Package ‘rayrender’

June 8, 2023

Type Package
Title Build and Raytrace 3D Scenes
Version 0.29.6
Date 2023-05-17
Maintainer Tyler Morgan-Wall <tylermw@gmail.com>
License GPL-3
Copyright file inst/COPYRIGHTS
Depends R (>= 3.0.2)
Imports Rcpp (>= 1.0.0), parallel, magrittr, purrr, png, raster, decidio, rayimage (>= 0.9.0), stats, progress
Suggests sf, spData, dplyr, Rvcg, testthat (>= 3.0.0)
LinkingTo Rcpp, RcppThread, progress, spacefillr (>= 0.3.0)
URL https://www.rayrender.net,
https://github.com/tylermorganwall/rayrender
RoxygenNote 7.2.3
SystemRequirements C++17
Biarch true
Encoding UTF-8
NeedsCompilation yes
Author Tyler Morgan-Wall [aut, cph, cre]
   (<https://orcid.org/0000-0002-3131-3814>),
   Syoyo Fujita [ctb, cph],
   Melissa O’Neill [ctb, cph],
   Vilya Harvey [ctb, cph]
Repository CRAN
Date/Publication 2023-06-08 04:40:02 UTC
R topics documented:

<table>
<thead>
<tr>
<th>R topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_object</td>
<td>3</td>
</tr>
<tr>
<td>animate_objects</td>
<td>4</td>
</tr>
<tr>
<td>arrow</td>
<td>6</td>
</tr>
<tr>
<td>bezier_curve</td>
<td>9</td>
</tr>
<tr>
<td>cone</td>
<td>12</td>
</tr>
<tr>
<td>csg_box</td>
<td>14</td>
</tr>
<tr>
<td>csg_capsule</td>
<td>15</td>
</tr>
<tr>
<td>csg_combine</td>
<td>16</td>
</tr>
<tr>
<td>csg_cone</td>
<td>18</td>
</tr>
<tr>
<td>csg_cylinder</td>
<td>19</td>
</tr>
<tr>
<td>csg_ellipsoid</td>
<td>21</td>
</tr>
<tr>
<td>csg_elongate</td>
<td>22</td>
</tr>
<tr>
<td>csg_group</td>
<td>24</td>
</tr>
<tr>
<td>csg_object</td>
<td>24</td>
</tr>
<tr>
<td>csg_onion</td>
<td>26</td>
</tr>
<tr>
<td>csg_plane</td>
<td>28</td>
</tr>
<tr>
<td>csg_pyramid</td>
<td>29</td>
</tr>
<tr>
<td>csg_rotate</td>
<td>30</td>
</tr>
<tr>
<td>csg_round</td>
<td>31</td>
</tr>
<tr>
<td>csg_rounded_cone</td>
<td>32</td>
</tr>
<tr>
<td>csg_scale</td>
<td>34</td>
</tr>
<tr>
<td>csg_sphere</td>
<td>35</td>
</tr>
<tr>
<td>csg_torus</td>
<td>36</td>
</tr>
<tr>
<td>csg_translate</td>
<td>37</td>
</tr>
<tr>
<td>csg_triangle</td>
<td>38</td>
</tr>
<tr>
<td>cube</td>
<td>39</td>
</tr>
<tr>
<td>cylinder</td>
<td>41</td>
</tr>
<tr>
<td>dielectric</td>
<td>43</td>
</tr>
<tr>
<td>diffuse</td>
<td>45</td>
</tr>
<tr>
<td>disk</td>
<td>48</td>
</tr>
<tr>
<td>ellipsoid</td>
<td>49</td>
</tr>
<tr>
<td>extruded_path</td>
<td>51</td>
</tr>
<tr>
<td>extruded_polygon</td>
<td>57</td>
</tr>
<tr>
<td>generate_camera_motion</td>
<td>60</td>
</tr>
<tr>
<td>generate_cornell</td>
<td>64</td>
</tr>
<tr>
<td>generate_ground</td>
<td>65</td>
</tr>
<tr>
<td>generate_studio</td>
<td>66</td>
</tr>
<tr>
<td>get_saved_keyframes</td>
<td>68</td>
</tr>
<tr>
<td>glossy</td>
<td>68</td>
</tr>
<tr>
<td>group_objects</td>
<td>71</td>
</tr>
<tr>
<td>hair</td>
<td>73</td>
</tr>
<tr>
<td>lambertian</td>
<td>75</td>
</tr>
<tr>
<td>light</td>
<td>76</td>
</tr>
<tr>
<td>mesh3d_model</td>
<td>78</td>
</tr>
<tr>
<td>metal</td>
<td>79</td>
</tr>
<tr>
<td>microfacet</td>
<td>82</td>
</tr>
</tbody>
</table>
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")))
if(rayrender:::run_documentation()) {
  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**

```r
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.
**animate_objects**

- **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**

```r
# Render a pig
if(rayrender:::run_documentation()) {
  generate_studio() %>%
    add_object(pig(y=-1.2,scale=0.5,angle=c(0,-70,0))) %>%
    add_object(sphere(y=5,x=5,z=5,radius=2,material=light())) %>%
    render_scene(samples=128,sample_method = "sobol_blue")
}

# Render a moving pig
if(rayrender:::run_documentation()) {
  generate_studio() %>%
    add_object(
      animate_objects(
        pig(y=-1.2,scale=0.5,angle=c(0,-70,0)),
        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=128,sample_method = "sobol_blue",clamp_value = 10)
}

# Render a shrinking pig
if(rayrender:::run_documentation()) {
  generate_studio() %>%
    add_object(
      animate_objects(
        pig(y=-1.2,scale=0.5,angle=c(0,-70,0)),
        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=128,sample_method = "sobol_blue",clamp_value = 10)
}

# Render a spinning pig
if(rayrender:::run_documentation()) {
  generate_studio() %>%
    add_object(
      animate_objects(
        pig(y=-1.2,scale=0.5,angle=c(0,-70,0)),
        start_axis_rotation = c(1,1,1), end_axis_rotation = c(1,1,1)
      ) %>%
    add_object(sphere(y=5,x=5,z=5,radius=2,material=light())) %>%
    render_scene(samples=128,sample_method = "sobol_blue",clamp_value = 10)
}
```

# Render a pig

# Render a moving pig

# Render a shrinking pig

# Render a spinning pig
generate_studio() %>%
  add_object(
    animate_objects(
      pig(y=-1.2, scale=0.5, angle=c(0,-70,0)),
      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=128, sample_method = "sobol_blue", clamp_value = 10)
}
if(rayrender:::run_documentation()) {
  # Shorten the open shutter time frame
  generate_studio() %>%
  add_object(
    animate_objects(
      pig(y=-1.2, scale=0.5, angle=c(0,-70,0)),
      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=128, sample_method = "sobol_blue", clamp_value = 10,
              shutteropen=0.4, shutterclose = 0.6)
}
if(rayrender:::run_documentation()) {
  # 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,-70,0)),
      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=128, sample_method = "sobol_blue", clamp_value = 10,
              shutteropen=0, shutterclose = 0.1)
}
if(rayrender:::run_documentation()) {
  # 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,-70,0)),
                  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=128, sample_method = "sobol_blue", clamp_value = 10,
              shutteropen=0, shutterclose = 0.1)
Description

Composite object (cone + segment)

Usage

\[
\text{arrow(}
\begin{align*}
    &\text{start } = \text{c(0, 0, 0)}, \\
    &\text{end } = \text{c(0, 1, 0)}, \\
    &\text{radius\_top } = \text{0.2}, \\
    &\text{radius\_tail } = \text{0.1}, \\
    &\text{tail\_proportion } = \text{0.5}, \\
    &\text{direction } = \text{NA}, \\
    &\text{from\_center } = \text{TRUE}, \\
    &\text{material } = \text{diffuse()}, \\
    &\text{flipped } = \text{FALSE}, \\
    &\text{scale } = \text{c(1, 1, 1)}
\end{align*}
\]

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 ‘0.2’. 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
if(rayrender:::run_documentation()) {
  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=128)
}

if(rayrender:::run_documentation()) {
  # 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=128)
}

if(rayrender:::run_documentation()) {
  # 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=128)
}

if(rayrender:::run_documentation()) {
  # 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=128, lookfrom=c(0,5,10), lookat=c(0,-0.5,0), fov=16)
}

if(rayrender:::run_documentation()) {
  # Plot a 3D vector field for a gravitational well:

blezier_curve

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=128,
    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:
if(rayrender:::run_documentation()) {
genenerate_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=128)

if(rayrender:::run_documentation()) {
    #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(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=128)
}

if(rayrender:::run_documentation()) {
    #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=128)
}

if(rayrender:::run_documentation()) {
    #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=128)
}

if(rayrender:::run_documentation()) {
    #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"))
cone

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**

