Package: raybevel (via r-universe)

October 23, 2024

Type Package

Title Generates Polygon Straight Skeletons and 3D Bevels

Version 0.1.3

Maintainer Tyler Morgan-Wall <tylermw@gmail.com>

Description Generates polygon straight skeletons and 3D models. Provides functions to create and visualize interior polygon offsets, 3D beveled polygons, and 3D roof models.

License GPL-3

Encoding UTF-8

Depends R (>= 3.0.2)

Imports progress, digest, decido, rayvertex (>= 0.10.4), sf, grid

LinkingTo Rcpp, BH, RcppCGAL (>= 5.6.3), progress, RcppThread (>= 2.1.6)

Suggests spData, rayrender, testthat (>= 3.0.0), ggplot2, png

RoxygenNote 7.3.1

Config/testthat/edition 3

URL https://www.raybevel.com, https://github.com/tylermorganwall/raybevel/

BugReports https://github.com/tylermorganwall/raybevel/issues

NeedsCompilation yes

Author Tyler Morgan-Wall [aut, cph, cre] (<https://orcid.org/0000-0002-3131-3814>)

Date/Publication 2024-05-14 07:30:02 UTC

Additional_repositories https://cranhaven.r-universe.dev

Repository https://cranhaven.r-universe.dev

RemoteUrl https://github.com/cranhaven/cranhaven.r-universe.dev

RemoteRef package/raybevel

RemoteSha 5c21c8c697639835ce4859e0a4bd6f2e94f06fcb

Contents

`change_polygon_bevel` *Change an existing polygon bevel's bevel profile.*

Description

This function generates a beveled 3D polygon model from the modified straight skeleton with pre-existing polygons generated from the 'generate_beveled_polygon' function when 'return_skeleton_polygons = TRUE'.

Usage

```
change_polygon_bevel(
  skeleton_polygons,
  bevel_offsets = NULL,
  bevel_heights = NULL,
  set_max_height = FALSE,
  max_height = 1,
  vertical_offset = 0,
  base = TRUE,
  base_height = NA,
  raw_offsets = FALSE,
  raw_heights = FALSE,
  swap_yz = TRUE,
  progress = TRUE,
  sides = FALSE,
  double_sided = FALSE,
  scale_all_max = FALSE,
  material = material_list(),
  bevel_material = NA,
  verbose = FALSE
)
```

Arguments

skeleton_polygons	Default 'NULL'. A straight skeleton generated from the 'generate_beveled_polygon' function when 'return_skeleton_polygons = TRUE'.
bevel_offsets	Default 'NULL'. The offset(s) of the bevel.
bevel_heights	Default is set to 'bevel_offsets'. Numeric vector specifying the heights of the bevels. Must be of the same length as 'bevel_offsets'.
set_max_height	Default 'FALSE'. A logical flag that controls whether to set the max height of the polygon based on the 'max_height' argument.
max_height	Default '1'. The maximum height of the polygon.
vertical_offset	Default '0'. The vertical offset of the polygon.
base	Default 'TRUE'. A logical flag that controls whether to generate the bottom of the polygon.
base_height	Default 'NA'. Height of the base, defaulting to the 'min(bevel_heights) + vertical_offset' .
raw_offsets	Default 'FALSE'. A logical flag indicating whether the 'bevel_offsets' are already in raw format and do not need to be multiplied by the maximum time of the skeleton.
raw_heights	Default 'FALSE'. A logical flag indicating whether the 'bevel_heights' are already in raw format and do not need to be multiplied by the maximum time of the skeleton.
swap_yz	Default 'TRUE'. A logical flag that controls whether to swap the y and z coordinates in the resulting mesh. If 'TRUE', the y and z coordinates will be swapped.
progress	Default 'TRUE'. Whether to display a progress bar.
sides	Default 'FALSE'. A logical flag on whether to draw the sides. This will automatically be set to 'TRUE' if 'base = TRUE' and the 'base_height' is less than 'vertical_offset'.
double_sided	Default 'FALSE'. A logical flag that controls whether the polygon should be double-sided.
scale_all_max	Default 'FALSE'. If passing in a list of multiple skeletons with polygons, whether to scale each polygon to the overall max height, or whether to scale each max height to the maximum internal distance in the polygon.
material	Default 'material_list()'. Interface to set the color/appearance/material options for the resulting 'ray_mesh' mesh.
bevel_material	Default 'NA', uses the material specified in 'material'. Interface to set the color/appearance/material options for the resulting 'ray_mesh' bevel mesh.
verbose	Default 'FALSE'. A logical flag to control whether additional timing information should be displayed.

Value

A 3D mesh of the beveled polygon model.

