plato Documentation

Release 1.10.0

Matthew Spellings

Mar 27, 2020

Contents

Plato is designed for efficient visualization of particle data. Think of it sort of like matplotlib, but being less focused on 2D plotting.

CHAPTER 1

Installation

Plato is available on PyPI for installation via pip:

```
$ pip install plato-draw
```

You can also install plato from source, like this:

```
$ git clone https://github.com/glotzerlab/plato.git
$ # now install
$ cd plato && python setup.py install
```

Note: Depending on which backends you want to use, there may be additional steps required; see the section on interactive backends below.

1.1 Using Interactive Backends

Plato contains a number of backends, each with its own set of dependencies. Getting the vispy backend working for both the desktop and jupyter notebook can be tricky. Make sure to check the official vispy documentation. We also keep some advice here regarding particular known-good versions of dependencies for pip and conda.

CHAPTER 2

Documentation

The documentation is available as standard sphinx documentation:

```
$ cd doc
$ make html
```

Automatically-built documentation is available at https://plato-draw.readthedocs.io .

CHAPTER 3

Examples

Several usage examples are available. Many simple, but less interesting, scenes can be found in the test demo scene script, available as live examples on mybinder.org. Somewhat less transparent examples can be found in the plato-gallery repository.

CHAPTER 4

Contents:

Plato primitives

4.1 Plato Primitives

Plato's graphics primitives all follow a fairly standard form. Depending on the shapes to be rendered, different properties may be per-particle (such as positions, orientations, and colors) or global (the ConvexPolyhedra primitive is restricted to drawing any number of identically-shaped convex polyhedra; in other words, the vertices given are for all particles rendered).

Primitives' data can be set and retrieved through properties, which are exposed as numpy arrays whenever possible. For example, to scale the diameter of each disk in a Disks primitive by 2:

```
disks = plato.draw.Disks(...)
disks.diameters *= 2
```

Primitives can be grouped together by placing them in the same *plato.draw.Scene*.

The classes inside *plato.draw* are simple containers and are not useful for visualization. Instead, a particular *backend* should be used, for example:

```
import plato.draw.matplotlib as draw
disks = draw.Disks(...)
scene = draw.Scene(disks, ...)
scene.show()
```

Note: For quick and simple visualizations, the imperative `plato.imp` module may be easier.

4.1.1 Base Drawing Module

class plato.draw.**Scene**(*primitives=[]*, *features={}*, *size=(40, 30)*, *translation=(0, 0, -50)*, *rotation=(1, 0, 0, 0)*, *zoom=1*, *pixel_scale=20*, ***kwargs*)

A container to hold and display collections of primitives.

Scene keeps track of global information about a set of things to be rendered and handles configuration of optional (possibly backend-specific) rendering parameters.

Global information managed by a *Scene* includes the *size* of the viewing window, *translation* and *rotation* applied to the scene as a whole, and a *zoom* level.

Primitives can be added to a scene through the *primitives* argument of the constructor or the *add_primitive* method. Primitives can be retrieved by iterating over the scene:

```
for prim in scene:
    # (do something with prim)
```

Primitives can also be accessed in the order they were added to the scene using list-like syntax:

```
first_three_prims = scene[:3]
last_prim = scene[-1]
```

Optional rendering arguments are enabled as *features*, which are name-value pairs identifying a feature by name and any configuration of the feature in the value.

add_primitive(*primitive*)

Adds a primitive to the scene.

convert(*backend*, *compatibility='warn'*, ***kwargs*)

Convert this scene and all of its primitives to another backend.

Parameters

- **backend** – Backend plato.draw.* module to use in the new scene
- **compatibility** – Behavior when unsupported primitives are encountered: 'warn', 'ignore', or 'error'
- **kwargs** – Additional keyword arguments to be passed into the backend *Scene* constructor

disable(*name*, *strict=True*)

Disable an optional rendering feature.

Parameters

- **name** – Name of the feature to disable
- **strict** – if True, raise a KeyError if the feature was not enabled

enable(*name*, *auto_value=None*, ***parameters*)

Enable an optional rendering feature.

Parameters

- **name** – Name of the feature to enable
- **auto_value** – Shortcut for features with single-value configuration. If given as a positional argument, will be given the default configuration name 'value'.

- **parameters** – Keyword arguments specifying additional configuration options for the given feature

get_feature_config(*name*)
Return the configuration dictionary for a given feature.

If the feature has not been enabled, return None.

remove_primitive(*primitive*, *strict=True*)
Removes a primitive from the scene.

Parameters

- **primitive** – primitive to (attempt to) remove
- **strict** – If True, raise an IndexError if the primitive was not in the scene

rotation
(r, x, y, z) rotation quaternion to be applied to the scene as a whole.

size
Width and height, in scene units, of the viewport.

size_pixels
Width and height, in pixels, of the viewport.

transform(*coords*, *source*, *dest='scene'*)
Transform one or more points between two coordinate systems.

Parameters

- **coords** – Nx2 array-like of coordinates to transform
- **source** – Coordinate system of coords: one of 'pixels_gui' (display pixel units, top left is (0, 0)), 'pixels' (display pixel units, bottom left is (0, 0)), 'ndc' ((-1, -1) to (1, 1) at the two corners), or 'scene' (working scene world coordinates)
- **source** – Coordinate system of returned values: one of 'pixels_gui' (display pixel units, top left is (0, 0)), 'pixels' (display pixel units, bottom left is (0, 0)), 'ndc' ((-1, -1) to (1, 1) at the two corners), or 'scene' (working scene world coordinates)

translation
(x, y, z) translation to be applied to the scene as a whole after rotating.

x is to the right, *y* is up, and *z* comes toward you out of the screen.

2D Graphics Primitives

class plato.draw.**Arrows2D**(**args*, ***kwargs*)
A collection of 2D arrows.

Each arrow has an independent position, orientation, color, and magnitude. The shape of arrows can be configured by changing its *vertices* attribute. The default orientation and scale of the vertices is an arrow centered at (0, 0), pointing in the (1, 0) direction, with length 1.

The origin of the arrows can be shifted to have the base lie on the given position by modifying *vertices*:

```
arrows.vertices = arrows.vertices + (0.5, 0)
```

This primitive has the following attributes:

- positions: Position of each particle

- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

colors
Color, RGBA, [0, 1] for each particle

magnitudes
Magnitude (size scale) of each particle

orientations
Orientation quaternion of each particle

outline
Outline width for all particles

positions
Position of each particle

vertices
Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)

class plato.draw.**DiskUnions**(***kwargs*)
A collection of identical disk-union bodies in 2D.

A *DiskUnions* object consists of one or more disks, each with its own radius and color. Each object has its own position and orientation that affect the final position of the constituent disks.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each disk in the union
- points: Positions in local coordinates for the disks in the union, to be replicated for each particle
- radii: Radius of each disk in the union
- outline: Outline width for all particles

angles
Orientation of each union, in radians

colors
Color, RGBA, [0, 1] for each disk in the union

diameters
Diameter of each disk in the union.

orientations
Orientation quaternion of each particle

outline
Outline width for all particles

points
Positions in local coordinates for the disks in the union, to be replicated for each particle

positions
Position of each particle

radii
Radius of each disk in the union

class plato.draw.**Disks**(***kwargs*)
A collection of disks in 2D.

Each disk can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- outline: Outline width for all particles

colors
Color, RGBA, [0, 1] for each particle

diameters
Diameter of each particle.

outline
Outline width for all particles

positions
Position of each particle

radii
Radius of each particle

class plato.draw.**Polygons**(***kwargs*)
A collection of polygons.

A *Polygons* object has a common shape for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

angles
Orientation of each particle, in radians

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

outline
Outline width for all particles

positions
Position of each particle

vertices
Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)

class plato.draw.**Spheropolygons**(***kwargs*)
A collection of rounded polygons.

A *Spheropolygons* object has a common shape and rounding radius for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles
- radius: Rounding radius for all particles

angles
Orientation of each particle, in radians

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

outline
Outline width for all particles

positions
Position of each particle

radius
Rounding radius for all particles

vertices
Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle (CCW order)

class plato.draw.**Voronoi**(***kwargs*)
A Voronoi diagram of a set of 2D points.

The region of space nearest to each given point will be colored by the color associated with that point.

This primitive has the following attributes:

- positions: Position of each point
- colors: Color, RGBA, [0, 1] for each point

colors
Color, RGBA, [0, 1] for each point

positions
Position of each point

3D Graphics Primitives

class plato.draw.**Box**(**args*, ***kwargs*)

A triclinic box frame.

