Raskin et al., 2023 R markdown

Levi Raskin

3/20/2023

Please Cite All R Code as Follows:

Raskin, Levi Y.; Reeves, Jonathan S.; Douglass, Matthew J.; and Braun, David R. (March 30, 2023). Least-Effort Knapping as a Baseline to Study Social Transmission in the Early Stone Age [Poster]. Society for American Archaeology, Portland, Oregon.

Packages

```
knitr::opts_chunk$set(echo = TRUE)
knitr::opts_chunk$set(include = TRUE)
knitr::opts_chunk$set(eval = FALSE)
library(geometry)
library(plyr)
library(rgl)
library(ggplot2)
library(geomorph)
```

```
## Loading required package: RRPP

## Loading required package: Matrix
```

```
library(Momocs)
```

```
##
## Attaching package: 'Momocs'

## The following object is masked from 'package:geomorph':
##
##     mosquito

## The following objects are masked from 'package:plyr':
##
##     arrange, mutate, rename

## The following object is masked from 'package:stats':
##
##     filter
```

```
library(concaveman)
library(abind)
library(Rvcg)
library(Morpho)
```

```
##
## Attaching package: 'Morpho'

## The following objects are masked from 'package:Momocs':
##
##     export, tps2d

## The following object is masked from 'package:RRPP':
##
##     classify
```

```
library(RColorBrewer)
library(readobj)
library(vegan)
```

```
## Loading required package: permute

## Loading required package: lattice

## This is vegan 2.6-4
```

```
set.seed(5)
```

