Package: rayrender (via r-universe)

October 18, 2026

Type Package

Title Build and Raytrace 3D Scenes

Version 0.42.0

Date 2026-09-15

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

Description Render scenes using pathtracing. Build 3D scenes out of spheres, cubes, planes, disks, triangles, cones, curves, line segments, cylinders, ellipsoids, and 3D models in the 'Wavefront' OBJ file format or the PLY Polygon File Format. Supports several material types, textures, multicore rendering, and tone-mapping. Based on the ``Ray Tracing in One Weekend'' book series. Peter Shirley (2018) <https://raytracing.github.io>.

License GPL-3

Copyright file inst/COPYRIGHTS

Depends R (>= 4.3.0)

Imports Rcpp (>= 1.0.0), parallel, png, raster, decido, rayimage (>= 0.27.1), stats, progress, rayvertex (>= 0.15.0), withr, vctrs, cli, pillar, skymodelr (>= 0.6.3),

Suggests ambient, knitr, rmarkdown, sf, spData, dplyr, Rvcg, testthat (>= 3.2.3), tibble, tree3d, rayshader (>= 0.38.13), xml2, rgl

LinkingTo Rcpp, RcppThread (>= 2.4.0), progress, spacefillr (>= 0.3.0), testthat, libopenexr (>= 3.4.12-6), libimath (>= 3.2.2), skymodelr (>= 0.6.3)

URL https://www.rayrender.net, https://github.com/tylermorganwall/rayrender, http://www.rayrender.net/

SystemRequirements C++20

Biarch true

Encoding UTF-8

Config/testthat/edition 3

Config/build/compilation-database true

Config/roxygen2/version 8.0.0

NeedsCompilation yes

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

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

Date/Publication 2026-09-16 06:11:00 UTC

RemoteUrl https://github.com/cran/rayrender

RemoteRef HEAD

RemoteSha 89a800560fb5e2d62b6add18581d083045639bc5

Contents

add_camera . . . 4
add_infinite_light . . . 5
add_object . . . 5
animate_objects . . . 6
arrow . . . 8
bezier_curve . . . 11
camera . . . 13
cloud . . . 17
cone . . . 20
create_instances . . . 22
csg_box . . . 25
csg_capsule . . . 26
csg_combine . . . 27
csg_cone . . . 29
csg_cylinder . . . 30
csg_ellipsoid . . . 31
csg_elongate . . . 32
csg_group . . . 34
csg_object . . . 35
csg_onion . . . 36
csg_plane . . . 38
csg_pyramid . . . 39
csg_rotate . . . 40
csg_round . . . 41
csg_rounded_cone . . . 42
csg_scale . . . 43
csg_sphere . . . 44
csg_torus . . . 45
csg_translate . . . 46
csg_triangle . . . 47

cube . . . 48
cylinder . . . 50
dielectric . . . 51
diffuse . . . 53
disk . . . 56
ellipsoid . . . 58
extruded_path . . . 59
extruded_polygon . . . 64
generate_camera_motion . . . 70
generate_cornell . . . 74
generate_ground . . . 75
generate_studio . . . 76
get_camera . . . 77
get_infinite_light . . . 78
get_saved_keyframes . . . 78
glossy . . . 79
grid_medium . . . 82
group_objects . . . 84
hair . . . 85
has_denoiser . . . 87
homogeneous_medium . . . 88
infinite_light . . . 90
lambertian . . . 93
light . . . 93
list_cameras . . . 95
list_infinite_lights . . . 96
mesh3d_model . . . 96
metal . . . 98
microfacet . . . 101
nanovdb_medium . . . 105
obj_model . . . 106
path . . . 109
pig . . . 112
ply_model . . . 114
preview_camera . . . 115
r_obj . . . 116
raymesh_model . . . 117
remove_camera . . . 120
remove_infinite_light . . . 121
render_animation . . . 121
render_ao . . . 127
render_preview . . . 130
render_scene . . . 131
screen_line . . . 139
screen_text . . . 142
segment . . . 146
set_active_camera . . . 149
set_medium . . . 149

set_scene_material 150
sky_light 151
sky_light_image 161
sphere 165
sun_light 167
text3d 169
triangle 171
xy_rect 173
xz_rect 175
yz_rect 176

Index **178**

`add_camera` *Add Camera*

Description

Adds a camera to a 'ray_scene'.

Usage

```
add_camera(scene, camera, name = NULL, active = TRUE, replace = FALSE)
```

Arguments

`scene`	Scene to modify.
`camera`	Camera created with 'camera()'.
`name`	Default 'NULL'. Optional name that overrides 'camera$name'.
`active`	Default 'TRUE'. Whether to set this camera as the active scene camera.
`replace`	Default 'FALSE'. Whether to replace an existing camera with the same name.

Value

A modified 'ray_scene'.

Examples

```
scene = generate_ground(material=diffuse(color="grey20")) |>
  add_object(sphere()) |>
  add_camera(camera(
    name = "main",
    lookfrom = c(0, 1, -10),
    lookat = c(0, 0, 0),
    fov = 35
  ))

render_scene(scene, samples = 16)
```

add_infinite_light *Add an Infinite Light*

Description

Add an Infinite Light

Usage

```
add_infinite_light(scene, light, name = NULL, replace = FALSE)
```

Arguments

scene	Scene to modify.
light	Light created with infinite_light(), sky_light(), sky_light_image(), sun_light(), or moon_light().
name	Default NULL. Optional name overriding the light's name.
replace	Default FALSE. Replace an existing light with the same name.

Value

A modified scene.

add_object *Add Object*

Description

Add Object

Usage

```
add_object(scene, objects = NULL)
```

Arguments

scene	Tibble of pre-existing object locations and properties.
objects	A tibble row or collection of rows representing each object.

Value

Tibble of object locations and properties.

Examples

```
#Generate the ground and add some objects
scene = generate_ground(depth=-0.5,material = diffuse(checkercolor="blue")) |>
  add_object(cube(x=0.7,
              material=diffuse(noise=5,noisecolor="purple",color="black",noisephase=45),
                  angle=c(0,-30,0))) |>
  add_object(sphere(x=-0.7,radius=0.5,material=metal(color="gold")))
render_scene(scene,parallel=TRUE)
```

`animate_objects` *Animate Objects*

Description

This function animates an object between two states. This animates objects separately from the transformations set in 'group_objects()' and in the object transformations themselves. This creates motion blur, controlled by the shutter open/close options in 'render_scene()'.

Usage

```
animate_objects(
  scene,
  start_time = 0,
  end_time = 1,
  start_pivot_point = c(0, 0, 0),
  start_position = c(0, 0, 0),
  start_angle = c(0, 0, 0),
  start_order_rotation = c(1, 2, 3),
  start_scale = c(1, 1, 1),
  start_axis_rotation = NA,
  end_pivot_point = c(0, 0, 0),
  end_position = c(0, 0, 0),
  end_angle = c(0, 0, 0),
  end_order_rotation = c(1, 2, 3),
  end_scale = c(1, 1, 1),
  end_axis_rotation = NA
)
```

Arguments

`scene`	Tibble of pre-existing object locations.
`start_time`	Default '0'. Start time of movement.
`end_time`	Default '1'. End time of movement.
`start_pivot_point`	Default 'c(0,0,0)'. The point about which to pivot, scale, and move the objects.

`start_position` Default ‘c(0,0,0)‘. Vector indicating where to offset the objects.

`start_angle` Default ‘c(0,0,0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.

`start_order_rotation`
Default ‘c(1,2,3)‘. The order to apply the rotations, referring to "x", "y", and "z".

`start_scale` Default ‘c(1,1,1)‘. Scaling factor for x, y, and z directions for all objects.

`start_axis_rotation`
Default ‘NA‘. Provide an axis of rotation and a single angle (via ‘angle‘) of rotation

`end_pivot_point`
Default ‘c(0,0,0)‘. The point about which to pivot, scale, and move the group.

`end_position` Default ‘c(0,0,0)‘. Vector indicating where to offset the objects.

`end_angle` Default ‘c(0,0,0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.

`end_order_rotation`
Default ‘c(1,2,3)‘. The order to apply the rotations, referring to "x", "y", and "z".

`end_scale` Default ‘c(1,1,1)‘. Scaling factor for x, y, and z directions for all objects.

`end_axis_rotation`
Default ‘NA‘. Provide an axis of rotation and a single angle (via ‘angle‘) of rotation around that axis.

Value

Tibble of animated object.

Examples

```
#Render a pig
generate_studio() |>
  add_object(pig(y=-1.2,scale=0.5,angle=c(0,110,0)))|>
  add_object(sphere(y=5,x=5,z=-5,radius=2,material=light())) |>
  render_scene(samples=16,sample_method = "sobol_blue")
#Render a moving pig
generate_studio() |>
  add_object(
    animate_objects(
      pig(y=-1.2,scale=0.5,angle=c(0,110,0)),
      start_position = c(-0.1,0,0), end_position = c(0.1,0.2,0))
  ) |>
  add_object(sphere(y=5,x=5,z=-5,radius=2,material=light())) |>
  render_scene(samples=16,sample_method = "sobol_blue",clamp_value = 10)

#Render a shrinking pig
generate_studio() |>
  add_object(
    animate_objects(
```

```
      pig(y=-1.2,scale=0.5,angle=c(0,110,0)),
      start_scale = c(1,1,1), end_scale = c(0.5,0.5,0.5))
  ) |>
  add_object(sphere(y=5,x=5,z=-5,radius=2,material=light())) |>
  render_scene(samples=16,sample_method = "sobol_blue",clamp_value = 10)
#Render a spinning pig
generate_studio() |>
  add_object(
    animate_objects(
      pig(y=-1.2,scale=0.5,angle=c(0,110,0)),
      start_angle = c(0,-30,0), end_angle = c(0,30,0))
  ) |>
  add_object(sphere(y=5,x=5,z=-5,radius=2,material=light())) |>
  render_scene(samples=16,sample_method = "sobol_blue",clamp_value = 10)

#Shorten the open shutter time frame
generate_studio() |>
  add_object(
    animate_objects(
      pig(y=-1.2,scale=0.5,angle=c(0,110,0)),
      start_angle = c(0,-30,0), end_angle = c(0,30,0))
  ) |>
  add_object(sphere(y=5,x=5,z=-5,radius=2,material=light())) |>
  render_scene(samples=16,sample_method = "sobol_blue",clamp_value = 10,
               shutteropen=0.4, shutterclose = 0.6)
#Change the time frame when the shutter is open
generate_studio() |>
  add_object(
    animate_objects(
      pig(y=-1.2,scale=0.5,angle=c(0,110,0)),
      start_angle = c(0,-30,0), end_angle = c(0,30,0))
  ) |>
  add_object(sphere(y=5,x=5,z=-5,radius=2,material=light())) |>
  render_scene(samples=16,sample_method = "sobol_blue",clamp_value = 10,
               shutteropen=0, shutterclose = 0.1)
#Shorten the time span in which the movement occurs (which, in effect,
#increases the speed of the transition).
generate_studio() |>
  add_object(
    animate_objects(start_time = 0, end_time=0.1,
      pig(y=-1.2,scale=0.5,angle=c(0,110,0)),
      start_angle = c(0,-30,0), end_angle = c(0,30,0))
  ) |>
  add_object(sphere(y=5,x=5,z=-5,radius=2,material=light())) |>
  render_scene(samples=16,sample_method = "sobol_blue",clamp_value = 10,
               shutteropen=0, shutterclose = 0.1)
```

`arrow` *Arrow Object*

Description

Composite object (cone + segment)

Usage

```
arrow(
  start = c(0, 0, 0),
  end = c(0, 1, 0),
  radius_top = 0.2,
  radius_tail = 0.1,
  tail_proportion = 0.5,
  direction = NA,
  from_center = TRUE,
  material = diffuse(),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

`start`	Default ‘c(0, 0, 0)‘. Base of the arrow, specifying ‘x‘, ‘y‘, ‘z‘.
`end`	Default ‘c(0, 1, 0)‘. Tip of the arrow, specifying ‘x‘, ‘y‘, ‘z‘.
`radius_top`	Default ‘0.5‘. Radius of the top of the arrow.
`radius_tail`	Default ‘0.2‘. Radius of the tail of the arrow.
`tail_proportion`	Default ‘0.5‘. Proportion of the arrow that is the tail.
`direction`	Default ‘NA‘. Alternative to ‘start‘ and ‘end‘, specify the direction (via a length-3 vector) of the arrow. Arrow will be centered at ‘start‘, and the length will be determined by the magnitude of the direction vector.
`from_center`	Default ‘TRUE‘. If orientation specified via ‘direction‘, setting this argument to ‘FALSE‘ will make ‘start‘ specify the bottom of the cone, instead of the middle.
`material`	Default `diffuse`.The material, called from one of the material functions `diffuse`, `metal`, or `dielectric`.
`flipped`	Default ‘FALSE‘. Whether to flip the normals.
`scale`	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Notes: this will change the stated start/end position of the cone. Emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the cone in the scene.

Examples

```
#Draw a simple arrow from x = -1 to x = 1
generate_studio() |>
  add_object(arrow(start = c(-1,0,0), end = c(1,0,0), material=glossy(color="red"))) |>
  add_object(sphere(y=5,material=light(intensity=20))) |>
  render_scene(clamp_value=10,  samples=16)
#Change the proportion of tail to top
generate_studio(depth=-2) |>
  add_object(arrow(start = c(-1,-1,0), end = c(1,-1,0), tail_proportion = 0.5,
                   material=glossy(color="red"))) |>
  add_object(arrow(start = c(-1,0,0), end = c(1,0,0), tail_proportion = 0.75,
                   material=glossy(color="red"))) |>
  add_object(arrow(start = c(-1,1,0), end = c(1,1,0), tail_proportion = 0.9,
                   material=glossy(color="red"))) |>
  add_object(sphere(y=5,z=-5,x=2,material=light(intensity=30))) |>
  render_scene(clamp_value=10, fov=25,  samples=16)
#Change the radius of the tail/top segments
generate_studio(depth=-1.5) |>
  add_object(arrow(start = c(-1,-1,0), end = c(1,-1,0), tail_proportion = 0.75,
                   radius_top = 0.1, radius_tail=0.03,
                   material=glossy(color="red"))) |>
  add_object(arrow(start = c(-1,0,0), end = c(1,0,0), tail_proportion = 0.75,
                   radius_top = 0.2, radius_tail=0.1,
                   material=glossy(color="red"))) |>
  add_object(arrow(start = c(-1,1,0), end = c(1,1,0), tail_proportion = 0.75,
                   radius_top = 0.3, radius_tail=0.2,
                   material=glossy(color="red"))) |>
  add_object(sphere(y=5,z=-5,x=2,material=light(intensity=30))) |>
  render_scene(clamp_value=10, samples=16)
#We can also specify arrows via a midpoint and direction:
generate_studio(depth=-1) |>
  add_object(arrow(start = c(-1,-0.5,0), direction = c(0,0,1),
                   material=glossy(color="green"))) |>
  add_object(arrow(start = c(1,-0.5,0), direction = c(0,0,-1),
                   material=glossy(color="red"))) |>
  add_object(arrow(start = c(0,-0.5,1), direction = c(1,0,0),
                   material=glossy(color="yellow"))) |>
  add_object(arrow(start = c(0,-0.5,-1), direction = c(-1,0,0),
                   material=glossy(color="purple"))) |>
  add_object(sphere(y=5,z=-5,x=2,material=light(intensity=30))) |>
  render_scene(clamp_value=10, samples=16,
               lookfrom=c(0,5,10), lookat=c(0,-0.5,0), fov=16)
#Plot a 3D vector field for a gravitational well:

r = 1.5
theta_vals = seq(0,2*pi,length.out = 16)[-16]
phi_vals = seq(0,pi,length.out = 16)[-16][-1]
arrow_list = list()
counter = 1
for(theta in theta_vals) {
  for(phi in phi_vals) {
    rval = c(r*sin(phi)*cos(theta),r*cos(phi),r*sin(phi)*sin(theta))
```

```
    arrow_list[[counter]] = arrow(rval, direction = -1/2*rval/sqrt(sum(rval*rval))^3,
                                  tail_proportion = 0.66, radius_top=0.03, radius_tail=0.01,
                                      material = diffuse(color="red"))
    counter = counter + 1
  }
}
vector_field = do.call(rbind,arrow_list)
sphere(material=diffuse(noise=1,color="blue",noisecolor="darkgreen")) |>
  add_object(vector_field) |>
  add_object(sphere(y=0,x=10,z=-5,material=light(intensity=200))) |>
  render_scene(fov=20, ambient=TRUE, samples=16,
               backgroundlow="black",backgroundhigh="white")
```

`bezier_curve` *Bezier Curve Object*

Description

Bezier curve, defined by 4 control points.

Usage

```
bezier_curve(
  p1 = c(0, 0, 0),
  p2 = c(-1, 0.33, 0),
  p3 = c(1, 0.66, 0),
  p4 = c(0, 1, 0),
  x = 0,
  y = 0,
  z = 0,
  width = 0.1,
  width_end = NA,
  u_min = 0,
  u_max = 1,
  type = "cylinder",
  normal = c(0, 0, -1),
  normal_end = NA,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

`p1`	Default 'c(0,0,0)'. First control point. Can also be a list of 4 length-3 numeric vectors or 4x3 matrix/data.frame specifying the x/y/z control points.

p2	Default ‘c(-1,0.33,0)‘. Second control point.
p3	Default ‘c(1,0.66,0)‘. Third control point.
p4	Default ‘c(0,1,0)‘. Fourth control point.
x	Default ‘0‘. x-coordinate offset for the curve.
y	Default ‘0‘. y-coordinate offset for the curve.
z	Default ‘0‘. z-coordinate offset for the curve.
width	Default ‘0.1‘. Curve width.
width_end	Default ‘NA‘. Width at end of path. Same as ‘width‘, unless specified.
u_min	Default ‘0‘. Minimum parametric coordinate for the curve.
u_max	Default ‘1‘. Maximum parametric coordinate for the curve.
type	Default ‘cylinder‘. Other options are ‘flat‘ and ‘ribbon‘.
normal	Default ‘c(0,0,-1)‘. Orientation surface normal for the start of ribbon curves.
normal_end	Default ‘NA‘. Orientation surface normal for the start of ribbon curves. If not specified, same as ‘normal‘.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the cube in the scene.

Examples

```
#Generate the default curve:
generate_studio(depth=-0.2) |>
  add_object(bezier_curve(material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,radius=0.3,
                    material=light(intensity=200, spotlight_focus = c(0,0.5,0)))) |>
  render_scene(clamp_value = 10, lookat = c(0,0.5,0), fov=13,
               samples=16)

#Change the control points to change the direction of the curve. Here, we place spheres
#at the control point locations.
generate_studio(depth=-0.2) |>
  add_object(bezier_curve(material=diffuse(color="red"))) |>
  add_object(sphere(radius=0.075,material=glossy(color="green"))) |>
  add_object(sphere(radius=0.075,x=-1,y=0.33,material=glossy(color="green"))) |>
```

```
  add_object(sphere(radius=0.075,x=1,y=0.66,material=glossy(color="green"))) |>
  add_object(sphere(radius=0.075,y=1,material=glossy(color="green"))) |>
  add_object(sphere(y=3,z=-5,x=2,radius=0.3,
                    material=light(intensity=200, spotlight_focus = c(0,0.5,0)))) |>
  render_scene(clamp_value = 10, lookat = c(0,0.5,0), fov=15,
               samples=16)
#We can make the curve flat (always facing the camera) by setting the type to `flat`
generate_studio(depth=-0.2) |>
  add_object(bezier_curve(type="flat", material=glossy(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,radius=0.3,
                    material=light(intensity=200, spotlight_focus = c(0,0.5,0)))) |>
  render_scene(clamp_value = 10, lookat = c(0,0.5,0), fov=13,
               samples=16)
#We can also plot a ribbon, which is further specified by a start and end orientation with
#two surface normals.
generate_studio(depth=-0.2) |>
  add_object(bezier_curve(type="ribbon", width=0.2,
                    p1 = c(0,0,0), p2 = c(0,0.33,0), p3 = c(0,0.66,0), p4 = c(0.3,1,0),
                    normal_end = c(0,0,1),
                    material=glossy(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,radius=0.3,
                    material=light(intensity=200, spotlight_focus = c(0,0.5,0)))) |>
  render_scene(clamp_value = 10, lookat = c(0,0.5,0), fov=13,
               samples=16)
#Create a single curve and copy and rotate it around the y-axis to create a wavy fountain effect:
scene_curves = list()
for(i in 1:90) {
  scene_curves[[i]] = bezier_curve(p1 = c(0,0,0),p2 = c(0,5-sinpi(i*16/180),2),
                            p3 = c(0,5-0.5 * sinpi(i*16/180),4),p4 = c(0,0,6),
                            angle=c(0,i*4,0), type="cylinder",
                            width = 0.1, width_end =0.1,material=glossy(color="red"))
}
all_curves = do.call(rbind, scene_curves)
generate_ground(depth=0,material=diffuse(checkercolor="grey20")) |>
  add_object(all_curves) |>
  add_object(sphere(y=7,z=0,x=0,material=light(intensity=100))) |>
  render_scene(lookfrom = c(12,20,50),samples=100,
               lookat=c(0,1,0), fov=15, clamp_value = 10)
```

`camera` *Camera*

Description

Creates a rayrender camera that can be attached to a ‘ray_scene‘ with ‘add_camera()‘.

Usage

```
camera(
```

```
  lookfrom = c(0, 1, -10),
  lookat = c(0, 0, 0),
  camera_up = c(0, 1, 0),
  fov = 20,
  aperture = 0.1,
  focal_distance = NULL,
  ortho_dimensions = c(1, 1),
  motion = NULL,
  keyframe_motion_args = list(),
  name = "camera",
  filename = NA_character_,
  camera_description_file = NA,
  camera_scale = 1,
  iso = 100,
  film_size = 22,
  shutteropen = 0,
  shutterclose = 1,
  camera_motion_blur = FALSE,
  shutter_speed = 2
)
```

Arguments

lookfrom	Default ‘c(0, 1, -10)‘. Location of the camera.
lookat	Default ‘c(0, 0, 0)‘. Location where the camera is pointed.
camera_up	Default ‘c(0, 1, 0)‘. Vector indicating the up direction of the camera.
fov	Default ‘20‘. Field of view, in degrees.
aperture	Default ‘0.1‘. Aperture of the camera.
focal_distance	Default ‘NULL‘. Focal distance. If ‘NULL‘, this is the distance between ‘lookfrom‘ and ‘lookat‘.
ortho_dimensions	Default ‘c(1, 1)‘. Width and height of the orthographic camera when ‘fov = 0‘.
motion	Default ‘NULL‘. Camera motion data frame from ‘generate_camera_motion()‘.
keyframe_motion_args	Default ‘list()‘. Named list of additional arguments passed to ‘generate_camera_motion()‘ when pressing ‘M‘ in interactive preview to preview the saved keyframes. The saved keyframes always supply the camera positions. Defaults are ‘type = "spline"‘, 30 frames per saved keyframe, and ‘damp_motion = TRUE‘. Press Shift-L in the preview window to toggle the current path between open and closed.
name	Default ‘"camera"‘. Camera name.
filename	Default ‘NA_character_‘. Optional output filename or animation filename pattern.
camera_description_file	Default ‘NA‘. Filename of a realistic camera description file.
camera_scale	Default ‘1‘. Amount to scale a realistic camera.

`iso` Default '100'. Camera exposure.

`film_size` Default '22'. Film size in millimeters for realistic cameras.

`shutteropen` Default '0'. Time at which the shutter opens.

`shutterclose` Default '1'. Time at which the shutter closes.

`camera_motion_blur` Default 'FALSE'. Whether to blur animated camera movement over the shutter interval.

`shutter_speed` Default '2'. Frame-relative shutter speed controlling motion blur. A value of '1' samples the full frame-to-frame motion interval, '2' samples one-half, and '4' samples one-quarter. Higher values produce less motion blur. 'Inf' disables temporal motion blur.

Value

A 'ray_camera' object.

Examples

```
# Static camera attached to the scene, equivalent to passing lookfrom/lookat
# directly to render_scene().
scene = generate_ground(depth = -0.5, material = diffuse(checkercolor = "blue")) |>
  add_object(sphere(y = 0.5, radius = 0.5, material = diffuse(color = "red"))) |>
  add_camera(camera(
    name = "main",
    lookfrom = c(7, 1.5, 10),
    lookat = c(0, 0.5, 0),
    fov = 15,
    filename = NA_character_
  ))
render_scene(scene, samples = 16, parallel = TRUE)

# Shutter speed is frame-relative: higher values sample a smaller fraction
# of camera/object motion without changing exposure.
fast_shutter_camera = camera(
  lookfrom = c(7, 1.5, 10),
  lookat = c(0, 0.5, 0),
  shutter_speed = 4
)

# Animated camera attached to the scene, equivalent to passing camera_motion
# directly to render_animation().
camera_pos = list(c(0, 1, 15), c(5, -5, 5), c(-5, 5, -5), c(0, 1, -15))
camera_motion = generate_camera_motion(
  positions = camera_pos,
  lookats = camera_pos,
  offset_lookat = 1,
  fovs = 80,
  frames = 12,
  type = "bezier"
)
```

```
animated_scene = generate_ground(material = diffuse(checkercolor = "grey20"), depth = -10) |>
  add_object(sphere(y = 50, radius = 10, material = light(intensity = 30))) |>
  add_object(path(camera_pos, y = -0.2, material = diffuse(color = "red"))) |>
  add_camera(camera(
    name = "flythrough_blur",
    motion = camera_motion,
    camera_motion_blur = TRUE,
    shutter_speed = 2
  )) |>
  add_camera(camera(
    name = "flythrough_no_blur",
    motion = camera_motion,
    camera_motion_blur = FALSE
  )) |>
  add_camera(camera(
    name = "flythrough_medium_blur",
    motion = camera_motion,
    camera_motion_blur = FALSE,
    shutter_speed = 4
  ))
#We can render these individual cameras by calling out their specific name in render_scene()
#With no blur
render_scene(
  animated_scene,
  camera = "flythrough_no_blur",
  mode = "animation",
  samples = 16,
  start_frame = 1,
  end_frame = 2,
  sample_method = "sobol_blue",
  clamp_value = 10,
  width = 400,
  height = 400
)
#Now, with blur
render_scene(
  animated_scene,
  camera = "flythrough_blur",
  mode = "animation",
  samples = 16,
  start_frame = 1,
  end_frame = 2,
  sample_method = "sobol_blue",
  clamp_value = 10,
  width = 400,
  height = 400
)
#Now, with less blur
#' #Now, with blur
render_scene(
  animated_scene,
  camera = "flythrough_medium_blur",
  mode = "animation",
```

```
  samples = 16,
  start_frame = 1,
  end_frame = 2,
  sample_method = "sobol_blue",
  clamp_value = 10,
  width = 400,
  height = 400
)
```

---

cloud *Procedural Cloud Object*

Description

Create a cloud volume with billowing Perlin-noise density. Add it to a scene with `add_object()`; rendering automatically selects `integrator_type = "nee"`. Requires the suggested package `ambient` to generate the density field.

Usage

```
cloud(
  x = 0,
  y = 0,
  z = 0,
  width = 100,
  height = 25,
  depth = 75,
  style = c("cumulus", "stratus"),
  seed = 42,
  resolution = 128,
  coverage = 0.5,
  detail = 0.35,
  optical_depth = 8,
  g = 0.65,
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  scale = c(1, 1, 1),
  t = 0,
  animation_seed = 1,
  haze = FALSE,
  haze_density_threshold = 0.05
)
```

Arguments

x	Default `0`. x-coordinate of the center of the cloud's bounding box.
y	Default `0`. y-coordinate of the center of the cloud's bounding box.

z	Default 0. z-coordinate of the center of the cloud's bounding box.
width	Default 100. Width of the cloud volume along its local x-axis.
height	Default 25. Height of the cloud volume along its local y-axis.
depth	Default 75. Depth of the cloud volume along its local z-axis.
style	Default c("cumulus", "stratus"). Cloud shape. "cumulus" creates rounded bodies with billowing tops; "stratus" creates a shallow cloud bank.
seed	Default 42. Nonnegative integer seed for the cloud shape. Generating a cloud preserves the caller's random-number state.
resolution	Default 128. Integer number of density cells along the longest dimension, at least 24. Other dimensions follow the aspect ratio, with at least eight cells each. Higher values add detail and use more memory.
coverage	Default 0.5. Number between zero and one controlling the size and connection of cloud bodies. Zero does not make the volume empty; use optical_depth = 0 for a non-scattering cloud.
detail	Default 0.35. Number between zero and one controlling the strength of small-scale Perlin detail.
optical_depth	Default 8. Nonnegative extinction through a fully dense column of length height. Actual optical depth depends on the density along the ray. Extinction is 99.9% scattering and 0.1% absorption.
g	Default 0.65. Henyey-Greenstein scattering asymmetry, strictly between -1 and 1. Positive values scatter forward along the light direction.
angle	Default c(0, 0, 0). Rotation in degrees around the x, y, and z axes, applied in the order specified by order_rotation.
order_rotation	Default c(1, 2, 3). Order of rotations, referring to x, y, and z. Must be a permutation of c(1, 2, 3).
scale	Default c(1, 1, 1). Nonzero scale factors along x, y, and z. A single value scales uniformly. Scales the density field and its boundary together, retaining extinction per world-space unit.
t	Default 0. Continuous, dimensionless evolution time. Small changes gently reshape the broad and fine density features without moving the bounding box. Zero reproduces the original static cloud. Keep both seeds fixed and increase this value between frames; for example, use seq(0, 1, length.out = 30) for a gentle transition. Negative times work.
animation_seed	Default 1. Nonnegative integer seed for the local evolution, independent of the shape's seed. Changing it selects a different evolution of the same cloud; it has no effect at t = 0.
haze	Default FALSE. Include clear-air atmospheric haze inside the cloud boundary when enabled by sky_light(). The default omits haze throughout the boundary, including empty cells, while retaining cloud scattering and altitude-dependent illumination. Set TRUE to enable haze subject to haze_density_threshold. Omitting haze is an approximation most useful for dense clouds at high altitude; thin clouds and wispy edges can show larger differences. See homogeneous_medium().

haze_density_threshold
: Default 0.05. With haze = TRUE, omit haze only where the interpolated cloud density is at least this positive value. When haze is enabled, the default retains it in empty space and regions below density 0.05. Cloud density ranges from zero to one. NULL enables haze throughout the boundary. Ignored with haze = FALSE. This is a density threshold, not an opacity threshold; optical_depth still controls the strength of the cloud's scattering. See homogeneous_medium().

Details

The local bounding box is centered at zero before object transforms, spanning -c(width, height, depth) / 2 to c(width, height, depth) / 2. The density fades to vacuum at all six faces; the box has no visible surface. An unrotated, unscaled cloud with base altitude b has y = b + height / 2. The center describes the box, not the irregular density's center of mass.

Positions and dimensions use scene units. Setting the dimensions rebuilds the field and normalizes extinction by height. Applying scale stretches the existing volume without renormalizing extinction, so stretching it along a ray increases that ray's optical depth. Standard group_objects(), animate_objects(), and create_instances() operations transform the cloud using the same object machinery as other closed shapes.

Evolution adds small, bounded perturbations to the broad and fine noise fields using separately seeded four-dimensional simplex noise (space and time). Subtracting the perturbation at time zero anchors the original shape. Broad domes, the base profile, and edge fades stay fixed while local density swells and erodes. Evolution is procedural rather than a fluid simulation; cloud mass is not conserved. Use x, y, and z for bulk movement.

Rebuild the cloud with a new t value for each rendered frame. animate_objects() animates its transform; it does not evolve the density within a frame or during the shutter interval.

Cloud scattering is separate from sky_light()'s clear-air atmospheric haze. Avoid intersecting, non-nested cloud boxes, including their empty edge cells: these are separate medium boundaries. Use one larger field for a connected bank. The attached grid_medium() is stored in object$shape_info[[1]]$medium for further density or scattering adjustments.

Value

A single-row ray_scene containing an invisible box with an attached cloud density grid.

See Also

sky_light(), grid_medium(), set_medium()

Examples

```
# The default cloud is centered at the origin. Raise a cloud by its center
# to put its base above the ground, then rotate the entire density field.
puff = cloud(y = 20, width = 60, height = 20, depth = 40,
             angle = c(0, 25, 0), resolution = 64, optical_depth = 4)
scene = generate_ground(material = diffuse("#699447")) |>
  add_object(puff) |>
  add_object(sphere(x = -50, y = 80, z = -30, radius = 15,
                    material = light(intensity = 40)))
```

```
  render_scene(scene, lookfrom = c(80, 35, -100), lookat = c(0, 20, 0),
               fov = 35, integrator_type = "nee", samples = 64,
               clamp_value = Inf, aperture = 0)

  # Reuse the shape at another position, or change the style and its detail.
  bank = cloud(x = 0, y = w0, z = 6, style = "stratus", seed = 17,
               width = 80, height = 10, depth = 50,
               coverage = 0.7, detail = 0.2, optical_depth = 6, g = 0.6)
  scaled = cloud(scale = c(1.5, 1, 0.75), angle = c(10, 30, 0),
                 order_rotation = c(2, 1, 3), resolution = 64)
generate_ground(material = diffuse("#699447")) |>
    add_object(bank) |>
    add_object(sphere(x = -50, y = 80, z = -30, radius = 15,
                      material = light(intensity = 40))) |>
 render_scene(lookfrom = c(80, 100, -100), lookat = c(0, 20, 0),
               fov = 35, integrator_type = "nee", samples = 64,
               clamp_value = Inf, aperture = 0)

generate_ground(material = diffuse("#699447")) |>
    add_object(scaled) |>
    add_object(sphere(x = -50, y = 80, z = -30, radius = 15,
                      material = light(intensity = 40))) |>
 render_scene(lookfrom = c(80, 100, -100), lookat = c(0, 20, 0),
               fov = 35, integrator_type = "nee", samples = 64,
               clamp_value = Inf, aperture = 0)
  # Keep seeds and position fixed to evolve the cloud locally over time.
  # Rebuilding a frame at the same t value always gives the same cloud.
  for (frame in 0:3) {
    set.seed(2026)
    generate_ground(material = diffuse("#699447")) |>
      add_object(cloud(y = 20, width = 60, height = 20, depth = 40,
      seed = 42, t = frame / 3, animation_seed = 17,
      resolution = 64, optical_depth = 4)) |>
      add_object(sphere(x = -50, y = 80, z = -30, radius = 15,
        material = light(intensity = 40))) |>
    render_scene(lookfrom = c(80, 35, -100), lookat = c(0, 20, 0),
      width = 256, height = 160, fov = 35, integrator_type = "nee",
      samples = 64, iso = 100, aperture = 0)
  }
```

---

cone *Cone Object*

Description

Cone Object

Usage

```
cone(
  start = c(0, 0, 0),
  end = c(0, 1, 0),
  radius = 0.5,
  direction = NA,
  from_center = TRUE,
  material = diffuse(),
  angle = c(0, 0, 0),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

start	Default 'c(0, 0, 0)'. Base of the cone, specifying 'x', 'y', 'z'.
end	Default 'c(0, 1, 0)'. Tip of the cone, specifying 'x', 'y', 'z'.
radius	Default '1'. Radius of the bottom of the cone.
direction	Default 'NA'. Alternative to 'start' and 'end', specify the direction (via a length-3 vector) of the cone. Cone will be centered at 'start', and the length will be determined by the magnitude of the direction vector.
from_center	Default 'TRUE'. If orientation specified via 'direction', setting this argument to 'FALSE' will make 'start' specify the bottom of the cone, instead of the middle.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default 'c(0, 0, 0)'. Rotation angle. Note: This will change the 'start' and 'end' coordinates.
flipped	Default 'FALSE'. Whether to flip the normals.
scale	Default 'c(1, 1, 1)'. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Notes: this will change the stated start/end position of the cone. Emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the cone in the scene.

Examples

```
#Generate a cone in a studio, pointing upwards:
generate_studio() |>
 add_object(cone(start=c(0,-1,0), end=c(0,1,0), radius=1,material=diffuse(color="red"))) |>
 add_object(sphere(y=5,x=5,material=light(intensity=40))) |>
 render_scene(samples=16,clamp_value=10)
 #Change the radius, length, and direction
generate_studio() |>
 add_object(cone(start=c(0,0,0), end=c(0,-1,0), radius=0.5,material=diffuse(color="red"))) |>
```

```
 add_object(sphere(y=5,x=5,material=light(intensity=40))) |>
 render_scene(samples=16,clamp_value=10)
#Give custom start and end points (and customize the color/texture)
generate_studio() |>
 add_object(cone(start=c(-1,0.5,-1), end=c(0,0,0), radius=0.5,material=diffuse(color="red"))) |>
 add_object(cone(start=c(1,0.5,-1), end=c(0,0,0), radius=0.5,material=diffuse(color="green"))) |>
 add_object(cone(start=c(0,1,-1), end=c(0,0,0), radius=0.5,material=diffuse(color="orange"))) |>
 add_object(cone(start=c(-1,-0.5,0), end=c(1,-0.5,0), radius=0.25,
   material = diffuse(color="red",gradient_color="green"))) |>
 add_object(sphere(y=5,x=5,material=light(intensity=40))) |>
 render_scene(samples=16,clamp_value=10)
#Specify cone via direction and location, instead of start and end positions
#Length is derived from the magnitude of the direction.
gold_mat = microfacet(roughness=0.1,eta=c(0.216,0.42833,1.3184), kappa=c(3.239,2.4599,1.8661))
generate_studio() |>
  add_object(cone(start = c(-1,0,0), direction = c(-0.5,0.5,0), material = gold_mat)) |>
  add_object(cone(start = c(1,0,0), direction = c(0.5,0.5,0), material = gold_mat)) |>
  add_object(cone(start = c(0,0,-1), direction = c(0,0.5,-0.5), material = gold_mat)) |>
  add_object(cone(start = c(0,0,1), direction = c(0,0.5,0.5), material = gold_mat)) |>
  add_object(sphere(y=5,material=light())) |>
  add_object(sphere(y=3,x=-3,z=-3,material=light(color="red"))) |>
  add_object(sphere(y=3,x=3,z=-3,material=light(color="green"))) |>
  render_scene(lookfrom=c(0,4,-10), clamp_value=10, samples=16)
 #Render the position from the base, instead of the center of the cone:
 noise_mat = material = glossy(color="purple",noisecolor="blue", noise=5)
 generate_studio() |>
  add_object(cone(start = c(0,-1,0), from_center = FALSE, radius=1, direction = c(0,2,0),
    material = noise_mat)) |>
  add_object(cone(start = c(-1.5,-1,0), from_center = FALSE, radius=0.5, direction = c(0,1,0),
    material = noise_mat)) |>
  add_object(cone(start = c(1.5,-1,0), from_center = FALSE, radius=0.5, direction = c(0,1,0),
    material = noise_mat)) |>
  add_object(cone(start = c(0,-1,1.5), from_center = FALSE, radius=0.5, direction = c(0,1,0),
    material = noise_mat)) |>
  add_object(sphere(y=5,x=5,material=light(intensity=40))) |>
  render_scene(lookfrom=c(0,4,-10), clamp_value=10,fov=25, samples=16)
```

`create_instances` *Create Instances of an Object*

Description

This creates multiple instances of the 'ray_scene' passed, each with it's own transformation applied (measured from the origin of the ray_scene). This means the scene only uses the memory of the object once and each copy only requires a 4x4 matrix in memory.

Usage