This primitive draws a triclinic box centered at the origin. It is specified in terms of three lattice vector lengths *Lx*, *Ly*, *Lz* and tilt factors, defined using the hoomd-blue schema.

Rather than directly initializing via attributes, *Box* objects can also be automatically created from box-type objects using the *from_box()* method.

Examples:

```
Lx = Ly = Lz = 10
xy = xz = yz = 0
box_primitive = draw.Box(Lx=Lx, Ly=Ly, Lz=Lz, width=width, color=color)
box_tuple = (Lx, Ly, Lz, xy, xz, yz)
box_primitive = draw.Box.from_box(box_tuple)
```

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- Lx: Length of first box vector
- Ly: Length of second box vector
- Lz: Length of third box vector
- xy: Tilt factor between the first and second box vectors
- xz: Tilt factor between the first and third box vectors
- yz: Tilt factor between the second and third box vectors
- width: Width of box line segments
- color: Color, RGBA, [0, 1] for the box line segments

Lx
Length of first box vector

Ly
Length of second box vector

Lz
Length of third box vector

color
Color, RGBA, [0, 1] for the box line segments

colors
Color, RGBA, [0, 1] for each line segment

end_points
Ending coordinate for each line segment

classmethod from_box(*box*, *width=0.01*, *color=(0, 0, 0, 1)*)
Duck type the box from a valid input.

Boxes can be a list, dictionary, or object with attributes.

start_points
Beginning coordinate for each line segment

width
Width of box line segments

widths
Width of each line segment

xy
Tilt factor between the first and second box vectors

xz
Tilt factor between the first and third box vectors

yz
Tilt factor between the second and third box vectors

class plato.draw.**ConvexPolyhedra**(***kwargs*)
A collection of identically-shaped convex polyhedra.

Each shape can have its own position, orientation, and color.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle
- outline: Outline width for all shapes

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

outline
Outline width for all shapes

positions
Position of each particle

vertices
Vertices in local coordinates for the shape, to be replicated for each particle

class plato.draw.**ConvexSpheropolyhedra**(***kwargs*)
A collection of identically-shaped convex spheropolyhedra.

Each shape can have its own position, orientation, and color. The rounding radius is shared over all shapes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle

- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle
- radius: Rounding radius to be applied to all shapes

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

positions
Position of each particle

radius
Rounding radius to be applied to all shapes

vertices
Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle

class plato.draw.**Ellipsoids**(***kwargs*)
A collection of ellipsoids with identical dimensions.

Each ellipsoid can have its own position, orientation, and color. All shapes drawn by this primitive share common principal axis lengths.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- a: Radius in the x-direction
- b: Radius in the y-direction
- c: Radius in the z-direction

a
Radius in the x-direction

b
Radius in the y-direction

c
Radius in the z-direction

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

positions
Position of each particle

class plato.draw.**Lines**(***kwargs*)
A collection of line segments.

Each segment can have a different color and width. *Lines* can be used in both 2D and 3D scenes, but they are currently not shaded and may look out of place in 3D.

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment

colors
Color, RGBA, [0, 1] for each line segment

end_points
Ending coordinate for each line segment

start_points
Beginning coordinate for each line segment

widths
Width of each line segment

class plato.draw.**Mesh**(***kwargs*)
A 3D triangle mesh.

Meshes are specified by an array of vertices and indices identifying triangles within that vertex array. Colors are assigned per-vertex and interpolated between vertices.

Meshes with a common set of vertices and face indices can be replicated multiple times using a set of positions and orientations. In order to set the color of individual replicas of the Mesh object, use the *shape_colors* and *shape_color_fraction* attributes.

This primitive has the following attributes:

- vertices: Vertex array specifying coordinates of the mesh nodes
- indices: Indices of the vertex array specifying individual triangles (Nx3)
- colors: Color, RGBA, [0, 1] for each vertex
- positions: Central positions for each mesh to be replicated
- orientations: Orientations for each mesh to be replicated
- shape_colors: Color, RGBA, [0, 1] for each replica (shape) of the mesh
- shape_color_fraction: Fraction of a vertex's color that should be assigned based on shape_colors

colors
Color, RGBA, [0, 1] for each vertex

classmethod double_sided(*vertices*, *indices*, *colors*, *thickness=0.001*, ***kwargs*)
Create a double-sided Mesh object.

Typically the "inside" of a Mesh (as determined by the order of triangle indices) is unlit. This method replicates the vertices, one for each side, after computing the appropriate normals.

indices
Indices of the vertex array specifying individual triangles (Nx3)

orientations
Orientations for each mesh to be replicated

positions
Central positions for each mesh to be replicated

shape_color_fraction
Fraction of a vertex's color that should be assigned based on shape_colors

shape_colors
Color, RGBA, [0, 1] for each replica (shape) of the mesh

vertices
Vertex array specifying coordinates of the mesh nodes

class plato.draw.**SpherePoints**(***kwargs*)
A collection of points, useful for illustrating 3D density maps.

This primitive has the following attributes:

- points: Points to be rendered
- blur: Blurring factor dictating the size of each point
- intensity: Scaling factor dictating the magnitude of the color value of each point
- on_surface: True if the points should always be projected onto the surface of a sphere

blur
Blurring factor dictating the size of each point

intensity
Scaling factor dictating the magnitude of the color value of each point

on_surface
True if the points should always be projected onto the surface of a sphere

points
Points to be rendered

class plato.draw.**SphereUnions**(***kwargs*)
A collection of identical sphere-union bodies in 3D.

A *SphereUnions* object is a union of spheres, each of which has its own color, radius, and local position. The *SphereUnions* object can be rigidly rotated and translated via its position and orientation attributes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each sphere in the union
- points: Positions in local coordinates for the spheres in the union, to be replicated for each particle
- radii: Radius of each sphere in the union

colors
Color, RGBA, [0, 1] for each sphere in the union

diameters
Diameter of each particle.

orientations
Orientation quaternion of each particle

points
Positions in local coordinates for the spheres in the union, to be replicated for each particle

positions
Position of each particle

radii
Radius of each sphere in the union

class plato.draw.**Spheres**(***kwargs*)
A collection of spheres in 3D.

Each sphere can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle

colors
Color, RGBA, [0, 1] for each particle

diameters
Diameter of each particle.

positions
Position of each particle

radii
Radius of each particle

4.2 Fresnel Backend

The fresnel backend uses fresnel to generate high-quality, ray-traced images of scenes.

All fresnel primitives accept an argument `material` of type `fresnel.material.Material` to define how lights interact with the primitives.

Note: Translucency is not currently supported in the fresnel backend. All particles will be opaque.

class plato.draw.fresnel.**Scene**(**args*, *tracer_kwargs={}*, ***kwargs*)
A container to hold and display collections of primitives.

Scene keeps track of global information about a set of things to be rendered and handles configuration of optional (possibly backend-specific) rendering parameters.

Global information managed by a *Scene* includes the *size* of the viewing window, *translation* and *rotation* applied to the scene as a whole, and a *zoom* level.

Primitives can be added to a scene through the *primitives* argument of the constructor or the *add_primitive* method. Primitives can be retrieved by iterating over the scene:

```
for prim in scene:
    # (do something with prim)
```

Primitives can also be accessed in the order they were added to the scene using list-like syntax:

```
first_three_prims = scene[:3]
last_prim = scene[-1]
```

Optional rendering arguments are enabled as *features*, which are name-value pairs identifying a feature by name and any configuration of the feature in the value.

This Scene supports the following features:

- *antialiasing*: Enable antialiasing, for the preview tracer only. This uses fresnel's aa_level=3 if set, 0 otherwise.
- *pathtracer*: Enable the path tracer. Accepts parameter `samples` with default value 64.
- *directional_light*: Add directional lights. The given vector(s) indicates the light direction. The length of the vector(s) determines the magnitude of the light(s).
- *ambient_light*: Enable ambient lighting. The given value indicates the magnitude of the light.

render()
Render this Scene object.

save(*filename*)
Render and save an image of this Scene.

Parameters **filename** – target filename to save the image into

show()
Render the scene to an image and display using IPython.

4.2.1 2D Graphics Primitives

class plato.draw.fresnel.**Arrows2D**(**args*, ***kwargs*)
A collection of 2D arrows.

Each arrow has an independent position, orientation, color, and magnitude. The shape of arrows can be configured by changing its *vertices* attribute. The default orientation and scale of the vertices is an arrow centered at (0, 0), pointing in the (1, 0) direction, with length 1.

The origin of the arrows can be shifted to have the base lie on the given position by modifying *vertices*:

```
arrows.vertices = arrows.vertices + (0.5, 0)
```

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

class plato.draw.fresnel.**Disks**(**args*, ***kwargs*)
A collection of disks in 2D.

