shadow: R Package for Geometric Shadow Calculations in an Urban Environment

by Michael Dorman, Evyatar Erell, Adi Vulkan, Itai Kloog

Abstract This paper introduces the **shadow** package for R. The package provides functions for shadow-related calculations in the urban environment, namely shadow height, shadow footprint and Sky View Factor (SVF) calculations, as well as a wrapper function to estimate solar radiation load while taking shadow effects into account. All functions operate on a layer of polygons with a height attribute, also known as "extruded polygons" or 2.5D vector data. Such data are associated with accuracy limitations in representing urban environments. However, unlike 3D models, polygonal layers of building outlines along with their height are abundantly available and their processing does not require specialized closed-source 3D software. The present package thus brings spatio-temporal shadow, SVF and solar radiation calculation capabilities to the open-source spatial analysis workflow in R. Package functionality is demonstrated using small reproducible examples for each function. Wider potential use cases include urban environment applications such as evaluation of micro-climatic influence for urban planning, studying urban climatic comfort and estimating photovoltaic energy production potential.

Introduction

Spatial analysis of the urban environment (Biljecki et al., 2015) frequently requires estimating whether a given point is shaded or not, given a representation of spatial obstacles (e.g. buildings) and a time-stamp with its associated solar position. For example, we may be interested in -

- Calculating the amount of time a given roof or facade is shaded, to determine the utility of installing photovoltaic cells for **electricity production** (e.g. Redweik et al., 2013).
- Calculating shadow footprint on vegetated areas, to determine the expected influence of a tall new building on the surrounding **microclimate** (e.g. Bourbia and Boucheriba, 2010).

Such calculations are usually carried out using GIS-based models (Freitas et al., 2015), in either **vector-based 3D** or **raster-based 2.5D** settings. Both approaches have their advantages and limitations, as discussed in the following paragraphs.

Shadow calculations on vector-based 3D models of the urban environment are mostly restricted to proprietary closed-source software such as **ArcGIS** (ESRI, 2017) or **SketchUp** (@Last and Google, 2017), though recently some open-source models such as **SURFSUN3D** have been developed (Liang et al., 2015). One of the drawbacks of using closed-source software in this context is the difficulty of adjusting the software for specific needs and uncommon scenarios. This problem is especially acute in research settings, where flexibility and extensibility are essential for exploring new computational approaches. The other difficulty with using 3D software in urban spatial analysis concerns interoperability of file formats. Since ordinary vector spatial data formats, such as the *ESRI Shapefile*, cannot represent three-dimensional surfaces, 3D software is associated with specialized file formats. The latter cannot be readily imported to a general-purpose geocomputational environment such as **R** or **Python** (Van Rossum and Drake, 2011), thus fragmenting the analysis workflow. Moreover, most 3D software, such as those mentioned above, are design-oriented, thus providing advanced visualization capabilities but limited quantitative tools (calculating areas, angles, coordinates, etc.). Finally, true-3D models of large urban areas are difficult to obtain, while vector-based 2.5D models (building outline and height, see below) are almost universal. The advantages of true-3D software are "wasted" when the input data are 2.5D, while the disadvantages, such as lack of quantitative procedures and data interoperability difficulties, still remain.

Raster-based 2.5D solutions, operating on a Digital Elevation Model (DEM) raster, are much simpler and have thus been more widely implemented in various software for several decades (Kumar et al., 1997; Ratti and Richens, 2004). For example, raster-based shadow calculations are available in open-source software such as the **r.sun** command (Hofierka and Suri, 2002) in **GRASS GIS** (GRASS Development Team, 2017), the **UMEP** plugin (Lindberg et al., 2018) for **QGIS** (QGIS Development Team, 2017) and package **insol** (Corripio, 2014) in R. In the proprietary **ArcGIS** software, raster-based shadow calculations are provided through the **Solar Analyst** extension (Fu and Rich, 1999). Thanks to this variety of tools, raster-based shadow modelling can be easily incorporated within a general

spatial analysis workflow. However, raster-based models are more suitable for large-scale analysis of natural terrain, rather than fine-scale urban environments, for the following reasons -

- A raster representing surface elevation, known as a DEM, at sufficiently high resolution for the urban context, may not be available and is expensive to produce, e.g. using airborne Light Detection And Ranging (LiDAR) surveys (e.g. Redweik et al., 2013). Much more commonly, municipalities and other sources such as OpenStreetMap (Haklay and Weber, 2008) offer 2.5D vector-based data on cities, i.e. polygonal layers of building outlines associated with height attributes.
- Rasters are composed of pixels, which have no natural association to specific urban elements, such as an individual building, thus making it more difficult to associate analysis results with the corresponding urban elements.
- Vertical surfaces, such as building facades, are rare in natural terrain yet very common in urban environments. Raster-based representation of facades is problematic since the latter correspond to (vertical) discontinuities in the 2.5D digital elevation model, requiring unintuitive workarounds (Redweik et al., 2013).

It should be noted that more specialized approaches have been recently developed to address some of the above-mentioned difficulties, but they are usually not available as software packages (e.g. Redweik et al., 2013; Hofierka and Zlocha, 2012).

The **shadow** package (Dorman, 2018) aims at addressing these limitations by introducing a simple 2.5D vector-based algorithm for calculating shadows, Sky View Factor (SVF) and solar radiation estimates in the urban environment. The algorithms operate on a polygonal layer extruded to 2.5D, also known as *Levels-of-Detail (LoD) 1* in the terminology of the CityGML standard (Gröger and Plümer, 2012). On the one hand, the advantages of individual urban element representation (over raster-based approach) and input data availability (over both raster-based and full 3D approaches) are maintained. On the other hand, the drawbacks of closed-source software and difficult interoperability (as opposed to full 3D environment) are avoided.

As demonstrated below, functions in the **shadow** package operate on a vector layer of obstacle outlines (e.g. buildings) along with their heights, passed as a `SpatialPolygonsDataFrame` object defined in package **sp** (Bivand et al., 2013; Pebesma and Bivand, 2005). The latter makes incorporating shadow calculations in *Spatial* analysis workflow in R straightforward. Functions to calculate shadow height, shadow ground footprint, Sky View Factor (SVF) and radiation load are implemented in the package.

Theory

Shadow height

All functions currently included in **shadow** are based on trigonometric relations in the triangle defined by the sun's rays, the ground - or a plane parallel to the ground - and an obstacle.

For example, **shadow height** at any given ground point can be calculated based on (1) sun elevation, (2) the height of the building(s) that stand in the way of sun rays and (3) the distance(s) between the queried point and the building(s) along the sun rays projection on the ground. Figure 1 depicts a scenario where shadow is being cast by building A onto the facade of building B, given the solar position defined by its elevation angle α_{elev} and azimuth angle α_{az}. Once the intersection point is identified (marked with x in Figure 1), shadow height (h_{shadow}) at the queried point (*viewer*) can be calculated based on (1) sun elevation (α_{elev}), (2) the height of building B (h_{build}) and (3) the distance ($dist_1$) between the *viewer* and intersection point x (Equation 1).

$$h_{shadow} = h_{build} - dist_1 \cdot tan(\alpha_{elev}) \tag{1}$$

The latter approach can be extended to the general case of shadow height calculation at any ground location and given any configuration of obstacles. For example, if there is more than one obstacle potentially casting shadow on the queried location, we can calculate h_{shadow} for each obstacle and then take the maximum value.

Logical shadow flag

Once the shadow height is determined, we may evaluate whether any given 3D point is in shadow or not. This is done simply by comparing the Z-coordinate (i.e. height) of the queried point with the calculated shadow height at the same X-Y (i.e. ground) location.

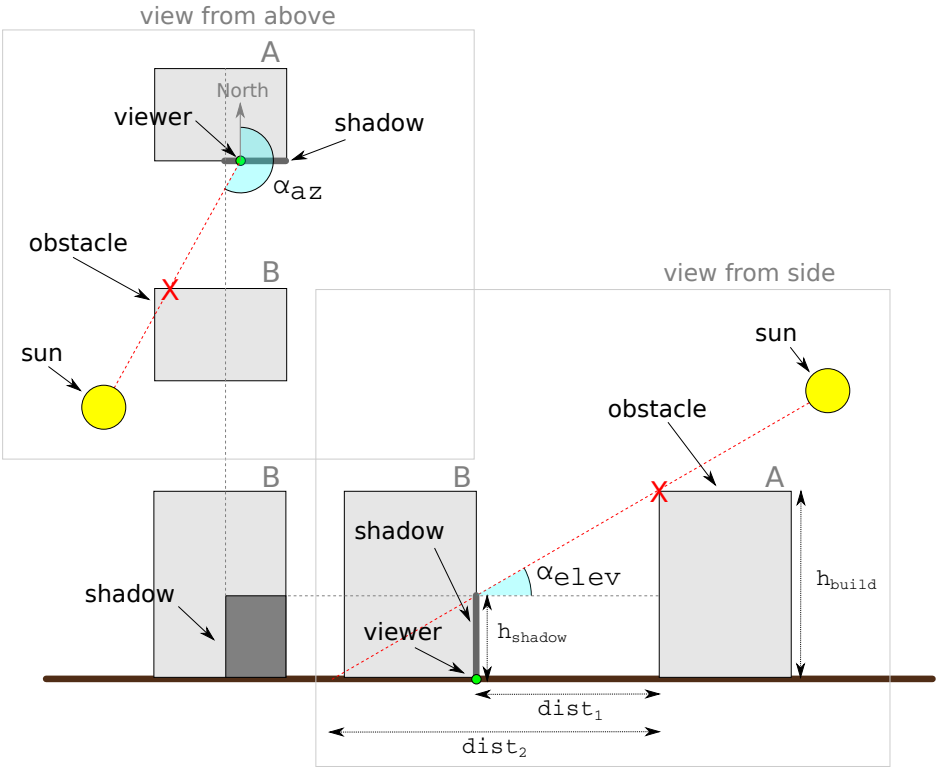

Figure 1: Shadow height calculation

Shadow footprint

Instead of calculating shadow height at a pre-specified point (e.g. the *viewer* in Figure 1), we can set h_{shadow} to zero and calculate the distance ($dist_2$) where the shadow intersects ground level (Equation 2).