```
create_instances(
```

```
  ray_scene,
  x = 0,
  y = 0,
  z = 0,
  angle_x = 0,
  angle_y = 0,
  angle_z = 0,
  scale_x = 1,
  scale_y = 1,
  scale_z = 1,
  material = diffuse(),
  order_rotation = c(1, 2, 3)
)
```

Arguments

ray_scene	A ‘ray_scene‘ object to be copied at the specified transformed coordinates.
x	Default ‘0‘. A vector of x-coordinates to offset the instances. Note that this can also be a 3 column matrix or ‘data.frame()‘ parsable by ‘xyz.coords()‘: if so, the other axes will be ignored.
y	Default ‘0‘. A vector of y-coordinates to offset the instances.
z	Default ‘0‘. A vector of z-coordinates to offset the instances.
angle_x	Default ‘0‘. A vector of angles around the x axis to rotate the instances.
angle_y	Default ‘0‘. A vector of angles around the y axis to rotate the instances.
angle_z	Default ‘0‘. A vector of angles around the z axis to rotate the instances.
scale_x	Default ‘0‘. A vector of values around the scale the instances on the x-axis.
scale_y	Default ‘0‘. A vector of values around the scale the instances on the y-axis.
scale_z	Default ‘0‘. A vector of values around the scale the instances on the z-axis.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z" axes.

Value

Single row of a tibble describing the instance in the scene.

Examples

```
# Generate the base scene
base_scene = generate_ground(material = diffuse(checkercolor = "grey20")) |>
  add_object(sphere(z = -100, radius = 10, material = light(intensity = 70)))

# Start with a single sphere with an R in it
sphere_scene = sphere(y = 0, material = glossy(color = "#2b6eff", reflectance = 0.05)) |>
  add_object(obj_model(r_obj(simple_r = TRUE), z = -0.9, y = -0.2,
```

```
  scale_obj = 0.45, angle = c(0,180,0), material = diffuse())) |>
  group_objects(scale = 0.1)

# Render the scene
sphere_scene |>
  add_object(base_scene) |>
 render_scene(lookat = c(0, 1, 0), width = 800, sample_method = "sobol_blue", aperture = 0.2,
              height = 800, samples = 16, clamp_value = 20)

# Create instances at different x positions, with random rotations applied
create_instances(sphere_scene,
                 x = seq(-1.5, 1.5, length.out = 10),
                 angle_x = 90 * (runif(10) - 0.5),
                 angle_y = 90 * (runif(10) - 0.5),
                 angle_z = 90 * (runif(10) - 0.5)) |>
  add_object(base_scene) |>
  render_scene(lookat = c(0, 1, 0), width = 800, sample_method = "sobol_blue",
              height = 800, samples = 16, clamp_value = 20)

# Create instances at different x/z positions, with random scaling factors
create_instances(sphere_scene,
                 x = seq(-1.5, 1.5, length.out = 10),
                 y = seq(0, 1.5, length.out = 10),
                 scale_x = 0.5 + runif(10),
                 scale_y = 0.5 + runif(10),
                 scale_z = 0.5 + runif(10)) |>
  add_object(base_scene) |>
  render_scene(lookat = c(0, 1, 0), width = 800, sample_method = "sobol_blue",
              height = 800, samples = 16, clamp_value = 20)

# Create instances of instances
create_instances(sphere_scene,
                 x = seq(-1.5, 1.5, length.out = 10),
                 angle_y = 90 * (runif(10) - 0.5)) |>
  create_instances(y = seq(0, 2, length.out = 10)) |>
  add_object(base_scene) |>
  render_scene(lookat = c(0, 1, 0), width = 800, sample_method = "sobol_blue",
              height = 800, samples = 16, clamp_value = 20)

# Create instances of instances of instances of instances
create_instances(sphere_scene,
                 x = seq(-1.5, 1.5, length.out = 10),
                 angle_y = 90 * (runif(10) - 0.5)) |>
  create_instances(y = seq(0, 1, length.out = 5)) |>
  create_instances(y = seq(0, 2, length.out = 20) * 10,
                   angle_y = seq(0, 360, length.out = 20)) |>
  create_instances(x = c(-5, 0, 5),
                   scale_y = c(0.5, 1, 0.75)) |>
  add_object(base_scene) |>
  render_scene(lookat = c(0, 10, 0), lookfrom = c(0, 10, -50),
              width = 800, sample_method = "sobol_blue", fov = 30,
              height = 800, samples = 16, clamp_value = 20)
```

```
# Generate a complex scene in a Cornell box and replicate it in a 3x3 grid
# Here, a single `data.frame` with all three coordinates is passed to the `x` argument.
tempfileplot = tempfile()
png(filename = tempfileplot, height = 1600, width = 1600)
plot(iris$Petal.Length, iris$Sepal.Width, col = iris$Species, pch = 18, cex = 12)
dev.off()
image_array = png::readPNG(tempfileplot)

# Note that if a instanced scene has importance sampled lights and there are many instances,
# it will be slow to render.
generate_cornell(importance_sample=FALSE) |>
  add_object(ellipsoid(x = 555 / 2, y = 100, z = 555 / 2, a = 50, b = 100, c = 50,
             material = metal(color = "lightblue"))) |>
  add_object(cube(x = 100, y = 130 / 2, z = 200, xwidth = 130,
                  ywidth = 130, zwidth = 130, angle = c(0, 10, 0),
                  material = diffuse(checkercolor = "purple", checkerperiod = 30))) |>
  add_object(pig(x = 100, y = 190, z = 200, scale = 40, angle = c(0, 30, 0))) |>
  add_object(sphere(x = 420, y = 555 / 8, z = 100, radius = 555 / 8,
                    material = dielectric(color = "orange"))) |>
  add_object(yz_rect(x = 5, y = 300, z = 555 / 2, zwidth = 400, ywidth = 400,
                     material = diffuse(image_texture = image_array))) |>
  add_object(yz_rect(x = 555 / 2, y = 300, z = 555 - 5, zwidth = 400, ywidth = 400,
                material = diffuse(image_texture = image_array), angle = c(0, 90, 0))) |>
  add_object(yz_rect(x = 555 - 5, y = 300, z = 555 / 2, zwidth = 400, ywidth = 400,
                material = diffuse(image_texture = image_array), angle = c(0, 180, 0))) |>
  create_instances(x = expand.grid(x = seq(-1, 1, by = 1) * 570 - 555 / 2,
                                   y = seq(-1, 1, by = 1) * 570 - 555 / 2,
                                   z = 0)) |>
  render_scene(lookfrom = c(0, 0, -800) * 3, fov = 40,
               samples = 16, sample_method = "sobol_blue",
               parallel = TRUE, width = 800, height = 800)
```

---

csg_box *CSG Box*

Description

CSG Box

Usage

```
csg_box(x = 0, y = 0, z = 0, width = c(1, 1, 1), corner_radius = 0)
```

Arguments

x	Default '0'. An x-coordinate on the box.
y	Default '0'. A y-coordinate on the box.
z	Default '0'. A z-coordinate on the box

width	Default ‘c(1,1,1)‘. Length-3 vector describing the x/y/z widths of the box
corner_radius	Default ‘0‘. Radius if rounded box.

Value

List describing the box in the scene.

Examples

```
#Generate a box
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_box(), material=glossy(color="#FF69B4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=5))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(7,3,7))
#Change the width
generate_ground(material=diffuse(checkercolor="grey20")) |>
 add_object(csg_object(csg_box(width = c(2,1,0.5)), material=glossy(color="#FF69B4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=5))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(7,3,7))
#Subtract two boxes to make stairs
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
    csg_box(),
    csg_box(x=0.5,y=0.5,width=c(1,1,1.1)),operation="subtract"),
   material=glossy(color="#FF69B4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=5))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(7,3,7),fov=13)
```

---

csg_capsule *CSG Capsule*

Description

CSG Capsule

Usage

```
csg_capsule(start = c(0, 0, 0), end = c(0, 1, 0), radius = 1)
```

Arguments

start	Default ‘c(0, 0, 0)‘. Start point of the capsule, specifying ‘x‘, ‘y‘, ‘z‘.
end	Default ‘c(0, 1, 0)‘. End point of the capsule, specifying ‘x‘, ‘y‘, ‘z‘.
radius	Default ‘1‘. Capsule radius.

Value

List describing the capsule in the scene.

Examples

```
#Generate a basic capsule:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_capsule(radius=0.5),material=glossy(color="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20)
#Change the orientation by specifying a start and end
generate_ground(material=diffuse(color="dodgerblue4",checkercolor="grey10")) |>
  add_object(csg_object(csg_capsule(start = c(-1,0.5,-2), end = c(1,0.5,-2),
  radius=0.5),material=glossy(checkercolor="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20,
               lookat=c(0,0.5,-2),lookfrom=c(3,3,10))
#Show the effect of changing the radius
generate_ground(material=diffuse(color="dodgerblue4",checkercolor="grey10")) |>
  add_object(csg_object(
    csg_combine(
    csg_capsule(start = c(-1,0.5,-2), end = c(1,0.5,-2), radius=0.5),
    csg_capsule(start = c(-0.5,1.5,-2), end = c(0.5,1.5,-2), radius=0.25)),
    material=glossy(checkercolor="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20,
               lookat=c(0,0.5,-2),lookfrom=c(-3,3,10))
#Render a capsule in a Cornell box
generate_cornell() |>
  add_object(csg_object(
   csg_capsule(start = c(555/2-100,555/2,555/2), end = c(555/2+100,555/2,555/2), radius=100),
    material=glossy(color="dodgerblue4"))) |>
  render_scene(clamp_value=10, samples=16)
```

`csg_combine` *CSG Combine*

Description

Note: Subtract operations aren't commutative: the second object is subtracted from the first.

Usage

```
csg_combine(object1, object2, operation = "union", radius = 0.5)
```

Arguments

`object1`	First CSG object
`object2`	Second CSG object
`operation`	Default 'union'. Can be 'union', 'subtract', 'intersection', 'blend', 'subtract-blend', or 'mix'.
`radius`	Default '0.5'. Blending radius.

Value

List describing the combined csg object in the scene.

Examples

```
#Combine two spheres:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
     csg_sphere(x=-0.4,z=-0.4),
     csg_sphere(x=0.4,z=0.4), operation="union"),
  material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(-3,5,10))
#Subtract one sphere from another:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
     csg_sphere(x=-0.4,z=-0.4),
     csg_sphere(x=0.4,z=0.4), operation="subtract"),
  material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(-3,5,10))
#Get the intersection of two spheres:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
     csg_sphere(x=-0.4,z=-0.4),
     csg_sphere(x=0.4,z=0.4), operation="intersection"),
  material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(-3,5,10))
#Get the blended union of two spheres:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
     csg_sphere(x=-0.4,z=-0.4),
     csg_sphere(x=0.4,z=0.4), operation="blend"),
  material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(-3,5,10))
#Get the blended subtraction of two spheres:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
     csg_sphere(x=-0.4,z=-0.4),
     csg_sphere(x=0.4,z=0.4), operation="subtractblend"),
  material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(-3,5,10))
#Change the blending radius:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
     csg_sphere(x=-0.4,z=-0.4),
     csg_sphere(x=0.4,z=0.4), operation="blend", radius=0.2),
  material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
```

```
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(-3,5,10))
#Change the subtract blending radius:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
     csg_sphere(x=-0.4,z=-0.4),
     csg_sphere(x=0.4,z=0.4), operation="subtractblend", radius=0.2),
  material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(-3,5,10))
#Get the mixture of various objects:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
     csg_sphere(),
     csg_box(), operation="mix"),
  material=glossy(color="dodgerblue4"))) |>
  add_object(csg_object(csg_translate(csg_combine(
     csg_box(),
     csg_torus(), operation="mix"),z=-2.5),
  material=glossy(color="red"))) |>
  add_object(csg_object(csg_translate(csg_combine(
     csg_pyramid(),
     csg_box(), operation="mix"),z=2.5),
  material=glossy(color="green"))) |>
  add_object(sphere(y=10,x=-5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(-15,10,10))
```

---

csg_cone *CSG Cone*

Description

CSG Cone

Usage

```
csg_cone(start = c(0, 0, 0), end = c(0, 1, 0), radius = 0.5)
```

Arguments

start	Default ‘c(0, 0, 0)‘. Start point of the cone, specifing ‘x‘, ‘y‘, ‘z‘.
end	Default ‘c(0, 1, 0)‘. End point of the cone, specifing ‘x‘, ‘y‘, ‘z‘.
radius	Default ‘1‘. Radius of the bottom of the cone.

Value

List describing the box in the scene.

Examples

```
#Generate a basic cone:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_cone(),material=glossy(color="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20)
#Change the orientation by specifying a start and end
generate_ground(material=diffuse(color="dodgerblue4",checkercolor="grey10")) |>
  add_object(csg_object(csg_cone(start = c(-1,0.5,-2), end = c(1,0.5,-2),
  radius=0.5),material=glossy(checkercolor="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20,
               lookat=c(0,0.5,-2),lookfrom=c(3,3,10))
#Show the effect of changing the radius
generate_ground(material=diffuse(color="dodgerblue4",checkercolor="grey10")) |>
  add_object(csg_object(
    csg_combine(
    csg_cone(start = c(-1,0.5,-2), end = c(1,0.5,-2), radius=0.5),
    csg_cone(start = c(-0.5,1.5,-2), end = c(0.5,1.5,-2), radius=0.2)),
    material=glossy(checkercolor="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20,
               lookat=c(0,0.5,-2),lookfrom=c(-3,3,10))
#Render a glass cone in a Cornell box
generate_cornell() |>
  add_object(csg_object(
    csg_cone(start = c(555/2,0,555/2), end = c(555/2,555/2+100,555/2), radius=100),
    material=dielectric(attenuation=c(1,1,0.3)/100))) |>
  render_scene(clamp_value=10, samples=16)
```

`csg_cylinder` *CSG Cylinder*

Description

CSG Cylinder

Usage

```
csg_cylinder(
  start = c(0, 0, 0),
  end = c(0, 1, 0),
  radius = 1,
  corner_radius = 0
)
```

Arguments

`start`	Default ‘c(0, 0, 0)‘. Start point of the cylinder, specifing ‘x‘, ‘y‘, ‘z‘.
`end`	Default ‘c(0, 1, 0)‘. End point of the cylinder, specifing ‘x‘, ‘y‘, ‘z‘.
`radius`	Default ‘1‘. Cylinder radius.
`corner_radius`	Default ‘0‘. Radius if rounded cylinder.

Value

List describing the cylinder in the scene.

Examples

```
#Generate a basic cylinder:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_cylinder(radius=0.25),material=glossy(color="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20)
#Change the orientation by specifying a start and end
generate_ground(material=diffuse(color="dodgerblue4",checkercolor="grey10")) |>
  add_object(csg_object(csg_cylinder(start = c(-1,0.5,-2), end = c(1,0.5,-2),
    radius=0.5),material=glossy(checkercolor="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20,
               lookat=c(0,0.5,-2),lookfrom=c(3,3,10))
#Show the effect of changing the radius
generate_ground(material=diffuse(color="dodgerblue4",checkercolor="grey10")) |>
  add_object(csg_object(
    csg_combine(
    csg_cylinder(start = c(-1,0.5,-2), end = c(1,0.5,-2), radius=0.5),
    csg_cylinder(start = c(-0.5,1.5,-2), end = c(0.5,1.5,-2), radius=0.25)),
    material=glossy(checkercolor="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20,
               lookat=c(0,0.5,-2),lookfrom=c(-3,3,10))
#Render a red marble cylinder in a Cornell box
generate_cornell(light=FALSE) |>
  add_object(csg_object(
    csg_cylinder(start = c(555/2,0,555/2), end = c(555/2,350,555/2), radius=100),
    material=glossy(color="darkred",noisecolor="white",noise=0.03))) |>
    add_object(sphere(y=555,x=5,z=5, radius=5,
               material=light(intensity=10000,
                        spotlight_focus = c(555/2,555/2,555/2),spotlight_width = 45))) |>
  render_scene(clamp_value=4)
```

---

csg_ellipsoid *CSG Ellipsoid*

Description

CSG Ellipsoid

Usage

```
csg_ellipsoid(x = 0, y = 0, z = 0, axes = c(0.5, 1, 0.5))
```

Arguments

x	Default '0'. x-coordinate on the ellipsoid.
y	Default '0'. y-coordinate on the ellipsoid.
z	Default '0'. z-coordinate on the ellipsoid.
axes	Default 'c(0.5,1,0.5)'. Ellipsoid principle axes.

Value

List describing the ellipsoid in the scene.

Examples

```
#Generate a basic ellipsoid:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_ellipsoid(),material=glossy(color="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20)
#Three different ellipsoids:
generate_ground(material=diffuse(checkercolor="grey20")) |>
 add_object(csg_object(csg_group(list(
   csg_ellipsoid(x=-1.2, axes = c(0.2,0.5,0.5)),
   csg_ellipsoid(x=0, axes = c(0.5,0.2,0.5)),
   csg_ellipsoid(x=1.2, axes = c(0.5,0.5,0.2)))),
   material=glossy(color="red"))) |>
 render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(0,5,10))
#Generate a glass ellipsoid:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_ellipsoid(),material=dielectric(attenuation = c(1,1,0.3)))) |>
  render_scene(clamp_value=10, samples=16,fov=20)
#Generate a glass ellipsoid in a Cornell box:
generate_cornell() |>
  add_object(csg_object(csg_ellipsoid(x=555/2,y=555/2,z=555/2,axes=c(100,150,200)),
    material=dielectric(attenuation = c(1,0.3,1)/200))) |>
  render_scene(clamp_value=10, samples=16)
```

---

csg_elongate *CSG Elongate*

Description

This operation elongates an existing CSG object in a direction.

Usage

```
csg_elongate(object, x = 0, y = 0, z = 0, elongate = c(0, 0, 0), robust = TRUE)
```

Arguments

object	CSG object.
x	Default '0'. Center of x-elongation.
y	Default '0'. Center of y-elongation.
z	Default '0'. Center of z-elongation.
elongate	Default 'c(0,0,0)' (no elongation). Elongation amount.
robust	Default 'TRUE'. 'FALSE' switches to a faster (but less robust in 2D) method.

Value

List describing the triangle in the scene.

Examples

```
#Elongate a sphere to create a capsule in 1D or a rounded rectangle in 2D:
generate_ground(material=diffuse(checkercolor="grey20",color="dodgerblue4")) |>
 add_object(csg_object(csg_sphere(z=-3,x=-3),
                       material=glossy(color="purple"))) |>
 add_object(csg_object(csg_elongate(csg_sphere(z=-3,x=3),x=3,z=-3, elongate = c(0.8,0,0)),
                       material=glossy(color="red"))) |>
 add_object(csg_object(csg_elongate(csg_sphere(z=2),z=2, elongate = c(0.8,0,0.8)),
                       material=glossy(color="white"))) |>
 add_object(sphere(y=10,radius=3,material=light(intensity=8))) |>
 render_scene(clamp_value=10, samples=16,fov=40,lookfrom=c(0,10,10))
#Elongate a torus:
generate_ground(material=diffuse(checkercolor="grey20",color="dodgerblue4")) |>
 add_object(csg_object(csg_torus(z=-3,x=-3),
                       material=glossy(color="purple"))) |>
 add_object(csg_object(csg_elongate(csg_torus(z=-3,x=3),x=3,z=-3, elongate = c(0.8,0,0)),
                       material=glossy(color="red"))) |>
 add_object(csg_object(csg_elongate(csg_torus(z=2),z=2, elongate = c(0.8,0,0.8)),
                       material=glossy(color="white"))) |>
 add_object(sphere(y=10,radius=3,material=light(intensity=8))) |>
 render_scene(clamp_value=10, samples=16,fov=40,lookfrom=c(0,10,10))
#Elongate a cylinder:
generate_ground(material=diffuse(checkercolor="grey20",color="dodgerblue4")) |>
 add_object(csg_object(csg_cylinder(start=c(-3,0,-3), end = c(-3,1,-3)),
                       material=glossy(color="purple"))) |>
 add_object(csg_object(csg_elongate(csg_cylinder(start=c(3,0,-3), end = c(3,1,-3)), x=3, z=-3,
                      elongate = c(0.8,0,0)),
                      material=glossy(color="red"))) |>
 add_object(csg_object(csg_elongate(csg_cylinder(start=c(0,0,3), end = c(0,1,3)), z=3,
                      elongate = c(0.8,0,0.8)),
                      material=glossy(color="white"))) |>
 add_object(sphere(y=10,radius=3,material=light(intensity=8))) |>
 render_scene(clamp_value=10, samples=16,fov=40,lookfrom=c(0,10,10))
#Elongate a pyramid:
generate_ground(material=diffuse(checkercolor="grey20",color="dodgerblue4")) |>
 add_object(csg_object(csg_pyramid(z=-3,x=-3),
                       material=glossy(color="purple"))) |>
```

```
 add_object(csg_object(csg_elongate(csg_pyramid(z=-3,x=3),x=3,z=-3, elongate = c(0.8,0,0)),
                       material=glossy(color="red"))) |>
 add_object(csg_object(csg_elongate(csg_pyramid(z=2),z=2, elongate = c(0.8,0,0.8)),
                       material=glossy(color="white"))) |>
 add_object(sphere(y=10,radius=3,material=light(intensity=8))) |>
 render_scene(clamp_value=10, samples=16,fov=40,lookfrom=c(0,10,10))
#Change the elongation point to start the elongation on the side of the pyramid:
generate_ground(material=diffuse(checkercolor="grey20",color="dodgerblue4")) |>
 add_object(csg_object(csg_pyramid(z=-3,x=-3),
                       material=glossy(color="purple"))) |>
 add_object(csg_object(csg_elongate(csg_pyramid(z=-3,x=3),x=2.75,z=-2.75, elongate = c(0.8,0,0)),
                       material=glossy(color="red"))) |>
 add_object(csg_object(csg_elongate(csg_pyramid(z=2),z=2.25, elongate = c(0.8,0,0.8)),
                       material=glossy(color="white"))) |>
 add_object(sphere(y=10,radius=3,material=light(intensity=8))) |>
 render_scene(clamp_value=10, samples=16,fov=40,
              lookfrom=c(5,5,10),lookat=c(0,0,-1.5))
```

---

csg_group *CSG Group*

Description

CSG Group

Usage

```
csg_group(object_list)
```

Arguments

object_list	List of objects created with the csg_* functions. This will make all further operations be applied to this object as a group.

Value

List describing the group in the scene.

Examples

```
#Group four spheres together and merge them with a box:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
  csg_group(list(csg_sphere(x=1,z=1, radius=0.5),csg_sphere(x=-1,z=1, radius=0.5),
                 csg_sphere(x=1,z=-1, radius=0.5),csg_sphere(x=-1,z=-1, radius=0.5))),
  csg_box(y=0.5, width=c(2,0.2,2)), operation="blend"), material=glossy(color="red"))) |>
  add_object(sphere(y=10,x=-5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(5,5,10))
```

csg_object *Constructive Solid Geometry Object*

Description

This object takes an object constructed using the ‘csg_*‘ functions. The object is drawn using ray marching/sphere tracing.

Usage

```
csg_object(
  object,
  x = 0,
  y = 0,
  z = 0,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

object	Object created with CSG interface.
x	Default ‘0‘. x-offset of the center of the object.
y	Default ‘0‘. y-offset of the center of the object.
z	Default ‘0‘. z-offset of the center of the object.
material	Default diffuse. The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Details

Note: For dielectric objects, any other objects not included in the CSG object and nested inside will be ignored.

Value

Single row of a tibble describing the sphere in the scene.

Examples

```
#We will combine these three objects:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_box(), material=glossy(color="red"))) |>
  add_object(csg_object(csg_sphere(radius=0.707), material=glossy(color="green"))) |>
 add_object(csg_object(csg_group(list(csg_cylinder(start=c(-1,0,0), end=c(1,0,0), radius=0.4),
                  csg_cylinder(start=c(0,-1,0), end=c(0,1,0), radius=0.4),
                  csg_cylinder(start=c(0,0,-1), end=c(0,0,1), radius=0.4))),
                  material=glossy(color="blue"))) |>
  add_object(sphere(y=5,x=3,radius=1,material=light(intensity=30))) |>
  render_scene(clamp_value=10, fov=15,lookfrom=c(5,5,10),
               samples=16, sample_method="sobol_blue")
#Standard CSG sphere + box - crossed cylinder combination:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
    csg_combine(
      csg_box(),
      csg_sphere(radius=0.707),
      operation="intersection"),
    csg_group(list(csg_cylinder(start=c(-1,0,0), end=c(1,0,0), radius=0.4),
                  csg_cylinder(start=c(0,-1,0), end=c(0,1,0), radius=0.4),
                  csg_cylinder(start=c(0,0,-1), end=c(0,0,1), radius=0.4))),
    operation="subtract"),
    material=glossy(color="red"))) |>
  add_object(sphere(y=5,x=3,radius=1,material=light(intensity=30))) |>
  render_scene(clamp_value=10, fov=10,lookfrom=c(5,5,10),
               samples=16, sample_method="sobol_blue")
#Blend them all instead:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
    csg_combine(
      csg_box(),
      csg_sphere(radius=0.707),
      operation="blend"),
    csg_group(list(csg_cylinder(start=c(-1,0,0), end=c(1,0,0), radius=0.4),
                  csg_cylinder(start=c(0,-1,0), end=c(0,1,0), radius=0.4),
                  csg_cylinder(start=c(0,0,-1), end=c(0,0,1), radius=0.4))),
    operation="blend"),
    material=glossy(color="purple"))) |>
  add_object(sphere(y=5,x=3,radius=1,material=light(intensity=30))) |>
  render_scene(clamp_value=10, fov=15,lookfrom=c(5,5,10),
               samples=16, sample_method="sobol_blue")
```

`csg_onion` *CSG Onion*

Description

Note: This operation has no overt effect on the external appearance of an object–it carves regions on the interior. Thus, you will only see an effect with a transparent material or when you carve into the object.

Usage

```
csg_onion(object, thickness = 0.1)
```

Arguments

object	CSG object.
thickness	Default '0.1'. Onioning distance.

Value

List describing the triangle in the scene.

Examples

```
#Cut and onion a sphere:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
    csg_onion(csg_sphere(z=2,x=2,radius=1), thickness = 0.2),
    csg_box(y=1,width=c(10,2,10)), operation = "subtract"),
    material=glossy(color="red"))) |>
    add_object(csg_object(csg_combine(
      csg_onion(csg_sphere(radius=1), thickness = 0.4),
      csg_box(y=1,width=c(10,2,10)), operation = "subtract"),
      material=glossy(color="purple"))) |>
    add_object(csg_object(csg_combine(
      csg_onion(csg_sphere(z=-2.5,x=-2.5,radius=1), thickness = 0.6),
      csg_box(y=1,width=c(10,2,10)), operation = "subtract"),
      material=glossy(color="green"))) |>
 add_object(sphere(y=5,x=5,radius=2,material=light())) |>
 render_scene(clamp_value=10, samples=16,lookat=c(0,-0.5,0),
              lookfrom=c(3,5,10),fov=35)
#Multiple onion layers:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
    csg_onion(csg_onion(csg_onion(csg_sphere(radius=1), 0.4), 0.2),0.1),
    csg_box(y=1,width=c(10,2,10)), operation = "subtract"),
    material=glossy(color="purple"))) |>
  add_object(sphere(y=5,x=5,radius=2,material=light())) |>
  render_scene(clamp_value=10, samples=16,lookat=c(0,-0.5,0),
               lookfrom=c(3,5,10),fov=20)
#Onion with dielectric sphere to make a bubble:
generate_cornell() |>
  add_object(csg_object(
    csg_onion(csg_sphere(x=555/2,y=555/2,z=555/2, radius=150), 5),
    material=dielectric(attenuation=c(1,1,0.3)/100))) |>
```

```
  render_scene(clamp_value=10, samples=16)
#Multiple onion operations to make a bubble within a bubble:
generate_cornell() |>
  add_object(csg_object(
    csg_onion(csg_onion(csg_sphere(x=555/2,y=555/2,z=555/2, radius=150), 10),5),
    material=dielectric(attenuation=c(1,1,0.3)/100))) |>
  render_scene(clamp_value=10, samples=16)
```

`csg_plane` *CSG Plane*

Description

Note: This shape isn't closed, so there may be odd lighting issues if it's oriented the wrong way.

Usage

```
csg_plane(x = 0, y = 0, z = 0, normal = c(0, 1, 0), width_x = 4, width_z = 4)
```

Arguments

`x`	Default '0'. An x-coordinate on the plane.
`y`	Default '0'. A y-coordinate on the plane.
`z`	Default '0'. A z-coordinate on the plane.
`normal`	Default 'c(0,1,0)'. Surface normal of the plane.
`width_x`	Default '10'.
`width_z`	Default '10'.

Value

List describing the plane in the scene.

Examples

```
#Generate a plane
csg_object(csg_plane(width_x=4, width_z=4), material=diffuse(checkercolor="purple")) |>
  add_object(sphere(y=5,x=5,material=light(intensity=40))) |>
  render_scene(clamp_value=10, samples=16)
#Combine the plane with a sphere
csg_object(csg_combine(
    csg_sphere(radius=0.5),
    csg_plane(width_x=4, width_z=4,y=-0.5),
    operation="blend"),material=diffuse(checkercolor="purple")) |>
  add_object(sphere(y=5,x=5,material=light(intensity=40))) |>
  render_scene(clamp_value=10, samples=16)
#Re-orient the plane using the normal and
csg_object(csg_combine(
```

```
  csg_sphere(radius=0.5),
  csg_plane(normal = c(1,1,0),width_x=4, width_z=4,y=-0.5),
  operation="blend"),material=diffuse(checkercolor="purple")) |>
add_object(sphere(y=5,x=5,material=light(intensity=40))) |>
render_scene(clamp_value=10, samples=16)
```

`csg_pyramid` *CSG Pyramid*

Description

Note: This primitive slows down immensely for large values of base and height. Try using csg_scale() with this object for large pyramids instead.

Usage

```
csg_pyramid(x = 0, y = 0, z = 0, height = 1, base = 1)
```

Arguments

`x`	Default '0'. x-coordinate on the pyramid.
`y`	Default '0'. y-coordinate on the pyramid.
`z`	Default '0'. z-coordinate on the pyramid.
`height`	Default '1'. Pyramid height.
`base`	Default '1'. Pyramid base width.

Value

List describing the box in the scene.

Examples

```
#Generate a simple pyramid:
generate_ground() |>
  add_object(csg_object(csg_pyramid(y=-0.99),
                        material=glossy(color="red"))) |>
  add_object(sphere(y=5,x=5,z=5,material=light(intensity=20))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(-3,1,10),
               fov=15, lookat=c(0,-0.5,0))
#Make a taller pyramid
generate_ground() |>
  add_object(csg_object(csg_pyramid(y=-0.95, height=1.5),
                        material=glossy(color="red"))) |>
  add_object(sphere(y=5,x=5,z=5,material=light(intensity=20))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(-3,1,10),
               fov=15, lookat=c(0,-0.5,0))
#Make a wider pyramid
```

```
generate_ground() |>
  add_object(csg_object(csg_pyramid(y=-0.95, base=1.5),
                        material=glossy(color="red"))) |>
  add_object(sphere(y=5,x=5,z=5,material=light(intensity=20))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(-3,1,10),
               fov=15, lookat=c(0,-0.5,0))
```

`csg_rotate` *CSG Rotate*

Description

CSG Rotate

Usage

```
csg_rotate(
  object,
  pivot_point = c(0, 0, 0),
  angles = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  up = c(0, 1, 0),
  axis_x = NULL,
  axis_z = NULL
)
```

Arguments

`object`	CSG object.
`pivot_point`	Default ‘c(0,0,0)‘. Pivot point for the rotation.
`angles`	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
`order_rotation`	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
`up`	Default ‘c(0,1,0). Alternative method for specifying rotation–change the new "up" vector.
`axis_x`	Default ‘NULL‘, computed automatically if not passed. Given the ‘up‘ vector as the y-axis, this is the x vector.
`axis_z`	Default ‘NULL‘, computed automatically if not passed. Given the ‘up‘ vector as the y-axis, this is the z vector.

Value

List describing the triangle in the scene.

Examples

```
#Rotate a pyramid (translating it upwards because the object is scaled from the center):
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_pyramid(z=1,y=-0.99),
                        material=glossy(color="red"))) |>
  add_object(csg_object(csg_rotate(csg_pyramid(z=-1.5,y=-0.99),
                        pivot_point = c(0,-0.99,-1.5),angle=c(0,45,0)),
                        material=glossy(color="green"))) |>
  add_object(sphere(y=5,x=5,z=5,material=light(intensity=40))) |>
  render_scene(lookfrom=c(-3,4,10), fov=15,
               lookat=c(0,-0.5,0),clamp_value=10)
#Rotate by specifying a new up vector:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_pyramid(z=1,y=-0.99),
                        material=glossy(color="red"))) |>
  add_object(csg_object(csg_rotate(csg_pyramid(z=-1.5,y=-0.49),
                        pivot_point = c(0,-0.49,-1.5), up =c(1,1,0)),
                        material=glossy(color="green"))) |>
  add_object(sphere(y=5,x=5,z=5,material=light(intensity=40))) |>
  render_scene(lookfrom=c(-3,4,10), fov=15,
               lookat=c(0,-0.5,0),clamp_value=10)
```

`csg_round` *CSG Round*

Description

CSG Round

Usage

```
csg_round(object, radius = 0.1)
```

Arguments

`object`	CSG object.
`radius`	Default '0.1'. Rounding distance.

Value

List describing the triangle in the scene.

Examples

```
#Generate a rounded pyramid:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_pyramid(x=-1,y=-0.99,z=1),
                        material=glossy(color="red"))) |>
  add_object(csg_object(csg_round(csg_pyramid(x=1,y=-0.89)),
                        material=glossy(color="blue"))) |>
  add_object(csg_object(csg_round(csg_pyramid(x=0,z=-2,y=-0.5), radius=0.5),
                        material=glossy(color="green"))) |>
  add_object(sphere(y=5,x=5,z=5,radius=1,material=light(intensity=50))) |>
  render_scene(lookfrom=c(-3,4,10), fov=22,
               lookat=c(0,-0.5,0),clamp_value=10)
#Round a blend of two objects
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_round(csg_combine(
    csg_pyramid(x=-0.5,y=-0.99,z=1.5),
    csg_pyramid(x=0.5,y=-0.99,z=2), operation="blend"), radius=0),
                        material=glossy(color="red"))) |>
  add_object(csg_object(csg_round(csg_combine(
    csg_pyramid(x=-0.5,y=-0.79,z=-1.5),
    csg_pyramid(x=0.5,y=-0.79,z=-1), operation="blend"), radius=0.2),
                        material=glossy(color="green"))) |>
  add_object(sphere(y=5,x=5,z=5,radius=1,material=light(intensity=50))) |>
  render_scene(lookfrom=c(-3,5,10), fov=22,
               lookat=c(0,-0.5,0),clamp_value=10)
```

---

csg_rounded_cone *CSG Rounded Cone*

Description

CSG Rounded Cone

Usage

```
csg_rounded_cone(
  start = c(0, 0, 0),
  end = c(0, 1, 0),
  radius = 0.5,
  upper_radius = 0.2
)
```

Arguments

start	Default ‘c(0, 0, 0)‘. Start point of the cone, specifing ‘x‘, ‘y‘, ‘z‘.
end	Default ‘c(0, 1, 0)‘. End point of the cone, specifing ‘x‘, ‘y‘, ‘z‘.
radius	Default ‘0.5‘. Radius of the bottom of the cone.
upper_radius	Default ‘0.2‘. Radius from the top of the cone.

Value

List describing the box in the scene.

Examples

```
#Generate a basic rounded cone:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_rounded_cone(),material=glossy(color="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20)
#Change the orientation by specifying a start and end
generate_ground(material=diffuse(color="dodgerblue4",checkercolor="grey10")) |>
  add_object(csg_object(csg_rounded_cone(start = c(-1,0.5,-2), end = c(1,0.5,-2),
  radius=0.5),material=glossy(checkercolor="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20,
               lookat=c(0,0.5,-2),lookfrom=c(3,3,10))
#Show the effect of changing the radius
generate_ground(material=diffuse(color="dodgerblue4",checkercolor="grey10")) |>
  add_object(csg_object(
    csg_combine(
    csg_rounded_cone(start = c(-1,0.5,-2), end = c(1,0.5,-2), radius=0.5),
   csg_rounded_cone(start = c(-0.5,1.5,-2), end = c(0.5,1.5,-2), radius=0.2,upper_radius = 0.5)),
    material=glossy(checkercolor="red"))) |>
  render_scene(clamp_value=10, samples=16,fov=20,
               lookat=c(0,0.5,-2),lookfrom=c(-3,3,10))
#Render a glass rounded cone in a Cornell box
generate_cornell() |>
  add_object(csg_object(
   csg_rounded_cone(start = c(555/2,555/2-100,555/2), end = c(555/2,555/2+100,555/2), radius=100),
    material=dielectric(attenuation=c(1,1,0.3)/100))) |>
  render_scene(clamp_value=10, samples=16)
```

---

csg_scale *CSG Scale*

Description

CSG Scale

Usage

```
csg_scale(object, scale = 1)
```

Arguments

object	CSG object.
scale	Default '1'.

Value

List describing the triangle in the scene.

Examples

```
#Scale a pyramid (translating it upwards because the object is scaled from the center):
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_pyramid(z=1,y=-0.99),
                        material=glossy(color="red"))) |>
  add_object(csg_object(csg_scale(csg_pyramid(z=-1,y=-0.5),2),
                        material=glossy(color="green"))) |>
  add_object(sphere(y=5,x=5,z=5,material=light(intensity=40))) |>
  render_scene(lookfrom=c(-3,4,10), fov=20,
               lookat=c(0,-0.5,-0.5),clamp_value=10)
```

---

csg_sphere *CSG Sphere*

Description

CSG Sphere

Usage

```
csg_sphere(x = 0, y = 0, z = 0, radius = 1)
```

Arguments

x	Default '0'. x-coordinate of the center of the sphere.
y	Default '0'. y-coordinate of the center of the sphere.
z	Default '0'. z-coordinate of the center of the sphere.
radius	Default '1'. Radius of the sphere.

Value

List describing the sphere in the scene.

Examples

```
#Generate a simple sphere:
generate_ground() |>
  add_object(csg_object(csg_sphere(),
                        material=glossy(color="purple"))) |>
  render_scene(clamp_value=10, samples=16)
#Generate a bigger sphere in the cornell box.
generate_cornell() |>
  add_object(csg_object(csg_sphere(x=555/2,y=555/2,z=555/2,radius=100),
```

```
                        material=glossy(checkercolor="purple", checkerperiod=100))) |>
  render_scene(clamp_value=10, samples=16)
#Combine two spheres of different sizes
generate_cornell() |>
  add_object(csg_object(
    csg_combine(
      csg_sphere(x=555/2,y=555/2-50,z=555/2,radius=100),
      csg_sphere(x=555/2,y=555/2+50,z=555/2,radius=80)),
    material=glossy(color="purple"))) |>
  render_scene(clamp_value=10, samples=16)
#Subtract two spheres to create an indented region
generate_cornell() |>
  add_object(csg_object(
    csg_combine(
      csg_sphere(x=555/2,y=555/2-50,z=555/2,radius=100),
      csg_sphere(x=555/2+30,y=555/2+20,z=555/2-90,radius=40),
      operation="subtract"),
    material=glossy(color="grey20"))) |>
  render_scene(clamp_value=10, samples=16)
#Use csg_combine(operation="blend") to melt the two together
generate_cornell() |>
  add_object(csg_object(
    csg_combine(
      csg_sphere(x=555/2,y=555/2-50,z=555/2,radius=100),
      csg_sphere(x=555/2,y=555/2+50,z=555/2,radius=80),
      operation="blend", radius=20),
    material=glossy(color="purple"))) |>
  render_scene(clamp_value=10, samples=16)
```

`csg_torus` *CSG Torus*

Description

CSG Torus

Usage

```
csg_torus(x = 0, y = 0, z = 0, radius = 1, minor_radius = 0.5)
```

Arguments

`x`	Default '0'. x-coordinate on the torus.
`y`	Default '0'. y-coordinate on the torus.
`z`	Default '0'. z-coordinate on the torus.
`radius`	Default '1'. Torus radius.
`minor_radius`	Default '0.5'. Cross section radius of the torus.

