Análisis de nubes de puntos masivas en R

Nataly Romarís-Lodeiro[1], Olamar Benavente-Fernández[1], Rubén Fernández-Casal[2], y Salvador Naya[2].

[1]Centro Mixto de Investigación UDC-Navantia, Universidade da Coruña.
[2]Grupo MODES, Departamento de Matemáticas, CITIC, Universidade da Coruña.

XI Xornada de Usuarios de R en Galicia.
24/10/2024.

1. Paquetes recomendados para el análisis 3D en R
2. Creación e importación de nubes de puntos en R
3. Representación de las nubes de puntos 3D
4. Aproximación de formas mediante mallas triangulares
5. Herramientas adicionales para la manipulación de mallas en
6. Integración con Python usando reticulate

Paquetes recomendados para el análisis 3D en R

Paquete	Campo
`rgl`	Visualización y manipulación de objetos 3D, nubes de puntos y mallas triangulares.
`Rvcg`	Procesamiento y optimización de mallas triangulares.
`lidR` y `lasR`	Manipulación de datos LiDAR en análisis forestales.
`Morpho`	Análisis morfológico de estructuras biológicas.

El paquete `rgl` es fundamental para analizar grandes conjuntos de datos.

Web del paquete `rgl`: https://dmurdoch.github.io/rgl/

Entre algunas de las funciones disponibles del paquete `rgl` tenemos:

- `open3d()`: Abre una nueva ventana gráfica.
- `clear3d()`: Limpia la escena gráfica actual de la ventana.
- `close3d()`: Cierra una o más ventanas gráficas.

Creación e importación de nubes de puntos en R

Los formatos más comunes de las nubes de puntos 3D son:

- Archivos `.las` o `.laz` (usando la función `readLAS()` del paquete `lidR`).
- Archivos de texto (usando `read.csv()` o `fread()` del paquete `data.table`).

Para crear una nube de puntos, se puede utilizar:

- Un `data.frame`.
- Un array por filas o por columnas.

A continuación, se muestra un ejemplo de cómo generar una nube de puntos como una matriz por filas:

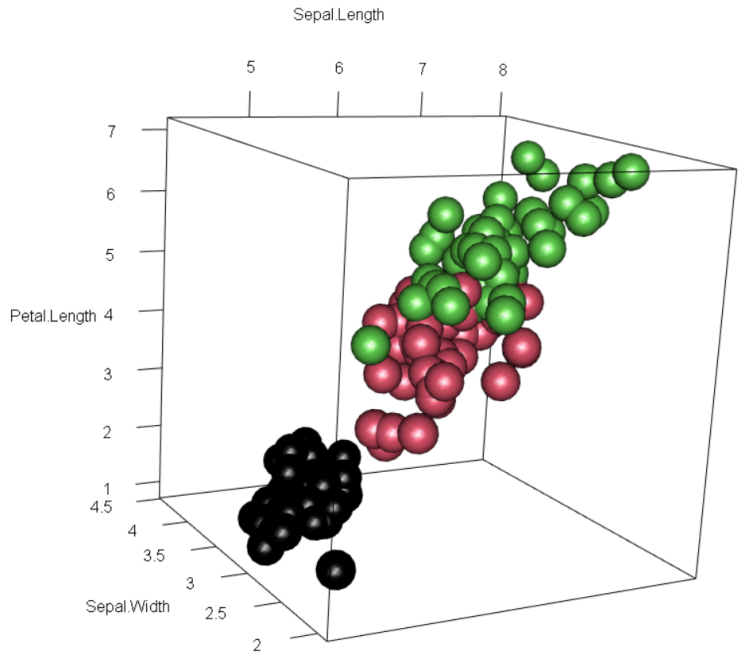
```
n <- 4000 # Número total de puntos
T1 <- rtorus(n/2, 0.5, 2)  # Toro 1
T2 <- rtorus(n/2, 0.5, 2, ct = c(2, 0, 0), rotx = pi/3)  # Toro 2
x <- rbind(T1, T2) # Combinar los puntos de ambos toros
```

Representación de las nubes de puntos 3D

Para representar nubes de puntos en 3D se puede utilizar la función `plot3d()` que nos permite obtener gráficos tridimensionales interactivos.

