

Isovistas en R

Rodrigo Tapia McClung

Centro de Investigación en Ciencias de Información Geoespacial - CentroGeo

rtapia@centrogeo.edu.mx

15 de enero, 2020

Crear *isovistas* en R

Una *isovista* es el conjunto de puntos visibles desde alguna ubicación en el espacio. Vamos a suponer que tenemos unas ciertas posiciones de cámaras o de observadores en algún lado y queremos calcular qué es lo que pueden ver, de acuerdo a algunos obstáculos que tengan enfrente y suponiendo que pueden ver 360 grados.

Los datos para esta práctica los puedes descargar de [aquí](#). Una vez descargados los datos, descomprime el archivo en alguna carpeta en tu computadora.

Primero nos cambiamos al directorio de trabajo donde tenemos los datos descargados para esta práctica.

```
# Cambiar directorio de trabajo:
setwd("C:/Descargas/escuela-de-metodos-2020/practica1")
```

Nos aseguramos de tener instaladas las dos librerías que vamos a usar y las cargamos:

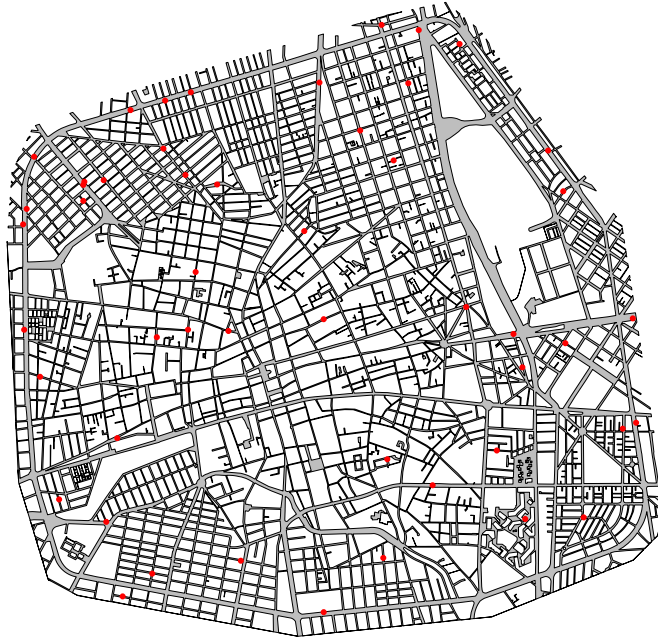
```
if (!require("pacman")) install.packages("pacman")
pacman::p_load(sf, tidyverse)
```

Y entonces podemos leer los datos, que son SHPs (*shapefiles*):

```
calles <- read_sf("calles.shp", "calles")
puntos <- read_sf("points.shp", "points")
```

NOTA: estamos usando unos puntos que están en el primer anillo de la ciudad de Aguascalientes. Si quieres usar puntos aleatorios, puedes crearlos con `puntos <- st_sf(id = 0:49, geometry = st_sample(calles, 50))`. Podemos hacer un primer mapa de las calles y de los puntos que estamos utilizando:

```
par(mar = c(0, 0, 0, 0))
plot(st_geometry(calles), col = "grey", lwd = 0.5)
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
```



Ahora hacemos un *buffer* de aproximadamente 100 metros alrededor de cada punto y los agregamos al mapa que ya tenemos:

```
buffer <- st_buffer(puntos, dist = 0.001, nQuadSegs = 4)

## Warning in st_buffer.sfc(st_geometry(x), dist, nQuadSegs, endCapStyle =
## endCapStyle, : st_buffer does not correctly buffer longitude/latitude data
## dist is assumed to be in decimal degrees (arc_degrees).

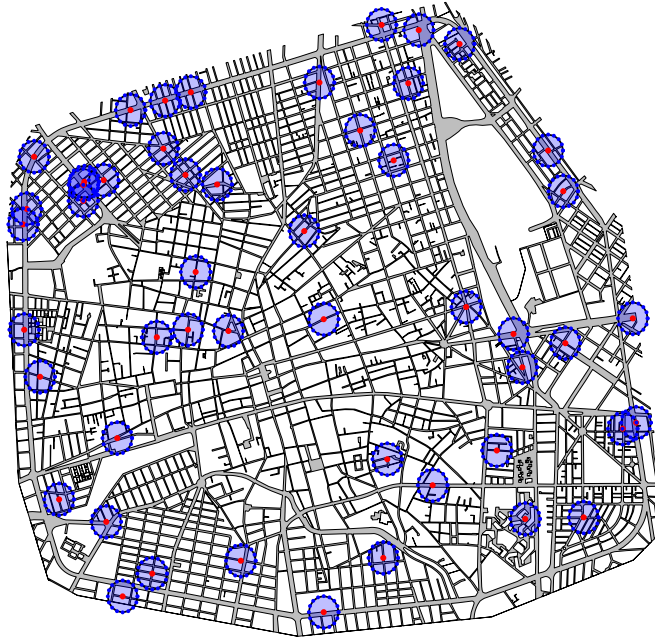
#st_crs(buffer)
par(mar = c(0, 0, 0, 0))
plot(st_geometry(calles), col = "grey", lwd = 0.5)
plot(st_geometry(buffer), add = T, col = rgb(red = 0, green = 0, blue = 1, alpha = 0.25))
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
```