Examples

```
# Skeletonize a complex {sf} object and set return_skeleton_polygons = TRUE in
# generate_beveled_polygon(). This returns skeleton object with polygons included, which
# allows for quickly generating 3D models with different bevels.
if(run_documentation()) {
  library(rayrender)
  library(rayvertex)
  us_states = spData::us_states
  cali = us_states[us_states$NAME == "California",]
  cali_skeleton = skeletonize(cali)
  plot_skeleton(cali_skeleton)
  # We add manual offsets to ensure that the polygon can be morphed all along its interior
  bevel = generate_bevel(manual_offsets = seq(0,1,by=0.01), max_height=0.5)
  bevel_model_cali = generate_beveled_polygon(cali_skeleton,
                                              bevel_offsets = bevel,
                                              return_skeleton_polygons = TRUE)

  bevel_new = change_polygon_bevel(bevel_model_cali,
                                   bevel_offsets = generate_bevel(max_height=0.5,
                                   bevel_end=0.5)) |>
    center_mesh()

  scene_base = xz_rect(xwidth=100,zwidth=100,
                       material=diffuse(color="grey20", checkercolor="white")) |>
    add_object(sphere(y=8,z=10,x=-3,material=light(intensity=100))) |>
    add_object(sphere(y=800,z=10,x=-3,radius=100,material=light(intensity=5)))

  raymesh_model(bevel_new, y=0.5, override_material = TRUE,
                material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(0,30,-10), sample_method = "sobol_blue",clamp_value = 10,
                 width=800,height=800,fov=0,ortho_dimensions=c(12,12))
}
# Change to a smooth bevel
if(run_documentation()) {
  new_bevel = generate_bevel("circular", bevel_start = 0, bevel_end=1)
  bevel_new = change_polygon_bevel(bevel_model_cali,
                                   bevel_offsets = new_bevel, solid ) |>
    center_mesh()
 raymesh_model(bevel_new, override_material = TRUE, y=1,material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(0,30,-10), sample_method = "sobol_blue",clamp_value = 10,
                 width=800,height=800,fov=0,ortho_dimensions=c(12,12))
}

# Make a complex bevel
if(run_documentation()) {
  complex_coords = generate_complex_bevel(
    bevel_type  = c("angled","flat", "angled", "flat"),
    bevel_start = head(seq(0,1,by=0.05),-1),
    bevel_end   = tail(seq(0,1,by=0.05),-1),
    overall_height = 1,
```

```
    angle = c(45,45,15,15),
    reverse = c(FALSE, FALSE,TRUE,TRUE),
    plot_bevel = TRUE
  )
  bevel_new = change_polygon_bevel(bevel_model_cali,
                                   bevel_offsets = complex_coords) |>
    center_mesh()
 raymesh_model(bevel_new, override_material = TRUE, y=1,material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(0,30,-20), sample_method = "sobol_blue",clamp_value = 10,
                 width=800,height=800,fov=0,ortho_dimensions=c(12,12))
}

# Quickly generate new bevels to inflate California like a balloon using the arctan function.
if(run_documentation()) {
  inflate_california = function(magnitudes) {
  for(val in magnitudes) {
    bevel_new = change_polygon_bevel(bevel_model_cali,
                                     bevel_heights = 1/2*atan(seq(0,val,length.out=100)),
                                      bevel_offsets = seq(0,1, length.out=100),
                                      base = TRUE) |>
      translate_mesh(c(-120.49,0,-38.72))
    raymesh_model(bevel_new, y = 0, override_material = TRUE,
                  material = glossy(color="darkred")) |>
      add_object(scene_base) |>
    add_object(sphere(x=-30,z=30,y=18,radius=30,material=light(color="white", intensity=5))) |>
      render_scene(lookfrom=c(-1, 28, -20.32), lookat=c(-1, 1.46, -2),
                   sample_method = "sobol_blue", clamp_value = 10,
                   width=800,height=800,fov=20,samples=256)
    }
  }
  inflate_california(c(1,4,16,64))
}
```

`generate_bevel` *Generate 2D Bevel Profile for 3D Polygons*

Description

Generate 2D Bevel Profile for 3D Polygons

Usage

```
generate_bevel(
  bevel_type = "angled",
  bevel_start = 0,
  bevel_end = 0.2,
  max_height = 1,
  angle = NULL,
  curve_points = 50,
```

```
  reverse = FALSE,
  flip = FALSE,
  initial_height = 0,
  add_end_points = TRUE,
  manual_offsets = NULL,
  step_epsilon = 1e-08,
  plot_bevel = FALSE,
  set_minimum_zero = TRUE,
  zero_offset_epsilon = 1e-05
)
```

Arguments

bevel_type	Character ‘angled‘. Type of the bevel, one of the following options: - "circular": Creates a rounded bevel resembling a quarter-circle. - "exp": Creates an exponential curve, starting slow and accelerating. - "bump": Creates a bump-like profile, rising and falling within the coverage. - "step": Generates a step-like bevel with a flat top. - "block": Generates a block-like bevel, jumping straight to the max_height and back to the base. - "angled": Generates a straight angled bevel. You can optionally set the ’angle’ parameter for this bevel. - "flat": Generates a flat area.
bevel_start	Default ‘0‘. The starting point of the bevel along the curve, ranges between 0 and 1.
bevel_end	Default ‘0.2‘. The ending point of the bevel along the curve, ranges between 0 and 1.
max_height	Default ‘1‘. The maximum height of the bevel, as measured from the initial height.
angle	Default ‘NULL‘. Optional angle parameter in degrees for angular bevels.
curve_points	Default ‘50‘. Number of points to plot for curve-based bevels.
reverse	Default ‘FALSE‘. If ‘TRUE‘, the curve is reversed vertically.
flip	Default ‘FALSE‘. If ‘TRUE‘, the curve is flipped horizontally.
initial_height	Default ‘0‘. The initial height from which the bevel starts. The bevel is rescaled to fit within the range from initial_height to max_height.
add_end_points	Default ‘TRUE‘. Whether to ensure there is a point at zero and a point at one.
manual_offsets	Default ‘NULL‘, none. This will force the bevel to add a point (interpolating between the two nearest points) at the specified offsets. This is useful when you want to add points at specific distances along the curve.
step_epsilon	Default ‘1e-5‘. The size for the small percentage step when using a step bevel.
plot_bevel	Default ‘FALSE‘. Whether to plot the bevel.
set_minimum_zero	Default ‘TRUE‘. Whether to offset the lowest point of the bevel so it’s at zero.
zero_offset_epsilon	Default ‘1e-5‘. Amount to offset the bevel to ensure no self-intersection with the base.