```r
# Generate a cone in a studio, pointing upwards:
if(rayrender:::run_documentation()) {
  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=128, clamp_value=10)
}
```

```r
# Change the radius, length, and direction
if(rayrender:::run_documentation()) {
  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=128, clamp_value=10)
}
```

```r
# Give custom start and end points (and customize the color/texture)
if(rayrender:::run_documentation()) {
  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")) %>%
  render_scene(samples=128, clamp_value=10)
}
```

```r
# Specify cone via direction and location, instead of start and end positions
# Length is derived from the magnitude of the direction.
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(sphere(y=5,material=light())) %>%
  render_scene(samples=128, clamp_value=10)
```

```r
# Render the position from the base, instead of the center of the cone:
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)) %>%
```

csg_box

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=128)

---

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**

```r
if(rayrender::run_documentation()) {
    # 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, lookfrom=c(7,3,7))
}
```

```r
if(rayrender::run_documentation()) {
    # 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, lookfrom=c(7,3,7))
}
```

```r
if(rayrender::run_documentation()) {
```
# Subtract two boxes to make stairs

```r
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,lookfrom=c(7,3,7),fov=13)
```

---

## csg_capsule

### Description

CSG Capsule

### Usage

```r
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

```r
if(rayrender:::run_documentation()) {
  # 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,fov=20)
}

if(rayrender:::run_documentation()) {
  # 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,fov=20,
                 lookat=c(0,0.5,-2),lookfrom=c(3,3,10))
}
if(rayrender:::run_documentation()) {
  # Show the effect of changing the radius
```
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
radius Default ‘0.5’. Blending radius.

Value

List describing the combined csg object in the scene.

Examples

if(rayrender:::run_documentation()) {
  #Combine two spheres:
  generate_ground(material=diffuse(checkercolor="grey20")) %>%
    add_object(csg_object(csg_combine(
      csg_sphere(x=-0.4,z=-0.4),
    ))) %>%
    render_scene(clamp_value=10,fov=20,
      lookat=c(0,0.5,-2),lookfrom=c(-3,3,10))
}
csg_combine

`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,fov=20,lookfrom=c(-3,5,10))`}

if(rayrender:::run_documentation()) {
  #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,fov=20,lookfrom=c(-3,5,10))`}

if(rayrender:::run_documentation()) {
  #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,fov=20,lookfrom=c(-3,5,10))`}

if(rayrender:::run_documentation()) {
  #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,fov=20,lookfrom=c(-3,5,10))`}

if(rayrender:::run_documentation()) {
  #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,fov=20,lookfrom=c(-3,5,10))`}

if(rayrender:::run_documentation()) {
  #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,fov=20,lookfrom=c(-3,5,10))`
render_scene(clamp_value=10, fov=20, lookfrom=c(-3, 5, 10))
}
if(rayrender:::run_documentation()) {
# 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, fov=20, lookfrom=c(-3, 5, 10))
}
if(rayrender:::run_documentation()) {
# 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, 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, specifying 'x', 'y', 'z'.
- **end**: Default `c(0, 1, 0)`. End point of the cone, specifying 'x', 'y', 'z'.
- **radius**: Default `1`. Radius of the bottom of the cone.
csg_cylinder

Value

List describing the box in the scene.

Examples

```r
if(rayrender:::run_documentation()) {
  #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,fov=20)
}
if(rayrender:::run_documentation()) {
  #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,fov=20,
         lookat=c(0,0.5,-2),lookfrom=c(3,3,10))
}
if(rayrender:::run_documentation()) {
  #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,fov=20,
         lookat=c(0,0.5,-2),lookfrom=c(-3,3,10))
}
if(rayrender:::run_documentation()) {
  #Render a glass cone in a Cornell box
  generate_cornell() %>%
    add_object(csg_object(
        csg_cylinder(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)
}
```
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, specifying 'x', 'y', 'z'.
end Default 'c(0, 1, 0)'. End point of the cylinder, specifying '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

if(rayrender:::run_documentation()) {
  #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,fov=20)
}

if(rayrender:::run_documentation()) {
  #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,fov=20,
      lookat=c(0,0.5,-2),lookfrom=c(3,3,10))
}

if(rayrender:::run_documentation()) {
  #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,fov=20,
      lookat=c(0,0.5,-2),lookfrom=c(3,3,10))
}

if(rayrender:::run_documentation()) {
  #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

if(rayrender:::run_documentation()) {
  #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,fov=20)
}

if(rayrender:::run_documentation()) {
  #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,fov=20,lookfrom=c(0,5,10))
}

if(rayrender:::run_documentation()) {
#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,fov=20)
}
if(rayrender:::run_documentation()) {
  #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)
}

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

if(rayrender:::run_documentation()) {
  #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, fov=40, lookfrom=c(0, 10, 10))
}
if(rayrender:::run_documentation()) {
# 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, fov=40, lookfrom=c(0, 10, 10))
}
if(rayrender:::run_documentation()) {
# 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, fov=40, lookfrom=c(0, 10, 10))
}
if(rayrender:::run_documentation()) {
# 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, fov=40, lookfrom=c(0, 10, 10))
}
if(rayrender:::run_documentation()) {
# 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, 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**

```r
if(rayrender:::run_documentation()) {
  # 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, 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

if(rayrender::run_documentation()) {
# 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_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.
csg_onion

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

if(rayrender:::run_documentation()) {
  #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,lookat=c(0,-0.5,0),
               lookfrom=c(3,5,10),fov=35)
}

if(rayrender:::run_documentation()) {
  #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,lookat=c(0,-0.5,0),
               lookfrom=c(3,5,10),fov=20)
}

if(rayrender:::run_documentation()) {
  #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)
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

```r
if(rayrender:::run_documentation()) {
  #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)
}
```

```r
if(rayrender:::run_documentation()) {
  #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)
}
if(rayrender:::run_documentation()) {
  # Re-orient the plane using the normal and add_object(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)
}

---

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**

```
if(rayrender:::run_documentation()) {
  # 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,lookfrom=c(-3,1,10),
             fov=15, lookat=c(0,-0.5,0))
}
```
if(rayrender:::run_documentation()) {
# 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,lookfrom=c(-3,1,10),
    fov=15, lookat=c(0,-0.5,0))
}
if(rayrender:::run_documentation()) {
# 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,lookfrom=c(-3,1,10),
    fov=15, lookat=c(0,-0.5,0))
}

---

csg_rotate  

### Description

CSG Rotate

### Usage

```r
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

```r
if(rayrender:::run_documentation()) {
  #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)
}
if(rayrender:::run_documentation()) {
  #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)
csg_rounded_cone

**Arguments**

- **object**: CSG object.
- **radius**: Default ‘0.1’. Rounding distance.

**Value**

List describing the triangle in the scene.

