Preview

This article explains how to use 3D computer aided design (CAD) software to create 3D models of components and board outlines when using the popular KiCad software to create printed circuit boards (PCBs). The 3D CAD software used is CadQuery. CadQuery uses the Python programming language to design 2D and 3D objects through code.

Using CadQuery to create 3D models of components and board outlines in KiCad

Matthew Oppenheim
Lancaster University

KiCad[1] has rapidly become one of the most popular open-source and freely available packages for schematic entry and PCB layout. Every design package has its quirks and characteristics. I have spent more than a few hours searching for solutions for problems with the software. In this article I present two solutions solved through using 3D CAD. The CAD package presented is CadQuery[2] which is a library for Python. This library enables us to create complex CAD objects through code. I hope that others will benefit from and improve on these solutions. If you do have suggestions please search out my blog[3] and leave a post.

The two solutions I present in this article are how I use CadQuery to:

- Create 3D models of components.
- Draw complex PCB board outlines.

In this article I present how I used CadQuery to create a 3D model of a pogo pin and to create a complex board outline that can be imported into KiCad.

CadQuery is one of the few CAD packages that models objects by using code to describe the object. The final object is rendered using a program called CQ-editor. CadQuery uses the Python programming language. The advantages of using code to create the model are:

- We can create a template for the component or outline that can easily be modified to create similar items by changing the values of named variables.
- Writing code fits with how many embedded design folk think.

I don’t present CadQuery as being better than other CAD packages but as a one that fits with the way I think and I suspect will also work for other people who program embedded devices. The software allows me to create 2D and 3D objects using the same methodology that I create firmware with. Break down the design into the smallest components that are practical to work with and re-use as many of these components as I can.

For this article I used KiCad v7.0.9 running on Debian v11 (bullseye) and CadQuery v0.2.0.

Installing CadQuery

There are two components of CadQuery to install. The Python CadQuery library and the CQ-editor. CQ-editor is a visual editor which displays the 3D model that our Python code creates.

To install the CadQuery Python library use this command:

pip3 install --pre cadquery --user

We use --pre flag to install v2 of CadQuery. Leaving out this flag installs v1 of CadQuery which does not have all of the features that we need. The --user flag installs the CadQuery library to your user directory instead of the system installation. Sometimes downloading one Python library causes other Python libraries to also be updated which can end up creating dependency issues with other libraries. This is called ‘dependency hell’. Been there. Installing Python libraries to your user directory makes it easier to undo this mess and not corrupt your system installation of Python. The cleanest method to avoid dependency issues between libraries in Python is to use a virtual environment to run different projects using the venv command. I am too slack to stick to doing this. I just tidy up the Python libraries in my home directory when I have problems.

We install the CQ-editor from its GitHub site[4]. At the time of writing (December 2023), we need to install the unstable build to get all of the necessary features. Download the file for your operating system from:

https://github.com/CadQuery/CQ-editor/releases/tag/nightly

Running the file you downloaded should bring up the CQ-editor. This shows both the Python code and displays the 3D model created by your code when you press the ‘play’ icon at the top of the screen.

We can load and edit code directly in the CQ-editor but I rarely do any editing in the CQ-editor. If you load your Python code into CQ-editor then select File, Automatic reload and preview, you can edit the Python code in whatever your personal favourite editor is. I use vim as vi is what we had when I started out using Unix and we considered ourselves lucky. Each time that the Python code is saved, CQ-editor reloads the code and displays an updated 3D object. Error messages are displayed in the ‘Current traceback’ window in the unlikely event you make a mistake.

Now we have the necessary CadQuery components installed and running, it is time to start using them!

Creating a 3D model of a pogo-pin

Having a 3D model of our final PCB and all of the components on it has a couple of uses. We can use the model to test that our PCB will fit inside a case or to design a case around the PCB. If we have several boards to fit together, then 3D models of each one will help with testing that the boards connect correctly.

Many manufacturers now supply 3D models for their components that can be installed into KiCad. However, the pogo-pin I specced for a recent project does not have a 3D model. So, I decided to draw one up from scratch. The pogo-pin I chose is made by Xinyangze and available from jlcpcb.com7.