Como ejemplo, utilizamos el conjunto de datos Iris, que contiene mediciones de flores de diferentes especies de lirios.

```
with(iris, plot3d(Sepal.Length, Sepal.Width, Petal.Length,
                  type = "s", col = as.numeric(Species)))
```



Aproximación de formas mediante mallas triangulares

En muchas ocasiones las nubes de puntos no son una representación adecuada del objeto escaneado e interesa aproximar su forma mediante una malla triangular.

En `rgl` una malla triangular se representa como una lista con las siguientes componentes:

Función	Descripción
`vb`:	Matriz $4 \times n$ con vértices en coordenadas homogéneas (incluye 4 dimensiones para facilitar el cálculo de transformaciones afines).
`it`:	Matriz $3 \times t$ de índices de vértices. Cada columna define un triángulo.
`Normals`:	Matriz con los vectores normales de cada vértice, utilizados para suavizar las superficies.
`Texcoords`:	Matriz $2 \times n$ con coordenadas de textura de cada vértice.
`Values`:	Vector de longitud n que almacena valores numéricos de cada vértice.
`Meshcolor`:	Valor de texto de interpretación de colores y coordenadas de textura.

Existen diversas funciones en R para importar mallas triangulares empleando distintos formatos. A continuación, se presentan los formatos más comunes junto con sus respectivos paquetes y funciones:

Formato	Paquete	Función
`PLY`	`Rvcg`	`vcgPlyRead()`
`STL`	`rgl`	`readSTL()`
`OBJ`	`rgl`	`readOBJ()`

Función recomendada: `vcgImport()`.

Ventajas:

- Permite importar varios formatos de mallas triangulares.
- No requiere funciones específicas para cada formato.
- Se adapta automáticamente al formato del archivo.

Para aproximar la forma de la nube de puntos a través de una malla triangular podemos aplicar diversas técnicas, entre ellas:

- La envoltura α-convexa de la nube de puntos.
- Niveles de densidad (isosuperficies).
- El método Ball-Pivoting.

Para aplicar estas técnicas de manera eficiente, es crucial reducir la dimensionalidad de los datos, lo que facilita la creación de una malla triangular. Si bien estas técnicas pueden gestionar grandes volúmenes de datos, reducir el número de puntos es fundamental para optimizar el procesamiento, mejorar la eficiencia y evitar sobrecargar los recursos computacionales, garantizando una representación más manejable.

Para ello se pueden emplear distintas técnicas de discretización, entre ellas tenemos:

- Binning.
- Voxelización.

Binning: Agrupa las observaciones en celdas regulares, conocidas como "bins", reduciendo la cantidad de datos a procesar.

Los métodos Binning más utilizados son:

- Binning simple: Asigna cada punto al nodo más cercano.
- Binning lineal: Asigna cada punto a los nodos más cercanos con un peso que depende de su proximidad, permitiendo una interpolación más precisa.

```
library(npsp)
bin.den(x, nbin = NULL, type = c("linear", "simple"))
```

Argumentos:

- `x`: Matriz de coordenadas espaciales.
- `nbin`: Vector con el número de bins en cada dimensión.
- `type`: Método de binning ("lineal" por defecto).

Voxelización: Es una técnica igual o similar al binning simple (sólo se consideran los nodos con pesos positivos). En algunos casos se emplea el promedio de las coordenadas en el vóxel en lugar del centro del vóxel.

Envoltura α-convexa de la nube de puntos

La envoltura α-convexa es una técnica de geometría computacional que permite delimitar un conjunto de puntos en un espacio tridimensional sin la restricción de que el conjunto sea convexo. Para su cálculo, se utilizan los siguientes paquetes:

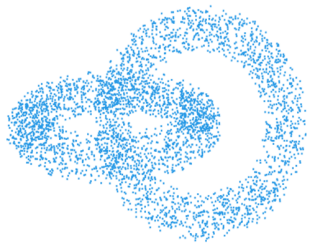
- alphahull en 2D (Pateiro-López y Rodríguez-Casal, 2010).
- alphashape3d en 3D (Lafarge et al., 2014).

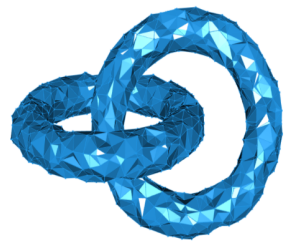
```
library(alphahull)
ashape(x, alpha)

library(alphashape3d)
ashape3d(x, alpha)
```

Argumentos:

- `x`: Una matriz de 3 columnas con las coordenadas de los puntos.
- `alpha`: Un único valor o vector de valores para α.