Ahora extraemos los bordes de los *buffers* como puntos y les ponemos el identificador del punto al que pertenecen. Luego los podemos dibujar en el mapa:

```
bordes <- st_sf(p.id = 1:17,
               0.id = rep(1:length(puntos$geometry), each = 17),
               geometry = st_cast(buffer$geometry, "POINT"))

par(mar = c(0, 0, 0, 0))
plot(st_geometry(calles), col = "grey", lwd = 0.5)
plot(st_geometry(buffer), add = T, col= rgb(red = 0, green = 0, blue = 1, alpha = 0.25))
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
plot(st_geometry(bordes), add = T, col = "blue", pch = 20, cex = 0.25)
```



Ahora vamos a construir los rayos desde cada punto central y hacia los puntos de los bordes. Vamos a iterar a través de los n puntos de cada borde, para los 50 puntos que tenemos en el mapa. Lo más fácil es tener las coordenadas de origen y destino de esos rayos en una tabla y hacer que cada renglón de esa tabla se vuelva un rayo. Algo así:

rayo ₁	$x_{1,1}$	$y_{1,1}$	$x_{1,2}$	$y_{1,2}$
rayo ₂	$x_{2,1}$	$y_{2,1}$	$x_{2,2}$	$y_{2,2}$
rayo ₃	$x_{3,1}$	$y_{3,1}$	$x_{3,2}$	$y_{3,2}$
\vdots	\vdots	\vdots	\vdots	\vdots

Para eso, primero vamos a construir los arreglos de coordenadas para cada conjunto. Necesitamos las coordenadas de los centros y de los puntos que conforman los bordes. Así que esto tenemos que hacerlo dos veces:

```
# Columnas de coordenadas de los puntos + ids
puntos_coords <- do.call(rbind, st_geometry(puntos)) %>%
  as_tibble() %>% setNames(c("lon", "lat")) %>%
  mutate(0.id = puntos$id)
```

Warning: `as_tibble.matrix()` requires a matrix with column names or a `.name_repair` argument. Using
This warning is displayed once per session.

```
# Columnas de coordenadas de los bordes + ids
bordes_coords <- do.call(rbind, st_geometry(bordes)) %>%
  as_tibble() %>% setNames(c("lon", "lat")) %>%
  mutate(p.id = bordes$p.id) %>%
  mutate(0.id = bordes$0.id - 1)
```

Si te importa el orden de las columnas, puedes agregar algo como `%>% select(0.id, lat, lon, everything())` o `%>% select(p.id, 0.id, lat, lon, everything())`, respectivamente.

Ya que tenemos estas listas de coordenadas, armamos un objeto tipo *data frame* que tenga un identificador y las coordenadas en el orden que las queremos. En particular, queremos que los rayos vayan del centro hacia afuera, así que las primeras coordenadas son las de los centros y hacemos un `join` con las coordenadas de los bordes:

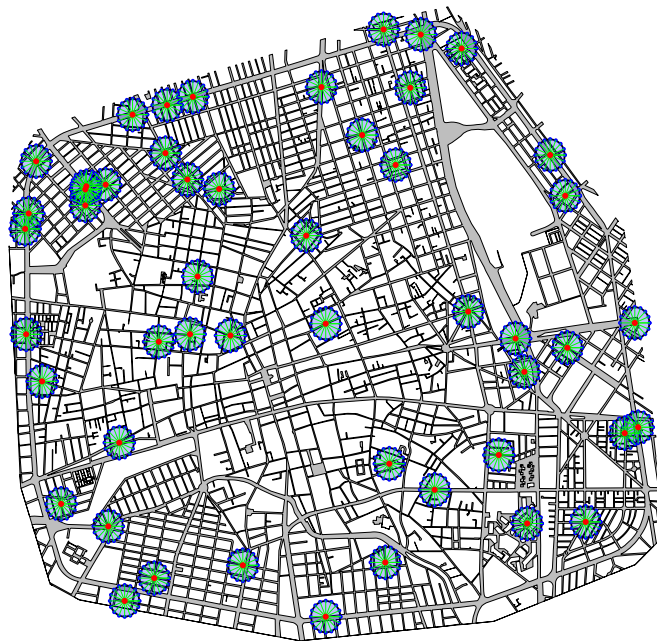
```
# Unir las coordenadas de los centros con los bordes para los rayos
puntos.lineas <- puntos_coords %>% left_join(bordes_coords, by = c("0.id"))
# El data frame anterior lo separamos en grupos de 1 por 1
lista.lineas <- split(puntos.lineas, seq(nrow(puntos.lineas)))
```

Ahora podemos definir los rayos con esas coordenadas:

```
# Hacer rayos desde el centro hacia afuera
hazRayos <- lapply(lista.lineas, function(row) {
  lmat <- matrix(unlist(row[c(1,2,4,5)]), ncol = 2, byrow = TRUE)
  st_linestring(lmat)
})
rayos <- st_sfc(hazRayos)
# Definir sf con CRS
rayos_sf <- st_sf(0.id = puntos.lineas$0.id, p.id = puntos.lineas$p.id, geometry = rayos, crs= 4326)
```

Usar `crs= 4326` es para definir el sistema de referencia de coordenadas de la capa que estamos creando. Ahora podemos pintar los rayos:

```
par(mar = c(0, 0, 0, 0))
plot(st_geometry(calles), col = "grey", lwd = 0.5)
plot(st_geometry(buffer), add = T, col = rgb(red = 0, green = 0, blue = 1, alpha = 0.25))
plot(st_geometry(bordes), add = T, col = "blue", pch = 20, cex = 0.25)
plot(st_geometry(rayos_sf), add = T, col = "green", lwd = 0.5)
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
```



Ahora intersectamos los rayos con las calles y los pintamos:

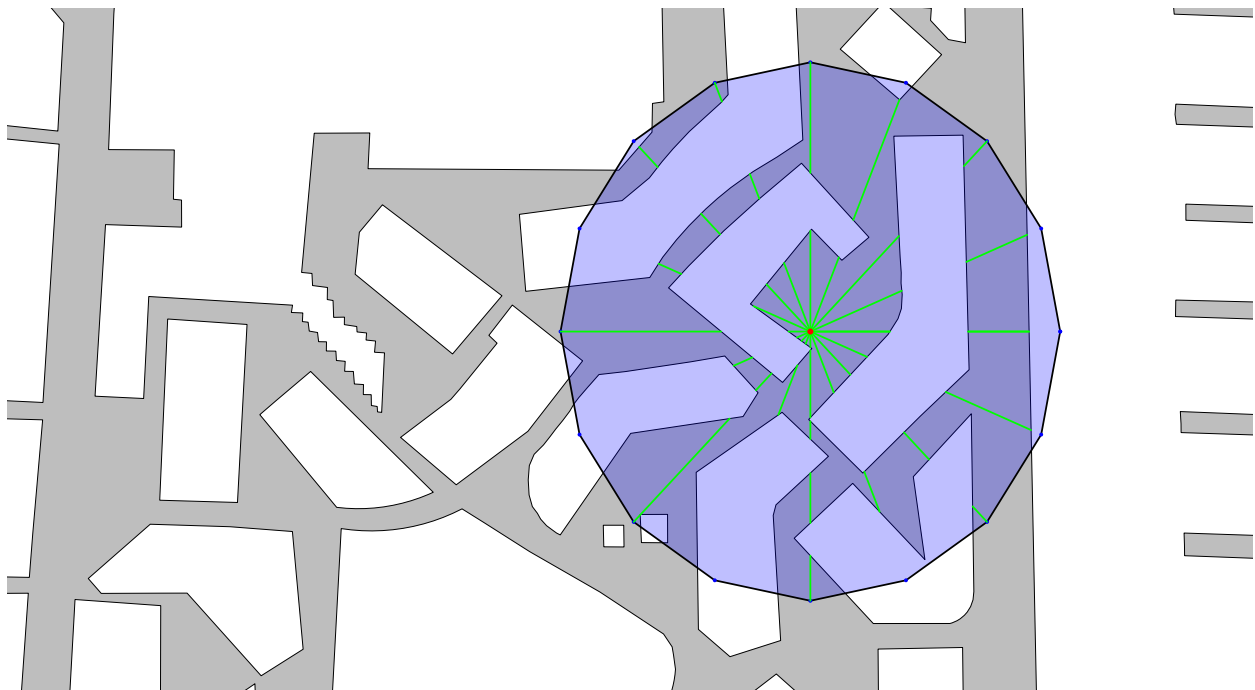
```
interseccion <- st_intersection(rayos_sf, calles) %>%
  st_cast("MULTILINESTRING") %>%
  st_cast("LINESTRING")
```