Functions

```
#A simple function that adds a matrix at the end of a list. Surprisingly useful. Used in
↪  this code multiple times from importing ply functions as meshes containing the
↪  vertices of the triangles to actually running the efourier funciton. This function,
↪  and some others, were made in consultation with OpenAI ChatGPT-3.
append_matrix_to_list <- function(my_list, matrix) {
  # Append the matrix to the end of the list
  my_list[[length(my_list)+1]] <- matrix

  return(my_list)
}

#plots two meshes over top each other in different colors in the rgl window. Used ensure
↪  that the meshes are aligned.
plot_xyz <- function(matrix1, matrix2) {
  # create a 3D plot of the first matrix in blue
  plot3d(matrix1, col = "blue", type = "p", main = "XYZ Plot")

  # add the second matrix to the plot in red
```

```
  points3d(matrix2, col = "red")
}

#samples n rows from a matrix for plotting; to reduce computational power and time to
↪  visualize the 3D meshes. Not used a ton and doesn't have a functional consequence for
↪  the efourier function.
sample_rows <- function(mat, n) {
  sample_indices <- sample(nrow(mat), n) # Get n random row indices
  return(mat[sample_indices, ]) # Return new matrix with n random rows
}

#centers mesh at origin; used for centering a mesh and then plotting eigenvectors over
↪  top. Used aloing with sample_rows.
center_mesh <- function(mesh) {
  center <- colMeans(mesh) # Calculate mean x, y, and z coordinates
  centered_mesh <- cbind(mesh[,1] - center[1], mesh[,2] - center[2], mesh[,3] -
↪  center[3]) # Center the mesh on the origin
  return(centered_mesh) # Return the centered mesh
}

#Super simple function to read a ply file and return the vertices as an xyz matrix.
readplyfunc <- function(path){
  x <- read.ply(path, ShowSpecimen = F)
  print(path)
  return(t(x$vb)[,1:3])
}
```

```
# Takes a 3D mesh and returns n slices which each define parallel planes. Returns a list
↪  of length numslices with x,y matrix for slice + vector containing the z values of
↪  those slices.
slicingfunction <- function(vertsmatrix, numslices, slicewidth){
  zmax <- max(vertsmatrix[,3])
  zmin <- min(vertsmatrix[,3])
  #spacing
  spacing <- (zmax-zmin)/numslices
  centerpntsvec <- 1:numslices

  #generating center pnts
  for(i in 1:numslices){
    if(i==1){
      centerpntsvec[i] <- zmin
    }
    else if(i==numslices){
      centerpntsvec[i] <- zmax
    }
    else{
      centerpntsvec[i] <- zmin+i*spacing
    }
  }


  for(i in 1:numslices){
    if(i==1){
```

```
      # Define the range of values
      min_val <- zmin
      max_val <- zmin+slicewidth

      # Subset the matrix for entries in the specified column within the range
      subset_matrix <- vertsmatrix[vertsmatrix[,3] >= min_val & vertsmatrix[,3] <=
↪  max_val, 1:2]

      datalist <- list(subset_matrix)

    }
    else if(i==numslices){
      # Define the range of values
      min_val <- zmax-slicewidth
      max_val <- zmax

      # Subset the matrix for entries in the specified column within the range
      subset_matrix <- vertsmatrix[vertsmatrix[,3] >= min_val & vertsmatrix[,3] <=
↪  max_val, 1:2]

      datalist <- append_matrix_to_list(datalist, subset_matrix)

    }
    else{
      # Define the range of values
      min_val <- centerpntsvec[i]-slicewidth
      max_val <- centerpntsvec[i]+slicewidth

      # Subset the matrix for entries in the specified column within the range
      subset_matrix <- vertsmatrix[vertsmatrix[,3] >= min_val & vertsmatrix[,3] <=
↪  max_val, 1:2]
      datalist <- append_matrix_to_list(datalist, subset_matrix)
    }
  }
  return(list(datalist, centerpntsvec))
}

#Where the wheels hit the road
#Takes each slice from the slicing function, applies a concave hull algorithm to the 2D
↪  matrix for each plane which finds the points which define the perimeter, and then
↪  runs efourier on the concave hull. 12 harmonics were applied. This function returns
↪  the inverse efourier points, which define the closed outline, as well as the z values
↪  defining the locaiton of each slice in space.
efourierfunction <- function(verts) {
    #10 slices too few, 100 is very (too?) highly resolved; 50 seems like sweetspot
  sliced <- slicingfunction(verts, 50, 0.25)
  slicematrix <- sliced[[1]]
  zvals <- sliced[[2]]

  efavec <- list(1:length(slicematrix))
  for(i in 1:length(slicematrix)){
    #concave hull algorithm is required as the slicing matrix, because of the 0.5 buffer
    ↪  zone, outputs messy outline
```

```
    #concavehull finds the points that define the perimeter of the slices and then
    ↪  applies efa to the outline
    concavehull <- concaveman(slicematrix[[i]])
    efa <- efourier(concavehull, 12)
    #need to convert from efa-- shape data-- into a class that procrustes will recognize
    #efourier_i outputs the points which define the EFA shape.
    efalmks <- efourier_i(efa)
    efavec[[i]]<-efalmks
  }
  return(list(efavec, zvals))
}