Value

List containing 'x' and 'y', which are the coordinates of the 2D bevel profile.

Examples

```
# Generate a single bevel profile and plot it
coords = generate_bevel("circular", 0.2, 0.8, 0.2, plot_bevel = TRUE)

# Plot all bevel profiles in a grid
plot_all_bevels = function() {
  oldpar = par(mfrow = c(4, 3), mai = c(0.2, 0.2, 0.5, 0.2))
  on.exit(par(oldpar))
  max_height = c(1,1,1,1)
  types = rep(c("circular", "exp", "bump", "step", "block", "angled"),2)
  reverses = c(rep(FALSE,6),rep(TRUE,6))
  for(i in seq_len(length(types))) {
    coords = generate_bevel(types[i], 0.2, 0.8, 1, flip = TRUE,
                            angle = 45, reverse = reverses[i], plot_bevel = TRUE)
  }
}
plot_all_bevels()
```

---

generate_beveled_polygon

Generate a beveled 3D polygon

Description

This function generates a beveled 3D polygon from a straight skeleton.

Usage

```
generate_beveled_polygon(
  skeleton,
  bevel_offsets = generate_bevel(),
  bevel_heights = NULL,
  set_max_height = FALSE,
  max_height = NA,
  vertical_offset = 0,
  base = TRUE,
  base_height = 0,
  raw_offsets = FALSE,
  raw_heights = FALSE,
  swap_yz = TRUE,
  progress = TRUE,
  double_sided = FALSE,
  sides = FALSE,
  return_skeleton_polygons = FALSE,
```

```
  scale_all_max = FALSE,
  material = material_list(),
  bevel_material = NA,
  verbose = FALSE
)
```

Arguments

skeleton	Default 'NULL'. A straight skeleton generated from the 'skeletonize' function.
bevel_offsets	Default 'NULL'. The offset(s) of the bevel.
bevel_heights	Default is set to 'bevel_offsets'. Numeric vector specifying the heights of the bevels. Must be of the same length as 'bevel_offsets'.
set_max_height	Default 'FALSE'. A logical flag that controls whether to set the max height of the roof based on the 'max_height' argument.
max_height	Default '1'. The maximum height of the polygon.
vertical_offset	Default '0'. The vertical offset of the polygon.
base	Default 'TRUE'. A logical flag that controls whether to generate the bottom of the polygon.
base_height	Default 'NA'. Height of the base, defaulting to 'min(bevel_heights) + vertical_offset' .
raw_offsets	Default 'FALSE'. A logical flag indicating whether the 'bevel_offsets' are already in raw format and do not need to be multiplied by the maximum time of the skeleton.
raw_heights	Default 'FALSE'. A logical flag indicating whether the 'bevel_heights' are already in raw format and do not need to be multiplied by the maximum time of the skeleton.
swap_yz	Default 'TRUE'. A logical flag that controls whether to swap the y and z coordinates in the resulting mesh. If 'TRUE', the y and z coordinates will be swapped.
progress	Default 'TRUE'. A logical flag to control whether a progress bar is displayed during roof generation.
double_sided	Default 'FALSE'. A logical flag that controls whether the polygon should be double-sided.
sides	Default 'FALSE'. A logical flag on whether to draw the sides. This will automatically be set to 'TRUE' if 'base = TRUE' and the 'base_height' is less than 'vertical_offset'.
return_skeleton_polygons	Default 'FALSE'. A logical flag that controls whether to return the skeleton polygons along with the 3D mesh.
scale_all_max	Default 'FALSE'. If passing in a list of multiple skeletons with polygons, whether to scale each polygon to the overall max height, or whether to scale each max height to the maximum internal distance in the polygon.
material	Default 'material_list()'. Interface to set the color/appearance/material options for the resulting 'ray_mesh' mesh.

bevel_material Default 'NA', uses the material specified in 'material'. Interface to set the color/appearance/material options for the resulting 'ray_mesh' bevel mesh.

verbose Default 'FALSE'. A logical flag to control whether additional timing information should be displayed.

Value

A 3D mesh of the beveled polygon model.