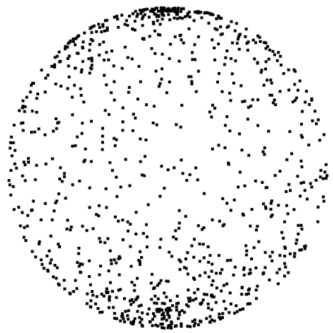
Niveles de densidad (Isosuperficies)

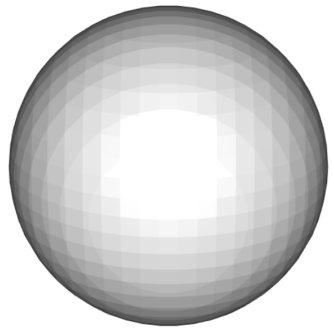
Una técnica para aproximar la forma del objeto es la generación de isosuperficies basadas en niveles de densidad. Este método emplea estimaciones de densidad sobre una rejilla regular, obtenidas mediante suavizado tipo núcleo (con paquetes como `ks` o `npsp`) o directamente a partir de la rejilla binning (ver e.g. Chacón y Duong, 2018, Sección 6.1).

```
library(misc3d)
contour3d(f, level, x, y, z)
```

Argumentos:

- `f`: Una función de 3 argumentos o una matriz tridimensional.
- `level`: El nivel o niveles en los que construir las superficies de contorno.
- `x, y, z`: Localización de los puntos de la rejilla donde se evalúa la función.





Método Ball – Pivoting

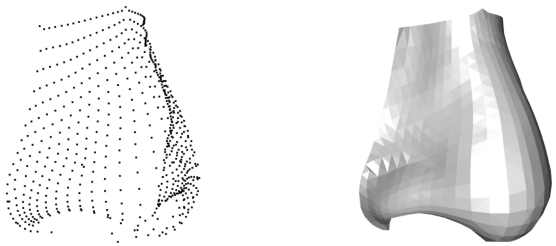
Este algoritmo es relativamente sencillo: Tres puntos forman un triángulo en la malla si una bola de radio p (definida por el usuario) puede tocarlos sin incluir otros puntos de la nube.

El proceso comienza con un triángulo inicial, llamado "semilla". Luego, la bola gira alrededor de los bordes de este triángulo para encontrar nuevos puntos y formar más triángulos, repitiéndose este proceso hasta que se hayan probado todos los bordes accesibles y todos los puntos hayan sido considerados (Schlager, 2017).

```
library(Rvcg)
vcgBallPivoting(x, radius = 0)
```

Argumentos:

- `x`: matriz $k \times 3$ o un objeto de clase `mesh3d`.
- `radius`: El radio de la bola que pivota (rueda) sobre el conjunto de puntos.



Herramientas adicionales para la manipulación de mallas

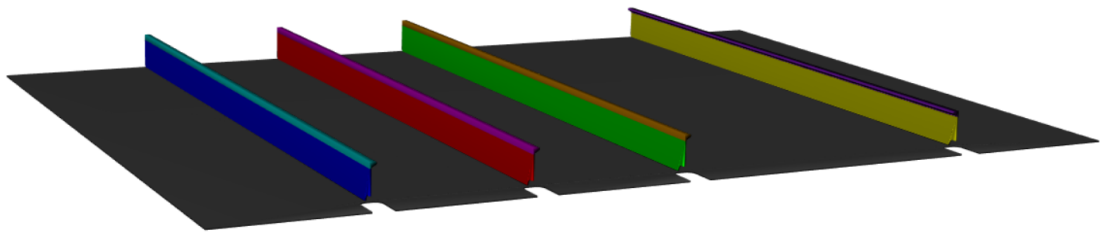
El paquete `rgl`, `Rvcg` y `Morpho` proporcionan funciones avanzadas para manipular y analizar mallas 3D.

`vcgIsolated()`: Segmenta una malla 3D identificando y separando componentes aislados, permitiendo eliminar piezas desconectadas de la malla original a partir de una cantidad mínima de caras o de un diámetro inferior a un umbral dado.

```
library(Rvcg)
vcgIsolated(mesh, split = TRUE)
```

Argumentos:

- `mesh`: Malla triangular de clase `mesh3d`.
- `split`: Lógico: si es `TRUE`, devuelve una lista con todos los componentes conectados de la malla.