Each disk can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- outline: Outline width for all particles

class plato.draw.fresnel.**Polygons**(**args*, ***kwargs*)
A collection of polygons.

A *Polygons* object has a common shape for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

class plato.draw.fresnel.**Spheropolygons**(**args*, ***kwargs*)
A collection of rounded polygons.

A *Spheropolygons* object has a common shape and rounding radius for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles
- radius: Rounding radius for all particles

4.2.2 3D Graphics Primitives

class plato.draw.fresnel.**Box**(**args*, ***kwargs*)
A triclinic box frame.

This primitive draws a triclinic box centered at the origin. It is specified in terms of three lattice vector lengths *Lx*, *Ly*, *Lz* and tilt factors, defined using the hoomd-blue schema.

Rather than directly initializing via attributes, *Box* objects can also be automatically created from box-type objects using the `from_box()` method.

Examples:

```
Lx = Ly = Lz = 10
xy = xz = yz = 0
box_primitive = draw.Box(Lx=Lx, Ly=Ly, Lz=Lz, width=width, color=color)
box_tuple = (Lx, Ly, Lz, xy, xz, yz)
box_primitive = draw.Box.from_box(box_tuple)
```

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment

- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- Lx: Length of first box vector
- Ly: Length of second box vector
- Lz: Length of third box vector
- xy: Tilt factor between the first and second box vectors
- xz: Tilt factor between the first and third box vectors
- yz: Tilt factor between the second and third box vectors
- width: Width of box line segments
- color: Color, RGBA, [0, 1] for the box line segments

class plato.draw.fresnel.**ConvexPolyhedra**(**args*, ***kwargs*)
A collection of identically-shaped convex polyhedra.

Each shape can have its own position, orientation, and color.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle
- outline: Outline width for all shapes
- outline: Outline width for all particles

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

outline
Outline width for all particles

positions
Position of each particle

vertices
Vertices in local coordinates for the shape, to be replicated for each particle

class plato.draw.fresnel.**Ellipsoids**(**args*, ***kwargs*)
A collection of ellipsoids with identical dimensions.

Each ellipsoid can have its own position, orientation, and color. All shapes drawn by this primitive share common principal axis lengths.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle

- a: Radius in the x-direction
- b: Radius in the y-direction
- c: Radius in the z-direction
- outline: Outline width for all particles
- vertex_count: Number of vertices used to render ellipsoid

a
Radius in the x-direction

b
Radius in the y-direction

c
Radius in the z-direction

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

outline
Outline width for all particles

positions
Position of each particle

vertex_count
Number of vertices used to render ellipsoid

class plato.draw.fresnel.**Lines**(**args*, ***kwargs*)
A collection of line segments.

Each segment can have a different color and width. *Lines* can be used in both 2D and 3D scenes, but they are currently not shaded and may look out of place in 3D.

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- outline: Outline width for all particles

colors
Color, RGBA, [0, 1] for each line segment

end_points
Ending coordinate for each line segment

outline
Outline width for all particles

start_points
Beginning coordinate for each line segment

widths
Width of each line segment

class plato.draw.fresnel.**SphereUnions**(**args*, ***kwargs*)
A collection of identical sphere-union bodies in 3D.

A *SphereUnions* object is a union of spheres, each of which has its own color, radius, and local position. The *SphereUnions* object can be rigidly rotated and translated via its position and orientation attributes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each sphere in the union
- points: Positions in local coordinates for the spheres in the union, to be replicated for each particle
- radii: Radius of each sphere in the union

class plato.draw.fresnel.**Spheres**(**args*, ***kwargs*)
A collection of spheres in 3D.

Each sphere can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle

4.3 Matplotlib Backend

The matplotlib backend uses matplotlib to render shapes. Different matplotlib backends can be configured for interactivity, but plato does not currently otherwise support interactive manipulation of shapes using this backend.

Matplotlib has extensive support for a wide range of graphical formats, so it is ideal for saving vector versions of figures.

class plato.draw.matplotlib.**Scene**(*primitives=[]*, *features={}*, *size=(40, 30)*, *translation=(0, 0, -50)*, *rotation=(1, 0, 0, 0)*, *zoom=1*, *pixel_scale=20*, ***kwargs*)
A container to hold and display collections of primitives.

Scene keeps track of global information about a set of things to be rendered and handles configuration of optional (possibly backend-specific) rendering parameters.

Global information managed by a *Scene* includes the *size* of the viewing window, *translation* and *rotation* applied to the scene as a whole, and a *zoom* level.

Primitives can be added to a scene through the *primitives* argument of the constructor or the *add_primitive* method. Primitives can be retrieved by iterating over the scene:

```
for prim in scene:
    # (do something with prim)
```

Primitives can also be accessed in the order they were added to the scene using list-like syntax:

```
first_three_prims = scene[:3]
last_prim = scene[-1]
```

Optional rendering arguments are enabled as *features*, which are name-value pairs identifying a feature by name and any configuration of the feature in the value.

This Scene supports the following features:

- *antialiasing*: Enable antialiasing. Primitives that support antialiasing will fudge some distances (typically for drawing outlines) to reduce visual artifacts.

render (*figure=None*, *axes=None*)
Render all the shapes in this Scene.

Parameters

- **figure** – Figure object to render within (created using pyplot if not given)
- **axes** – Axes object to render within (created from the figure if not given)

save (*filename*)
Render and save an image of this Scene.

Parameters **filename** – target filename to save the image into

show (*figure=None*, *axes=None*)
Render and show the shapes in this Scene.

Parameters

- **figure** – Figure object to render within (created using pyplot if not given)
- **axes** – Axes object to render within (created from the figure if not given)

4.3.1 2D Graphics Primitives

class plato.draw.matplotlib.**Arrows2D**(**args*, ***kwargs*)
A collection of 2D arrows.

Each arrow has an independent position, orientation, color, and magnitude. The shape of arrows can be configured by changing its *vertices* attribute. The default orientation and scale of the vertices is an arrow centered at (0, 0), pointing in the (1, 0) direction, with length 1.

The origin of the arrows can be shifted to have the base lie on the given position by modifying *vertices*:

```
arrows.vertices = arrows.vertices + (0.5, 0)
```

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

class plato.draw.matplotlib.**Disks**(***kwargs*)
A collection of disks in 2D.

Each disk can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle

- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- outline: Outline width for all particles

class plato.draw.matplotlib.**DiskUnions**(***kwargs*)
A collection of identical disk-union bodies in 2D.

A *DiskUnions* object consists of one or more disks, each with its own radius and color. Each object has its own position and orientation that affect the final position of the constituent disks.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each disk in the union
- points: Positions in local coordinates for the disks in the union, to be replicated for each particle
- radii: Radius of each disk in the union
- outline: Outline width for all particles

class plato.draw.matplotlib.**Polygons**(***kwargs*)
A collection of polygons.

A *Polygons* object has a common shape for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

class plato.draw.matplotlib.**Spheropolygons**(***kwargs*)
A collection of rounded polygons.

A *Spheropolygons* object has a common shape and rounding radius for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles
- radius: Rounding radius for all particles

4.3.2 3D Graphics Primitives

class plato.draw.matplotlib.**Box**(**args*, ***kwargs*)
A triclinic box frame.

This primitive draws a triclinic box centered at the origin. It is specified in terms of three lattice vector lengths *Lx*, *Ly*, *Lz* and tilt factors, defined using the hoomd-blue schema.

Rather than directly initializing via attributes, *Box* objects can also be automatically created from box-type objects using the `from_box()` method.

Examples:

```
Lx = Ly = Lz = 10
xy = xz = yz = 0
box_primitive = draw.Box(Lx=Lx, Ly=Ly, Lz=Lz, width=width, color=color)
box_tuple = (Lx, Ly, Lz, xy, xz, yz)
box_primitive = draw.Box.from_box(box_tuple)
```

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- Lx: Length of first box vector
- Ly: Length of second box vector
- Lz: Length of third box vector
- xy: Tilt factor between the first and second box vectors
- xz: Tilt factor between the first and third box vectors
- yz: Tilt factor between the second and third box vectors
- width: Width of box line segments
- color: Color, RGBA, [0, 1] for the box line segments

class plato.draw.matplotlib.**ConvexPolyhedra**(***kwargs*)
A collection of identically-shaped convex polyhedra.

Each shape can have its own position, orientation, and color.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle
- outline: Outline width for all shapes

class plato.draw.matplotlib.**Lines**(***kwargs*)
A collection of line segments.

Each segment can have a different color and width. *Lines* can be used in both 2D and 3D scenes, but they are currently not shaded and may look out of place in 3D.