Examples

```
#Generate vertices and holes
vertices = matrix(c(0,0, 7,0, 7,7, 0,7, 0,0), ncol = 2, byrow = TRUE)-3.5
hole_1 = matrix(c(1,1, 2,1, 2,2, 1,2, 1,1), ncol = 2, byrow = TRUE)[5:1,]-3.5
hole_2 = matrix(c(5,5, 6,5, 6,6, 5,6, 5,5), ncol = 2, byrow = TRUE)[5:1,]-3.5
skeleton = skeletonize(vertices, holes = list(hole_1, hole_2))
plot_skeleton(skeleton)

#Generate a roof model and specify the material
if(run_documentation()) {
  library(rayrender)
  library(rayvertex)
  scene_base = xz_rect(xwidth=100,zwidth=100,
                       material=diffuse(color="grey20", checkercolor="white")) |>
    add_object(sphere(y=8,z=10,x=-3,material=light(intensity=100))) |>
    add_object(sphere(y=800,z=10,x=-3,radius=100,material=light(intensity=5))) |>
    add_object(sphere(x=-10,z=-10,y=5,material=light(color="red", intensity=40))) |>
    add_object(sphere(x=10,z=-10,y=5,material=light(color="orange", intensity=40)))

  bevel = generate_bevel("angled", bevel_start = 0, bevel_end = 0.2, max_height=0.25)
  roof_model = generate_beveled_polygon(skeleton,
                                        bevel_offsets = bevel,
                                        material = material_list(diffuse="purple"))

  raymesh_model(roof_model, override_material = FALSE) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,30,20), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}

# Change the bevel to be circular
if(run_documentation()) {
  bevel = generate_bevel("circular", bevel_start = 0, bevel_end = 0.2, max_height=0.25)
  roof_model = generate_beveled_polygon(skeleton,
                                        bevel_offsets = bevel,
                                        material = material_list(diffuse="purple"))

  raymesh_model(roof_model, override_material = FALSE) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,30,20), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}
```

```
# Change the bevel to type "bump", change the max height, and raise it off the surface
if(run_documentation()) {
  bevel = generate_bevel("bump", bevel_start = 0, bevel_end = 0.4, max_height=0.25)
  roof_model = generate_beveled_polygon(skeleton, base_height=1,
                                        bevel_offsets = bevel,
                                        material = material_list(diffuse="purple"))

  raymesh_model(roof_model, override_material = FALSE) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,30,20), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}

# Generate a complex bevel and use the exact specified heights
if(run_documentation()) {
  bevel = generate_complex_bevel(c("bump", "exp", "circular","step"),
                                 bevel_start = c(0,0.3,0.7,0.95),
                                 bevel_end = c(0.1,0.6,0.95,1),
                                 reverse = c(F,F,T,F),
                                 segment_height = c(0.25,0.5,0.5,4),
                                 plot_bevel = TRUE)

  roof_model = generate_beveled_polygon(skeleton, vertical_offset=0.1,
                                        bevel_offsets = bevel,
                                        raw_heights = TRUE,
                                        material = material_list(diffuse="purple"))

  raymesh_model(roof_model, override_material = FALSE) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,30,20), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}

# Turn the polygon into a ziggurat, using the step bevel type
if(run_documentation()) {
  offs = seq(0, 1, by = 0.05)
  bevel = generate_complex_bevel("step",
                                 bevel_start = offs[-length(offs)],
                                 bevel_end = offs[-1],
                                 segment_height = 0.2)

  roof_model = generate_beveled_polygon(skeleton, vertical_offset=0.2,
                                        bevel_offsets = bevel,
                                        raw_heights = TRUE,
                                        material = material_list(diffuse = "purple"))

  raymesh_model(roof_model, override_material = FALSE) |>
    add_object(scene_base) |>
    render_scene(lookfrom = c(10,30,20), sample_method = "sobol_blue",
                 width = 800, height = 800, fov = 0, ortho_dimensions = c(10,10))
}
```

```
# Turn the polygon into a smooth wavy slide, taking advantage of vector recycling to flip/reverse
if(run_documentation()) {
  offs = seq(0, 1, by = 0.1)
  bevel = generate_complex_bevel("exp",
                                 bevel_start = offs[-length(offs)],
                                 bevel_end = offs[-1],
                                 reverse = c(TRUE, FALSE),
                                 flip = c(TRUE, FALSE),
                                 segment_height = 0.25)

  roof_model = generate_beveled_polygon(skeleton, vertical_offset=0.2,
                                        bevel_offsets = bevel,
                                        raw_heights = TRUE,
                                        material = material_list(diffuse = "purple"))

  raymesh_model(roof_model, override_material = FALSE) |>
    add_object(scene_base) |>
    render_scene(lookfrom = c(10,30,20), sample_method = "sobol_blue",
                 width = 800, height = 800, fov = 0, ortho_dimensions = c(10,10))
}

# Skeletonize and turn an {sf} object into a beveled polygon
if(run_documentation()) {
  us_states = spData::us_states
  texas = us_states[us_states$NAME == "Texas",]
  texas_skeleton = skeletonize(texas)
  plot_skeleton(texas_skeleton)

  bevel = generate_bevel("angled" , bevel_end=0.3, max_height = 0.3)
  roof_model_texas = generate_beveled_polygon(texas_skeleton,
                                        bevel_offsets = bevel,
                                        material = material_list(diffuse = "purple")) |>
    center_mesh() |>
    translate_mesh(c(0,0.3,0))

  raymesh_model(roof_model_texas, material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    add_object(sphere(x=-10,z=-10,y=5,material=light(color="red", intensity=40))) |>
    add_object(sphere(x=10,z=-10,y=5,material=light(color="orange", intensity=40))) |>
    render_scene(lookfrom=c(0,10,0),camera_up=c(0,0,1), sample_method = "sobol_blue",
                 width=800,height=800,fov=0, ortho_dimensions=c(15,15))
}

# Generate a smooth bevel
if(run_documentation()) {
  bevel = generate_bevel("exp", bevel_start = 0, bevel_end=0.5, max_height=2)
  roof_model_texas = generate_beveled_polygon(texas_skeleton,
                                        bevel_offsets = bevel,
                                        material = material_list(diffuse = "purple")) |>
    center_mesh() |>
    translate_mesh(c(0,0.5,0))

  raymesh_model(roof_model_texas, material = diffuse(color="purple")) |>
```

```
    add_object(scene_base) |>
    add_object(sphere(x=-10,z=-10,y=5,material=light(color="red", intensity=40))) |>
    add_object(sphere(x=10,z=-10,y=5,material=light(color="orange", intensity=40))) |>
    render_scene(lookfrom=c(0,10,0),camera_up=c(0,0,1), sample_method = "sobol_blue",
                 width=800,height=800,fov=0, ortho_dimensions=c(15,15))
}
```

`generate_complex_bevel` *Generate Complex 2D Bevel Profile for 3D Polygons*

Description

All arguments are recycled to the length of the longest argument, allowing for the generation of complex and repetitive bevel patterns without manual replication of argument values.