#combining the point data with the z values from the slicing function to output one
↪  array.
combine_matrices_and_vector <- function(matrices, z_values) {
  do.call(rbind, lapply(1:length(matrices), function(i) {
    cbind(matrices[[i]], z = z_values[i])
  }))
}

#Code to align meshes contained within an array; rigidAlign is excellent.
#code from https://rdrr.io/github/zarquon42b/RvtkStatismo/man/meshalign.html
rigidAlign <-
↪  function(array,scale=FALSE,use.lm=NULL,deselect=FALSE,reference=NULL,reflection=FALSE)
↪  {
  k <- dim(array)[1]
  if (!is.null(use.lm)) {
    use.lm <- unique(sort(use.lm))
    if (deselect) {
      use.lm <- c(1:k)[-use.lm]
    }
  }
  out <- partialAlign(array,use.lm =
↪  use.lm,scale=scale,reference=reference,reflection=reflection)
}
meshlist2array <- function(meshlist) {
  n <- length(meshlist)
  k <- ncol(meshlist[[1]]$vb)
  vertarr <- array(NA,dim=c(k,3,n))
  for (i in 1:n)
    vertarr[,,i] <- vert2points(meshlist[[i]])
  dimnames(vertarr)[[3]] <- names(meshlist)
  if (is.null(names(meshlist)))
    dimnames(vertarr)[[3]] <- paste("specimen",1:n,sep="_")
  return(vertarr)
}
partialAlign <- function(array,use.lm=NULL,scale=FALSE,reference=NULL,reflection=FALSE) {
  if (is.null(reference)){
    if (!is.null(use.lm)){
      procMod <-
↪  ProcGPA(array[use.lm,,],scale=scale,CSinit=F,reflection=reflection,silent =
↪  TRUE)##register all data using Procrustes fitting based on the non-missing
↪  coordinates
```

```
      tmp <- array
      a.list <-  1:(dim(array)[3])
      tmp <- lapply(a.list, function(i) {mat <-
↪  rotonmat(array[,,i],array[use.lm,,i],procMod$rotated[,,i],scale=scale,reflection =
↪  reflection);return(mat)})
      tmp1 <- Morpho::list2array(tmp)
      procMod$rotated <- tmp1
      procMod$mshape <- arrMean3(tmp1)
    } else {
      procMod <- ProcGPA(array,scale=scale,CSinit = F,reflection = reflection,silent = T)
    }
  } else {
    if (is.null(use.lm))
      use.lm <- 1:dim(array)[1]
    a.list <-  1:(dim(array)[3])
    tmp <- lapply(a.list, function(i) {mat <-
↪  rotonmat(array[,,i],array[use.lm,,i],array[,,reference],scale=scale,reflection =
↪  reflection);return(mat)})
    procMod <- list()
    procMod$rotated <- Morpho::list2array(tmp)
    procMod$mshape <- arrMean3(procMod$rotated)

  }
  return(procMod)
}
meshalign <-
↪  function(meshlist,scale=FALSE,use.lm=NULL,deselect=FALSE,array=FALSE,reference=NULL,reflection=FALS
↪  {
  vertarr <- meshlist2array(meshlist)
  out <-
↪  rigidAlign(vertarr,scale=scale,use.lm=use.lm,deselect=FALSE,reference=reference,reflection=reflecti
  if (array) {
    return(out)
  } else {
    if (is.null(reference)) {
      out <- lapply(1:length(meshlist),function(i){ res <- meshlist[[i]]
      res$vb[1:3,] <- t(out[,,i])
      res$normals <- NULL
      return(res)})
    } else {
      out <- lapply(1:length(meshlist),function(i){ res <- meshlist[[i]]
      res$vb[1:3,] <- t(out[,,i])
      res$it <- meshlist[[reference]]$it
      res$normals <- NULL
      return(res)})
    }
    names(out) <- names(meshlist)
    return(out)
  }
}
```

```
#a simple function to test for whether any rotation at all occured between the original
↪  and rotated array.
```

```
count_different_values <- function(arr1, arr2) {
  count <- 0
  for (i in 1:length(arr1)) {
    if (arr1[i] != arr2[i]) {
      count <- count + 1
    }
  }

  cat("Number of different values:", count, "\n")
}

# Used to find the removals for the specific file naming system we used
↪  (Blank_#_r_removal#.ply)
get_num_following_r <- function(string) {
  if(grepl("r\\d+", string)) {
    num <- as.numeric(gsub(".*r(\\d+).*", "\\1", string))
    return(num)
  } else {
    return(0)
  }
}
#Used to find the specific blank each experimental handaxe belonged to
lithicnumfunc <- function(input_string) {
  num <- as.numeric(sub("^.*Span_22_Blank\\s*(\\d+).*$", "\\1", input_string))
  return(num)
}

#Finds the extremes of a matrix; used to find the lithics which occupy PC extremes
get_max_min_row <- function(matrix, col) {
  max_val <- max(matrix[, col])
  min_val <- min(matrix[, col])
  max_row <- rownames(matrix)[which(matrix[, col] == max_val)]
  min_row <- rownames(matrix)[which(matrix[, col] == min_val)]

  cat("Maximum value:", max_val, "at row:", max_row, "\n")
  cat("Minimum value:", min_val, "at row:", min_row, "\n")
}

#Finds the second most extreme entries of a matrix; used to find the lithics which occupy
↪  PC extremes
get_second_largest_smallest_row <- function(matrix, col) {
  sorted_col <- sort(matrix[, col])
  second_largest_val <- sorted_col[length(sorted_col) - 1]
  second_smallest_val <- sorted_col[2]
  second_largest_row <- rownames(matrix)[which(matrix[, col] == second_largest_val)]
  second_smallest_row <- rownames(matrix)[which(matrix[, col] == second_smallest_val)]

  cat("Second largest value:", second_largest_val, "at row:", second_largest_row, "\n")
  cat("Second smallest value:", second_smallest_val, "at row:", second_smallest_row,
  ↪  "\n")
}
```