**Examples**

```r
if(rayrender:::run_documentation()) {
  #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)
}
if(rayrender:::run_documentation()) {
  #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=2), 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, specifying 'x', 'y', 'z'.
end    Default 'c(0, 1, 0)'. End point of the cone, specifying '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

if(rayrender:::run_documentation()) {
  #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, fov=20)
}
if(rayrender:::run_documentation()) {
  #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, fov=20,
                 lookat=c(0,0.5,-2), lookfrom=c(3,3,10))
}
if(rayrender:::run_documentation()) {
  #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, fov=20,
                 lookat=c(0,0.5,-2), lookfrom=-c(3,3,10))
}
if(rayrender:::run_documentation()) {
  #Render a glass rounded cone in a Cornell box:
  generate_cornell() %>%
    add_object(csg_object(}
csg_scale

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)
}

---

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**

```r
if(rayrender:::run_documentation()) {
  #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**

\[
csg\_sphere(x = 0, y = 0, z = 0, \text{radius} = 1)
\]

**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**

```r
if(rayrender:::run_documentation()) {
  #Generate a simple sphere:
  generate\_ground() %>%
    add\_object(csg\_object(csg\_sphere(),
      material=glossy(color="purple"))) %>%
    render\_scene(clamp\_value=10)
}
if(rayrender:::run_documentation()) {
  #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,\text{radius}=100),
      material=glossy(checkercolor="purple", checkerperiod=100))) %>%
    render\_scene(clamp\_value=10)
}
if(rayrender:::run_documentation()) {
  #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,\text{radius}=100),
      csg\_sphere(x=555/2,y=555/2+50,z=555/2,\text{radius}=80)),
      material=glossy(color="purple"))) %>%
    render\_scene(clamp\_value=10)
}
```
if(rayrender:::run_documentation()) {
    #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)
    }
}

if(rayrender:::run_documentation()) {
    #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)
    }
}

---

csg_torus  |  CSG Torus

### Description

CSG Torus

### Usage

```r
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

```r
if(rayrender::run_documentation()) {
  # 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,lookfrom=c(0,5,10),fov=30)
}
if(rayrender::run_documentation()) {
  # 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,lookfrom=c(0,5,10),fov=30)
}
if(rayrender::run_documentation()) {
  # 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,lookfrom=c(0,5,10),fov=30)
}
if(rayrender::run_documentation()) {
  # 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)
}
```

---

**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

```r
if(rayrender:::run_documentation()) {
  #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,lookfrom=c(0,5,10),fov=30,
        lookat=c(-1,0.5,-1))
}
if(rayrender:::run_documentation()) {
  #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,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

```r
if(rayrender::run_documentation()) {
  # 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, fov=20)
}
if(rayrender::run_documentation()) {
  # 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, fov=20)
}
if(rayrender::run_documentation()) {
  # 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, 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),
cube

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.
if(rayrender:::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}

#Generate a gold cube in the cornell box
if(rayrender:::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}
cylinder #Generate a rotated dielectric box in the cornell box
if(rayrender:::run_documentation()) {
    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 = 128, 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Default ‘0’; x-coordinate of the center of the cylinder</td>
</tr>
<tr>
<td>y</td>
<td>Default ‘0’; y-coordinate of the center of the cylinder</td>
</tr>
<tr>
<td>z</td>
<td>Default ‘0’; z-coordinate of the center of the cylinder</td>
</tr>
<tr>
<td>radius</td>
<td>Default ‘1’. Radius of the cylinder.</td>
</tr>
<tr>
<td>length</td>
<td>Default ‘1’. Length of the cylinder.</td>
</tr>
<tr>
<td>phi_min</td>
<td>Default ‘0’. Minimum angle around the segment.</td>
</tr>
<tr>
<td>phi_max</td>
<td>Default ‘360’. Maximum angle around the segment.</td>
</tr>
</tbody>
</table>
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.

if(rayrender:::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}

#Rotate the cylinder

if(rayrender:::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}

# Only render a subtended arc of the cylinder,

if(rayrender:::run_documentation()) {
  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())) %>%
    render_scene(lookfrom = c(278, 278, -800), lookat = c(278, 278, 0), fov = 40, 
                ambient_light = FALSE, samples = 128, parallel = TRUE, clamp_value = 5)
}
Dielectric (glass) Material

Description

Dielectric (glass) Material

Usage

dielectric(
  color = "white",
  refraction = 1.5,
  attenuation = c(0, 0, 0),
  priority = 0,
  importance_sample = FALSE,
  bump_texture = NA,
  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 attenu-
ation through the material. Higher numbers will result in less of that color
making it through the material. Note: This assumes the object has a closed
surface.

priority
  Default '0'. When two dielectric materials overlap, the one with the lower prior-
ity value is used for intersection. NOTE: If the camera is placed inside a dielec-
tric 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 'NA'. 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

```r
# Generate a checkered ground
scene = generate_ground(depth=-0.5, material = diffuse(checkercolor="grey30", checkerperiod=2))
if(rayrender:::run_documentation()) {
  render_scene(scene, parallel=TRUE)
}

# Add a glass sphere
if(rayrender:::run_documentation()) {
  scene %>%
    add_object(sphere(x=-0.5, radius=0.5, material=dielectric())) %>%
    render_scene(parallel=TRUE, samples=128)
}

# Add a rotated colored glass cube
if(rayrender:::run_documentation()) {
  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=128)
}

# Add an area light behind and at an angle and turn off the ambient lighting
if(rayrender:::run_documentation()) {
  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=1000)
}

# 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.
if(rayrender:::run_documentation()) {
  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 = 1000)
}

# 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.
if(rayrender:::run_documentation()) {
  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))) %>%
```
```r
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 = 128,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.
if(rayrender:::run_documentation()) {
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 = 128,lookfrom=c(5,3,5))
}
```

diffuse

**Diffuse Material**

**Description**
Diffuse Material

**Usage**
```r
diffuse(
color = "#ffffff",
checkercolor = NA,
checkerperiod = 3,
noise = 0,
noisephase = 0,
noiseintensity = 10,
noisecolor = "#000000",
gradien\_color = NA,
gradien\_transpose = FALSE,
gradien\_point\_start = NA,
gradien\_point\_end = NA,
gradien\_type = "hsv",
image\_texture = NA,
image\_repeat = 1,
alpha\_texture = NA,
bump\_texture = NA,
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 ‘NA’. 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 ‘NA’. A matrix or filename (specifying a greyscale image) to be used to specify the transparency.
bump_texture Default ‘NA’. 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

```r
#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()))
if(rayrender:::run_documentation()) {
  render_scene(scene, lookfrom=c(278,278,-800), lookat = c(278,278,0), samples=128,
  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)))
if(rayrender:::run_documentation()) {
  render_scene(scene, lookfrom=c(278,278,-800), lookat = c(278,278,0), samples=128,
  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)))
if(rayrender:::run_documentation()) {
  render_scene(scene, lookfrom=c(278,278,-800), lookat = c(278,278,0), samples=128,
  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")))
if(rayrender:::run_documentation()) {
    render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=128,
                 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")))
if(rayrender:::run_documentation()) {
    render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=128,
                 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

```r
#Generate a disk in the cornell box.
if(rayrender:::run_documentation()) {
  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),
      ambient_light = FALSE, samples = 128, parallel = TRUE, clamp_value = 5)
}

#Rotate the disk.
if(rayrender:::run_documentation()) {
  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),
      ambient_light = FALSE, samples = 128, parallel = TRUE, clamp_value = 5)
}

#Pass a value for the inner radius.
if(rayrender:::run_documentation()) {
  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),
      ambient_light = FALSE, samples = 128, parallel = TRUE, clamp_value = 5)
}
```
Description

Note: light importance sampling for this shape is currently approximated by a sphere. This will fail for ellipsoids with large differences between axes.

Usage

```r
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
if(rayrender:::run_documentation()) {
    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),
                 ambient_light = FALSE, samples = 128, parallel = TRUE, clamp_value = 5)
}

# Change the axes to make it taller rather than wide:
if(rayrender:::run_documentation()) {
    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),
                 ambient_light = FALSE, samples = 128, parallel = TRUE, clamp_value = 5)
}

# Rotate it and make it dielectric:
if(rayrender:::run_documentation()) {
    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),
                 ambient_light = FALSE, samples = 128, 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,
extruded_path

```r
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.
extruded_path

- **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’.


- **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

```r
if(rayrender:::run_documentation()) {
  # 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)
}
if(rayrender:::run_documentation()) {
  # 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=128, sample_method="sobol_blue")
}
if(rayrender:::run_documentation()) {
  # 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=128, sample_method="sobol_blue")
}
if(rayrender:::run_documentation()) {
  # 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=128, sample_method="sobol_blue")
}
if(rayrender:::run_documentation()) {
  # 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(0.25,0.5,0.5,0.25))
  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=128, sample_method="sobol_blue")
}
if(rayrender:::run_documentation()) {
  # 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
```