$$dist_2 = \frac{h_{build}}{tan(\alpha_{elev})} \tag{2}$$

Shifting the obstacle outline by the resulting distance ($dist_2$) in a direction opposite to sun azimuth (α_{az}) yields a **shadow footprint** outline (Weisthal, 2014). Shadow footprints are useful to calculate the exact ground area that is shaded at specific time. For example, Figure 2 shows the shadow footprints produced by a single building at different times of a given day.

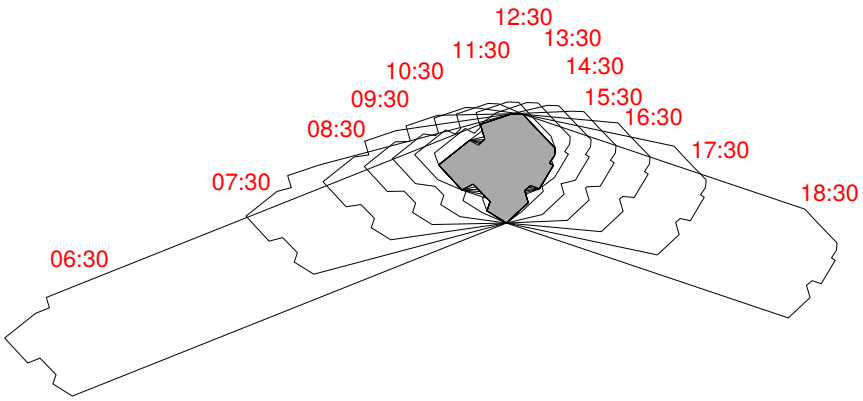

Figure 2: Shadow footprints cast by a building on a horizontal ground surface at hourly intervals on 2004-06-24. The building, indicated by the gray shaded area, is located at 31.97°N 34.78°E, and is 21.38 meters tall

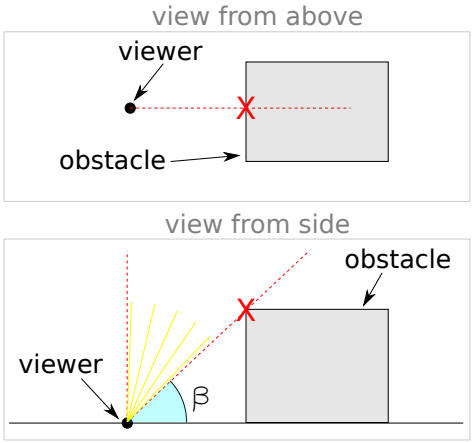

Figure 3: Sky View factor calculation

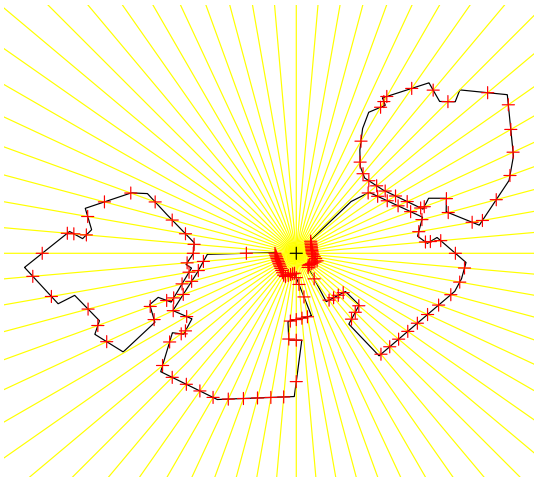

Figure 4: Angular cross sections for calculating the Sky View Factor (SVF)

Sky View Factor (SVF)

The **Sky View Factor** (Beckers, 2013; Erell et al., 2011; Grimmond et al., 2001) is the extent of sky observed from a point as a proportion of the entire sky hemisphere. The SVF can be calculated based on the maximal angles (β) formed in triangles defined by the queried location and the obstacles (Figure 3), evaluated in multiple circular cross-sections surrounding the queried location. Once the maximal angle β_i is determined for a given angular section i, SVF_i for that particular section is defined (Gál and Unger, 2014) in Equation 3.

$$SVF_i = 1 - sin^2(\beta_i) \tag{3}$$

For example, in case ($\beta_i = 45°$), as depicted in Figure 3, SVF_i is equal to -

$$SVF_i = 1 - sin^2(45°) = 0.5$$

Averaging SVF_i values for all $i = 1, 2, ..., n$ circular cross-sections gives the final SVF estimate for the queried location (Equation 4).

$$SVF = \frac{\sum_{i=1}^{n} SVF_i}{n} \tag{4}$$

The number of evaluated cross sections depends on the chosen angular resolution. For example, an angular resolution of 5° means the number of cross sections is $n = 360°/5° = 72$ (Figure 4).

Solar radiation

Components

Frequently, evaluating whether a given location is shaded, and when, is just a first steps towards evaluating the solar radiation load for a given period of time. The annual insolation at a given point is naturally affected by the degree of shading throughout the year, but shading is not the only factor.

The three components of the solar radiation are the **direct**, **diffuse** and **reflected** radiation -

- **Direct** radiation refers to solar radiation traveling on a straight line from the sun to the surface of the earth. Direct radiation can be estimated by taking into account: (1) shading, (2) surface orientation relatively to the sun, and (3) meteorological measurements of direct radiation on a horizontal plane or on a plane normal to the beam of sunlight.
- **Diffuse** radiation refers to solar radiation reaching the Earth's surface after having been scattered from the direct solar beam by molecules or particulates in the atmosphere. Diffuse radiation can be estimated by taking into account: (1) SVF, and (2) meteorological measurements of diffuse radiation at an exposed location.
- **Reflected** radiation refers to the sunlight that has been reflected off non-atmospheric obstacles such as ground surface cover or buildings. Most urban surfaces have a low albedo: asphalt reflects only 5-10 percent of incident solar radiation, brick and masonry 20-30 percent, and vegetation about 20 percent. Because a dense urban neighborhood will typically experience multiple reflections, an iterative process is required for a complete analysis. Calculating reflected radiation requires taking into account reflective properties of the various surfaces, their geometrical arrangement (Givoni, 1998) and their view factors from the receiving surface, which is beyond the scope of the **shadow** package.

The diffuse radiation component is the dominant one on overcast days, when most radiation is scattered, while the direct radiation component is dominant under clear sky conditions when direct radiation reaches the earth's surface.

Direct Normal Irradiance

Equation 5 specifies the Coefficient of Direct Normal Irradiance for a vertical **facade** surface, as function of solar position given by the difference between facade azimuth and sun azimuth angles, and sun elevation angle, at time t.

$$\theta_{facade,t} = cos(\alpha_{az,t} - \alpha'_{az}) \cdot cos(\alpha_{elev,t}) \tag{5}$$

Where $\theta_{facade,t}$ is the Coefficient of Direct Normal Irradiance on a facade at time t, $\alpha_{az,t}$ is the sun azimuth angle at time t (see Figure 1), α'_{az} is the facade azimuth angle, i.e. the direction where the facade is facing, and $\alpha_{elev,t}$ is sun elevation angle at time t (see Figure 1). Note that all of latter variables, with the exception of facade azimuth angle α'_{az}, are specific for the time interval t due to the variation in solar position.

Horizontal roof surfaces, unlike facades, are not tilted towards any particular azimuth[1]. Equation 5 thus simplifies to Equation 6 when referring to a **roof**, rather than a facade, surface.

$$\theta_{roof,t} = cos(90^\circ - \alpha_{elev,t}) \tag{6}$$

Figure 5 demonstrates the relation given in Equations 5 and 6 for the entire relevant range of solar positions relative to facade or roof orientation. Again, note that for roof surfaces, the $\theta_{roof,t}$ coefficient is only dependent on sun elevation angle $\alpha_{elev,t}$ (Equation 6) as illustrated on the **right** panel of Figure 5. (The code for producing Figure 5 can be found in the help page of function `coefDirect` from **shadow**).