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment

class plato.draw.matplotlib.**SpherePoints**(***kwargs*)

A collection of points, useful for illustrating 3D density maps.

This primitive has the following attributes:

- points: Points to be rendered
- blur: Blurring factor dictating the size of each point
- intensity: Scaling factor dictating the magnitude of the color value of each point
- on_surface: True if the points should always be projected onto the surface of a sphere

class plato.draw.matplotlib.**Spheres**(***kwargs*)

A collection of spheres in 3D.

Each sphere can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- light_levels: Number of quantized light levels to use

colors

Color, RGBA, [0, 1] for each particle

light_levels

Number of quantized light levels to use

positions

Position of each particle

radii

Radius of each particle

4.4 Povray Backend

The povray backend generates high-quality, ray-traced snapshots of scenes by externally calling a povray binary. To use this backend, povray should be installed and accessible on your executable path.

class plato.draw.povray.**Scene**(*primitives=[]*, *features={}*, *size=(40*, *30)*, *translation=(0*, *0*, *-50)*, *rotation=(1*, *0*, *0*, *0)*, *zoom=1*, *pixel_scale=20*, ***kwargs*)

A container to hold and display collections of primitives.

Scene keeps track of global information about a set of things to be rendered and handles configuration of optional (possibly backend-specific) rendering parameters.

Global information managed by a *Scene* includes the *size* of the viewing window, *translation* and *rotation* applied to the scene as a whole, and a *zoom* level.

Primitives can be added to a scene through the *primitives* argument of the constructor or the *add_primitive* method. Primitives can be retrieved by iterating over the scene:

```
for prim in scene:
    # (do something with prim)
```

Primitives can also be accessed in the order they were added to the scene using list-like syntax:

```
first_three_prims = scene[:3]
last_prim = scene[-1]
```

Optional rendering arguments are enabled as *features*, which are name-value pairs identifying a feature by name and any configuration of the feature in the value.

This Scene supports the following features:

- *antialiasing*: Enable antialiasing using the given value (default 0.3).
- *ambient_light*: Enable trivial ambient lighting. The given value indicates the magnitude of the light (in [0, 1]).
- *directional_light*: Add directional lights. The given value indicates the magnitude*direction normal vector.
- *multithreading*: Enable multithreaded rendering. The given value indicates the number of threads to use.

render()
Render all the shapes in this scene.

Returns povray string representing the entire scene

save(*filename*)
Save the scene, either as povray source or a rendered image.

Parameters **filename** – target filename to save the result into. If filename ends in .pov, save the povray source, otherwise call povray to render the image

show()
Render the scene to an image and display using ipython.

4.4.1 3D Graphics Primitives

class plato.draw.povray.**Box**(**args*, ***kwargs*)
A triclinic box frame.

This primitive draws a triclinic box centered at the origin. It is specified in terms of three lattice vector lengths *Lx*, *Ly*, *Lz* and tilt factors, defined using the hoomd-blue schema.

Rather than directly initializing via attributes, *Box* objects can also be automatically created from box-type objects using the `from_box()` method.

Examples:

```
Lx = Ly = Lz = 10
xy = xz = yz = 0
box_primitive = draw.Box(Lx=Lx, Ly=Ly, Lz=Lz, width=width, color=color)
box_tuple = (Lx, Ly, Lz, xy, xz, yz)
box_primitive = draw.Box.from_box(box_tuple)
```

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- Lx: Length of first box vector
- Ly: Length of second box vector
- Lz: Length of third box vector
- xy: Tilt factor between the first and second box vectors
- xz: Tilt factor between the first and third box vectors
- yz: Tilt factor between the second and third box vectors
- width: Width of box line segments
- color: Color, RGBA, [0, 1] for the box line segments

class plato.draw.povray.**ConvexPolyhedra**(***kwargs*)

A collection of identically-shaped convex polyhedra.

Each shape can have its own position, orientation, and color.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle
- outline: Outline width for all shapes
- outline: Outline width for all particles

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

outline
Outline width for all particles

positions
Position of each particle

vertices
Vertices in local coordinates for the shape, to be replicated for each particle

class plato.draw.povray.**ConvexSpheropolyhedra**(***kwargs*)

A collection of identically-shaped convex spheropolyhedra.

Each shape can have its own position, orientation, and color. The rounding radius is shared over all shapes.

This primitive has the following attributes:

- positions: Position of each particle

- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle
- radius: Rounding radius to be applied to all shapes

class plato.draw.povray.**Ellipsoids**(***kwargs*)
A collection of ellipsoids with identical dimensions.

Each ellipsoid can have its own position, orientation, and color. All shapes drawn by this primitive share common principal axis lengths.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- a: Radius in the x-direction
- b: Radius in the y-direction
- c: Radius in the z-direction

class plato.draw.povray.**Lines**(***kwargs*)
A collection of line segments.

Each segment can have a different color and width. *Lines* can be used in both 2D and 3D scenes, but they are currently not shaded and may look out of place in 3D.

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- cap_mode: Cap mode for lines (0: default, 1: round)

cap_mode
Cap mode for lines (0: default, 1: round)

colors
Color, RGBA, [0, 1] for each line segment

end_points
Ending coordinate for each line segment

start_points
Beginning coordinate for each line segment

widths
Width of each line segment

class plato.draw.povray.**Mesh**(***kwargs*)
A 3D triangle mesh.

Meshes are specified by an array of vertices and indices identifying triangles within that vertex array. Colors are assigned per-vertex and interpolated between vertices.

Meshes with a common set of vertices and face indices can be replicated multiple times using a set of positions and orientations. In order to set the color of individual replicas of the Mesh object, use the *shape_colors* and *shape_color_fraction* attributes.

This primitive has the following attributes:

- vertices: Vertex array specifying coordinates of the mesh nodes
- indices: Indices of the vertex array specifying individual triangles (Nx3)
- colors: Color, RGBA, [0, 1] for each vertex
- positions: Central positions for each mesh to be replicated
- orientations: Orientations for each mesh to be replicated
- shape_colors: Color, RGBA, [0, 1] for each replica (shape) of the mesh
- shape_color_fraction: Fraction of a vertex's color that should be assigned based on shape_colors

class plato.draw.povray.**Spheres**(***kwargs*)

A collection of spheres in 3D.

Each sphere can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle

class plato.draw.povray.**SphereUnions**(***kwargs*)

A collection of identical sphere-union bodies in 3D.

A *SphereUnions* object is a union of spheres, each of which has its own color, radius, and local position. The *SphereUnions* object can be rigidly rotated and translated via its position and orientation attributes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each sphere in the union
- points: Positions in local coordinates for the spheres in the union, to be replicated for each particle
- radii: Radius of each sphere in the union

4.5 Pythreejs Backend

The pythreejs backend renders scenes using three.js and is ideal for viewing scenes within Jupyter notebooks.

Note: To enable translucency in the pythreejs backend, a primitive must have the same value of alpha (less than 1) for all colors.

---

class plato.draw.pythreejs.**Scene**(**args*, ***kwargs*)

add_primitive(*prim*)

Adds a primitive to the scene.

disable (*name*, *strict=True*)
Disable an optional rendering feature.

Parameters

- **name** – Name of the feature to disable
- **strict** – if True, raise a KeyError if the feature was not enabled

enable (*name*, *auto_value=None*, ***parameters*)
Enable an optional rendering feature.

Parameters

- **name** – Name of the feature to enable
- **auto_value** – Shortcut for features with single-value configuration. If given as a positional argument, will be given the default configuration name 'value'.
- **parameters** – Keyword arguments specifying additional configuration options for the given feature

remove_primitive (*primitive*, *strict=True*)
Removes a primitive from the scene.

Parameters

- **primitive** – primitive to (attempt to) remove
- **strict** – If True, raise an IndexError if the primitive was not in the scene

rotation
(r, x, y, z) rotation quaternion to be applied to the scene as a whole.

size
Width and height, in scene units, of the viewport.

translation
(x, y, z) translation to be applied to the scene as a whole after rotating.

x is to the right, *y* is up, and *z* comes toward you out of the screen.

4.5.1 3D Graphics Primitives

class plato.draw.pythreejs.**Box**(**args*, ***kwargs*)
A triclinic box frame.

This primitive draws a triclinic box centered at the origin. It is specified in terms of three lattice vector lengths *Lx*, *Ly*, *Lz* and tilt factors, defined using the hoomd-blue schema.

Rather than directly initializing via attributes, *Box* objects can also be automatically created from box-type objects using the `from_box()` method.