Reading in the meshes/the Orientation Problem

In typical landmark based 3D geometric morphometrics, the landmarks are in consistent positions in a specific order across the specimens. This consistent placement allows a Procrustes Transformation to align all the landmarks because there's known direction. However, the efourier function outputs 50 slices without any specific orientation information and the Procrustes Transformation doesn't really know what do. This problem is still open and there are better solutions than the one we implemented here, but we were able to align everything well enough with Morpho::pcAlign (raw mesh data), rigidAlign (on efourier 3D matrix) from https://rdrr.io/github/zarquon42b/RvtkStatismo/man/meshalign.html, and finally, a Procrustes (on rigidAlign output).

```
setwd("your folder with the 3D models or where you want to save the objects to")

# Set the folder path
folder_path <- "folder path with lithcs"

# Get the file names in the folder
file_names <- list.files(path = folder_path, full.names = TRUE)

#function to read .ply/.stl/.obj files
#outputs list containing the xyz matricies which define the verticies of the 3D models
for(i in 1:length(file_names)){
  if(grepl("\\.ply$", file_names[i])==T){
    vertices <- read.ply(file_names[i], ShowSpecimen = F)
    lithicsunaligned <- list(t(vertices$vb)[,1:3])
  }
  if(grepl("\\.stl$", file_names[i])==T){
    lithicsunaligned <- append_matrix_to_list(lithicsunaligned, readSTL(file_names[i],
↪  plot = F))
  }
  if(grepl("\\.obj$", file_names[i])==T){
    lithicsunaligned <- append_matrix_to_list(lithicsunaligned,
↪  t(read.obj(file_names[i])[["shapes"]][[1]][["positions"]]))
  }
  print(i)
}

#Morpho pcAlign aligns all the meshes to the first object
#It helps for the first entry to have definite directionality which the pcAlign can pic
↪  up on (i.e., in this context a highly reduced archaeological handaxe as opposed to a
↪  relatively unreduced experimental lithic)
for(i in 1:length(lithicsunaligned)){
  if(i == 1){
    lithicsunaligned[[i]]<-lithicsunaligned[[i]]
  }
  else{
    lithicsunaligned[[i]]<-Morpho::pcAlign(lithicsunaligned[[i]],lithicsunaligned[[1]])
  }
  print(i)
}
```

After pcAlign, efourier is then possible on the pcAligned mesh data.

```
lithicsefa <- lapply(lithicsunaligned, efourierfunction)

for(i in 1:length(lithicsefa)){
  if(i==1){
    lithicsarray <- combine_matrices_and_vector(lithicsefa[[i]][[1]],
↪  lithicsefa[[i]][[2]])
  }
  else if(i>1){
    lithicsarray <- abind(lithicsarray, combine_matrices_and_vector(lithicsefa[[i]][[1]],
↪  lithicsefa[[i]][[2]]), along = 3)
  }
}

#This code runs the efa function and outputs an array containing the shape data from the
↪  efourir function. This array can then be run through the standard Morpho/geomorph
↪  functions
```

After efourier, the slices can be rigidAligned and then sen tthrough a Procrustes transformation.

```
#rigidAlign
lithicsarrayrigidaligned<-rigidAlign(lithicsarray, scale = T, reflection = T, reference =
↪  1)

#procrustes
comproc <- gpagen(lithicsarrayrigidaligned)

#this code validates that all the lithics are truly aligned
for(i in 1:(length(comproc$coords)-1)){
  plot_xyz(comproc$coords[,,i],comproc$coords[,,i+1])
  invisible(readline(prompt="Press [enter] to proceed"))
}

PCA <- gm.prcomp(comproc$coords)
```