Usage

```
generate_complex_bevel(
  bevel_type,
  bevel_start = 0,
  bevel_end = 1,
  segment_height = 1,
  angle = 45,
  curve_points = 30,
  reverse = FALSE,
  flip = FALSE,
  manual_offsets = NULL,
  add_end_points = TRUE,
  plot_bevel = FALSE,
  overall_height = NA,
  set_minimum_zero = TRUE,
  zero_offset_epsilon = 1e-06
)
```

Arguments

`bevel_type`	Vector of bevel types. Options are: ‘"circular"‘, ‘"exp"‘, ‘"bump"‘, ‘"step"‘, ‘"block"‘, ‘"angled"‘. Note that for the ‘"step"‘ type, the transition occurs at ‘bevel_start‘.
`bevel_start`	Numeric vector of values between ‘0‘ and ‘1‘. Percentage distance in the interior the polygon at which to begin the corresponding ‘bevel_type‘. Note that for the ‘"step"‘ type, this is ignored.
`bevel_end`	Numeric vector of values between ‘0‘ and ‘1‘. Percentage distance in the interior the polygon at which to end the corresponding ‘bevel_type‘.
`segment_height`	Numeric vector. The maximum heights of each bevel, as measured from the initial height at the end of the previous bevel.

angle	Default 'NULL'. Numeric vector. Optional angle parameter in degrees for angular bevels (overrides values in 'max_height').
curve_points	Default '50'. Integer vector of number of points for each curve.
reverse	Default 'FALSE'. Whether to reverse each bevel.
flip	Default 'FALSE'. Whether to reverse each bevel horizontally.
manual_offsets	Default 'NULL', none. This will force the bevel to add a point (interpolating between the two nearest points) at the specified offsets. This is useful when you want to add points at specific distances along the curve.
add_end_points	Default 'TRUE'. Whether to ensure there is a point at zero and a point at one.
plot_bevel	Default 'FALSE'. Whether to plot the resulting bevel.
overall_height	Default 'NA'. Numeric value specifying the overall height of the curve.
set_minimum_zero	Default 'TRUE'. Whether to offset the lowest point of the bevel so it's at zero.
zero_offset_epsilon	Default '1e-5'. Amount to offset the bevel to ensure no self-intersection with the base.

Value

List containing 'x' and 'y', which are the coordinates of the complex 2D bevel profile

Examples

```
# Generate a complex bevel profile and plot it
complex_coords = generate_complex_bevel(
  bevel_type  = c("circular", "bump", "step", "block", "angled"),
  bevel_start = c(0,   0.2, 0.6, 0.7, 0.9),
  bevel_end   = c(0.2, 0.5, 0.7, 0.8, 1.0),
  segment_height  = c(0.1, 0.2, 0.2, 0.2, 0.4),
  angle = 45,
  curve_points = c(50, 50, 50, 1, 1),
  reverse = c(FALSE, TRUE, FALSE, FALSE, FALSE),
  plot_bevel = TRUE
)
# Create a step function with reverses to generate a square wave pattern
complex_coords = generate_complex_bevel(
  bevel_type  = "step",
  bevel_start = head(seq(0,1,by=0.05),-1),
  bevel_end   = 1,
  segment_height  = 0.1,
  angle = 45,
  reverse = c(FALSE, TRUE),
  plot_bevel = TRUE
)
#Generate an increasing sawtooth pattern with angles
complex_coords = generate_complex_bevel(
  bevel_type  = "angled",
  bevel_start = head(seq(0,1,by=0.05),-1),
  bevel_end   = tail(seq(0,1,by=0.05),-1),
```

```
  segment_height  = 0.1,
  angle = c(45,30),
  reverse = c(FALSE, TRUE),
  plot_bevel = TRUE
)
# Create a step function to turn polygons into a ziggurat (note bevel_end is ignored)
complex_coords = generate_complex_bevel(
  bevel_type  = "step",
  bevel_start = head(seq(0,1,by=0.05),-1),
  bevel_end   = 1,
  segment_height  = 0.1,
  reverse = FALSE,
  plot_bevel = TRUE
)
```

`generate_offset_polygon` *Generate an offset polygon*

Description

This function generates an interior offset polygon from a straight skeleton.

Usage

```
generate_offset_polygon(skeleton, offset, progress = FALSE)
```

Arguments

`skeleton`	Default 'NULL'. A straight skeleton generated from the 'skeletonize' function.
`offset`	Default 'NULL'. The offset(s) of the polygon.
`progress`	Default 'FALSE'. Whether to display a progress bar.

Value

A list of data frames, each representing a polygon offset by the specified amount.

Examples

```
# Simple polygon example
simple_poly = matrix(c(0,0, 3,0, 3,3, 0,3, 0,0), ncol=2, byrow=TRUE)
skeleton = skeletonize(simple_poly)
offset_polys = generate_offset_polygon(skeleton, c(0.25, 0.5))
print(offset_polys)

# Polygon with hole example
# Outer polygon
vertices = matrix(c(0,0, 7,0, 7,7, 0,7, 0,0), ncol = 2, byrow = TRUE)
# Holes inside the polygon
```

```
hole_1 = matrix(c(1,1, 2,1, 2,2, 1,2, 1,1), ncol = 2, byrow = TRUE)[5:1,]
hole_2 = matrix(c(5,5, 6,5, 6,6, 5,6, 5,5), ncol = 2, byrow = TRUE)[5:1,]
skeleton = skeletonize(vertices, holes = list(hole_1, hole_2))
plot_skeleton(skeleton)

#Generate three offsets
plot_offset_polygon(generate_offset_polygon(skeleton, c(0.25,0.75,1.5,2)))

#Generate many offsets
plot_offset_polygon(generate_offset_polygon(skeleton, seq(0,2.5,by=0.1)+0.05))

# Skeletonize and plot an {sf} object
if(length(find.package("spData",quiet = TRUE)) > 0) {
  us_states = spData::us_states
  texas = us_states[us_states$NAME == "Texas",]
  texas_skeleton = skeletonize(texas)
  plot_offset_polygon(generate_offset_polygon(texas_skeleton, seq(0, 2.5, by = 0.1)),
                      border = heat.colors,
                      linewidth = 1)
}
```

`generate_roof` *Generate a 3D roof model*

Description

This function generates a 3D roof model from a straight skeleton.