With 3D CAD the trick is to see what simple shapes can be assembled to make the final complex shape. If we look at the data sheet for the pogo-pin we can see that it comprises three co-axial pins with a washer that sits on top of the PCB to support the pogo-pin. The washer can be considered as a short co-axial pin as well. Each pin has a different curvature or filleting at one end. The washer can be treated as a co-axial pin with no filleting. So, I created a method called pin which creates a cylinder with a fillet at one end. By calling this method four times, each time with the parameters for a different part of the pogo-pin we can build up the complete model.

The protruding pin at the base of the pogo-pin has the fillet at the bottom while the other pin objects that make up the complete model have a fillet at the top. Apart from the washer shape that sits on top of the PCB, which has no fillet. So, I mirrored the bottom most pin before adding it to the assembly so that the fillet is at the bottom. The washer part that sits on the PCB does not have any fillet. However, CadQuery does not like it when we assign a fillet of value zero. I gave the washer a very small fillet and the part rendered. All we then need to do is to place each of the component pins in the correct location and combine them into a single assembly.

Listing 1 shows the Python code that generates the 3D model of my pogo pin. The dimensions of each of the four pins that comprise the complete assembly are defined. The pin method is called four times, each time with the parameters for the pin that is being added to the assembly. As the pins are added to the assembly, I keep track of where they are added with the ‘stack_height’ variable.

```
import cadquery as cq
from cadquery import exporters

# base of pin that goes through PCB
base_pin_diameter = 0.7
base_pin_fillet = 0.05
base_pin_length = 0.8

# body pin, that goes above washer
body_pin_diameter = 1.5
# body_pin_taper not specified on data sheet, guesstimated
body_pin_fillet = 0.2
body_pin_length = 3.5

# top pin
top_pin_diameter = 0.9
top_pin_fillet = 0.45
top_pin_length = 1.5

# washer shape that sits on PCB
washer_diameter = 2.0
# fractional fillet as a value of 0 causes the code to crash
washer_fillet = 0.001
washer_height = 0.5


def pin(diameter, length, fillet_radius):
    ''' Create a pin '''
    pin = cq.Workplane('XY')\
            .circle(diameter/2)\
```

```
        .extrude(length)\
        .faces("+Z").fillet(fillet_radius)
    return pin


pogo_pin = cq.Assembly()


# create the base pin that fits into the PCB
base_pin = pin(base_pin_diameter, base_pin_length, base_pin_fillet)
# mirror this pin so that the fillet is at the base
base_pin = base_pin.mirror(mirrorPlane="XY", basePointVector=(0, 0, base_pin_length/2))

pogo_pin.add(base_pin)


# keep track of high the assembly is
stack_height=base_pin_length

# add the washer that sits on the PCB
pogo_pin.add(pin(washer_diameter, washer_height, washer_fillet),\
      loc=cq.Location(cq.Vector(0, 0, stack_height)))


stack_height+=washer_height

# create the body pin that fits on top of the 'washer'
pogo_pin.add(pin(body_pin_diameter, body_pin_length, body_pin_fillet),\
      loc=cq.Location(cq.Vector(0, 0, stack_height)))

stack_height+=body_pin_length

# create the top pin
pogo_pin.add(pin(top_pin_diameter, top_pin_length, top_pin_fillet),\
      loc=cq.Location(cq.Vector(0, 0, stack_height)))

# render object
show_object(pogo_pin)

# save step file
pogo_pin.save('pogo_pin.step')
```

Listing 1: Python script to create a 3D model of a pogo pin in CadQuery.

The step file for the 3D model that is created using the script can be imported into KiCad and associated with the component. This allows the component to be displayed in the PCB editor. Please see Figure 1 for a screenshot of the CQ-editor showing the pogo-pin.