Value

List describing the torus in the scene.

Examples

```
#Generate a torus:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_torus(), material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(0,5,10),fov=30)
#Change the radius of the torus:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_torus(radius=2), material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(0,5,10),fov=30)
#Change the minor radius of the torus:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_torus(radius=2, minor_radius=0.25),
                        material=glossy(color="dodgerblue4"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(0,5,10),fov=30)
#Generate a rotated torus in the Cornell Box
generate_cornell() |>
  add_object(csg_object(csg_rotate(
    csg_torus(x=555/2,y=555/2,z=555/2,radius=100, minor_radius=50),
    pivot_point = c(555/2,555/2,555/2), up =c(0,1,-1)),
                        material=glossy(color="dodgerblue4"))) |>
  render_scene(clamp_value=10, samples=16)
```

`csg_translate` *CSG Translate*

Description

CSG Translate

Usage

```
csg_translate(object, x = 0, y = 0, z = 0)
```

Arguments

`object`	CSG object.
`x`	Default '0'. x translation.
`y`	Default '0'. y translation.
`z`	Default '0'. z translation.

Value

List describing the triangle in the scene.

Examples

```
#Translate a simple object:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_torus(), material=glossy(color="dodgerblue4"))) |>
  add_object(csg_object(csg_translate(csg_torus(),x=-2,y=1,z=-2),
                        material=glossy(color="red"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(0,5,10),fov=30,
               lookat=c(-1,0.5,-1))
#Translate a blended object:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_combine(
    csg_torus(),
   csg_torus(y=1, radius=0.8), operation="blend"), material=glossy(color="dodgerblue4"))) |>
  add_object(csg_object(csg_translate(
    csg_combine(
      csg_torus(),
      csg_torus(y=1, radius=0.8), operation="blend"),
    x=-3,y=1,z=-3),
    material=glossy(color="red"))) |>
  add_object(sphere(y=5,x=5,radius=3,material=light(intensity=10))) |>
  render_scene(clamp_value=10, samples=16,lookfrom=c(0,5,10),fov=30,
               lookat=c(-1.5,0.5,-1.5))
```

---

csg_triangle *CSG Triangle*

Description

CSG Triangle

Usage

```
csg_triangle(v1 = c(0, 1, 0), v2 = c(1, 0, 0), v3 = c(-1, 0, 0))
```

Arguments

v1	Default ‘c(0,1,0)‘. First vertex.
v2	Default ‘c(1,0,0)‘. Second vertex.
v3	Default ‘c(-1,0,0)‘. Third vertex.

Value

List describing the triangle in the scene.

Examples

```
#Generate a basic triangle:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_triangle(),material=diffuse(color="red"))) |>
  add_object(sphere(y=5,z=-3,material=light(intensity=30))) |>
  render_scene(clamp_value=10, samples=16,fov=20)
#Change a vertex:
generate_ground(material=diffuse(checkercolor="grey20")) |>
  add_object(csg_object(csg_triangle(v1 = c(1,1,0)),material=diffuse(color="green"))) |>
  add_object(sphere(y=5,z=-3,material=light(intensity=30))) |>
  render_scene(clamp_value=10, samples=16,fov=20)
#Change all three vertices:
generate_ground(material=diffuse(checkercolor="grey20")) |>
 add_object(csg_object(csg_triangle(v1 = c(0.5,1,0), v2 = c(1,-0.5,0), v3 = c(-1,0.5,0)),
                       material=diffuse(color="blue"))) |>
  add_object(sphere(y=5,z=3,material=light(intensity=30))) |>
  render_scene(clamp_value=10, samples=16,fov=20,lookfrom=c(0,5,10))
```

---

cube *Cube Object*

Description

Cube Object

Usage

```
cube(
  x = 0,
  y = 0,
  z = 0,
  width = 1,
  xwidth = 1,
  ywidth = 1,
  zwidth = 1,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

x	Default '0'. x-coordinate of the center of the cube
y	Default '0'. y-coordinate of the center of the cube

z	Default ‘0‘. z-coordinate of the center of the cube
width	Default ‘1‘. Cube width.
xwidth	Default ‘1‘. x-width of the cube. Overrides ‘width‘ argument for x-axis.
ywidth	Default ‘1‘. y-width of the cube. Overrides ‘width‘ argument for y-axis.
zwidth	Default ‘1‘. z-width of the cube. Overrides ‘width‘ argument for z-axis.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the cube in the scene.

Examples

```
#Generate a cube in the cornell box.
generate_cornell() |>
  add_object(cube(x = 555/2, y = 100, z = 555/2,
                  xwidth = 200, ywidth = 200, zwidth = 200, angle = c(0, 30, 0))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
#Generate a gold cube in the cornell box
generate_cornell() |>
  add_object(cube(x = 555/2, y = 100, z = 555/2,
                  xwidth = 200, ywidth = 200, zwidth = 200, angle = c(0, 30, 0),
                  material = metal(color = "gold", fuzz = 0.2))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

#Generate a rotated dielectric box in the cornell box
generate_cornell() |>
  add_object(cube(x = 555/2, y = 200, z = 555/2,
                  xwidth = 200, ywidth = 100, zwidth = 200, angle = c(-30, 30, -30),
                  material = dielectric())) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

`cylinder` *Cylinder Object*

Description

Cylinder Object

Usage

```
cylinder(
  x = 0,
  y = 0,
  z = 0,
  radius = 1,
  length = 1,
  phi_min = 0,
  phi_max = 360,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1),
  capped = TRUE
)
```

Arguments

`x`	Default ‘0‘. x-coordinate of the center of the cylinder
`y`	Default ‘0‘. y-coordinate of the center of the cylinder
`z`	Default ‘0‘. z-coordinate of the center of the cylinder
`radius`	Default ‘1‘. Radius of the cylinder.
`length`	Default ‘1‘. Length of the cylinder.
`phi_min`	Default ‘0‘. Minimum angle around the segment.
`phi_max`	Default ‘360‘. Maximum angle around the segment.
`material`	Default `diffuse`.The material, called from one of the material functions `diffuse`, `metal`, or `dielectric`.
`angle`	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
`order_rotation`	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
`flipped`	Default ‘FALSE‘. Whether to flip the normals.
`scale`	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly.
`capped`	Default ‘TRUE‘. Whether to add caps to the segment. Turned off when using the ‘light()‘ material. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the cylinder in the scene.

Examples

```
#Generate a cylinder in the cornell box. Add a cap to both ends.

generate_cornell() |>
  add_object(cylinder(x = 555/2, y = 250, z = 555/2,
                      length = 300, radius = 100, material = metal())) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
#Rotate the cylinder
generate_cornell() |>
  add_object(cylinder(x = 555/2, y = 250, z = 555/2,
                      length = 300, radius = 100, angle = c(0, 0, 45),
                      material = diffuse())) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

# Only render a subtended arc of the cylinder, flipping the normals.
generate_cornell(lightintensity=3) |>
  add_object(cylinder(x = 555/2, y = 250, z = 555/2, capped = FALSE,
                length = 300, radius = 100, angle = c(45, 0, 0), phi_min = 0, phi_max = 180,
                      material = diffuse(), flipped = TRUE)) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

---

dielectric *Dielectric (glass) Material*

Description

Dielectric (glass) Material

Usage

```
dielectric(
  color = "white",
  refraction = 1.5,
  attenuation = c(0, 0, 0),
  attenuation_intensity = 1,
  priority = 0,
  importance_sample = FALSE,
  bump_texture = "",
  bump_intensity = 1
)
```

Arguments

color	Default 'white'. The color of the surface. Can be either a hexadecimal code, R color string, or a numeric rgb vector listing three intensities between '0' and '1'.
refraction	Default '1.5'. The index of refraction.
attenuation	Default 'c(0,0,0)'. The Beer-Lambert color-channel specific exponential attenuation through the material. Higher numbers will result in less of that color making it through the material. If a character string is provided (either as a named R color or a hex string), this will be converted to a length-3 vector equal to one minus the RGB color vector, which should approximate the color being passed. Note: This assumes the object has a closed surface.
attenuation_intensity	Default '1'. Changes the attenuation by a multiplicative factor. Values lower than one will make the dielectric more transparent, while values greater than one will make the glass more opaque.
priority	Default '0'. When two dielectric materials overlap, the one with the lower priority value is used for intersection. NOTE: If the camera is placed inside a dielectric object, its priority value will not be taken into account when determining hits to other objects also inside the object.
importance_sample	Default 'FALSE'. If 'TRUE', the object will be sampled explicitly during the rendering process. If the object is particularly important in contributing to the light paths in the image (e.g. light sources, refracting glass ball with caustics, metal objects concentrating light), this will help with the convergence of the image.
bump_texture	Default '""'. A matrix, array, or filename (specifying a greyscale image) to be used to specify a bump map for the surface.
bump_intensity	Default '1'. Intensity of the bump map. High values may lead to unphysical results.

Value

Single row of a tibble describing the dielectric material.

Examples

```
#Generate a checkered ground
scene = generate_ground(depth=-0.5, material = diffuse(checkercolor="grey30",checkerperiod=2))
render_scene(scene,parallel=TRUE, samples=16)

#Add a glass sphere
scene |>
  add_object(sphere(x=-0.5,radius=0.5,material=dielectric())) |>
  render_scene(parallel=TRUE,samples=16)

#Add a rotated colored glass cube
scene |>
  add_object(sphere(x=-0.5,radius=0.5,material=dielectric())) |>
 add_object(cube(x=0.5,xwidth=0.5,material=dielectric(color="darkgreen"),angle=c(0,-45,0))) |>
```

```
  render_scene(parallel=TRUE,samples=16)

#Add an area light behind and at an angle and turn off the ambient lighting
scene |>
  add_object(sphere(x=-0.5,radius=0.5,material=dielectric())) |>
 add_object(cube(x=0.5,xwidth=0.5,material=dielectric(color="darkgreen"),angle=c(0,-45,0))) |>
  add_object(yz_rect(z=-3,y=1,x=0,zwidth=3,ywidth=1.5,
                     material=light(intensity=15),
                     angle=c(0,-90,45), order_rotation = c(3,2,1))) |>
  render_scene(parallel=TRUE,aperture=0, ambient_light=FALSE,samples=16)

#Color glass using Beer-Lambert attenuation, which attenuates light on a per-channel
#basis as it travels through the material. This effect is what gives some types of glass
#a green glow at the edges. We will get this effect by setting a lower attenuation value
#for the `green` (second) channel in the dielectric `attenuation` argument.
generate_ground(depth=-0.5,material=diffuse(checkercolor="grey30",checkerperiod=2)) |>
  add_object(sphere(z=5,x=-0.5,y=1,material=light(intensity=10))) |>
  add_object(cube(y=0.3,ywidth=0.1,xwidth=2,zwidth=2,
                  material=dielectric(attenuation=c(1.2,0.2,1.2)),angle=c(45,110,0))) |>
  render_scene(parallel=TRUE, samples = 16)

#If you have overlapping dielectrics, the `priority` value can help disambiguate what
#object wins. Here, I place a bubble inside a cube by setting a lower priority value and
#making the inner sphere have a index of refraction of 1. I also place spheres at the corners.
generate_ground(depth=-0.51,material=diffuse(checkercolor="grey30",checkerperiod=2)) |>
  add_object(cube(material = dielectric(priority=2, attenuation = c(10,3,10)))) |>
  add_object(sphere(radius=0.49,material = dielectric(priority=1, refraction=1))) |>
  add_object(sphere(radius=0.25,x=0.5,z=-0.5,y=0.5,
                    material = dielectric(priority=0,attenuation = c(10,3,10) ))) |>
  add_object(sphere(radius=0.25,x=-0.5,z=0.5,y=0.5,
                    material = dielectric(priority=0,attenuation = c(10,3,10)))) |>
  render_scene(parallel=TRUE, samples = 16,lookfrom=c(5,1,5))

# We can also use this as a basic Constructive Solid Geometry interface by setting
# the index of refraction equal to empty space, 1. This will subtract out those regions.
# Here I make a concave lens by subtracting two spheres from a cube.
generate_ground(depth=-0.51,material=diffuse(checkercolor="grey30",checkerperiod=2,sigma=90)) |>
  add_object(cube(material = dielectric(attenuation = c(3,3,1),priority=1))) |>
  add_object(sphere(radius=1,x=1.01,
                    material = dielectric(priority=0,refraction=1))) |>
  add_object(sphere(radius=1,x=-1.01,
                    material = dielectric(priority=0,refraction=1))) |>
  add_object(sphere(y=10,x=3,material=light(intensit=150))) |>
  render_scene(parallel=TRUE, samples = 16,lookfrom=c(5,3,5))
```

`diffuse` *Diffuse Material*

Description

Diffuse Material

Usage

```
diffuse(
  color = "#ffffff",
  checkercolor = NA,
  checkerperiod = 3,
  noise = 0,
  noisephase = 0,
  noiseintensity = 10,
  noisecolor = "#000000",
  gradient_color = NA,
  gradient_transpose = FALSE,
  gradient_point_start = NA,
  gradient_point_end = NA,
  gradient_type = "hsv",
  image_texture = "",
  image_repeat = 1,
  alpha_texture = "",
  bump_texture = "",
  bump_intensity = 1,
  fog = FALSE,
  fogdensity = 0.01,
  sigma = NULL,
  importance_sample = FALSE
)
```

Arguments

color	Default ‘white‘. The color of the surface. Can be either a hexadecimal code, R color string, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.
checkercolor	Default ‘NA‘. If not ‘NA‘, determines the secondary color of the checkered surface. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.
checkerperiod	Default ‘3‘. The period of the checker pattern. Increasing this value makes the checker pattern bigger, and decreasing it makes it smaller
noise	Default ‘0‘. If not ‘0‘, covers the surface in a turbulent marble pattern. This value will determine the amount of turbulence in the texture.
noisephase	Default ‘0‘. The phase of the noise. The noise will repeat at ‘360‘.
noiseintensity	Default ‘10‘. Intensity of the noise.
noisecolor	Default ‘#000000‘. The secondary color of the noise pattern. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.
gradient_color	Default ‘NA‘. If not ‘NA‘, creates a secondary color for a linear gradient between the this color and color specified in ‘color‘. Direction is determined by ‘gradient_transpose‘.
gradient_transpose	Default ‘FALSE‘. If ‘TRUE‘, this will use the ‘v‘ coordinate texture instead of the ‘u‘ coordinate texture to map the gradient.

gradient_point_start
: Default ‘NA‘. If not ‘NA‘, this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value ‘color‘.

gradient_point_end
: Default ‘NA‘. If not ‘NA‘, this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value ‘gradient_color‘.

gradient_type
: Default ‘hsv‘. Colorspace to calculate the gradient. Alternative ‘rgb‘.

image_texture
: Default ‘""‘. A 3-layer RGB array or filename to be used as the texture on the surface of the object.

image_repeat
: Default ‘1‘. Number of times to repeat the image across the surface. ‘u‘ and ‘v‘ repeat amount can be set independently if user passes in a length-2 vector.

alpha_texture
: Default ‘""‘. A matrix or filename (specifying a greyscale image) to be used to specify the transparency.

bump_texture
: Default ‘""‘. A matrix, array, or filename (specifying a greyscale image) to be used to specify a bump map for the surface.

bump_intensity
: Default ‘1‘. Intensity of the bump map. High values may lead to unphysical results.

fog
: Default ‘FALSE‘. If ‘TRUE‘, the object will be a volumetric scatterer.

fogdensity
: Default ‘0.01‘. The density of the fog. Higher values will produce more opaque objects.

sigma
: Default ‘NULL‘. A number between 0 and Infinity specifying the roughness of the surface using the Oren-Nayar microfacet model. Higher numbers indicate a roughed surface, where sigma is the standard deviation of the microfacet orientation angle. When 0, this reverts to the default lambertian behavior.

importance_sample
: Default ‘FALSE‘. If ‘TRUE‘, the object will be sampled explicitly during the rendering process. If the object is particularly important in contributing to the light paths in the image (e.g. light sources, refracting glass ball with caustics, metal objects concentrating light), this will help with the convergence of the image.

Value

Single row of a tibble describing the diffuse material.

Examples

```
#Generate the cornell box and add a single white sphere to the center
scene = generate_cornell() |>
  add_object(sphere(x=555/2,y=555/2,z=555/2,radius=555/8,material=diffuse()))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
```

```
#Add a checkered rectangular cube below
scene = scene |>
  add_object(cube(x=555/2,y=555/8,z=555/2,xwidth=555/2,ywidth=555/4,zwidth=555/2,
  material = diffuse(checkercolor="purple",checkerperiod=20)))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)

#Add a marbled sphere
scene = scene |>
  add_object(sphere(x=555/2+555/4,y=555/2,z=555/2,radius=555/8,
  material = diffuse(noise=1/20)))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)

#Add an orange volumetric (fog) cube
scene = scene |>
  add_object(cube(x=555/2-555/4,y=555/2,z=555/2,xwidth=555/4,ywidth=555/4,zwidth=555/4,
  material = diffuse(fog=TRUE, fogdensity=0.05,color="orange")))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)

#' #Add an line segment with a color gradient
scene = scene |>
  add_object(segment(start = c(555,450,450),end=c(0,450,450),radius = 50,
                     material = diffuse(color="#1f7326", gradient_color = "#a60d0d")))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
```

`disk` *Disk Object*

Description

Disk Object

Usage

```
disk(
  x = 0,
  y = 0,
  z = 0,
  radius = 1,
  inner_radius = 0,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

x	Default ‘0‘. x-coordinate of the center of the disk
y	Default ‘0‘. y-coordinate of the center of the disk
z	Default ‘0‘. z-coordinate of the center of the disk
radius	Default ‘1‘. Radius of the disk.
inner_radius	Default ‘0‘. Inner radius of the disk.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the disk in the scene.

Examples

```
#Generate a disk in the cornell box.
generate_cornell() |>
  add_object(disk(x = 555/2, y = 50, z = 555/2, radius = 150,
                  material = diffuse(color = "orange"))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
#Rotate the disk.
generate_cornell() |>
  add_object(disk(x = 555/2, y = 555/2, z = 555/2, radius = 150, angle = c(-45, 0, 0),
                  material = diffuse(color = "orange"))) |>
  render_scene(lookfrom = c(278, 278, -800) , lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
#Pass a value for the inner radius.
generate_cornell() |>
  add_object(disk(x = 555/2, y = 555/2, z = 555/2,
                  radius = 150, inner_radius = 75, angle = c(-45, 0, 0),
                  material = diffuse(color = "orange"))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

`ellipsoid` *Ellipsoid Object*

Description

Note: this is just a scaled sphere.

Usage

```
ellipsoid(
  x = 0,
  y = 0,
  z = 0,
  a = 1,
  b = 1,
  c = 1,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

`x`	Default ‘0‘. x-coordinate of the center of the ellipsoid.
`y`	Default ‘0‘. y-coordinate of the center of the ellipsoid.
`z`	Default ‘0‘. z-coordinate of the center of the ellipsoid.
`a`	Default ‘1‘. Principal x-axis of the ellipsoid.
`b`	Default ‘1‘. Principal y-axis of the ellipsoid.
`c`	Default ‘1‘. Principal z-axis of the ellipsoid.
`material`	Default `diffuse`.The material, called from one of the material functions `diffuse`, `metal`, or `dielectric`.
`angle`	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
`order_rotation`	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
`flipped`	Default ‘FALSE‘. Whether to flip the normals.
`scale`	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the ellipsoid in the scene.

Examples

```
#Generate an ellipsoid in a Cornell box
generate_cornell() |>
  add_object(ellipsoid(x = 555/2, y = 555/2, z = 555/2,
                       a = 100, b = 50, c = 50)) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

#Change the axes to make it taller rather than wide:
generate_cornell() |>
  add_object(ellipsoid(x = 555/2, y = 555/2, z = 555/2,
                       a = 100, b = 200, c = 100, material = metal())) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

#Rotate it and make it dielectric:
generate_cornell() |>
  add_object(ellipsoid(x = 555/2, y = 555/2, z = 555/2,
                       a = 100, b = 200, c = 100, angle = c(0, 0, 45),
                       material = dielectric())) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

`extruded_path` *Extruded Path Object*

Description

Note: Bump mapping with non-diffuse materials does not work correctly, and smoothed normals will be flat when using a bump map.

Usage

```
extruded_path(
  points,
  x = 0,
  y = 0,
  z = 0,
  polygon = NA,
  polygon_end = NA,
  breaks = NA,
  closed = FALSE,
  closed_smooth = TRUE,
  polygon_add_points = 0,
  twists = 0,
  texture_repeats = 1,
  straight = FALSE,
```

```
  precomputed_control_points = FALSE,
  width = 1,
  width_end = NA,
  width_ease = "spline",
  smooth_normals = FALSE,
  u_min = 0,
  u_max = 1,
  linear_step = FALSE,
  end_caps = c(TRUE, TRUE),
  material = diffuse(),
  material_caps = NA,
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

points	Either a list of length-3 numeric vectors or 3-column matrix/data.frame specifying the x/y/z points that the path should go through.
x	Default '0'. x-coordinate offset for the path.
y	Default '0'. y-coordinate offset for the path.
z	Default '0'. z-coordinate offset for the path.
polygon	Defaults to a circle. A polygon with no holes, specified by a data.frame() parsable by 'xy.coords()'. Vertices are taken as sequential rows. If the polygon isn't closed (the last vertex equal to the first), it will be closed automatically.
polygon_end	Defaults to 'polygon'. If specified, the number of vertices should equal the to the number of vertices of the polygon set in 'polygon'. Vertices are taken as sequential rows. If the polygon isn't closed (the last vertex equal to the first), it will be closed automatically.
breaks	Defaults to '20' times the number of control points in the bezier curve.
closed	Default 'FALSE'. If 'TRUE', the path will be closed by smoothly connecting the first and last points, also ensuring the final polygon is aligned to the first.
closed_smooth	Default 'TRUE'. If 'closed = TRUE', this will ensure C2 (second derivative) continuity between the ends. If 'closed = FALSE', the curve will only have C1 (first derivative) continuity between the ends.
polygon_add_points	Default '0'. Positive integer specifying the number of points to fill in between polygon vertices. Higher numbers can give smoother results (especially when combined with 'smooth_normals = TRUE'.
twists	Default '0'. Number of twists in the polygon from one end to another.
texture_repeats	Default '1'. Number of times to repeat the texture along the length of the path.
straight	Default 'FALSE'. If 'TRUE', straight lines will be used to connect the points instead of bezier curves.

precomputed_control_points
: Default ‘FALSE‘. If ‘TRUE‘, ‘points‘ argument will expect a list of control points calculated with the internal rayrender function ‘rayrender:::calculate_control_points()‘.

width
: Default ‘0.1‘. Curve width. If ‘width_ease == "spline"‘, ‘width‘ is specified in a format that can be read by ‘xy.coords()‘ (with ‘y‘ as the width), and the ‘x‘ coordinate is between ‘0‘ and ‘1‘, this can also specify the exact positions along the curve for the corresponding width values. If a numeric vector, specifies the different values of the width evenly along the curve. If not a single value, ‘width_end‘ will be ignored.

width_end
: Default ‘NA‘. Width at end of path. Same as ‘width‘, unless specified. Ignored if multiple width values specified in ‘width‘.

width_ease
: Default ‘spline‘. Ease function between width values. Other options: ‘linear‘, ‘quad‘, ‘cubic‘, ‘exp‘.

smooth_normals
: Default ‘FALSE‘. Whether to smooth the normals of the polygon to remove sharp angles.

u_min
: Default ‘0‘. Minimum parametric coordinate for the path. If ‘closed = TRUE‘, values greater than one will refer to the beginning of the loop (but the path will be generated as two objects).

u_max
: Default ‘1‘. Maximum parametric coordinate for the path. If ‘closed = TRUE‘, values greater than one will refer to the beginning of the loop (but the path will be generated as two objects).

linear_step
: Default ‘FALSE‘. Whether the polygon intervals should be set at linear intervals, rather than intervals based on the underlying bezier curve parameterization.

end_caps
: Default ‘c(TRUE, TRUE)‘. Specifies whether to add an end cap to the beginning and end of a path.

material
: Default diffuse. The material, called from one of the material functions.

material_caps
: Defaults to the same material set in ‘material‘. Note: emissive objects may not currently function correctly when scaled.

angle
: Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.

order_rotation
: Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".

flipped
: Default ‘FALSE‘. Whether to flip the normals.

scale
: Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly.

Value

Single row of a tibble describing the cube in the scene.

Examples

```
#Specify the points for the path to travel though and the ground material
points = list(c(0,0,1),c(-0.5,0,-1),c(0,1,-1),c(1,0.5,0),c(0.6,0.3,1))
ground_mat = material=diffuse(color="grey50",
```

```
                              checkercolor = "grey20",checkerperiod = 1.5)
#Default path shape is a circle
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points, width=0.25,
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,0.5),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Change the width evenly along the tube
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points, width=0.25,
                           width_end = 0.5,
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,0.5),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Change the width along the full length of the tube
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points,
                           width=0.25*sinpi(0:72*20/180),
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,0.5),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Specify the exact parametric x positions for the width values:
custom_width = data.frame(x=c(0,0.2,0.5,0.8,1), y=c(0.25,0.5,0,0.5,0.25))
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points,
                           width=custom_width,
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,0.5),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Generate a star polygon
angles = seq(360,0,length.out=21)
xx = c(rep(c(1,0.75,0.5,0.75),5),1) * sinpi(angles/180)/4
yy = c(rep(c(1,0.75,0.5,0.75),5),1) * cospi(angles/180)/4
star_polygon = data.frame(x=xx,y=yy)

#Extrude a path using a star polygon
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points, width=0.5,
                           polygon = star_polygon,
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,1),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Specify a circle polygon
angles = seq(360,0,length.out=21)
xx = sinpi(angles/180)/4
yy = cospi(angles/180)/4
circ_polygon = data.frame(x=xx,y=yy)
```

```
#Transform from the circle polygon to the star polygon and change the end cap material
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points, width=0.5,
                           polygon=circ_polygon, polygon_end = star_polygon,
                           material_cap  = diffuse(color="white"),
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,0.5),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Add three and a half twists along the path, and make sure the breaks are evenly spaced
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points, width=0.5, twists = 3.5,
                           polygon=star_polygon, linear_step = TRUE, breaks=360,
                           material_cap  = diffuse(color="white"),
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,0),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Smooth the normals for a less sharp appearance:
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points, width=0.5, twists = 3.5,
                           polygon=star_polygon,
                           linear_step = TRUE, breaks=360,
                           smooth_normals = TRUE,
                           material_cap  = diffuse(color="white"),
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,0),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Only generate part of the curve, specified by the u_min and u_max arguments
generate_studio(depth=-0.4,material=ground_mat) |>
  add_object(extruded_path(points = points, width=0.5, twists = 3.5,
                           u_min = 0.2, u_max = 0.8,
                           polygon=star_polygon, linear_step = TRUE, breaks=360,
                           material_cap  = diffuse(color="white"),
                           material=diffuse(color="red"))) |>
  add_object(sphere(y=3,z=-5,x=2,material=light(intensity=15))) |>
  render_scene(lookat=c(0.3,0.5,0),fov=12, width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
#Render a Mobius strip with 1.5 turns
points = list(c(0,0,0),c(0.5,0.5,0),c(0,1,0),c(-0.5,0.5,0))
square_polygon = matrix(c(-1, -0.1, 0,
                           1, -0.1, 0,
                           1,  0.1, 0,
                          -1,  0.1, 0)/10, ncol=3,byrow = T)

generate_studio(depth=-0.2,
               material=diffuse(color = "dodgerblue4", checkercolor = "#002a61",
                                checkerperiod = 1)) |>
 add_object(extruded_path(points = points,  polygon=square_polygon, closed = TRUE,
                          linear_step = TRUE, twists = 1.5, breaks = 720,
                          material = diffuse(noisecolor = "black", noise = 10,
                                             noiseintensity = 10))) |>
```

```
 add_object(sphere(y=20,x=0,z=-21,material=light(intensity = 1000))) |>
 render_scene(lookat=c(0,0.5,0), fov=10, samples=16, sample_method = "sobol_blue",
             width = 800, height=800)
#Create a green glass tube with the dielectric priority interface
#and fill it with a purple neon tube light
generate_ground(depth=-0.4,material=diffuse(color="grey50",
                                      checkercolor = "grey20",checkerperiod = 1.5)) |>
  add_object(extruded_path(points = points, width=0.7, linear_step = TRUE,
                           polygon = star_polygon, twists = 2, closed = TRUE,
                           polygon_end = star_polygon, breaks=500,
                           material=dielectric(priority = 1, refraction = 1.2,
                                               attenuation=c(1,0.3,1),
                                               attenuation_intensity=20))) |>
  add_object(extruded_path(points = points, width=0.4, linear_step = TRUE,
                           polygon = star_polygon,twists = 2, closed = TRUE,
                           polygon_end = star_polygon, breaks=500,
                           material=dielectric(priority = 0,refraction = 1))) |>
  add_object(extruded_path(points = points, width=0.05, closed = TRUE,
                           material=light(color="purple", intensity = 5,
                                          importance_sample = FALSE))) |>
 add_object(sphere(y=10,z=-5,x=0,radius=5,material=light(color = "white",intensity = 5))) |>
  render_scene(lookat=c(0,0.5,1),fov=10,
               width=800,height=800, clamp_value = 10,
               aperture=0.025, samples=16, sample_method="sobol_blue")
```

---

extruded_polygon *Extruded Polygon Object*

Description

Extruded Polygon Object

Usage

```
extruded_polygon(
  polygon = NULL,
  x = 0,
  y = 0,
  z = 0,
  plane = "xz",
  top = 1,
  bottom = 0,
  holes = NULL,
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  material = diffuse(),
  center = FALSE,
  flip_horizontal = FALSE,
```

```
  flip_vertical = FALSE,
  data_column_top = NULL,
  data_column_bottom = NULL,
  scale_data = 1,
  scale = c(1, 1, 1)
)
```

Arguments

polygon	'sf' object, "SpatialPolygon" 'sp' object, or xy coordinates of polygon represented in a way that can be processed by 'xy.coords()'. If xy-coordinate based polygons are open, they will be closed by adding an edge from the last point to the first. If the 'sf' object contains MULTIPOLYGONZ data, it will flattened.
x	Default '0'. x-coordinate to offset the extruded model.
y	Default '0'. y-coordinate to offset the extruded model.
z	Default '0'. z-coordinate to offset the extruded model.
plane	Default 'xz'. The plane the polygon is drawn in. All possibile orientations are 'xz', 'zx', 'xy', 'yx', 'yz', and 'zy'.
top	Default '1'. Extruded top distance. If this equals 'bottom', the polygon will not be extruded and just the one side will be rendered.
bottom	Default '0'. Extruded bottom distance. If this equals 'top', the polygon will not be extruded and just the one side will be rendered.
holes	Default '0'. If passing in a polygon directly, this specifies which index represents the holes in the polygon. See the 'earcut' function in the 'decido' package for more information.
angle	Default 'c(0, 0, 0)'. Angle of rotation around the x, y, and z axes, applied in the order specified in 'order_rotation'.
order_rotation	Default 'c(1, 2, 3)'. The order to apply the rotations, referring to "x", "y", and "z".
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
center	Default 'FALSE'. Whether to center the polygon at the origin.
flip_horizontal	Default 'FALSE'. Flip polygon horizontally in the plane defined by 'plane'.
flip_vertical	Default 'FALSE'. Flip polygon vertically in the plane defined by 'plane'.
data_column_top	Default 'NULL'. A string indicating the column in the 'sf' object to use to specify the top of the extruded polygon.
data_column_bottom	Default 'NULL'. A string indicating the column in the 'sf' object to use to specify the bottom of the extruded polygon.
scale_data	Default '1'. If specifying 'data_column_top' or 'data_column_bottom', how much to scale that value when rendering.
scale	Default 'c(1, 1, 1)'. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Multiple row tibble describing the extruded polygon in the scene.

Examples

```
angles = seq(0, 360, by = 36)
xx = rev(c(rep(c(1, 0.5), 5), 1) * sinpi(angles / 180))
yy = rev(c(rep(c(1, 0.5), 5), 1) * cospi(angles / 180))
star_polygon = data.frame(x = xx, y = yy)

generate_ground(
 depth = 0,
 material = diffuse(color = "grey50", checkercolor = "grey20")
) |>
 add_object(extruded_polygon(
  star_polygon,
  top = 0.5,
  bottom = 0,
  material = diffuse(color = "red", sigma = 90)
 )) |>
 add_object(sphere(
  y = 4,
  x = -3,
  z = -3,
  material = light(intensity = 30)
 )) |>
 render_scene(
  parallel = TRUE,
  lookfrom = c(0, 2, 3),
  samples = 16,
  lookat = c(0, 0.5, 0),
  fov = 60
 )

#Now, let's add a hole to the center of the polygon. We'll make the polygon
#hollow by shrinking it, combining it with the normal size polygon,
#and specify with the `holes` argument that everything after `nrow(star_polygon)`
#in the following should be used to draw a hole:

hollow_star = rbind(star_polygon, 0.8 * star_polygon)

generate_ground(
 depth = -0.01,
 material = diffuse(color = "grey50", checkercolor = "grey20")
) |>
 add_object(extruded_polygon(
  hollow_star,
  top = 0.25,
  bottom = 0,
  holes = nrow(star_polygon) + 1,
  material = diffuse(color = "red", sigma = 90)
 )) |>
```

```
 add_object(sphere(
  y = 4,
  x = -3,
  z = -3,
  material = light(intensity = 30)
 )) |>
 render_scene(
  parallel = TRUE,
  lookfrom = c(0, 2, 4),
  samples = 16,
  lookat = c(0, 0, 0),
  fov = 30
 )

# Render one in the y-x plane as well by changing the `plane` argument,
# as well as offset it slightly.
generate_ground(
 depth = -0.01,
 material = diffuse(color = "grey50", checkercolor = "grey20")
) |>
 add_object(extruded_polygon(
  hollow_star,
  top = 0.25,
  bottom = 0,
  holes = nrow(star_polygon),
  material = diffuse(color = "red", sigma = 90)
 )) |>
 add_object(extruded_polygon(
  hollow_star,
  top = 0.25,
  bottom = 0,
  y = 1.2,
  z = -1.2,
  holes = nrow(star_polygon) + 1,
  plane = "yx",
  material = diffuse(color = "green", sigma = 90)
 )) |>
 add_object(sphere(y = 4, x = -3, material = light(intensity = 30))) |>
 render_scene(
  parallel = TRUE,
  lookfrom = c(0, 2, 4),
  samples = 16,
  lookat = c(0, 0.9, 0),
  fov = 40
 )

# Now add the zy plane:
generate_ground(
 depth = -0.01,
 material = diffuse(color = "grey50", checkercolor = "grey20")
) |>
 add_object(extruded_polygon(
  hollow_star,
```

```
  top = 0.25,
  bottom = 0,
  holes = nrow(star_polygon) + 1,
  material = diffuse(color = "red", sigma = 90)
 )) |>
 add_object(extruded_polygon(
  hollow_star,
  top = 0.25,
  bottom = 0,
  y = 1.2,
  z = -1.2,
  holes = nrow(star_polygon) + 1,
  plane = "yx",
  material = diffuse(color = "green", sigma = 90)
 )) |>
 add_object(extruded_polygon(
  hollow_star,
  top = 0.25,
  bottom = 0,
  y = 1.2,
  x = 1.2,
  holes = nrow(star_polygon) + 1,
  plane = "zy",
  material = diffuse(color = "blue", sigma = 90)
 )) |>
   add_object(sphere(y = 4, x = 3, material = light(intensity = 30))) |>
     render_scene(
       parallel = TRUE,
       lookfrom = c(4, 2, 4),
       samples = 16,
       lookat = c(0, 0.9, 0),
       fov = 40
     )

#We can also directly pass in sf polygons:
if (length(find.package("spData", quiet = TRUE)) > 0) {
 us_states = spData::us_states
 texas = us_states[us_states$NAME == "Texas", ]
 #Fix no sfc class in us_states geometry data
 class(texas$geometry) = c("list", "sfc")
}

#This uses the raw coordinates, unless `center = TRUE`, which centers the bounding box
#of the polygon at the origin.
generate_ground(
 depth = -0.01,
 material = diffuse(color = "grey50", checkercolor = "grey20")
) |>
 add_object(extruded_polygon(
  texas,
  center = TRUE,
  material = diffuse(color = "#ff2222", sigma = 90)
 )) |>
```

```
 add_object(sphere(
  y = 30,
  x = -30,
  radius = 10,
  material = light(color = "lightblue", intensity = 40)
 )) |>
 render_scene(
  parallel = TRUE,
  lookfrom = c(0, 10, -10),
  samples = 16,
  fov = 60
 )

#Here we use the raw coordinates, but offset the polygon manually.
generate_ground(
 depth = -0.01,
 material = diffuse(color = "grey50", checkercolor = "grey20")
) |>
 add_object(extruded_polygon(
  us_states,
  x = 96,
  z = -40,
  top = 2,
  material = diffuse(color = "#ff2222", sigma = 90)
 )) |>
 add_object(sphere(
  y = 30,
  x = -100,
  radius = 10,
  material = light(color = "dodgerblue", intensity = 200)
 )) |>
 add_object(sphere(
  y = 30,
  x = 100,
  radius = 10,
  material = light(color = "orange", intensity = 200)
 )) |>
 render_scene(
  parallel = TRUE,
  lookfrom = c(0, 120, -120),
  samples = 160,
  fov = 20
 )

#We can also set the map the height of each polygon to a column in the sf object,
#scaling it down by the maximum population state.

generate_ground(
 depth = 0,
 material = diffuse(color = "grey50", checkercolor = "grey20", sigma = 90)
) |>
 add_object(extruded_polygon(
  us_states,
```

```
 x = 96,
 z = -45,
 data_column_top = "total_pop_15",
 scale_data = 1 / max(us_states$total_pop_15) * 5,
 material = diffuse(color = "#ff2222", sigma = 90)
)) |>
add_object(sphere(
 y = 30,
 x = -100,
 z = 60,
 radius = 10,
 material = light(color = "dodgerblue", intensity = 250)
)) |>
add_object(sphere(
 y = 30,
 x = 100,
 z = -60,
 radius = 10,
 material = light(color = "orange", intensity = 200)
)) |>
render_scene(
 parallel = TRUE,
 lookfrom = c(60, 50, -40),
 lookat = c(0, -5, 0),
 samples = 160,
 fov = 30
)
```

`generate_camera_motion` *Generate Camera Movement*

Description

Takes a series of key frame camera positions and smoothly interpolates between them. Generates a data.frame that can be passed to 'render_animation()'.

