

Package: raymolecule (via r-universe)

September 24, 2026

Type Package

Title Parse and Render Molecular Structures in 3D

Version 0.9.0

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

Description Downloads and parses 'SDF' (Structural Description Format) and 'PDB' (Protein Database) files for 3D rendering.

License GPL-3

Encoding UTF-8

Depends R (>= 4.1)

Imports rayrender (>= 0.41.3), PeriodicTable, httr, rayvertex (>= 0.15.1)

Suggests testthat (>= 3.0.0), skymodelr, rayimage

Config/testthat/edition 3

LazyData true

URL <http://www.raymolecule.com/>,
<https://github.com/tylermorganwall/raymolecule>

BugReports <https://github.com/tylermorganwall/raymolecule/issues>

Remotes tylermorganwall/rayrender, tylermorganwall/rayvertex,
tylermorganwall/skymodelr

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev libgdal-dev gdal-bin libgeos-dev make libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libwebp-dev libssl-dev libproj-dev libsqlite3-dev

Repository <https://tylermorganwall.r-universe.dev>

Date/Publication 2026-06-25 04:00:22 UTC

RemoteUrl <https://github.com/tylermorganwall/raymolecule>

RemoteRef HEAD

RemoteSha 1f12ebede77ed2391671e93586b7c77dbfe3aa48

Contents

download_pdb	2
generate_atom_scene	3
generate_bond_scene	5
generate_full_scene	7
generate_ribbon_scene	9
get_example_molecule	15
get_molecule	16
read_pdb	17
read_sdf	18
render_model	19

Index	23
--------------	-----------

download_pdb	<i>Download a PDB Structure from RCSB</i>
--------------	---

Description

Looks up a protein name in the RCSB PDB search API and downloads the best matching legacy PDB file. A 4-character PDB ID can also be supplied directly.

Usage

```
download_pdb(
    protein,
    out_dir = ".",
    filename = NULL,
    overwrite = FALSE,
    max_results = 10L,
    verbose = FALSE
)
```

Arguments

protein	Protein name or 4-character PDB ID.
out_dir	Default ‘.’. Directory where the PDB file will be written.
filename	Default ‘NULL’. Optional output file name. If ‘NULL’, the downloaded file is named with the matched PDB ID.
overwrite	Default ‘FALSE’. Whether to overwrite an existing file.
max_results	Default ‘10L’. Maximum number of RCSB search matches to try when downloading a legacy PDB file.
verbose	Default ‘FALSE’. If ‘TRUE’, report the matched PDB ID and destination path.

Value

Path to the downloaded PDB file.

Examples

```
# Start with a direct PDB ID download. A temporary directory, custom
# filename, overwrite flag, and verbose output make the file handling clear.
pdb_file = download_pdb(
    "4fsp",
    out_dir = tempdir(),
    filename = "outer-membrane-barrel.pdb",
    overwrite = TRUE,
    verbose = TRUE
)

read_pdb(pdb_file, verbose = TRUE) |>
  generate_ribbon_scene() |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey12"
  )

# Protein-name lookup searches RCSB and tries up to three legacy PDB matches.
hemoglobin_file = download_pdb(
  "hemoglobin",
  out_dir = tempdir(),
  max_results = 3L
)
read_pdb(hemoglobin_file, verbose = TRUE) |>
  generate_ribbon_scene() |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey12"
  )
```

generate_atom_scene	<i>Build Scene (atoms only)</i>
---------------------	---------------------------------

Description

Builds a scene containing only atom spheres from a PDB or SDF model.

Usage

```
generate_atom_scene(
  model,
  x = 0,
  y = 0,
```

```

    z = 0,
    scale = 1,
    center = TRUE,
    material = rayrender::glossy,
    material_args = list(),
    material_vertex = material_list(type = "phong")
)