Creating a complex board outline

Drawing complex PCB board outlines in KiCad can be tricky. KiCad enables us to draw straight edges and curves in the Edge.Cuts layer to define the edge of the board. This is a clumsy way of

creating a complex board outline. One ‘feature’ of KiCad that has me saying naughty words occurs when I run the design rule check (DRC) in the PCB layout package and am told that the board outline has gaps. I can’t see where these gaps are and I am careful to snap the ends of each straight edge and curve that comprise the board outline to the same grid point so that they connect correctly. I resort to repeatedly shuffling the components of the board edge around at increasing zoom factors until I pass the DRC or my wife turns the computer off, as she would really like to get to sleep without listening to my ongoing ‘conversation’ with KiCad.

My solution is to draw the PCB outline using CadQuery then insert this outline into KiCad. Using code, we can create a complex outline using equations which are impractical to draw in KiCad directly. CadQuery can save this outline as a dxf file. However, this dxf file cannot be imported directly into KiCad.

To get the dxf file created by CadQuery into KiCad I use a the free and open source CAD package FreeCAD7 with a plug-in. The plug-in does some magic on the dxf file and saves it as a KiCad board or directly inserts it into your PCB board file in the outline layer. With practice this only takes a couple of minutes. It would be nice to automate this through a script. I have to weigh the time taken to write and debug this script vs. the fact I don’t use this procedure more than once each couple of weeks.

CadQuery exports svg format files as well as dxf format. One question is why not export an svg file and import this into the Edge.Cuts layer of KiCad? I tried this and the svg file will import into the Edge.Cuts layer. However, the dimensions of the board outline are wrong. For whatever reason, the size information is lost and the svg outline is scaled down in size. This can be fixed with some simple arithmetic. Measure a known dimension in KiCad and apply a scaling factor to the imported svg file. However, getting an accurate dimension in KiCad can be a little tricky as the resolution of your measurement is constrained to the grid resolution that you chose for your PCB design. It is easy to forget this and get an incorrect measurement, resulting in an incorrect scaling factor and an outline that is not what you intended it to be. Going the dxf route is a more robust solution.

First of all, we need to design the board outline in CadQuery. For this example I am copying the board shape of the micro:bit educational board7 with cut outs where the buttons on the micro:bit are and the 4mm holes that allow the board to be used with 4mm banana plugs. This is for a project to create a board that will fit on top of a micro:bit board. The signals and power from the micro:bit are connected to this daughter-board using the pogo-pins that I modelled earlier. The boards connect using screws, nuts and stand-offs through the 4mm holes of the micro:bit and the new PCB.

Listing 2 shows the CadQuery code used to create the 3D model of this board and save this model as a dxf file. A screenshot taken from CQ-editor showing the board outline that this code creates can be seen in Figure 2. The final 3D view of the board rendered in KiCad can be seen in Figure 3a and Figure 3b. Note the lovely pogo-pins created from the code presented earlier in this article.

```
import cadquery as cq

# square buttons on micro:bit
microbit_button_size = 6.2
microbit_x = 51.8
microbit_y = 42.0
microbit_z = 1.6
microbit_corner_radius = 3.0
```

```
# distance from edge of each side of microbit to nearest button edge
# measured from board, supplied drawings to not exactly match
button_x = 2.8
# distance from top edge of microbit to button
button_y = 18.0
# button height above the board
button_z = 4.0

# 4mm holes
hole_dia = 4.0
# y offset from middle of board
hole_y = -(microbit_y/2-7)
# hole names, reading from left to right
hole_names = ['0', '1', '2', '3V','GND']
# x offset from left edge
holes= [(-21.5,hole_y), (-11.5,hole_y), (0,hole_y), (11.5,hole_y), (21.5,hole_y)]
holes_2= [(-21.5,hole_y), (-11.5,hole_y), (0,hole_y), (11.5,hole_y), (21.5,hole_y)]

def microbit_outline(x, y, z):
    # microbit outline and 4mm holes
    microbit_outline = cq.Workplane('XY')\
        .rect(x, y)\
        .extrude(z)\
        .edges('|Z')\
        .fillet(microbit_corner_radius)
        # .pushPoints(holes).circle(hole_dia/2).cutThruAll()
    return microbit_outline


button_separation = microbit_x - 2 * (button_x+microbit_button_size/2)
microbit = (
    cq.Workplane('XY')\
        .box(microbit_x, microbit_y, microbit_z)
        .edges('|Z')
        .fillet(microbit_corner_radius)
        .pushPoints(holes).circle(hole_dia/2).cutThruAll()
        .pushPoints([(button_separation/2, 0), (-button_separation/2, 0)])
        .rect(microbit_button_size, microbit_button_size).cutThruAll()
        )


# Render the solid
show_object(microbit)

cq.exporters.export(microbit, "microbit_2d.dxf", cq.exporters.ExportTypes.DXF)
```