Figure Code

Following just shows the code we used to make the figures used in our poster.

```
#Spandrels vs arch
PCAcomdata <- as.data.frame(PCA[["x"]])

#Adding archaeology/spandrels row
PCAcomdata2 <- cbind(c(rep("FxJj 63", 13),rep("FxJj 31", 20),rep("Experimental",
↪  72)),PCAcomdata)
colnames(PCAcomdata2) <- append(c("Class"), colnames(PCAcomdata))

group1 <- PCAcomdata2[PCAcomdata2$Class == "Experimental", 2:17]
group2 <- PCAcomdata2[PCAcomdata2$Class == "FxJj 31", 2:17]
group3 <- PCAcomdata2[PCAcomdata2$Class == "FxJj 63", 2:17]

exphull <- coo_chull(PCAcomdata2[PCAcomdata2$Class == "Experimental", 2:3])
```

```r
fxjj31hull <- coo_chull(PCAcomdata2[PCAcomdata2$Class == "FxJj 31", 2:3])
fxjj63hull <- coo_chull(PCAcomdata2[PCAcomdata2$Class == "FxJj 63", 2:3])


colors <- c("#E89302","#F23002","#DB00CD","#340CF2","#049DEB")

# Plot convex hulls on top of the PCA plot
ggplot() +
  geom_point(data=PCAcomdata2, aes(x = Comp1, y = Comp2, color = Class))+
  scale_color_manual(values = colors[1:3])+
  geom_polygon(data = as.data.frame(exphull),
               aes(x = Comp1, y = Comp2),
               fill = colors[1],
               alpha = 0.2) +
  geom_polygon(data = as.data.frame(fxjj31hull),
               aes(x = Comp1, y = Comp2),
               fill = colors[2],
               alpha = 0.2)+
  geom_polygon(data = as.data.frame(fxjj63hull),
               aes(x = Comp1, y = Comp2),
               fill = colors[3],
               alpha = 0.2)+
  ggtitle("PC1 vs PC2")+
  xlab("PC1 (18.40%)")+
  ylab("PC2 (12.20%)")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))


exphull <- coo_chull(PCAcomdata2[PCAcomdata2$Class == "Experimental", 3:4])
fxjj31hull <- coo_chull(PCAcomdata2[PCAcomdata2$Class == "FxJj 31", 3:4])
fxjj63hull <- coo_chull(PCAcomdata2[PCAcomdata2$Class == "FxJj 63", 3:4])


# Plot convex hulls on top of the PCA plot
ggplot() +
  geom_point(data=PCAcomdata2, aes(x = Comp3, y = Comp2, color = Class))+
  scale_color_manual(values = colors[1:3])+
  geom_polygon(data = as.data.frame(exphull),
               aes(x = Comp3, y = Comp2),
               fill = colors[1],
               alpha = 0.2) +
  geom_polygon(data = as.data.frame(fxjj31hull),
               aes(x = Comp3, y = Comp2),
               fill = colors[2],
               alpha = 0.2)+
  geom_polygon(data = as.data.frame(fxjj63hull),
               aes(x = Comp3, y = Comp2),
               fill = colors[3],
               alpha = 0.2)+
  ggtitle("PC3 vs PC2")+
  xlab("PC3 (9.37%)")+
  ylab("PC2 (12.20%)")+
```

```
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))



#Reduction through morphospace
spandrelspca<- readRDS("spandrelspca")

#Getting removal data
get_num_following_r <- function(string) {
  if(grepl("r\\d+", string)) {
    num <- as.numeric(gsub(".*r(\\d+).*", "\\1", string))
    return(num)
  } else {
    return(0)
  }
}

# Set the folder path
folder_path <- "C:\\Users\\levir\\Desktop\\KFFS 22 exp arch scans ascii"

# Get the file names in the folder
file_names <- list.files(path = folder_path, full.names = TRUE)

# apply the function to the vector of file names
removal <- sapply(file_names, get_num_following_r)
#removing the exp array entry
removal <- removal[1:72]

thicnum <- sapply(file_names, lithicnumfunc)
#removing the exp array entry
lithicnum <- lithicnum[1:72]
#one example where Span_22_Blank_1 instead of Blank 1, just doing manually
lithicnum[72] <- 1

#graphing

#graphing
PCAexparchdata <- as.data.frame(PCAcomalignedtoarch[["x"]])
#Adding archaeology/spandrels row
PCAexparchdata2 <- cbind(lithicnum,removal,PCAexparchdata[21:92,])
colnames(PCAexparchdata2) <- append(c("Lithic","Removal"), colnames(PCAexparchdata))

lith11df <- filter(PCAexparchdata2, `Lithic`==11)[,1:4]
lith5df <- filter(PCAexparchdata2, `Lithic`==5)[,1:4]
lith2df <- filter(PCAexparchdata2, `Lithic`==2)[,1:4]
lith7df <- filter(PCAexparchdata2, `Lithic`==7)[,1:4]
lith8df <- filter(PCAexparchdata2, `Lithic`==8)[,1:4]

colors <- c("#E89302","#F23002","#DB00CD","#340CF2","#049DEB")
textdf <- rbind(filter(PCAexparchdata2, Lithic == 5),filter(PCAexparchdata2, Lithic ==
↪  11),filter(PCAexparchdata2, Lithic == 2),filter(PCAexparchdata2, Lithic ==
↪  7),filter(PCAexparchdata2, Lithic == 8))
```

```
pointdf <- PCAexparchdata2[!(PCAexparchdata2[, 1] %in% c(5, 11, 2, 7, 8)), ]

ggplot()+
  geom_point(data = PCAexparchdata, aes(x=Comp1, y=Comp2), show.legend = F)+
  geom_path(data=lith5df[order(lith5df$Removal),][1:2,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[1], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith5df[order(lith5df$Removal),][2:3,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[1], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith5df[order(lith5df$Removal),][3:4,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[1], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith5df[order(lith5df$Removal),][4:5,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[1], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith5df[order(lith5df$Removal),][5:6,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[1], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith5df[order(lith5df$Removal),][6:7,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[1], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith11df[order(lith11df$Removal),][1:2,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[2], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith11df[order(lith11df$Removal),][2:3,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[2], