

# Package ‘mascarade’

January 12, 2026

**Type** Package

**Title** Generating Cluster Masks for Single-Cell Dimensional Reduction Plots

**Version** 0.3.0

**Description** Implements a procedure to automatically generate 2D masks for clusters on dimensional reduction plots from methods like t-SNE (t-distributed stochastic neighbor embedding) or UMAP (uniform manifold approximation and projection), with a focus on single-cell RNA-sequencing data.

**Imports** data.table, spatstat.geom, spatstat.explore, lifecycle, ggplot2, scales, polyclip, ggforce, vctrs, rlang, cli, systemfonts

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.3.3

**Depends** R (>= 3.5)

**Suggests** testthat (>= 3.0.0), rmarkdown, knitr, patchwork, ggnewscale, ggsci, Seurat, SeuratObject

**Config/testthat.edition** 3

**VignetteBuilder** knitr

**URL** <https://alserglab.github.io/mascarade/>

**BugReports** <https://github.com/alserglab/mascarade/issues>

**NeedsCompilation** no

**Author** Alexey Sergushichev [aut, cre]

**Maintainer** Alexey Sergushichev <alsergbox@gmail.com>

**Repository** CRAN

**Date/Publication** 2026-01-12 20:20:02 UTC

## Contents

exampleMascarade . . . . .	2
fancyMask . . . . .	2
generateMask . . . . .	4
generateMaskSeurat . . . . .	5
geom_mark_shape . . . . .	6

## Index

11

exampleMascarade      *Example data with UMAP points from PBMC3K dataset.*

### Description

The object is a list with three elements:

1. `dims` – matrix of UMAP coordinates of the cells,
2. `clusters` – vector of cell population annotations,
3. `features` – matrix with gene expression for several genes.

fancyMask      *Generate ggplot2 layers for a labeled cluster mask*

### Description

Convenience helper that returns a list of ggplot2 components that draws polygon-like outlines and places cluster labels. The plotting limits are expanded (via `limits.expand`) to provide extra room for labels.

### Usage

```
fancyMask(
  maskTable,
  ratio = NULL,
  limits.expand = ifelse(label, 0.1, 0.05),
  linewidth = 1,
  shape.expand = linewidth * unit(-1, "pt"),
  label = TRUE,
  label.fontsize = 10,
  label.buffer = unit(0, "cm"),
  label.fontface = "plain",
  label.margin = margin(2, 2, 2, 2, "pt")
)
```

## Arguments

<code>maskTable</code>	A <code>data.frame</code> of mask coordinates. The first two columns are interpreted as x/y coordinates (in that order). Must contain at least the columns <code>cluster</code> (a factor) and <code>group</code> (grouping identifier passed to <code>geom_mark_shape()</code> ).
<code>ratio</code>	Optional aspect ratio passed to <code>ggplot2::coord_cartesian()</code> . Use 1 for equal scaling. Default is <code>NULL</code> (no fixed ratio).
<code>limits.expand</code>	Numeric scalar giving the fraction of the x/y range to expand on both sides when setting plot limits. Default is <code>0.1</code> with labels and <code>0.05</code> with no labels.
<code>linewidth</code>	Line width passed to <code>geom_mark_shape()</code> for the outline. Default is <code>1</code> .
<code>shape.expand</code>	Expansion or contraction applied to the marked shapes, passed to <code>geom_mark_shape(expand = ...)</code> . Default is <code>unit(-linewidth, "pt")</code> .
<code>label</code>	Boolean flag whether the labels should be displayed.
<code>label.fontsize</code>	Label font size passed to <code>geom_mark_shape()</code> . Default is <code>10</code> .
<code>label.buffer</code>	Label buffer distance passed to <code>geom_mark_shape()</code> . Default is <code>unit(0, "cm")</code> .
<code>label.fontface</code>	Label font face passed to <code>geom_mark_shape()</code> . Default is <code>"plain"</code> .
<code>label.margin</code>	Label margin passed to <code>geom_mark_shape()</code> . Default is <code>margin(2, 2, 2, 2, "pt")</code> .