```r
extruded_path
```
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(0.3,0.5,0.5), fov=12, width=800, height=800, clamp_value = 10,
               aperture=0.025, samples=128, sample_method="sobol_blue")
if(rayrender:::run_documentation()) {
  # 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=128, sample_method="sobol_blue")
}
if(rayrender:::run_documentation()) {
  # 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(0.3,0.5,0.5), fov=12, width=800, height=800, clamp_value = 10,
                 aperture=0.025, samples=128, sample_method="sobol_blue")
}
if(rayrender:::run_documentation()) {
  # 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(0.3,0.5,0.5), fov=12, width=800, height=800, clamp_value = 10,
                 aperture=0.025, samples=128, sample_method="sobol_blue")
}
if(rayrender:::run_documentation()) {

extruded_path

Only generate part of the curve, specified by the u_min and u_max arguments

```r
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=128, sample_method="sobol_blue")
}
```

```r
if(rayrender:::run_documentation()) {
  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=128, sample_method = "sobol_blue",  
    width = 800, height=800)
}
```

```r
if(rayrender:::run_documentation()) {
  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 = "#002a61",checkerperiod = 1.5)) |
  add_object(extruded_path(points = points, width=0.7, linear_step = TRUE,  
    polygon = circ_polygon, twists = 2, closed = TRUE,  
    polygon_end = star_polygon,  
    material=dielectric(priority = 1, refraction = 1.2,  
    attenuation=c(1,0.3,1)*10))) |
  add_object(extruded_path(points = points, width=0.4, linear_step = TRUE,  
    polygon = circ_polygon, twists = 2, closed = TRUE,  
    polygon_end = star_polygon,  
    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 = "purple",intensity = 5))) |
  render_scene(lookat=c(0,0.5,1),fov=12,  
    width=800,height=800, clamp_value = 10,  
    aperture=0.025, samples=128, 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 possible 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 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
# Manually create a polygon object, here a star:
if(rayrender:::run_documentation()) {
  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)
}

if(rayrender:::run_documentation()) {
  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=128, 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:

```r
if(rayrender:::run_documentation()) {
  hollow_star = rbind(star_polygon, 0.8*star_polygon)
}

if(rayrender:::run_documentation()) {
  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=128, 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.
if(rayrender:::run_documentation()) {
  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=128, lookat=c(0,0.9,0), fov=40)
}

# Now add the zy plane:
if(rayrender:::run_documentation()) {
  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 = "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=128, lookat=c(0,0.9,0), fov=40)
}

# We can also directly pass in sf polygons:
if(rayrender:::run_documentation()) {
  if(length(find.package("spData", quiet=TRUE)) > 0) {
    us_states = spData::us_states
  }
  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.
if(rayrender:::run_documentation()) {
  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=128,fov=60)
}

# Here we use the raw coordinates, but offset the polygon manually.
if(rayrender:::run_documentation()) {
  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="lightblue",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=128,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.
if(rayrender:::run_documentation()) {
  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="lightblue",intensity=250))) %>%
    add_object(sphere(y=30,x=100,z=-60,radius=10,
                      material=light(color="orange",intensity=250))) %>%
    render_scene(parallel=TRUE,lookfrom = c(-60,50,-40),lookat=c(0,-5,0),samples=128,fov=30)
}
**generate_camera_motion**

**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**

```r
generate_camera_motion(
  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 = NULL,  # 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 = 0,  # Default `0`. A numeric vector of aperture values.
  fovs = 40,  # Default `40`. A numeric vector of field of view values.
  focal_distances = NULL,  # Default `NULL`, automatically the distance between positions and lookats. Numeric vector of focal distances.
  ortho_dims = NULL,  # Default `NULL`, which results in `c(1,1)` orthographic dimensions. A list or 2-column matrix of orthographic dimensions.
  camera_ups = NULL,  # Default `NULL`, which gives at up vector of `c(0,1,0)`. Camera up orientation.
  type = "cubic",  # type = "cubic",
  frames = 30,  # Default `30` frames.
  closed = FALSE,  # Default `FALSE`,
  aperture_linear = TRUE,  # Default `TRUE`,
  fov_linear = TRUE,  # Default `TRUE`,
  focal_linear = TRUE,  # Default `TRUE`,
  ortho_linear = TRUE,  # Default `TRUE`,
  constant_step = TRUE,  # Default `TRUE`,
  curvature_adjust = "none",  # curvature_adjust = "none",
  curvature_scale = 30,  # curvature_scale = 30,
  offset_lookat = 0,  # offset_lookat = 0,
  damp_motion = FALSE,  # damp_motion = FALSE,
  damp_magnitude = 0.1,  # damp_magnitude = 0.1,
  progress = TRUE  # progress = 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.
**generate_camera_motion**

- **type**: Default ‘cubic’. Type of transition between keyframes. Other options are ‘linear’, ‘quad’, ‘bezier’, ‘exp’, and ‘manual’. ‘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’. This will make the camera travel at a constant speed.

- **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 shaky 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.

**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
if(rayrender:::run_documentation()) {
  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)
}
if(rayrender:::run_documentation()) {
  # 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)
}
if(rayrender:::run_documentation()) {
  # 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)
}
if(rayrender:::run_documentation()) {
  # 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, fov=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=-10, scale_obj=3, angle=c(0,-45,0),
    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,-20),
    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=100,
    sample_method="sobol_blue",
    clamp_value=10, width=400, height=400)
}

generate_cornell

---

**Generate Cornell Box**

**Description**

Generate Cornell Box

**Usage**

```r
generate_cornell(
  light = TRUE,
  lightintensity = 5,
  lightcolor = "white",
  lightwidth = 332,
  lightdepth = 343,
  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.
**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

```r
# Generate and render the default Cornell box.
scene = generate_cornell()
if(rayrender:::run_documentation()) {
  render_scene(scene, samples=128, aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
}

# Make a much smaller light in the center of the room.
scene = generate_cornell(lightwidth=200, lightdepth=200)
render_scene(scene, samples=128, aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)

# Place a sphere in the middle of the box.
scene = scene %>%
  add_object(sphere(x=555/2, y=555/2, z=555/2, radius=555/4))
render_scene(scene, samples=128, aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)

# Reduce “fireflies” by setting a clamp_value in render_scene()
render_scene(scene, samples=128, aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE, clamp_value=3)

# Change the color scheme of the Cornell box
new_cornell = generate_cornell(leftcolor="purple", rightcolor="yellow")
render_scene(new_cornell, samples=128, 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

```r
generate_ground(
    depth = -1,
    spheresize = 1000,
    material = diffuse(color = "#ccff00")
)
```

Arguments

- **depth**
  - Default `-1`. Depth of the surface.
- **spheresize**
  - Default `1000`. Radius of the sphere representing the surface.
- **material**
  - Default `diffuse` with `color= "#ccff00"`. 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

```r
#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")))
if(rayrender:::run_documentation()) {
    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="grey50")) %>%
    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")))
if(rayrender:::run_documentation()) {
    render_scene(scene, parallel=TRUE,lookfrom=c(0,1,10))
}
```

generate_studio

Generate Studio

Description

Generates a curved studio backdrop.
Usage

```r
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

```r
# Generate the ground and add some objects
scene = generate_studio(depth=-1, material = diffuse(color="white")) %>%
  add_object(obj_model(r_obj(),y=-1,x=0.7,material=glossy(color="darkred"),angle=c(0,-20,0))) %>%
  add_object(sphere(x=-0.7,radius=0.5,material=dielectric())) %>%
  add_object(sphere(y=3,x=-2,z=20,material=light(intensity=600)))
if(rayrender::run_documentation()) {
  render_scene(scene, parallel=TRUE,lookfrom=c(0,2,10),fov=20,clamp_value=10,samples=128)
}

# Zooming out to show the full default scene
if(rayrender::run_documentation()) {
  render_scene(scene, parallel=TRUE,lookfrom=c(0,200,400),clamp_value=10,samples=128)
}
```
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**

```r
get_saved_keyframes()
```

**Value**

Data frame of keyframes

**Examples**

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

glossy  

*Glossy Material*

**Description**

Glossy Material

**Usage**