```

Arguments

<code>model</code>	Model extracted from a PDB or SDF file.
<code>x</code>	Default '0'. X offset, applied after centering.
<code>y</code>	Default '0'. Y offset, applied after centering.
<code>z</code>	Default '0'. Z offset, applied after centering.
<code>scale</code>	Default '1'. Amount to scale the inter-atom spacing.
<code>center</code>	Default 'TRUE'. Centers the bounding box of the model.
<code>material</code>	Default 'rayrender::glossy'. Optional rayrender material used to initialize the mesh material when 'material_vertex' is not supplied. Must be either 'glossy', 'diffuse', or 'dielectric'.
<code>material_args</code>	Default 'list()'. Named list of additional arguments passed to 'material'. Arguments supplied by raymolecule for colors and textures override entries with the same names. For example, use 'list(gloss = 0.35, reflectance = 0.12)' with 'rayrender::glossy', or 'list(sigma = 0.4)' with 'rayrender::diffuse'.
<code>material_vertex</code>	Default 'rayvertex::material_list()'. Mesh material. 'diffuse'/'ambient' colors and 'ambient_intensity' are determined automatically, but all other material properties can be changed.

Value

Raymesh scene containing only the atoms in a molecule/protein.

Examples

```

atom_model = read_sdf(get_example_molecule("benzene"))

# Start with a centered raster atom scene using the default atom colors.
atom_model |>
  generate_atom_scene() |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey15"
  )

# This version shifts and scales the same atoms, then uses a diffuse mesh
# material for the spheres before pathtracing.

```

```
atom_model |>
  generate_atom_scene(
    x = -1,
    y = 0,
    z = 1,
    scale = 0.75,
    center = TRUE,
    material = rayrender::diffuse,
    material_args = list(sigma = 0.4)
  ) |>
  render_model(pathtrace = TRUE, width = 800, height = 800, samples = 32)

# The toon material changes only the raster shader. Keeping `center = TRUE`
# makes the result easy to compare to the baseline scene above.
toon_material = rayvertex::material_list(
  type = "toon_phong",
  toon_levels = 3,
  toon_outline_width = 0.08
)
caffeine_model = read_sdf(get_example_molecule("caffeine"))
caffeine_model |>
  generate_atom_scene(
    center = TRUE,
    material_vertex = toon_material
  ) |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey15"
  )
```

generate_bond_scene	<i>Build Scene (bonds only)</i>
---------------------	---------------------------------

Description

Builds a scene containing only bond geometry from a PDB or SDF model.

Usage

```
generate_bond_scene(
  model,
  x = 0,
  y = 0,
  z = 0,
  scale = 1,
  center = TRUE,
  force_single_bonds = FALSE,
```

```

material = rayrender::glossy,
material_args = list(),
material_vertex = material_list(diffuse = "grey33", ambient = "grey33", type = "phong",
    ambient_intensity = 0.3)
)

```

Arguments

<code>model</code>	Model extracted from a PDB or SDF file.
<code>x</code>	Default '0'. X offset, applied after centering.
<code>y</code>	Default '0'. Y offset, applied after centering.
<code>z</code>	Default '0'. Z offset, applied after centering.
<code>scale</code>	Default '1'. Amount to scale the interatom spacing.
<code>center</code>	Default 'TRUE'. Centers the bounding box of the model.
<code>force_single_bonds</code>	Default 'FALSE'. Whether to force all bonds to show as a single connection.
<code>material</code>	Default 'rayrender::glossy'. Optional rayrender material used to initialize the mesh material when 'material_vertex' is not supplied. Must be either 'glossy', 'diffuse', or 'dielectric'.
<code>material_args</code>	Default 'list()'. Named list of additional arguments passed to 'material'. Arguments supplied by raymolecule for colors and textures override entries with the same names. For example, use 'list(gloss = 0.35, reflectance = 0.12)' with 'rayrender::glossy', or 'list(sigma = 0.4)' with 'rayrender::diffuse'.
<code>material_vertex</code>	Default 'material_list(diffuse="grey33",ambient="grey33",type="phong",ambient_intensity=0.3)'. Mesh material to use for the bonds.

Value

Raymesh scene containing only the connections between atoms in a molecule/protein.

Examples

```

bond_model = read_sdf(get_example_molecule("benzene"))

# Start with a centered bond scene using the molecule's recorded bond
# orders and default bond material.
bond_model |>
  generate_bond_scene() |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey10"
  )

# This version reduces the spacing, forces every connection to a single
# bond, and lights the ligand from above and below while pathtracing.

