1, -1, 0, 1, -1, 0, -1, 1, 0, 1, 1, 0))

Vertex = matrix(ncol = 3, byrow = TRUE, data = c(1, 1, 1, xlim[1], ylim[1], zlim[1] 1, 1, 2, xlim[1], ylim[1], zlim[2] 1, 2, 1, 1, 2, 2, 2, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 2))

Face = matrix (ncol = 4, byrow = TRUE, data = c(1, 2, 6, 5, 3, 7, 8, 4, 1, 3, 4, 2, 5, 6, 8, 7, 1, 5, 7, 3, 2, 4, 8, 6))

Edge = matrix (ncol = 4, byrow = TRUE, data = c(0, 1, 2, 3, 4, 5, 6, 7, 8, 7, 9, 0, 2,10, 5,11, 3,11, 4, 8, 9, 6,10, 1)) + 1

AxisStart = c(1, 1, 3, 5, 1, 5, 3, 7)

1.2 filled.contour.R

[language = R] vectorization version (main in used) FindPolygonVertices = function(low, high, x1, x2, y1, y2, z11, z21, z12, z22, colrep)
v1 = FindCutPoints(low, high, x1, y1, x2, y1, z11, z21) v2 = FindCutPoints(low, high, y1, x2, y2, x2, z21, z22) v3 = FindCutPoints(low, high, x2, y2, x1, y2, z22, z12) v4 = FindCutPoints(low, high, y2, x1, y1, x1, z12, z11)
vx = cbind(v1[[1]], v2[[2]], v3[[1]], v4[[2]]) vy = cbind(v1[[2]], v2[[1]], v3[[2]], v4[[1]])
track the coordinate for x and y(if non-NA's) index = rowSums(!is.na(vx)) keep if non-NAs row ¿= 2 (npt ¿= 2) vx = t(vx) vy = t(vy) xcoor.na = as.vector(vx[, index ¿ 2]) ycoor.na = as.vector(vy[, index ¿ 2]) delete all NA's, xcoor = xcoor.na[!is.na(xcoor.na)] ycoor = ycoor.na[!is.na(ycoor.na)]
id.length = index[index ¿ 2] cols = colrep[index ¿ 2]
out = list(x = xcoor, y = ycoor, id.length = id.length, cols = cols) outs = out out
FindCutPoints = function(low, high, x1, y1, x2, y2, z1, z2) inner condiction begin first ocndiction c = (z1 - high) / (z1 - z2) cond1 = z1 ¡ high cond2 = z1 == Inf cond3 = z2 ¿ high — z1 ¡ low
x.1 = ifelse(cond1, x1, ifelse(cond2, x2, x1 + c * (x2 - x1))) x.1 = ifelse(cond3, NA, x.1)
y.1 = ifelse(cond1, y1, ifelse(cond2, y1, y1)) y.1 = ifelse(cond3, NA, y.1)
cond4 = z2 == -Inf cond5 = z2 ¡= low cond6 = z2 ¿ high — z1 ¡ low
c = (z2 -low) / (z2 - z1) x.2 = ifelse(cond4, x1, ifelse(cond5, x2 - c * (x2 - x1), NA)) x.2 = ifelse(cond6, NA, x.2)
y.2 = ifelse(cond4, y1, ifelse(cond5, y1, NA)) y.2 = ifelse(cond6, NA, y.2)
second condiction cond7 = z1 ¿ low cond8 = z1 == -Inf cond9 = z2 ¡ low — z1 ¿ high
c = (z1 - low) / (z1 - z2) $x_1 = ifelse(cond7, x1, ifelse(cond8, x2, x1 + c * (x2 - x1)))x_1 = ifelse(cond9, NA, x_1)$
$y_1 = ifelse(cond7, y1, ifelse(cond8, y1, y1))y_1 = ifelse(cond9, NA, y_1)$
cond10 = z2 ¡ high cond11 = z2 == Inf cond12 = z2 ¡ low — z1 ¿ high
c = (z2 - high) / (z2 - z1) $x_2 = ifelse(cond10, NA, ifelse(cond11, x1, x2 - c * (x2 - x1)))x_2 = ifelse(cond12, NA, x_2)$
$y_2 = ifelse(cond10, NA, ifelse(cond11, y1, y1))y_2 = ifelse(cond12, NA, y_2)$