```r
glossy(
  color = "white",
  gloss = 1,
  reflectance = 0.05,
  microfacet = "tbr",
  checkercolor = NA,
  checkerperiod = 3,
  noise = 0,
  noise-phase = 0,
  noise-intensity = 10,
  noise-color = "#000000",
  gradient-color = NA,
  gradient-transpose = FALSE,
  gradient-point-start = NA,
  gradient-point-end = NA,
  gradient-type = "hsv",
  image-texture = NA,
)```
```r
glossy
image_repeat = 1,
alpha_texture = NA,
bump_texture = NA,
roughness_texture = NA,
bump_intensity = 1,
roughness_range = c(0e-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.

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

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 be-
tween 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 ‘NA’. 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 ‘NA’. A matrix or filename (specifying a greyscale image) to be used to specify the transparency.

bump_texture

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

roughness_texture

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

bump_intensity

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

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

```r
if(rayrender:::run_documentation()) {
  #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=128,sample_method="sobol_blue")
}
if(rayrender:::run_documentation()) {
  #Change the color of the underlying diffuse layer
  generate_ground(material=diffuse(sigma=90)) %>%
```

**group_objects**

Group Objects

**Description**

Group and transform objects together.

**Usage**

```r
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),
)```
args = 

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
if(rayrender:::run_documentation()) {
  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=128, 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=128, 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)))
hair

Hair Material

Description
Hair Material

Usage

```r
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pigment</td>
<td>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.</td>
</tr>
<tr>
<td>red_pigment</td>
<td>Default '0'. Concentration of the pheomelanin pigment in the hair. Pheomelanin makes red hair red.</td>
</tr>
</tbody>
</table>
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

```r
#Create a hairball
if(rayrender:::run_documentation()) {
  #Generate random 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=64, lookfrom=c(0,3,10),clamp_value = 10,
                 fov=20)
}
if(rayrender:::run_documentation()) {
  #Specify the color directly and increase hair roughness
  for(i in 1:length(phi)) {
```
}
lambertian = \left( \sin(\theta[i]) \times \sin(\phi[i]), \\
\cos(\theta[i]), \\
\sin(\theta[i]) \times \cos(\phi[i]) \right)

bez\_\text{er\_list}[i] = \text{bez\_\text{er\_curve}(width=0.01, width\_end=0.008,}

\begin{align*}
\text{p1} &= \text{pointval}, \\
\text{p2} &= (1+(\text{lengthval} \times 0.33)) \times \text{pointval}, \\
\text{p3} &= (1+(\text{lengthval} \times 0.66)) \times \text{pointval}, \\
\text{p4} &= (1+(\text{lengthval})) \times \text{pointval}, \\
\text{material} &= \text{hair(color="purple",}
\begin{align*}
\beta_m &= 0.5, \\
\beta_n &= 0.5), \\
\text{type} &= \text{"flat"}
\end{align*}
\}

\}

d\text{dplyr::bind\_rows(\text{bez\_er\_list})}
\text{generate\_ground(depth=-2, material=diffuse(color=\"grey20\")}) \rightarrow
\begin{align*}
\text{add\_object(sphere())} & \rightarrow
\text{add\_object(hairball)} \\
\text{add\_object(sphere(y=20, z=20, radius=5, material=light(color=\"white\", intensity = 100)))} & \rightarrow
\text{render\_scene(samples=64, lookfrom=c(0, 3, 10), clamp\_value = 10,}
\begin{align*}
\text{fov=20)
\end{align*}
\}

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.
if(rayrender:::run_documentation()) {
    scene = generate_cornell() \rightarrow
\begin{align*}
\text{add\_object(sphere(x=555/2, y=555/2, z=555/2, radius=555/8, material=lambertian()))} \\
\text{render\_scene(scene, lookfrom=c(278, 278, -800), lookat = c(278, 278, 0), samples=10,}
\begin{align*}
aperture=0, \text{fov=40, ambient\_light=FALSE, parallel=TRUE)}
\end{align*}
\}
}
**Light Material**

**Description**

Light Material

**Usage**

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

*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 ‘NA’. 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**

```r
#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()))
if(rayrender:::run_documentation()) {
  render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=128, 
              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)))
if(rayrender:::run_documentation()) {
  render_scene(scene, lookfrom=c(278,278,-800),lookat = c(278,278,0), samples=128, 
              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=0.5,material=light(intensity=5,color="red"))) %>%
  add_object(obj_model(r_obj(), z=-3,x=-1.5,y=-1, angle=c(0,45,0))) %>%
  add_object(pig(scale=0.3, x=1.5,z=-2,y=-1.5,angle=c(0,-135,0)))
if(rayrender:::run_documentation()) {
  render_scene(scene, samples=128, parallel=TRUE, clamp_value=10)
}

### mesh3d_model

**Description**

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

**Usage**

```r
mesh3d_model(
  mesh,
  x = 0,
  y = 0,
  z = 0,
  swap_yz = FALSE,
  reverse = FALSE,
  scale_mesh = 1,
  verbose = 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.
The document contains information about a mesh3d model in the scene, including parameters such as `scale_mesh`, `verbose`, `override_material`, `material`, `angle`, `order_rotation`, `flipped`, and `scale`. It also includes an example of loading a mesh3d object and rendering it.

```
#Load a mesh3d object (from the Rvcg) and render it:
if(length(find.package("Rcvg", quiet=TRUE)) > 0) {
  library(Rvcg)
  data(humface)

  generate_studio() %>%
    add_object(mesh3d_model(humface,y=-0.3,x=0,z=0,
                            material=glossy(color="dodgerblue4"), scale_mesh = 1/70)) %>%
    add_object(sphere(y=5,x=5,z=5,material=light(intensity=50))) %>%
    render_scene(samples=128,width=800,height=800,
                 lookat = c(0,0.5,1), aperture=0.0)
}
```

### Value

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

### Examples

```
#Load a mesh3d object (from the Rvcg) and render it:
if(length(find.package("Rcvg", quiet=TRUE)) > 0) {
  library(Rvcg)
  data(humface)

  generate_studio() %>%
    add_object(mesh3d_model(humface,y=-0.3,x=0,z=0,
                            material=glossy(color="dodgerblue4"), scale_mesh = 1/70)) %>%
    add_object(sphere(y=5,x=5,z=5,material=light(intensity=50))) %>%
    render_scene(samples=128,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 = NA,
  image_repeat = 1,
  alpha_texture = NA,
  bump_texture = NA,
  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.
metal

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 ‘NA’. 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 ‘NA’. A matrix or filename (specifying a greyscale image) to be used to specify the transparency.

bump_texture  Default ‘NA’. 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 # 880nm (R), 530nm (G), and 430nm (B).
scene = generate_cornell() %>%
```r
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))))
if(rayrender:::run_documentation()) {
  render_scene(scene, lookfrom=c(278,278,-800), lookat = c(278,278,0), samples=50, 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)))
if(rayrender:::run_documentation()) {
  render_scene(scene, lookfrom=c(278,278,-800), lookat = c(278,278,0), samples=128, 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)))
if(rayrender:::run_documentation()) {
  render_scene(scene, lookfrom=c(278,278,-800), lookat = c(278,278,0), samples=128, 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)))
if(rayrender:::run_documentation()) {
  render_scene(scene2, lookfrom=c(278,278,-800), lookat = c(278,278,0), samples=128, aperture=0, fov=40, ambient_light=FALSE, parallel=TRUE)
}
```