For example, the **left** panel in Figure 5 shows that maximal proportion of incoming solar radiation load (i.e. $\theta_{facade,t} = 1$) on a facade surface is attained when facade azimuth is equal to sun azimuth and sun elevation is 0 ($\alpha_{elev,t} = 0^\circ$, i.e. facade directly facing the sun). Similarly, the right panel shows that maximal radiation load on a roof surface (i.e. $\theta_{roof,t} = 1$) is attained when the sun is at the zenith ($\alpha_{elev,t} = 90^\circ$, i.e. sun directly above the roof).

Once the Coefficient of Direct Normal Irradiance $\theta_{facade,t}$ or $\theta_{roof,t}$ is determined, the Direct Normal Irradiance meteorological measurement $rad_{direct,t}$ referring to the same time interval t, usually on an

[1]It should be noted that roof surfaces may be pitched rahter than horizontal; however 2.5D models, which **shadow** supports, can only represent horizontal roofs

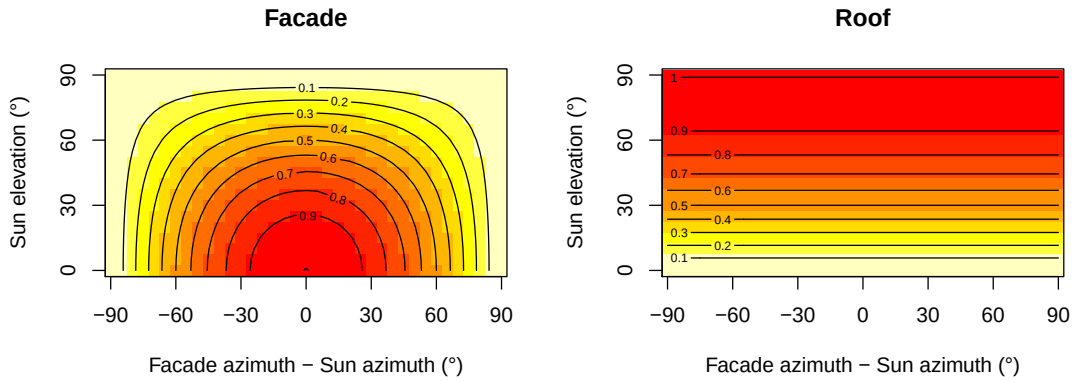

Figure 5: Coefficient of Direct Normal Irradiance, as function of solar position, expressed as the difference between facade and sun azimuths (X-axis) and sun elevation (Y-axis). The **left** panel refers to a facade, the **right** panel refers to a roof. Note that a roof has no azimuth, thus the X-axis is irrelevant for the **right** panel and only shown for uniformity

hourly time step, is multiplied by the coefficient at a point on the building surface to give the local irradiation at that point (Equation 7). The result $rad'_{direct,t}$ is the corrected Direct Irradiance the surface receives given its orientation relative to the solar position.

$$rad'_{direct,t} = \theta_t \cdot rad_{direct,t} \tag{7}$$

Both $rad_{direct,t}$ and $rad'_{direct,t}$, as well as $rad_{diffuse,t}$, $rad'_{diffuse,t}$ (Equation 8) and rad_{total} (Equation 9) (see below), are given for each time interval t in units of power per unit area, such as kWh/m^2.

Diffuse Horizontal Irradiance

Moving on to discussing the second component in the radiation balance, the diffuse irradiance. Diffuse irradiance is given by the meteorological measurement of Diffuse Horizontal Irradiance $rad_{diffuse,t}$, which needs to be corrected for the specific proportion of viewed sky given surrounding obstacles expressed by SVF. Assuming isotropic contribution (Freitas et al., 2015), $rad'_{diffuse,t}$ is the corrected diffuse irradiance the surface receives (Equation 8). Note that SVF is unrelated to solar position; it is a function of the given configuration of the queried location and surrounding obstacles, and is thus invariable for all time intervals t.

$$rad'_{diffuse,t} = SVF \cdot rad_{diffuse,t} \tag{8}$$

Total irradiance

Finally, the direct and diffuse radiation estimates are summed for all time intervals t to obtain the total (e.g. annual) insolation for the given surface rad_{total} (Equation 9). The sum refers to n intervals $t = 1, 2, ..., n$, commonly $n = 24 \times 365 = 8,760$ when referring to an annual radiation estimate using an hourly time step.

$$rad_{total} = \sum_{t=1}^{n} rad'_{direct,t} + \sum_{t=1}^{n} rad'_{diffuse,t} \tag{9}$$

Package structure

The **shadow** package contains four "low-level" functions, one "high-level" function, and several "helper functions".

The "low-level" functions calculate distinct aspects of shading, and the SVF -

- `shadowHeight` - Calculates shadow height
- `inShadow` - Determines a logical shadow flag (in shadow or not)
- `shadowFootprint` - Calculates shadow footprint
- `SVF` - Calculates the SVF

Function	Location	Obstacles	Sun Pos.	Output
`shadowHeight`	Points (2D) / Raster	Polygons	Matrix	Numeric matrix / Raster
`inShadow`	Points (2D/3D) / Raster	Polygons	Matrix	Logical matrix / Raster
`shadowFootprint`	-	Polygons	Matrix	Polygons
`SVF`	Points (2D/3D) / Raster	Polygons	-	Numeric vector / Raster

Table 1: Inputs and outputs for main functions in package `shadow`

Table 1 gives a summary of the (main) input and output object types for each of the "low-level" functions. The following list clarifies the exact object classes referenced in the table -

- The queried locations **points** (e.g. the *viewer* point in Figure 1) can be specified in several ways. Points (`SpatialPoints*`) can be either 2D, specifying ground locations, or 3D[2] - specifying any location on the ground or above ground. Alternatively, a **raster** (`Raster*`) can be used to specify a regular grid of ground locations. Note that the shadow height calculation only makes sense for ground locations, as height above ground is what the function calculates, so it is not applicable for 3D points
- The obstacle **polygons** are specified as a `SpatialPolygonsDataFrame` object having a **height** attribute ("extrusion" height) given in the same units as the layer Coordinate Reference System (CRS), usually meters. Geographic coordinates (long/lat) are not allowed because these units are meaningless for specifying height
- Solar position **matrix** is given as a `matrix` object, where the first column specifies **sun azimuth** angle and the second column specifies **sun elevation** angle. Both angles should be given in decimal degrees, where -
  - **sun azimuth** (e.g. α_{az} in Figure 1) is measured clockwise relative to North, i.e North = 0°, East = 90°, South = 180°, West = 270°
  - **sun elevation** (e.g. α_{elev} in Figure 1) is measured relatively to a horizontal surface, i.e. sun on the horizon = 0°, sun at its zenith = 90°
- The **output** of `shadowHeight` and `inShadow` is a numeric or logical `matrix`, repectively, where rows represent locations and columns represent times or solar positions. The output of `shadowFootprint` is a polygonal layer of footprints. The output of `SVF` is a numeric vector where values correspond to locations. All functions that can accept a raster of ground locations return a corresponding raster of computed values

The "high-level" function `radiation` is a wrapper around `inShadow` and `SVF` for calculating direct and diffuse radiation load on the obstacle surface area (i.e. building roofs and facades). In addition to the geometric layers, this function also requires meteorological measurements of direct and diffuse radiation at an unobstructed weather station. The package provides a sample Typical Meteorological Year (TMY) dataset `tmy` to illustrate the usage of the `radiation` function (see below). Similar TMY datasets were generated for many areas (e.g. Pusat et al., 2015) and are generally available from meteorological agencies or from databases for building energy simulation such as EnergyPlus (EnergyPlus, 2018).

Finally, the **shadow** package provides several "helper functions" which are used internally by "low-level" and "high-level" functions, but can also be used independently -

- `classifyAz` - Determines the azimuth where the perpendicular of a line segment is facing. Used internally to classify facade azimuth.
- `coefDirect` - Calculates the Coefficient of Direct Normal Irradiance reduction (Equations 5 and 6)
- `plotGrid` - Makes an interactive plot of 3D spatial points. This is a wrapper around `scatterplot3js` from package **threejs** (Lewis, 2017)
- `ray` - Creates a spatial line between two given points
- `shiftAz` - Shifts spatial features by azimuth and distance
- `surfaceGrid` - Creates a 3D point layer with a grid which covers the facades and roofs of obstacles
- `toSeg` - Splits polygons or lines to segments

The following section provides a manual for using these functions through a simple example with four buildings.