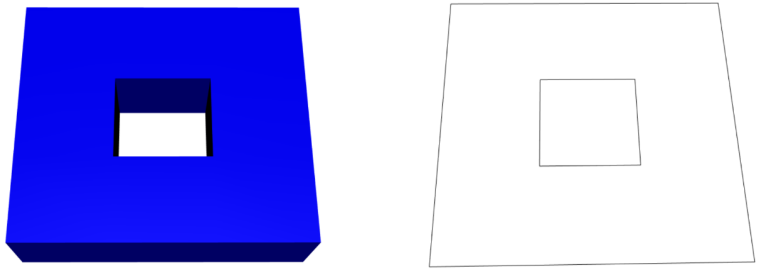
`facing3d()`: Permite obtener un objeto malla formado por los triángulos orientados en una dirección deseada.

```
library(rgl)
facing3d(mesh, up = c(0, 0, 1))
```

`getBoundary3d()`: Construye una malla de segmentos de línea correspondientes a las aristas no compartidas de triángulos o cuadriláteros de la malla original.

```
library(rgl)
getBoundary3d(mesh, sorted = FALSE)
```

- `mesh`: Un objeto de malla.
- `up`: La dirección considerada "hacia arriba".
- `sorted`: Si el resultado debe tener los segmentos ordenados secuencialmente.

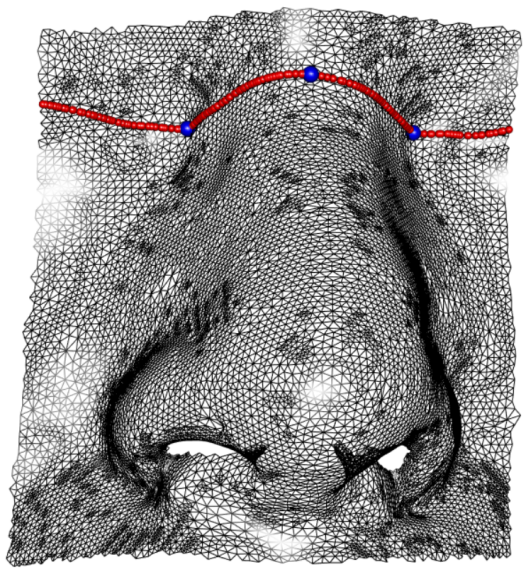


`meshPlaneIntersect()`: Permite obtener las intersecciones entre la malla y un plano.

```
library(Morpho)
meshPlaneIntersect(mesh, v1, v2 , v3)
```

Argumentos:

- `mesh`: Malla triangular de clase `mesh3d`.
- `v1, v2, v3`: Vectores numéricos de longitud 3 que especifican los puntos del plano.



Entre otras funciones destacadas del paquete `Rvcg` tenemos:

`vcgClean()`: Elimina componentes irrelevantes según criterios como el número mínimo de caras o el diámetro máximo.

```
library(Rvcg)
vcgClean(mesh, sel = 0)
```

- `mesh`: Malla triangular de clase `mesh3d`.
- `sel`: Vector entero que selecciona el tipo de limpieza (ver paquete para detalles).

`vcgSmooth()`: Suaviza la malla.

```
library(Rvcg)
vcgSmooth(mesh,
          type = c("taubin", "laplace", "HClaplace", "fujiLaplace",
                   "angWeight", "surfPreserveLaplace")
```

- `mesh`: Malla triangular de clase `mesh3d`.
- `type`: Selecciona el algoritmo de suavizado.

`vcgUpdateNormals()`: Recalcula los vectores normales.

```
library(Rvcg)
vcgUpdateNormals(mesh)
```

- `mesh`: Malla triangular de clase `mesh3d` o una matriz $n \times 3$ que contenga coordenadas 3D.

`vcgGeodist()`: Cálcula distancias geodésicas entre puntos.

```
library(Rvcg)
vcgGeodist(x, pt1, pt2)
```

- `x`: Malla triangular de clase `mesh3d`.
- `pt1`: Coordenada 3D de la malla o índice del vértice.
- `pt2`: coordenada 3D de la malla o índice del vértice.

Integración con Python usando `reticulate`

Podemos aprovechar herramientas adicionales implementadas en Python e integrarlas con R utilizando el paquete `reticulate`. Esto nos permite lograr:

- Mejor rendimiento en la generación de mallas triangulares, como con el método Ball - Pivoting.
- Mayor eficiencia al trabajar con grandes nubes de puntos, donde R puede ser ineficaz.
- Reducción de los tiempos de computación en ciertos casos.