---

**Description**

Microfacet Material

**Usage**

```r
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, gradient_point_end = NA, gradient_type = "hsv", image_texture = NA, image_repeat = 1, alpha_texture = NA, bump_texture = NA, bump_intensity = 1, roughness_texture = NA, 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).
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>microfacet</td>
<td>‘tbr’</td>
<td>Type of microfacet distribution. Alternative option ‘beckmann’.</td>
</tr>
<tr>
<td>checkercolor</td>
<td>‘NA’</td>
<td>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’.</td>
</tr>
<tr>
<td>checkerperiod</td>
<td>‘3’</td>
<td>The period of the checker pattern. Increasing this value makes the checker pattern bigger, and decreasing it makes it smaller.</td>
</tr>
<tr>
<td>noise</td>
<td>‘0’</td>
<td>If not ‘0’, covers the surface in a turbulent marble pattern. This value will determine the amount of turbulence in the texture.</td>
</tr>
<tr>
<td>noisephase</td>
<td>‘0’</td>
<td>The phase of the noise. The noise will repeat at ‘360’.</td>
</tr>
<tr>
<td>noiseintensity</td>
<td>‘10’</td>
<td>Intensity of the noise.</td>
</tr>
<tr>
<td>noisecolor</td>
<td>‘#000000’</td>
<td>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’.</td>
</tr>
<tr>
<td>gradient_color</td>
<td>‘NA’</td>
<td>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’.</td>
</tr>
<tr>
<td>gradient_transpose</td>
<td>‘FALSE’</td>
<td>If ‘TRUE’, this will use the ‘v’ coordinate texture instead of the ‘u’ coordinate texture to map the gradient.</td>
</tr>
<tr>
<td>gradient_point_start</td>
<td>‘NA’</td>
<td>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’.</td>
</tr>
<tr>
<td>gradient_point_end</td>
<td>‘NA’</td>
<td>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’.</td>
</tr>
<tr>
<td>gradient_type</td>
<td>‘hsv’</td>
<td>Colorspace to calculate the gradient. Alternative ‘rgb’.</td>
</tr>
<tr>
<td>image_texture</td>
<td>‘NA’</td>
<td>A 3-layer RGB array or filename to be used as the texture on the surface of the object.</td>
</tr>
<tr>
<td>image_repeat</td>
<td>‘1’</td>
<td>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.</td>
</tr>
<tr>
<td>alpha_texture</td>
<td>‘NA’</td>
<td>A matrix or filename (specifying a greyscale image) to be used to specify the transparency.</td>
</tr>
<tr>
<td>bump_texture</td>
<td>‘NA’</td>
<td>A matrix, array, or filename (specifying a greyscale image) to be used to specify a bump map for the surface.</td>
</tr>
<tr>
<td>bump_intensity</td>
<td>‘1’</td>
<td>Intensity of the bump map. High values may lead to unphysical results.</td>
</tr>
<tr>
<td>roughness_texture</td>
<td>‘NA’</td>
<td>A matrix, array, or filename (specifying a greyscale image) to be used to specify a roughness map for the surface.</td>
</tr>
</tbody>
</table>
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).
if(rayrender:::run_documentation()) {
  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=128,
    aperture=0, fov=40, parallel=TRUE,clamp_value=10)
}

if(rayrender:::run_documentation()) {
  #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=128,
    aperture=0, fov=40, parallel=TRUE,clamp_value=10)
}

if(rayrender:::run_documentation()) {
  #Render a rough silver R with a smaller golden egg in front
  generate_cornell() %>%
    add_object(obj_model(r_obj(),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,}


**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**

```r
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,
  importance_sample_lights = TRUE,
```
obj_model

```r
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.
- **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.

if(rayrender:::run_documentation()) {
  generate_ground(material = diffuse(checkercolor = "grey50")) %>%
  add_object(obj_model(y = -0.8, filename = r_obj(),
    material = microfacet(color = "gold", roughness = 0.05))) %>%
  add_object(obj_model(x = 1.8, y = -0.8, filename = r_obj(),
    material = diffuse(color = "dodgerblue"))) %>%
  add_object(obj_model(x = -1.8, y = -0.8, filename = r_obj(),
    material = dielectric(attenuation = c(1,0.3,1)*2))) %>%
  add_object(sphere(z = 20, x = 20, y = 20, radius = 10,
    material = light(intensity = 10))) %>%
  render_scene(parallel = TRUE, samples = 128, aperture = 0.05,
    fov = 32, lookfrom = c(0, 2, 10))
}

# Use scale_obj to make objects bigger--this is more robust than the generic scale argument.
if(rayrender:::run_documentation()) {
  generate_ground(material = diffuse(checkercolor = "grey50")) %>%
  add_object(obj_model(y = -0.8, filename = r_obj(), scale_obj = 2,
    material = diffuse(noise = TRUE, noiseintensity = 10,noisephase=45))) %>%
  add_object(sphere(z = 20, x = 20, y = 20, radius = 10,
    material = light(intensity = 10))) %>%
  render_scene(parallel = TRUE, samples = 128, ambient = TRUE,
    backgroundhigh="blue", backgroundlow="red",
    aperture = 0.05, fov = 32, lookfrom = c(0, 2, 10),
    lookat = c(0,1,0))
}

---

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,
)
path

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

```r
if(rayrender:::run_documentation()) {
  #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=128, clamp_value=10,fov=30)
}
if(rayrender:::run_documentation()) {
  #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=128, clamp_value=10,fov=30)
}
if(rayrender:::run_documentation()) {
  #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=128, clamp_value=10)
}
if(rayrender:::run_documentation()) {
  #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=128, clamp_value=10)
}
```
render_scene(samples=128, clamp_value=10)
}
if(rayrender:::run_documentation()) {
# 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(0.6,-0.5,-0.1),c(0.5,0.5,-0.1), c(0.3,-0.1),c(-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=128, clamp_value=10)
}
if(rayrender:::run_documentation()) {
# 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=128, clamp_value=10)
}
if(rayrender:::run_documentation()) {
# 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=128, clamp_value=10)
}
if(rayrender:::run_documentation()) {
# 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=128, clamp_value=10, lookfrom=c(0,3,10))
}
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
)

Arguments

x Default '0'. x-coordinate of the center of the pig.
y Default '0'. y-coordinate of the center of the pig.
z Default '0'. z-coordinate of the center of the pig.
emotion Default 'neutral'. Other options include ‘skeptical’, ‘worried’, and ‘angry’.
spider Default 'FALSE'. Spiderpig.
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)'. Scale transformation in the x, y, and z directions. If this is a
  single value, number, the object will be scaled uniformly.
diffuse_sigma Default '0'. Controls the Oren-Nayar sigma parameter for the pig’s diffuse ma-
  terial.

Value

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

Examples

#Generate a pig in the cornell box.
if(rayrender::run_documentation()) {
  generate_cornell() %>%
    add_object(pig(x=555/2,z=555/2,y=120,
                   scale=c(80,80,80), angle = c(0,135,0))) %>
    render_scene(parallel=TRUE, samples=128,clamp_value=10)
}
if(rayrender::run_documentation()) {
  # Show the pig staring into a mirror, worried
  generate_cornell() %>%
    add_object(pig(x=555/2-70,z=555/2+50,y=120, scale=c(80,80,80),
                   angle = c(90,0,0)),
                   diffuse_sigma = 0)
}

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,
  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.
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).
render_animation

Usage

render_animation(
  scene,
  camera_motion,
  start_frame = 1,
  end_frame = NA,
  width = 400,
  height = 400,
  preview = interactive(),
  camera_description_file = NA,
  camera_scale = 1,
  iso = 100,
  film_size = 22,
  samples = 100,
  min_variance = 5e-05,
  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,
  focal_distance = NULL,
  ortho_dimensions = c(1, 1),
  tonemap = "gamma",
  bloom = TRUE,
  parallel = TRUE,
  bvh_type = "sah",
  environment_light = NULL,
  rotate_env = 0,
  intensity_env = 1,
  debug_channel = "none",
  return_raw_array = FALSE,
  progress = interactive(),
  verbose = FALSE,
  preview_light_direction = c(0, -1, 0),
  preview_exponent = 6
)
# Arguments

- **scene**: Tibble of object locations and properties.
- **camera_motion**: Data frame of camera motion vectors, calculated with `generate_camera_motion()`.
- **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.
- **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 '0.00005'. 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 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 'background-high'.
- **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 ‘NULL’. 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.

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 ‘gamma’. Choose the tone mapping function. Default ‘gamma’ solely adjusts for gamma and clamps values greater than 1 to 1. ‘reinhold’ 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. Note: If set to anything other than ‘gamma’, objects with material ‘light()’ may not be anti-aliased. If ‘raw’, the raw array of HDR values will be returned, rather than an image or a plot.

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 both HDR (‘.hdr’) and low-dynamic range (‘.png’, ‘.jpg’) images.
rotate_env  Default '0'. The number of degrees to rotate the environment map around the scene.

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 'v' 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.

return_raw_array  Default 'FALSE'. If 'TRUE', function will return raw array with RGB intensity information.

progress  Default 'TRUE' if interactive session, 'FALSE' otherwise.

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

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.