## Details

The first two columns of `maskTable` are used as x/y coordinates. Cluster labels are taken from `maskTable$cluster`. Shapes are grouped by `maskTable$group`.

## Value

A list of `ggplot2` components suitable for adding to a plot with `+`, containing:

1. a `ggplot2::coord_cartesian()` specification, and
2. a `geom_mark_shape()` layer.

## See Also

- `geom_mark_shape()`

## Examples

```
data("exampleMascarade")
maskTable <- generateMask(dims=exampleMascarade$dims,
                           clusters=exampleMascarade$clusters)
library(ggplot2)
ggplot(do.call(cbind, exampleMascarade)) +
  geom_point(aes(x=UMAP_1, y=UMAP_2, color=GNLY)) +
  fancyMask(maskTable, ratio=1) +
  theme_classic()
```

generateMask

*Generate mask for clusters on 2D dimensional reduction plots*

## Description

Internally the function rasterizes and smoothes the density plots.

## Usage

```
generateMask(
  dims,
  clusters,
  gridSize = 200,
  expand = 0.005,
  minDensity = lifecycle::deprecated(),
  smoothSigma = NA,
  minSize = 10,
  kernel = lifecycle::deprecated(),
  type = lifecycle::deprecated()
)
```

## Arguments

<code>dims</code>	matrix of point coordinates. Rows are points, columns are dimensions. Only the first two columns are used.
<code>clusters</code>	vector of cluster annotations. Should be the same length as the number of rows in <code>dims</code> .
<code>gridSize</code>	target width and height of the raster used internally
<code>expand</code>	distance used to expand borders, represented as a fraction of $\sqrt{\text{width} * \text{height}}$ . Default: 1/200.
<code>minDensity</code>	Deprecated. Doesn't do anything.
<code>smoothSigma</code>	Deprecated. Parameter controlling smoothing and joining close cells into groups, represented as a fraction of $\sqrt{\text{width} * \text{height}}$ . Increasing this parameter can help dealing with sparse regions.
<code>minSize</code>	Groups of less than <code>minSize</code> points are ignored, unless it is the only group for a cluster
<code>kernel</code>	Deprecated. Doesn't do anything.
<code>type</code>	Deprecated. Doesn't do anything.

## Value

data.table with points representing the mask borders. Each individual border line corresponds to a single level of group column. Cluster assignment is in `cluster` column.

## Examples

```
data("exampleMascarade")
maskTable <- generateMask(dims=exampleMascarade$dims,
                           clusters=exampleMascarade$clusters)
data <- data.frame(exampleMascarade$dims,
                    cluster=exampleMascarade$clusters,
                    exampleMascarade$features)

library(ggplot2)
ggplot(data, aes(x=UMAP_1, y=UMAP_2)) +
  geom_point(aes(color=cluster)) +
  geom_path(data=maskTable, aes(group=group)) +
  coord_fixed() +
  theme_classic()
```

generateMaskSeurat

*Generates mask from a Seurat object. Requires SeuratObject package.*

## Description

Generates mask from a Seurat object. Requires SeuratObject package.

## Usage

```
generateMaskSeurat(
  object,
  reduction = NULL,
  group.by = NULL,
  gridSize = 200,
  expand = 0.005,
  minSize = 10
)
```

## Arguments

object	Seurat object
reduction	character vector specifying which reduction to use (default: DefaultDimReduc(object))
group.by	character vector specifying which field to use for clusters (default: "ident")
gridSize	target width and height of the raster used internally
expand	distance used to expand borders, represented as a fraction of sqrt(width*height). Default: 1/200.
minSize	Groups of less than minSize points are ignored, unless it is the only group for a cluster

## Value

data.table with points representing the mask borders. Each individual border line corresponds to a single level of group column. Cluster assignment is in cluster column.