`## although coordinates are longitude/latitude, st_intersection assumes that they are planar`

```
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
```

```
## Warning in st_cast.sf(., "LINESTRING"): repeating attributes for all sub-
## geometries for which they may not be constant
```

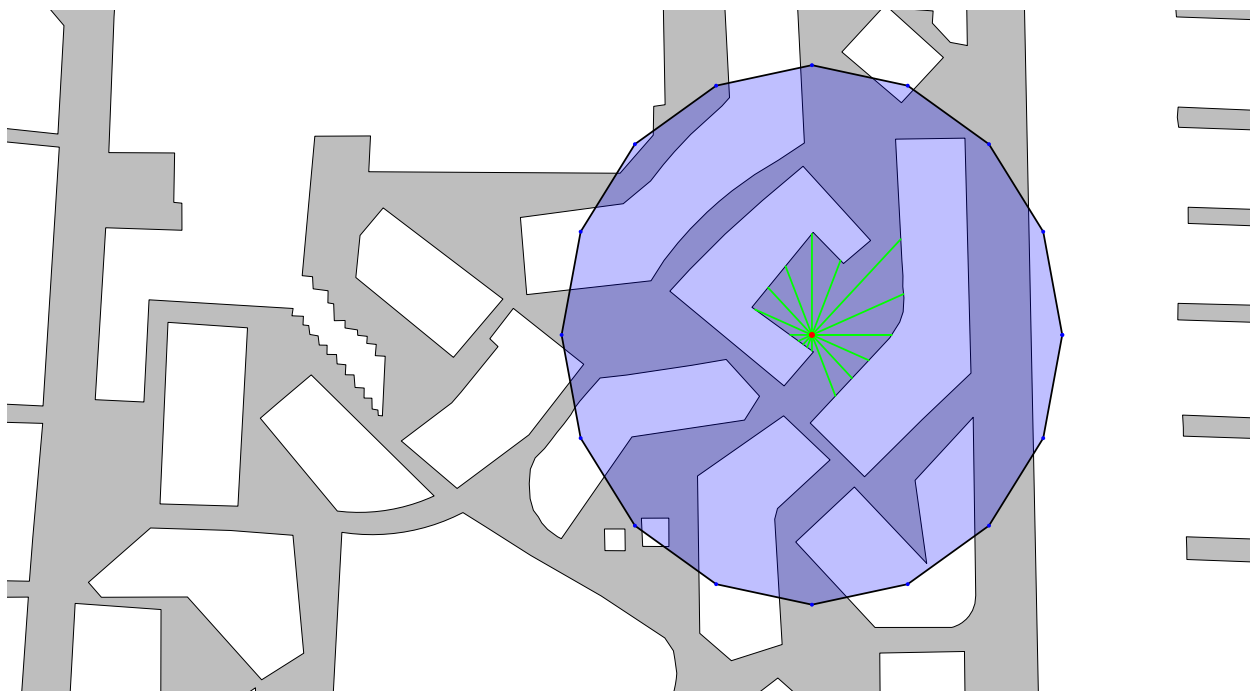
```
par(mar = c(0, 0, 0, 0))
plot(st_geometry(calles), col = "grey", lwd = 0.5, xlim = c(-102.279550, -102.279500), ylim = c(21.8702
plot(st_geometry(buffer), add = T, col = rgb(red = 0, green = 0, blue = 1, alpha = 0.25))
plot(st_geometry(bordes), add = T, col = "blue", pch = 20, cex = 0.25)
plot(st_geometry(interseccion), add = T, col = "green")
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
```