```

```
bond_model |>
  generate_bond_scene(
    x = 0,
    y = 0,
    z = 0,
    scale = 0.7,
    center = TRUE,
    force_single_bonds = TRUE,
    material = rayrender::glossy,
    material_args = list(gloss = 0.35)
  ) |>
  render_model(
    pathtrace = TRUE,
    lights = "both",
    width = 800,
    height = 800,
    samples = 32
  )

# A custom rayvertex material changes the raster bond color and ambient
# contribution while keeping the same geometry.
bond_material = rayvertex::material_list(
  diffuse = "grey85",
  ambient = "grey25",
  type = "phong",
  ambient_intensity = 0.4
)
cinnamaldehyde_model = read_sdf(get_example_molecule("cinnamaldehyde"))
cinnamaldehyde_model |>
  generate_bond_scene(
    material_vertex = bond_material
  ) |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey10"
  )
```

generate_full_scene	<i>Build Scene (bonds + atoms)</i>
---------------------	------------------------------------

Description

Builds a combined atom and bond scene from a PDB or SDF model.

Usage

```
generate_full_scene(
```

```

model,
x = 0,
y = 0,
z = 0,
scale = 1,
center = TRUE,
force_single_bonds = FALSE,
material = rayrender::glossy,
material_args = list(),
material_vertex = material_list(type = "phong")
)

```

Arguments

model	Model extracted from a PDB or SDF file.
x	Default '0'. X offset, applied after centering.
y	Default '0'. Y offset, applied after centering.
z	Default '0'. Z offset, applied after centering.
scale	Default '1'. Amount to scale the interatom spacing.
center	Default 'TRUE'. Centers the bounding box of the model.
force_single_bonds	Default 'FALSE'. Whether to force all bonds to show as a single connection.
material	Default 'rayrender::glossy'. Optional rayrender material used to initialize the mesh material when 'material_vertex' is not supplied. Must be either 'glossy', 'diffuse', or 'dielectric'.
material_args	Default 'list()'. Named list of additional arguments passed to 'material'. Arguments supplied by raymolecule for colors and textures override entries with the same names. For example, use 'list(gloss = 0.35, reflectance = 0.12)' with 'rayrender::glossy', or 'list(sigma = 0.4)' with 'rayrender::diffuse'.
material_vertex	Default 'rayvertex::material_list()'. Mesh material. 'diffuse'/'ambient' colors and 'ambient_intensity' are determined automatically, but all other material properties can be changed.

Value

Raymesh scene

Examples

```

molecule_model = read_sdf(get_example_molecule("caffeine"))

# Start with a centered raster scene that combines atom spheres and bond
# geometry using the default atom colors.
molecule_model |>
  generate_full_scene(force_single_bonds = TRUE) |>
  render_model(

```

```
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey12"
  )

# This version changes the scale, keeps the model centered, and uses a
# diffuse mesh material for both atoms and bonds before pathtracing.
molecule_model |>
  generate_full_scene(
    x = 0,
    y = 0,
    z = 0,
    scale = 0.75,
    center = TRUE,
    force_single_bonds = TRUE,
    material = rayrender::diffuse,
    material_args = list(sigma = 0.3)
  ) |>
  render_model(pathtrace = TRUE, width = 800, height = 800, samples = 32)

# A toon material changes the raster shader and pass FSAA to render_model()
# so rayvertex adds anti-aliasing.
shiny_toon_material = rayvertex::material_list(
  type = "toon_phong",
  toon_levels = 3,
  toon_outline_width = 10,
  toon_outline_color = "white"
)
morphine_model = read_sdf(get_example_molecule("morphine"))
morphine_model |>
  generate_full_scene(
    material_vertex = shiny_toon_material
  ) |>
  render_model(
    fsaa = 2,
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey50"
  )
```

generate_ribbon_scene *Build Scene (ribbon)*

Description

Generates a mesh-based protein ribbon from a PDB model parsed with [read_pdb()]. Segments with contiguous peptide-plane backbone data are rendered with a peptide-plane cartoon sweep,

while incomplete backbone segments fall back to a CA-driven centripetal Catmull-Rom ribbon. The mesh is emitted as indexed watertight geometry with UV coordinates, and eligible ‘SHEET’ annotations are rendered with peptide-plane strand profiles and tapered C-terminal arrowheads.