[2]The third dimension of 3D points has to be specified using three-dimensional *coordinates*, rather than a "height" attribute in a 2D point layer (see Examples section)

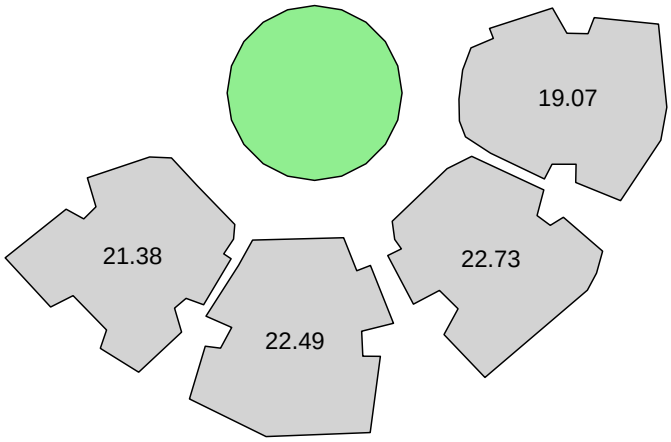

Figure 6: Sample data: a buildings layer and a green park layer. Text labels express building height in meters.

Examples

In this section we demonstrate the main functionality of **shadow**, namely calculating -

- Shadow height (function `shadowHeight`)
- Logical shadow flag (function `inShadow`)
- Shadow footprint (function `shadowFootprint`)
- Sky View Factor (function `SVF`)
- Radiation load (function `radiation`)

Before going into the examples, we load the **shadow** package. Package **sp** is loaded automatically along with **shadow**. Packages **raster** (Hijmans, 2017) and **rgeos** (Bivand and Rundel, 2017) are used throughout the following code examples for preparing the inputs and presenting the results, so they are loaded as well.

```
> library(shadow)
> library(raster)
> library(rgeos)
```

In the examples, we will use a small real-life dataset representing four buildings in Rishon-Le-Zion, Israel (Figure 6), provided with package **shadow** and named `build`.

The following code section also creates a hypothetical circular green park located 20 meters to the north and 8 meters to the west from the buildings layer centroid (hereby named `park`).

```
> location = gCentroid(build)
> park_location = shift(location, y = 20, x = -8)
> park = gBuffer(park_location, width = 12)
```

The following expressions visualize the `build` and `park` layers as shown in Figure 6. Note that the `build` layer has an attribute named `BLDG_HT` specifying the height of each building (in meters), as shown using text labels on top of each building outline.

```
> plot(build, col = "lightgrey")
> text(gCentroid(build, byid = TRUE), build[["BLDG_HT"]])
> plot(park, col = "lightgreen", add = TRUE)
```

Shadow height

The `shadowHeight` function calculates shadow height(s) at the specified point location(s), given a layer of obstacles and solar position(s). The `shadowHeight` function, as well as other functions that require a solar position argument such as `inShadow`, `shadowFootprint` and `radiation` (see below), alternatively accept a time argument instead of the solar position. In case a time (`time`) argument is passed instead of solar position (`solar_pos`), the function internally calculates solar position using the lon/lat of

the `location` layer centroid and the specified time, using function `solarpos` from package **maptools** (Bivand and Lewin-Koh, 2017).

In the following example, we would like to calculate shadow height at the centroid of the buildings layer (`build`) on 2004-12-24 at 13:30:00. First we create the queried points layer, in this case consisting of a single point, the `build` layer centroid. This is our layer of locations where we would like to calculate shadow height.

```
> location = gCentroid(build)
```

Next we need to specify the solar position, i.e. sun elevation and azimuth, at the particular time and location (31.967°N 34.777°E), or let the function calculate it automatically based on the time. Using the former option, we can figure out solar position using function `solarpos` from package **maptools**. To do that, we first define a `POSIXct` object specifying the time we are interested in -

```
> time = as.POSIXct(
+   x = "2004-12-24 13:30:00",
+   tz = "Asia/Jerusalem"
+ )
```

Second, we find the longitude and latitude of the point by reprojecting it to a geographic CRS[3].

```
> location_geo = spTransform(
+   x = location,
+   CRSobj = "+proj=longlat +datum=WGS84"
+ )
```

Finally, we use the `solarpos` function to find solar position, given longitude, latitude and time -

```
> library(maptools)
> solar_pos = solarpos(
+   crds = location_geo,
+   dateTime = time
+ )
```

We now know the sun azimuth (208.7°) and elevation (28.8°) -

```
> solar_pos

#>          [,1]     [,2]
#> [1,] 208.7333 28.79944
```

Given the solar position along with the layer of obstacles `build`, shadow height in `location` can be calculated using the `shadowHeight` function as follows -

```
> h = shadowHeight(
+   location = location,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos
+ )
```

The resulting object contains the shadow height value of 19.86 meters -

```
> h

#>          [,1]
#> [1,] 19.86451
```

The second (shorter) approach is letting the function calculate solar position for us, in which case we can pass just the spatial layers and the `time`, without needing to calculate solar position ourselves -

```
> shadowHeight(
+   location = location,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   time = time
+ )
```

[3]Note that calculating solar position is the only example where lon/lat coordinates are needed when working with **shadow**. All other spatial inputs are required to be passed in a projected CRS, due to the fact that obstacles height is meaningless to specify in lon/lat degree units

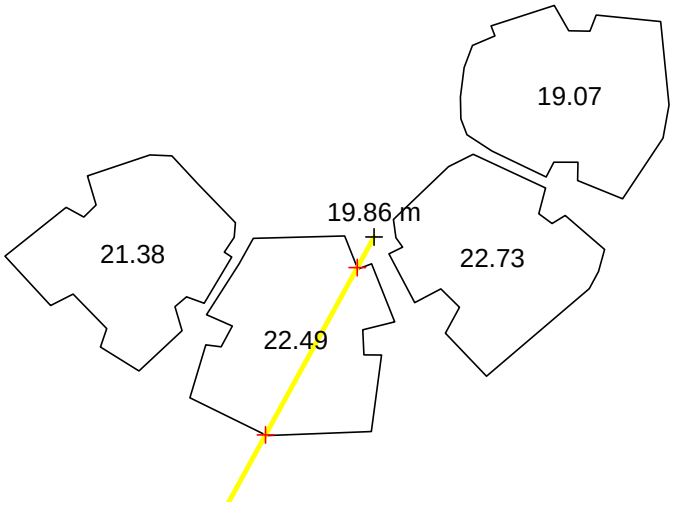

Figure 7: Shadow height (m) at a single point (indicated by black + symbol)

```
#>          [,1]
#> [1,] 19.86451
```

The results of both approaches are identical. The first approach, where solar position is manually defined, takes more work and thus may appear unnecessary. However, it is useful for situations when we want to use specific solar positions from an external data source, or to evaluate arbitrary solar positions that cannot be observed in the queried location in real life.

Either way, the resulting object `h` is a `matrix`, though in this case it only has a single row and a single column. The `shadowHeight` function accepts location layers with more than one point, in which case the resulting `matrix` will have additional rows. It also accepts more than one solar position or time value (see below), in which case the resulting `matrix` will have additional columns. It is thus possible to obtain a matrix of shadow height values for a set of locations in a set of times.

Figure 7 illustrates how the shadow height calculation was carried out. First, a line of sight is drawn between the point of interest and the sun direction based on its azimuth (shown as a yellow line). Next, potential intersections are detected (shown as red + symbols). Finally, shadow height induced by each intersection is calculated based on the distance towards intersection, sun elevation and intersected building height (see Figure 1). The final result is the maximum of the per-intersection heights.

The procedure can be readily expanded to calculate a continuous surface of shadow heights, as the `shadowHeight` function also accepts `Raster*` objects (package **raster**). The raster serves as a template, defining the grid where shadow height values will be calculated. For example, in the following code section we create such a template raster covering the examined area plus a 50-meter buffer on all sides, with a spatial resolution of 2 meters -

```
> ext = as(extent(build) + 50, "SpatialPolygons")
> r = raster(ext, res = 2)
> proj4string(r) = proj4string(build)
```

Now we can calculate a shadow height raster by simply replacing the `location` argument with the raster `r` -

```
> height_surface = shadowHeight(
+   location = r,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos,
+   parallel = 5
+ )
```

The result, in this case, is not a matrix - it is a shadow height surface (`RasterLayer` object) of the same spatial dimensions as the input template `r` (Figure 8). Note that unshaded pixels get an `NA` shadow height value, thus plotted in white. Also note the partial shadow on the roof of the north-eastern building (top-right) caused by the neighboring building to the south-west.

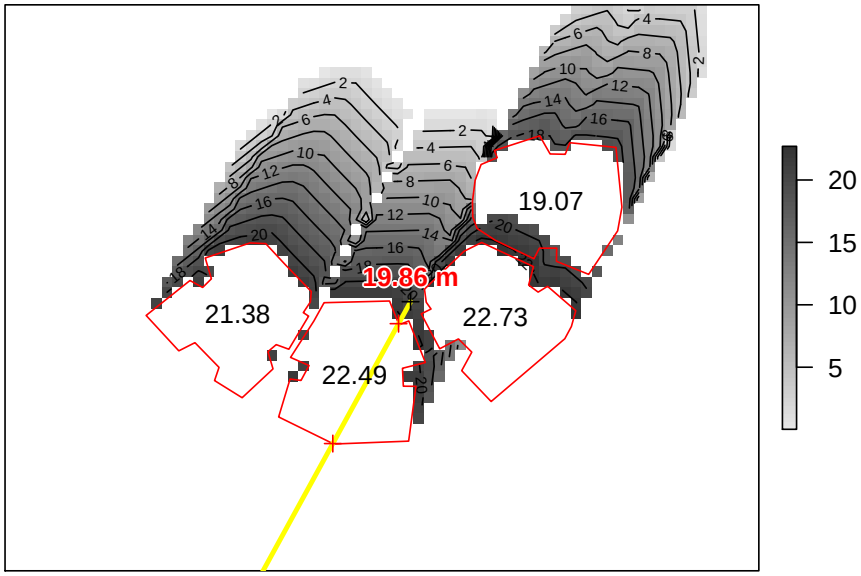

Figure 8: Shadow height (m) surface, and an individual shadow height value (indicated by black + symbol at the center of the image)

The additional `parallel=5` argument splits the calculation of raster cells among 5 processor cores, thus making it faster. A different number can be specified, depending the number of available cores. Behind the scenes, parallel processing relies on the **parallel** package (R Core Team, 2018).