## Examples

```
# only run if Seurat is installed
if (require("Seurat")) {
  data("pbmc_small")
  maskTable <- generateMaskSeurat(pbmc_small)

  library(ggplot2)
  # not the best plot, see vignettes for better examples
  DimPlot(pbmc_small) +
    geom_path(data=maskTable, aes(x=tSNE_1, y=tSNE_2, group=group))
}
```

**geom\_mark\_shape**

*Annotate areas with polygonal shapes*

## Description

This geom lets you annotate sets of points via polygonal shapes. Unlike other `ggforce::geom_mark_*` functions, `geom_mark_shape` should be explicitly provided with the shape coordinates. As in `ggforce::geom_shape`, the polygon can be expanded/contracted and corners can be rounded, which is controlled by `expand` and `radius` parameters.

## Usage

```
geom_mark_shape(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  expand = 0,
  radius = 0,
  label.margin = margin(2, 2, 2, 2, "mm"),
  label.width = NULL,
  label.minwidth = unit(50, "mm"),
  label.hjust = 0,
  label.fontsize = 12,
  label.family = "",
  label.lineheight = 1,
  label.fontface = c("bold", "plain"),
  label.fill = "white",
  label.colour = "black",
  label.buffer = unit(10, "mm"),
  con.colour = "black",
  con.size = 0.5,
  con.type = "elbow",
  con.linetype = 1,
  con.border = "one",
```

```

con.cap = unit(3, "mm"),
con.arrow = NULL,
...,
na.rm = FALSE,
show.legend = NA,
inherit.aes = TRUE
)

```

## Arguments

<b>mapping</b>	Set of aesthetic mappings created by <a href="#">aes()</a> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply <code>mapping</code> if there is no plot mapping.
<b>data</b>	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <a href="#">ggplot()</a> . A <code>data.frame</code> , or other object, will override the plot data. All objects will be fortified to produce a data frame. See <a href="#">fortify()</a> for which variables will be created. A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code> , and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code> ).
<b>stat</b>	The statistical transformation to use on the data for this layer. When using a <code>geom_*</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following: <ul style="list-style-type: none"> <li>• A Stat ggproto subclass, for example <code>StatCount</code>.</li> <li>• A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>.</li> <li>• For more information and other ways to specify the stat, see the <a href="#">layer stat</a> documentation.</li> </ul>
<b>position</b>	A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The <code>position</code> argument accepts the following: <ul style="list-style-type: none"> <li>• The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position.</li> <li>• A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as <code>"jitter"</code>.</li> <li>• For more information and other ways to specify the position, see the <a href="#">layer position</a> documentation.</li> </ul>
<b>expand</b>	A numeric or unit vector of length one, specifying the expansion amount. Negative values will result in contraction instead. If the value is given as a numeric it will be understood as a proportion of the plot area width.
<b>radius</b>	As <code>expand</code> but specifying the corner radius.

label.margin	The margin around the annotation boxes, given by a call to <code>ggplot2::margin()</code> .
label.width	A fixed width for the label. Set to NULL to let the text or <code>label.minwidth</code> decide.
label.minwidth	The minimum width to provide for the description. If the size of the label exceeds this, the description is allowed to fill as much as the label.
label.hjust	The horizontal justification for the annotation. If it contains two elements the first will be used for the label and the second for the description.
label.fontsize	The size of the text for the annotation. If it contains two elements the first will be used for the label and the second for the description.
label.family	The font family used for the annotation. If it contains two elements the first will be used for the label and the second for the description.
label.lineheight	The height of a line as a multiplier of the fontsize. If it contains two elements the first will be used for the label and the second for the description.
label.fontface	The font face used for the annotation. If it contains two elements the first will be used for the label and the second for the description.
label.fill	The fill colour for the annotation box. Use "inherit" to use the fill from the enclosure or "inherit_col" to use the border colour of the enclosure.
label.colour	The text colour for the annotation. If it contains two elements the first will be used for the label and the second for the description. Use "inherit" to use the border colour of the enclosure or "inherit_fill" to use the fill colour from the enclosure.
label.buffer	The size of the region around the mark where labels cannot be placed.
con.colour	The colour for the line connecting the annotation to the mark. Use "inherit" to use the border colour of the enclosure or "inherit_fill" to use the fill colour from the enclosure.
con.size	The width of the connector. Use "inherit" to use the border width of the enclosure.
con.type	The type of the connector. Either "elbow", "straight", or "none".
con.linetype	The linetype of the connector. Use "inherit" to use the border linetype of the enclosure.
con.border	The bordertype of the connector. Either "one" (to draw a line on the horizontal side closest to the mark), "all" (to draw a border on all sides), or "none" (not going to explain that one).
con.cap	The distance before the mark that the line should stop at.
con.arrow	An arrow specification for the connection using <code>grid::arrow()</code> for the end pointing towards the mark.
...	<p>Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through .... Unknown arguments that are not part of the 4 categories below are ignored.</p> <ul style="list-style-type: none"> <li>• Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an <b>Aesthetics</b> section that lists the</li> </ul>