Usage

```
generate_ribbon_scene(
    model,
    x = 0,
    y = 0,
    z = 0,
    scale = 1,
    center = TRUE,
    model_id = NA_integer_,
    ribbon_width = 2,
    ribbon_thickness = 0.25,
    cross_section_resolution = 24,
    subdivisions = 8,
    color_mode = c("chain", "uv"),
    chain_colors = NULL,
    texture = NULL,
    material = rayrender::glossy,
    material_args = list(),
    material_vertex = rayvertex::material_list(type = "phong"),
    raster_ambient_mix = 0.5,
    show_hetero_atoms = TRUE,
    show_hetero_bonds = TRUE,
    show_waters = FALSE,
    show_protein_atoms = FALSE,
    show_protein_bonds = FALSE,
    atom_scale = 1,
    bond_width = 1,
    use_vertex_normals = FALSE,
    verbose = FALSE
)
```

Arguments

<code>model</code>	Model extracted from a PDB file with <code>[read_pdb()]</code> .
<code>x</code>	Default ‘0’. X offset, applied after centering.
<code>y</code>	Default ‘0’. Y offset, applied after centering.
<code>z</code>	Default ‘0’. Z offset, applied after centering.
<code>scale</code>	Default ‘1’. Amount to scale the ribbon geometry.
<code>center</code>	Default ‘TRUE’. Centers the bounding box of the model.
<code>model_id</code>	Default ‘NA_integer_’. PDB ‘MODEL’ identifier(s) to render. The default ‘NA’ renders all parsed models in the ensemble.
<code>ribbon_width</code>	Default ‘2’. Width of the ribbon cross-section.

ribbon_thickness	Default '0.25'. Thickness of the ribbon cross-section.
cross_section_resolution	Default '24'. Number of perimeter vertices used to approximate the ribbon cross-section.
subdivisions	Default '8'. Minimum number of spline samples per residue interval. Longer backbone spans are automatically refined to a smaller internal step size to avoid visible faceting.
color_mode	Either '"chain"' or '"uv"'. If omitted, single-chain proteins default to '"uv"' and multi-chain proteins default to '"chain"'.
chain_colors	Optional named vector or named list keyed by chain ID. Used when 'color_mode = "chain"'.
texture	Optional texture path. Used when 'color_mode = "uv"'. If omitted in UV mode, a built-in rainbow gradient is used.
material	Default 'rayrender::glossy'. Optional rayrender material used to initialize the mesh material when 'material_vertex' is not supplied. Must be either 'glossy', 'diffuse', or 'dielectric'.
material_args	Default 'list()'. Named list of additional arguments passed to 'material'. Arguments supplied by raymolecule for colors and textures override entries with the same names. For example, use 'list(gloss = 0.35, reflectance = 0.12)' with 'rayrender::glossy', or 'list(sigma = 0.4)' with 'rayrender::diffuse'.
material_vertex	Default 'rayvertex::material_list(type = "phong)'. Mesh material template.
raster_ambient_mix	Default '0.5'. Fraction of raster ribbon color contributed by the ambient, unlit material channel. Values closer to '0' emphasize diffuse directional lighting; values closer to '1' flatten lighting and preserve texture/color more directly.
show_hetero_atoms	Default 'TRUE'. If 'TRUE', display non-protein 'HETATM' records as bare spheres alongside the ribbon.
show_hetero_bonds	Default 'TRUE'. If 'TRUE', display bonds between shown hetero atoms as a ball-and-stick overlay alongside the ribbon.
show_waters	Default 'FALSE'. If 'TRUE', include water 'HETATM' records when 'show_hetero_atoms = TRUE' and 'show_hetero_bonds = TRUE'.
show_protein_atoms	Default 'FALSE'. If 'TRUE', display protein 'ATOM' records as small spheres alongside the ribbon.
show_protein_bonds	Default 'FALSE'. If 'TRUE', display inferred covalent bonds between protein 'ATOM' records as a thin stick overlay.
atom_scale	Default '1'. Multiplier applied to the radii of optional atom overlays.
bond_width	Default '1'. Multiplier applied to the radii of optional bond overlays.