Usage

```
generate_roof(
  skeleton,
  max_height = NA,
  vertical_offset = 0,
  base = FALSE,
  base_height = 0,
  angle = 45,
  sides = FALSE,
  double_sided = FALSE,
  scale_all_max = FALSE,
  swap_yz = TRUE,
  progress = TRUE,
  material = material_list(),
  roof_material = NA,
  verbose = FALSE
)
```

Arguments

skeleton	Default ‘NULL‘. A straight skeleton generated from the ‘skeletonize‘ function.
max_height	Default ‘NA‘. The maximum height of the roof.
vertical_offset	Default ‘0‘. The vertical offset of the roof.
base	Default ‘TRUE‘. A logical flag that controls whether to generate the bottom of the roof.
base_height	Default ‘vertical_offset‘. Height of the base.
angle	Default ‘45‘. Angle of the roof.
sides	Default ‘FALSE‘. A logical flag on whether to draw the sides. This will automatically be set to ‘TRUE‘ if ‘base = TRUE‘ and the ‘base_height‘ is less than ‘vertical_offset‘.
double_sided	Default ‘FALSE‘. A logical flag that controls whether the polygon should be double-sided.
scale_all_max	Default ‘FALSE‘. If passing in a list of multiple skeletons with polygons, whether to scale each polygon to the overall max height, or whether to scale each max height to the maximum internal distance in the polygon.
swap_yz	Default ‘TRUE‘. A logical flag that controls whether to swap the y and z coordinates in the resulting mesh. If ‘TRUE‘, the y and z coordinates will be swapped.
progress	Default ‘TRUE‘. A logical flag to control whether a progress bar is displayed during roof generation.
material	Default ‘material_list()‘. Interface to set the color/appearance/material options for the resulting ‘ray_mesh‘ mesh.
roof_material	Default ‘NA‘, uses the material specified in ‘material‘. Interface to set the color/appearance/material options for the resulting ‘ray_mesh‘ rooftop mesh.
verbose	Default ‘FALSE‘. A logical flag to control whether additional timing information should be displayed.

Value

A 3D mesh of the roof model.

Examples

```
#Generate vertices and holes
vertices = matrix(c(0,0, 7,0, 7,7, 0,7, 0,0), ncol = 2, byrow = TRUE)-3.5
hole_1 = matrix(c(1,1, 2,1, 2,2, 1,2, 1,1), ncol = 2, byrow = TRUE)[5:1,]-3.5
hole_2 = matrix(c(5,5, 6,5, 6,6, 5,6, 5,5), ncol = 2, byrow = TRUE)[5:1,]-3.5
skeleton = skeletonize(vertices, holes = list(hole_1, hole_2))
if(run_documentation()) {
plot_skeleton(skeleton)
}

#Generate a roof model and specify the material
if(run_documentation()) {
```

```
  library(rayrender)
  library(rayvertex)
  roof_model = generate_roof(skeleton, material = material_list(diffuse="purple"))
  scene_base = xz_rect(xwidth=100,zwidth=100,
                       material=diffuse(color="grey20", checkercolor="white")) |>
    add_object(sphere(y=8,z=10,x=-3,material=light(intensity=100))) |>
    add_object(sphere(y=800,z=10,x=-3,radius=100,material=light(intensity=5)))

  raymesh_model(roof_model, override_material = FALSE) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,30,20), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}

# Change the maximum height of the roof
if(run_documentation()) {
  roof_model = generate_roof(skeleton, max_height=5)
  raymesh_model(roof_model, material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,30,20), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}

#Add a vertical_offset to the roof, without a base
if(run_documentation()) {
  roof_model = generate_roof(skeleton, vertical_offset = 2, base = FALSE)
  raymesh_model(roof_model, material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,10,20), lookat=c(0,2,0), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}

# Add a base
if(run_documentation()) {
  roof_model = generate_roof(skeleton, vertical_offset = 2, base = TRUE)
  raymesh_model(roof_model, material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,10,20), lookat=c(0,2,0), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}

# Change the base height (note that the vertical_offset is measured from the base, not from zero)
if(run_documentation()) {
  roof_model = generate_roof(skeleton, vertical_offset = 2, base = TRUE, base_height=1)
  raymesh_model(roof_model, material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    render_scene(lookfrom=c(10,10,20), lookat=c(0,2,0), sample_method = "sobol_blue",
                 width=800,height=800,fov=0,ortho_dimensions=c(10,10))
}


# Skeletonize and turn an {sf} object into a roof
if(run_documentation()) {
```

```
  us_states = spData::us_states
  cali = us_states[us_states$NAME == "California",]
  cali_skeleton = skeletonize(cali)
  plot_skeleton(cali_skeleton)
  roof_model_cali = generate_roof(cali_skeleton, max_height = 2) |>
    center_mesh() |>
    translate_mesh(c(0,1,0))

  raymesh_model(roof_model_cali, material = diffuse(color="purple")) |>
    add_object(scene_base) |>
    add_object(sphere(x=-10,z=-10,y=4,material=light(color="red", intensity=40))) |>
    add_object(sphere(x=10,z=-10,y=4,material=light(color="orange", intensity=40))) |>
    render_scene(lookfrom=c(0,10,-1), sample_method = "sobol_blue",
                 width=800,height=800,fov=0, ortho_dimensions=c(12,12))
}
```

`plot_offset_polygon` *Plot Offset Polygons*

Description

Plot the offset polygons generated by the ‘generate_offset_polygon‘ function.