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
if(rayrender:::run_documentation()) {
  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)
}
if(rayrender:::run_documentation()) {
#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)
}
if(rayrender:::run_documentation()) {
#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)
}
if(rayrender:::run_documentation()) {
#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=-10,scale_obj=3, angle=c(0,-45,0),
  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(30,225,30),
emotion="angry", spider=TRUE)) %>
render_animation(filename = NA, camera_motion = camera_motion, samples=100,
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 = NULL,
  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 'NULL'. 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
if(rayrender:::run_documentation()) {
  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(y=-1,x=-1.8,r_obj(), angle=c(0,135,0),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=128, aperture=0.1,
      lookfrom=c(-0.9,1.2,-4.5),lookat=c(0,-1,0))
}

if(rayrender:::run_documentation()) {
  #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(y=-1,x=-1.8,r_obj(), angle=c(0,135,0),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=128, aperture=0.1,
lookfrom=c(-0.9,1.2,-4.5),lookat=c(0,-1,0))
}

if(rayrender:::run_documentation()) {
#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(y=-1,x=-1.8,r_obj(), angle=c(0,135,0),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=128, aperture=0.1,
lookfrom=c(-0.9,1.2,-4.5),lookat=c(0,-1,0))
}

if(rayrender:::run_documentation()) {
#Turn on colors
generate_ground(material = diffuse(color="grey20", checkercolor = "grey50",sigma=90)) %>%
add_object(sphere(material=metal())) %>%
add_object(obj_model(y=-1,x=-1.8,r_obj(), angle=c(0,135,0),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=128, 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)`
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,
)

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

if(rayrender:::run_documentation()) {
  generate_ground(material=diffuse(color="darkgreen")) %>%
    add_object(sphere(material=diffuse(checkercolor="red"))) %>%
    render_preview()
}

if(rayrender:::run_documentation()) {
  # 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))
}

if(rayrender:::run_documentation()) {
  # 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)
}
camera_description_file = NA,
preview = interactive(),
interactive = TRUE,
camera_scale = 1,
iso = 100,
film_size = 22,
min_variance = 5e-05,
min_adaptive_size = 8,
sample_method = "sobol",
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 = NULL,
backgroundhigh = "#80b4ff",
backgroundlow = "#ffffff",
shutteropen = 0,
shutterclose = 1,
focal_distance = NULL,
ortho_dimensions = c(1, 1),
tonemap = "gamma",
bloom = TRUE,
parallel = TRUE,
bvh_type = "sah",
environment_light = NULL,
rotate_env = 0,
intensity_env = 1,
debug_channel = "none",
return_raw_array = FALSE,
progress = interactive(),
verbose = FALSE,
new_page = TRUE
)

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 ‘TRUE’. Whether to display a real-time progressive preview of the render. Press ESC to cancel the render.

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, 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, R = Reset Camera, TAB: Toggle Orbit Mode, Left Mouse Click: Change Look Direction, Right Mouse Click: Change Look At 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) F: Toggle Fast Travel Mode

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

Note: Clicking on the environment image will only redirect the view direction, not change the orbit point. Some options aren’t available all cameras. When using a realistic camera, the aperture and field of view cannot be changed from their initial settings. Additionally, clicking to direct the camera at the background environment image while using a realistic camera will not always point to the exact position selected.

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).
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>max_depth</td>
<td>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).</td>
<td></td>
</tr>
<tr>
<td>roulette_active_depth</td>
<td>Default '100'. Number of ray bounces until a ray can stop bouncing via Russian roulette.</td>
<td></td>
</tr>
<tr>
<td>ambient_light</td>
<td>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).</td>
<td></td>
</tr>
<tr>
<td>lookfrom</td>
<td>Default 'c(0,1,10)'. Location of the camera.</td>
<td></td>
</tr>
<tr>
<td>lookat</td>
<td>Default 'c(0,0,0)'. Location where the camera is pointed.</td>
<td></td>
</tr>
<tr>
<td>camera_up</td>
<td>Default 'c(0,1,0)'. Vector indicating the &quot;up&quot; position of the camera.</td>
<td></td>
</tr>
<tr>
<td>aperture</td>
<td>Default '0.1'. Aperture of the camera. Smaller numbers will increase depth of field, causing less blurring in areas not in focus.</td>
<td></td>
</tr>
<tr>
<td>clamp_value</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>filename</td>
<td>Default 'NULL'. If present, the renderer will write to the filename instead of the current device.</td>
<td></td>
</tr>
<tr>
<td>backgroundhigh</td>
<td>Default '#80b4ff'. The &quot;high&quot; color in the background gradient. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between '0' and '1'.</td>
<td></td>
</tr>
<tr>
<td>backgroundlow</td>
<td>Default '#ffffff'. The &quot;low&quot; color in the background gradient. Can be either a hexadecimal code, or a numeric rgb vector listing three intensities between '0' and '1'.</td>
<td></td>
</tr>
<tr>
<td>shutteropen</td>
<td>Default '0'. Time at which the shutter is open. Only affects moving objects.</td>
<td></td>
</tr>
<tr>
<td>shutterclose</td>
<td>Default '1'. Time at which the shutter is open. Only affects moving objects.</td>
<td></td>
</tr>
<tr>
<td>focal_distance</td>
<td>Default 'NULL', automatically set to the 'lookfrom-lookat' distance unless otherwise specified.</td>
<td></td>
</tr>
<tr>
<td>ortho_dimensions</td>
<td>Default 'c(1,1)'. Width and height of the orthographic camera. Will only be used if 'fov = 0'.</td>
<td></td>
</tr>
<tr>
<td>tonemap</td>
<td>Default 'gamma'. Choose the tone mapping function, Default 'gamma' solely adjusts for gamma and clamps values greater than 1 to 1. 'reinhold' 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. If 'raw', the raw array of HDR values will be returned, rather than an image or a plot.</td>
<td></td>
</tr>
<tr>
<td>bloom</td>
<td>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</td>
<td></td>
</tr>
</tbody>
</table>
**render_scene**

The `render_scene` function allows for 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 both HDR (‘.hdr’) and low-dynamic range (‘.png’, ‘.jpg’) images.

- **rotate_env**
  Default ‘0’. The number of degrees to rotate the environment map around the scene.

- **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 ‘v’ coordinates. If ‘color’, function will return the raw albedo values (with white for ‘metal’ and ‘dielectric’ materials).

- **return_raw_array**
  Default ‘FALSE’. If ‘TRUE’, function will return raw array with RGB intensity information.

- **progress**
  Default ‘TRUE’ if interactive session, ‘FALSE’ otherwise.

- **verbose**
  Default ‘FALSE’. Prints information and timing information about scene construction and raytracing progress.

- **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).

**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**

```r
# Generate a large checkered sphere as the ground
if(rayrender:::run_documentation()) {
  scene = generate_ground(depth=-0.5, material = diffuse(color="white", checkercolor="darkgreen"))
  render_scene(scene, parallel=TRUE, samples=128, sample_method="sobol")
}
```

if(rayrender:::run_documentation()) {

}
#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=128)
}
if(rayrender:::run_documentation()) {
  #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=128)
}
if(rayrender:::run_documentation()) {
  #Add a metallic gold sphere, using stratified sampling for a higher quality render
  scene = scene %>%
    add_object(sphere(x=-1.1,y=0,z=0, radius=0.5, material = metal(color="gold",fuzz=0.1)))
  render_scene(scene,fov=20,parallel=TRUE,samples=128)
}
if(rayrender:::run_documentation()) {
  #Lower the number of samples to render more quickly (here, we also use only one core).
  render_scene(scene, samples=4, parallel=FALSE)
}
if(rayrender:::run_documentation()) {
  #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(0,180,0),
                  material = diffuse(image_texture = image_array)))
  render_scene(scene,fov=20,parallel=TRUE,samples=128)
}
if(rayrender:::run_documentation()) {
  #Move the camera
  render_scene(scene,lookfrom = c(7,1.5,10),lookat = c(0,0.5,0),fov=15,parallel=TRUE)
}
if(rayrender:::run_documentation()) {
  #Change the background gradient to a night time ambiance
  render_scene(scene,lookfrom = c(7,1.5,10),lookat = c(0,0.5,0),fov=15,
               backgroundhigh = "#282375", backgroundlow = "#7e77ea", parallel=TRUE,
               samples=128)
}
if(rayrender:::run_documentation()) {
  #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 = 0.5,parallel=TRUE,samples=128)
}
if(rayrender:::run_documentation()) {
  #We can also capture a 360 environment image by setting `fov = 360` (can be used for VR)
  generate_cornell() %>%
render_scene

```r
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")))
%>
r
render_scene(lookfrom=c(278,278,30), lookat=c(278,278,500), clamp_value=10,
    fov = 360, samples = 128, width=800, height=800)
```

```r
if(rayrender:::run_documentation()) {
  # We can also use a realistic camera by specifying a camera description file (several of which
  # are built-in to rayrender. Note the curvature introduced by the fisheye lens:
  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")))
%>
r
render_scene(lookfrom=c(278,278,30), lookat=c(278,278,500), clamp_value=10,
    aperture=1, iso = 100000,
    camera_description_file = "fisheye", samples = 128, width=800, height=400)
```

```r
if(rayrender:::run_documentation()) {
  # 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=1:30
xpos = 10 * sin(t*12*pi/180+pi/2)
zpos = 10 * cos(t*12*pi/180+pi/2)
# Save old par() settings
old.par = par(no.readonly = TRUE)
on.exit(par(old.par))
par(mfrow=c(5,6))
for(i in 1:30) {
  render_scene(scene, samples=16,
    lookfrom = c(xpos[i],1.5,zpos[i]),lookat = c(0,0.5,0), parallel=TRUE)
}
}
**r_obj**

**R 3D Model**

**Description**

3D obj model of the letter R, to be used with ‘obj_model()’

**Usage**