Usage

```
generate_camera_motion(
  positions,
  lookats = NULL,
  apertures = 0,
  fovs = 40,
  focal_distances = NULL,
  ortho_dims = NULL,
  camera_ups = NULL,
  type = "spline",
```

```
  frames = 30,
  closed = FALSE,
  aperture_linear = TRUE,
  fov_linear = TRUE,
  focal_linear = TRUE,
  ortho_linear = TRUE,
  constant_step = TRUE,
  curvature_adjust = "none",
  curvature_scale = 30,
  offset_lookat = 0,
  damp_motion = FALSE,
  damp_magnitude = 0.1,
  progress = TRUE,
  smooth_orientation = TRUE
)
```

Arguments

positions	A list or 3-column XYZ matrix of camera positions. These will serve as key frames for the camera position. Alternatively, this can also be the a dataframe of the keyframe output from an interactive rayrender session (‘ray_keyframes‘).
lookats	Default ‘NULL‘, which sets the camera lookat to the origin ‘c(0,0,0)‘ for the animation. A list or 3-column XYZ matrix of ‘lookat‘ points. Must be the same number of points as ‘positions‘.
apertures	Default ‘0‘. A numeric vector of aperture values.
fovs	Default ‘40‘. A numeric vector of field of view values.
focal_distances	Default ‘NULL‘, automatically the distance between positions and lookats. Numeric vector of focal distances.
ortho_dims	Default ‘NULL‘, which results in ‘c(1,1)‘ orthographic dimensions. A list or 2-column matrix of orthographic dimensions.
camera_ups	Default ‘NULL‘, which gives at up vector of ‘c(0,1,0)‘. Camera up orientation.
type	Default ‘spline‘. Type of transition between keyframes. Other options are ‘linear‘, ‘quad‘, ‘cubic‘, ‘bezier‘, ‘exp‘, and ‘manual‘. ‘spline‘ keeps the path linear between keyframes while using a shape-preserving cubic Hermite spline to smoothly vary speed. Closed paths use matching boundary speeds. Direction can still change abruptly at a corner because the keyed path is preserved. ‘manual‘ just returns the values passed in, properly formatted to be passed to ‘render_animation()‘.
frames	Default ‘30‘. Total number of frames.
closed	Default ‘FALSE‘. Whether to close the camera curve so the first position matches the last. Set this to ‘TRUE‘ for perfect loops.
aperture_linear	Default ‘TRUE‘. This linearly interpolates focal distances, rather than using a smooth Bezier curve or easing function.

`fov_linear` Default ‘TRUE‘. This linearly interpolates focal distances, rather than using a smooth Bezier curve or easing function.

`focal_linear` Default ‘TRUE‘. This linearly interpolates focal distances, rather than using a smooth Bezier curve or easing function.

`ortho_linear` Default ‘TRUE‘. This linearly interpolates orthographic dimensions, rather than using a smooth Bezier curve or easing function.

`constant_step` Default ‘TRUE‘. Whether to make the camera travel at a constant speed when ‘type = "bezier"‘.

`curvature_adjust`
Default ‘none‘. Other options are ‘position‘, ‘lookat‘, and ‘both‘. Whether to slow down the camera at areas of high curvature to prevent fast swings. Only used for curve ‘type = bezier‘. This does not preserve key frame positions. Note: This feature will likely result in the ‘lookat‘ and ‘position‘ diverging if they do not have similar curvatures at each point. This feature is best used when passing the same set of points to ‘positions‘ and ‘lookats‘ and providing an ‘offset_lookat‘ value, which ensures the curvature will be the same.

`curvature_scale`
Default ‘30‘. Constant dividing factor for curvature. Higher values will subdivide the path more, potentially finding a smoother path, but increasing the calculation time. Only used for curve ‘type = bezier‘. Increasing this value after a certain point will not increase the quality of the path, but it is scene-dependent.

`offset_lookat` Default ‘0‘. Amount to offset the lookat position, either along the path (if ‘constant_step = TRUE‘) or towards the derivative of the Bezier curve.

`damp_motion` Default ‘FALSE‘. Whether to damp the motion of the camera, so that quick movements are damped and don’t result in shakey motion. This function tracks the current position, and linearly interpolates between that point and the next point using value ‘damp_magnitude‘. The equation for the position is ‘cam_current = cam_current * damp_magnitude + cam_next_point * (1 - damp_magnitude)‘.

`damp_magnitude` Default ‘0.1‘. Amount to damp the motion, a numeric value greater than ‘0‘ (no damping) and less than ‘1‘.

`progress` Default ‘TRUE‘. Whether to display a progress bar.

`smooth_orientation`
Default ‘TRUE‘. Whether to use a quaternion orientation spline for ‘spline‘, ‘linear‘, ‘quad‘, ‘cubic‘, and ‘exp‘ motion. This preserves the keyed camera positions and orientations while making angular velocity continuous through orientation keyframes. Closed paths use periodic orientation tangents. Set this to ‘FALSE‘ to interpolate lookat and up vectors directly with the selected ‘type‘.

Value

Data frame of camera positions, orientations, apertures, focal distances, and field of views

Examples

```
#Create and animate flying through a scene on a simulated roller coaster
set.seed(3)
```

```
elliplist = list()
ellip_colors = rainbow(8)
for(i in 1:1200) {
  elliplist[[i]] = ellipsoid(x=10*runif(1)-5,y=10*runif(1)-5,z=10*runif(1)-5,
                             angle = 360*runif(3), a=0.1,b=0.05,c=0.1,
                             material=glossy(color=sample(ellip_colors,1)))
}
ellip_scene = do.call(rbind, elliplist)

camera_pos = list(c(0,1,15),c(5,-5,5),c(-5,5,-5),c(0,1,-15))

#Plot the camera path and render from above using the path object:
generate_ground(material=diffuse(checkercolor="grey20"),depth=-10) |>
  add_object(ellip_scene) |>
  add_object(sphere(y=50,radius=10,material=light(intensity=30))) |>
  add_object(path(camera_pos, material=diffuse(color="red"))) |>
  render_scene(lookfrom=c(0,20,0), width=800,height=800,samples=16,
               camera_up = c(0,0,1),
               fov=80)
#Side view
generate_ground(material=diffuse(checkercolor="grey20"),depth=-10) |>
  add_object(ellip_scene) |>
  add_object(sphere(y=50,radius=10,material=light(intensity=30))) |>
  add_object(path(camera_pos, material=diffuse(color="red"))) |>
  render_scene(lookfrom=c(20,0,0),width=800,height=800,samples=16,
                 fov=80)
#View from the start
generate_ground(material=diffuse(checkercolor="grey20"),depth=-10) |>
  add_object(ellip_scene) |>
  add_object(sphere(y=50,radius=10,material=light(intensity=30))) |>
  add_object(path(camera_pos, material=diffuse(color="red"))) |>
  render_scene(lookfrom=c(0,1.5,16),width=800,height=800,samples=16,
                 fov=80)
#Generate Camera movement, setting the lookat position to be same as camera position, but offset
#slightly in front. We'll render 12 frames, but you'd likely want more in a real animation.

camera_motion =  generate_camera_motion(positions = camera_pos, lookats = camera_pos,
                                        offset_lookat = 1, fovs=80, frames=12,
                                        type="bezier")

#This returns a data frame of individual camera positions, interpolated by cubic bezier curves.
camera_motion

#Pass NA filename to plot to the device. We'll keep the path and offset it slightly to see
#where we're going. This results in a "roller coaster" effect.
generate_ground(material=diffuse(checkercolor="grey20"),depth=-10) |>
  add_object(ellip_scene) |>
  add_object(sphere(y=50,radius=10,material=light(intensity=30))) |>
  add_object(obj_model(r_obj(simple_r = TRUE),x=10,y=-10,scale_obj=3, angle=c(0,315,0),
                       material=dielectric(attenuation=c(1,1,0.3)))) |>
  add_object(pig(x=-7,y=10,z=-5,scale=1,angle=c(0,-45,80),emotion="angry")) |>
 add_object(pig(x=0,y=-0.25,z=-15,scale=1,angle=c(0,225,-22), order_rotation = c(3,2,1),
                 emotion="angry", spider=TRUE)) |>
```

```
  add_object(path(camera_pos, y=-0.2,material=diffuse(color="red"))) |>
  render_animation(filename = NA, camera_motion = camera_motion, samples=16,
                   sample_method="sobol_blue",
                   clamp_value=10, width=400, height=400)
```

`generate_cornell` *Generate Cornell Box*

Description

Generate Cornell Box

Usage

```
generate_cornell(
  light = TRUE,
  lightintensity = 5,
  lightcolor = "white",
  lightwidth = 332,
  lightdepth = 343,
  light_position = c(555/2, 554, 555/2),
  sigma = 0,
  leftcolor = "#1f7326",
  rightcolor = "#a60d0d",
  roomcolor = "#bababa",
  importance_sample = TRUE
)
```

Arguments

`light`	Default 'TRUE'. Whether to include a light on the ceiling of the box.
`lightintensity`	Default '5'. The intensity of the light.
`lightcolor`	Default 'white'. The color the of the light.
`lightwidth`	Default '332'. Width (z) of the light.
`lightdepth`	Default '343'. Depth (x) of the light.
`light_position`	Default 'c(555/2,554,555/2)'. Position of the light.
`sigma`	Default '0'. Oren-Nayar microfacet angle.
`leftcolor`	Default '#1f7326' (green).
`rightcolor`	Default '#a60d0d' (red).
`roomcolor`	Default '#bababa' (light grey).
`importance_sample`	Default 'TRUE'. Importance sample the light in the room.

Value

Tibble containing the scene description of the Cornell box.

Examples

```
#Generate and render the default Cornell box.
render_scene(generate_cornell(),
             samples=16,aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
#Make a much smaller light in the center of the room.
render_scene(generate_cornell(lightwidth=200,lightdepth=200),
             samples=16,aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
#Place a sphere in the middle of the box.
scene = generate_cornell(lightwidth=200,lightdepth=200) |>
  add_object(sphere(x=555/2,y=555/2,z=555/2,radius=555/4))
render_scene(scene, samples=16,aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
#Reduce "fireflies" by setting a clamp_value in render_scene()
render_scene(scene, samples=16,aperture=0, fov=40, ambient_light=FALSE,
             parallel=TRUE,clamp_value=3)
# Change the color scheme of the cornell box
generate_cornell(leftcolor="purple", rightcolor="yellow") |>
  render_scene(samples=16,aperture=0, fov=40, ambient_light=FALSE,
               parallel=TRUE,clamp_value=3)
```

---

generate_ground *Generate Ground*

Description

Generates a large sphere that can be used as the ground for a scene.

Usage

```
generate_ground(
  depth = -1,
  spheresize = 1000,
  material = diffuse(color = "grey50")
)
```

Arguments

depth	Default '-1'. Depth of the surface.
spheresize	Default '1000'. Radius of the sphere representing the surface.
material	Default diffuse with 'color= "#grey50"'.The material, called from one of the material functions diffuse, metal, or dielectric.
color	Default '#ccff00'. The color of the sphere. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between '0' and '1'.

Value

Single row of a tibble describing the ground.

Examples

```
#Generate the ground and add some objects
scene = generate_ground(depth=-0.5,
                        material = diffuse(noise=1,noisecolor="blue",noisephase=10)) |>
  add_object(cube(x=0.7,material=diffuse(color="red"),angle=c(0,-15,0))) |>
  add_object(sphere(x=-0.7,radius=0.5,material=dielectric(color="white")))
render_scene(scene, parallel=TRUE,lookfrom=c(0,2,10))

# Make the sphere representing the ground larger and make it a checkered surface.
scene = generate_ground(depth=-0.5, spheresize=10000,
                        material = diffuse(checkercolor="grey20")) |>
  add_object(cube(x=0.7,material=diffuse(color="red"),angle=c(0,-15,0))) |>
  add_object(sphere(x=-0.7,radius=0.5,material=dielectric(color="white")))
render_scene(scene, parallel=TRUE,lookfrom=c(0,1,10))
```

---

generate_studio *Generate Studio*

Description

Generates a curved studio backdrop.

Usage

```
generate_studio(
  depth = -1,
  distance = -10,
  width = 100,
  height = 100,
  curvature = 8,
  material = diffuse()
)
```

Arguments

depth	Default '-1'. Depth of the ground in the scene.
distance	Default '-10'. Distance to the backdrop in the scene from the origin, on the z-axis.
width	Default '100'. Width of the backdrop.
height	Default '100'. height of the backdrop.
curvature	Default '2'. Radius of the curvature connecting the bottom plane to the vertical backdrop.
material	Default diffuse with 'color= "#ccff00"'.The material, called from one of the material functions diffuse, metal, or dielectric.

Value

Tibble representing the scene.

Examples

```
#Generate the ground and add some objects
scene = generate_studio(depth=-1, material = diffuse(color="white")) |>
   add_object(obj_model(r_obj(),y=-0.5,x=0.5, scale=1.2,
                        material=glossy(color="darkred"),angle=c(0,20,0))) |>
   add_object(sphere(x=-0.5,radius=0.5,material=dielectric())) |>
   add_object(sphere(y=3,x=-2,z=-20,material=light(intensity=600)))
render_scene(scene, parallel = TRUE, lookfrom = c(0,2,-10), lookat=c(0,-0.25,0),
             fov = 14, clamp_value = 10, samples = 16)

#Zooming out to show the full default scene
render_scene(scene, parallel=TRUE,lookfrom=c(0,200,-400),clamp_value=10,samples=16)
```

---

get_camera *Get Camera*

Description

Gets a camera from a 'ray_scene'.

Usage

```
get_camera(scene, camera = NULL)
```

Arguments

scene	Scene containing cameras.
camera	Default 'NULL'. Camera name, 'ray_camera' object, or 'NULL' to use the active camera.

Value

A 'ray_camera' object.

Examples

```
scene = generate_ground(material=diffuse(color="grey20")) |>
  add_camera(camera(name = "wide", fov = 55), active = FALSE) |>
  add_camera(camera(name = "main", fov = 35), active = TRUE)

# Returns the active camera.
get_camera(scene)

# Returns a named camera.
```

```
get_camera(scene, "wide")
```

`get_infinite_light` *Get an Infinite Light*

Description

Get an Infinite Light

Usage

```
get_infinite_light(scene, name)
```

Arguments

`scene`	Scene containing infinite lights.
`name`	Name of the infinite light.

Value

A ‘ray_infinite_light‘ object.

`get_saved_keyframes` *Get Saved Keyframes*

Description

Get a dataframe of the saved keyframes (using the interactive renderer) to pass to ‘generate_camera_motion()‘

Usage

```
get_saved_keyframes()
```

Value

Data frame of keyframes

Examples

```
#This will return an empty data frame if no keyframes have been set.
get_saved_keyframes()
```

glossy *Glossy Material*

Description

Glossy Material

Usage

```
glossy(
  color = "white",
  gloss = 1,
  reflectance = 0.05,
  microfacet = "tbr",
  checkercolor = NA,
  checkerperiod = 3,
  noise = 0,
  noisephase = 0,
  noiseintensity = 10,
  noisecolor = "#000000",
  gradient_color = NA,
  gradient_transpose = FALSE,
  gradient_point_start = NA_real_,
  gradient_point_end = NA_real_,
  gradient_type = "hsv",
  image_texture = "",
  image_repeat = 1,
  alpha_texture = "",
  bump_texture = "",
  bump_intensity = 1,
  roughness_texture = "",
  roughness_range = c(1e-04, 0.2),
  roughness_flip = FALSE,
  importance_sample = FALSE
)
```

Arguments

color	Default ‘white‘. The color of the surface. Can be either a hexadecimal code, R color string, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.
gloss	Default ‘0.8‘. Gloss of the surface, between ‘1‘ (completely glossy) and ‘0‘ (rough glossy). Can be either a single number, or two numbers indicating an anisotropic distribution of normals (as in ‘microfacet()‘).
reflectance	Default ‘0.03‘. The reflectivity of the surface. ‘1‘ is a full mirror, ‘0‘ is diffuse with a glossy highlight.
microfacet	Default ‘tbr‘. Type of microfacet distribution. Alternative option ‘beckmann‘.

checkercolor Default ‘NA‘. If not ‘NA‘, determines the secondary color of the checkered surface. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.

checkerperiod Default ‘3‘. The period of the checker pattern. Increasing this value makes the checker pattern bigger, and decreasing it makes it smaller

noise Default ‘0‘. If not ‘0‘, covers the surface in a turbulent marble pattern. This value will determine the amount of turbulence in the texture.

noisephase Default ‘0‘. The phase of the noise. The noise will repeat at ‘360‘.

noiseintensity Default ‘10‘. Intensity of the noise.

noisecolor Default ‘#000000‘. The secondary color of the noise pattern. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.

gradient_color Default ‘NA‘. If not ‘NA‘, creates a secondary color for a linear gradient between the this color and color specified in ‘color‘. Direction is determined by ‘gradient_transpose‘.

gradient_transpose
Default ‘FALSE‘. If ‘TRUE‘, this will use the ‘v‘ coordinate texture instead of the ‘u‘ coordinate texture to map the gradient.

gradient_point_start
Default ‘NA‘. If not ‘NA‘, this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value ‘color‘.

gradient_point_end
Default ‘NA‘. If not ‘NA‘, this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value ‘gradient_color‘.

gradient_type Default ‘hsv‘. Colorspace to calculate the gradient. Alternative ‘rgb‘.

image_texture Default ‘""‘. A 3-layer RGB array or filename to be used as the texture on the surface of the object.

image_repeat Default ‘1‘. Number of times to repeat the image across the surface. ‘u‘ and ‘v‘ repeat amount can be set independently if user passes in a length-2 vector.

alpha_texture Default ‘""‘. A matrix or filename (specifying a greyscale image) to be used to specify the transparency.

bump_texture Default ‘""‘. A matrix, array, or filename (specifying a greyscale image) to be used to specify a bump map for the surface.

bump_intensity Default ‘1‘. Intensity of the bump map. High values may lead to unphysical results.

roughness_texture
Default ‘""‘. A matrix, array, or filename (specifying a greyscale image) to be used to specify a roughness map for the surface.

roughness_range
Default ‘ c(0.0001, 0.2)‘. This is a length-2 vector that specifies the range of roughness values that the ‘roughness_texture‘ can take.

roughness_flip Default 'FALSE'. Setting this to 'TRUE' flips the roughness values specified in the 'roughness_texture' so high values are now low values and vice versa.

importance_sample
Default 'FALSE'. If 'TRUE', the object will be sampled explicitly during the rendering process. If the object is particularly important in contributing to the light paths in the image (e.g. light sources, refracting glass ball with caustics, metal objects concentrating light), this will help with the convergence of the image.

Value

Single row of a tibble describing the glossy material.

Examples

```
#Generate a glossy sphere
generate_ground(material=diffuse(sigma=90)) |>
  add_object(sphere(y=0.2,material=glossy(color="#2b6eff"))) |>
  add_object(sphere(y=2.8,material=light())) |>
  render_scene(parallel=TRUE,clamp_value=10,samples=16,sample_method="sobol_blue")
#Change the color of the underlying diffuse layer
generate_ground(material=diffuse(sigma=90)) |>
  add_object(sphere(y=0.2,x=-2.1,material=glossy(color="#fc3d03"))) |>
  add_object(sphere(y=0.2,material=glossy(color="#2b6eff"))) |>
  add_object(sphere(y=0.2,x=2.1,material=glossy(color="#2fed4f"))) |>
  add_object(sphere(y=8,z=-5,radius=3,material=light(intensity=20))) |>
 render_scene(parallel=TRUE,clamp_value=10,samples=16,fov=40,sample_method="sobol_blue")
#Change the amount of gloss
generate_ground(material=diffuse(sigma=90)) |>
  add_object(sphere(y=0.2,x=-2.1,material=glossy(gloss=1,color="#fc3d03"))) |>
  add_object(sphere(y=0.2,material=glossy(gloss=0.5,color="#2b6eff"))) |>
  add_object(sphere(y=0.2,x=2.1,material=glossy(gloss=0,color="#2fed4f"))) |>
  add_object(sphere(y=8,z=-5,radius=3,material=light(intensity=20))) |>
 render_scene(parallel=TRUE,clamp_value=10,samples=16,fov=40,sample_method="sobol_blue")
#Add gloss to a pattern
generate_ground(material=diffuse(sigma=90)) |>
  add_object(sphere(y=0.2,x=-2.1,material=glossy(noise=2,noisecolor="black"))) |>
  add_object(sphere(y=0.2,material=glossy(color="#ff365a",checkercolor="#2b6eff"))) |>
 add_object(sphere(y=0.2,x=2.1,material=glossy(color="blue",gradient_color="#2fed4f"))) |>
  add_object(sphere(y=8,z=-5,radius=3,material=light(intensity=20))) |>
 render_scene(parallel=TRUE,clamp_value=10,samples=16,fov=40,sample_method="sobol_blue")
#Add an R and a fill light (this may look familiar)
generate_ground(material=diffuse()) |>
  add_object(sphere(y=0.2,material=glossy(color="#2b6eff",reflectance=0.05))) |>
  add_object(obj_model(r_obj(simple_r = TRUE),
                       z=-1,y=-0.05,scale=0.45,angle=c(0,180,0),material=diffuse())) |>
  add_object(sphere(y=6,z=-1,radius=4,material=light(intensity=3))) |>
  add_object(sphere(z=-15,material=light(intensity=50))) |>
  render_scene(parallel=TRUE,clamp_value=10,samples=16,sample_method="sobol_blue")
```

grid_medium *Dense Grid Participating Medium*

Description

Density samples are cell centered and trilinearly interpolated. Array indices correspond to x, y, z, with x varying fastest. Outside ‘bounds‘ the medium is vacuum. Inside the bounds, interpolation clamps to the outermost samples.

Usage

```
grid_medium(
  density,
  sigma_a = 0,
  sigma_s = 1,
  density_scale = 1,
  g = 0,
  bounds = rbind(c(-0.5, -0.5, -0.5), c(0.5, 0.5, 0.5)),
  emission = 0,
  temperature = NULL,
  emission_scale = 1,
  temperature_scale = 1,
  temperature_offset = 0,
  medium_transform = diag(4),
  haze = TRUE,
  haze_density_threshold = NULL
)
```

Arguments

density	A nonnegative numeric array with dimensions ‘c(nx, ny, nz)‘.
sigma_a	Default ‘0‘. Absorption coefficient, a nonnegative number or RGB vector, per world-space distance unit.
sigma_s	Default ‘1‘. Scattering coefficient, a nonnegative number or RGB vector, per world-space distance unit.
density_scale	Default ‘1‘. Nonnegative multiplier for both coefficients.
g	Default ‘0‘. Henyey-Greenstein asymmetry, strictly between -1 and 1. Positive values scatter forward along the incident light direction.
bounds	Default ‘rbind(c(-0.5, -0.5, -0.5), c(0.5, 0.5, 0.5))‘. Two rows giving minimum and maximum medium-space coordinates.
emission	Default ‘0‘. Scalar/RGB radiance or an array with dimensions ‘c(nx, ny, nz, 3)‘. The volume source is ‘sigma_a * Le‘.
temperature	Default ‘NULL‘. Scalar kelvin temperature or an array with the same dimensions as ‘density‘. Mutually exclusive with nonzero emission.
emission_scale	Default ‘1‘. Nonnegative multiplier for emitted radiance.

`temperature_scale`
: Default ‘1‘. Nonnegative multiplier applied after subtracting ‘temperature_offset‘ from the temperature.

`temperature_offset`
: Default ‘0‘. Offset subtracted from temperatures.

`medium_transform`
: Default ‘diag(4)‘. Invertible affine matrix mapping medium coordinates into the containing object’s local coordinates.

`haze`
: Default ‘TRUE‘. Include clear-air atmospheric haze inside this medium when enabled by [sky_light()], subject to ‘haze_density_threshold‘. Set ‘FALSE‘ to skip its atmospheric in-scattering and extinction throughout the entire boundary, including empty cells, regardless of the threshold. The medium’s own scattering, absorption, and emission remain active. In nested media the innermost medium’s setting applies; ‘sky_light(haze_in_volumes = FALSE)‘ overrides all media. This is an approximation and may affect thin clouds, edges, or low-altitude haze.

`haze_density_threshold`
: Default ‘NULL‘. With ‘haze = TRUE‘, omit haze only where interpolated density times ‘density_scale‘ is at least this positive number. Lower-density regions and empty space retain haze. A homogeneous medium has density one before scaling. ‘NULL‘ includes haze throughout the boundary. Ignored with ‘haze = FALSE‘ or global haze exclusion. The threshold measures density, not optical depth or scattering strength; choose which media to tag accordingly. Crossings follow the trilinear field, including between scattering events, and transform with the medium.

Value

A reusable ‘ray_medium‘ description.

Examples

```
if (requireNamespace("ambient", quietly = TRUE)) {
  noise_field = ambient::noise_perlin(
    dim = c(24, 24, 24), frequency = 0.02, octaves = 8, gain = 0.1
  )
  noise_field[noise_field < 0] = 0
  smoke = grid_medium(noise_field, g = 0.2, sigma_a = 10)
  scene = set_medium(cube(), smoke) |>
    add_object(sphere(x = 3, material = light(intensity = 10)))
  render_scene(
    scene, integrator_type = "nee", width = 64, height = 64,
    samples = 4, parallel = FALSE, plot_scene = FALSE
  )
}
```

group_objects *Group Objects*

Description

Group and transform objects together.

Usage

```
group_objects(
  scene,
  pivot_point = c(0, 0, 0),
  translate = c(0, 0, 0),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  scale = c(1, 1, 1),
  axis_rotation = NA
)
```

Arguments

scene	Tibble of pre-existing object locations and properties to group together.
pivot_point	Default ‘c(0,0,0)‘. The point about which to pivot, scale, and move the group.
translate	Default ‘c(0,0,0)‘. Vector indicating where to offset the group.
angle	Default ‘c(0,0,0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1,2,3)‘. The order to apply the rotations, referring to "x", "y", and "z".
scale	Default ‘c(1,1,1)‘. Scaling factor for x, y, and z directions for all objects in group.
axis_rotation	Default ‘NA‘. Provide an axis of rotation and a single angle (via ‘angle‘) of rotation around that axis.

Value

Tibble of grouped object locations and properties.

Examples

```
#Generate the ground and add some objects
scene = generate_cornell() |>
        add_object(cube(x=555/2,y=555/8,z=555/2,width=555/4)) |>
        add_object(cube(x=555/2,y=555/4+555/16,z=555/2,width=555/8))
render_scene(scene,lookfrom=c(278,278,-800),lookat = c(278,278,0), aperture=0,
             samples=16, fov=50, parallel=TRUE, clamp_value=5)
```

```
#Group the entire room and rotate around its center, but keep the cubes in the same place.
scene2 = group_objects(generate_cornell(),
                       pivot_point=c(555/2,555/2,555/2),
                       angle=c(0,30,0)) |>
         add_object(cube(x=555/2,y=555/8,z=555/2,width=555/4)) |>
        add_object(cube(x=555/2,y=555/4+555/16,z=555/2,width=555/8))

render_scene(scene2,lookfrom=c(278,278,-800),lookat = c(278,278,0), aperture=0,
             samples=16, fov=50, parallel=TRUE, clamp_value=5)
#Now group the cubes instead of the Cornell box, and rotate/translate them together
twocubes = cube(x=555/2,y=555/8,z=555/2,width=555/4) |>
           add_object(cube(x=555/2, y=555/4 + 555/16, z=555/2, width=555/8))
scene3 = generate_cornell() |>
         add_object(group_objects(twocubes, translate = c(0,50,0),angle = c(0,45,0),
         pivot_point = c(555/2,0,555/2)))

render_scene(scene3,lookfrom=c(278,278,-800),lookat = c(278,278,0), aperture=0,
             samples=16, fov=50, parallel=TRUE, clamp_value=5)
#Flatten and stretch the cubes together on two axes
scene4 = generate_cornell() |>
         add_object(group_objects(twocubes, translate = c(0,-40,0),
                                  angle = c(0,45,0), scale = c(2,0.5,1),
                                  pivot_point = c(555/2,0,555/2)))

render_scene(scene4,lookfrom=c(278,278,-800),lookat = c(278,278,0), aperture=0,
             samples=16, fov=50, parallel=TRUE, clamp_value=5)
#Add another layer of grouping, including the Cornell box
scene4 |>
 group_objects(pivot_point = c(555/2,555/2,555/2),scale=c(1.5,0.5,0.3), angle=c(-20,0,20)) |>
  render_scene(lookfrom=c(278,278,-800),lookat = c(278,278,0), aperture=0,
             samples=509, fov=50, parallel=TRUE, clamp_value=5)
```

`hair` *Hair Material*

Description

Hair Material

Usage

```
hair(
  pigment = 1.3,
  red_pigment = 0,
  color = NA,
  sigma_a = NA,
  eta = 1.55,
  beta_m = 0.3,
  beta_n = 0.3,
```

```
  alpha = 2
)
```

Arguments

pigment	Default ‘1.3‘. Concentration of the eumelanin pigment in the hair. Blonde hair has concentrations around 0.3, brown around 1.3, and black around 8.
red_pigment	Default ‘0‘.Concentration of the pheomelanin pigment in the hair. Pheomelanin makes red hair red.
color	Default ‘NA‘. Approximate color. Overrides ‘pigment‘/‘redness‘ arguments.
sigma_a	Default ‘NA‘. Attenuation. Overrides ‘color‘ and ‘pigment‘/‘redness‘ arguments.
eta	Default ‘1.55‘. Index of refraction of the hair medium.
beta_m	Default ‘0.3‘. Longitudinal roughness of the hair. Should be between 0 and 1. This roughness controls the size and shape of the hair highlight.
beta_n	Default ‘0.3‘. Azimuthal roughness of the hair. Should be between 0 and 1.
alpha	Default ‘2‘. Angle of scales on the hair surface, in degrees.

Value

Single row of a tibble describing the hair material.

Examples

```
#Create a hairball
#Generate rendom points on a sphere
lengthval = 0.5
theta = acos(2*runif(10000)-1.0);
phi = 2*pi*(runif(10000))
bezier_list = list()

#Grow the hairs
for(i in 1:length(phi)) {
  pointval = c(sin(theta[i]) * sin(phi[i]),
               cos(theta[i]),
               sin(theta[i]) * cos(phi[i]))
  bezier_list[[i]] = bezier_curve(width=0.01, width_end=0.008,
                                  p1 = pointval,
                                  p2 = (1+(lengthval*0.33))*pointval,
                                  p3 = (1+(lengthval*0.66))*pointval,
                                  p4 = (1+(lengthval)) * pointval,
                                  material=hair(pigment = 0.3, red_pigment = 1.3,
                                                beta_m = 0.3, beta_n= 0.3),
                                  type="flat")
}
hairball = dplyr::bind_rows(bezier_list)

generate_ground(depth=-2,material=diffuse(color="grey20")) |>
  add_object(sphere()) |>
```

```
  add_object(hairball) |>
 add_object(sphere(y=20,z=20,radius=5,material=light(color="white",intensity = 100))) |>
  render_scene(samples=16, lookfrom=c(0,3,10),clamp_value = 10,
               fov=20, width=800, height=800)

#Specify the color directly and increase hair roughness
for(i in 1:length(phi)) {
  pointval = c(sin(theta[i]) * sin(phi[i]),
               cos(theta[i]),
               sin(theta[i]) * cos(phi[i]))
  bezier_list[[i]] = bezier_curve(width=0.01, width_end=0.008,
                                  p1 = pointval,
                                  p2 = (1+(lengthval*0.33))*pointval,
                                  p3 = (1+(lengthval*0.66))*pointval,
                                  p4 = (1+(lengthval)) * pointval,
                                  material=hair(color="purple",
                                                beta_m = 0.5, beta_n= 0.5),
                                  type="flat")
}
hairball = dplyr::bind_rows(bezier_list)
generate_ground(depth=-2,material=diffuse(color="grey20")) |>
  add_object(sphere()) |>
  add_object(hairball) |>
 add_object(sphere(y=20,z=20,radius=5,material=light(color="white",intensity = 100))) |>
  render_scene(samples=16, lookfrom=c(0,3,10),clamp_value = 10,
               fov=20, width=800, height=800)
```

`has_denoiser` *Check for Denoiser Support*

Description

Returns TRUE if rayrender was compiled with Open Image Denoise (OIDN) support.

Usage

```
has_denoiser()
```

Value

Logical value.

Examples

```
has_denoiser()
```

homogeneous_medium *Homogeneous Participating Medium*

Description

Describe a medium independently of the surface that contains it. Attach it to closed objects with [set_medium()]. Rendering automatically selects `integrator_type = "nee"` for scenes containing attached media.

Usage

```
homogeneous_medium(
  sigma_a = 0,
  sigma_s = 1,
  density_scale = 1,
  g = 0,
  emission = 0,
  temperature = NULL,
  emission_scale = 1,
  temperature_scale = 1,
  temperature_offset = 0,
  medium_transform = diag(4),
  haze = TRUE,
  haze_density_threshold = NULL
)
```

Arguments

`sigma_a`	Default ‘0‘. Absorption coefficient, a nonnegative number or RGB vector, per world-space distance unit.
`sigma_s`	Default ‘1‘. Scattering coefficient, a nonnegative number or RGB vector, per world-space distance unit.
`density_scale`	Default ‘1‘. Nonnegative multiplier for both coefficients.
`g`	Default ‘0‘. Henyey-Greenstein asymmetry, strictly between -1 and 1. Positive values scatter forward along the incident light direction.
`emission`	Default ‘0‘. Nonnegative scalar or RGB emitted radiance ‘Le‘. Color names are also accepted. The volume source is ‘sigma_a * Le‘.
`temperature`	Default ‘NULL‘. Blackbody temperature in kelvin. Mutually exclusive with nonzero ‘emission‘.
`emission_scale`	Default ‘1‘. Nonnegative multiplier for emitted radiance.
`temperature_scale`	Default ‘1‘. Nonnegative multiplier applied after subtracting ‘temperature_offset‘ from the temperature.
`temperature_offset`	Default ‘0‘. Offset subtracted from temperatures.

`medium_transform`
: Default 'diag(4)'. Invertible affine matrix mapping medium coordinates into the containing object's local coordinates.

`haze`
: Default 'TRUE'. Include clear-air atmospheric haze inside this medium when enabled by [sky_light()], subject to 'haze_density_threshold'. Set 'FALSE' to skip its atmospheric in-scattering and extinction throughout the entire boundary, including empty cells, regardless of the threshold. The medium's own scattering, absorption, and emission remain active. In nested media the innermost medium's setting applies; 'sky_light(haze_in_volumes = FALSE)' overrides all media. This is an approximation and may affect thin clouds, edges, or low-altitude haze.

`haze_density_threshold`
: Default 'NULL'. With 'haze = TRUE', omit haze only where interpolated density times 'density_scale' is at least this positive number. Lower-density regions and empty space retain haze. A homogeneous medium has density one before scaling. 'NULL' includes haze throughout the boundary. Ignored with 'haze = FALSE' or global haze exclusion. The threshold measures density, not optical depth or scattering strength; choose which media to tag accordingly. Crossings follow the trilinear field, including between scattering events, and transform with the medium.

Value

A reusable 'ray_medium' description.

Examples

```
fog = homogeneous_medium(sigma_s = 5, g = 0.65)
scene = set_medium(cube(width=1.2), fog) |>
add_object(sphere(y=1,x=1,radius=0.3,material=light(intensity=20))) |>
add_object(generate_studio(material=diffuse(color="dodgerblue") ))
render_scene(scene, integrator_type = "nee", sample_method="sobol")

# Foggy cornell box
fog = homogeneous_medium(sigma_s = 0.1, sigma_a = 0.1, g = 0, density_scale = 0.01, temperature = 0)
 scene = generate_cornell(
 lightwidth = 10,
 lightdepth = 10,
 lightintensity = 3000
) |>
 add_object(set_medium(
   cube(x = 555 / 2, y = 555 / 2, z = 555 / 2, width = 554),
   fog
 ))
render_scene(
 scene,
 integrator_type = "nee",
 fov = 40,
 samples = 256
)
```

`infinite_light` *Image-Based Infinite Light*

Description

Creates an infinite light from an equirectangular environment image. Attach lights with `add_infinite_light()`. Their radiance adds together in the background, reflections, and illumination of surfaces and volumes.

Usage

```
infinite_light(filename, intensity = 1, rotation = 0, name = "environment")
```

Arguments

`filename`	Environment image filename. Supports EXR, HDR, PNG, and JPEG, using the same image loading and color conventions as `environment_light` in `render_scene()`. HDR and EXR preserve linear high dynamic range values.
`intensity`	Default `1`. Nonnegative multiplier for this light's radiance.
`rotation`	Default `0`. Rotation in degrees around the world Y axis, with the same direction as `rotate_env` in `render_scene()`.
`name`	Default `"environment"`. Unique name within the scene.

Details

Infinite lights are scene metadata, like cameras, and do not add geometry or change scene bounds. Grouping geometry does not transform them. `add_object()` preserves lights when combining scenes and rejects duplicate names. Scene lights suppress automatic fallback illumination, including when all their intensities are zero. An explicit `environment_light` render argument adds another image light; `intensity_env` controls that legacy light only. `rotate_env` and interactive environment rotation rotate all infinite lights together, in addition to each light's own rotation.

Additive lighting lets you keep an existing HDRI and attach independently adjustable fill lights. At each direction, the background radiance is the sum of the lights after applying their intensities and rotations. For example, a cool fill can reveal detail in blue materials under a warm environment. Keep exposure fixed when comparing the result to see the added illumination.

Value

A `ray_infinite_light` object.

Examples

```
# Write explicitly linear RGB radiance to EXR, including values greater than 1.
write_environment = function(image) {
  file = tempfile(fileext = ".exr")
  image = rayimage::ray_read_image(
    image,
```

```
    normalize = FALSE,
    source_linear = TRUE,
    assume_colorspace = rayimage::CS_SRGB
  )
  rayimage::ray_write_image(image, file, clamp = FALSE)
  file
}

# 1. Add an independently adjustable fill to an existing environment.
# These constant-color maps make the color and brightness changes easy to see.
warm = cool = array(0, c(32, 64, 3))
warm_rgb = c(0.8, 0.3, 0.1)
cool_rgb = c(0.1, 0.3, 0.8)
for (channel in 1:3) {
  warm[,, channel] = warm_rgb[channel]
  cool[,, channel] = cool_rgb[channel]
}
warm_file = write_environment(warm)
cool_file = write_environment(cool)

# The included R logo retains its blue lettering and gray surround.
# The ground and oblique camera reveal contact shadows and the beveled edges.
base_scene = obj_model(r_obj(), scale = 2.5) |>
  add_object(generate_ground(
    depth = -0.96,
    material = diffuse(color = "grey20")
  )) |>
  add_camera(camera(
    lookfrom = c(2.5, 1.2, -6),
    lookat = c(0, -0.05, 0),
    fov = 28,
    aperture = 0
  ))
warm_scene = base_scene |>
  add_infinite_light(infinite_light(warm_file, name = "warm"))
set.seed(724)
render_scene(
  warm_scene,
  width = 600,
  height = 500,
  samples = 128,
  integrator_type = "nee",
  bloom = FALSE
)

# The two lights contribute warm + 0.5 * cool. The extra cool radiance
# lifts the blue lettering and shifts the background color.
# Keep exposure fixed between renders to see both the color and brightness change.
# The same approach adds fill to a photographic HDRI without editing that image.
additive_scene = warm_scene |>
  add_infinite_light(infinite_light(cool_file, intensity = 0.5, name = "cool"))
set.seed(724)
render_scene(
```

```
  additive_scene,
  width = 600,
  height = 500,
  samples = 128,
  integrator_type = "nee",
  bloom = FALSE
)

# 2. Warm and cool fills on opposite sides, like two broad studio softboxes.
# Each EXR is dark except for a bright patch above the horizon. The patch is
# one-quarter across the map. Rotations of 30 and 210 degrees put the patches
# on opposite sides. A dim neutral environment keeps the front readable.
u = (seq_len(256) - 0.5) / 256
v = (seq_len(128) - 0.5) / 128
# Longitude wraps around: keep the softbox continuous at the EXR's edges.
du = ((u - 0.25 + 0.5) %% 1) - 0.5
softbox = outer(
  exp(-((v - 0.32) / 0.18)^2),
  exp(-(du / 0.12)^2)
)
warm_side = cool_side = array(0, c(128, 256, 3))
for (channel in 1:3) {
  warm_side[,, channel] = 8 * warm_rgb[channel] * softbox
  cool_side[,, channel] = 8 * cool_rgb[channel] * softbox
}
neutral_file = write_environment(array(0.08, c(32, 64, 3)))
warm_side_file = write_environment(warm_side)
cool_side_file = write_environment(cool_side)
fill_scene = base_scene |>
  add_infinite_light(infinite_light(neutral_file, name = "neutral")) |>
  add_infinite_light(infinite_light(
    warm_side_file,
    rotation = 30,
    name = "warm_fill"
  )) |>
  add_infinite_light(infinite_light(
    cool_side_file,
    rotation = 210,
    intensity = 0.75,
    name = "cool_fill"
  ))
set.seed(724)
render_scene(
  fill_scene,
  width = 600,
  height = 500,
  samples = 128,
  integrator_type = "nee",
  bloom = FALSE
)

# Change either fill's intensity or rotation independently to shape the lighting.
# This gives independent control over both sides while retaining the base light.
```

lambertian *Lambertian Material (deprecated)*

Description

Lambertian Material (deprecated)

Usage

```
lambertian(...)
```

Arguments

...	Arguments to pass to diffuse() function.