Listing 2: Python script to create a 2D board outline to import into the KiCad Edge.Cuts layer.

I installed FreeCAD version 0.19. Instructions on how to install this software are on the FreeCAD website. We need to install a plug-in to convert the dxf file to the KiCad PCB board format. On the top menu bar of FreeCAD go to Tools, Addon manager, Workbenches and select kicadStepUpMod.

Once this is installed, restart FreeCAD. Don’t think you can get away with not restarting FreeCAD. You really do. Guess how I learned this?

After restarting FreeCAD, select the KiCadStepUp plug-in from the drop-down list in the middle of the top menu bar. This list defaults to ‘Start’ on startup. A new menu bar with many incomprehensible symbols appears. We will use two of these, as shown in Figure 4.

Import the dxf file created by your CadQuery script using File, Import.

You need to select all of the parts of the dxf file that you want to use as your board outline. These parts all need to be on the same plane. Selecting the parts might be tedious if there are a lot of them. I’m not familiar enough with FreeCAD to know if there is a ‘select all’ method for a single plane. I press down on the control key and select all of the segments that I need for the board outline. Once I figure out a slicker way of doing this I’ll write it up as a blog post on my website.

Now click on the ‘ksu 2D object (or DXF) to Sketch’ button on the toolbar that was created when you installed the plug-in. See the annotated screenshot of the menu bar in Figure 4.

Scroll to the bottom of the Model window on the left in the Combo View pane. Select the sketch that has just been created. Figure 5 shows an example screenshot of this.

Click on the ‘ksu Push Sketch to PCB Edge’ button in the menu bar. This option is indicated in Figure 4.

You will get the option to push the outline directly to your KiCad PCB file, into the outline layer. I’ve not tried this. I’ve always selected the option to create a separate KiCad format PCB file.

Import the newly created file into your KiCad PCB using the menu option File, Append Board in the KiCad PCB editor.

Run DRC to check this is all good. Access the 3D viewer using the menu options View, 3D Viewer. Enjoy the glory of a fully joined up outline!

Discussion

The trick with any problem is to break the problem down into small pieces, each of which can be solved. This is how we build our circuits and write our firmware. Using CadQuery enabled me to solve a couple of stages towards finishing a PCB design project using the same mindset. I welcome any suggestions on how to improve solving the two problems I’ve presented here.

References

[1] https://www.kicad.org

[2] https://github.com/CadQuery/cadquery

[3] mattoppenheim.com

[4] https://github.com/CadQuery/CQ-editor

[5] https://jlcpcb.com/partdetail/Xinyangze-YZ08015063P02/C5157301

[6] https://www.freecad.org

[7] https://microbit.org

Figure 1: CQ-editor screenshot showing the pogo-pin

Figure 2: PCB board outline created with CadQuery

Figure 3a: 3D model of a KiCad PCB top

Figure 3b: 3D model of a KiCad PCB base

Figure 4: FreeCAD screenshot showing KiCadStepUp menu options

Figure 5: FreeCAD screenshot showing which file to export to KiCad

Author Profile

Matthew Oppenheim works in marine geophysical survey and loiters at InfoLab21, Lancaster University when not working on geophysical research vessels. At InfoLab21 Matt has been working on assistive technology projects and how to deploy them into the Real World. Visit www.mattoppenheim.com for more details of his projects.