Examples:

```
Lx = Ly = Lz = 10
xy = xz = yz = 0
box_primitive = draw.Box(Lx=Lx, Ly=Ly, Lz=Lz, width=width, color=color)
box_tuple = (Lx, Ly, Lz, xy, xz, yz)
box_primitive = draw.Box.from_box(box_tuple)
```

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- Lx: Length of first box vector
- Ly: Length of second box vector
- Lz: Length of third box vector
- xy: Tilt factor between the first and second box vectors
- xz: Tilt factor between the first and third box vectors
- yz: Tilt factor between the second and third box vectors
- width: Width of box line segments
- color: Color, RGBA, [0, 1] for the box line segments

class plato.draw.pythreejs.**ConvexPolyhedra**(***kwargs*)

A collection of identically-shaped convex polyhedra.

Each shape can have its own position, orientation, and color.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle
- outline: Outline width for all shapes

class plato.draw.pythreejs.**ConvexSpheropolyhedra**(***kwargs*)

A collection of identically-shaped convex spheropolyhedra.

Each shape can have its own position, orientation, and color. The rounding radius is shared over all shapes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle
- radius: Rounding radius to be applied to all shapes

class plato.draw.pythreejs.**Ellipsoids**(***kwargs*)

A collection of ellipsoids with identical dimensions.

Each ellipsoid can have its own position, orientation, and color. All shapes drawn by this primitive share common principal axis lengths.

This primitive has the following attributes:

- positions: Position of each particle

- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- a: Radius in the x-direction
- b: Radius in the y-direction
- c: Radius in the z-direction
- vertex_count: Number of vertices used to render ellipsoid

a
Radius in the x-direction

b
Radius in the y-direction

c
Radius in the z-direction

colors
Color, RGBA, [0, 1] for each particle

orientations
Orientation quaternion of each particle

positions
Position of each particle

vertex_count
Number of vertices used to render ellipsoid

class plato.draw.pythreejs.**Lines**(***kwargs*)
A collection of line segments.

Each segment can have a different color and width. *Lines* can be used in both 2D and 3D scenes, but they are currently not shaded and may look out of place in 3D.

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment

class plato.draw.pythreejs.**Mesh**(***kwargs*)
A 3D triangle mesh.

Meshes are specified by an array of vertices and indices identifying triangles within that vertex array. Colors are assigned per-vertex and interpolated between vertices.

Meshes with a common set of vertices and face indices can be replicated multiple times using a set of positions and orientations. In order to set the color of individual replicas of the Mesh object, use the *shape_colors* and *shape_color_fraction* attributes.

This primitive has the following attributes:

- vertices: Vertex array specifying coordinates of the mesh nodes
- indices: Indices of the vertex array specifying individual triangles (Nx3)
- colors: Color, RGBA, [0, 1] for each vertex

- positions: Central positions for each mesh to be replicated
- orientations: Orientations for each mesh to be replicated
- shape_colors: Color, RGBA, [0, 1] for each replica (shape) of the mesh
- shape_color_fraction: Fraction of a vertex's color that should be assigned based on shape_colors

class plato.draw.pythreejs.**Spheres**(***kwargs*)

A collection of spheres in 3D.

Each sphere can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- vertex_count: Number of vertices used to render sphere

colors
Color, RGBA, [0, 1] for each particle

positions
Position of each particle

radii
Radius of each particle

vertex_count
Number of vertices used to render sphere

4.6 Vispy Backend

The vispy backend uses vispy to render shapes interactively using openGL. It supports both desktop use with a variety of GUI backends and use inline in jupyter notebooks. While the GUI backends are essentially interchangeable, the notebook backend is more restrictive in its capabilities and some features are not currently available with it.

Select the **vispy** backend to use with the standard vispy mechanism before calling *Scene.show()*:

```
import vispy, vispy.app
# use in ipython notebook
vispy.app.use_app('ipynb_webgl')
# use pyside2
vispy.app.use_app('pyside2')
scene = plato.draw.vispy.Scene(...)
scene.show()
vispy.app.run()
```

Mouse controls: Live vispy windows support rotating the scene in three dimensions by dragging the mouse. Dragging while holding the control or meta keys causes the mouse movement to rotate the scene about the z axis and zoom in or out. Holding the alt key while dragging the mouse cursor will translate the scene; for two-dimensional scenes, it may be preferable to enable the *pan* feature, which causes mouse motion to translate, rather than rotate, the scene by default.

Keyboard controls: Live vispy windows also support controlling the camera via the keyboard. Control or meta in conjunction with the arrow keys rotate the system in 15 degree increments. The same functionality is mapped to the

I (up), J (left), K (down), and L (right) keys. X, Y, and Z directly snap the scene to look down the x, y, or z axes, respectively.

class plato.draw.vispy.**Scene**(**args*, *canvas_kwargs={}*, ***kwargs*)

A container to hold and display collections of primitives.

Scene keeps track of global information about a set of things to be rendered and handles configuration of optional (possibly backend-specific) rendering parameters.

Global information managed by a *Scene* includes the *size* of the viewing window, *translation* and *rotation* applied to the scene as a whole, and a *zoom* level.

Primitives can be added to a scene through the *primitives* argument of the constructor or the *add_primitive* method. Primitives can be retrieved by iterating over the scene:

```
for prim in scene:
    # (do something with prim)
```

Primitives can also be accessed in the order they were added to the scene using list-like syntax:

```
first_three_prims = scene[:3]
last_prim = scene[-1]
```

Optional rendering arguments are enabled as *features*, which are name-value pairs identifying a feature by name and any configuration of the feature in the value.

This Scene supports the following features:

- *pan*: If enabled, mouse movement will translate the scene instead of rotating it
- *directional_light*: Add directional lights. The given value indicates the magnitude*direction normal vector.
- *ambient_light*: Enable trivial ambient lighting. The given value indicates the magnitude of the light (in [0, 1]).
- *translucency*: Enable order-independent transparency rendering
- *fxaa*: Enable fast approximate anti-aliasing
- *ssao*: Enable screen space ambient occlusion
- *additive_rendering*: Enable additive rendering. This mode is good for visualizing densities projected through the viewing direction. Takes an optional 'invert' argument to invert the additive rendering (i.e., black-on-white instead of white-on-black).
- *outlines*: Enable cartoony outlines. The given value indicates the width of the outlines (start small, perhaps 1e-5 to 1e-3).
- *select_point*: Perform a callback on the next mouse click. The callback receives the clicked position (in the coordinate system of the scene unless the 'units' parameter is set to another valid target for `Scene.transform()`) and any additional keyword arguments passed in the feature config.
- *select_rect*: Perform a callback on the next mouse drag event. The callback receives the start and end point of the selected area (in the coordinate system of the scene unless the 'units' parameter is set to another valid target for `Scene.transform()`) and any additional keyword arguments passed in the feature config.
- *static*: Enable static rendering. When possible (when vispy is using a non-notebook backend), display a statically-rendered image of a scene instead of the live webGL version when *Scene.show()* is called.

add_primitive(*primitive*)

Adds a primitive to the scene.

disable(*name*, *strict=True*)

Disable an optional rendering feature.

Parameters

- **name** – Name of the feature to disable
- **strict** – if True, raise a KeyError if the feature was not enabled

enable(*name*, *auto_value=None*, ***parameters*)
Enable an optional rendering feature.

Parameters

- **name** – Name of the feature to enable
- **auto_value** – Shortcut for features with single-value configuration. If given as a positional argument, will be given the default configuration name 'value'.
- **parameters** – Keyword arguments specifying additional configuration options for the given feature

render()
Have vispy redraw this Scene object.

save(*filename*)
Render and save an image of this Scene.

Parameters **filename** – target filename to save the image into

show()
Display this Scene object.

size
Width and height, in scene units, of the viewport.

4.6.1 2D Graphics Primitives

class plato.draw.vispy.**Arrows2D**(**args*, ***kwargs*)
A collection of 2D arrows.

Each arrow has an independent position, orientation, color, and magnitude. The shape of arrows can be configured by changing its *vertices* attribute. The default orientation and scale of the vertices is an arrow centered at (0, 0), pointing in the (1, 0) direction, with length 1.

The origin of the arrows can be shifted to have the base lie on the given position by modifying *vertices*:

```
arrows.vertices = arrows.vertices + (0.5, 0)
```

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

This primitive has the following opengl-specific attributes:

- outline: Outline width for shapes

camera
Internal: 4x4 Camera matrix for world projection

outline
Outline width for shapes

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

class plato.draw.vispy.**DiskUnions**(**args*, ***kwargs*)
A collection of identical disk-union bodies in 2D.