Value

Single row of a tibble describing the diffuse material.

Examples

```
#Deprecated lambertian material. Will display a warning.
scene = generate_cornell() |>
  add_object(sphere(x=555/2,y=555/2,z=555/2,radius=555/8,material=lambertian()))
  render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
              aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
```

light *Light Material*

Description

Light Material

Usage

```
light(
  color = "#ffffff",
  intensity = 10,
  importance_sample = TRUE,
  spotlight_focus = NA,
  spotlight_width = 30,
  spotlight_start_falloff = 15,
  invisible = FALSE,
```

```
  image_texture = "",
  image_repeat = 1,
  gradient_color = NA,
  gradient_transpose = FALSE,
  gradient_point_start = NA,
  gradient_point_end = NA,
  gradient_type = "hsv"
)
```

Arguments

color	Default 'white'. The color of the light Can be either a hexadecimal code, R color string, or a numeric rgb vector listing three intensities between '0' and '1'.
intensity	Default '10'. If a positive value, this will turn this object into a light emitting the value specified in 'color' (ignoring other properties). Higher values will produce a brighter light.
importance_sample	Default 'TRUE'. Keeping this on for lights improves the convergence of the rendering algorithm, in most cases. If the object is particularly important in contributing to the light paths in the image (e.g. light sources, refracting glass ball with caustics, metal objects concentrating light), this will help with the convergence of the image.
spotlight_focus	Default 'NA', no spotlight. Otherwise, a length-3 numeric vector specifying the x/y/z coordinates that the spotlight should be focused on. Only works for spheres and rectangles.
spotlight_width	Default '30'. Angular width of the spotlight.
spotlight_start_falloff	Default '15'. Angle at which the light starts fading in intensity.
invisible	Default 'FALSE'. If 'TRUE', the light itself will be invisible.
image_texture	Default '""'. A 3-layer RGB array or filename to be used as the texture on the surface of the object.
image_repeat	Default '1'. Number of times to repeat the image across the surface. 'u' and 'v' repeat amount can be set independently if user passes in a length-2 vector.
gradient_color	Default 'NA'. If not 'NA', creates a secondary color for a linear gradient between the this color and color specified in 'color'. Direction is determined by 'gradient_transpose'.
gradient_transpose	Default 'FALSE'. If 'TRUE', this will use the 'v' coordinate texture instead of the 'u' coordinate texture to map the gradient.
gradient_point_start	Default 'NA'. If not 'NA', this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value 'color'.

gradient_point_end
: Default 'NA'. If not 'NA', this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value 'gradient_color'.

gradient_type
: Default 'hsv'. Colorspace to calculate the gradient. Alternative 'rgb'.

Value

Single row of a tibble describing the light material.

Examples

```
#Generate the cornell box without a light and add a single white sphere to the center
scene = generate_cornell(light=FALSE) |>
  add_object(sphere(x=555/2,y=555/2,z=555/2,radius=555/8,material=light()))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)

#Remove the light for direct camera rays, but keep the lighting
scene = generate_cornell(light=FALSE) |>
  add_object(sphere(x=555/2,y=555/2,z=555/2,radius=555/8,
             material=light(intensity=15,invisible=TRUE)))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)

#All gather around the orb
scene = generate_ground(material = diffuse(checkercolor="grey50")) |>
  add_object(sphere(radius=0.25,material=light(intensity=90,color="#f11"))) |>
  add_object(obj_model(r_obj(), scale=2.5,z=-3,x=-1.25,y=0, angle=c(0,235,0))) |>
  add_object(pig(scale=0.3, x=1.5,z=-2,y=-1.5,angle=c(0,-135,0)))
render_scene(scene, samples=16, parallel=TRUE, clamp_value=10, lookfrom=c(0,0,10))
```

---

list_cameras *List Cameras*

Description

Lists cameras attached to a 'ray_scene'.

Usage

```
list_cameras(scene)
```

Arguments

scene
: Scene containing cameras.

Value

A data frame with one row per camera.

Examples

```
scene = generate_ground(material=diffuse(color="grey20")) |>
  add_camera(camera(name = "wide", fov = 55, filename = "wide.png")) |>
  add_camera(camera(name = "detail", fov = 20, filename = "detail.png"))

list_cameras(scene)
```

---

list_infinite_lights *List Infinite Lights*

Description

List Infinite Lights

Usage

```
list_infinite_lights(scene)
```

Arguments

scene	Scene containing infinite lights.

Value

A named list of ‘ray_infinite_light‘ objects.

---

mesh3d_model *‘mesh3d‘ model*

Description

Load an ‘mesh3d‘ (or ‘shapelist3d‘) object, as specified in the ‘rgl‘ package.

Usage

```
mesh3d_model(
  mesh,
  x = 0,
  y = 0,
  z = 0,
  swap_yz = FALSE,
  reverse = FALSE,
  subdivision_levels = 1,
  verbose = FALSE,
  displacement_texture = "",
  displacement_intensity = 1,
  displacement_vector = FALSE,
  recalculate_normals = FALSE,
  override_material = FALSE,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

mesh	A 'mesh3d' or 'shapelist3d' object. Pulls the vertex, index, texture coordinates, normals, and material information. If the material references an image texture, the 'mesh$material$texture' argument should be set to the image filename. The 'mesh3d' format only supports one image texture per mesh. All quads will be triangulated.
x	Default '0'. x-coordinate to offset the model.
y	Default '0'. y-coordinate to offset the model.
z	Default '0'. z-coordinate to offset the model.
swap_yz	Default 'FALSE'. Swap the Y and Z coordinates.
reverse	Default 'FALSE'. Reverse the orientation of the indices, flipping their normals.
subdivision_levels	Default '1'. Number of Loop subdivisions to be applied to the mesh.
verbose	Default 'FALSE'. If 'TRUE', prints information about the mesh to the console.
displacement_texture	Default '""'. File path to the displacement texture. This texture is used to displace the vertices of the mesh based on the texture's pixel values.
displacement_intensity	Default '1'. Intensity of the displacement effect. Higher values result in greater displacement.
displacement_vector	Default 'FALSE'. Whether to use vector displacement. If 'TRUE', the displacement texture is interpreted as providing a 3D displacement vector. Otherwise, the texture is interpreted as providing a scalar displacement.

recalculate_normals
: Default 'FALSE'. Whether to recalculate vertex normals based on the connecting face orientations. This can be used to compute normals for meshes lacking them or to calculate new normals after a displacement map has been applied to the mesh.

override_material
: Default 'FALSE'. If 'TRUE', overrides the material specified in the 'mesh3d' object with the one specified in 'material'.

material
: Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.

angle
: Default 'c(0, 0, 0)'. Angle of rotation around the x, y, and z axes, applied in the order specified in 'order_rotation'.

order_rotation
: Default 'c(1, 2, 3)'. The order to apply the rotations, referring to "x", "y", and "z".

flipped
: Default 'FALSE'. Whether to flip the normals.

scale
: Default 'c(1, 1, 1)'. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the mesh3d model in the scene.

Examples

```
#Load a mesh3d object (from the Rvcg) and render it:
  library(Rvcg)
  data(humface)

  generate_studio() |>
    add_object(mesh3d_model(humface,y=-0.3,x=0,z=0,
                            material=glossy(color="dodgerblue4"), scale = 1/70,
                            angle = c(0,180,0))) |>
    add_object(sphere(y=5,x=5,z=-5,material=light(intensity=50))) |>
    render_scene(samples=16,width=800,height=800,
                 lookat = c(0,0.5,1), aperture=0.0)
```

---

metal *Metallic Material*

Description

Metallic Material

Usage

```
metal(
  color = "#ffffff",
  eta = 0,
  kappa = 0,
  fuzz = 0,
  checkercolor = NA,
  checkerperiod = 3,
  noise = 0,
  noisephase = 0,
  noiseintensity = 10,
  noisecolor = "#000000",
  gradient_color = NA,
  gradient_transpose = FALSE,
  gradient_point_start = NA,
  gradient_point_end = NA,
  gradient_type = "hsv",
  image_texture = "",
  image_repeat = 1,
  alpha_texture = "",
  bump_texture = "",
  bump_intensity = 1,
  importance_sample = FALSE
)
```

Arguments

color	Default 'white'. The color of the sphere. Can be either a hexadecimal code, R color string, or a numeric rgb vector listing three intensities between '0' and '1'.
eta	Default '0'. Wavelength dependent refractivity of the material (red, green, and blue channels). If single number, will be repeated across all three channels.
kappa	Default '0'. Wavelength dependent absorption of the material (red, green, and blue channels). If single number, will be repeated across all three channels.
fuzz	Default '0'. Deprecated–Use the microfacet material instead, as it is designed for rough metals. The roughness of the metallic surface. Maximum '1'.
checkercolor	Default 'NA'. If not 'NA', determines the secondary color of the checkered surface. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between '0' and '1'.
checkerperiod	Default '3'. The period of the checker pattern. Increasing this value makes the checker pattern bigger, and decreasing it makes it smaller
noise	Default '0'. If not '0', covers the surface in a turbulent marble pattern. This value will determine the amount of turbulence in the texture.
noisephase	Default '0'. The phase of the noise. The noise will repeat at '360'.
noiseintensity	Default '10'. Intensity of the noise.

noisecolor Default '#000000'. The secondary color of the noise pattern. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between '0' and '1'.

gradient_color Default 'NA'. If not 'NA', creates a secondary color for a linear gradient between the this color and color specified in 'color'. Direction is determined by 'gradient_transpose'.

gradient_transpose
Default 'FALSE'. If 'TRUE', this will use the 'v' coordinate texture instead of the 'u' coordinate texture to map the gradient.

gradient_point_start
Default 'NA'. If not 'NA', this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value 'color'.

gradient_point_end
Default 'NA'. If not 'NA', this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value 'gradient_color'.

gradient_type Default 'hsv'. Colorspace to calculate the gradient. Alternative 'rgb'.

image_texture Default '""'. A 3-layer RGB array or filename to be used as the texture on the surface of the object.

image_repeat Default '1'. Number of times to repeat the image across the surface. 'u' and 'v' repeat amount can be set independently if user passes in a length-2 vector.

alpha_texture Default '""'. A matrix or filename (specifying a greyscale image) to be used to specify the transparency.

bump_texture Default '""'. A matrix, array, or filename (specifying a greyscale image) to be used to specify a bump map for the surface.

bump_intensity Default '1'. Intensity of the bump map. High values may lead to unphysical results.

importance_sample
Default 'FALSE'. If 'TRUE', the object will be sampled explicitly during the rendering process. If the object is particularly important in contributing to the light paths in the image (e.g. light sources, refracting glass ball with caustics, metal objects concentrating light), this will help with the convergence of the image.

Value

Single row of a tibble describing the metallic material.

Examples

```
# Generate the cornell box with a single chrome sphere in the center. For other metals,
# See the website refractiveindex.info for eta and k data, use wavelengths 5
# 80nm (R), 530nm (G), and 430nm (B).
scene = generate_cornell() |>
```

```
  add_object(sphere(x=555/2,y=555/2,z=555/2,radius=555/8,
  material=metal(eta=c(3.2176,3.1029,2.1839), k = c(3.3018,3.33,3.0339))))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
#Add an aluminum rotated shiny metal block
scene = scene |>
  add_object(cube(x=380,y=150/2,z=200,xwidth=150,ywidth=150,zwidth=150,
 material = metal(eta = c(1.07,0.8946,0.523), k = c(6.7144,6.188,4.95)),angle=c(0,45,0)))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
#Add a copper metal cube
scene = scene |>
  add_object(cube(x=150,y=150/2,z=300,xwidth=150,ywidth=150,zwidth=150,
                  material = metal(eta = c(0.497,0.8231,1.338),
                                   k = c(2.898,2.476,2.298)),
                  angle=c(0,-30,0)))
render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)

#Finally, let's add a lead pipe
scene2 = scene |>
  add_object(cylinder(x=450,y=200,z=400,length=400,radius=30,
                  material = metal(eta = c(1.44,1.78,1.9),
                                   k = c(3.18,3.36,3.43)),
                  angle=c(0,-30,0)))
render_scene(scene2, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
```

`microfacet` *Microfacet Material*

Description

Microfacet Material

Usage

```
microfacet(
  color = "white",
  roughness = 1e-04,
  transmission = FALSE,
  eta = 0,
  kappa = 0,
  microfacet = "tbr",
  checkercolor = NA,
  checkerperiod = 3,
  noise = 0,
  noisephase = 0,
```

```
  noiseintensity = 10,
  noisecolor = "#000000",
  gradient_color = NA,
  gradient_transpose = FALSE,
  gradient_point_start = NA_real_,
  gradient_point_end = NA_real_,
  gradient_type = "hsv",
  image_texture = "",
  image_repeat = 1,
  alpha_texture = "",
  bump_texture = "",
  bump_intensity = 1,
  roughness_texture = "",
  roughness_range = c(1e-04, 0.2),
  roughness_flip = FALSE,
  importance_sample = FALSE
)
```

Arguments

color	Default ‘white‘. The color of the surface. Can be either a hexadecimal code, R color string, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.
roughness	Default ‘0.0001‘. Roughness of the surface, between ‘0‘ (smooth) and ‘1‘ (diffuse). Can be either a single number, or two numbers indicating an anisotropic distribution of normals. ‘0‘ is a smooth surface, while ‘1‘ is extremely rough. This can be used to create a wide-variety of materials (e.g. ‘0-0.01‘ is specular metal, ‘0.02‘-‘0.1‘ is brushed metal, ‘0.1‘-‘0.3‘ is a rough metallic surface , ‘0.3‘-‘0.5‘ is diffuse, and above that is a rough satin-like material). Two numbers will specify the x and y roughness separately (e.g. ‘roughness = c(0.01, 0.001)‘ gives an etched metal effect). If ‘0‘, this defaults to the ‘metal()‘ material for faster evaluation.
transmission	Default ‘FALSE‘. If ‘TRUE‘, this material will be a rough dielectric instead of a rough metallic surface.
eta	Default ‘0‘. Wavelength dependent refractivity of the material (red, green, and blue channels). If single number, will be repeated across all three channels. If ‘transmission = TRUE‘, this is a single value representing the index of refraction of the material.
kappa	Default ‘0‘. Wavelength dependent absorption of the material (red, green, and blue channels). If single number, will be repeated across all three channels. If ‘transmission = TRUE‘, this length-3 vector specifies the attenuation of the dielectric (analogous to the dielectric ‘attenuation‘ argument).
microfacet	Default ‘tbr‘. Type of microfacet distribution. Alternative option ‘beckmann‘.
checkercolor	Default ‘NA‘. If not ‘NA‘, determines the secondary color of the checkered surface. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.
checkerperiod	Default ‘3‘. The period of the checker pattern. Increasing this value makes the checker pattern bigger, and decreasing it makes it smaller

noise Default ‘0‘. If not ‘0‘, covers the surface in a turbulent marble pattern. This value will determine the amount of turbulence in the texture.

noisephase Default ‘0‘. The phase of the noise. The noise will repeat at ‘360‘.

noiseintensity Default ‘10‘. Intensity of the noise.

noisecolor Default ‘#000000‘. The secondary color of the noise pattern. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.

gradient_color Default ‘NA‘. If not ‘NA‘, creates a secondary color for a linear gradient between the this color and color specified in ‘color‘. Direction is determined by ‘gradient_transpose‘.

gradient_transpose
Default ‘FALSE‘. If ‘TRUE‘, this will use the ‘v‘ coordinate texture instead of the ‘u‘ coordinate texture to map the gradient.

gradient_point_start
Default ‘NA‘. If not ‘NA‘, this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value ‘color‘.

gradient_point_end
Default ‘NA‘. If not ‘NA‘, this changes the behavior from mapping texture coordinates to mapping to world space coordinates. This should be a length-3 vector specifying the x,y, and z points where the gradient begins with value ‘gradient_color‘.

gradient_type Default ‘hsv‘. Colorspace to calculate the gradient. Alternative ‘rgb‘.

image_texture Default ‘""‘. A 3-layer RGB array or filename to be used as the texture on the surface of the object.

image_repeat Default ‘1‘. Number of times to repeat the image across the surface. ‘u‘ and ‘v‘ repeat amount can be set independently if user passes in a length-2 vector.

alpha_texture Default ‘""‘. A matrix or filename (specifying a greyscale image) to be used to specify the transparency.

bump_texture Default ‘""‘. A matrix, array, or filename (specifying a greyscale image) to be used to specify a bump map for the surface.

bump_intensity Default ‘1‘. Intensity of the bump map. High values may lead to unphysical results.

roughness_texture
Default ‘""‘. A matrix, array, or filename (specifying a greyscale image) to be used to specify a roughness map for the surface.

roughness_range
Default ‘ c(0.0001, 0.2)‘. This is a length-2 vector that specifies the range of roughness values that the ‘roughness_texture‘ can take.

roughness_flip Default ‘FALSE‘. Setting this to ‘TRUE‘ flips the roughness values specified in the ‘roughness_texture‘ so high values are now low values and vice versa.

importance_sample
Default ‘FALSE‘. If ‘TRUE‘, the object will be sampled explicitly during the rendering process. If the object is particularly important in contributing to the

light paths in the image (e.g. light sources, refracting glass ball with caustics, metal objects concentrating light), this will help with the convergence of the image.

Value

Single row of a tibble describing the microfacet material.

Examples

```
# Generate a golden egg, using eta and kappa taken from physical measurements
# See the website refractiveindex.info for eta and k data, use
# wavelengths 580nm (R), 530nm (G), and 430nm (B).
generate_cornell() |>
  add_object(ellipsoid(x=555/2,555/2,y=150, a=100,b=150,c=100,
             material=microfacet(roughness=0.1,
                         eta=c(0.216,0.42833,1.3184), kappa=c(3.239,2.4599,1.8661)))) |>
 render_scene(lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, parallel=TRUE,clamp_value=10)
#Make the roughness anisotropic (either horizontal or vertical), adding an extra light in front
#to show off the different microfacet orientations
generate_cornell() |>
  add_object(sphere(x=555/2,z=50,y=75,radius=20,material=light())) |>
  add_object(ellipsoid(x=555-150,555/2,y=150, a=100,b=150,c=100,
             material=microfacet(roughness=c(0.3,0.1),
                         eta=c(0.216,0.42833,1.3184), kappa=c(3.239,2.4599,1.8661)))) |>
 add_object(ellipsoid(x=150,555/2,y=150, a=100,b=150,c=100,
             material=microfacet(roughness=c(0.1,0.3),
                         eta=c(0.216,0.42833,1.3184), kappa=c(3.239,2.4599,1.8661)))) |>
 render_scene(lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40,  parallel=TRUE,clamp_value=10)
#Render a rough silver R with a smaller golden egg in front
generate_cornell() |>
  add_object(obj_model(r_obj(simple_r = TRUE),
                       x=555/2,z=350,y=0, scale_obj = 200, angle=c(0,200,0),
             material=microfacet(roughness=0.2,
                          eta=c(1.1583,0.9302,0.5996), kappa=c(6.9650,6.396,5.332)))) |>
 add_object(ellipsoid(x=200,z=200,y=80, a=50,b=80,c=50,
             material=microfacet(roughness=0.1,
                         eta=c(0.216,0.42833,1.3184), kappa=c(3.239,2.4599,1.8661)))) |>
 render_scene(lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
             aperture=0, fov=40, parallel=TRUE,clamp_value=10)
#Increase the roughness
generate_cornell() |>
  add_object(obj_model(r_obj(simple_r = TRUE),
                       x=555/2,z=350,y=0, scale_obj = 200, angle=c(0,200,0),
             material=microfacet(roughness=0.5,
                          eta=c(1.1583,0.9302,0.5996), kappa=c(6.9650,6.396,5.332)))) |>
 add_object(ellipsoid(x=200,z=200,y=80, a=50,b=80,c=50,
             material=microfacet(roughness=0.3,
                         eta=c(0.216,0.42833,1.3184), kappa=c(3.239,2.4599,1.8661)))) |>
 render_scene(lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=16,
```

```
            aperture=0, fov=40, parallel=TRUE,clamp_value=10)
 #Use transmission for a rough dielectric
generate_cornell() |>
  add_object(obj_model(r_obj(simple_r = TRUE),
                       x=555/2,z=350,y=0, scale_obj = 200, angle=c(0,200,0),
            material=microfacet(roughness=0.3, transmission=T, eta=1.6))) |>
 add_object(ellipsoid(x=200,z=200,y=80, a=50,b=80,c=50,
            material=microfacet(roughness=0.3, transmission=T, eta=1.6))) |>
 render_scene(lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=64,
            aperture=0, fov=40, parallel=TRUE,clamp_value=10, min_variance=1e-6)
```

---

nanovdb_medium *NanoVDB Participating Medium*

Description

Read uncompressed float grids from a NanoVDB file. Native grid coordinates and transforms are retained; use ‘medium_transform‘ to place them inside the boundary. Convert OpenVDB or compressed NanoVDB files externally before use.

Usage

```
nanovdb_medium(
  filename,
  sigma_a = 0,
  sigma_s = 1,
  density_scale = 1,
  g = 0,
  density_grid = "density",
  temperature_grid = NULL,
  emission = 0,
  emission_scale = 1,
  temperature_scale = 1,
  temperature_offset = 0,
  medium_transform = diag(4),
  haze = TRUE,
  haze_density_threshold = NULL
)
```

Arguments

filename	Path to an uncompressed ‘.nvdb‘ file.
sigma_a	Default ‘0‘. Absorption coefficient, a nonnegative number or RGB vector, per world-space distance unit.
sigma_s	Default ‘1‘. Scattering coefficient, a nonnegative number or RGB vector, per world-space distance unit.

`density_scale` Default ‘1‘. Nonnegative multiplier for both coefficients.

`g` Default ‘0‘. Henyey-Greenstein asymmetry, strictly between -1 and 1. Positive values scatter forward along the incident light direction.

`density_grid` Default ‘"density"‘. Name of the float density grid.

`temperature_grid`
Default ‘NULL‘. Optional float temperature grid name.

`emission` Default ‘0‘. Nonnegative scalar or RGB emitted radiance ‘Le‘. Color names are also accepted. The volume source is ‘sigma_a * Le‘.

`emission_scale` Default ‘1‘. Nonnegative multiplier for emitted radiance.

`temperature_scale`
Default ‘1‘. Nonnegative multiplier applied after subtracting ‘temperature_offset‘ from the temperature.

`temperature_offset`
Default ‘0‘. Offset subtracted from temperatures.

`medium_transform`
Default ‘diag(4)‘. Invertible affine matrix mapping medium coordinates into the containing object’s local coordinates.

`haze` Default ‘TRUE‘. Include clear-air atmospheric haze inside this medium when enabled by [sky_light()], subject to ‘haze_density_threshold‘. Set ‘FALSE‘ to skip its atmospheric in-scattering and extinction throughout the entire boundary, including empty cells, regardless of the threshold. The medium’s own scattering, absorption, and emission remain active. In nested media the innermost medium’s setting applies; ‘sky_light(haze_in_volumes = FALSE)‘ overrides all media. This is an approximation and may affect thin clouds, edges, or low-altitude haze.

`haze_density_threshold`
Default ‘NULL‘. With ‘haze = TRUE‘, omit haze only where interpolated density times ‘density_scale‘ is at least this positive number. Lower-density regions and empty space retain haze. A homogeneous medium has density one before scaling. ‘NULL‘ includes haze throughout the boundary. Ignored with ‘haze = FALSE‘ or global haze exclusion. The threshold measures density, not optical depth or scattering strength; choose which media to tag accordingly. Crossings follow the trilinear field, including between scattering events, and transform with the medium.

Value

A reusable ‘ray_medium‘ description. Files are loaded during scene construction.

`obj_model` *‘obj‘ File Object*

Description

Load an obj file via a filepath. Currently only supports the diffuse texture with the ‘texture‘ argument. Note: light importance sampling currently not supported for this shape.

Usage

```
obj_model(
  filename,
  x = 0,
  y = 0,
  z = 0,
  scale_obj = 1,
  load_material = TRUE,
  load_textures = TRUE,
  load_normals = TRUE,
  vertex_colors = FALSE,
  calculate_consistent_normals = TRUE,
  subdivision_levels = 1,
  displacement_texture = "",
  displacement_intensity = 1,
  displacement_vector = FALSE,
  recalculate_normals = FALSE,
  importance_sample_lights = TRUE,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

filename	Filename and path to the 'obj' file. Can also be a 'txt' file, if it's in the correct 'obj' internally.
x	Default '0'. x-coordinate to offset the model.
y	Default '0'. y-coordinate to offset the model.
z	Default '0'. z-coordinate to offset the model.
scale_obj	Default '1'. Amount to scale the model. Use this to scale the object up or down on all axes, as it is more robust to numerical precision errors than the generic scale option.
load_material	Default 'TRUE'. Whether to load the obj file material (MTL file). If material for faces aren't specified, the default material will be used (specified by the user in 'material').
load_textures	Default 'TRUE'. If 'load_material = TRUE', whether to load textures in the MTL file (versus just using the colors specified for each material).
load_normals	Default 'TRUE'. Whether to load the vertex normals if they exist in the OBJ file.
vertex_colors	Default 'FALSE'. Set to 'TRUE' if the OBJ file has vertex colors to apply them to the model.
calculate_consistent_normals	Default 'TRUE'. Whether to calculate consistent vertex normals to prevent energy loss at edges.

`subdivision_levels`
: Default '1'. Number of Loop subdivisions to be applied to the mesh.

`displacement_texture`
: Default '""'. File path to the displacement texture. This texture is used to displace the vertices of the mesh based on the texture's pixel values.

`displacement_intensity`
: Default '1'. Intensity of the displacement effect. Higher values result in greater displacement.

`displacement_vector`
: Default 'FALSE'. Whether to use vector displacement. If 'TRUE', the displacement texture is interpreted as providing a 3D displacement vector. Otherwise, the texture is interpreted as providing a scalar displacement.

`recalculate_normals`
: Default 'FALSE'. Whether to recalculate vertex normals based on the connecting face orientations. This can be used to compute normals for meshes lacking them or to calculate new normals after a displacement map has been applied to the mesh.

`importance_sample_lights`
: Default 'TRUE'. Whether to importance sample lights specified in the OBJ material (objects with a non-zero Ke MTL material).

`material`
: Default `diffuse`.The material, called from one of the material functions `diffuse`, `metal`, or `dielectric`.

`angle`
: Default 'c(0, 0, 0)'. Angle of rotation around the x, y, and z axes, applied in the order specified in 'order_rotation'.

`order_rotation`
: Default 'c(1, 2, 3)'. The order to apply the rotations, referring to "x", "y", and "z".

`flipped`
: Default 'FALSE'. Whether to flip the normals.

`scale`
: Default 'c(1, 1, 1)'. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the obj model in the scene.

Examples

```
#Load the included example R object file, by calling the r_obj() function. This
#returns the local file path to the `r.txt` obj file. The file extension is "txt"
#due to package constraints, but the file contents are identical and it does not
#affect the function.

#Load the basic 3D R logo with the included materials
generate_ground(material = diffuse(checkercolor = "grey50")) |>
  add_object(obj_model(y = 0.2, filename = rayrender::r_obj(),
                       scale_obj=3)) |>
   add_object(sphere(z = -20, x = 20, y = 20, radius = 10,
                     material = light(intensity = 10))) |>
```

```
  render_scene(parallel = TRUE, samples = 16, aperture = 0.05,
               sample_method="sobol_blue",
               fov = 20, lookfrom = c(0, 2, -10))

# Smooth a mesh by setting the number of subdivision levels
generate_ground(material = diffuse(checkercolor = "grey50")) |>
  add_object(obj_model(y = 0.2, filename = rayrender::r_obj(),
                       scale_obj=3, subdivision_levels = 3)) |>
   add_object(sphere(z = -20, x = 20, y = 20, radius = 10,
                     material = light(intensity = 10))) |>
  render_scene(parallel = TRUE, samples = 16, aperture = 0.05,
               sample_method="sobol_blue",
               fov = 20, lookfrom = c(0, 2, -10))

#Override the materials for each object
generate_ground(material = diffuse(checkercolor = "grey50")) |>
  add_object(obj_model(y = 1.4, filename = rayrender::r_obj(), load_material = FALSE,
                       scale_obj = 1.8, angle=c(10,0,0),
                       material = microfacet(color = "gold", roughness = 0.2))) |>
 add_object(obj_model(x = 0.9, y = 0, filename = rayrender::r_obj(), load_material = FALSE,
                       scale_obj = 1.8, angle=c(0,20,0),
                       material = diffuse(color = "dodgerblue"))) |>
 add_object(obj_model(x = -0.9, y = 0, filename = rayrender::r_obj() , load_material = FALSE,
                       scale_obj = 1.8, angle=c(0,-40,0),
                       material = dielectric(attenuation = c(1,0.3,1), priority = 1,
                                             attenuation_intensity = 20))) |>
  add_object(sphere(z = -20, x = 20, y = 20, radius = 10,
                    material = light(intensity = 10))) |>
  render_scene(parallel = TRUE, samples = 16, aperture = 0.05,
               sample_method="sobol_blue", lookat=c(0,0.5,0),
               fov = 22, lookfrom = c(0, 2, -10))
```

---

path *Path Object*

Description

Either a closed or open path made up of bezier curves that go through the specified points (with continuous first and second derivatives), or straight line segments.

Usage

```
path(
  points,
  x = 0,
  y = 0,
  z = 0,
  closed = FALSE,
  closed_smooth = TRUE,
```

```
  straight = FALSE,
  precomputed_control_points = FALSE,
  width = 0.1,
  width_end = NA,
  u_min = 0,
  u_max = 1,
  type = "cylinder",
  normal = c(0, 0, -1),
  normal_end = NA,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

points	Either a list of length-3 numeric vectors or 3-column matrix/data.frame specifying the x/y/z points that the path should go through.
x	Default '0'. x-coordinate offset for the path.
y	Default '0'. y-coordinate offset for the path.
z	Default '0'. z-coordinate offset for the path.
closed	Default 'FALSE'. If 'TRUE', the path will be closed by smoothly connecting the first and last points.
closed_smooth	Default 'TRUE'. If 'closed = TRUE', this will ensure C2 (second derivative) continuity between the ends. If 'closed = FALSE', the curve will only have C1 (first derivative) continuity between the ends.
straight	Default 'FALSE'. If 'TRUE', straight lines will be used to connect the points instead of bezier curves.
precomputed_control_points	Default 'FALSE'. If 'TRUE', 'points' argument will expect a list of control points calculated with the internal rayrender function 'rayrender:::calculate_control_points()'.
width	Default '0.1'. Curve width.
width_end	Default 'NA'. Width at end of path. Same as 'width', unless specified.
u_min	Default '0'. Minimum parametric coordinate for the path.
u_max	Default '1'. Maximum parametric coordinate for the path.
type	Default 'cylinder'. Other options are 'flat' and 'ribbon'.
normal	Default 'c(0,0,-1)'. Orientation surface normal for the start of ribbon curves.
normal_end	Default 'NA'. Orientation surface normal for the start of ribbon curves. If not specified, same as 'normal'.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default 'c(0, 0, 0)'. Angle of rotation around the x, y, and z axes, applied in the order specified in 'order_rotation'.

order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the cube in the scene.

Examples

```
#Generate a wavy line, showing the line goes through the specified points:
wave = list(c(-2,1,0),c(-1,-1,0),c(0,1,0),c(1,-1,0),c(2,1,0))
point_mat = glossy(color="green")
generate_studio(depth=-1.5) |>
  add_object(path(points = wave,material=glossy(color="red"))) |>
  add_object(sphere(x=-2,y=1,radius=0.1,material=point_mat)) |>
  add_object(sphere(x=-1,y=-1,radius=0.1,material=point_mat)) |>
  add_object(sphere(x=0,y=1,radius=0.1,material=point_mat)) |>
  add_object(sphere(x=1,y=-1,radius=0.1,material=point_mat)) |>
  add_object(sphere(x=2,y=1,radius=0.1,material=point_mat)) |>
  add_object(sphere(z=-5,x=5,y=5,radius=2,material=light(intensity=15))) |>
  render_scene(samples=16, clamp_value=10,fov=30)
#Here we use straight lines by setting `straight = TRUE`:
generate_studio(depth=-1.5) |>
  add_object(path(points = wave,straight = TRUE, material=glossy(color="red"))) |>
  add_object(sphere(z=-5,x=5,y=5,radius=2,material=light(intensity=15))) |>
  render_scene(samples=16, clamp_value=10,fov=30)
#We can also pass a matrix of values, specifying the x/y/z coordinates. Here,
#we'll create a random curve:
set.seed(21)
random_mat = matrix(runif(3*9)*2-1, ncol=3)
generate_studio(depth=-1.5) |>
  add_object(path(points=random_mat, material=glossy(color="red"))) |>
  add_object(sphere(y=5,radius=1,material=light(intensity=30))) |>
  render_scene(samples=16, clamp_value=10)
#We can ensure the curve is closed by setting `closed = TRUE`
generate_studio(depth=-1.5) |>
  add_object(path(points=random_mat, closed = TRUE, material=glossy(color="red"))) |>
  add_object(sphere(y=5,radius=1,material=light(intensity=30))) |>
  render_scene(samples=16, clamp_value=10)
#Finally, let's render a pretzel to show how you can render just a subset of the curve:
pretzel = list(c(-0.8,-0.5,0.1),c(0,-0.2,-0.1),c(0,0.3,0.1),c(-0.5,0.5,0.1), c(-0.6,-0.5,-0.1),
               c(0,-0.8,-0.1),
          c(0.6,-0.5,-0.1),c(0.5,0.5,-0.1), c(0,0.3,-0.1),c(-0,-0.2,0.1), c(0.8,-0.5,0.1))

#Render the full pretzel:
generate_studio(depth = -1.1) |>
  add_object(path(pretzel, width=0.17,  material = glossy(color="#db5b00"))) |>
 add_object(sphere(y=5,x=2,z=-4,material=light(intensity=20,spotlight_focus = c(0,0,0)))) |>
```

```
  render_scene(samples=16, clamp_value=10)
#Here, we'll render only the first third of the pretzel by setting `u_max = 0.33`
generate_studio(depth = -1.1) |>
  add_object(path(pretzel, width=0.17, u_max=0.33, material = glossy(color="#db5b00"))) |>
  add_object(sphere(y=5,x=2,z=-4,material=light(intensity=20,spotlight_focus = c(0,0,0)))) |>
  render_scene(samples=16, clamp_value=10)
#Here's the last third, by setting `u_min = 0.66`
generate_studio(depth = -1.1) |>
  add_object(path(pretzel, width=0.17, u_min=0.66, material = glossy(color="#db5b00"))) |>
  add_object(sphere(y=5,x=2,z=-4,material=light(intensity=20,spotlight_focus = c(0,0,0)))) |>
  render_scene(samples=16, clamp_value=10)
#Here's the full pretzel, decomposed into thirds using the u_min and u_max coordinates
generate_studio(depth = -1.1) |>
  add_object(path(pretzel, width=0.17, u_max=0.33, x = -0.8, y =0.6,
                  material = glossy(color="#db5b00"))) |>
  add_object(path(pretzel, width=0.17, u_min=0.66, x = 0.8, y =0.6,
                  material = glossy(color="#db5b00"))) |>
  add_object(path(pretzel, width=0.17, u_min=0.33, u_max=0.66, x=0,
                  material = glossy(color="#db5b00"))) |>
  add_object(sphere(y=5,x=2,z=-4,material=light(intensity=20,spotlight_focus = c(0,0,0)))) |>
  render_scene(samples=16, clamp_value=10, lookfrom=c(0,3,-10))
```

---

pig *Pig Object*

Description

Builds an earless, low-poly mesh pig from compact data included in rayrender. The skiing outfit includes goggles, a scarf and skis; the spider variant has eight legs, four eyes and procedural copper-colored hair curves.

Usage

```
pig(
  x = 0,
  y = 0,
  z = 0,
  emotion = "neutral",
  spider = FALSE,
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  scale = c(1, 1, 1),
  diffuse_sigma = 0,
  ski = FALSE,
  hair_count = 12000,
  hair_length = 1,
  hair_seed = 8,
  hair_material = NULL
)
```

Arguments

x	Default 0. x-coordinate offset of the pig.
y	Default 0. y-coordinate offset of the pig.
z	Default 0. z-coordinate offset of the pig.
emotion	Default "neutral". Facial expression: neutral, skeptical, worried, angry, surprised, or excited. cheerful is an alias for neutral. When omitted with ski = TRUE, the expression is excited.
spider	Default FALSE. Generate an eight-legged, four-eyed spider pig.
angle	Default c(0, 0, 0). Rotation in degrees about the x, y and z axes.
order_rotation	Default c(1, 2, 3). Order of the rotation axes.
scale	Default c(1, 1, 1). Uniform scalar or x/y/z scale factors.
diffuse_sigma	Default 0. Oren-Nayar roughness for diffuse surfaces, including the skin. Does not affect glossy eyes, lenses or hair.
ski	Default FALSE. Add the ski outfit and skiing pose. If both ski and spider are TRUE, warns "spiders can't ski" and generates the spider without ski gear.
hair_count	Default 12000. Number of procedural spider hair curves. Set to zero for a bare spider mesh. Ignored for a regular or skiing pig.
hair_length	Default 1. Multiplier for spider hair lengths.
hair_seed	Default 8. Reproducible spider hair seed; preserves the caller's random-number state.
hair_material	Default NULL. Use the built-in rust/copper hair palette. Supply a single hair() material to override the entire coat.

Details

Geometry, expression heads and the ski pose are assembled in memory; no OBJ paths, downloads, Blender or rgl installation are required. The character faces +X, uses Y up, and retains the rotation/scale pivot at c(0, 1, 0). Feet or skis rest approximately at y - 0.6 before rotation and scaling. The mask and goggles can obscure eyebrows in skiing poses. Hair uses tapered bezier_curve() objects with hair() materials. A finite bounce limit, such as max_depth = 12, is useful for dense fur.

Value

A scene tibble containing mesh objects and, for a hairy spider, curves.

Examples

```
generate_ground(depth = -0.6) |>
  add_object(pig()) |>
  render_scene(lookfrom = c(8, 5, 8), lookat = c(0, 1, 0), samples = 64)

generate_ground(depth = -0.6, material = glossy(color="white")) |>
  add_object(pig(ski = TRUE)) |>
  render_scene(lookfrom = c(8, 5, 8),
```

```
                lookat = c(0, 1, 0),
                fov=40,samples = 64)

generate_ground(depth = -0.6) |>
  add_object(pig(spider = TRUE)) |>
  render_scene(lookfrom = c(10, 7, 8), lookat = c(0, 1, 0),
                samples = 64, max_depth = 12)
```

`ply_model` *'ply' File Object*

Description

Load an PLY file via a filepath. Note: light importance sampling currently not supported for this shape.

Usage

```
ply_model(
  filename,
  x = 0,
  y = 0,
  z = 0,
  scale_ply = 1,
  subdivision_levels = 1,
  recalculate_normals = FALSE,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

`filename`	Filename and path to the 'ply' file. Can also be a 'txt' file, if it's in the correct 'ply' internally.
`x`	Default '0'. x-coordinate to offset the model.
`y`	Default '0'. y-coordinate to offset the model.
`z`	Default '0'. z-coordinate to offset the model.
`scale_ply`	Default '1'. Amount to scale the model. Use this to scale the object up or down on all axes, as it is more robust to numerical precision errors than the generic scale option.
`subdivision_levels`	Default '1'. Number of Loop subdivisions to be applied to the mesh.

recalculate_normals
: Default 'FALSE'. Whether to recalculate vertex normals based on the connecting face orientations. This can be used to compute normals for meshes lacking them or to calculate new normals after a displacement map has been applied to the mesh.

material
: Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.

angle
: Default 'c(0, 0, 0)'. Angle of rotation around the x, y, and z axes, applied in the order specified in 'order_rotation'.

order_rotation
: Default 'c(1, 2, 3)'. The order to apply the rotations, referring to "x", "y", and "z".

flipped
: Default 'FALSE'. Whether to flip the normals.

scale
: Default 'c(1, 1, 1)'. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the obj model in the scene.