Shadow (logical)

Function `shadowHeight`, introduced in the previous section, calculates *shadow height* for a given ground location. In practice, the metric of interest is very often *whether* a given 3D location is in shade or not. Such a logical flag can be determined by comparing the Z-coordinate (i.e. the height) of the queried point with the calculated shadow height at the same X-Y location. The `inShadow` function is a wrapper around `shadowHeight` for doing that.

The `inShadow` function gives the logical shadow/non-shadow classification for a set of 3D points. The function basically calculates shadow height for a given unique ground location (X-Y), then compares it with the elevation (Z) of all points in that location. The points which are positioned "above" the shadow are considered non-shaded (receiving the value of `FALSE`), while the points which are positioned "below" the shadow are considered shaded (receiving the value of `TRUE`).

The 3D points we are interested in when doing urban analysis are usually located on the surface of elements such as buildings. The `surfaceGrid` helper function can be used to automatically generate a **grid** of such surface points. The inputs for this function include the obstacle layer for which to generate a surface grid and the required grid resolution. The returned object is a 3D point layer.

For example, the following expression calculates a 3D point layer named `grid` covering the `build` surface at a resolution of 2 meters -

```
> grid = surfaceGrid(
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   res = 2
+ )
```

The resulting grid points are associated with all attributes of the original obstacles each surface point corresponds to, as well as six new attributes:

- `obs_id` - Unique consecutive ID for each feature in `obstacles`
- `type` - Either `"facade"` or `"roof"`
- `seg_id` - Unique consecutive ID for each facade segment (only for "facade" points)
- `xy_id` - Unique consecutive ID for each ground location (only for "facade" points)
- `facade_az` - The azimuth of the corresponding facade, in decimal degrees (only for "facade" points)

In this case, the resulting 3D point grid has 2,688 features, starting with `"roof"` points -

```
> head(grid)

#>   build_id BLDG_HT obs_id type seg_id xy_id facade_az
#> 1      722   22.49      3 roof     NA    NA        NA
#> 2      722   22.49      3 roof     NA    NA        NA
#> 3      722   22.49      3 roof     NA    NA        NA
#> 4      722   22.49      3 roof     NA    NA        NA
#> 5      722   22.49      3 roof     NA    NA        NA
#> 6      722   22.49      3 roof     NA    NA        NA
```

Then going through the "facade" points -

```
> tail(grid)

#>       build_id BLDG_HT obs_id   type seg_id xy_id facade_az
#> 19610      831   19.07      4 facade     74    44  100.2650
#> 19710      831   19.07      4 facade     75    45  123.6695
#> 19810      831   19.07      4 facade     75    46  123.6695
#> 19910      831   19.07      4 facade     75    47  123.6695
#> 20010      831   19.07      4 facade     75    48  123.6695
#> 20110      831   19.07      4 facade     75    49  123.6695
```

Printing the coordinates confirms that, indeed, `grid` is a 3D point layer having three-dimensional coordinates where the third dimension `h` represents height above ground -

```
> head(coordinates(grid))

#>         x1      x2    h
#> 1 667887.5 3538084 22.5
#> 2 667889.5 3538084 22.5
#> 3 667883.5 3538086 22.5
#> 4 667885.5 3538086 22.5
#> 5 667887.5 3538086 22.5
#> 6 667889.5 3538086 22.5
```

Once the 3D grid is available, we can evaluate whether each point is in shadow or not, at the specified solar position(s), using the `inShadow` wrapper function -

```
> s = inShadow(
+   location = grid,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos
+ )
```

The resulting object is a `logical` matrix with rows corresponding to the grid features and columns corresponding to the solar positions. In this particular case a single solar position was evaluated, thus the matrix has just one column -

```
> dim(s)

#> [1] 2688    1
```

The `scatter3D` function from package **plot3D** is useful for visualizing the result. In the following code section, we use two separate `scatter3D` function calls to plot the grid with both variably colored filled circles (yellow or grey) and constantly colored (black) outlines.

```
> library(plot3D)
> scatter3D(
+   x = coordinates(grid)[, 1],
+   y = coordinates(grid)[, 2],
+   z = coordinates(grid)[, 3],
+   theta = 55,
+   colvar = s[, 1],
+   col = c("yellow", "grey"),
+   pch = 16,
+   scale = FALSE,
```

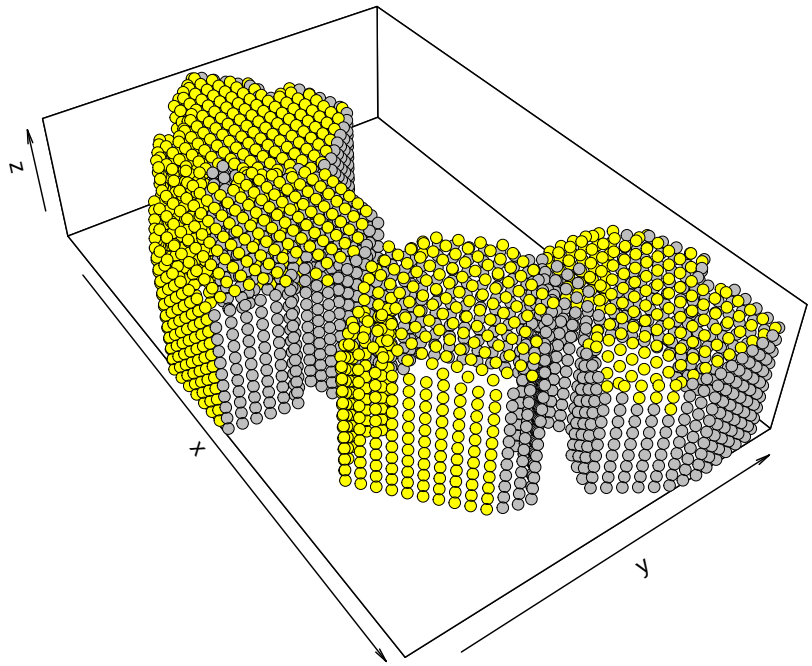

Figure 9: Buildings surface points in shadow (grey) and in direct sunlight (yellow) on 2004-12-24 13:30:00

```
+   colkey = FALSE,
+   cex = 1.1
+ )
> scatter3D(
+   x = coordinates(grid)[, 1],
+   y = coordinates(grid)[, 2],
+   z = coordinates(grid)[, 3],
+   theta = 55,
+   col = "black",
+   pch = 1,
+   lwd = 0.1,
+   scale = FALSE,
+   colkey = FALSE,
+   cex = 1.1,
+   add = TRUE
+ )
```

The output is shown in Figure 9. It shows the 3D `grid` points, along with the `inShadow` classification encoded as point color: grey for shaded surfaces, yellow for sun-exposed surfaces.

Shadow footprint

The `shadowFootprint` function calculates the geometry of shadow projection on the ground. The resulting footprint layer can be used for various applications. For example, a shadow footprint layer can be used to calculate the proportion of shaded surface in a defined area, or to examine which obstacles are responsible for shading a given urban element.