`use_vertex_normals` Default 'FALSE'. If 'TRUE', attach the ribbon mesh's swept vertex normals. If 'FALSE', scenes omit explicit vertex normals and pathtraced renders use flat face normals.

`verbose` Default 'FALSE'. If 'TRUE', report the PDB name and model identifiers being rendered.

Value

Raymesh scene containing a ribbon mesh.

Examples

```
ribbon_file = download_pdb("2w5o", out_dir = tempdir(), overwrite = TRUE)
ribbon_model = read_pdb(ribbon_file, verbose = TRUE)

# Start with a centered raster ribbon using the default ribbon width,
# thickness, color mode, and ligand overlays.
ribbon_model |>
  generate_ribbon_scene() |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey12"
  )

# This pathtraced version widens the ribbon, increases cross-section
# resolution, applies a custom UV texture, turns on atom/bond overlays, and
# uses the mesh vertex normals.
texture_file = tempfile(fileext = ".png")
grDevices::png(texture_file, width = 64, height = 8, bg = "transparent")
graphics::par(mar = c(0, 0, 0, 0))
graphics::image(
  matrix(seq(0, 1, length.out = 64), ncol = 1),
  col = grDevices::hcl.colors(64, "Spectral", rev = TRUE),
  axes = FALSE,
  xlab = "",
  ylab = ""
)
grDevices::dev.off()

ribbon_model |>
  generate_ribbon_scene(
    x = 0,
    y = 0,
    z = 0,
    scale = 1,
    center = TRUE,
    ribbon_width = 1.8,
    ribbon_thickness = 0.3,
    cross_section_resolution = 32,
```

```

        subdivisions = 10,
        color_mode = "uv",
        texture = texture_file,
        material = rayrender::diffuse,
        material_args = list(sigma = 0.2),
        show_hetero_atoms = TRUE,
        show_hetero_bonds = TRUE,
        show_waters = TRUE,
        show_protein_atoms = TRUE,
        show_protein_bonds = TRUE,
        atom_scale = 1.2,
        bond_width = 0.8,
        use_vertex_normals = TRUE,
        verbose = TRUE
    ) |>
    render_model(pathtrace = TRUE, width = 800, height = 800, samples = 128)

# Start the beta-barrel example with the default UV texture in raster mode.
# We rotate the model to match the render in the protein data bank.
barrel_model = read_pdb(download_pdb("4fsp", out_dir = tempdir()))
barrel_scene = barrel_model |>
  generate_ribbon_scene(color_mode = "uv", raster_ambient_mix = 0.7)
barrel_render = render_model(
  barrel_scene,
  pathtrace = FALSE,
  width = 500,
  height = 500,
  background = "white",
  lookfrom = c(-89.95, 66.11, -109.95),
  angle = c(-60, 270, 180),
  lookat = c(6.06, -7.62, 1.41),
  fov = 27.2,
  plot = FALSE
)

pdb_4fsp_image = rayimage::render_title(system.file(
  "extdata",
  "4fsp_assembly-1.jpeg",
  package = "raymolecule",
  mustWork = TRUE
), title_text = "Protein Data Bank Render")

ray_4fsp_image = rayimage::render_title(
  barrel_render,
  title_text = "Raymolecule Render"
)
rayimage::plot_image_grid(list(pdb_4fsp_image, ray_4fsp_image), dim=c(1,2))
# Raising the raster ambient mix gives the same barrel less directional
# lighting
barrel_model |>
  generate_ribbon_scene(
    color_mode = "uv",
    texture = NULL,