```r
r_obj()
```

**Value**

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

**Examples**

```r
if(rayrender:::run_documentation()) {
  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(), y = -1, material = diffuse(color="red"))) %>%
  render_scene(parallel=TRUE, lookfrom = c(0, 1, 10), clamp_value = 5, samples = 200)
}
```

---

**segment**

**Segment Object**

**Description**

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

**Usage**

```r
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, specifying ‘x’, ‘y’, ‘z’.
end   Default ‘c(0, 1, 0)’. End point of the cylinder segment, specifying ‘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

```r
#Generate a segment in the cornell box.
if(rayrender:::run_documentation()) {
  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 = 128, 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)
if(rayrender:::run_documentation()) {
  generate_cornell() %>%
    add_object(scene_segments) %>%
    render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
                 ambient_light = FALSE, samples = 128, 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)) %>%
  add_object(segment(start = c(100, 455, 100), end = c(455, 455, 455), radius = 10)) %>%

if(rayrender:::run_documentation()) {
  generate_cornell() %>%
    add_object(cube_outline) %>%
    render_scene(lookfrom = c(278, 278, -800) ,lookat = c(278, 278, 0), fov = 40,
                 ambient_light = FALSE, samples = 128, parallel = TRUE, clamp_value = 5)
}

#Shrink and rotate the cube

if(rayrender:::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}

---

**sphere**  

**Sphere Object**

---

**Description**

Sphere Object

**Usage**

sphere(
  x = 0,
)
sphere

```r
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**

```r
#Generate a sphere in the cornell box.
if(rayrender:::run_documentation()) {
  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 = 128, clamp_value = 5)
}

#Generate a gold sphere in the cornell box
if(rayrender:::run_documentation()) {
  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 = 128, clamp_value = 5)
}
```
text3d

Text Object

Description

Text Object

Usage

text3d(
  label,
  x = 0,
  y = 0,
  z = 0,
  text_height = 1,
  orientation = "xy",
  material = diffuse(),
  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**.
- **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

```r
# Generate a label in the cornell box.
if(rayrender:::run_documentation()) {
  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(0,180,0))) %>%
    render_scene(samples=128, clamp_value=10)
}
if(rayrender:::run_documentation()) {
  # Change the orientation
  generate_cornell() %>%
    add_object(text3d(label="YZ Plane", x=550,y=555/2,z=555/2,text_height=100,
                      orientation = "yz",
                      material=diffuse(color="grey10"), angle=c(0,180,0))) %>%
    add_object(text3d(label="XY Plane", z=550,y=555/2,x=555/2,text_height=100,
                      orientation = "xy",
                      material=diffuse(color="grey10"), angle=c(0,180,0))) %>%
    add_object(text3d(label="XZ Plane", z=555/2,y=5,x=555/2,text_height=100,
                      orientation = "xz",
                      material=diffuse(color="grey10"))) %>%
    render_scene(samples=128, clamp_value=10)
}
if(rayrender:::run_documentation()) {
  # 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=60,
                      material=diffuse(color="grey10"), angle=c(0,180,0))) %>%
    add_object(text3d(label="Sphere", x=555/2,y=100,z=100,text_height=30,
                      material=diffuse(color="white"), angle=c(0,180,0))) %>%
    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=128, clamp_value=10)
}
if(rayrender:::run_documentation()) {
  # 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(0,180,0))
  }
  bees = do.call(rbind,bee_list)
  generate_cornell() %>%
    add_object(bees) %>%
    render_scene(samples=128, clamp_value=10)
}
```

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

```r
#Generate a triangle in the Cornell box.
if(rayrender:::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}

#Pass individual colors to each vertex:
if(rayrender:::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}
```
xy_rect

Rectangular XY Plane Object

Description
Rectangular XY Plane Object

Usage

```r
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.
if(rayrender::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}

# Generate a gold plane in the cornell box
if(rayrender::run_documentation()) {
  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 = 128, 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.
yz_rect

Rectangular YZ Plane Object

Description

Rectangular YZ Plane Object

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ywidth</td>
<td>Default ‘1’. z-width of the rectangle.</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>Default ‘0’. y-coordinate of the center of the rectangle.</td>
<td></td>
</tr>
<tr>
<td>material</td>
<td>Default diffuse. The material, called from one of the material functions diffuse, metal, or dielectric.</td>
<td></td>
</tr>
<tr>
<td>angle</td>
<td>Default ‘c(0, 0, 0)’. Angle of rotation around the x, y, and z axes, applied in the order specified in ‘order_rotation’.</td>
<td></td>
</tr>
<tr>
<td>order_rotation</td>
<td>Default ‘c(1, 2, 3)’. The order to apply the rotations, referring to “x”, “y”, and “z”.</td>
<td></td>
</tr>
<tr>
<td>flipped</td>
<td>Default ‘FALSE’. Whether to flip the normals.</td>
<td></td>
</tr>
<tr>
<td>scale</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

Value

Single row of a tibble describing the XZ plane in the scene.

Examples

#Generate a purple rectangle in the cornell box.
if(rayrender:::run_documentation()) {
  generate_cornell() %>%
    add_object(xz_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 = 128, parallel = TRUE, clamp_value = 5)
}

#Generate a gold plane in the cornell box
if(rayrender:::run_documentation()) {
  generate_cornell() %>%
    add_object(xz_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 = 128, parallel = TRUE, clamp_value = 5)
}
Usage

```r
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

```r
#Generate a purple rectangle in the cornell box.
if(rayrender:::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}
#Generate a gold plane in the cornell box
```
if(rayrender::run_documentation()) {
  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 = 128, parallel = TRUE, clamp_value = 5)
}
Index

add_object, 3
animate_objects, 4
arrow, 6
bezier_curve, 9
cone, 12
csg_box, 14
csg_capsule, 15
csg_combine, 16
csg_cone, 18
csg_cylinder, 19
csg_ellipsoid, 21
csg_elongate, 22
csg_group, 24
csg_object, 24
csg_onion, 26
csg_triangle, 28
csg_pyramid, 29
csg_rotate, 30
csg_round, 31
csg RoundedRectangle, 32
csg_scale, 34
csg_sphere, 35
csg_torus, 36
csg_translate, 37
csg_triangle, 38
cube, 39
cylinder, 41
disk, 48
ellipsoid, 49
extruded_path, 51
extruded_polygon, 57
generate_camera_motion, 60
generate_cornell, 64
generate_ground, 65
generate_studio, 66
get_saved_keyframes, 68
glossy, 68
group_objects, 71
hair, 73
lambertian, 75
light, 76
mesh3d_model, 78
microfacet, 82
obj_model, 86
path, 88
pig, 91
ply_model, 93
r_obj, 111
render_animation, 94
render_ao, 100
render_preview, 103
render_scene, 104
segment, 111
sphere, 113
text3d, 115
triangle, 117
xy_rect, 119
xz_rect, 120
yz_rect, 121