Podemos ver que hay intersecciones de los rayos que no tocan al punto de origen. Osea, cruzan una manzana y siguen de largo. Hay que quitar esos segmentos. Por eso es útil construir los rayos **desde** el centro hacia los bordes, porque al filtrar estas intersecciones y quedarnos solo con la primera geometría, nos quedamos con las intersecciones que queremos.

```
interseccion <- interseccion[!grepl("[:punct:]", rownames(interseccion)), ]
# o de un solo paso:
#interseccion <- st_intersection(rayos_sf, calles) %>% st_cast("LINESTRING")

par(mar = c(0, 0, 0, 0))
plot(st_geometry(calles), col = "grey", lwd = 0.5, xlim = c(-102.279550, -102.279500), ylim = c(21.8702
plot(st_geometry(buffer), add = T, col = rgb(red = 0, green = 0, blue = 1, alpha = 0.25))
plot(st_geometry(bordes), add = T, col = "blue", pch = 20, cex = 0.25)
plot(st_geometry(interseccion), add = T, col = "green")
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
```



Ahora queremos unir los vértices de los rayos. Para esto, obtenemos un conjunto de puntos para las intersecciones y siempre descartamos las primeras geometrías (que corresponden a las coordenadas del punto central) y nos quedamos con los extremos externos de los rayos:

```
pnts <- interseccion %>% st_cast("POINT")
```

```
## Warning in st_cast.sf(., "POINT"): repeating attributes for all sub-geometries
## for which they may not be constant
```

```
pnts <- pnts[grepl("[:punct:]", rownames(pnts)), ]
```

```
par(mar = c(0, 0, 0, 0))
```

```
plot(st_geometry(calles), col = "grey", lwd = 0.5, xlim = c(-102.279550, -102.279500), ylim = c(21.8702
```

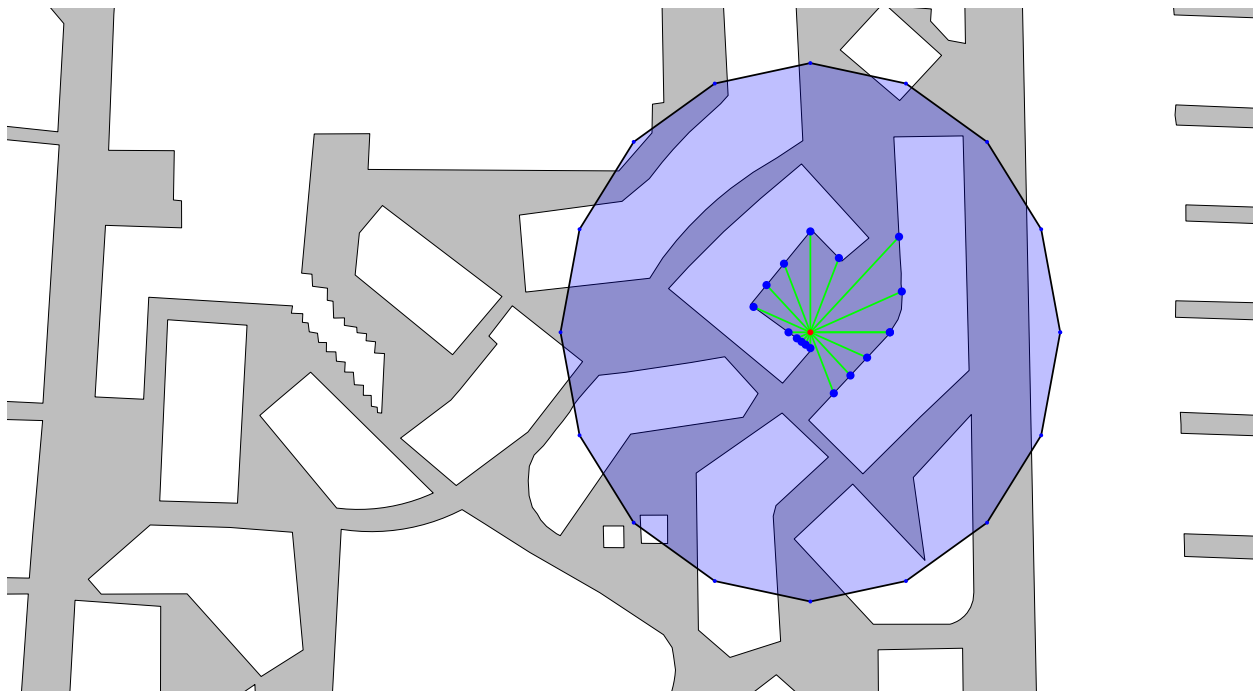
```
plot(st_geometry(buffer), add = T, col = rgb(red = 0, green = 0, blue = 1, alpha = 0.25))
```

```
plot(st_geometry(bordes), add = T, col = "blue", pch = 20, cex = 0.25)
```

```
plot(st_geometry(interseccion), add = T, col = "green")
```

```
plot(st_geometry(pnts), add = T, col = "blue", pch = 20, cex=0.75)
```

```
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
```