Examples

```
#See the documentation for `obj_model()`--no example PLY models are included with this package,
#but the process of loading a model is the same (without support for vertex colors).
```

---

preview_camera *Preview Camera*

Description

Previews a scene-attached camera without writing image files.

Usage

```
preview_camera(
  scene,
  camera = NULL,
  width = 800,
  height = 800,
  fps = 24,
  samples = 1,
  ...
)
```

Arguments

scene	Scene containing cameras.
camera	Default ‘NULL‘. Camera name or ‘ray_camera‘ object to preview.
width	Default ‘800‘. Preview width, in pixels.
height	Default ‘800‘. Preview height, in pixels.
fps	Default ‘24‘. Intended preview frame rate for animated cameras.
samples	Default ‘1‘. Number of samples per pixel for the preview.
...	Additional arguments passed to ‘render_scene()‘.

Value

Invisibly returns the rendered preview result.

Examples

```
#Zooming over the R logo
motion = generate_camera_motion(
  positions = list(c(0, 1, -10), c(0, 1, -2), c(0, 1, 10)),
  lookats = list(c(0, 0, 0), c(0, 0.5, 0), c(0, 0, 0)),
  fovs = c(60, 45, 90),
  frames = 24,
  type = "linear",
  damp_motion = TRUE,
  closed = TRUE
)

scene = generate_ground(material=diffuse(color="grey20")) |>
  add_object(obj_model(r_obj())) |>
  add_object(sphere(y=10,x=-10,z=-5,material=light(intensity=100))) |>
  add_camera(camera(name = "flythrough", motion = motion))

preview_camera(scene, camera = "flythrough", width = 400, height = 400)
```

---

r_obj *R 3D Model*

Description

3D obj model of R logo (created from the R SVG logo with the ‘raybevel‘ package), to be used with ‘obj_model()‘

Usage

```
r_obj(simple_r = FALSE)
```

Arguments

simple_r	Default 'FALSE'. If 'TRUE', this will return a 3D R (instead of the R logo).

Value

File location of the 3d_r_logo.obj file (saved with a .txt extension)

Examples

```
#Load and render the included example R object file.
generate_ground(material = diffuse(noise = TRUE, noisecolor = "grey20")) |>
  add_object(sphere(x = 2, y = 3, z = -2, radius = 1,
                    material = light(intensity = 10))) |>
 add_object(obj_model(r_obj(), scale=2.5, angle = c(0,0,0), material = diffuse(color="red"))) |>
  render_scene(parallel=TRUE, lookfrom = c(0, 1, -10), clamp_value = 5, samples = 200)
```

---

raymesh_model *'raymesh' model*

Description

Load an 'raymesh' object, as specified in the 'rayvertex' package.

Usage

```
raymesh_model(
  mesh,
  x = 0,
  y = 0,
  z = 0,
  flip_transmittance = TRUE,
  verbose = FALSE,
  importance_sample_lights = FALSE,
  calculate_consistent_normals = TRUE,
  subdivision_levels = 1,
  displacement_texture = "",
  displacement_intensity = 1,
  displacement_vector = FALSE,
  recalculate_normals = FALSE,
  override_material = NULL,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1),
  validate_mesh = TRUE
)
```

Arguments

mesh	A ‘raymesh‘ object. Pulls the vertex, index, texture coordinates, normals, and material information.
x	Default ‘0‘. x-coordinate to offset the model.
y	Default ‘0‘. y-coordinate to offset the model.
z	Default ‘0‘. z-coordinate to offset the model.
flip_transmittance	Default ‘TRUE‘. Flips ‘(1-t)‘ the transmittance values to match the way the colors would be interpreted in a rasterizer (where it specifies the transmitted color). Turn off to specify the attenuation values directly.
verbose	Default ‘FALSE‘. If ‘TRUE‘, prints information about the mesh to the console.
importance_sample_lights	Default ‘TRUE‘. Whether to importance sample lights specified in the OBJ material (objects with a non-zero Ke MTL material).
calculate_consistent_normals	Default ‘TRUE‘. Whether to calculate consistent vertex normals to prevent energy loss at edges.
subdivision_levels	Default ‘1‘. Number of Loop subdivisions to be applied to the mesh.
displacement_texture	Default ‘""‘. File path to the displacement texture. This texture is used to displace the vertices of the mesh based on the texture's pixel values.
displacement_intensity	Default ‘1‘. Intensity of the displacement effect. Higher values result in greater displacement.
displacement_vector	Default ‘FALSE‘. Whether to use vector displacement. If ‘TRUE‘, the displacement texture is interpreted as providing a 3D displacement vector. Otherwise, the texture is interpreted as providing a scalar displacement.
recalculate_normals	Default ‘FALSE‘. Whether to recalculate vertex normals based on the connecting face orientations. This can be used to compute normals for meshes lacking them or to calculate new normals after a displacement map has been applied to the mesh.
override_material	Default ‘NULL‘, which overrides the material specified in the ‘raymesh‘ object only when the user passes a value to ‘material‘. If ‘TRUE‘, overrides the material specified in the ‘raymesh‘ object with the one specified in ‘material‘.
material	Default diffuse, but ignored unless ‘override_material = TRUE‘. The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".

flipped	Default 'FALSE'. Whether to flip the normals.
scale	Default 'c(1, 1, 1)'. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.
validate_mesh	Default 'TRUE'. Validates the 'raymesh' object using 'rayvertex::validate_mesh()' before parsing to ensure correct parsing. Set to 'FALSE' to speed up scene construction if 'raymesh_model()' is taking a long time (Note: this does not affect rendering time).

Value

Single row of a tibble describing the raymesh model in the scene.

Examples

```
#Render a simple raymesh object
library(rayvertex)
raymesh_model(sphere_mesh(position = c(-1, 0, 0),
              material = material_list(transmittance = "red"))) |>
  add_object(generate_ground(material = diffuse(checkercolor="grey20"))) |>
  render_scene(fov = 30, samples=16, sample_method="sobol_blue")

# We create a complex rayvertex mesh, using the `rayvertex::add_shape` function which
# creates a new `raymesh` object out of individual `raymesh` objects
rm_scene = sphere_mesh(position = c(-1, 0, 0),
            material = material_list(transmittance = "red")) |>
    add_shape(sphere_mesh(position = c(1, 0, 0),
            material = material_list(transmittance = "green", ior = 1.5)))

# Pass the single raymesh object to `raymesh_model()`
# `raymesh_model()`
raymesh_model(rm_scene) |>
  add_object(generate_ground(material = diffuse(checkercolor="grey20"))) |>
  render_scene(fov = 30, samples=16, sample_method="sobol_blue")

# Set `flip_transmittance = FALSE` argument to specify attenuation coefficients directly
# (as specified in the `dielectric()` material). We change the material's numerical attenuation
# constants using `rayvertex::change_material`
rm_scene_new= change_material(rm_scene, transmittance = c(1,2,0.3), id = 1) |>
  change_material(transmittance = c(3,1,2), id = 2)
raymesh_model(rm_scene_new, flip_transmittance = FALSE) |>
  add_object(generate_ground(material = diffuse(checkercolor="grey20"))) |>
  render_scene(fov = 30, samples=16, sample_method="sobol_blue")

# Override the material specified in the `raymesh` object and render the scene
raymesh_model(rm_scene,
           material = dielectric(attenuation = "dodgerblue2", attenuation_intensity = 4),
  override_material = TRUE) |>
  add_object(generate_ground(material = diffuse(checkercolor="grey20"))) |>
  render_scene(fov = 30, samples=16, sample_method="sobol_blue")
```

```
# Adjusting the scale, position, and rotation parameters of the `raymesh` model
raymesh_model(rm_scene,
              x = 0, y = 0.5, z = -1, angle = c(0, 0, 20)) |>
  add_object(generate_ground(material = diffuse(checkercolor="grey20"))) |>
  render_scene(fov = 30,lookat=c(0,0.5,0), samples=16, sample_method="sobol_blue")
```

`remove_camera` *Remove Camera*

Description

Removes a camera from a 'ray_scene'.

Usage

```
remove_camera(scene, camera)
```

Arguments

`scene`	Scene containing cameras.
`camera`	Camera name.

Value

A modified 'ray_scene'.

Examples

```
scene = generate_ground(material=diffuse(color="grey20")) |>
  add_camera(camera(name = "wide"), active = FALSE) |>
  add_camera(camera(name = "detail"), active = TRUE)

scene = remove_camera(scene, "detail")
list_cameras(scene)
```

remove_infinite_light *Remove an Infinite Light*

Description

Remove an Infinite Light

Usage

```
remove_infinite_light(scene, name)
```

Arguments

scene	Scene to modify.
name	Name of the infinite light to remove.

Value

A modified scene.

render_animation *Render Animation*

Description

Takes the scene description and renders an image, either to the device or to a filename.

Usage

```
render_animation(
  scene,
  camera_motion = NULL,
  start_frame = 1,
  end_frame = NA,
  width = 400,
  height = 400,
  preview = interactive(),
  denoise = TRUE,
  camera_description_file = NA,
  camera_scale = 1,
  iso = 100,
  film_size = 22,
  samples = 100,
  min_variance = 0,
  min_adaptive_size = 8,
```

```
  sample_method = "sobol",
  ambient_occlusion = FALSE,
  keep_colors = FALSE,
  sample_dist = 10,
  max_depth = 50,
  roulette_active_depth = 10,
  ambient_light = FALSE,
  clamp_value = Inf,
  filename = NA,
  backgroundhigh = "#80b4ff",
  backgroundlow = "#ffffff",
  shutteropen = 0,
  shutterclose = 1,
  camera_motion_blur = FALSE,
  focal_distance = NULL,
  ortho_dimensions = c(1, 1),
  tonemap = "raw",
  bloom = TRUE,
  parallel = TRUE,
  bvh_type = "sah",
  environment_light = NULL,
  rotate_env = 0,
  intensity_env = 1,
  debug_channel = "none",
  plot_scene = TRUE,
  progress = interactive(),
  verbose = FALSE,
  transparent_background = FALSE,
  preview_light_direction = c(0, -1, 0),
  preview_exponent = 6,
  integrator_type = "rtiow",
  camera = NULL
)
```

Arguments

scene	Tibble of object locations and properties.
camera_motion	Default ‘NULL‘. Data frame of camera motion vectors, calculated with ‘generate_camera_motion()‘. If ‘NULL‘, the camera is resolved from the scene.
start_frame	Default ‘1‘. Frame to start the animation.
end_frame	Default ‘NA‘. By default, this is set to ‘nrow(camera_motion)‘, the full number of frames.
width	Default ‘400‘. Width of the render, in pixels.
height	Default ‘400‘. Height of the render, in pixels.
preview	Default ‘interactive()‘. Whether to display a realtime progressive preview of the render. Press ESC to cancel the render.

denoise Default ‘TRUE‘. Whether to de-noise the final image and preview images. Note, this requires the free Intel Open Image Denoise (OIDN) library be installed on your system. Pre-compiled binaries can be installed from ppenimagedenoise.org, as well as . Linking during rayrender installation is done by defining the environment variable OIDN_PATH (set it in the .Renviron file by calling ‘usethis::edit_r_environ()‘) to the top-level directory for OIDN (the directory containing the "lib", "bin", and "include" directories) and re-installing this package from source.

camera_description_file
Default ‘NA‘. Filename of a camera description file for rendering with a realistic camera. Several camera files are built-in: ‘"50mm"‘,‘"wide"‘,‘"fisheye"‘, and ‘"telephoto"‘.

camera_scale Default ‘1‘. Amount to scale the camera up or down in size. Use this rather than scaling a scene.

iso Default ‘100‘. Camera exposure.

film_size Default ‘22‘, in ‘mm‘ (scene units in ‘m‘. Size of the film if using a realistic camera, otherwise ignored.

samples Default ‘100‘. The maximum number of samples for each pixel. If this is a length-2 vector and the ‘sample_method‘ is ‘stratified‘, this will control the number of strata in each dimension. The total number of samples in this case will be the product of the two numbers.

min_variance Default ‘0‘. Minimum acceptable variance for a block of pixels for the adaptive sampler. Smaller numbers give higher quality images, at the expense of longer rendering times. If this is set to zero, the adaptive sampler will be turned off and the renderer will use the maximum number of samples everywhere.

min_adaptive_size
Default ‘8‘. Width of the minimum block size in the adaptive sampler.

sample_method Default ‘sobol‘. The type of sampling method used to generate random numbers. The other options are ‘random‘ (worst quality but simple), ‘stratified‘ (only implemented for completion), and ‘sobol_blue‘ (best option for sample counts below 256).

ambient_occlusion
Default ‘FALSE‘. If ‘TRUE‘, the animation will be rendered with the ambient occlusion renderer. This uses the background color specified in ‘backgroundhigh‘

keep_colors Default ‘FALSE‘. Whether to keep the diffuse material colors.

sample_dist Default ‘10‘. Sample distance if ‘debug_channel = "ao"‘.

max_depth Default ‘50‘. Maximum number of bounces a ray can make in a scene.

roulette_active_depth
Default ‘10‘. Number of ray bounces until a ray can stop bouncing via Russian roulette.

ambient_light Default ‘FALSE‘, unless there are no emitting objects in the scene. If ‘TRUE‘, the background will be a gradient varying from ‘backgroundhigh‘ directly up (+y) to ‘backgroundlow‘ directly down (-y).

`clamp_value` Default ‘Inf‘. If a bright light or a reflective material is in the scene, occasionally there will be bright spots that will not go away even with a large number of samples. These can be removed (at the cost of slightly darkening the image) by setting this to a small number greater than 1.

`filename` Default ‘NA‘. If present, the renderer will write to the filename instead of the current device.

`backgroundhigh` Default ‘#ffffff‘. The "high" color in the background gradient. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.

`backgroundlow` Default ‘#ffffff‘. The "low" color in the background gradient. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.

`shutteropen` Default ‘0‘. Time at which the shutter is open. Only affects moving objects.

`shutterclose` Default ‘1‘. Time at which the shutter is open. Only affects moving objects.

`camera_motion_blur`
Default ‘FALSE‘. Whether to blur camera movement over the shutter interval.

`focal_distance` Default ‘NULL‘, automatically set to the ‘lookfrom-lookat‘ distance unless otherwise specified.

`ortho_dimensions`
Default ‘c(1,1)‘. Width and height of the orthographic camera. Will only be used if ‘fov = 0‘.

`tonemap` Default ‘raw‘, no tonemapping. See ‘rayimage::render_tonemap()‘ for options.

`bloom` Default ‘TRUE‘. Set to ‘FALSE‘ to get the raw, pathtraced image. Otherwise, this performs a convolution of the HDR image of the scene with a sharp, long-tailed exponential kernel, which does not visibly affect dimly pixels, but does result in emitters light slightly bleeding into adjacent pixels. This provides an antialiasing effect for lights, even when tonemapping the image. Pass in a matrix to specify the convolution kernel manually, or a positive number to control the intensity of the bloom (higher number = more bloom).

`parallel` Default ‘FALSE‘. If ‘TRUE‘, it will use all available cores to render the image (or the number specified in ‘options("cores")‘ if that option is not ‘NULL‘).

`bvh_type` Default ‘"sah"‘, "surface area heuristic". Method of building the bounding volume hierarchy structure used when rendering. Other option is "equal", which splits tree into groups of equal size.

`environment_light`
Default ‘NULL‘. An image to be used for the background for rays that escape the scene. Supports EXR, HDR, PNG, and JPEG images. Scene lights added with `add_infinite_light()` are included in every frame, together with this image.

`rotate_env` Default ‘0‘. The number of degrees to rotate all infinite lights around the scene, in addition to their individual rotations.

`intensity_env` Default ‘1‘. The amount to increase the intensity of the environment lighting. Useful if using a LDR (JPEG or PNG) image as an environment map.

debug_channel
: Default ‘none‘. If ‘depth‘, function will return a depth map of rays into the scene instead of an image. If ‘normals‘, function will return an image of scene normals, mapped from 0 to 1. If ‘uv‘, function will return an image of the uv coords. If ‘variance‘, function will return an image showing the number of samples needed to take for each block to converge. If ‘dpdu‘ or ‘dpdv‘, function will return an image showing the differential ‘u‘ and ‘u‘ coordinates. If ‘color‘, function will return the raw albedo values (with white for ‘metal‘ and ‘dielectric‘ materials). If ‘preview‘, an image rendered with ‘render_preview()‘ will be returned. Can set to ‘ao‘ to render an animation with the ambient occlusion renderer.

plot_scene
: Default ‘TRUE‘. Whether to plot the rendered scene. If ‘preview = TRUE‘ and ‘filename‘ is omitted, this defaults to ‘FALSE‘ so the animation runs through the preview window.

progress
: Default ‘TRUE‘ if interactive session, ‘FALSE‘ otherwise.

verbose
: Default ‘FALSE‘. Prints information and timing information about scene construction and raytracing progress.

transparent_background
: Default ‘FALSE‘. If ‘TRUE‘, any initial camera rays that escape the scene will be marked as transparent in the final image. If for a pixel some rays escape and others hit a surface, those pixels will be partially transparent.

preview_light_direction
: Default ‘c(0,-1,0)‘. Vector specifying the orientation for the global light using for phong shading.

preview_exponent
: Default ‘6‘. Phong exponent.

integrator_type
: Default ‘"rtiow"‘ (the algorithm specified in the book "Raytracing in One Weekend", a basic form of path guiding). Other options include ‘"nee"‘ (Next Event Estimation, with direct light sampling) and ‘"basic"‘ (basic pathtracing, for high sample reference renders and debugging only). `nee` is selected automatically for scenes containing `sky_light()` or attached media (including clouds and media inside instances), overriding `rtiow` or `basic`.

camera
: Default ‘NULL‘. Scene-attached camera name, ‘"all"‘, or a ‘ray_camera‘ object created with ‘camera()‘.

Value

Raytraced plot to current device, or an image saved to a file.

Examples

```
#Create and animate flying through a scene on a simulated roller coaster
set.seed(3)
elliplist = list()
ellip_colors = rainbow(8)
for(i in 1:1200) {
  elliplist[[i]] = ellipsoid(x=10*runif(1)-5,y=10*runif(1)-5,z=10*runif(1)-5,
```

```
                            angle = 360*runif(3), a=0.1,b=0.05,c=0.1,
                            material=glossy(color=sample(ellip_colors,1)))
}
ellip_scene = do.call(rbind, elliplist)

camera_pos = list(c(0,1,15),c(5,-5,5),c(-5,5,-5),c(0,1,-15))

#Plot the camera path and render from above using the path object:
generate_ground(material=diffuse(checkercolor="grey20"),depth=-10) |>
  add_object(ellip_scene) |>
  add_object(sphere(y=50,radius=10,material=light(intensity=30))) |>
  add_object(path(camera_pos, material=diffuse(color="red"))) |>
  render_scene(lookfrom=c(0,20,0), width=800,height=800,samples=32,
               camera_up = c(0,0,1),
               fov=80)
#Side view
generate_ground(material=diffuse(checkercolor="grey20"),depth=-10) |>
  add_object(ellip_scene) |>
  add_object(sphere(y=50,radius=10,material=light(intensity=30))) |>
  add_object(path(camera_pos, material=diffuse(color="red"))) |>
  render_scene(lookfrom=c(20,0,0),width=800,height=800,samples=32,
                 fov=80)
#View from the start
generate_ground(material=diffuse(checkercolor="grey20"),depth=-10) |>
  add_object(ellip_scene) |>
  add_object(sphere(y=50,radius=10,material=light(intensity=30))) |>
  add_object(path(camera_pos, material=diffuse(color="red"))) |>
  render_scene(lookfrom=c(0,1.5,16),width=800,height=800,samples=32,
                 fov=80)
#Generate Camera movement, setting the lookat position to be same as camera position, but offset
#slightly in front. We'll render 12 frames, but you'd likely want more in a real animation.

camera_motion =  generate_camera_motion(positions = camera_pos, lookats = camera_pos,
                                        offset_lookat = 1, fovs=80, frames=12,
                                        type="bezier")

#This returns a data frame of individual camera positions, interpolated by cubic bezier curves.
camera_motion

#Pass NA filename to plot to the device. We'll keep the path and offset it slightly to see
#where we're going. This results in a "roller coaster" effect.
generate_ground(material=diffuse(checkercolor="grey20"),depth=-10) |>
  add_object(ellip_scene) |>
  add_object(sphere(y=50,radius=10,material=light(intensity=30))) |>
  add_object(obj_model(r_obj(),x=10,y=-5,z=10,scale=7, angle=c(45,135,0),
                       material=dielectric(attenuation=c(1,1,0.3)))) |>
  add_object(pig(x=-7,y=10,z=-5,scale=1,angle=c(0,-45,80),emotion="angry")) |>
 add_object(pig(x=0,y=-0.25,z=-15,scale=1,angle=c(0,225,-22), order_rotation = c(3,2,1),
                 emotion="angry", spider=TRUE)) |>
  add_object(path(camera_pos, y=-0.2,material=diffuse(color="red"))) |>
  render_animation(camera_motion = camera_motion, samples=16,
                   sample_method="sobol_blue",
                   clamp_value=10, width=400, height=400)
```

render_ao *Render Ambient Occlusion*

Description

Takes the scene description and renders an image using ambient occlusion, either to the device or to a filename.

Usage

```
render_ao(
  scene,
  width = 400,
  height = 400,
  fov = 20,
  sample_dist = 10,
  keep_colors = FALSE,
  samples = 100,
  camera_description_file = NA,
  camera_scale = 1,
  iso = 100,
  film_size = 22,
  min_variance = 0,
  min_adaptive_size = 8,
  sample_method = "sobol",
  background_color = "white",
  lookfrom = c(0, 1, 10),
  lookat = c(0, 0, 0),
  camera_up = c(0, 1, 0),
  aperture = 0.1,
  clamp_value = Inf,
  filename = NA,
  shutteropen = 0,
  shutterclose = 1,
  focal_distance = NULL,
  ortho_dimensions = c(1, 1),
  parallel = TRUE,
  bvh_type = "sah",
  progress = interactive(),
  verbose = FALSE
)
```

Arguments

scene	Tibble of object locations and properties.

`width` Default ‘400‘. Width of the render, in pixels.

`height` Default ‘400‘. Height of the render, in pixels.

`fov` Default ‘20‘. Field of view, in degrees. If this is ‘0‘, the camera will use an orthographic projection. The size of the plane used to create the orthographic projection is given in argument ‘ortho_dimensions‘. From ‘0‘ to ‘180‘, this uses a perspective projections. If this value is ‘360‘, a 360 degree environment image will be rendered.

`sample_dist` Default ‘10‘. Ambient occlusion sampling distance.

`keep_colors` Default ‘FALSE‘. Whether to keep the diffuse material colors.

`samples` Default ‘100‘. The maximum number of samples for each pixel. If this is a length-2 vector and the ‘sample_method‘ is ‘stratified‘, this will control the number of strata in each dimension. The total number of samples in this case will be the product of the two numbers.

`camera_description_file`
Default ‘NA‘. Filename of a camera description file for rendering with a realistic camera. Several camera files are built-in: ‘"50mm"‘,‘"wide"‘,‘"fisheye"‘, and ‘"telephoto"‘.

`camera_scale` Default ‘1‘. Amount to scale the camera up or down in size. Use this rather than scaling a scene.

`iso` Default ‘100‘. Camera exposure.

`film_size` Default ‘22‘, in ‘mm‘ (scene units in ‘m‘. Size of the film if using a realistic camera, otherwise ignored.

`min_variance` Default ‘0.00005‘. Minimum acceptable variance for a block of pixels for the adaptive sampler. Smaller numbers give higher quality images, at the expense of longer rendering times. If this is set to zero, the adaptive sampler will be turned off and the renderer will use the maximum number of samples everywhere.

`min_adaptive_size`
Default ‘8‘. Width of the minimum block size in the adaptive sampler.

`sample_method` Default ‘sobol‘. The type of sampling method used to generate random numbers. The other options are ‘random‘ (worst quality but fastest), ‘stratified‘ (only implemented for completion), ‘sobol_blue‘ (best option for sample counts below 256), and ‘sobol‘ (slowest but best quality, better than ‘sobol_blue‘ for sample counts greater than 256).

`background_color`
Default ‘"white"‘. Background color.

`lookfrom` Default ‘c(0,1,10)‘. Location of the camera.

`lookat` Default ‘c(0,0,0)‘. Location where the camera is pointed.

`camera_up` Default ‘c(0,1,0)‘. Vector indicating the "up" position of the camera.

`aperture` Default ‘0.1‘. Aperture of the camera. Smaller numbers will increase depth of field, causing less blurring in areas not in focus.

`clamp_value` Default ‘Inf‘. If a bright light or a reflective material is in the scene, occasionally there will be bright spots that will not go away even with a large number of samples. These can be removed (at the cost of slightly darkening the image) by setting this to a small number greater than 1.

filename	Default 'NA'. If present, the renderer will write to the filename instead of the current device.
shutteropen	Default '0'. Time at which the shutter is open. Only affects moving objects.
shutterclose	Default '1'. Time at which the shutter is open. Only affects moving objects.
focal_distance	Default 'NULL', automatically set to the 'lookfrom-lookat' distance unless otherwise specified.
ortho_dimensions	Default 'c(1,1)'. Width and height of the orthographic camera. Will only be used if 'fov = 0'.
parallel	Default 'FALSE'. If 'TRUE', it will use all available cores to render the image (or the number specified in 'options("cores")' if that option is not 'NULL').
bvh_type	Default '"sah"', "surface area heuristic". Method of building the bounding volume hierarchy structure used when rendering. Other option is "equal", which splits tree into groups of equal size.
progress	Default 'TRUE' if interactive session, 'FALSE' otherwise.
verbose	Default 'FALSE'. Prints information and timing information about scene construction and raytracing progress.

Value

Raytraced plot to current device, or an image saved to a file. Invisibly returns the array (containing either debug data or the RGB)

Examples

```
#Generate and render a regular scene and an ambient occlusion version of that scene
angles = seq(0,360,by=36)
xx = rev(c(rep(c(1,0.5),5),1) * sinpi(angles/180))
yy = rev(c(rep(c(1,0.5),5),1) * cospi(angles/180))
star_polygon = data.frame(x=xx,y=yy)
hollow_star = rbind(star_polygon,0.8*star_polygon)

generate_ground(material = diffuse(color="grey20", checkercolor = "grey50",sigma=90)) |>
 add_object(sphere(material=metal())) |>
 add_object(obj_model(r_obj(),y=-0.25,x=-1.8,scale=2,
                      angle=c(0,-45,0),material = diffuse(sigma=90))) |>
 add_object(pig(x=1.8,y=-1.2,scale=0.5,angle=c(0,90,0),diffuse_sigma = 90)) |>
 add_object(extruded_polygon(hollow_star,top=-0.5,bottom=-1, z=-2,
                             hole = nrow(star_polygon),
                             material=diffuse(color="red",sigma=90))) |>
 render_scene(parallel = TRUE,width=800,height=800,
              fov=70,clamp_value=10,samples=16, aperture=0.1,
              lookfrom=c(-0.9,1.2,-4.5),lookat=c(0,-1,0))

#Render the scene with ambient occlusion
generate_ground(material = diffuse(color="grey20", checkercolor = "grey50",sigma=90)) |>
 add_object(sphere(material=metal())) |>
 add_object(obj_model(r_obj(),y=-0.25,x=-1.8,scale=2,
                      angle=c(0,-45,0),material = diffuse(sigma=90))) |>
```

```
 add_object(pig(x=1.8,y=-1.2,scale=0.5,angle=c(0,90,0),diffuse_sigma = 90)) |>
 add_object(extruded_polygon(hollow_star,top=-0.5,bottom=-1, z=-2,
                             hole = nrow(star_polygon),
                             material=diffuse(color="red",sigma=90))) |>
 render_ao(parallel = TRUE,width=800,height=800, sample_dist=10,
           fov=70,samples=16, aperture=0.1,
           lookfrom=c(-0.9,1.2,-4.5),lookat=c(0,-1,0))
#Decrease the ray occlusion search distance
generate_ground(material = diffuse(color="grey20", checkercolor = "grey50",sigma=90)) |>
 add_object(sphere(material=metal())) |>
 add_object(obj_model(r_obj(),y=-0.25,x=-1.8,scale=2,
                      angle=c(0,-45,0),material = diffuse(sigma=90))) |>
 add_object(pig(x=1.8,y=-1.2,scale=0.5,angle=c(0,90,0),diffuse_sigma = 90)) |>
 add_object(extruded_polygon(hollow_star,top=-0.5,bottom=-1, z=-2,
                             hole = nrow(star_polygon),
                             material=diffuse(color="red",sigma=90))) |>
 render_ao(parallel = TRUE,width=800,height=800, sample_dist=1,
           fov=70,samples=16, aperture=0.1,
           lookfrom=c(-0.9,1.2,-4.5),lookat=c(0,-1,0))
#Turn on colors
generate_ground(material = diffuse(color="grey20", checkercolor = "grey50",sigma=90)) |>
 add_object(sphere(material=metal())) |>
 add_object(obj_model(r_obj(), y=-0.25,x=-1.8,scale=2,
                      angle=c(0,-45,0),material = diffuse(sigma=90))) |>
 add_object(pig(x=1.8,y=-1.2,scale=0.5,angle=c(0,90,0),diffuse_sigma = 90)) |>
 add_object(extruded_polygon(hollow_star,top=-0.5,bottom=-1, z=-2,
                             hole = nrow(star_polygon),
                             material=diffuse(color="red",sigma=90))) |>
 render_ao(parallel = TRUE,width=800,height=800, sample_dist=1,
           fov=70,samples=16, aperture=0.1, keep_colors = TRUE,
           lookfrom=c(-0.9,1.2,-4.5),lookat=c(0,-1,0))
```

`render_preview` *Render Preview*

Description

Takes the scene description and renders an image, either to the device or to a filename.

Usage

```
render_preview(..., light_direction = c(0, -1, 0), exponent = 6)
```

Arguments

`...`	All arguments that would be passed to 'render_scene()'.
`light_direction`	Default 'c(0,-1,0)'. Vector specifying the orientation for the global light using for phong shading.
`exponent`	Default '6'. Phong exponent.

Value

Raytraced plot to current device, or an image saved to a file.

Examples

```
generate_ground(material=diffuse(color="darkgreen")) |>
  add_object(sphere(material=diffuse(checkercolor="red"))) |>
  render_preview()
#Change the light direction
generate_ground(material=diffuse(color="darkgreen")) |>
  add_object(sphere(material=diffuse(checkercolor="red"))) |>
  render_preview(light_direction = c(-1,-1,0))
#Change the Phong exponent
generate_ground(material=diffuse(color="darkgreen")) |>
  add_object(sphere(material=diffuse(checkercolor="red"))) |>
  render_preview(light_direction = c(-1,-1,0), exponent=100)
```

---

render_scene *Render Scene*

Description

Takes the scene description and renders an image, either to the device or to a filename. The user can also interactively fly around the 3D scene if they have X11 support on their system or are on Windows.

Usage

```
render_scene(
  scene,
  width = 400,
  height = 400,
  fov = 20,
  samples = 100,
  camera_description_file = NA,
  preview = interactive(),
  interactive = TRUE,
  deferred_render = FALSE,
  denoise = TRUE,
  camera_scale = 1,
  iso = 100,
  auto_exposure = FALSE,
  film_size = 22,
  min_variance = 0,
  min_adaptive_size = 8,
  sample_method = "sobol_blue",
  max_depth = NA,
```

```
  roulette_active_depth = 100,
  ambient_light = NULL,
  lookfrom = c(0, 1, -10),
  lookat = c(0, 0, 0),
  camera_up = c(0, 1, 0),
  aperture = 0.1,
  clamp_value = Inf,
  filename = NA,
  backgroundhigh = "#80b4ff",
  backgroundlow = "#ffffff",
  shutteropen = 0,
  shutterclose = 1,
  camera_motion_blur = FALSE,
  shutter_speed = NULL,
  focal_distance = NULL,
  ortho_dimensions = c(1, 1),
  tonemap = "raw",
  bloom = TRUE,
  parallel = TRUE,
  bvh_type = "sah",
  environment_light = NULL,
  rotate_env = 0,
  intensity_env = 1,
  transparent_background = FALSE,
  debug_channel = "none",
  plot_scene = TRUE,
  progress = interactive(),
  verbose = FALSE,
  print_debug_info = FALSE,
  new_page = TRUE,
  integrator_type = "rtiow",
  screen_text = NULL,
  screen_line = NULL,
  camera = NULL,
  start_frame = 1,
  end_frame = NA,
  mode = c("auto", "image", "animation", "preview")
)
```

Arguments

scene	Tibble of object locations and properties.
width	Default ‘400‘. Width of the render, in pixels.
height	Default ‘400‘. Height of the render, in pixels.
fov	Default ‘20‘. Field of view, in degrees. If this is ‘0‘, the camera will use an orthographic projection. The size of the plane used to create the orthographic projection is given in argument ‘ortho_dimensions‘. From ‘0‘ to ‘180‘, this uses

a perspective projections. If this value is '360', a 360 degree environment image will be rendered.

`samples` Default '100'. The maximum number of samples for each pixel. If this is a length-2 vector and the 'sample_method' is 'stratified', this will control the number of strata in each dimension. The total number of samples in this case will be the product of the two numbers.

`camera_description_file`
Default 'NA'. Filename of a camera description file for rendering with a realistic camera. Several camera files are built-in: '"50mm"','"wide"','"fisheye"', and '"telephoto"'.

`preview` Default 'interactive()'. Whether to display a real-time progressive preview of the render. Press ESC to cancel the render. If 'deferred_render = TRUE', the preview stays interactive until the final render is explicitly started.

`interactive` Default 'interactive()'. Whether the scene preview should be interactive. Camera movement orbits around the lookat point (unless the mode is switched to free flying), with the following control mapping: W = Forward, S = Backward, A = Left, D = Right, Q = Up, Z = Down, Shift-W/Shift-S = Pitch Camera Forward/Back, Shift-A/Shift-D = Roll Camera Left/Right, E = 2x Step Distance (max 128), C = 0.5x Step Distance, Up Key = Zoom In (decrease FOV), Down Key = Zoom Out (increase FOV), Left Key = Decrease Aperture, Right Key = Increase Aperture, 1 = Decrease Focal Distance, 2 = Increase Focal Distance, 3/4 = Rotate Environment Light, Right bracket/left bracket = Increase/Decrease Preview Exposure, Shift + right bracket/left bracket = Increase/Decrease shutter speed, Shift-Enter = Save Preview Snapshot, R = Reset Camera, Return = Toggle between deferred and final render if 'deferred_render = TRUE', TAB: Toggle Orbit Mode, Left Mouse Click: Set Look At and Focal Distance, Right Mouse Click: Set Look At. If the interactive preview window is wide enough, a status bar at the bottom shows the current camera, exposure, environment rotation, and keyframe state. K: Save Keyframe (at the conclusion of the render, this will create the 'ray_keyframes' data.frame in the global environment, which can be passed to 'generate_camera_motion()' to tween between those saved positions. L: Reset Camera to Last Keyframe (if set), Shift-L: Toggle keyframe path open/closed, < and >: Jump to previous/next keyframe, /: Delete current keyframe, M: Preview/cancel keyframe motion, F: Toggle Fast Travel Mode, B: Toggle Camera Motion Blur, H: Toggle Atmospheric Haze, Y: Toggle Altitude Queries (with 'sky_light()'). Haze requires altitude queries: enabling haze also enables altitude queries, and disabling altitude queries also disables haze. Each change restarts sampling.

Initial step size is 1/20th of the distance from 'lookat' to 'lookfrom'.

With `integrator_type = "nee"`, clicks select the first point where accumulated volume opacity reaches 15%, or the first ordinary surface if reached sooner. Picking integrates extinction deterministically with a fixed shutter sample and a centered lens sample. Thin or empty regions allow selection of surfaces behind them; invisible container faces are skipped. Clicking the background without reaching this opacity turns perspective and panoramic cameras toward that direction, preserving focal distance and orbit radius. Orthographic background clicks leave the view unchanged. Selected surface and volume points remain the

orbit center when rotating. Right clicking preserves the focal distance. Some options aren't available for all cameras. When using a realistic camera, the aperture and field of view cannot be changed from their initial settings.

`deferred_render`
: Default 'FALSE'. If 'TRUE' and interactive preview is enabled, rayrender will keep updating the progressive preview until Return is pressed. Pressing Return toggles the full render in the same window; pressing Return again returns to deferred mode.

`denoise`
: Default 'TRUE'. Whether to de-noise the final image and preview images. Note, this requires the free Intel Open Image Denoise (OIDN) library be installed on your system. Pre-compiled binaries can be installed from ppenimagedenoise.org, as well as . Linking during rayrender installation is done by defining the environment variable OIDN_PATH (set it in the .Renviron file by calling 'usethis::edit_r_environ()') to the top-level directory for OIDN (the directory containing the "lib", "bin", and "include" directories) and re-installing this package from source.

`camera_scale`
: Default '1'. Amount to scale the camera up or down in size. Use this rather than scaling a scene.

`iso`
: Default '100'. Camera exposure.

`auto_exposure`
: Default 'FALSE'. If 'TRUE', automatically adjust the exposure with 'rayimage::render_exposure(auto = TRUE)'. If 'preview = TRUE', the preview window exposure is calibrated from the 90% luminance quantile of the first rendered frame.

`film_size`
: Default '22', in 'mm' (scene units in 'm'. Size of the film if using a realistic camera, otherwise ignored.

`min_variance`
: Default '0'. Minimum acceptable variance for a block of pixels for the adaptive sampler. Smaller numbers give higher quality images, at the expense of longer rendering times. If this is set to zero, the adaptive sampler will be turned off and the renderer will use the maximum number of samples everywhere.

`min_adaptive_size`
: Default '8'. Width of the minimum block size in the adaptive sampler.

`sample_method`
: Default 'sobol'. The type of sampling method used to generate random numbers. The other options are 'random' (worst quality but fastest), 'stratified' (only implemented for completion), 'sobol_blue' (best option for sample counts below 256), and 'sobol' (slowest but best quality, better than 'sobol_blue' for sample counts greater than 256). If 'samples > 256' and 'sobol_blue' is selected, the method will automatically switch to 'sample_method = "sobol"'.

`max_depth`
: Default 'NA', automatically sets to 50. Maximum number of bounces a ray can make in a scene. Alternatively, if a debugging option is chosen, this sets the bounce to query the debugging parameter (only for some options).

`roulette_active_depth`
: Default '100'. Number of ray bounces until a ray can stop bouncing via Russian roulette.