In the following example, the `shadowFootprint` function is used to determine the extent of shading on the hypothetical green park (Figure 6) at different times of day. First, let us consider a single time instance of 2004-06-24 09:30:00. At this particular time and geographical location, the solar position is at an azimuth of 88.8° and at an elevation of 46.7° -

```
> time2 = as.POSIXct(
+   x = "2004-06-24 09:30:00",
+   tz = "Asia/Jerusalem"
+ )
> solar_pos2 = solarpos(
+   crds = location_geo,
```

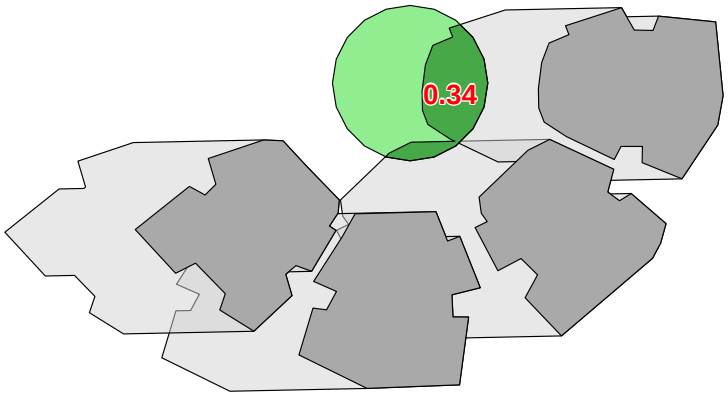

Figure 10: Shaded park proportion on 2004-06-24 09:30:00

```
+   dateTime = time2
+ )
> solar_pos2

#>          [,1]   [,2]
#> [1,] 88.83113 46.724
```

The following expression calculates the shadow footprint for this particular solar position.

```
> footprint = shadowFootprint(
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos2
+ )
```

The resulting object `footprint` is a polygonal layer (`SpatialPolygonsDataFrame` object) which can be readily used in other spatial calculations. For example, the footprint and park polygons can be *intersected* to calculate the proportion of shaded park area within total park area, as follows.

```
> park_shadow = gIntersection(park, footprint)
> shade_prop = gArea(park_shadow) / gArea(park)
> shade_prop

#> [1] 0.3447709
```

The numeric result `shade_prop` gives the proportion of shaded park area, 0.34 in this case (Figure 10).

The shadow footprint calculation can also be repeated for a sequence of times, rather than a single one, to monitor the daily (monthly, annual, etc.) course of shaded park area proportion. To do that, we first need to prepare the set of solar positions in the evaluated dates/times. Again, this can be done using function `solarpos`. For example, the following code creates a matrix named `solar_pos_seq` containing solar positions over the 2004-06-24 at hourly intervals -

```
> time_seq = seq(
+   from = as.POSIXct("2004-06-24 03:30:00", tz = "Asia/Jerusalem"),
+   to = as.POSIXct("2004-06-24 22:30:00", tz = "Asia/Jerusalem"),
+   by = "1 hour"
+ )
> solar_pos_seq = solarpos(
+   crds = location_geo,
+   dateTime = time_seq
+ )
```

Note that the choice of an *hourly* interval is arbitrary. Shorter intervals (e.g. 30 mins) can be used for increased accuracy.

To calculate the shaded park proportion at each time step we can loop over the `solar_pos_seq` matrix, each time -

- Calculating shadow footprint

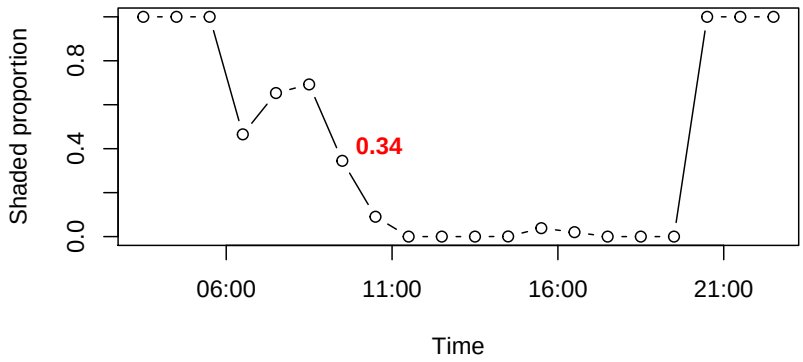

Figure 11: Shaded park proportion at each hourly time step on 2004-06-24

- Intersecting the shadow footprint with the park outline
- Calculating the ratio of intersection and total park area

The code of such a `for` loop is given below.

```
> shadow_props = rep(NA, nrow(solar_pos_seq))
> for(i in 1:nrow(solar_pos_seq)) {
+   if(solar_pos_seq[i, 2] < 0) shadow_props[i] = 1 else {
+       footprint =
+         shadowFootprint(
+           obstacles = build,
+           obstacles_height_field = "BLDG_HT",
+           solar_pos = solar_pos_seq[i, , drop = FALSE]
+           )
+     park_shadow = gIntersection(park, footprint)
+     if(is.null(park_shadow))
+       shadow_props[i] = 0
+     else
+       shadow_props[i] = gArea(park_shadow) / gArea(park)
+     }
+ }
```

The loop creates a numeric vector named `shadow_props`. This vector contains shaded proportions for the park in agreement with the times we specified in `time_seq`. Note that two conditional statements are being used to deal with special cases -

- Shadow proportion is set to 1 (i.e. maximal) when sun is below the horizon and the result of `shadowFootprint` is `NULL`
- Shadow proportion is set to 0 (i.e. minimal) when no intersections are detected between the park and the shadow footprint

Plotting `shadow_props` as function of `time_seq` (Figure 11) summarizes the daily course of shaded park proportion on the 2004-06-24. The individual value of 0.34 which we have calculated for 09:30 in the previous example (Figure 10) is highlighted in red.

Sky View Factor

The `SVF` function can be used to estimate the SVF at any 3D point location. For example, the following expression calculates the SVF on the ground[4] at the centroid of the `build` layer (Figure 4).

```
> s = SVF(
+   location = location,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT"
+ )
```

[4]Recall (Table 1) that the `inShadow` and `SVF` functions accept either 2D or 3D points, whereas 2D points are treated as ground locations

Figure 12: Sky View Factor (SVF) surface, with SVF value for an individual point (indicated by black + symbol at the center of the image)

The resulting SVF is 0.396, meaning that about 39.6% of the sky area are visible (Figure 12) from this particular location.

```
> s

#> [1] 0.3959721
```

Note that the `SVF` function has a tuning parameter named `res_angle` which can be used to modify angular resolution (default is 5°, as shown in Figure 4). A smaller `res_angle` value will give more accurate SVF but slower calculation.

Given a "template" grid, the latter calculation can be repeated to generate a continuous surface of SVF estimates for a grid of ground locations. In the following code section we calculate an SVF surface using the same raster template with a resolution of 2 meters from the shadow height example (see above).

```
> svf_surface = SVF(
+   location = r,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   parallel = 5
+ )
```

Note that the `parallel=5` option is used once again to make the calculation run simultaneously on 5 cores. The resulting SVF surface is shown in Figure 12. As could be expected, SVF values are lowest in the vicinity of buildings due to their obstruction of the sky.

Solar radiation

Shadow height, shadow footprint and SVF can be considered as low-level geometric calculations. Frequently, the ultimate aim of an analysis is the estimation of insolation, which is dependent on shadow and SVF but also on surface orientation and meteorological solar radiation conditions. Thus, the low-level geometric calculations are frequently combined and wrapped with meteorological solar radiation estimates to take the geometry into account when evaluating insolation over a given time interval. The **shadow** package provides this kind of wrapper function named `radiation`.

The `radiation` function needs several parameters to run -

- **3D points grid** representing surfaces where the solar radiation load is evaluated. It is important to specify whether each grid point is on a "roof" or on a "facade", and the azimuth it is facing (only for "facade"). A grid with those attributes can be automatically produced using the `surfaceGrid` function (see above)

- **Obstacles layer** defined with `obstacles`, having an `obstacles_height_field` attribute (see above)
- **Solar positions** defined with `solar_pos` (see above)
- **Meteorological estimates** defined with `solar_normal` and `solar_diffuse`, corresponding to the same time intervals given by `solar_pos`

Given this set of inputs, the `radiation` function:

- calculates whether each `grid` surface point is in shadow or not, for each solar position `solar_pos`, using the `inShadow` function (Equation 1),
- calculates the Coefficient of Direct Normal Irradiance reduction, for each `grid` surface point at each solar position `solar_pos`, using the `coefDirect` function (Equations 5 and 6),
- combines shadow, the coefficient and the meteorological estimate `solar_normal` to calculate the **direct** radiation (Equation 7),
- calculates the SVF for each `grid` surface point, using the `SVF` function (Equations 3 and 4),
- combines the SVF and the meteorological estimate `solar_diffuse` to calculate the **diffuse** radiation (Equation 8),
- and calculates sums of the **direct**, **diffuse** and **total** (i.e. direct+diffuse) solar radiation per `grid` surface point for the entire period (Equation 9).

To demonstrate the `radiation` function, we need one more component not used in the previous examples: the reference solar radiation data. The `shadow` package comes with a sample Typical Meteorological Year (TMY) dataset named `tmy` that can be used for this purpose. This dataset was compiled for the same geographical area where the buildings are located, and therefore can be realistically used in our example.