available options. The 'required' aesthetics cannot be passed on to the params. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data.

- When constructing a layer using a `stat_*`() function, the `...` argument can be used to pass on parameters to the geom part of the layer. An example of this is `stat_density(geom = "area", outline.type = "both")`. The geom's documentation lists which parameters it can accept.
- Inversely, when constructing a layer using a `geom_*`() function, the `...` argument can be used to pass on parameters to the stat part of the layer. An example of this is `geom_area(stat = "density", adjust = 0.5)`. The stat's documentation lists which parameters it can accept.
- The `key_glyph` argument of `layer()` may also be passed on through `...`. This can be one of the functions described as [key glyphs](#), to change the display of the layer in the legend.

<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
<code>inherit.aes</code>	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>borders()</code> .

## Value

A ggplot2 layer (`ggplot2::layer`) that adds polygonal shape annotations to a plot.

## Aesthetics

`geom_mark_shape` understand the following aesthetics (required aesthetics are in bold):

- **x**
- **y**
- **x0** (*used to anchor the label*)
- **y0** (*used to anchor the label*)
- filter
- label
- description
- color
- fill
- group
- size
- linetype
- alpha

## Annotation

All `geom_mark_*` allow you to put descriptive textboxes connected to the mark on the plot, using the `label` and `description` aesthetics. The textboxes are automatically placed close to the mark, but without obscuring any of the datapoints in the layer. The placement is dynamic so if you resize the plot you'll see that the annotation might move around as areas become big enough or too small to fit the annotation. If there's not enough space for the annotation without overlapping data it will not get drawn. In these cases try resizing the plot, change the size of the annotation, or decrease the buffer region around the marks.

## Filtering

Often marks are used to draw attention to, or annotate specific features of the plot and it is thus not desirable to have marks around everything. While it is possible to simply pre-filter the data used for the mark layer, the `geom_mark_*` geoms also comes with a dedicated `filter` aesthetic that, if set, will remove all rows where it evaluates to `FALSE`. There are multiple benefits of using this instead of prefILTERing. First, you don't have to change your data source, making your code more adaptable for exploration. Second, the data removed by the filter aesthetic is remembered by the geom, and any annotation will take care not to overlap with the removed data.

## Examples

```
library(ggplot2)
shape1 <- data.frame(
  x = c(0, 3, 3, 2, 2, 1, 1, 0),
  y = c(0, 0, 3, 3, 1, 1, 3, 3),
  label="bracket"
)
shape2 <- data.frame(
  x = c(0, 3, 3, 0)+4,
  y = c(0, 0, 3, 3),
  label="square"
)
shape3 <- data.frame(
  x = c(0, 1.5, 3, 1.5)+8,
  y = c(1.5, 0, 1.5, 3),
  label="diamond"
)

ggplot(rbind(shape1, shape2, shape3), aes(x=x, y=y, label=label, color=label, fill=label)) +
  geom_mark_shape() +
  ylim(0, 5)
```

# Index

- \* **mark geoms**
  - geom\_mark\_shape, 6
- aes(), 7
- borders(), 9
- exampleMascarade, 2
- fancyMask, 2
- fortify(), 7
- generateMask, 4
- generateMaskSeurat, 5
- geom\_mark\_shape, 6
- ggplot(), 7
- ggplot2::margin(), 8
- grid::arrow(), 8
- key\_glyphs, 9
- layer\_position, 7
- layer\_stat, 7
- layer(), 8, 9