`ambient_light`
: Default 'FALSE', unless there are no emitting objects in the scene. If 'TRUE', the background will be a gradient varying from 'backgroundhigh' directly up (+y) to 'backgroundlow' directly down (-y).

`lookfrom` Default ‘c(0,1,10)‘. Location of the camera.

`lookat` Default ‘c(0,0,0)‘. Location where the camera is pointed.

`camera_up` Default ‘c(0,1,0)‘. Vector indicating the "up" position of the camera.

`aperture` Default ‘0.1‘. Aperture of the camera. Smaller numbers will increase depth of field, causing less blurring in areas not in focus.

`clamp_value` Default ‘Inf‘. If a bright light or a reflective material is in the scene, occasionally there will be bright spots that will not go away even with a large number of samples. These can be removed (at the cost of slightly darkening the image) by setting this to a small number greater than 1.

`filename` Default ‘NULL‘. If present, the renderer will write to the filename instead of the current device. Can write to JPEG/JPG, PNG, and high dynamic range EXR images. In the interactive preview, press Shift+Enter to save the current preview. A source filename with an extension produces numbered snapshots with the number inserted before the extension; otherwise snapshots are saved as ‘rayrender_snapshot1.png‘, ‘rayrender_snapshot2.png‘, and so on in the current directory.

`backgroundhigh` Default ‘#80b4ff‘. The "high" color in the background gradient. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.

`backgroundlow` Default ‘#ffffff‘. The "low" color in the background gradient. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between ‘0‘ and ‘1‘.

`shutteropen` Default ‘0‘. Time at which the shutter is open. Only affects moving objects.

`shutterclose` Default ‘1‘. Time at which the shutter is open. Only affects moving objects.

`camera_motion_blur`
Default ‘FALSE‘. Whether to blur camera movement over the shutter interval. Press ‘B‘ in interactive preview to toggle.

`shutter_speed` Default ‘NULL‘. Optional render-time override for the selected camera’s frame-relative shutter speed. A value of ‘1‘ samples the full frame-to-frame motion interval, ‘2‘ samples one-half, and ‘4‘ samples one-quarter. Higher values produce less motion blur. ‘Inf‘ disables temporal motion blur. This does not affect exposure or brightness.

`focal_distance` Default ‘NULL‘, automatically set to the ‘lookfrom-lookat‘ distance unless otherwise specified.

`ortho_dimensions`
Default ‘c(1,1)‘. Width and height of the orthographic camera. Will only be used if ‘fov = 0‘.

`tonemap` Default ‘raw‘, no tonemapping. Choose the tone mapping function, ‘reinhard‘ scales values by their individual color channels ‘color/(1+color)‘ and then performs the gamma adjustment. ‘uncharted‘ uses the mapping developed for Uncharted 2 by John Hable. ‘hbd‘ uses an optimized formula by Jim Hejl and Richard Burgess-Dawson.

`bloom` Default ‘TRUE‘. Set to ‘FALSE‘ to get the raw, pathtraced image. Otherwise, this performs a convolution of the HDR image of the scene with a sharp, long-tailed exponential kernel, which does not visibly affect dimly pixels, but does

result in emitters light slightly bleeding into adjacent pixels. This provides an antialiasing effect for lights, even when tonemapping the image. Pass in a matrix to specify the convolution kernel manually, or a positive number to control the intensity of the bloom (higher number = more bloom).

`parallel` Default ‘TRUE‘. If ‘FALSE‘, it will use all available cores to render the image (or the number specified in ‘options("cores")‘ or ‘options("Ncpus")‘ if that option is not ‘NULL‘).

`bvh_type` Default ‘"sah"‘, "surface area heuristic". Method of building the bounding volume hierarchy structure used when rendering. Other option is "equal", which splits tree into groups of equal size.

`environment_light`
Default ‘NULL‘. An image to be used for the background for rays that escape the scene. Supports EXR, HDR, PNG, and JPEG images. For reusable scene lights and multiple environments, use `infinite_light()` and `add_infinite_light()`. This argument adds a light to any infinite lights already attached to the scene.

`rotate_env` Default ‘0‘. The number of degrees to rotate all infinite lights around the scene, in addition to their individual rotations.

`intensity_env` Default ‘1‘. The amount to increase the intensity of the environment lighting. Useful if using a LDR (JPEG or PNG) image as an environment map. Applies only to the ‘environment_light‘ argument; scene lights have their own intensity.

`transparent_background`
Default ‘FALSE‘. If ‘TRUE‘, any initial camera rays that escape the scene will be marked as transparent in the final image. If for a pixel some rays escape and others hit a surface, those pixels will be partially transparent.

`debug_channel` Default ‘none‘. If ‘depth‘, function will return a depth map of rays into the scene instead of an image. If ‘normals‘, function will return an image of scene normals, mapped from 0 to 1. If ‘uv‘, function will return an image of the uv coords. If ‘variance‘, function will return an image showing the number of samples needed to take for each block to converge. If ‘dpdu‘ or ‘dpdv‘, function will return an image showing the differential ‘u‘ and ‘u‘ coordinates. If ‘color‘, function will return the raw albedo values (with white for ‘metal‘ and ‘dielectric‘ materials).

`plot_scene` Default ‘TRUE‘. Whether to plot the rendered scene.

`progress` Default ‘interactive()‘ if interactive session, ‘FALSE‘ otherwise.

`verbose` Default ‘FALSE‘. Prints information and timing information about scene construction and raytracing progress.

`print_debug_info`
Default ‘FALSE‘. This will print out additional information on the compilation environment with each render.

`new_page` Default ‘TRUE‘. Whether to call ‘grid::grid.newpage()‘ when plotting the image (if no filename specified). Set to ‘FALSE‘ for faster plotting (does not affect render time).

`integrator_type`
Default ‘"rtiow"‘ (the algorithm specified in the book "Raytracing in One Weekend", a basic form of path guiding). Other options include ‘"nee"‘ (Next Event

	Estimation, with direct light sampling) and '"basic"' (basic pathtracing, for high sample reference renders and debugging only). With 'nee', surfaces and participating media use RGB null-scattering transport; nee is selected automatically for scenes containing sky_light() or attached media (including clouds and media inside instances), overriding rtiow or basic. See set_medium().
screen_text	Default 'NULL'. Optional screen-space text overlay created with 'screen_text()', or a list of 'screen_text()' outputs to draw in order.
screen_line	Default 'NULL'. Optional screen-space line overlay created with 'screen_line()', or a list of 'screen_line()' outputs to draw in order. Labels are anchored to 3D world-space points, projected through the current camera, and drawn after rendering so text size and justification are independent of scene scale and view distance.
camera	Default 'NULL'. Scene-attached camera name, '"all"', or a 'ray_camera' object created with 'camera()'.
start_frame	Default '1'. First camera frame to render when using an animated camera.
end_frame	Default 'NA'. Last camera frame to render when using an animated camera. If 'NA', renders through the final frame.
mode	Default '"auto"'. Rendering mode. '"auto"' renders a still image for static cameras and an animation for animated cameras.

Value

A pathtraced image to the current device, or an image saved to a file. Invisibly returns the array (containing either debug data or the RGB).

Examples

```
# Generate a large checkered sphere as the ground
  scene = generate_ground(depth = -0.5,
                          material = diffuse(color = "white", checkercolor = "darkgreen"))
  render_scene(scene, parallel = TRUE, samples = 16, sample_method = "sobol")
  # Add a sphere to the center
  scene = scene |>
   add_object(sphere(x = 0, y = 0, z = 0, radius = 0.5, material = diffuse(color = c(1, 0, 1))))
  render_scene(scene, fov = 20, parallel = TRUE, samples = 16)
  # Add a marbled cube
  scene = scene |>
    add_object(cube(x = 1.1, y = 0, z = 0, material = diffuse(noise = 3)))
  render_scene(scene, fov = 20, parallel = TRUE, samples = 16)
  # Add a metallic gold sphere, using stratified sampling for a higher quality render
  # We also add a light, which turns off the default ambient lighting
  scene = scene |>
    add_object(sphere(x = -1.1, y = 0, z = 0, radius = 0.5,
                      material = metal(color = "gold", fuzz = 0.1))) |>
    add_object(sphere(y=10,z=-13,radius=2,material=light(intensity=40)))
  render_scene(scene, fov = 20, parallel = TRUE, samples = 16)
  # Lower the number of samples to render more quickly (here, we also use only one core).
  render_scene(scene, samples = 4, parallel = FALSE)
  # Add a floating R plot using the iris dataset as a png onto a floating 2D rectangle
```

```
tempfileplot = tempfile()
png(filename = tempfileplot, height = 400, width = 800)
plot(iris$Petal.Length, iris$Sepal.Width, col = iris$Species, pch = 18, cex = 4)
dev.off()
image_array = aperm(png::readPNG(tempfileplot), c(2, 1, 3))
scene = scene |>
  add_object(xy_rect(x = 0, y = 1.1, z = 0, xwidth = 2, angle = c(0, 0, 0),
                     flipped = TRUE,
                     material = diffuse(image_texture = image_array)))
render_scene(scene, fov = 20, parallel = TRUE, samples = 16)
# Move the camera
render_scene(scene, lookfrom = c(7, 1.5, 10), lookat = c(0, 0.5, 0), fov = 15, parallel = TRUE)
# Change the background gradient to a firey sky
render_scene(scene, lookfrom = c(7, 1.5, 10), lookat = c(0, 0.5, 0), fov = 15,
             backgroundhigh = "orange", backgroundlow = "red", parallel = TRUE,
             ambient = TRUE,
             samples = 16)
# Increase the aperture to blur objects that are further from the focal plane.
render_scene(scene, lookfrom = c(7, 1.5, 10), lookat = c(0, 0.5, 0), fov = 15,
             aperture = 1, parallel = TRUE, samples = 16)
# We can also capture a 360 environment image by setting `fov = 360` (can be used for VR).
# The left edge of the image is directly where the camera is pointing--we point the image
# backwards so the full cornell box is in the "center" of the environment map.
generate_cornell() |>
  add_object(ellipsoid(x = 555 / 2, y = 100, z = 555 / 2, a = 50, b = 100, c = 50,
                       material = metal(color = "lightblue"))) |>
 add_object(cube(x = 100, y = 130 / 2, z = 200, xwidth = 130, ywidth = 130, zwidth = 130,
                 material = diffuse(checkercolor = "purple",
                                    checkerperiod = 30), angle = c(0, 10, 0))) |>
  add_object(pig(x = 100, y = 190, z = 200, scale = 40, angle = c(0, 30, 0))) |>
  add_object(sphere(x = 420, y = 555 / 8, z = 100, radius = 555 / 8,
                    material = dielectric(color = "orange"))) |>
  add_object(xz_rect(x = 555 / 2, z = 555 / 2, y = 1, xwidth = 555, zwidth = 555,
                     material = glossy(checkercolor = "white",
                                       checkerperiod = 10, color = "dodgerblue"))) |>
 render_scene(lookfrom = c(278, 278, -10), lookat = c(278, 278, -300), clamp_value = 100,
              fov = 360,  samples = 16, width = 800, height = 800)
# Spin the camera around the scene, decreasing the number of samples to render faster. To make
# an animation, specify the a filename in `render_scene` for each frame and use the `av` package
# or ffmpeg to combine them all into a movie.
t = 1:9
xpos = 10 * sin(t * 18 * pi / 180 + pi / 2)
zpos = 10 * cos(t * 18 * pi / 180 + pi / 2)
image_output = list()
for (i in 1:9) {
  image_output[[i]] = render_scene(scene, samples = 16, plot_scene = FALSE,
            lookfrom = c(xpos[i], 1.5, zpos[i]), lookat = c(0, 0.5, 0), parallel = TRUE)
}
rayimage::plot_image_grid(image_output, dim = c(3,3) )
```

screen_line *Screen-space Lines*

Description

Creates line annotations for ‘render_scene()‘ that are anchored to 3D world-space endpoints but drawn in 2D screen space after rendering.

Usage

```
screen_line(
  x = 0,
  y = 0,
  z = 0,
  xend = 0,
  yend = 0,
  zend = 0,
  start = NULL,
  end = NULL,
  offset = c(0, 0),
  end_offset = offset,
  width = 2,
  color = "black",
  alpha = 1,
  lineend = "round",
  clip = TRUE,
  occlusion = FALSE,
  occlusion_mode = "anchor",
  occlusion_tolerance = 0.001
)
```

Arguments

x, y, z	Default ‘0‘. World-space coordinates of the line start point.
xend, yend, zend	Default ‘0‘. World-space coordinates of the line end point.
start, end	Default ‘NULL‘. Optional 3-column matrix/data frame of world-space start and end points. If supplied, overrides ‘x‘, ‘y‘, ‘z‘ and ‘xend‘, ‘yend‘, ‘zend‘.
offset	Default ‘c(0,0)‘. Pixel offset for the start point, as ‘c(x,y)‘, with positive y moving down the image.
end_offset	Default ‘offset‘. Pixel offset for the end point.
width	Default ‘2‘. Line width in pixels.
color	Default ‘"black"‘. Line color.
alpha	Default ‘1‘. Line alpha.
lineend	Default ‘"round"‘. Line end style. Options are ‘"round"‘, ‘"butt"‘, and ‘"square"‘.

clip Default ‘TRUE‘. If ‘TRUE‘, lines whose endpoints are both behind the camera are skipped.

occlusion Default ‘FALSE‘. If ‘TRUE‘, use scene geometry to hide the line.

occlusion_mode Default ‘"anchor"‘. If ‘"anchor"‘, occlusion skips the entire line when its midpoint is blocked. If ‘"line"‘ or ‘"partial"‘, each line pixel is hidden only when the scene depth at that pixel is closer than the interpolated screen-space line depth.

occlusion_tolerance Default ‘0.001‘. Endpoint tolerance for occlusion.

Value

A data frame describing screen-space line annotations.

Examples

```
# Build the scene first, then add independent screen-space line layers.
scene = generate_cornell(lightwidth = 250, lightdepth = 250) |>
  add_object(sphere(
    x = 180, y = 90, z = 260, radius = 90,
    material = diffuse(color = "#f2c14e")
  )) |>
  add_object(cube(
    x = 385, y = 90, z = 330, xwidth = 120, ywidth = 180, zwidth = 120,
    angle = c(0, 25, 0), material = diffuse(color = "#243846")
  ))

# A round-ended measurement across the sphere uses matrix start/end points.
sphere_measure = screen_line(
  start = matrix(c(90, 6, 260), ncol = 3),
  end = matrix(c(270, 6, 260), ncol = 3),
  offset = c(0, 0),
  end_offset = c(0, 0),
  width = 5,
  color = "#f2c14e",
  alpha = 1,
  lineend = "round",
  clip = TRUE,
  occlusion = FALSE,
  occlusion_mode = "anchor",
  occlusion_tolerance = 0.001
)

# A vertical guide on the rotated box exercises line-level occlusion.
box_measure = screen_line(
  start = matrix(c(385, 0, 330), ncol = 3),
  end = matrix(c(385, 180, 330), ncol = 3),
  offset = c(-12, 0),
  end_offset = c(-12, 0),
  width = 7,
  color = "#4d9de0",
  alpha = 0.8,
```

```
  lineend = "butt",
  clip = TRUE,
  occlusion = TRUE,
  occlusion_mode = "line",
  occlusion_tolerance = 0.004
)

# A back-wall rule uses square caps, no clipping, and the "partial" alias.
wall_rule = screen_line(
  start = matrix(c(85, 18, 545), ncol = 3),
  end = matrix(c(505, 18, 545), ncol = 3),
  offset = c(0, -8),
  end_offset = c(0, -8),
  width = 4,
  color = "white",
  alpha = 0.6,
  lineend = "square",
  clip = FALSE,
  occlusion = TRUE,
  occlusion_mode = "partial",
  occlusion_tolerance = 0.01
)

# Scalar coordinates work too, which is handy for one-off callout lines.
callout_line = screen_line(
  x = 180, y = 190, z = 260,
  xend = 245, yend = 255, zend = 230,
  offset = c(0, -4),
  end_offset = c(30, -24),
  width = 3,
  color = "black",
  alpha = 0.9,
  lineend = "round",
  clip = TRUE,
  occlusion = TRUE,
  occlusion_mode = "anchor",
  occlusion_tolerance = 0.002
)

# Build a 3D spiral from many short screen-space line segments.
spiral_theta = seq(0, 5 * pi, length.out = 120)
spiral_radius = seq(6, 65, length.out = length(spiral_theta))
spiral_points = cbind(
  445 + spiral_radius * cos(spiral_theta),
  390 + spiral_radius * sin(spiral_theta),
  seq(245, 390, length.out = length(spiral_theta))
)
spiral_count = nrow(spiral_points) - 1
spiral_lines = screen_line(
  start = spiral_points[-nrow(spiral_points), ],
  end = spiral_points[-1, ],
  width = seq(1, 4, length.out = spiral_count),
  color = grDevices::hcl(seq(250, 360, length.out = spiral_count), 80, 65),
```

```
  alpha = seq(0.35, 0.95, length.out = spiral_count),
  lineend = "round",
  clip = TRUE
)

line_layers = list(
  sphere_measure,
  box_measure,
  wall_rule,
  callout_line,
  spiral_lines
)

render_scene(
  scene,
  samples = 32,
  clamp_value = 5,
  aperture = 0,
  ambient_light = FALSE,
  screen_line = line_layers
)
```

`screen_text` *Screen-space Text*

Description

Creates text annotations for ‘render_scene()‘ that are anchored to 3D world-space points but drawn in 2D screen space after rendering.

Usage

```
screen_text(
  label,
  x = 0,
  y = 0,
  z = 0,
  point = NULL,
  offset = c(0, 0),
  hjust = 0,
  vjust = 0.5,
  size = 16,
  color = "black",
  font = "sans",
  lineheight = 1,
  background_color = "white",
  background_alpha = 0,
  just = "left",
```

```
  clip = TRUE,
  halo_color = NA,
  halo_expand = 0,
  halo_alpha = 1,
  halo_offset = c(0, 0),
  halo_blur = 0,
  halo_edge_softness = 0.1,
  halo_gap_fill = 2,
  halo_gap_fill_alpha_threshold = 0.25,
  occlusion = FALSE,
  occlusion_mode = "anchor",
  occlusion_tolerance = 0.001
)
```

Arguments

label	Character vector of labels.
x	Default ‘0‘. World-space x-coordinate of the anchor point.
y	Default ‘0‘. World-space y-coordinate of the anchor point.
z	Default ‘0‘. World-space z-coordinate of the anchor point.
point	Default ‘NULL‘. Optional 3-column matrix/data frame of world-space anchor points. If supplied, overrides ‘x‘, ‘y‘, and ‘z‘.
offset	Default ‘c(0,0)‘. Pixel offset from the projected anchor point, as ‘c(x,y)‘, with positive y moving down the image.
hjust	Default ‘0‘. Horizontal text adjustment, where ‘0‘ places the left edge at the anchor, ‘0.5‘ centers the label, and ‘1‘ right-aligns it.
vjust	Default ‘0.5‘. Vertical text adjustment, where ‘0‘ places the top edge at the anchor, ‘0.5‘ centers the label, and ‘1‘ bottom-aligns it.
size	Default ‘16‘. Text size in pixels.
color	Default ‘"black"‘. Text color.
font	Default ‘"sans"‘. Font family.
lineheight	Default ‘1‘. Line height passed to ‘rayimage::render_text_image()‘.
background_color	Default ‘"white"‘. Background color passed to ‘rayimage::render_text_image()‘.
background_alpha	Default ‘0‘. Background alpha passed to ‘rayimage::render_text_image()‘.
just	Default ‘"left"‘. Text justification passed to ‘rayimage::render_text_image()‘.
clip	Default ‘TRUE‘. If ‘TRUE‘, labels whose anchor point projects outside the image are skipped.
halo_color	Default ‘NA‘, no halo. If a color is specified, the text label will be surrounded by a halo of this color.
halo_expand	Default ‘0‘. Number of pixels to expand the halo.
halo_alpha	Default ‘1‘. Transparency of the halo.

`halo_offset` Default ‘c(0,0)‘. Pixel offset to apply to the halo, as ‘c(x,y)‘, with positive y moving down the image.

`halo_blur` Default ‘0‘. Amount of blur to apply to the halo.

`halo_edge_softness`
Default ‘0.1‘. Width of the softened halo edge transition, in pixels.

`halo_gap_fill` Default ‘2‘. Maximum alpha gap width, in pixels, to bridge in the halo outline.

`halo_gap_fill_alpha_threshold`
Default ‘0.25‘. Alpha threshold used to protect enclosed interior halo gaps from ‘halo_gap_fill‘.

`occlusion` Default ‘FALSE‘. If ‘TRUE‘, trace a ray from the camera to the anchor point and skip the label when another scene object blocks it.

`occlusion_mode` Default ‘"anchor"‘. If ‘"anchor"‘, occlusion skips the entire label when the anchor point is blocked. If ‘"label"‘ or ‘"partial"‘, the label and halo are treated as a screen-space banner at the anchor depth and each text pixel is hidden only when the scene depth at that pixel is closer.

`occlusion_tolerance`
Default ‘0.001‘. Endpoint tolerance for the occlusion ray. Values below ‘1‘ are treated as a fraction of the camera-to-anchor distance, which helps avoid self-occlusion when the anchor lies on a surface. Values of ‘1‘ or greater are treated as scene-unit distances.

Value

A data frame describing screen-space text annotations.

Examples

```
# Start with a Cornell box that has one bright object and one dark object.
scene = generate_cornell(lightwidth = 250, lightdepth = 250) |>
  add_object(sphere(
    x = 180, y = 90, z = 260, radius = 90,
    material = diffuse(color = "#f2c14e")
  )) |>
  add_object(cube(
    x = 385, y = 90, z = 330, xwidth = 120, ywidth = 180, zwidth = 120,
    angle = c(0, 25, 0), material = diffuse(color = "#243846")
  ))

# Labels can be built in separate calls and passed as a list to render_scene().
sphere_label = screen_text(
  label = "matte sphere\nanchor test",
  x = 180, y = 190, z = 260,
  offset = c(18, -28),
  hjust = 0.5,
  vjust = 1,
  size = 18,
  color = "black",
  lineheight = 0.95,
  background_alpha = 0,
```

```
  just = "left",
  halo_color = "#f2c14e",
  halo_expand = 4,
  halo_alpha = 0.8,
  halo_offset = c(1, 1),
  halo_blur = 0.5,
  halo_edge_softness = 0.5,
  halo_gap_fill = 1,
  halo_gap_fill_alpha_threshold = 0.2,
  occlusion = TRUE,
  occlusion_mode = "anchor",
  occlusion_tolerance = 0.002
)

# Dark text over a dark object remains legible with a subtle white halo.
box_label = screen_text(
  label = "dark text\nwhite halo",
  x = 385, y = 195, z = 20,
  offset = c(-16, -34),
  hjust = 0.5,
  vjust = 1,
  size = 16,
  color = "#111111",
  lineheight = 1.05,
  background_alpha = 0,
  just = "right",
  halo_color = "white",
  halo_expand = 6,
  halo_alpha = 0.72,
  halo_offset = c(-1, 2),
  halo_blur = 1.5,
  halo_edge_softness = 10,
  halo_gap_fill = 3,
  halo_gap_fill_alpha_threshold = 0.35,
  occlusion = TRUE,
  occlusion_mode = "label",
  occlusion_tolerance = 0.01
)

# A floor label shows multi-line text, custom justification, and partial occlusion.
floor_label = screen_text(
  label = "floor label\n\nocclusion",
  point = matrix(c(280, 0, 50), ncol = 3),
  offset = c(0, 18),
  hjust = 0.5,
  vjust = 0.7,
  size = 15,
  color = "#222222",
  lineheight = 1,
  background_color = "#fff7cc",
  background_alpha = 0.65,
  just = "center",
  clip = FALSE,
```

```
  halo_color = "white",
  halo_expand = 3,
  halo_alpha = 0.7,
  halo_blur = 0,
  halo_edge_softness = 0.25,
  halo_gap_fill = 2,
  halo_gap_fill_alpha_threshold = 0.25,
  occlusion = TRUE,
  occlusion_mode = "partial",
  occlusion_tolerance = 0.005
)

# Later list entries draw after earlier entries.
wall_label = screen_text(
  label = "back wall\nalways in front",
  x = 278, y = 330, z = 545,
  offset = c(0, -20),
  hjust = 0.5,
  vjust = 1,
  size = 12,
  color = "white",
  background_color = "black",
  background_alpha = 0.35,
  halo_color = "black",
  halo_expand = 2,
  clip = TRUE
)

text_layers = list(sphere_label, box_label, floor_label, wall_label)

render_scene(
  scene,
  samples = 32,
  clamp_value = 5,
  aperture = 0,
  fov = 50,
  ambient_light = FALSE,
  screen_text = text_layers
)
```

`segment` *Segment Object*

Description

Similar to the cylinder object, but specified by start and end points.

Usage

```
segment(
  start = c(0, -1, 0),
  end = c(0, 1, 0),
  radius = 0.1,
  phi_min = 0,
  phi_max = 360,
  from_center = TRUE,
  direction = NA,
  material = diffuse(),
  capped = TRUE,
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

start	Default ‘c(0, -1, 0)‘. Start point of the cylinder segment, specifing ‘x‘, ‘y‘, ‘z‘.
end	Default ‘c(0, 1, 0)‘. End point of the cylinder segment, specifing ‘x‘, ‘y‘, ‘z‘.
radius	Default ‘1‘. Radius of the segment.
phi_min	Default ‘0‘. Minimum angle around the segment.
phi_max	Default ‘360‘. Maximum angle around the segment.
from_center	Default ‘TRUE‘. If orientation specified via ‘direction‘, setting this argument to ‘FALSE‘ will make ‘start‘ specify the bottom of the segment, instead of the middle.
direction	Default ‘NA‘. Alternative to ‘start‘ and ‘end‘, specify the direction (via a length-3 vector) of the segment. Segment will be centered at ‘start‘, and the length will be determined by the magnitude of the direction vector.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
capped	Default ‘TRUE‘. Whether to add caps to the segment. Turned off when using the ‘light()‘ material.
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Notes: this will change the stated start/end position of the segment. Emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the segment in the scene.

Examples

```
#Generate a segment in the cornell box.
generate_cornell() |>
  add_object(segment(start = c(100, 100, 100), end = c(455, 455, 455), radius = 50)) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

# Draw a line graph representing a normal distribution, but with metal:
xvals = seq(-3, 3, length.out = 30)
yvals = dnorm(xvals)

scene_list = list()
for(i in 1:(length(xvals) - 1)) {
  scene_list[[i]] = segment(start = c(555/2 + xvals[i] * 80, yvals[i] * 800, 555/2),
                            end = c(555/2 + xvals[i + 1] * 80, yvals[i + 1] * 800, 555/2),
                             radius = 10,
                             material = metal())
}
scene_segments = do.call(rbind,scene_list)
generate_cornell() |>
  add_object(scene_segments) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

#Draw the outline of a cube:

cube_outline = segment(start = c(100, 100, 100), end = c(100, 100, 455), radius = 10) |>
  add_object(segment(start = c(100, 100, 100), end = c(100, 455, 100), radius = 10)) |>
  add_object(segment(start = c(100, 100, 100), end = c(455, 100, 100), radius = 10)) |>
  add_object(segment(start = c(100, 100, 455), end = c(100, 455, 455), radius = 10)) |>
  add_object(segment(start = c(100, 100, 455), end = c(455, 100, 455), radius = 10)) |>
  add_object(segment(start = c(100, 455, 455), end = c(100, 455, 100), radius = 10)) |>
  add_object(segment(start = c(100, 455, 455), end = c(455, 455, 455), radius = 10)) |>
  add_object(segment(start = c(455, 455, 100), end = c(455, 100, 100), radius = 10)) |>
  add_object(segment(start = c(455, 455, 100), end = c(455, 455, 455), radius = 10)) |>
  add_object(segment(start = c(455, 100, 100), end = c(455, 100, 455), radius = 10)) |>
  add_object(segment(start = c(455, 100, 455), end = c(455, 455, 455), radius = 10)) |>
  add_object(segment(start = c(100, 455, 100), end = c(455, 455, 100), radius = 10))

generate_cornell() |>
  add_object(cube_outline) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

#Shrink and rotate the cube
generate_cornell() |>
  add_object(group_objects(cube_outline, pivot_point = c(555/2, 555/2, 555/2),
                           angle = c(45,45,45), scale = c(0.5,0.5,0.5))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

set_active_camera *Set Active Camera*

Description

Sets the active camera for a 'ray_scene'.

Usage

```
set_active_camera(scene, camera)
```

Arguments

scene	Scene containing cameras.
camera	Camera name.

Value

A modified 'ray_scene'.

Examples

```
scene = generate_ground(material=diffuse(color="grey20")) |>
  add_camera(camera(name = "wide", fov = 55), active = FALSE) |>
  add_camera(camera(name = "detail", fov = 20), active = FALSE)

scene = set_active_camera(scene, "detail")
get_camera(scene)
```

set_medium *Attach a Medium to Closed Objects*

Description

Media nest: the innermost medium replaces the surrounding one. Cameras inside media are detected automatically. A medium with zero absorption and scattering creates a vacuum cavity. Intersecting, non-nested volume boundaries are unsupported.

Usage

```
set_medium(scene, medium, keep_surface = FALSE)
```

Arguments

scene	A ‘ray_scene‘ containing closed spheres, cubes, ellipsoids, or watertight consistently oriented triangle meshes.
medium	A description from [homogeneous_medium()], [grid_medium()], or [nanovdb_medium()]. Use ‘NULL‘ to remove an attachment.
keep_surface	Default ‘FALSE‘. Keep the object’s surface material when ‘TRUE‘, for example to place a medium inside glass. Dielectric attenuation adds absorption to the medium; it does not add emission.

Value

The scene with medium attachments stored independently of materials.

Examples

```
water = homogeneous_medium(sigma_a = c(0.3, 0.05, 0.02), sigma_s = 0.01)
glass = sphere(material = dielectric())
scene = set_medium(glass, water, keep_surface = TRUE)
```

---

set_scene_material *Set Material for All Objects*

Description

Set Material for All Objects

Usage

```
set_scene_material(scene, material)
```

Arguments

scene	A ray_scene object.
material	A material specification created by diffuse(), metal(), dielectric(), etc.

Value

A modified ray_scene with the new material applied to all objects

Examples

```
# Create a scene with different materials
scene = generate_cornell() |>
  add_object(sphere(x=555/2, y=555/2, z=555/2, radius=100))

# Set all objects to be metallic
scene = set_scene_material(scene, metal(color="gold"))

# Set all objects to be glass
scene = set_scene_material(scene, dielectric())
```

---

sky_light *Atmospheric Location and Time Sky Light*

Description

Evaluate the native Prague atmosphere at scene interactions, including altitude-dependent Sun and sky lighting, finite-distance haze, and in-scattering. Sun and Moon disk lights are included automatically. Add the light with `add_infinite_light()`. Rendering automatically selects `integrator_type = "nee"`. Use `sky_light_image()` for a cached sky image.

Usage

```
sky_light(
  lat,
  long,
  datetime,
  intensity = 1,
  rotation = 0,
  name = "sky",
  meters_per_unit = 1,
  atmosphere_origin = c(0, 0, 0),
  haze = TRUE,
  query_altitude = TRUE,
  haze_in_volumes = FALSE,
  deferred_haze = TRUE,
  cache_spectra = TRUE,
  transmission_table = TRUE,
  transmission_table_max_mb = 512,
  altitude = 0,
  visibility = 131.8,
  albedo = 0.5,
  sampling_resolution = 64,
  render_mode = "all",
  prague_rgb_correction = TRUE,
  prague_rgb_correction_strength = 1,
```

```
  prague_rgb_correction_gain = "auto",
  sun = TRUE,
  moon = TRUE,
  sun_resolution = 256,
  moon_resolution = 256,
  earthshine = TRUE,
  earthshine_albedo = 0.19,
  solar_irradiance_w_m2 = 1300,
  stars = FALSE,
  star_width = 1,
  stars_exposure = 0,
  planets = FALSE,
  celestial_resolution = 2048,
  number_cores = 1,
  haze_filter = TRUE
)
```

Arguments

`lat`	Latitude in degrees, between -90 and 90.
`long`	Longitude in degrees, between -180 and 180.
`datetime`	A single `POSIXct` date and time. Specify its time zone when constructing it with `as.POSIXct()`.
`intensity`	Default `1`. Nonnegative multiplier for this light's radiance.
`rotation`	Default `0`. Additional rotation in degrees around the world Y axis, using the same convention as `infinite_light()`.
`name`	Default `"sky"`. Unique light name within the scene.
`meters_per_unit`	Default `1`. Physical meters per world-space unit. Applies to distances along every axis.
`atmosphere_origin`	Default `c(0, 0, 0)`. World-space location of the geographic reference point, at `altitude` meters above sea level. World +Y is up; horizontal offsets follow the model's spherical Earth.
`haze`	Default `TRUE`. Include finite-distance haze and in-scattering between scene interactions. Set `FALSE` to evaluate native Prague lighting without this finite-distance haze. Sun/sky radiance and celestial disk filtering still include the atmosphere between the query location and space.
`query_altitude`	Default `TRUE`. Query lighting at each surface, cloud, or camera position, using `meters_per_unit` and `atmosphere_origin`. Set `FALSE` with `haze = FALSE` to evaluate all lighting at the fixed reference point and `altitude`. Finite-distance haze requires position-dependent queries, so `haze = TRUE` requires `query_altitude = TRUE`.
`haze_in_volumes`	Default `FALSE`. Integrate clear-air haze inside attached volume materials as well as outside, except in regions excluded by a medium's `haze` and `haze_density_threshold`

settings. Set FALSE to pause finite haze while a ray is inside any volume boundary, resuming at its exit. This applies to camera paths, shadow connections, and background opacity. The entire enclosed volume is excluded, including empty cells; the volume's own scattering, absorption, and emission remain active.

deferred_haze Default TRUE. Defer haze queries between interactions and estimate changes in transport weights using one reservoir sample. This reduces model queries while preserving the estimator's mean, with additional Monte Carlo noise. Extinction is applied at each completed span; emitting medium events also end a span. Works with either haze_in_volumes setting. Set FALSE to evaluate every haze interval. Signed corrections are averaged before display processing. Keep render_scene(clamp_value = Inf) to avoid clipping that estimator.

cache_spectra Default TRUE. Reuse exactly matching Prague sky spectra in a small cache per rendering thread. This changes neither the model nor individual samples. Set FALSE to disable this cache.

transmission_table
Default TRUE. Precompute transmission reconstruction at Prague's existing grid points, preserving its interpolation and individual sample results. Adds about 128 MiB per model at 50 km visibility, shared across threads. Tables exceeding transmission_table_max_mb, allocation failures, and queries outside the cached visibility slices use the original compressed evaluator. Set FALSE to retain that evaluator for all queries.

transmission_table_max_mb
Default 512. Maximum additional memory per model for the transmission table, in MiB (1024^2 bytes), shared across threads. Accepts nonnegative numbers, including fractional values. Set 0 to disable table allocation or Inf to remove the cap. If the complete table does not fit, use the original exact evaluator. Applies when transmission_table = TRUE.

altitude Default 0. Reference altitude in meters above sea level, at atmosphere_origin. Prague supports 0–15000 m.

visibility Default 131.8. Prague meteorological visibility in kilometers, from 20 to 131.8. Smaller values produce stronger haze.

albedo Default 0.5. Uniform ground reflectance for the sky model, between 0 and 1. Local surface materials are specified separately.

sampling_resolution
Default 64. Height of directional importance-sampling tables, from 16 to 2048. This does not limit the rendered sky's detail.

render_mode Default "all". Select sky and Sun ("all"), sky without the solar disk ("atmosphere"), or the solar disk alone ("sun"). Moon, star, and planet switches are independent of this selection.

prague_rgb_correction
Default TRUE. Apply skymodelr's Prague RGB tint correction.

prague_rgb_correction_strength
Default 1. Strength of the Prague RGB tint correction. Must be finite and nonnegative: 0 disables correction, and 1 applies the full calibrated correction.

prague_rgb_correction_gain
Default "auto". Calibrated Prague RGB gains, or a numeric vector of three finite, positive linear RGB multipliers.

`sun` Default `TRUE`. Include an independently sampled Sun disk when selected by `render_mode`. Set `FALSE` to omit direct sunlight and the visible disk; solar sky radiance and haze remain. A separate `sun_light()` overrides the automatic Sun.

`moon` Default `TRUE`. Include an independently sampled Moon disk with its phase and earthshine for this location and time. A separate `moon_light()` overrides the automatic Moon. Set `FALSE` to omit the automatic disk.

`sun_resolution` Default `256`. Sun disk texture width and height in pixels, at least 16; independent of the atmospheric `sampling_resolution`.

`moon_resolution`
Default `256`. Moon disk texture width and height in pixels, at least 16. Cropping and edge coverage can change the final dimensions.

`earthshine` Default `TRUE`. Illuminate the Moon's dark side with earthshine.

`earthshine_albedo`
Default `0.19`. Effective Earth reflectance used to calculate earthshine.

`solar_irradiance_w_m2`
Default `1300`. Reference solar irradiance at 1 AU, in W/m^2, used to normalize earthshine.

`stars` Default `FALSE`. Include a star field, filtered by the native atmosphere at each interaction.

`star_width` Default `1`. Star and planet point-spread width in pixels in the celestial background image.

`stars_exposure` Default `0`. Exposure adjustment for stars only, in stops.

`planets` Default `FALSE`. Include bright planets, filtered by the native atmosphere at each interaction.

`celestial_resolution`
Default `2048`. Height of the star/planet background image; its width is twice this value. Independent of the atmospheric sampling and Sun/Moon texture resolutions. Used only when `stars` or `planets` is enabled.

`number_cores` Default `1`. CPU threads used to prepare celestial textures.

`haze_filter` Default `TRUE`. Reduce finite-haze bands by averaging complete nearby atmospheric paths with a 0.5-degree vertical Gaussian, truncated at plus or minus 1.5 degrees. Each path uses Prague's sky spectra and its own endpoint and transmission; paths entering Earth are excluded. This changes finite in-scattering, while retaining the sky, celestial lights, and the actual ray's transmission. Existing haze is retained within 3 degrees of the Sun, with a smooth transition to filtering at 6 degrees. Set `FALSE` to use the endpoint-smoothing calculation. Each filtered haze query samples one of 129 weighted directions, preserving the full filter's mean with additional Monte Carlo noise. This sampling applies with either value of `deferred_haze`.

Details

Install the full-altitude Prague data with `skymodelr::download_sky_data(sea_level = FALSE)` before rendering. Atmospheric queries share skymodelr's coefficients and registered native API.

With `query_altitude = TRUE`, the sky and Sun elevation change with the altitude of each surface or cloud interaction. With `haze = FALSE`, finite haze is disabled while this local lighting remains active. Setting both `haze = FALSE` and `query_altitude = FALSE` uses the fixed reference observer for all lighting. Date and time stay fixed during an animation.

World +Y is up. With zero rotation, north is world +Z and east is world -X. `meters_per_unit` sets the physical scene scale, and `atmosphere_origin` locates the geographic reference point at `altitude` meters above sea level. Horizontal offsets follow the model's spherical Earth. The Sun is sampled independently of the sky sampling resolution. Its visibility includes Earth's curvature, allowing elevated clouds to receive sunlight on their undersides after the Sun disappears from the ground. Refraction is not modeled. Ground surfaces can still receive diffuse twilight and indirect cloud light.

A scene can contain one atmospheric sky. When the reference Sun elevation is below -4.2 degrees, Prague contributes black sky and no solar in-scattering. Enabled Moon, star, and planet lights still contribute, with atmospheric transmission and Earth occlusion applied normally. Queries outside 0–15000 m use the nearest modeled altitude; keep scene interactions within that range. Do not add another medium modeling the same clear-air scattering or absorption. Separate clouds can be added normally. Clouds default to no interior haze; `cloud(haze = TRUE)` enables haze below density 0.05, and `haze_density_threshold = NULL` removes that cutoff. See `cloud()` for details.