```

```

    raster_ambient_mix = 0.8
  ) |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey12"
  )

# Start the multi-chain example with the automatic chain color palette.
multi_chain_model = read_pdb(download_pdb("1xn1", out_dir = tempdir()))
multi_chain_model |>
  generate_ribbon_scene(color_mode = "chain") |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey80"
  )

# An explicit chain color map replaces the automatic palette without
# changing the ribbon geometry.
chain_ids = unique(multi_chain_model$residues$chain_id)
chain_ids = chain_ids[!is.na(chain_ids)]
chain_colors = stats::setNames(grDevices::rainbow(length(chain_ids)), chain_ids)
multi_chain_model |>
  generate_ribbon_scene(
    color_mode = "chain",
    chain_colors = chain_colors,
    material_vertex = rayvertex::material_list(type = "phong")
  ) |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey80"
  )

# Start the NMR ensemble view by rendering every parsed model.
ensemble_model = read_pdb(download_pdb("1co1", out_dir = tempdir()))
ensemble_model |>
  generate_ribbon_scene(center = FALSE) |>
  render_model(
    pathtrace = TRUE,
    fov=26,
    lookfrom = c(-45.95, 58.56, 79.95),
    lookat = c(-7.19, 3.87, -1.52) ,
    width = 800,
    height = 800, samples=128,
    background = "black"
  )

# Selecting three model IDs shows a cleaner subset of the same ensemble.

```

```
ensemble_model |>
  generate_ribbon_scene(model_id = c(1L, 5L, 10L), center = FALSE) |>
  render_model(
    pathtrace = TRUE,
    fov=26,
    lookfrom = c(-45.95, 58.56, 79.95),
    lookat = c(-7.19, 3.87, -1.52) ,
    width = 800,
    height = 800, samples=128,
    background = "black"
  )
```

get_example_molecule	<i>Get Example Molecule</i>
----------------------	-----------------------------

Description

Loads the structure of the built-in molecules. All SDF files obtained from Pubchem. txt extension only included to pass R CHECK.

Usage

```
get_example_molecule(molecule)
```

Arguments

molecule	One of the built-in SDF files. These are "benzene", "buckyball", "caffeine", "capsaicin", "cinnemaldehyde", "geraniol", "luciferin", "morphine", "penicillin", "pfoa", "skatole", "tubocurarine_chloride".
----------	--

Value

List giving the atom locations and the connections between atoms.

Examples

```
get_example_molecule("benzene")
get_example_molecule("cinnemaldehyde")
get_example_molecule("geraniol")
```

`get_molecule`*Get Molecule*

Description

Loads the structure of a molecule by fetching an SDF file from Pubchem, which can be piped to `generate_full_scene`

Usage

```
get_molecule(molecule)
```

Arguments

<code>molecule</code>	A character variable of a compound name or a numeric variable of an official compound ID
-----------------------	--

Value

List giving the atom locations and the connections between atoms.

Examples

```
get_molecule("caffeine") |>
  generate_full_scene() |>
  render_model()

#estradiol (aka estrogen)
get_molecule(5757) |>
  generate_full_scene() |>
  render_model()

get_molecule("testosterone") |>
  generate_full_scene() |>
  render_model()

get_molecule("aspirin") |>
  generate_full_scene() |>
  render_model()

get_molecule("rutoside") |>
  generate_full_scene() |>
  render_model()
```

```
#If the 3D SDF doesn't exist, this function will pull the 2D SDF and inform the user
get_molecule("cyanocobalamin") |>
  generate_full_scene() |>
  render_model()
```

read_pdb

Read PDB File

Description

Reads a legacy fixed-width PDB file and extracts atom locations, bond connections, backbone residue information, and secondary structure records. 'model\$atoms' and 'model\$bonds' remain compatible with the existing atom and bond scene generators. Biological assemblies can be generated from 'REMARK 350 BIOMT' transforms when requested.

Usage

```
read_pdb(
  filename,
  atom = TRUE,
  nsr = TRUE,
  assembly = c("asymmetric_unit", "biological"),
  assembly_id = 1L,
  verbose = FALSE
)
```

Arguments

filename	Path to the PDB file, or a protein name/4-character PDB ID to download from RCSB when it is not an existing file and does not end in '.pdb'.
atom	Default 'TRUE'. Whether to include standard residue 'ATOM' records in 'model\$atoms'.
nsr	Default 'TRUE'. Whether to include 'HETATM' records in 'model\$atoms'.
assembly	Default "asymmetric_unit". Either the deposited asymmetric unit or the biological assembly generated from 'REMARK 350 BIOMT' transforms.
assembly_id	Default '1L'. Biological assembly identifier to use when 'assembly = "biological"'.
verbose	Default 'FALSE'. If 'TRUE', report parsed PDB metadata and atom/residue/model counts.