Por último, creamos un polígono a partir de los vértices teniendo cuidado de agrupar con respecto a cada punto de origen. Usar `summarize` con el parámetro `do_union = F` es para crear una columna de geometría al combinar los puntos de los distintos grupos y evitar que el orden de los puntos cambie:

```
isovist <- pnts %>% group_by(0.id) %>% summarize(do_union = F) %>% st_cast("POLYGON")
```

```
par(mar = c(0, 0, 0, 0))
```

```
plot(st_geometry(calles), col = "grey", lwd = 0.5, xlim = c(-102.279550, -102.279500), ylim = c(21.8702
```

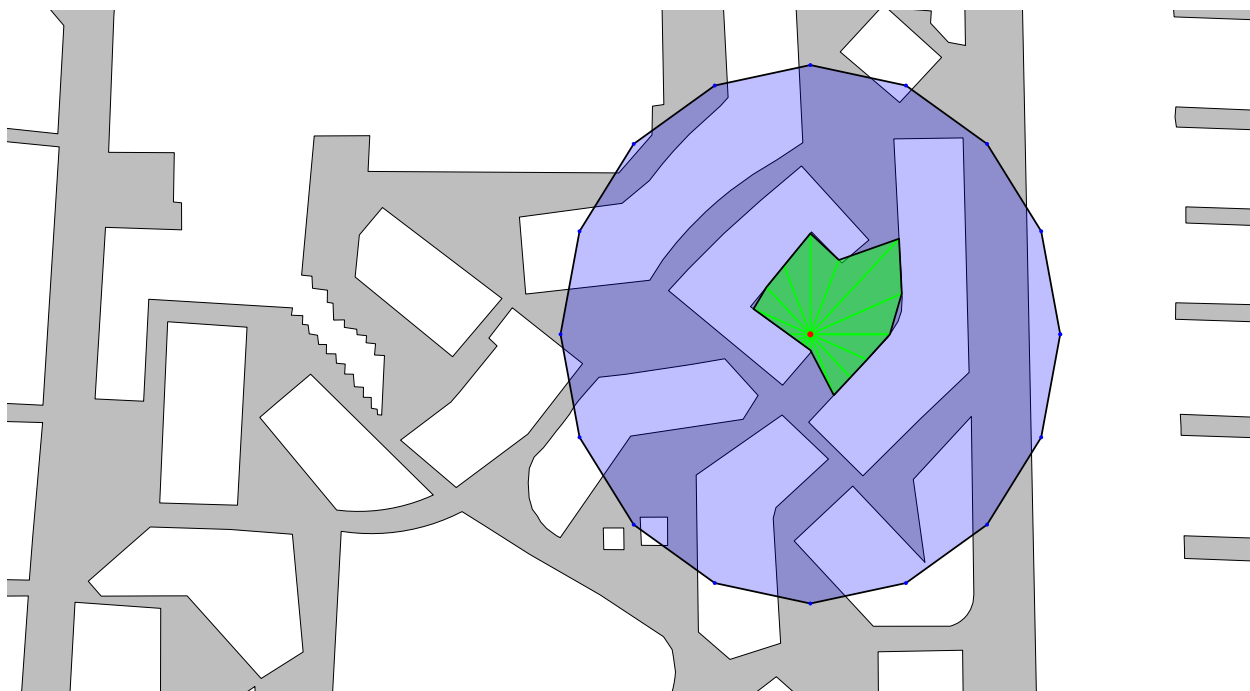
```
plot(st_geometry(buffer), add = T, col = rgb(red = 0, green = 0, blue = 1, alpha = 0.25))
```

```
plot(st_geometry(bordes), add = T, col = "blue", pch = 20, cex = 0.25)
```

```
plot(st_geometry(interseccion), add = T, col = "green")
```

```
plot(st_geometry(isovist), add = T, col = rgb(red = 0, green = 1, blue = 0, alpha = 0.5))
```

```
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
```

Y todo el mapa con isovistas:

```
par(mar = c(0, 0, 0, 0))
plot(st_geometry(calles), col = "grey", lwd = 0.5)
plot(st_geometry(isovist), add = T, col = rgb(red = 0, green = 1, blue = 0, alpha = 0.5))
plot(st_geometry(puntos), add = T, col = "red", pch = 20, cex = 0.5)
```