Con `reticulate`, es posible invocar funciones de Python desde R o trabajar directamente con un intérprete de Python dentro de R utilizando la función `repl_python()`, combinando así las ventajas de ambos lenguajes y superando las limitaciones computacionales de R.

Para la creación de mallas en Python, algunas funciones útiles de la librería `open3d` son:

- `voxel_down_sample()`: Reduce la resolución de una nube de puntos mediante un tamaño de vóxel.
- `ball_pivoting()`: Genera una malla 3D a partir de una nube de puntos usando radios para definir el tamaño de las bolas pivotantes.

`Open3D` es una biblioteca de código abierto diseñada para el desarrollo rápido de software que trabaja con datos 3D, disponible tanto en C++ como en Python. Está optimizada para la paralelización y cuenta con pocas dependencias, facilitando su configuración en diversas plataformas y su compilación desde el código fuente.

Conclusiones

Hemos explorado cómo R ofrece diversas herramientas para trabajar con nubes de puntos masivas, abarcando desde la visualización inicial hasta la creación de mallas triangulares que aproximan la forma del objeto.

R con sus paquetes especializados como `rgl`, `Rvcg`, `Morpho`, entre otros, es una plataforma poderosa para el análisis de grandes conjuntos de datos. Sin embargo, en algunos casos, la integración con Python puede optimizar el rendimiento de ciertas operaciones.

Referencias

- Chacón J.E., Duong T. (2018). *Multivariate kernel smoothing and its applications*. Chapman and Hall/CRC.
- Bernardini F., Mittleman J., Rushmeier H., Silva C., Taubin G. (1999). The ball-pivoting algorithm for surface reconstruction. *IEEE transactions on visualization and computer graphics*, 5(4), 349-359.
- Duong T. (2022). *ks: Kernel Smoothing*. R package version 1.14.0, https://CRAN.R-project.org/package=ks.
- Feng D., Tierney L. (2008). Computing and Displaying Isosurfaces in R. *Journal of Statistical Software*, 28, 1-24.
- Fernández-Casal R. (2024). *npsp: Nonparametric Spatial Statistics*. R package version 0.7-14. https://rubenfcasal.github.io/npsp.
- Lafarge T., Pateiro-López B., Possolo A., Dunkers J. (2014). R Implementation of a Polyhedral Approximation to a 3D Set of Points Using the α-Shape. *Journal of Statistical Software*, 56, 1-19.
- Lecigne B., Delagrange S., Messier C. (2018). Exploring trees in three dimensions: VoxR, a novel voxel-based R package dedicated to analysing the complex arrangement of tree crowns. *Annals of Botany*, 121, 589-601.

- Lorensen W.E., Cline H.E. (1987). Marching Cubes: A High Resolution 3D Surface Reconstruction Algorithm. *Computer Graphics*, 21(4), 163-169.
- Murdoch D., Adler D. (2023). *rgl: 3D Visualization Using OpenGL*. R package version 1.1.3, https://CRAN.R-project.org/package=rgl.
- Pateiro-López B., Rodríguez-Casal A. (2010). Generalizing the convex hull of a sample: the R package alphahull. *Journal of Statistical Software*, 34, 1-28.
- Roussel J.R., Auty D. (2023). *lidR: Airborne LiDAR Data Manipulation and Visualization for Forestry Applications*. R package version 3.1.0. https://cran.r-project.org/package=lidR.
- Roussel, J.R. (2024). *lasR: Fast and Pipeable Airborne LiDAR Data Tools*. R package version 0.10.2, https://r-lidar.github.io/lasR.
- Schlager S. (2017). Morpho and Rvcg - Shape Analysis in R. En Zheng G., Li S., Szekely G. (Eds.), *Statistical Shape and Deformation Analysis*, pp. 217-256. Academic Press.
- Ushey K, Allaire J, Tang Y (2024). *reticulate: Interface to 'Python'*. R package version 1.39.0, https://CRAN.R-project.org/package=reticulate.

Agradecimientos

Gracias por su atención!

Este trabajo ha sido financiado por la Axencia Galega de Innovación (GAIN) de la Xunta de Galicia y la empresa Navantia (SEPI), en el marco del Centro Mixto de Investigación UDC-NAVANTIA, con el proyecto "O estaleiro do futuro" (IN853C).