The `tmy` object is a `data.frame` with 8,760 rows, where each row corresponds to an hourly interval over an entire year ($24 \times 365 = 8,760$). The attributes given for each hourly interval include solar position (`sun_az`, `sun_elev`) and solar radiation estimates (`solar_normal`, `solar_diffuse`). Both solar radiation measurements are given in W/m^2 units.

```
> head(tmy, 10)

#>                   time sun_az sun_elev solar_normal solar_diffuse  dbt  ws
#> 2  1999-01-01 01:00:00  66.73   -70.94            0             0  6.6 1.0
#> 3  1999-01-01 02:00:00  82.02   -58.68            0             0  5.9 1.0
#> 4  1999-01-01 03:00:00  91.00   -45.99            0             0  5.4 1.0
#> 5  1999-01-01 04:00:00  98.13   -33.32            0             0  4.9 1.0
#> 6  1999-01-01 05:00:00 104.81   -20.86            0             0  4.4 1.0
#> 7  1999-01-01 06:00:00 111.73    -8.76            0             6  4.8 1.0
#> 8  1999-01-01 07:00:00 119.41     2.91          118            24  7.3 1.0
#> 9  1999-01-01 08:00:00 128.39    13.30          572            45 11.2 1.0
#> 10 1999-01-01 09:00:00 139.20    22.46          767            57 16.0 1.0
#> 11 1999-01-01 10:00:00 152.33    29.63          809            66 16.3 2.1
```

The Direct Normal Irradiance (`solar_normal`) is the amount of solar radiation received per unit area by a surface that is always held normal to the incoming rays from the sun's current position in the sky. This is an estimate of maximal **direct radiation**, obtained on an optimally tilted surface. The Diffuse Horizontal Irradiance (`solar_diffuse`) is the amount of radiation received per unit area at a surface that has not arrived on a direct path from the sun, but has been scattered by molecules and particles in the atmosphere. This is an estimate of **diffuse radiation**.

To use the solar positions from the `tmy` dataset, we create a separate matrix with just the `sun_az` and `sun_elev` columns -

```
> solar_pos = tmy[, c("sun_az", "sun_elev")]
> solar_pos = as.matrix(solar_pos)
```

The first few rows of this matrix are -

```
> head(solar_pos)

#>   sun_az sun_elev
#> 2  66.73   -70.94
#> 3  82.02   -58.68
```

```
#> 4  91.00   -45.99
#> 5  98.13   -33.32
#> 6 104.81   -20.86
#> 7 111.73    -8.76
```

Now we have everything needed to run the `radiation` function. We are hereby using the same `grid` layer with 3D points covering the roofs and facades of the four buildings created above using the `surfaceGrid` function (Figure 9), the layer of `obstacles`, and the solar position and measured solar radiation at a reference weather station from the `tmy` table.

```
> rad = radiation(
+   grid = grid,
+   obstacles = build,
+   obstacles_height_field = "BLDG_HT",
+   solar_pos = solar_pos,
+   solar_normal = tmy$solar_normal,
+   solar_diffuse = tmy$solar_diffuse,
+   parallel = 5
+ )
```

The returned object `rad` is a `data.frame` with the summed direct, diffuse and total (i.e. direct+diffuse) solar radiation estimates, as well as the SVF, for each specific surface location in `grid`. Summation takes place over the entire period given by `solar_pos`, `solar_normal` and `solar_diffuse`. In the present case it is an **annual** insolation. The units of measurement are therefore Wh/m^2 summed over an entire year.

For example, the following printout -

```
> head(rad)
```

```
#>         svf  direct  diffuse   total
#> 1 0.9999784 1242100 473329.8 1715430
#> 2 0.9999703 1242100 473325.9 1715426
#> 3 0.9999866 1242100 473333.6 1715434
#> 4 0.9999816 1242100 473331.3 1715431
#> 5 0.9999756 1242100 473328.5 1715428
#> 6 0.9999651 1242099 473323.5 1715423
```

refers to the first six surface points which are part of the same roof, thus sharing similar annual radiation load estimates. Overall, however, the differences in insolation are very substantial among different locations on the buildings surfaces, as shown in Figure 13. For example, the roofs receive about twice as much direct radiation as the south-facing facades. The code for producing Figure 13, using function `scatter3D` (see Figure 9), can be found on the help page of the `radiation` function and is thus omitted here to save space. Note that the figure shows radiation estimates in kWh/m^2 units, i.e. the values from the `rad` table (above) divided by 1000.

Discussion

The **shadow** package introduces a simple geometric model for shadow-related calculations in an urban environment. Specifically, the package provides functions for calculating shadow height, shadow footprint and SVF. The latter can be combined with TMY data to estimate insolation on built surfaces. It is, to the best of our knowledge, the only R package aimed at shadow calculations in a vector-based representation of the urban environment. It should be noted that the **insol** package provides similar functionality for a raster-based environment, but the latter is more suitable for modelling large-scale natural environments rather than detailed urban landscapes.

The unique aspect of our approach is that calculations are based on a vector layer of polygons extruded to a given height, known as 2.5D, such as building footprints with a height attribute. The vector-based 2.5D approach has several advantages over the two commonly used alternative ones: vector-based 3D and raster-based models. Firstly, the availability of 2.5D input data is much greater compared to both specialized 3D models and high-resolution raster surfaces. Building layers for entire cities are generally available from various sources, ranging from local municipality GIS systems to global crowd-sourced datasets (e.g. OpenStreetMap) (Haklay and Weber, 2008). Secondly, processing does not require closed-source software, or interoperability with complex specialized software, as opposed to working with 3D models. Thirdly, results are easily associated back to the respective urban elements such as buildings, parks, roofs facades, etc., as well as their attributes, via a spatial join

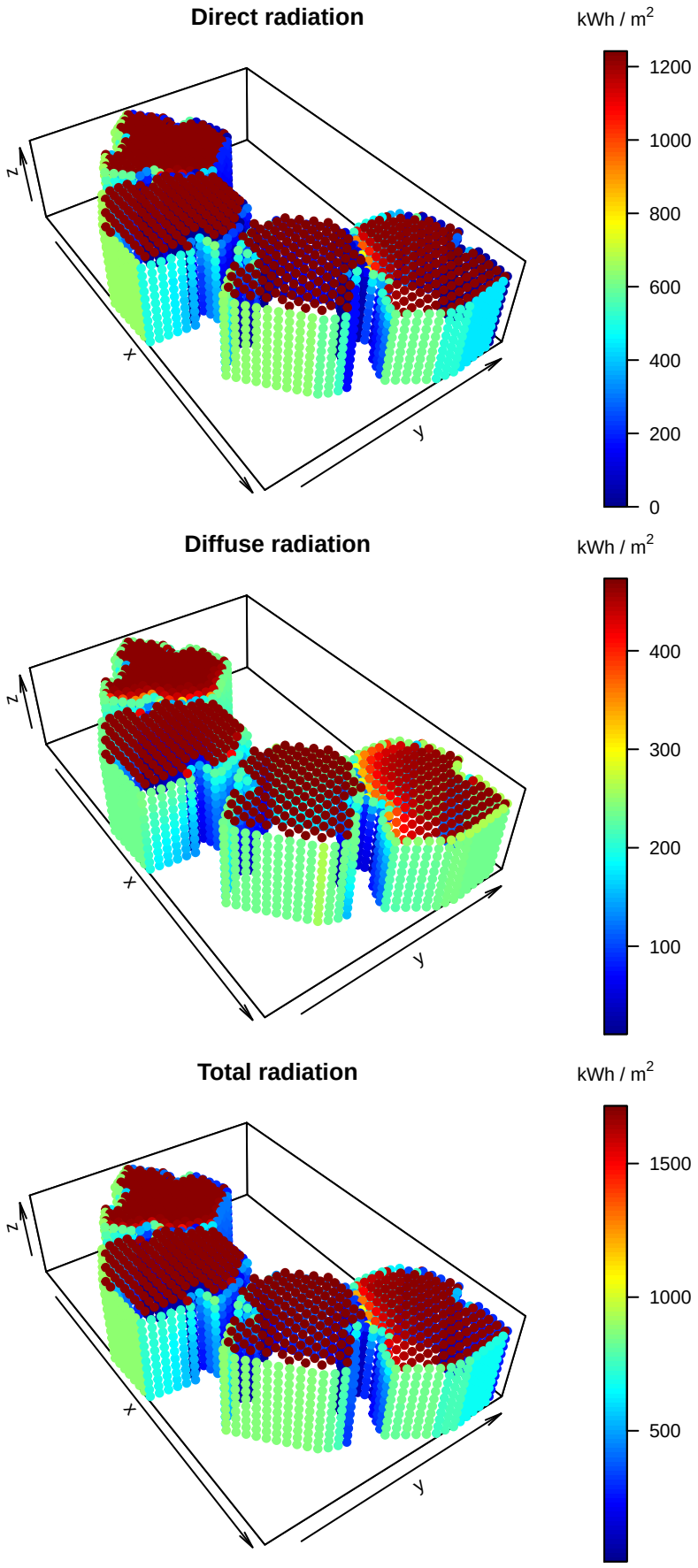

Figure 13: Annual direct, diffuse and total radiation estimates per grid point (kWh/m^2). Note that the y-axis points to the north.

operation (e.g. using function `over` in R package **sp**). For example, we can easily determine which building is responsible for shading the green park in the above shadow footprint example (Figure 10). This is unlike a raster-based approach, where the input is a continuous surface with no attributes, thus having no natural association to individual urban elements or objects.