Value

List giving the parsed PDB model.

Examples

```
# Start with a local PDB file and the deposited asymmetric unit. Both ATOM
# and HETATM records are kept, and verbose output prints parse metadata.
pdb_file = download_pdb("2w5o", out_dir = tempdir(), overwrite = TRUE)
model = read_pdb(
  pdb_file,
  atom = TRUE,
  nsr = TRUE,
  assembly = "asymmetric_unit",
  verbose = TRUE
)

model |>
  generate_ribbon_scene() |>
  render_model(
    pathtrace = FALSE,
    width = 800,
    height = 800,
    background = "grey12"
  )

# The biological assembly applies REMARK 350 BIOMT transforms when present.
biological_model = read_pdb(
  pdb_file,
  assembly = "biological",
  assembly_id = 1L
)

# This ligand-only parse keeps HETATM records and drops standard protein
# atoms, which is useful for ligand and ion views.
ligand_file = download_pdb("5hsv", out_dir = tempdir(), overwrite = TRUE)
ligand_model = read_pdb(ligand_file, atom = FALSE, nsr = TRUE)

ligand_model |>
  generate_full_scene(force_single_bonds = TRUE) |>
  render_model(pathtrace = TRUE, width = 800, height = 800, samples = 32)

# A bare PDB ID is downloaded automatically when no matching local file is
# found. Verbose output reports the multi-model ensemble.
old_dir = setwd(tempdir())
ensemble_model = read_pdb("1co1", verbose = TRUE)
setwd(old_dir)
```

read_sdf

Read SDF File

Description

Reads an SDF file and extracts the 3D molecule model

Usage

```
read_sdf(filename)
```

Arguments

filename Filename to the sdf file.

Value

List giving the atom locations and the connections between atoms.

Examples

```
#This assumes a hypothetical SDF file in your working directory:
if(file.exists("molecule.sdf")) {
  read_pdb("molecule.sdf") |>
    generate_full_scene() |>
    render_model()
}
```

render_model	<i>Render Molecule Model</i>
--------------	------------------------------

Description

Automatically plots the molecule with a camera position and field of view that includes the full model. For more control over the scene, pass the scene to ‘rayrender::render_scene()’ or ‘rayvertext::rasterize_scene()’ and specify the camera position manually.

Usage

```
render_model(
  scene,
  width = 800,
  height = 800,
  fov = NULL,
  lookat = NULL,
  lookfrom = NULL,
  angle = c(0, 0, 0),
  order_rotation = c(1, 2, 3),
  lights = "top",
  lightintensity = 80,
  pathtrace = TRUE,
  plot = TRUE,
  scene_elements = NULL,
  ...
)
```

Arguments

scene	Raymesh scene of molecule model.
width	Default '800'. Image width in pixels.
height	Default '800'. Image height in pixels.
fov	Default 'NULL', automatically calculated. Camera field of view.
lookat	Default 'NULL', automatically calculated as 'c(0,0,0)'. Camera target point.
lookfrom	Default 'NULL', automatically calculated from the scene bounds. Camera position.
angle	Default 'c(0,0,0)'. Degrees to rotate the model around the X, Y, and Z axes. If this is a single number, it is interpreted as 'c(0, angle, 0)' and only rotates around the Y axis.
order_rotation	Default 'c(1,2,3)'. What order to apply the rotations specified in 'angle'.
lights	Default 'top'. If 'none', removes all lights. If 'bottom', lights scene with light underneath model. If 'both', adds lights both above and below model. This can also be a matrix of light information generated with 'rayvertex'.
lightintensity	Default '80'. Light intensity for pathtraced scenes.
pathtrace	Default 'TRUE'. If 'TRUE', convert the raymesh scene to a 'rayrender' raymesh and pathtrace it. Scene-generator rayrender materials are passed to 'rayrender::raymesh_model()' with 'override_material = TRUE' when available. If 'FALSE', rasterize the raymesh scene with 'rayvertex'.
plot	Default 'TRUE'. If 'TRUE', plot the rendered image to the current graphics device. If 'FALSE', only return the rendered rayimage array.
scene_elements	Default 'NULL'. Additional backend scene elements to add after automatic camera calculation and model rotation. For 'pathtrace = TRUE', elements may be rayrender scenes or rayvertex raymesh scenes. For 'pathtrace = FALSE', elements must be rayvertex raymesh scenes.
...	Other arguments to pass to 'rayrender::render_scene()' or 'rayvertex::rasterize_scene()'. Arguments that only apply to the other backend are ignored. For pathtraced scenes, 'background' sets both 'backgroundlow' and 'backgroundhigh' in 'rayrender::render_scene()' and turns on 'ambient_light'. For raster scenes, 'background' is passed through to 'rayvertex::rasterize_scene()'.