The model precomputes clear-air multiple scattering over a spherical Earth with uniform ground albedo. Local geometry and clouds block direct Sun and sky lighting but do not cast shadows into this precomputed in-scattering. Haze is disabled inside dielectric solids. Radiance is integrated spectrally and converted to renderer RGB; haze of RGB materials uses a broadband approximation. Finite-distance fitted transmission is normalized at zero distance and interpolated in optical depth over the first 100 m. Ray-anchored cumulative transport avoids accumulating fit errors at cloud null events. Finite haze is filtered over complete neighboring paths by default to reduce bands from subtracting independently fitted sky spectra. This is an angular regularization of the finite source. It does not blur the environment image or surface geometry.

Sun and Moon are prepared automatically using `sun_light()` and `moon_light()` with this sky's location, time, altitude, rotation, intensity, and color settings. Disk textures are generated without atmospheric filtering or a fixed horizon mask, then cached. The renderer applies spectral atmospheric filtering and Earth occlusion at each interaction. Disabling finite haze does not disable this filtering. Separate Sun or Moon lights replace the matching automatic disk, preserving their own settings. Removing or replacing the sky also removes or replaces its automatic celestial components.

Stars and planets use cached, unattenuated images of the full sphere, with native RGB atmospheric filtering and Earth occlusion at each interaction. Their map resolution affects point-source detail, not the Prague atmosphere.

Other infinite lights add to the sky. Additional image lights represent radiance outside the atmosphere and receive atmospheric haze; do not use an image that already includes the same haze. `sun_light()` and `moon_light()` request unattenuated textures automatically. The renderer applies spectral atmospheric filtering and Earth occlusion at each interaction. The sampled Sun replaces the built-in solar disk while preserving the sky and haze. Without an explicit disk altitude, ephemerides use this sky's reference altitude. Match light rotations and intensities when they should describe the same illumination. The precomputed haze remains Sun-driven: a Moon disk lights surfaces and clouds but adds no moonlit in-scattering or lunar halo. Image-only model choices such as `hosek` and `moon_atmosphere` belong to `sky_light_image()`.

Use `render_scene()`'s `iso` to adjust exposure, keeping it fixed within each comparison. With a transparent background, atmospheric in-scattering remains foreground radiance and scalar opacity

comes from primary-ray transmission. RGB transmission into an arbitrary compositing background is approximate.

The standalone vignette `vignette("sky-light", package = "rayrender")` builds the full capsule landscape, river, question blocks, pipes, and clouds, and demonstrates image skies, celestial lights, and additional sky controls.

Value

A `ray_infinite_light` containing a native atmospheric sky description.

See Also

`sky_light_image()`, `cloud()`, `sun_light()`, `moon_light()`

Examples

```
# Install the full-altitude Prague data once before rendering:
# skymodelr::download_sky_data(sea_level = FALSE)
if (
  requireNamespace("ambient", quietly = TRUE) &&
    requireNamespace("tree3d", quietly = TRUE)
) {
  # Scene units are kilometres.
  # Rounded green hills, with their lower capsule ends buried in the ground.
  # The rows are roughly 3-7, 17-26, and 60-85 km from the camera.
  # Small foreground hills stay crisp while larger distant hills fade.
  hills = data.frame(
    x = c(-1.1, 3, -6, -2, 3.5, 8, -27, -17, -6, 7, 22, 35, -45),
    z = c(-5.6, -2.2, 9, 15, 11, 17, 55, 63, 69, 58, 66, 60, 62),
    radius = c(0.30, 0.72, 1.7, 1.4, 2, 2.2, 6, 5, 5.5, 5, 7, 4, 2),
    top = c(0.7, 1.7, 3.8, 3, 4.7, 4.2, 10, 9, 22, 20.5, 31, 30, 34)
  )
  terrain_mat = diffuse(color = "#469D60")
  terrain = xz_rect(xwidth = 160, zwidth = 160, material = terrain_mat)
  for (i in seq_len(nrow(hills))) {
    h = hills[i, ]
    terrain = add_object(
      terrain,
      csg_object(
        csg_capsule(
          start = c(h$x, -h$radius, h$z),
          end = c(h$x, h$top - h$radius, h$z),
          radius = h$radius
        ),
        material = terrain_mat
      )
    )
  }

  # A river winds around the capsule footprints and turns out of sight behind
  # the distant pair at (-6, 69) and (7, 58). Coordinates and width are in km.
  # fmt: skip
```

```
river_bends = data.frame(
 x = c(0.3, 0.1, 0.7, 0.8, -1.2, -2.6, -3.9, -4.2, 0.2, 2.5, -1.1, 0.7, 1.4, 1.1, -2, -4.5),
  z = c(-12, -7, -4, -1, 3, 7, 12, 17, 23, 32, 43, 53, 62, 69, 76, 79)
)
river_curve = stats::splinefun(
  river_bends$z,
  river_bends$x,
  method = "natural"
)
river_z = seq(min(river_bends$z), max(river_bends$z), length.out = 600)
river_center = cbind(x = river_curve(river_z), z = river_z)

# Offset perpendicular to the tangent, keeping the river 1 km wide even
# through bends. Reverse the second bank to make one closed polygon.
river_width = 1
river_slope = river_curve(river_z, deriv = 1)
bank_offset = river_width /
  2 *
  cbind(1, -river_slope) /
  sqrt(1 + river_slope^2)
river_banks = rbind(
  river_center + bank_offset,
  (river_center - bank_offset)[length(river_z):1, ]
)

# Keep the polygon's world x coordinates and lift its top 1 m above
# ground. A thin extrusion gives the river an upward-facing surface.
terrain = add_object(
  terrain,
  extruded_polygon(
    river_banks,
    plane = "xz",
    top = 0.001,
    bottom = -0.001,
    flip_horizontal = TRUE,
    material = microfacet(color="#168BC4",transmission=TRUE, roughness=0.2)
  )
)

# Redwood-sized trees: 60-100 m tall, in a scene measured in kilometres.
# Generate three solid tree meshes once, then share them across 20,000 instances.
# Crown widths are 12-25 m and trunk diameters are approximately 2.4-5 m.
tree_types = c("pyramidal1", "pyramidal2", "columnar")
tree_colors = c("#245638", "#2B603E", "#305A3B")
tree_models = lapply(seq_along(tree_types), function(i) {
  tree3d::tree_mesh(
    crown_type = tree_types[i],
    solid = TRUE,
    resolution = "medium",
    tree_height = 0.08,
    trunk_height_ratio = c(0.25, 0.3, 0.35)[i],
    crown_width = c(0.018, 0.016, 0.020)[i],
    trunk_width = c(0.0032, 0.0036, 0.0040)[i],
```

```
    crown_color = tree_colors[i],
    trunk_color = "#794A35",
    ambient_intensity = 0
  ) |>
    raymesh_model()
})

# Log-spaced distances give the foreground enough trees to establish scale.
# Candidate positions follow the camera's view across the flat valley floor.
tree_count = 20000
tree_candidates = 4 * tree_count
set.seed(2028)
tree_distance = exp(runif(tree_candidates, log(1.4), log(85)))
tree_positions = data.frame(
  x = runif(tree_candidates, -0.65, 0.65) * tree_distance,
  z = -8 + tree_distance,
  size = runif(tree_candidates, 0.75, 1.25),
  angle = runif(tree_candidates, 0, 360),
  model = sample(seq_along(tree_models), tree_candidates, replace = TRUE)
)

# Leave enough room for the widest crown along both riverbanks and hills.
# Measure distance to river segments so the exclusion follows every bend.
tree_clearance = 0.015
tree_clear = rep(TRUE, nrow(tree_positions))
for (i in seq_len(nrow(river_center) - 1)) {
  dx = river_center[i + 1, 1] - river_center[i, 1]
  dz = river_center[i + 1, 2] - river_center[i, 2]
  along = pmin(
    pmax(
      ((tree_positions$x - river_center[i, 1]) *
        dx +
        (tree_positions$z - river_center[i, 2]) * dz) /
        (dx^2 + dz^2),
      0
    ),
    1
  )
  river_dx = tree_positions$x - (river_center[i, 1] + along * dx)
  river_dz = tree_positions$z - (river_center[i, 2] + along * dz)
  tree_clear = tree_clear &
    river_dx^2 + river_dz^2 > (river_width / 2 + tree_clearance)^2
}
for (i in seq_len(nrow(hills))) {
  tree_clear = tree_clear &
    (tree_positions$x - hills$x[i])^2 +
      (tree_positions$z - hills$z[i])^2 >
      (hills$radius[i] + tree_clearance)^2
}
tree_positions = head(tree_positions[tree_clear, ], tree_count)

# Each group shares one mesh/BVH. Vary height and yaw without copying geometry.
for (i in seq_along(tree_models)) {
```

```
  grove = tree_positions[tree_positions$model == i, ]
  terrain = add_object(
    terrain,
    create_instances(
      tree_models[[i]],
      x = grove$x,
      z = grove$z,
      angle_y = grove$angle,
      scale_x = grove$size,
      scale_y = grove$size,
      scale_z = grove$size
    )
  )
}

# Billowing Perlin volumes sit above each row of hills. Optical depth sets
# the cloud's own scattering; sky_light() separately supplies clear-air haze.
cloud_rows = data.frame(
  z = c(3, 21, 63),
  base = c(5, 6, 12.5),
  width = c(16, 32, 90),
  depth = c(10, 16, 24)
)
landscape = terrain
for (i in seq_len(nrow(cloud_rows))) {
  cl = cloud_rows[i, ]
  landscape = add_object(
    landscape,
    cloud(
      z = cl$z,
      y = cl$base + 1.8 / 2,
      width = cl$width,
      depth = cl$depth,
      height = 1.8,
      resolution = 64,
      coverage = 0.4,
      detail = 0.4,
      optical_depth = 4,
      g = 0.65,
      seed = 41 + i
    )
  )
}
day = as.POSIXct("2026-06-21 18:00:00", tz = "America/New_York")
sunset = as.POSIXct("2026-06-21 20:35:00", tz = "America/New_York")

render_sky = function(light, iso = 4, caption = "") {
  set.seed(2026)
  image = landscape |>
    add_infinite_light(light) |>
    render_scene(
      lookfrom = c(0, 0.35, -8),
      lookat = c(0, 2.4, 3),
```

```
      fov = 47,
      aperture = 0,
      width = 384,
      height = 240,
      samples = 32,
      integrator_type = "nee",
      iso = iso,
      tonemap = "raw",
      plot_scene = FALSE
    )
  rayimage::render_stack(list(
    image,
    rayimage::render_text_image(
      caption,
      size = 14,
      font = "sans",
      width = dim(image)[2],
      height = 34,
      just = "center",
      check_text_width = FALSE,
      check_text_height = FALSE
    )
  ))
}

# Haze changes contrast and color with distance. Hold visibility and ISO fixed.
rayimage::plot_image_grid(
  list(
    render_sky(
      sky_light(
        40.7,
        -74,
        day,
        meters_per_unit = 1000,
        haze = FALSE
      ),
      caption = "No finite haze"
    ),
    render_sky(
      sky_light(40.7, -74, day, meters_per_unit = 1000, visibility = 120),
      caption = "Finite haze, 120km"
    ),
    render_sky(
      sky_light(40.7, -74, day, meters_per_unit = 1000, visibility = 20),
      caption = "Finite haze, 20km"
    )
  ),
  dim = c(1, 3)
)

# Isolate altitude-dependent lighting by disabling finite haze in both images.
# The Sun is below the ground horizon, but the elevated cloud can still see it.
rayimage::plot_image_grid(
```

```
    list(
      render_sky(
        sky_light(
          40.7,
          -74,
          sunset,
          meters_per_unit = 1000,
          haze = FALSE,
          query_altitude = FALSE
        ),
        iso = 175,
        caption = "Fixed observer altitude"
      ),
      render_sky(
        sky_light(
          40.7,
          -74,
          sunset,
          meters_per_unit = 1000,
          haze = FALSE,
          query_altitude = TRUE
        ),
        iso = 175,
        caption = "Altitude at each interaction"
      ),
      render_sky(
        sky_light(
          40.7,
          -74,
          sunset,
          meters_per_unit = 1000,
          haze = TRUE,
          query_altitude = TRUE
        ),
        iso = 175,
        caption = "Altitude + haze each interaction"
      )
    ),
    dim = c(1, 3)
  )
}
```

`sky_light_image` *Image-Based Location and Time Sky Light*

Description

Generate a cached sky EXR with `skymodelr::generate_sky_latlong()` and use it as an infinite light. Add it to a scene with `add_infinite_light()`. The image represents one observer and illuminates every scene position with the same sky. Use `sky_light()` for altitude-dependent lighting

and finite-distance haze. Both constructors include Sun and Moon by default, controlled with sun and moon, and support optional stars and planets.

Usage

```
sky_light_image(
  lat,
  long,
  datetime,
  intensity = 1,
  rotation = 0,
  name = "sky",
  altitude = 0,
  visibility = 131.8,
  albedo = 0.5,
  resolution = 2048,
  hosek = TRUE,
  render_mode = "all",
  turbidity = 3,
  wide_spectrum = FALSE,
  below_horizon = TRUE,
  prague_rgb_correction = TRUE,
  prague_rgb_correction_strength = 1,
  prague_rgb_correction_gain = "auto",
  stars = FALSE,
  star_width = 1,
  stars_exposure = 0,
  planets = FALSE,
  moon = TRUE,
  moon_atmosphere = FALSE,
  moon_hosek = TRUE,
  exr_adopted_white = "D60",
  exr_metadata = TRUE,
  number_cores = 1,
  verbose = FALSE,
  sun = TRUE,
  environment_light_bake_white = FALSE,
  environment_light_bake_white_target = "D65",
  ...
)
```

Arguments

lat	Latitude in degrees, between -90 and 90.
long	Longitude in degrees, between -180 and 180.
datetime	A single POSIXct date and time. Specify its time zone when constructing it with as.POSIXct().
intensity	Default 1. Nonnegative multiplier for this light's radiance.

`rotation` Default `0`. Additional rotation in degrees around the world Y axis, using the same convention as `infinite_light()`.

`name` Default `"sky"`. Unique light name within the scene.

`altitude` Default `0`. Observer altitude in meters above sea level for the entire sky image. Prague supports 0–15000 m with its full-altitude data.

`visibility` Default `131.8`. Prague meteorological visibility in kilometers, from 20 to 131.8. Smaller values produce stronger haze.

`albedo` Default `0.5`. Uniform ground reflectance for the sky model, between 0 and 1. Local surface materials are specified separately.

`resolution` Default `2048`. Height of the cached image in pixels; its width is twice the height.

`hosek` Default `TRUE`. Generate a Hosek sky. Set `FALSE` to use Prague.

`render_mode` Default `"all"`. Select sky and Sun (`"all"`), sky without the solar disk (`"atmosphere"`), or the solar disk alone (`"sun"`). Moon, star, and planet switches are independent of this selection.

`turbidity` Default `3`. Hosek turbidity, from 1.7 to 10.

`wide_spectrum` Default `FALSE`. Use Prague's 55-channel sea-level data.

`below_horizon` Default `TRUE`. Include atmospheric radiance below the horizon.

`prague_rgb_correction`
Default `TRUE`. Apply skymodelr's Prague RGB tint correction.

`prague_rgb_correction_strength`
Default `1`. Strength of the Prague RGB tint correction. Must be finite and non-negative: 0 disables correction, and 1 applies the full calibrated correction.

`prague_rgb_correction_gain`
Default `"auto"`. Calibrated Prague RGB gains, or a numeric vector of three finite, positive linear RGB multipliers.

`stars` Default `FALSE`. Composite stars into the sky image.

`star_width` Default `1`. Stellar point-spread size, passed to `skymodelr::generate_stars()`.

`stars_exposure` Default `0`. Artistic exposure adjustment for stars, in stops.

`planets` Default `FALSE`. Composite bright planets into the sky image.

`moon` Default `TRUE`. Composite a Moon image into the sky. Set `FALSE` when adding a separate `moon_light()`.

`moon_atmosphere`
Default `FALSE`. Include atmospheric scattering of moonlight.

`moon_hosek` Default `TRUE`. Use Hosek for moonlight scattering. Set `FALSE` to use Prague.

`exr_adopted_white`
Default `"D60"`. Adopted white for EXR metadata: `"D60"`, `"D65"`, or numeric XYZ with Y = 1. Does not change image pixels.

`exr_metadata` Default `TRUE`. Attach skymodelr color metadata to the EXR.

`number_cores` Default `1`. CPU threads used to generate the cached image.

`verbose` Default `FALSE`. Print sky-generation progress information.

`sun` Default `TRUE`. Include the solar disk when selected by `render_mode`. Set `FALSE` to omit it from the image.

environment_light_bake_white
: Default FALSE. Bake chromatic adaptation from the generated EXR's white_current metadata into its RGB pixels before using it for lighting. Requires exr_metadata = TRUE. The adapted image is cached and reused for still images and animations.

environment_light_bake_white_target
: Default "D65". Target white point when baking: "D50", "D55", "D60", "D65", "D75", "E", or a finite numeric XYZ vector with positive Y (normalized to Y = 1). Unlike exr_adopted_white, this changes the image pixels when baking is enabled.

...
: Additional named arguments forwarded by skymodelr::generate_sky_latlong() to its star, planet, and Moon generators. Pass settings directly; location, date-time, and the cached filename are managed by this light. Native atmospheric controls belong to sky_light().

Details

Image generation happens before rendering and is cached for the R session. Changing only intensity, rotation, or name reuses the image. Changing the baked target white point reuses the generated sky and caches a separate adapted image. White-balance controls belong to this light; pass the light to add_infinite_light() before calling render_scene() or render_animation(). Install any required Prague data with skymodelr::download_sky_data() first; rendering does not download datasets.

With zero rotation, north is world +Z and east is world -X. Date, time, and observer altitude remain fixed throughout the image. This light supports the same integrators as infinite_light() and adds no finite atmospheric haze. Use render_scene()'s iso to adjust exposure.

For a separately sampled Sun, set render_mode = "atmosphere" and add sun_light() with matching location, time, and atmospheric settings. This avoids relying on the environment image to resolve the small solar disk.

Value

A ray_infinite_light containing a cached-image sky description.

See Also

sky_light(), infinite_light(), sun_light(), moon_light()

Examples

```
time = as.POSIXct("2026-06-21 18:00:00", tz = "America/New_York")
scene = sphere(material = diffuse("white")) |>
  add_object(generate_ground())

scene |>
  add_infinite_light(sky_light_image(40.7, -74, time)) |>
  render_scene(
    lookfrom = c(0, 1, 10),
    lookat = c(0, 0, 0),
    width = 600,
```

```
    height = 300,
    samples = 32,
    iso = 3
  )

# A Prague image sky with an independently sampled solar disk.
# Install Prague data with skymodelr::download_sky_data() before rendering.
scene |>
  add_infinite_light(sky_light_image(
    40.7,
    -74,
    time,
    hosek = FALSE,
    render_mode = "atmosphere",
    altitude = 0,
    visibility = 50,
    albedo = 0.3
  )) |>
  add_infinite_light(sun_light(
    40.7,
    -74,
    time,
    sky_args = list(altitude = 0, visibility = 50, albedo = 0.3)
  )) |>
render_scene(
  lookfrom = c(0, 1, 10),
  lookat = c(0, 0, 0),
  aperture = 0,
  width = 600,
  height = 300,
  samples = 32,
  iso = 3
)
```

---

sphere *Sphere Object*

Description

Sphere Object

Usage

```
sphere(
  x = 0,
  y = 0,
  z = 0,
  radius = 1,
  material = diffuse(),
```

```
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

x	Default '0'. x-coordinate of the center of the sphere.
y	Default '0'. y-coordinate of the center of the sphere.
z	Default '0'. z-coordinate of the center of the sphere.
radius	Default '1'. Radius of the sphere.
material	Default diffuse. The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default 'c(0, 0, 0)'. Angle of rotation around the x, y, and z axes, applied in the order specified in 'order_rotation'.
order_rotation	Default 'c(1, 2, 3)'. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default 'FALSE'. Whether to flip the normals.
scale	Default 'c(1, 1, 1)'. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the sphere in the scene.

Examples

```
#Generate a sphere in the cornell box.
generate_cornell() |>
  add_object(sphere(x = 555/2, y = 555/2, z = 555/2, radius = 100)) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, clamp_value = 5)

#Generate a gold sphere in the cornell box
generate_cornell() |>
  add_object(sphere(x = 555/2, y = 100, z = 555/2, radius = 100,
                    material = microfacet(color = "gold"))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, clamp_value = 5)
```

sun_light *Sun and Moon Infinite Lights*

Description

Create detailed celestial disks without embedding them in a latitude-longitude environment map. skymodelr supplies the position and apparent angular size for the observer, date, and time. The Sun uses Prague solar radiance queries; the Moon uses skymodelr's surface texture, phase, earthshine, and radiometry routines. Add these descriptions to a scene with `add_infinite_light()`.

Usage

```
sun_light(
  lat,
  long,
  datetime,
  sky_args = list(),
  resolution = 256,
  intensity = 1,
  rotation = 0,
  name = "sun"
)

moon_light(
  lat,
  long,
  datetime,
  sky_args = list(),
  moon_args = list(),
  resolution = 256,
  intensity = 1,
  rotation = 0,
  name = "moon"
)
```

Arguments

`lat`	Latitude in degrees, between -90 and 90.
`long`	Longitude in degrees, between -180 and 180.
`datetime`	A single `POSIXct` date and time, with an explicit time zone.
`sky_args`	Default `list()`. Named atmospheric settings for skymodelr's Prague model: `altitude`, `visibility`, `albedo`, `wide_spectrum`, `number_cores`, and the `prague_rgb_correction` options accepted by `skymodelr::calculate_sky_values()`. `hosek = FALSE` is accepted; `hosek = TRUE` is unsupported because these lights use per-direction queries. Defaults match that function. Without a native atmospheric sky, altitude describes one observer for the whole light. With `sky_light()`, a missing

altitude uses that sky's reference altitude for ephemeris placement; atmospheric filtering uses each interaction's position.

`resolution` Default `256`. Target disk image width and height in pixels, at least 16. The Moon's padded image is cropped without downsampling; edge coverage can add a few pixels. Texture detail is evaluated directly at this resolution, independently of the sky map.

`intensity` Default `1`. Nonnegative multiplier on physical radiance.

`rotation` Default `0`. Rotation in degrees around world Y, with the same convention as `infinite_light()`.

`name` Default `"sun"` or `"moon"`. Unique light name within the scene.

`moon_args` Default `list()`. Named options for skymodelr's Moon routines: `earthshine = TRUE`, `earthshine_albedo = 0.19`, `solar_irradiance_w_m2 = 1300`, and `moon_extinction_kV = 0.172`. Native atmospheric scenes ignore `moon_extinction_kV`, using Prague transmission instead.

Details

Requires skymodelr and its Prague data. Install datasets with `skymodelr::download_sky_data()` before rendering. Requires the public `skymodelr::generate_sun_disk()` and `skymodelr::generate_moon_disk()` exports.

Pair a Sun disk with `sky_light_image(..., hosek = FALSE, render_mode = "atmosphere")` to exclude the sky map's rasterized Sun. Leave `moon = FALSE` in that sky when adding a Moon disk. Lights add radiance; they do not eclipse or occlude each other, and adding a second copy doubles its light. With `sky_light()`, explicit Sun and Moon lights replace the corresponding automatic disks; there is no need to change `sun`, `moon`, or `render_mode`. The clear-air sky and haze retain that sky light's own location, time, rotation, and intensity.

Both lights use north at world +Z, east at world -X, and up at world +Y, matching `sky_light()`. They have no parallax and add no scene geometry. The geometric horizon clips their emission. Preparation generates and caches linear EXRs before worker threads start; phase, time, and position stay fixed during animation. Direct illumination samples each disk's solid angle, so a small apparent diameter does not depend on a high-resolution sky sampler.

The Sun texture uses Prague's solar disk profile, mapped to the ephemeris angular diameter. Moon radiance preserves skymodelr's phase-dependent total irradiance and atmospheric extinction, with its spectral RGB and atmospheric tint. Earthshine is part of the generated phase texture. These are radiance images; exposure and tone mapping are applied only by the final render.

With a native atmospheric sky, preparation automatically requests `atmospheric_attenuation = FALSE` through skymodelr's public disk generators. Sun radiance and lunar phase, surface detail, and earthshine are generated before atmospheric filtering. The renderer applies the Sun's spectrum or the Moon's reference spectrum to Prague transmission along each light direction. Spherical Earth visibility clips each direction, so a partially set disk can illuminate high clouds while being hidden from the ground. Texture detail remains independent of the sky sampling resolution. Skymodelr must support the new argument. Without a native atmosphere, the existing fixed-observer haze and geometric horizon clipping remain in effect. The sky's `haze = FALSE` option removes finite scene haze while keeping this celestial filtering. Its `query_altitude` option controls whether the filtering and horizon follow each interaction or use the reference observer. The precomputed clear-air haze is Sun-driven; lunar atmospheric in-scattering and halos are not modeled. See `sky_light()` for the model's supported domain.

Value

A ray_infinite_light description.

Examples

```
time = as.POSIXct("2026-01-28 21:00:00", tz = "Pacific/Auckland")
scene = sphere(material = diffuse()) |>
  add_infinite_light(moon_light(-36.87593, 174.7647, time)) |>
  add_camera(camera(aperture = 0))
render_scene(scene, integrator_type = "nee", auto_exposure = TRUE)
```

---

text3d *Text Object*

Description

Text Object

Usage

```
text3d(
  label,
  x = 0,
  y = 0,
  z = 0,
  text_height = 1,
  orientation = "xy",
  material = diffuse(),
  font = "sans",
  font_style = "plain",
  font_color = "black",
  font_lineheight = 1,
  font_size = 100,
  background_color = "white",
  background_alpha = 0,
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

label	Text string.
x	Default ‘0‘. x-coordinate of the center of the label.
y	Default ‘0‘. y-coordinate of the center of the label.

z	Default ‘0‘. z-coordinate of the center of the label.
text_height	Default ‘1‘. Height of the text.
orientation	Default ‘xy‘. Orientation of the plane. Other options are ‘yz‘ and ‘xz‘.
material	Default diffuse. The material, called from one of the material functions diffuse, metal, or dielectric.
font	Default ‘"sans"‘. A character string specifying the font family (e.g., ‘"Arial"‘, ‘"Times"‘, ‘"Helvetica"‘).
font_style	A character string specifying the font style, such as ‘"plain"‘, ‘"italic"‘, or ‘"bold"‘. Default is ‘"plain"‘.
font_color	Default ‘"black"‘. The font color.
font_lineheight	Default ‘12‘. The lineheight for strings with newlines.
font_size	Default ‘100‘. The size of the font. Note that this does not control the size of the text, just the resolution as rendered in the texture.
background_color	Default ‘"white"‘. The background color.
background_alpha	Default ‘0‘. The background opacity. ‘1‘ is fully opaque.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the text in the scene.

Examples

```
#Generate a label in the cornell box.
generate_cornell() |>
  add_object(text3d(label="Cornell Box", x=555/2,y=555/2,z=555/2,text_height=60,
                    material=diffuse(color="grey10"), angle=c(0,0,0))) |>
  render_scene(samples=16)
#Change the orientation
generate_cornell() |>
  add_object(text3d(label="YZ Plane", x=550,y=555/2,z=555/2,text_height=200,
                    orientation = "yz",
                    material=diffuse(color="grey10"), angle=c(0,0,0))) |>
 add_object(text3d(label="XY Plane", z=550,y=555/2,x=555/2,text_height=200,
                    orientation = "xy",
                    material=diffuse(color="grey10"), angle=c(0,0,0))) |>
```

```
 add_object(text3d(label="XZ Plane", z=555/2,y=5,x=555/2,text_height=200,
                   orientation = "xz",
                   material=diffuse(color="grey10"))) |>
  render_scene(samples=16)
#Add an label in front of a sphere
generate_cornell() |>
  add_object(text3d(label="Cornell Box", x=555/2,y=555/2,z=555/2,text_height=90,
                   material=diffuse(color="grey10"), angle=c(0,0,0))) |>
  add_object(text3d(label="Sphere", x=555/2,y=100,z=100,text_height=60,
                   material=diffuse(color="white"), angle=c(0,0,0))) |>
  add_object(sphere(y=100,radius=100,z=555/2,x=555/2,
                   material=glossy(color="purple"))) |>
  add_object(sphere(y=555,radius=100,z=-1000,x=555/2,
                   material=light(intensity=100,
                                  spotlight_focus=c(555/2,100,100)))) |>
  render_scene(samples=16)

#A room full of bees
bee_list = list()
for(i in 1:100) {
bee_list[[i]] = text3d("B", x=20+runif(1)*525, y=20+runif(1)*525, z=20+runif(1)*525,
                       text_height = 50, angle=c(0,0,0))
}
bees = do.call(rbind,bee_list)
generate_cornell() |>
  add_object(bees) |>
  render_scene(samples=16)
# If you have ragg installed, you can also use color emojis.
library(rayrender)
generate_cornell(light_position = c(555/2,554,10),
                 lightwidth = 10, lightdepth = 100,
                 lightintensity = 800) |>
  add_object(text3d(label="\U1F30A",font_size = 500,angle=c(0,0,0),
                   x=555/2,y=555/2,z=260,text_height=1000)) |>
  add_object(text3d(label="\U1F6A3", x=180,y=140,z=260-50,
                   text_height=400, font_size = 500,
                   material=diffuse(color="black"),
                   angle=c(0,0,30))) |>
  add_object(text3d(label="\U1F5FB", x=180,y=230,z=260+50,text_height=300,
                   font_size = 500,material=diffuse(color="black"),
                   angle=c(0,0,0))) |>
  render_scene(samples=16)
```

`triangle` *Triangle Object*

Description

Triangle Object

Usage

```
triangle(
  v1 = c(1, 0, 0),
  v2 = c(0, 1, 0),
  v3 = c(-1, 0, 0),
  n1 = rep(NA, 3),
  n2 = rep(NA, 3),
  n3 = rep(NA, 3),
  color1 = rep(NA, 3),
  color2 = rep(NA, 3),
  color3 = rep(NA, 3),
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  reversed = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

v1	Default ‘c(1, 0, 0)‘. Length-3 vector indicating the x, y, and z coordinate of the first triangle vertex.
v2	Default ‘c(0, 1, 0)‘. Length-3 vector indicating the x, y, and z coordinate of the second triangle vertex.
v3	Default ‘c(-1, 0, 0)‘. Length-3 vector indicating the x, y, and z coordinate of the third triangle vertex.
n1	Default ‘NA‘. Length-3 vector indicating the normal vector associated with the first triangle vertex.
n2	Default ‘NA‘. Length-3 vector indicating the normal vector associated with the second triangle vertex.
n3	Default ‘NA‘. Length-3 vector indicating the normal vector associated with the third triangle vertex.
color1	Default ‘NA‘. Length-3 vector or string indicating the color associated with the first triangle vertex. If NA but other vertices specified, color inherits from material.
color2	Default ‘NA‘. Length-3 vector or string indicating the color associated with the second triangle vertex. If NA but other vertices specified, color inherits from material.
color3	Default ‘NA‘. Length-3 vector or string indicating the color associated with the third triangle vertex. If NA but other vertices specified, color inherits from material.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.

order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
reversed	Default ‘FALSE‘. Similar to the ‘flipped‘ argument, but this reverses the handedness of the triangle so it will be oriented in the opposite direction.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the XZ plane in the scene.

Examples

```
#Generate a triangle in the Cornell box.
generate_cornell() |>
  add_object(triangle(v1 = c(100, 100, 100), v2 = c(555/2, 455, 455), v3 = c(455, 100, 100),
                      material = diffuse(color = "purple"))) |>
   render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
                ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
#Pass individual colors to each vertex:
generate_cornell() |>
  add_object(triangle(v1 = c(100, 100, 100), v2 = c(555/2, 455, 455), v3 = c(455, 100, 100),
                      color1 = "green", color2 = "yellow", color3 = "red")) |>
   render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
                ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

---

xy_rect *Rectangular XY Plane Object*

Description

Rectangular XY Plane Object

Usage

```
xy_rect(
  x = 0,
  y = 0,
  z = 0,
  xwidth = 1,
  ywidth = 1,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
```

```
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

x	Default ‘0‘. x-coordinate of the center of the rectangle.
y	Default ‘0‘. x-coordinate of the center of the rectangle.
z	Default ‘0‘. z-coordinate of the center of the rectangle.
xwidth	Default ‘1‘. x-width of the rectangle.
ywidth	Default ‘1‘. y-width of the rectangle.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the XY plane in the scene.

Examples

```
#Generate a purple rectangle in the cornell box.
generate_cornell() |>
  add_object(xy_rect(x = 555/2, y = 100, z = 555/2, xwidth = 200, ywidth = 200,
             material = diffuse(color = "purple"))) |>
  render_scene(lookfrom = c(278, 278, -800), lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

#Generate a gold plane in the cornell box
generate_cornell() |>
  add_object(xy_rect(x = 555/2, y = 100, z = 555/2,
                     xwidth = 200, ywidth = 200, angle = c(0, 30, 0),
                     material = metal(color = "gold"))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

xz_rect *Rectangular XZ Plane Object*

Description

Rectangular XZ Plane Object

Usage

```
xz_rect(
  x = 0,
  xwidth = 1,
  z = 0,
  zwidth = 1,
  y = 0,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

x	Default '0'. x-coordinate of the center of the rectangle.
xwidth	Default '1'. x-width of the rectangle.
z	Default '0'. z-coordinate of the center of the rectangle.
zwidth	Default '1'. z-width of the rectangle.
y	Default '0'. y-coordinate of the center of the rectangle.
material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default 'c(0, 0, 0)'. Angle of rotation around the x, y, and z axes, applied in the order specified in 'order_rotation'.
order_rotation	Default 'c(1, 2, 3)'. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default 'FALSE'. Whether to flip the normals.
scale	Default 'c(1, 1, 1)'. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the XZ plane in the scene.

Examples

```
#Generate a purple rectangle in the cornell box.
generate_cornell() |>
  add_object(xz_rect(x = 555/2, y = 100, z = 555/2, xwidth = 200, zwidth = 200,
             material = diffuse(color = "purple"))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)

#Generate a gold plane in the cornell box
generate_cornell() |>
  add_object(xz_rect(x = 555/2, y = 100, z = 555/2,
             xwidth = 200, zwidth = 200, angle = c(0, 30, 0),
             material = metal(color = "gold"))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

---

yz_rect *Rectangular YZ Plane Object*

Description

Rectangular YZ Plane Object

Usage

```
yz_rect(
  x = 0,
  y = 0,
  z = 0,
  ywidth = 1,
  zwidth = 1,
  material = diffuse(),
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  flipped = FALSE,
  scale = c(1, 1, 1)
)
```

Arguments

x	Default '0'. x-coordinate of the center of the rectangle.
y	Default '0'. y-coordinate of the center of the rectangle.
z	Default '0'. z-coordinate of the center of the rectangle.
ywidth	Default '1'. y-width of the rectangle.
zwidth	Default '1'. z-width of the rectangle.

material	Default diffuse.The material, called from one of the material functions diffuse, metal, or dielectric.
angle	Default ‘c(0, 0, 0)‘. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation‘.
order_rotation	Default ‘c(1, 2, 3)‘. The order to apply the rotations, referring to "x", "y", and "z".
flipped	Default ‘FALSE‘. Whether to flip the normals.
scale	Default ‘c(1, 1, 1)‘. Scale transformation in the x, y, and z directions. If this is a single value, number, the object will be scaled uniformly. Note: emissive objects may not currently function correctly when scaled.

Value

Single row of a tibble describing the YZ plane in the scene.

Examples

```
#Generate a purple rectangle in the cornell box.
generate_cornell() |>
  add_object(yz_rect(x = 100, y = 100, z = 555/2, ywidth = 200, zwidth = 200,
                     material = diffuse(color = "purple"))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
#Generate a gold plane in the cornell box
generate_cornell() |>
  add_object(yz_rect(x = 100, y = 100, z = 555/2,
                     ywidth = 200, zwidth = 200, angle = c(0, 30, 0),
                     material = metal(color = "gold"))) |>
  render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
               ambient_light = FALSE, samples = 16, parallel = TRUE, clamp_value = 5)
```

Index

add_camera, 4
add_infinite_light, 5, *124*, *136*
add_infinite_light(), *90*, *151*, *161*, *164*, *167*
add_object, 5
add_object(), *17*
animate_objects, 6
animate_objects(), *19*
arrow, 8

bezier_curve, 11
bezier_curve(), *113*

camera, 13
cloud, 17
cloud(), *155, 156*
cone, 20
create_instances, 22
create_instances(), *19*
csg_box, 25
csg_capsule, 26
csg_combine, 27
csg_cone, 29
csg_cylinder, 30
csg_ellipsoid, 31
csg_elongate, 32
csg_group, 34
csg_object, 35
csg_onion, 36
csg_plane, 38
csg_pyramid, 39
csg_rotate, 40
csg_round, 41
csg_rounded_cone, 42
csg_scale, 43
csg_sphere, 44
csg_torus, 45
csg_translate, 46
csg_triangle, 47
cube, 48

cylinder, 50

dielectric, *9*, *12*, *21*, *23*, *35*, *49, 50*, 51, *57, 58*, *65*, *75, 76*, *98*, *108*, *110*, *115*, *118*, *147*, *166*, *170*, *172*, *174, 175*, *177*
diffuse, *9*, *12*, *21*, *23*, *35*, *49, 50*, 53, *57, 58*, *61*, *65*, *75, 76*, *98*, *108*, *110*, *115*, *118*, *147*, *166*, *170*, *172*, *174, 175*, *177*
disk, 56

ellipsoid, 58
extruded_path, 59
extruded_polygon, 64

generate_camera_motion, 70
generate_cornell, 74
generate_ground, 75
generate_studio, 76
get_camera, 77
get_infinite_light, 78
get_saved_keyframes, 78
glossy, 79
grid_medium, 82
grid_medium(), *19*
group_objects, 84
group_objects(), *19*

hair, 85
hair(), *113*
has_denoiser, 87
homogeneous_medium, 88
homogeneous_medium(), *18, 19*

infinite_light, 90, *136*
infinite_light(), *5*, *152*, *163, 164*, *168*

lambertian, 93
light, 93
list_cameras, 95

list_infinite_lights, 96

mesh3d_model, 96
metal, *9*, *12*, *21*, *23*, *35*, *49, 50*, *57, 58*, *65*, *75, 76*, 98, *98*, *108*, *110*, *115*, *118*, *147*, *166*, *170*, *172*, *174, 175*, *177*
microfacet, 101
moon_light *(*sun_light*)*, 167
moon_light(), *5*, *154–156*, *163, 164*

nanovdb_medium, 105

obj_model, 106

path, 109
pig, 112
ply_model, 114
preview_camera, 115

r_obj, 116
raymesh_model, 117
remove_camera, 120
remove_infinite_light, 121
render_animation, 121
render_animation(), *164*
render_ao, 127
render_preview, 130
render_scene, 131
render_scene(), *90*, *155*, *164*

screen_line, 139
screen_text, 142
segment, 146
set_active_camera, 149
set_medium, *137*, 149
set_medium(), *19*
set_scene_material, 150
sky_light, *125*, *137*, 151
sky_light(), *5*, *18, 19*, *161*, *164*, *167, 168*
sky_light_image, 161
sky_light_image(), *5*, *151*, *155, 156*
sphere, 165
sun_light, 167
sun_light(), *5*, *154–156*, *164*

text3d, 169
triangle, 171

xy_rect, 173
xz_rect, 175

yz_rect, 176