A *DiskUnions* object consists of one or more disks, each with its own radius and color. Each object has its own position and orientation that affect the final position of the constituent disks.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each disk in the union
- points: Positions in local coordinates for the disks in the union, to be replicated for each particle
- radii: Radius of each disk in the union
- outline: Outline width for all particles

This primitive has the following opengl-specific attributes:

- outline: Outline width for shapes

camera
Internal: 4x4 Camera matrix for world projection

outline
Outline width for shapes

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

class plato.draw.vispy.**Disks**(**args*, ***kwargs*)
A collection of disks in 2D.

Each disk can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- outline: Outline width for all particles

This primitive has the following opengl-specific attributes:

- outline: Outline for all particles

camera
Internal: 4x4 Camera matrix for world projection

outline
Outline for all particles

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

class plato.draw.vispy.**Polygons**(**args*, ***kwargs*)
A collection of polygons.

A *Polygons* object has a common shape for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

This primitive has the following opengl-specific attributes:

- outline: Outline width for shapes

camera
Internal: 4x4 Camera matrix for world projection

outline
Outline width for shapes

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

class plato.draw.vispy.**Spheropolygons**(**args*, ***kwargs*)
A collection of rounded polygons.

A *Spheropolygons* object has a common shape and rounding radius for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles
- radius: Rounding radius for all particles

This primitive has the following opengl-specific attributes:

- outline: Outline width for shapes
- radius: Rounding radius for shapes

camera
Internal: 4x4 Camera matrix for world projection

outline
Outline width for shapes

radius
Rounding radius for shapes

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

class plato.draw.vispy.**Voronoi**(**args*, ***kwargs*)
A Voronoi diagram of a set of 2D points.

The region of space nearest to each given point will be colored by the color associated with that point.

This primitive has the following attributes:

- positions: Position of each point
- colors: Color, RGBA, [0, 1] for each point

This primitive has the following opengl-specific attributes:

- radius: Maximum distance between displayed points
- clip_extent: Matrix specifying areas to not display when dot(clip_extent, position) is outside [-1, 1]

camera
Internal: 4x4 Camera matrix for world projection

clip_extent
Matrix specifying areas to not display when dot(clip_extent, position) is outside [-1, 1]

radius
Maximum distance between displayed points

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

4.6.2 3D Graphics Primitives

class plato.draw.vispy.**Box**(**args*, ***kwargs*)
A triclinic box frame.

This primitive draws a triclinic box centered at the origin. It is specified in terms of three lattice vector lengths *Lx*, *Ly*, *Lz* and tilt factors, defined using the hoomd-blue schema.

Rather than directly initializing via attributes, *Box* objects can also be automatically created from box-type objects using the `from_box()` method.

Examples:

```
Lx = Ly = Lz = 10
xy = xz = yz = 0
box_primitive = draw.Box(Lx=Lx, Ly=Ly, Lz=Lz, width=width, color=color)
box_tuple = (Lx, Ly, Lz, xy, xz, yz)
box_primitive = draw.Box.from_box(box_tuple)
```

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- Lx: Length of first box vector
- Ly: Length of second box vector
- Lz: Length of third box vector
- xy: Tilt factor between the first and second box vectors
- xz: Tilt factor between the first and third box vectors
- yz: Tilt factor between the second and third box vectors
- width: Width of box line segments
- color: Color, RGBA, [0, 1] for the box line segments

class plato.draw.vispy.**ConvexPolyhedra**(**args*, ***kwargs*)

A collection of identically-shaped convex polyhedra.

Each shape can have its own position, orientation, and color.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle
- outline: Outline width for all shapes

This primitive has the following opengl-specific attributes:

- outline: Outline width for shapes
- light_levels: Number of light levels to quantize to (0: disable)

ambientLight

Internal: Ambient (minimum) light level for all surfaces

camera

Internal: 4x4 Camera matrix for world projection

diffuseLight

Internal: Diffuse light direction*magnitude

light_levels

Number of light levels to quantize to (0: disable)

outline
Outline width for shapes

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

transparency_mode
Internal: Transparency stage (<0: opaque, 0: all, 1: translucency stage 1, 2: translucency stage 2)

class plato.draw.vispy.**ConvexSpheropolyhedra**(**args*, ***kwargs*)
A collection of identically-shaped convex spheropolyhedra.

Each shape can have its own position, orientation, and color. The rounding radius is shared over all shapes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle
- radius: Rounding radius to be applied to all shapes

This primitive has the following opengl-specific attributes:

- radius: Rounding radius to be applied to all shapes
- light_levels: Number of light levels to quantize to (0: disable)

ambientLight
Internal: Ambient (minimum) light level for all surfaces

camera
Internal: 4x4 Camera matrix for world projection

diffuseLight
Internal: Diffuse light direction*magnitude

light_levels
Number of light levels to quantize to (0: disable)

radius
Rounding radius to be applied to all shapes

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

transparency_mode
Internal: Transparency stage (<0: opaque, 0: all, 1: translucency stage 1, 2: translucency stage 2)

class plato.draw.vispy.**Ellipsoids**(**args*, ***kwargs*)
A collection of ellipsoids with identical dimensions.

Each ellipsoid can have its own position, orientation, and color. All shapes drawn by this primitive share common principal axis lengths.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- a: Radius in the x-direction
- b: Radius in the y-direction
- c: Radius in the z-direction

This primitive has the following opengl-specific attributes:

- a: Radius in the x-direction
- b: Radius in the y-direction
- c: Radius in the z-direction
- light_levels: Number of light levels to quantize to (0: disable)
- outline: Outline for all particles

a
Radius in the x-direction

ambientLight
Internal: Ambient (minimum) light level for all surfaces

b
Radius in the y-direction

c
Radius in the z-direction

camera
Internal: 4x4 Camera matrix for world projection

diffuseLight
Internal: Diffuse light direction*magnitude

light_levels
Number of light levels to quantize to (0: disable)

outline
Outline for all particles

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

transparency_mode
Internal: Transparency stage (<0: opaque, 0: all, 1: translucency stage 1, 2: translucency stage 2)

class plato.draw.vispy.**Lines**(**args*, ***kwargs*)
A collection of line segments.

Each segment can have a different color and width. *Lines* can be used in both 2D and 3D scenes, but they are currently not shaded and may look out of place in 3D.

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment

This primitive has the following opengl-specific attributes:

- cap_mode: Cap mode for lines (0: default, 1: round)

ambientLight
Internal: Ambient (minimum) light level for all surfaces

camera
Internal: 4x4 Camera matrix for world projection

cap_mode
Cap mode for lines (0: default, 1: round)

diffuseLight
Internal: Diffuse light direction*magnitude

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

transparency_mode
Internal: Transparency stage (<0: opaque, 0: all, 1: translucency stage 1, 2: translucency stage 2)

class plato.draw.vispy.**Mesh**(**args*, ***kwargs*)
A 3D triangle mesh.

Meshes are specified by an array of vertices and indices identifying triangles within that vertex array. Colors are assigned per-vertex and interpolated between vertices.

Meshes with a common set of vertices and face indices can be replicated multiple times using a set of positions and orientations. In order to set the color of individual replicas of the Mesh object, use the *shape_colors* and *shape_color_fraction* attributes.

This primitive has the following attributes:

- vertices: Vertex array specifying coordinates of the mesh nodes
- indices: Indices of the vertex array specifying individual triangles (Nx3)
- colors: Color, RGBA, [0, 1] for each vertex
- positions: Central positions for each mesh to be replicated
- orientations: Orientations for each mesh to be replicated
- shape_colors: Color, RGBA, [0, 1] for each replica (shape) of the mesh
- shape_color_fraction: Fraction of a vertex's color that should be assigned based on shape_colors

This primitive has the following opengl-specific attributes:

- light_levels: Number of light levels to quantize to (0: disable)
- shape_color_fraction: Fraction of a vertex's color that should be assigned based on shape_colors

ambientLight
Internal: Ambient (minimum) light level for all surfaces

camera
Internal: 4x4 Camera matrix for world projection

diffuseLight
Internal: Diffuse light direction*magnitude

light_levels
Number of light levels to quantize to (0: disable)

rotation
Internal: Rotation to be applied to each scene as a quaternion

shape_color_fraction
Fraction of a vertex's color that should be assigned based on shape_colors

translation
Internal: Translation to be applied to the scene

transparency_mode
Internal: Transparency stage (<0: opaque, 0: all, 1: translucency stage 1, 2: translucency stage 2)

class plato.draw.vispy.**SpherePoints**(**args*, ***kwargs*)
A collection of points, useful for illustrating 3D density maps.