third condiction cond13 = low ¡= z1 z1 ¡= high x..1 = ifelse(cond13, x1, NA)
y..1 = ifelse(cond13, y1, NA) inner condiction end
outer condiction cond14 = z1 ¿ z2 cond15 = z1 ¡ z2
xout.1 = ifelse(cond14, x.1, ifelse(cond15, $x_1, x..1$))$xout.2 = ifelse(cond14, x.2, ifelse(cond15, x_2, NA))$
yout.1 = ifelse(cond14, y.1, ifelse(cond15, $y_1, y..1$))$yout.2 = ifelse(cond14, y.2, ifelse(cond15, y_2, NA))oute$
return x1, x2, y1, y2 xout = cbind(xout.1, xout.2) yout = cbind(yout.1, yout.2)
list(xout, yout)
$C_filledcontour = function(plot)dev.set(recordDev())par = currentPar(NULL)dev.set(playDev())$
x = plot[[2]] y = plot[[3]] z = plot[[4]] s = plot[[5]] cols = plot[[6]]
ns = length(s) nx = length(x) ny = length(y)
x1 = rep(x[-nx], each = ny - 1) x2 = rep(x[-1], each = ny - 1) y1 = rep(y[-ny],
nx - 1) y2 = rep(y[-1], nx - 1)
z11 = as.numeric(t(z[-nx, -ny])) z21 = as.numeric(t(z[-1, -ny])) z12 = as.numeric(t(z[-
nx, -1])) z22 = as.numeric(t(z[-1, -1]))
x1 = rep(x1, each = ns - 1) x2 = rep(x2, each = ns - 1) y1 = rep(y1, each = ns
- 1) y2 = rep(y2, each = ns - 1) z11 = rep(z11, each = ns - 1) z12 = rep(z12,
each = ns - 1) z21 = rep(z21, each = ns - 1) z22 = rep(z22, each = ns - 1) low
= rep(s[-ns], (nx - 1) * (ny - 1)) high = rep(s[-1], (nx - 1) * (ny - 1))
rep color until the same length of x, then subsetting if(length(cols) ¿ ns) cols =
cols[1:(ns - 1)] else cols = rep$_len(cols, ns-1)colrep = rep(cols[1 : (ns-1)], nx*$
$ny)feedcoloraswellassubseetingasxandyout = FindPolygonVertices(low =$
$low, high = high, x1 = x1, x2 = x2, y1 = y1, y2 = y2, z11 = z11, z21 =$
$z21, z12 = z12, z22 = z22, colrep = colrep)actualdrawingdepth = gotovp(TRUE)grid.polygon(out$x,
out$y, default.units =' native', id.lengths = out$id.length, gp = gpar(fill =
out$cols, col = NA))upViewport(depth)$
for loop version identical to $C_filledcontourinplot3d.cbutveryslowlFindPolygonVertices =$
$function(low, high, x1, x2, y1, y2, z11, z21, z12, z22, x, y, z, npt)out = list()npt = 0out1 = lFindCutPoints$
y = out1$y; z = out1$z; npt = out1npt
out2 = lFindCutPoints(low, high, y1, x2, z21, y2, x2, z22, y, x, z, npt) x =
out2$x; y = out2$y; z = out2$z; npt = out2$npt$
out3 = lFindCutPoints(low, high, x2, y2, z22, x1, y2, z12, x, y, z, npt) x =
out3$x; y = out3$y; z = out3$z; npt = out3$npt$
out4 = lFindCutPoints(low, high, y2, x1, z12, y1, x1, z11, y, x, z, npt)
out$x = out1$x + out2$y + out3$x + out4$youty = out1$y + out2$x + out3$y + out4$x
out$npt = out4$npt out