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith11df[order(lith11df$Removal),][3:4,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[2], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith11df[order(lith11df$Removal),][4:5,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[2], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith11df[order(lith11df$Removal),][5:6,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[2], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith11df[order(lith11df$Removal),][6:7,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[2], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith2df[order(lith2df$Removal),][1:2,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[3], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith2df[order(lith2df$Removal),][2:3,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[3], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith2df[order(lith2df$Removal),][3:4,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[3], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith2df[order(lith2df$Removal),][4:5,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[3], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith2df[order(lith2df$Removal),][5:6,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[3], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith2df[order(lith2df$Removal),][6:7,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[3], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith7df[order(lith7df$Removal),][1:2,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[4], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith7df[order(lith7df$Removal),][2:3,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[4], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith7df[order(lith7df$Removal),][3:4,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[4], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith7df[order(lith7df$Removal),][4:5,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[4], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith7df[order(lith7df$Removal),][5:6,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[4], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith7df[order(lith7df$Removal),][6:7,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[4], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith8df[order(lith8df$Removal),][1:2,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[5], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
```

```
  geom_path(data=lith8df[order(lith8df$Removal),][2:3,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[5], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith8df[order(lith8df$Removal),][3:4,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[5], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith8df[order(lith8df$Removal),][4:5,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[5], arrow=arrow(angle = 30, length = unit(0.05, "cm"), type = "closed"))+
  geom_path(data=lith8df[order(lith8df$Removal),][5:6,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[5], arrow=arrow(angle = 30, length = unit(0.3, "cm")))+
  geom_path(data=lith8df[order(lith8df$Removal),][6:7,], aes(x=Comp1, y=Comp2), color =
  ↪  colors[5], arrow=arrow(angle = 30, length = unit(0.3, "cm")))+
  geom_label(data=filter(textdf, Lithic == 5), aes(x = Comp1, y= Comp2, label = Removal),
  ↪  color = colors[1], show.legend = F,label.padding = unit(0.035, "cm"))+
  geom_label(data=filter(textdf, Lithic == 11), aes(x = Comp1, y= Comp2, label =
  ↪  Removal), color = colors[2], show.legend = F,label.padding = unit(0.035, "cm"))+
  geom_label(data=filter(textdf, Lithic == 2), aes(x = Comp1, y= Comp2, label = Removal),
  ↪  color = colors[3], show.legend = F,label.padding = unit(0.035, "cm"))+
  geom_label(data=filter(textdf, Lithic == 7), aes(x = Comp1, y= Comp2, label = Removal),
  ↪  color = colors[4], show.legend = F,label.padding = unit(0.035, "cm"))+
  geom_label(data=filter(textdf, Lithic == 8), aes(x = Comp1, y= Comp2, label = Removal),
  ↪  color = colors[5], show.legend = F,label.padding = unit(0.035, "cm"))+
  ggtitle("Reduction Through Shapespace")+
  xlab("PC1 (18.40%)")+
  ylab("PC2 (12.20%)")+
  #labs(caption = "Each color represents an example individual lithic's shape changes
  ↪  during reduction, with the numbers describing the number of flakes removed.")+
  theme_light()+
  theme(plot.title = element_text(hjust = 0.5))

#Handaxe sliced up
plotly::plot_ly(x = comproc$coords[,,2][,1], y = comproc$coords[,,2][,2], z =
↪  comproc$coords[,,2][,3], type = "scatter3d", color=colors[2], size = 0.5)
```

Permanova stats test

The PERMANOVA test from the vegan population allows us to ask whether or not group (archaeological or experimental) data adds any descriptive information to the variation seen in the shape space data.

```
groups <- c(rep("Arch", 33),rep("Experimental", 72))
scores <- PCAcomdata2[,2:3] #PC1 and PC2 scores; repeat with PC2 and PC3 or over a range
↪  of PCs

#the research question is whether or not knowledge of group (archaeological or
↪  experimental) implies anything about the shape of the lithic. Therefore, we want to
↪  per mutate the group identities to test whether group contains any information about
↪  shape. Typically, in archaeology, the question is the other way around-- what does
↪  shape imply about group-- but the research question here is does "Acheulean" imply
↪  anything about shape.
permanova <- adonis2(scores ~ groups, method="mahalanobis", permutations=999)
```