Value

Rendered image

Examples

```
compact_model = read_pdb(download_pdb("6k16", out_dir = tempdir()))
compact_scene = compact_model |>
  generate_ribbon_scene(
    material = rayrender::diffuse,
    show_hetero_atoms = FALSE
  )
```

```

# Start with render_model's default camera, top lighting, and calculated FOV.
compact_scene |>
  render_model(pathtrace = TRUE, width = 800, height = 800, samples = 128)

# Use `plot = FALSE` when you want the image array without drawing it.
compact_image = compact_scene |>
  render_model(
    pathtrace = TRUE,
    width = 800,
    height = 800,
    samples = 128,
    plot = FALSE
  )

# This render uses an explicit camera, scalar Y rotation, a solid
# background, weaker top light, and rayrender sampling options. The
# camera orientation was extracted interactively in rayrender by
# pressing the "P" key.
compact_scene |>
  render_model(
    pathtrace = TRUE,
    width = 800,
    height = 800,
    fov = 72,
    lookat = c(-1.93, 11.99, 8.47),
    lookfrom = c(-25.50, 26.52, 14.99),
    aperture = 2,
    angle = 35,
    background = "grey2",
    lights = "top",
    lightintensity = 50,
    samples = 128,
    sample_method = "sobol_blue",
    clamp_value = 10
  )

# Vector rotation and a custom rotation order show the same scene from a
# different orientation with lights above and below.
compact_scene |>
  render_model(
    pathtrace = TRUE,
    angle = c(20, 90, 90),
    lights = "both",
    fov = 28,
    samples = 128,
    sample_method = "sobol_blue"
  )

# Disable render_model's automatic lights and add your own.
tmp_exr = tempfile(fileext=".exr")
background_light = matrix(0,nrow=500,ncol=1000)
background_light[140:160, 250:270] = 700 #key
background_light[100:170, 200:300+500] = 10 #fill

```

```

rayimage::ray_write_image(background_light, tmp_exr)
#Generate a key and fill light
rayimage::plot_image(background_light)
compact_scene |>
  render_model(
    pathtrace = TRUE,
    lights = "none",
    background = "grey12",
    samples = 128,
    fov = 25,
    sample_method = "sobol_blue",
    rotate_env = 180,
    environment_light = tmp_exr
  )

# Extra rayrender scene elements can be added after the automatic camera and
# model rotation are set, which is useful for fixed reference marks.
compact_scene |>
  render_model(
    pathtrace = TRUE,
    angle = 35,
    scene_elements = rayrender::sphere(
      x = -10,
      y = -10,
      z = -10,
      radius = 0.5,
      material = rayrender::light(intensity = 15)
    ),
    samples = 128,
    sample_method = "sobol_blue"
  )

# Raster scenes use rayvertex lighting. This example passes a directional
# light matrix through render_model to rasterize_scene().
raster_scene = compact_model |>
  generate_ribbon_scene()
raster_scene |>
  render_model(
    pathtrace = FALSE,
    background = "grey12",
    lights = rayvertex::directional_light(c(0.2, 1, -1))
  )

```

Index

`download_pdb`, [2](#)

`generate_atom_scene`, [3](#)

`generate_bond_scene`, [5](#)

`generate_full_scene`, [7](#)

`generate_ribbon_scene`, [9](#)

`get_example_molecule`, [15](#)

`get_molecule`, [16](#)

`read_pdb`, [17](#)

`read_sdf`, [18](#)

`render_model`, [19](#)