lC$_f$*illedcontour* = *function*(*plot*)*dev.set*(*recordDev*())*par* = *currentPar*(*NULL*)*dev.set*(*playDev*())
x = plot[[2]] y = plot[[3]] z = plot[[4]] sc = plot[[5]] px = py = pz = numeric(8)
scol = plot[[6]]
nx = length(x) ny = length(y) if (nx ¡ 2 —— ny ¡ 2) stop("insufficient 'x' or 'y' values")
do it this way as coerceVector can lose dims, e.g. for a list matrix if (nrow(z) != nx —— ncol(z) != ny) stop("dimension mismatch")
nc = length(sc) if (nc ¡ 1) warning("no contour values")
ncol = length(scol)
depth = gotovp(TRUE) for(i in 1:(nx - 1)) for(j in 1:(ny - 1)) for(k in 1:(nc - 1)) npt = 0 out = lFindPolygonVertices(sc[k], sc[k + 1], x[i], x[i + 1], y[j], y[j + 1], z[i, j], z[i + 1, j], z[i, j + 1], z[i + 1, j + 1], px, py, pz, npt)
npt = out*npt*
if(npt ¿ 2) grid.polygon(out$x$[1 : *npt*], *out*y[1:npt], default.units = 'native', gp = gpar(fill = scol[(k - 1) upViewport(depth)
lFindCutPoints = function(low, high, x1, y1, z1, x2, y2, z2, x, y, z, npt) x = y = z = numeric(8) if (z1 ¿ z2) if (z2 ¿ high —— z1 ¡ low) return(out = list(x = x, y = y, z = z, npt = npt))
if (z1 ¡ high) x[npt + 1] = x1 y[npt + 1] = y1 z[npt + 1] = z1 npt = npt + 1 else if (z1 == Inf) x[npt + 1] = x2 y[npt + 1] = y1 z[npt + 1] = z2 npt = npt + 1 else c = (z1 - high) / (z1 - z2) x[npt + 1] = x1 + c * (x2 - x1) y[npt + 1] = y1 z[npt + 1] = z1 + c * (z2 - z1) npt = npt + 1
if (z2 == -Inf) x[npt + 1] = x1 y[npt + 1] = y1 z[npt + 1] = z1 npt = npt + 1 else if (z2 ¡= low) c = (z2 -low) / (z2 - z1) x[npt + 1] = x2 - c * (x2 - x1) y[npt + 1] = y1 z[npt + 1] = z2 - c * (z2 - z1) npt = npt + 1
else if (z1 ¡ z2) if (z2 ¡ low —— z1 ¿ high) return(out = list(x = x, y = y, z = z, npt = npt))
if (z1 ¿ low) x[npt + 1] = x1 y[npt + 1] = y1 z[npt + 1] = z1 npt = npt + 1 else if (z1 == -Inf) x[npt + 1] = x2 y[npt + 1] = y1 z[npt + 1] = z2 npt = npt + 1 else c = (z1 - low) / (z1 - z2) x[npt + 1] = x1 + c * (x2 - x1) y[npt + 1] = y1 z[npt + 1] = z1 + c * (z2 - z1) npt = npt + 1
if (z2 ¡ high) else if (z2 == Inf) x[npt + 1] = x1 y[npt + 1] = y1 z[npt + 1] = z1 npt = npt + 1 else c = (z2 - high) / (z2 - z1) x[npt + 1] = x2 - c * (x2 - x1) y[npt + 1] = y1 z[npt + 1] = z2 - c * (z2 - z1) npt = npt + 1 else if(low ¡= z1 z1 ¡= high) x[npt + 1] = x1 y[npt + 1] = y1 z[npt + 1] = z1 npt = npt

```
+ 1    out = list(x = x, y = y, z = z, npt = npt) out
```