This primitive has the following attributes:

- points: Points to be rendered
- blur: Blurring factor dictating the size of each point
- intensity: Scaling factor dictating the magnitude of the color value of each point
- on_surface: True if the points should always be projected onto the surface of a sphere

This primitive has the following opengl-specific attributes:

- blur: Blurring factor dictating the size of each point
- intensity: Scaling factor dictating the magnitude of the color value of each point
- on_surface: True if the points should always be projected onto the surface of a sphere
- radius: Radius of the sphere to normalize to
- draw_front: If True, draw only the points facing the viewer

blur
Blurring factor dictating the size of each point

camera
Internal: 4x4 Camera matrix for world projection

draw_front
If True, draw only the points facing the viewer

intensity
Scaling factor dictating the magnitude of the color value of each point

inverse_size
Internal: inverse size of the given points array

on_surface
True if the points should always be projected onto the surface of a sphere

points
Points to be rendered

radius
Radius of the sphere to normalize to

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

class plato.draw.vispy.**Spheres**(**args*, ***kwargs*)
A collection of spheres in 3D.

Each sphere can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle

This primitive has the following opengl-specific attributes:

- light_levels: Number of light levels to quantize to (0: disable)
- outline: Outline for all particles

ambientLight
Internal: Ambient (minimum) light level for all surfaces

camera
Internal: 4x4 Camera matrix for world projection

diffuseLight
Internal: Diffuse light direction*magnitude

light_levels
Number of light levels to quantize to (0: disable)

outline
Outline for all particles

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

transparency_mode
Internal: Transparency stage (<0: opaque, 0: all, 1: translucency stage 1, 2: translucency stage 2)

class plato.draw.vispy.**SphereUnions**(**args*, ***kwargs*)
A collection of identical sphere-union bodies in 3D.

A *SphereUnions* object is a union of spheres, each of which has its own color, radius, and local position. The *SphereUnions* object can be rigidly rotated and translated via its position and orientation attributes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each sphere in the union
- points: Positions in local coordinates for the spheres in the union, to be replicated for each particle

- radii: Radius of each sphere in the union

This primitive has the following opengl-specific attributes:

- light_levels: Number of light levels to quantize to (0: disable)
- outline: Outline for all particles

ambientLight
Internal: Ambient (minimum) light level for all surfaces

camera
Internal: 4x4 Camera matrix for world projection

diffuseLight
Internal: Diffuse light direction*magnitude

light_levels
Number of light levels to quantize to (0: disable)

outline
Outline for all particles

rotation
Internal: Rotation to be applied to each scene as a quaternion

translation
Internal: Translation to be applied to the scene

transparency_mode
Internal: Transparency stage (<0: opaque, 0: all, 1: translucency stage 1, 2: translucency stage 2)

4.7 Zdog Backend

The zdog backend uses zdog to render shapes. Zdog is an HTML canvas-based engine that works best for simple, cartoon-style illustrations. Plato's implementation works inside notebook environments and also supports rendering standalone HTML for inclusion in other pages.

class plato.draw.zdog.**Scene**(*primitives=[]*, *features={}*, *size=(40, 30)*, *translation=(0, 0, -50)*, *rotation=(1, 0, 0, 0)*, *zoom=1*, *pixel_scale=20*, ***kwargs*)
A container to hold and display collections of primitives.

Scene keeps track of global information about a set of things to be rendered and handles configuration of optional (possibly backend-specific) rendering parameters.

Global information managed by a *Scene* includes the *size* of the viewing window, *translation* and *rotation* applied to the scene as a whole, and a *zoom* level.

Primitives can be added to a scene through the *primitives* argument of the constructor or the *add_primitive* method. Primitives can be retrieved by iterating over the scene:

```
for prim in scene:
    # (do something with prim)
```

Primitives can also be accessed in the order they were added to the scene using list-like syntax:

```
first_three_prims = scene[:3]
last_prim = scene[-1]
```

Optional rendering arguments are enabled as *features*, which are name-value pairs identifying a feature by name and any configuration of the feature in the value.

This Scene supports the following features:

- *ambient_light*: Enable trivial ambient lighting. The given value indicates the magnitude of the light (in [0, 1]).
- *directional_light*: Add directional lights. The given value indicates the magnitude*direction normal vector.
- *pan*: Translate, rather than rotate, when dragging with the mouse

render()
Render all the shapes in this scene.

Returns HTML string contents to be displayed

save(*filename*)
Save the scene as an HTML file.

Parameters **filename** – target filename to save the result into

show()
Render the scene to an image and display using ipython.

4.7.1 2D Graphics Primitives

class plato.draw.zdog.**Arrows2D**(**args*, ***kwargs*)
A collection of 2D arrows.

Each arrow has an independent position, orientation, color, and magnitude. The shape of arrows can be configured by changing its *vertices* attribute. The default orientation and scale of the vertices is an arrow centered at (0, 0), pointing in the (1, 0) direction, with length 1.

The origin of the arrows can be shifted to have the base lie on the given position by modifying *vertices*:

```
arrows.vertices = arrows.vertices + (0.5, 0)
```

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

class plato.draw.zdog.**Disks**(***kwargs*)
A collection of disks in 2D.

Each disk can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- outline: Outline width for all particles

colors
Color, RGBA, [0, 1] for each particle

outline
Outline width for all particles

positions
Position of each particle

radii
Radius of each particle

class plato.draw.zdog.**Polygons**(***kwargs*)
A collection of polygons.

A *Polygons* object has a common shape for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

class plato.draw.zdog.**Spheropolygons**(***kwargs*)
A collection of rounded polygons.

A *Spheropolygons* object has a common shape and rounding radius for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles
- radius: Rounding radius for all particles

4.7.2 3D Graphics Primitives

class plato.draw.zdog.**Box**(**args*, ***kwargs*)
A triclinic box frame.

This primitive draws a triclinic box centered at the origin. It is specified in terms of three lattice vector lengths *Lx*, *Ly*, *Lz* and tilt factors, defined using the hoomd-blue schema.

Rather than directly initializing via attributes, *Box* objects can also be automatically created from box-type objects using the `from_box()` method.

Examples:

```
Lx = Ly = Lz = 10
xy = xz = yz = 0
box_primitive = draw.Box(Lx=Lx, Ly=Ly, Lz=Lz, width=width, color=color)
box_tuple = (Lx, Ly, Lz, xy, xz, yz)
box_primitive = draw.Box.from_box(box_tuple)
```

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- Lx: Length of first box vector
- Ly: Length of second box vector
- Lz: Length of third box vector
- xy: Tilt factor between the first and second box vectors
- xz: Tilt factor between the first and third box vectors
- yz: Tilt factor between the second and third box vectors
- width: Width of box line segments
- color: Color, RGBA, [0, 1] for the box line segments

class plato.draw.zdog.**ConvexPolyhedra**(***kwargs*)

A collection of identically-shaped convex polyhedra.

Each shape can have its own position, orientation, and color.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle
- outline: Outline width for all shapes

class plato.draw.zdog.**ConvexSpheropolyhedra**(***kwargs*)

A collection of identically-shaped convex spheropolyhedra.

Each shape can have its own position, orientation, and color. The rounding radius is shared over all shapes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle
- radius: Rounding radius to be applied to all shapes

class plato.draw.zdog.**Lines**(***kwargs*)
A collection of line segments.

Each segment can have a different color and width. *Lines* can be used in both 2D and 3D scenes, but they are currently not shaded and may look out of place in 3D.

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment

class plato.draw.zdog.**Spheres**(***kwargs*)
A collection of spheres in 3D.

Each sphere can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- light_levels: Number of quantized light levels to use

colors
Color, RGBA, [0, 1] for each particle

light_levels
Number of quantized light levels to use

positions
Position of each particle

radii
Radius of each particle

4.8 Imperative API

The *imp* module defines an imperative API for convenient, immediate visualization of results without directly creating separate primitive and scene objects. The set of available primitives and attributes are the same as in *plato.draw*, but the functions in this module are named as lowercase_with_underscores rather than CamelCase class names. Final scenes can be shown either directly, allowing for more careful selection of backends and passing arguments to the underlying scene by using *show()* or automatically by using the *plato.imp* IPython extension.

Examples:

```
import plato.imp as imp
imp.spheres(positions=[1, 0, 0])
imp.lines(start_points=(0, 1, 0), end_points=(1, 0, 0))
imp.show(backend='zdog', zoom=10)

# the line below causes cell contents to automatically be shown in jupyter notebooks
%load_ext plato.imp
imp.polygons(outline=.1)
imp.arrows_2D(positions=(-1, 0))
```

plato.imp.**clear**()
Clears the imperative state.

plato.imp.**get**(*backend=None*, ***kwargs*)
Returns the last-shown imperative scene, or creates a new one.

