Deliverable 3: Feature Planning

Benjamin's Grooper
CSCD01
Anya Tafliovich
David Gao, Simon Ngo, Andrei Grumazescu, Raunak Mathur, Tony Ng
Winter 2019

Table of Contents

Feature Description

https://github.com/matplotlib/matplotlib/issues/13575

Using the façade design pattern, we plan to create a modified version of the requested feature described at issue #13575. This feature will allow the user to create a customized 3d plot by only using one function with a few optional arguments. For instance, the user would have the ability of plotting a 3d line using a string representation of it's equation and highlighting/plotting specific points on that line with projections from each axis, where these projections can have different colours, be dotted/solid, and where each of these projection are different depending on the specified point. The user can also use the functionality of plotting individual points and projections without giving a 3d line, or only plot a 3d line, which will have its own properties.

https://github.com/matplotlib/matplotlib/issues/6027

After analyzing issue #6027, our group noticed that there is no current way to assign gradient shading to a 3d polygon. This feature aims to provide the ability for users to assign gradient shading to 3d polygons in Poly3DCollection.

A Poly3DCollection contains an parameter called "verts", which is an array of all the polygons, each represented by its own array. Inside each individual array are all of the vertices of that polygon, ordered so that for some element x_i and x_{i+1} there is an edge between those two vertices.

Implementation Plans

Line/Point Plotting Facade:

The approach would be to create a new function in pyplot.py called something along the lines of "projection_plot". It would take in as its parameters: points to plot, options for customizing projection lines, axes limits, and the optional parameters for a line equation to generate a line as well as options for customizing that line. Using methods from the classes Axes3D, Axes, and Figure, we will create the desired plot with the given parameters so that the user can simply call this function when they want to be able to clearly view points with projection lines. First we will generate the axes with arrow heads to model a typical 3D plot with X, Y, and Z labels. This will be done with the quiver method found in Axes3D. Then we will iterate over the collection of points and also iterate over the collection of the custom options for the projection lines to be used for these points to individually plot each point and projection line starting from axes. In the case that the user wishes to also plot a line, we will parse the line equation that is passed as a string to plot on our axes using the same method as before to customize the properties of this line.

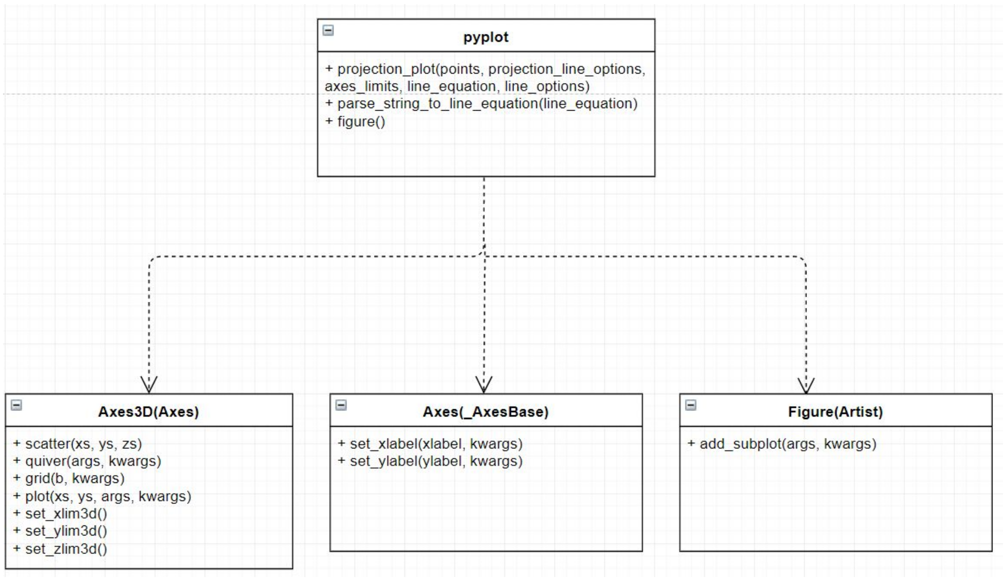
Polygon Gradient:

The approach is for every pair of vertices which share an edge, to create a unit vector to represent the edge, and then to create square polygons going down that edge aligned with the unit vector, until it reaches the next vertex. The colour of the polygon will be based on a linear interpolation, a weighted average between the colour of the two vertices which form the edge as well as the colour of the barycenter (which is determined by taking a weighted average of all the corner colours), and the weights of each color is how far the polygon is from the corresponding vertex. Once this entire edge has been completely plotted as small polygons, repeat the same process for the next edge. Once every edge has been plotted, assign new corners as the innermost corner of every currently plotted corner and start the process again, until the entire shape is shaded. A finished process may use triangle polygons for corners to make the rest of the polygon more representable by squares. In this process the user should also be able to set the size of the pixel by setting the length of the square, so heavy duty systems can get a very smooth gradient, and slower systems can still get a gradient.

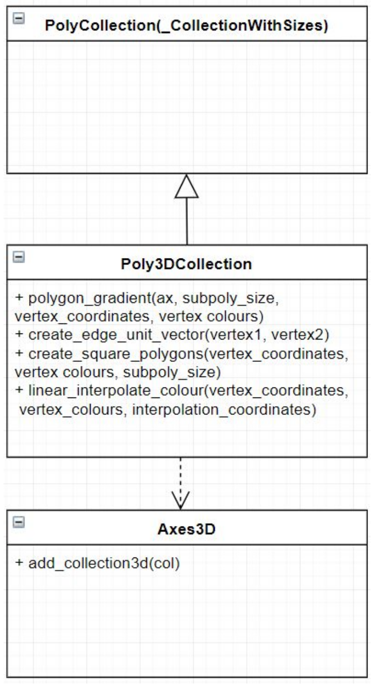
This means adding some kind of object that can store colour alongside a vertex, although this might be as simple as a dictionary.

UML Diagrams

UML displaying interaction between pyplot and classes within the subsystem for plotting points/lines on 3D axes along with options. The line_equation and line_options parameters within projection_plot method are optional.


pyplot
+ projection_plot(points, projection_line_options, axes_limits, line_equation, line_options)
+ parse_string_to_line_equation(line_equation)
+ figure()
Axes3D(Axes)
+ scatter(xs, ys, zs)
+ quiver(args, kwargs)
+ grid(b, kwargs)
+ plot(xs, ys, args, kwargs)
+ set_xlim3d()
+ set_ylim3d()
+ set_zlim3d()
Axes(_AxesBase)
+ set_xlabel(xlabel, kwargs)
+ set_ylabel(ylabel, kwargs)
Figure(Artist)
+ add_subplot(args, kwargs)

UML displaying methods used for generating a gradient for a polygon and plotting it on the axes.


PolyCollection(_CollectionWithSizes)
Poly3DCollection
+ polygon_gradient(ax, subpoly_size, vertex_coordinates, vertex colours)
+ create_edge_unit_vector(vertex1, vertex2)
+ create_square_polygons(vertex_coordinates, vertex colours, subpoly_size)
+ linear_interpolate_colour(vertex_coordinates, vertex_colours, interpolation_coordinates)
Axes3D
+ add_collection3d(col)

Code Traces

Line/Point Plotting Facade:

```
projection_plot( [ [point 1], … , [point n] ],  [ [dashed/solid/none, line
colour, arrowhead], … , [same but for point n] ], [x axis limit, y axis
limit, z axis limit], string equation of a line, line properties )
      --> fig = plt.figure()
      --> ax = fig.add_subplot(3d)
      --> ax.plot(given points and projection properties)
      --> parse_string_to_line_equation(line_equation)
      --> ax.plot(parsed equation and line properties)
      --> ax.quiver(arrowheads)
      --> ax.set_xlabel/set_ylabel/set_zlabel(x,y,z)
      --> ax.set_xlim/set_ylim/set_zlim(given xlim, ylim, zlim)
```

Polygon Gradient:

```
polygon_gradient(ax, subpoly_size, [co-ords for vertex 1, … , co-ords for
vertex n], [colour at vertex 1 in RGB, … , colour at vertex n in RGB])
      --> create_square_polygons([vertex1, vertex2, … , vertexn], [colour1,
colour2, …, colourn], subpoly_size) --> create_edge_unit_vector(v1,v2)
            -> linear interpolate colour([vertex1, vertex2, … , vertexn],
[colour1, colour2, …, colourn], interpolation_coords)
            --> ax.add_collection3d(subpoly)
```