Usage

```
plot_offset_polygon(
  offset_polygons,
  plot_original_polygon = TRUE,
  fill = NA,
  color = "dodgerblue",
  xlim = NULL,
  ylim = NULL,
  linewidth = 1,
  background = "white",
  plot_skeleton = FALSE,
  return_layers = FALSE,
  ...
)
```

Arguments

`offset_polygons`	Default ‘NULL‘. A ‘rayskeleton_polygon‘ or ‘rayskeleton_polygon_list‘ object, generated from ‘generate_offset_polygon()‘.
`plot_original_polygon`	Default ‘TRUE‘. Whether to plot the original polygon.
`fill`	Default ‘NULL‘. A color or palette function to generate the fill palette for the polygons’ interiors.

color	Default ‘grDevices::heat.colors‘. A color or palette function to generate the color palette for the offset polygons’ borders.
xlim	Default ‘NULL‘. The x-axis limits as a vector of two values (min, max). If ‘NULL‘, it calculates the limits from the data.
ylim	Default ‘NULL‘. The y-axis limits as a vector of two values (min, max). If ‘NULL‘, it calculates the limits from the data.
linewidth	Default ‘1‘. The linewidth of the polygon.
background	Default ‘"white"‘. Background color.
plot_skeleton	Default ‘FALSE‘. Whether to plot the straight skeleton.
return_layers	Default ‘FALSE‘, plots the figure. If ‘TRUE‘, this will instead return a list of the ggplot layers.
...	Additional arguments to pass to ‘plot_skeleton()‘ if ‘plot_skeleton = TRUE‘

Value

A plot showing the offset polygons.

Examples

```
# Outer polygon
vertices = matrix(c(0,0, 7,0, 7,7, 0,7, 0,0), ncol = 2, byrow = TRUE)
# Holes inside the polygon
hole_1 = matrix(c(1,1, 2,1, 2,2, 1,2, 1,1), ncol = 2, byrow = TRUE)[5:1,]
hole_2 = matrix(c(5,5, 6,5, 6,6, 5,6, 5,5), ncol = 2, byrow = TRUE)[5:1,]
skeleton = skeletonize(vertices, holes = list(hole_1, hole_2))
plot_skeleton(skeleton)

#Generate three offsets with the skeleton
plot_offset_polygon(generate_offset_polygon(skeleton, c(0.25,0.75,1.5,2)), plot_skeleton = TRUE)

#Generate many offsets
plot_offset_polygon(generate_offset_polygon(skeleton, seq(0.05,2.55,by=0.1)))

#Pass a palette
plot_offset_polygon(generate_offset_polygon(skeleton, seq(0.05,2.55,by=0.1)),
                    color = heat.colors)

#Pass colors manually (colors in excess of the number of offsets are ignored)
plot_offset_polygon(generate_offset_polygon(skeleton, seq(0.05,2.55,by=0.1)),
                    color = rep(c("red","red","blue","blue"),100))

# Skeletonize and plot an {sf} object
if(length(find.package("spData",quiet = TRUE)) > 0) {
  us_states = spData::us_states
  texas = us_states[us_states$NAME == "Texas",]
  texas_skeleton = skeletonize(texas)
  plot_offset_polygon(generate_offset_polygon(texas_skeleton, seq(0, 2.5, by = 0.1)),
                      color = heat.colors,
                      linewidth = 1)
}
```

plot_skeleton *Plot the Straight Skeleton of a Polygon*

Description

This function visualizes the straight skeleton derived from a given polygon. The original polygon (with holes if present) is plotted in black, while the straight skeleton is plotted in red.

Usage

```
plot_skeleton(
  skeleton,
  use_arrow = TRUE,
  use_points = TRUE,
  xlim = c(0, 1),
  ylim = c(0, 1),
  arrow_color = "red",
  polygon_color = "black",
  size = 1,
  arrow_size = 0.05,
  highlight_links = NULL,
  highlight_color = "green",
  return_layers = FALSE
)
```

Arguments

skeleton	A list object of class 'rayskeleton' containing the straight skeleton details. It should have 'nodes' and 'links' as its primary components.
use_arrow	Default 'TRUE'. A logical value indicating whether or not to use arrows to represent the links of the straight skeleton. Default is TRUE.
use_points	Default 'TRUE'. Whether to plot the vertex points as well.
xlim	Default'c(0,1)'. A numeric vector of length 2 specifying the x-limits of the plot in the form 'c(min, max)'. These are proportional limits relative to the bounding box around the skeleton.
ylim	Default 'c(0,1)'. A numeric vector of length 2 specifying the y-limits of the plot in the form c(min, max). These are proportional limits relative to the bounding box around the skeleton.
arrow_color	Default '"red"'. Color of the arrows.
polygon_color	Default '"black"'. Color of the polygon.
size	Default '1'. Size of the vertex points.
arrow_size	Default '1'. Scales the arrow size.

highlight_links
: Default 'NULL'. A numeric vector indicating which links (by their index) to highlight. If specified, the corresponding links will be colored with the 'highlight_color'.

highlight_color
: Default '"purple"'. Color of the highlighted links.

return_layers
: Default 'FALSE', plots the figure. If 'TRUE', this will instead return a list of the ggplot layers.