This method returns the most recent scene, either that has been shown via a call to *show()* or defined by calling primitive-creating functions. If a new scene is created, the user is responsible for calling *Scene.show()* as appropriate.

plato.imp.**show**(*backend=None*, ***kwargs*)
Immediately show all pending primitives that have been created.

A backend name can optionally be specified, but all other keyword arguments are passed to the *plato.draw.Scene* constructor. If no backend is specified, a backend that can be imported and supports all the pending primitives will be selected.

plato.imp.**arrows2D**(**args*, ***kwargs*)
Generates and immediately displays the object described below.

A collection of 2D arrows.

Each arrow has an independent position, orientation, color, and magnitude. The shape of arrows can be configured by changing its *vertices* attribute. The default orientation and scale of the vertices is an arrow centered at (0, 0), pointing in the (1, 0) direction, with length 1.

The origin of the arrows can be shifted to have the base lie on the given position by modifying *vertices*:

```
arrows.vertices = arrows.vertices + (0.5, 0)
```

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

plato.imp.**box**(**args*, ***kwargs*)
Generates and immediately displays the object described below.

A triclinic box frame.

This primitive draws a triclinic box centered at the origin. It is specified in terms of three lattice vector lengths *Lx*, *Ly*, *Lz* and tilt factors, defined using the hoomd-blue schema.

Rather than directly initializing via attributes, *Box* objects can also be automatically created from box-type objects using the `from_box()` method.

Examples:

```
Lx = Ly = Lz = 10
xy = xz = yz = 0
box_primitive = draw.Box(Lx=Lx, Ly=Ly, Lz=Lz, width=width, color=color)
box_tuple = (Lx, Ly, Lz, xy, xz, yz)
box_primitive = draw.Box.from_box(box_tuple)
```

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment
- colors: Color, RGBA, [0, 1] for each line segment
- Lx: Length of first box vector
- Ly: Length of second box vector
- Lz: Length of third box vector
- xy: Tilt factor between the first and second box vectors
- xz: Tilt factor between the first and third box vectors
- yz: Tilt factor between the second and third box vectors
- width: Width of box line segments
- color: Color, RGBA, [0, 1] for the box line segments

plato.imp.**convex_polyhedra**(***kwargs*)

Generates and immediately displays the object described below.

A collection of identically-shaped convex polyhedra.

Each shape can have its own position, orientation, and color.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle
- outline: Outline width for all shapes

plato.imp.**convex_spheropolyhedra**(***kwargs*)

Generates and immediately displays the object described below.

A collection of identically-shaped convex spheropolyhedra.

Each shape can have its own position, orientation, and color. The rounding radius is shared over all shapes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle
- radius: Rounding radius to be applied to all shapes

plato.imp.**disks**(***kwargs*)

Generates and immediately displays the object described below.

A collection of disks in 2D.

Each disk can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle
- outline: Outline width for all particles

plato.imp.**disk_unions**(***kwargs*)

Generates and immediately displays the object described below.

A collection of identical disk-union bodies in 2D.

A *DiskUnions* object consists of one or more disks, each with its own radius and color. Each object has its own position and orientation that affect the final position of the constituent disks.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each disk in the union
- points: Positions in local coordinates for the disks in the union, to be replicated for each particle
- radii: Radius of each disk in the union
- outline: Outline width for all particles

plato.imp.**ellipsoids**(***kwargs*)

Generates and immediately displays the object described below.

A collection of ellipsoids with identical dimensions.

Each ellipsoid can have its own position, orientation, and color. All shapes drawn by this primitive share common principal axis lengths.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- a: Radius in the x-direction
- b: Radius in the y-direction
- c: Radius in the z-direction

plato.imp.**lines**(***kwargs*)

Generates and immediately displays the object described below.

A collection of line segments.

Each segment can have a different color and width. *Lines* can be used in both 2D and 3D scenes, but they are currently not shaded and may look out of place in 3D.

This primitive has the following attributes:

- start_points: Beginning coordinate for each line segment
- end_points: Ending coordinate for each line segment
- widths: Width of each line segment

- colors: Color, RGBA, [0, 1] for each line segment

plato.imp.**mesh**(***kwargs*)
Generates and immediately displays the object described below.

A 3D triangle mesh.

Meshes are specified by an array of vertices and indices identifying triangles within that vertex array. Colors are assigned per-vertex and interpolated between vertices.

Meshes with a common set of vertices and face indices can be replicated multiple times using a set of positions and orientations. In order to set the color of individual replicas of the Mesh object, use the *shape_colors* and *shape_color_fraction* attributes.

This primitive has the following attributes:

- vertices: Vertex array specifying coordinates of the mesh nodes
- indices: Indices of the vertex array specifying individual triangles (Nx3)
- colors: Color, RGBA, [0, 1] for each vertex
- positions: Central positions for each mesh to be replicated
- orientations: Orientations for each mesh to be replicated
- shape_colors: Color, RGBA, [0, 1] for each replica (shape) of the mesh
- shape_color_fraction: Fraction of a vertex's color that should be assigned based on shape_colors

plato.imp.**polygons**(***kwargs*)
Generates and immediately displays the object described below.

A collection of polygons.

A *Polygons* object has a common shape for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles

plato.imp.**sphere_points**(***kwargs*)
Generates and immediately displays the object described below.

A collection of points, useful for illustrating 3D density maps.

This primitive has the following attributes:

- points: Points to be rendered
- blur: Blurring factor dictating the size of each point
- intensity: Scaling factor dictating the magnitude of the color value of each point
- on_surface: True if the points should always be projected onto the surface of a sphere

plato.imp.**spheres**(***kwargs*)
Generates and immediately displays the object described below.

A collection of spheres in 3D.

Each sphere can have a different color and diameter.

This primitive has the following attributes:

- positions: Position of each particle
- colors: Color, RGBA, [0, 1] for each particle
- radii: Radius of each particle

plato.imp.**sphere_unions**(***kwargs*)

Generates and immediately displays the object described below.

A collection of identical sphere-union bodies in 3D.

A *SphereUnions* object is a union of spheres, each of which has its own color, radius, and local position. The *SphereUnions* object can be rigidly rotated and translated via its position and orientation attributes.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each sphere in the union
- points: Positions in local coordinates for the spheres in the union, to be replicated for each particle
- radii: Radius of each sphere in the union

plato.imp.**spheropolygons**(***kwargs*)

Generates and immediately displays the object described below.

A collection of rounded polygons.

A *Spheropolygons* object has a common shape and rounding radius for the whole collection. Each shape can have a different orientation and color. Vertices should be specified in counterclockwise order.

This primitive has the following attributes:

- positions: Position of each particle
- orientations: Orientation quaternion of each particle
- colors: Color, RGBA, [0, 1] for each particle
- vertices: Vertices in local coordinates for the interior (non-rounded) shape, to be replicated for each particle (CCW order)
- outline: Outline width for all particles
- radius: Rounding radius for all particles

plato.imp.**voronoi**(***kwargs*)

Generates and immediately displays the object described below.

A Voronoi diagram of a set of 2D points.

The region of space nearest to each given point will be colored by the color associated with that point.

This primitive has the following attributes:

- positions: Position of each point
- colors: Color, RGBA, [0, 1] for each point

4.9 Troubleshooting and FAQ

Note: Depending on which backends you want to use, there may be additional steps required for installation; consult the advice here.

When starting a jupyter notebook, I get a "Permission denied" error for a linking operation

This may be related to jupyter upgrades. Manually remove the symlink and the notebook should be able to proceed once more.

When running in a jupyter notebook, nothing is displayed

The solution to this problem depends on more details.

- *The canvas is displayed entirely black with "Uncaught TypeError: Cannot read property 'handle' of undefined" (or similar language)*: After the *canvas.show()* command in the cell, add a line *import time;time.sleep(.1)*. You may need to increase the argument of *time.sleep()*. This is due to a race condition in vispy.
- *I get an error 404 in the browser console for vispy.min.js* - Make sure that jupyter, ipywidgets, and all of the jupyter components are up to date (and have compatible versions, see https://bitbucket.org/snippets/glotzer/nMg8Gr/plato-dependency-installation-tips).
- *I get an error 404 in the browser console for webgl-backend.js* - Try removing your jupyter notebook cache (~/.jupyter and ~/Library/Jupyter on OSX) and restarting jupyter
- Make sure the *jupyter* executable you are using is in the same virtualenv or conda environment as plato and its dependencies

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

D

E

F

G

I

R

S

T

V

W

X

Y