However, it should be noted that the 2.5D vector-based approach requires several assumptions and has some limitations. When the assumptions do not hold, results may be less accurate compared to the above-mentioned alternative approaches. For example, it is impossible to represent geometric shapes that are not a simple extrusion in 2.5D (though, as mentioned above, urban surveys providing such detailed data are not typically available). An ellipsoid tree, a bridge with empty space underneath, a balcony extruding outwards from a building facade, etc., can only be represented with a polyhedral surface in a full vector-based 3D environment (Gröger and Plümer, 2012; Biljecki et al., 2016). Recently, classes for representing true-3D urban elements, such as the Simple Feature type `POLYHEDRALSURFACE`, have been implemented in R package **sf** (Pebesma, 2018). However, functions for working with those classes, such as calculating three-dimensional intersection, are still lacking. Implementing such functions in R could bring new urban analysis capabilities to the R environment in the future, in which solar analysis of 3D city models probably comprise a major use case (Biljecki et al., 2015).

It should also be noted that a vector-based calculation may be generally slower than a raster based calculation. This becomes important when the study area is very large. Though the present algorithms can be optimized to some extent, they probably cannot compete with raster-based calculations where sun ray intersections can be computed using fast ray-tracing algorithms based on matrix input (Amanatides et al., 1987), as opposed to computationally intensive search for intersections between a line and a polygonal layer in a vector-based environment. For example, calculating the SVF surface shown in Figure 12 requires processing 72 angular sections × 3,780 raster cells = 272,160 SVF calculations, which takes about 7.3 minutes using five cores on an ordinary desktop computer (Intel® Core™ i7-6700 CPU @ 3.40GHz × 8). The annual radiation estimate shown in Figure 13 however takes about 3.9 hours to calculate, as it requires SVF calculation for 2,688 grid points, as well as 727 ground locations × 8,760 hours = 6,368,520 shadow height calculations.

To summarize, the **shadow** package can be used to calculate shadow, SVF and radiation load in an urban environment using widely available polygonal building data, inside the R environment (e.g. Vulkan et al., 2018). Potential use cases include urban environment applications such as evaluation of micro-climatic influence for urban planning, studying urban well-being (e.g. climatic comfort) and estimating photovoltaic energy production potential.

Acknowledgements

The **shadow** package was developed as part of a study funded by the Israel Ministry of National Infrastructures, Energy and Water Resources under research grant # 021-11-215.

Bibliography

J. Amanatides, A. Woo, et al. A fast voxel traversal algorithm for ray tracing. In *Eurographics*, volume 87, pages 3–10, 1987. [p20]

B. Beckers. *Solar energy at urban scale*. John Wiley & Sons, 2013. [p4]

F. Biljecki, J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin. Applications of 3d city models: State of the art review. *ISPRS International Journal of Geo-Information*, 4(4):2842–2889, 2015. [p1, 20]

F. Biljecki, H. Ledoux, and J. Stoter. An improved lod specification for 3d building models. *Computers, Environment and Urban Systems*, 59:25–37, 2016. [p20]

R. Bivand and N. Lewin-Koh. *maptools: Tools for Reading and Handling Spatial Objects*, 2017. URL `https://CRAN.R-project.org/package=maptools`. R package version 0.9-2. [p9]

R. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*, 2017. URL `https://CRAN.R-project.org/package=rgeos`. R package version 0.3-26. [p8]

R. S. Bivand, E. Pebesma, and V. Gomez-Rubio. *Applied spatial data analysis with R, Second edition*. Springer, NY, 2013. URL `http://www.asdar-book.org/`. [p2]

F. Bourbia and F. Boucheriba. Impact of street design on urban microclimate for semi arid climate (constantine). *Renewable Energy*, 35(2):343–347, 2010. [p1]

J. G. Corripio. *insol: Solar Radiation*, 2014. URL https://CRAN.R-project.org/package=insol. R package version 1.1.1. [p1]

M. Dorman. *shadow: Geometric Shadow Calculations*, 2018. URL https://CRAN.R-project.org/package=shadow. R package version 0.5.0. [p2]

EnergyPlus. EnergyPlus weather data, 2018. URL https://energyplus.net/weather. [p7]

E. Erell, D. Pearlmutter, and T. Williamson. *Urban microclimate: designing the spaces between buildings*. Earthscan/James & James Science Publishers, 2011. [p4]

ESRI. *ArcGIS desktop: release 10.5*. Environmental Systems Research Institute, CA, 2017. URL https://www.arcgis.com. [p1]

S. Freitas, C. Catita, P. Redweik, and M. C. Brito. Modelling solar potential in the urban environment: State-of-the-art review. *Renewable and Sustainable Energy Reviews*, 41:915–931, 2015. [p1, 6]

P. Fu and P. M. Rich. Design and implementation of the solar analyst: an arcview extension for modeling solar radiation at landscape scales. In *Proceedings of the Nineteenth Annual ESRI User Conference*, pages 1–31, 1999. [p1]

T. Gál and J. Unger. A new software tool for svf calculations using building and tree-crown databases. *Urban Climate*, 10:594–606, 2014. [p4]

B. Givoni. *Climate considerations in building and urban design*. John Wiley & Sons, 1998. [p5]

GRASS Development Team. *Geographic Resources Analysis Support System (GRASS GIS) Software, Version 7.2*. Open Source Geospatial Foundation, 2017. URL http://grass.osgeo.org. [p1]

C. Grimmond, S. Potter, H. Zutter, and C. Souch. Rapid methods to estimate sky-view factors applied to urban areas. *International Journal of Climatology*, 21(7):903–913, 2001. [p4]

G. Gröger and L. Plümer. Citygml–interoperable semantic 3d city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71:12–33, 2012. [p2, 20]

M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4): 12–18, 2008. [p2, 18]

R. J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2017. URL https://CRAN.R-project.org/package=raster. R package version 2.6-7. [p8]

J. Hofierka and M. Suri. The solar radiation model for open source gis: implementation and applications. In *Proceedings of the Open source GIS-GRASS users conference*, volume 2002, pages 51–70, 2002. [p1]

J. Hofierka and M. Zlocha. A new 3-d solar radiation model for 3-d city models. *Transactions in GIS*, 16 (5):681–690, 2012. [p2]

L. Kumar, A. K. Skidmore, and E. Knowles. Modelling topographic variation in solar radiation in a gis environment. *International Journal of Geographical Information Science*, 11(5):475–497, 1997. [p1]

@Last and Google. *SketchUp: release 17*. Trimble Inc., CA, 2017. URL https://www.sketchup.com/. [p1]

B. W. Lewis. *threejs: Interactive 3D Scatter Plots, Networks and Globes*, 2017. URL https://CRAN.R-project.org/package=threejs. R package version 0.3.1. [p7]

J. Liang, J. Gong, J. Zhou, A. N. Ibrahim, and M. Li. An open-source 3d solar radiation model integrated with a 3d geographic information system. *Environmental Modelling & Software*, 64:94–101, 2015. [p1]

F. Lindberg, C. S. B. Grimmond, A. Gabey, B. Huang, C. W. Kent, T. Sun, N. E. Theeuwes, L. Järvi, H. C. Ward, I. Capel-Timms, et al. Urban multi-scale environmental predictor (umep): An integrated tool for city-based climate services. *Environmental Modelling & Software*, 99:70–87, 2018. [p1]

E. Pebesma. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1): 439–446, 2018. URL https://journal.r-project.org/archive/2018/RJ-2018-009/index.html. [p20]

E. Pebesma and R. S. Bivand. Classes and methods for spatial data: the sp package. *R news*, 5(2):9–13, 2005. [p2]

S. Pusat, İ. Ekmekçi, and M. T. Akkoyunlu. Generation of typical meteorological year for different climates of turkey. *Renewable Energy*, 75:144–151, 2015. [p7]

QGIS Development Team. *QGIS Geographic Information System*. Open Source Geospatial Foundation, 2017. URL http://qgis.osgeo.org. [p1]

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL https://www.R-project.org/. [p11]

C. Ratti and P. Richens. Raster analysis of urban form. *Environment and Planning B: Planning and Design*, 31(2):297–309, 2004. [p1]

P. Redweik, C. Catita, and M. Brito. Solar energy potential on roofs and facades in an urban landscape. *Solar Energy*, 97:332–341, 2013. [p1, 2]

G. Van Rossum and F. L. Drake. *The python language reference manual*. Network Theory Ltd., 2011. [p1]

A. Vulkan, I. Kloog, M. Dorman, and E. Erell. Modelling the potential for pv installation in residential buildings in dense urban areas. *Energy and Buildings*, 2018. [p20]

M. Weisthal. Assessment of potential energy savings in israel through climate-aware residential building design. Master's thesis, Ben-Gurion University of the Negev, Jacob Blaustein Institutes for Desert Research, Albert Katz International School for Desert Studies, 2014. [p3]

Michael Dorman
BGU
Department of Geography and Environmental Development
Ben-Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel
dorman@post.bgu.ac.il

Evyatar Erell
BGU
The Jacob Blaustein Institutes for Desert Research
and the Department of Geography and Environmental Development
Ben-Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel

Adi Vulkan
BGU
Department of Geography and Environmental Development
Ben-Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel

Itai Kloog
BGU
Department of Geography and Environmental Development
Ben-Gurion University of the Negev, P.O.B. 653 Beer-Sheva, Israel