Details

The function uses the 'ggplot2' package for plotting. The straight skeleton is visualized based on the details provided in the 'skeleton' object. The original polygon and holes are plotted based on attributes stored in the 'skeleton' object.

Value

A ggplot object visualizing the straight skeleton and the original polygon.

Examples

```
# Assuming skeleton1 is already defined as in the previous example
# Outer polygon
vertices = matrix(c(0,0, 7,0, 7,7, 0,7, 0,0), ncol = 2, byrow = TRUE)
# Holes inside the polygon
hole1 = matrix(c(1,1, 1,2, 2,2, 2,1, 1,1), ncol = 2, byrow = TRUE)
hole2 = matrix(c(5,5, 5,6, 6,6, 6,5, 5,5), ncol = 2, byrow = TRUE)
skeleton = skeletonize(vertices, holes = list(hole1, hole2))
if(length(find.package("ggplot2",quiet = TRUE)) > 0) {
  plot_skeleton(skeleton)
}
# Skeletonize and plot an {sf} object
if(length(find.package("spData",quiet = TRUE)) > 0) {
  us_states = spData::us_states
  texas = us_states[us_states$NAME == "Texas",]
  plot_skeleton(skeletonize(texas))
}
# Highlighting certain links in the skeleton
max_links =which(skeleton$links$destination_time == max(skeleton$links$destination_time))
if(length(find.package("ggplot2",quiet = TRUE)) > 0) {
  plot_skeleton(skeleton, highlight_links = max_links, highlight_color = "green")
}
```

---

run_documentation *Run Documentation*

Description

This function determines if the examples are being run in pkgdown. It is not meant to be called by the user.

Usage

```
run_documentation()
```

Value

Boolean value.

Examples

```
# See if the documentation should be run.
run_documentation()
```

`skeletonize` *Skeletonize a polygon*

Description

This function generates a straight skeleton of a polygon, based on a set of vertices and holes. It uses the CGAL library to create the straight skeleton using exact arithmetic, and then parses that file into a more manageable format.

Usage

```
skeletonize(
  vertices,
  holes = list(),
  debug = FALSE,
  merge_nodes_tolerance = 1e-05,
  return_raw_ss = FALSE,
  progress = TRUE
)
```

Arguments

`vertices`	Default 'NULL'. A matrix of x and y coordinates representing the vertices of the polygon in counter-clockwise (CCW) order.
`holes`	Default 'list()'. A list of matrices, each representing a hole in the polygon with x and y coordinates in clockwise (CW) order.
`debug`	Default 'FALSE'. A logical flag that controls whether debugging information should be printed.
`merge_nodes_tolerance`	Default '1e-5'. A numeric value specifying the tolerance level for merging nodes. It should be a value between 0 and 1. This value species the size of the grid that the nodes are snapped to determining identical nodes.
`return_raw_ss`	Default 'FALSE'. A logical flag that controls whether the raw straight skeleton should be returned.
`progress`	Default 'TRUE'. A logical flag that controls whether a progress bar should be displayed while skeletonizing.

Value

If ‘return_raw_ss‘ is FALSE, a list with two data frames, ’nodes’ and ’links’, which represent the nodes and edges of the straight skeleton, respectively. If ‘return_raw_ss‘ is TRUE, a data frame representing the raw straight skeleton is returned. If the polygon is not simple, a warning is issued and NULL is returned.

Examples

```
# Example 1: Simple rectangle polygon with no holes
vertices1 = matrix(c(0,0, 4,0, 4,3, 0,3, 0,0), ncol = 2, byrow = TRUE)
skeleton1 = skeletonize(vertices1)
plot_skeleton(skeleton1)

# Example 2: Triangle polygon with no holes
vertices2 = matrix(c(0,0, 2,0, 1,2, 0,0), ncol = 2, byrow = TRUE)
skeleton2 = skeletonize(vertices2)
plot_skeleton(skeleton2)

# Example 3: Polygon with a hole
# Outer polygon
vertices3 = matrix(c(0,0, 5,0, 5,5, 0,5, 0,0), ncol = 2, byrow = TRUE)
# Hole inside the polygon
hole3 = matrix(c(1,1, 4,1, 4,4, 1,4, 1,1), ncol = 2, byrow = TRUE)[5:1,]
skeleton3 = skeletonize(vertices3, holes = list(hole3))
plot_skeleton(skeleton3)

# Example 4: Polygon with multiple holes
# Outer polygon
vertices4 = matrix(c(0,0, 7,0, 7,7, 0,7, 0,0), ncol = 2, byrow = TRUE)
# Holes inside the polygon
hole4_1 = matrix(c(1,1, 2,1, 2,2, 1,2, 1,1), ncol = 2, byrow = TRUE)[5:1,]
hole4_2 = matrix(c(5,5, 6,5, 6,6, 5,6, 5,5), ncol = 2, byrow = TRUE)[5:1,]
skeleton4 = skeletonize(vertices4, holes = list(hole4_1, hole4_2))
plot_skeleton(skeleton4)

# Example 5: Using debug and returning raw straight skeleton
vertices5 = matrix(c(0,0, 3,0, 3,3, 0,3, 0,0), ncol = 2, byrow = TRUE)
raw_skeleton5 = skeletonize(vertices5, debug = TRUE, return_raw_ss = TRUE)

# Skeletonize and plot an {sf} object
if(length(find.package("spData",quiet = TRUE)) > 0) {
  us_states = spData::us_states
  texas = us_states[us_states$NAME == "Texas",]
  plot_skeleton(skeletonize(texas))
}
```

Index