FreeCAD Articles
UV Surfaces

Carlo Dormeletti

Version: 1.01

FreeCAD reference version: 0.20.1

Contents

1 UV Surfaces

1.1 Introduction

This monography will try to explain, UV surfaces and in general the projection of a 2D object on a 3D Surface.

This is considered advanced and as it is not usually available on **FreeCAD** without using an additional WB, it is not very explained in details in the Wiki.

I could not consider myself an expert, but I will try to explain, what I have found experimenting with this sort of things.

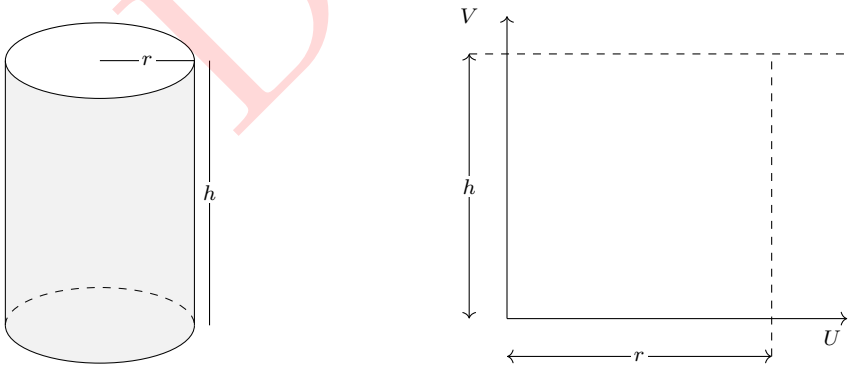
The explanation will involve some mathematics and some drawings.

1.2 Concepts

Usually in 3D we speak about coordinates using conventions, one of the most used is the fact that we use the XYZ to identify 3D position in the space, that is an extension of the "Cartesian space" in 2D that is traced using the XY plane, that is widespread known.

But as here we have to deal with a different concept, we should use a different notation, so the use UV instead of XY.

Let's start with something



(a) A Cylinder (b) UV Surface.

Figure 1.1: UV surfaces

In figure figure 1.1a we see a representation of a cylinder, as is modeled in **FreeCAD**.

It is modeled as by three faces:

1. Upper circular face.
2. Lower circular face.

3. “side” surface.

This side surface could be “developed” in a similar way you wrap a sheet of paper around a cylindrical shape.

You could note easily that the face is bend and that there is “seam” were the sheet start and end side meet together.

1.2.1 common code

Let’s introduce some code used at start of most experiments in this article:

```
import FreeCAD
import FreeCADGui
from FreeCAD import Placement, Rotation, Vector
import Part

V2d = FreeCAD.Base.Vector2d

### CODE START HERE ###
```

When in this article we say start new code, add the new code to the code above.

1.3 Consideration about Periodic Surfaces

Add code below to the code presented in section 1.2.1

```
c_rad = 10

cyl = Part.Cylinder()
cyl.Radius = c_rad

print(f'Is UPeriodic = {cyl.isUPeriodic()}')

print(f'Is VPeriodic = {cyl.isVPeriodic()}')

Uper = cyl.UPeriod()

print(f"Uper: {Uper}")
```

This will result with something printed in the **Report View**.

```
Is UPeriodic = True
Is VPeriodic = False
Uper: 6.283185307179586
```

Nothing complicated, we have created a cylinder and analyzed the “primitive”, using some methods.

We could note something:

- the cylinder is **UPeriodic** with a Period of 6.283185.
- the cylinder is not **VPeriodic**

Not surprisingly **UPeriod** is equal to 2π as the full circumference is parametrized in the interval (0 to 2π)

To do some tests try to change as example the value of `c_rad` to 20 and rerun the code. You will see that the **UPeriod** is not changing, why?

Because is periodic so you don't have to bother about the real dimension of the surface.

We are dealing with and "abstract" cylinder here, that has no "height".

Let's see:

```
cyl_shp = cyl.toShape()

Part.show(cyl_shp, 'cylinder_shape')
```

You will see something "strange", as the cylinder is visualized, but if you see it from top, it will disappear.

This is due to the fact that we have used a "geometric primitive", in fact if you issue.

```
print(f'cyl_shp = {cyl.TypeId}')
```

You will see that our `cyl` is a:

```
Part::GeomCylinder
```

This show that what we have obtained is not a solid.

1.4 Face and Surface

There is a relation between the "geometric primitive" and the face.

Some considerations

First of all each **Face** has a **Surface** property that will return the unlimited surface from which the face is a portion. if you have any doubt read the article about **Topology and Geometry** as the concepts are the same explained for the **Curve** property.

Some caveats, if the surface is **UPeriodic** the U coordinate is Periodic.

Start new code and add the code below; You don't need to copy values after # as for brevity they are what you will see in the **Report View**.

```
t_cyl = Part.Cylinder()
print(f'Cylinder bounds:{t_cyl.bounds()}')
print(t_cyl.value(0.5, 1.0))
# Vector (0.8775825618903728, 0.479425538604203, 1.0)
```

```
print(t_cyl.value(0.5 + Uper, 1.0))
# Vector (0.8775825618903729, 0.4794255386042028, 1.0)
print(t_cyl.value(0.5 + 2 * Uper, 1.0))
# Vector (0.877582561890373, 0.47942553860420256, 1.0)
```

Some things to remark:

1. **Vector**'s X value is the same even if the value is multiplied by **Uper**.
2. using `bounds()` we have obtained a tuple that contains the limits of priomitive UV parameters.

With `bounds()` the tuple is composed as follows:

`(UMin, Umax, VMin, VMax)`

first two parameters are the interval for U, and last two parameters are interval for V, it prints:

```
Cylinder bounds: (0.0, 6.283185307179586, -2e+100, 2e+100)
```

V parameter show peculiar numbers, in OCCT cylinder geometric primitive are created with V values in the interval **-infinite** and **+infinite**.

Radius is taken in account, as it is needed to return correct position in 3d space, as example if you use two different radiuses, as follows:

Z value is returned as `1.0` instead of **2e+100**, as we have used `1.0` as V parameter in VNM.value(U, V), this will not harm as we are dealiong with a primitive.

1.4.1 Experiment 1

Start new code as below:

```
t1_cyl = Part.Cylinder()
t1_cyl.radius = 10
Uper1 = t1_cyl.Uperiod()
print(t1_cyl.value(0.5, 1.0))
print(t1_cyl.value(0.5 + Uper1, 1.0))
print(t1_cyl.value(0.5 + 2 * Uper1, 1.0))

t2_cyl = Part.Cylinder()
t2_cyl.radius = 20
Uper2 = t2_cyl.Uperiod()
print(t2_cyl.value(0.5, 1.0))
print(t2_cyl.value(0.5 + Uper2, 1.0))
print(t2_cyl.value(0.5 + 2 * Uper2, 1.0))
```

You will obtain some data, data are derived from the parametric equation of the cylinder that for a cylinder centered on origin and that the axis of the cylinder is parallel to the global **Z** axis is:

$$\begin{bmatrix} x & = & r * cos(u) \\ y & = & r * sin(u) \\ z & = & v \end{bmatrix}$$

Using `value(u, v)` we have 3d coordinates relative to the UV position passed, same results for Z as we are using here again the geometric primitive.

If we use a real face this is different;

Start a new code:

```
c_rad = 10
c_hei = 10
cyl1_ts = Part.makeCylinder(c_rad, c_hei)
cyl1_fc = cyl1_ts.Faces[0]
cyl1_sf = cyl1_fc.Surface

print(f'Face TypeId: {cyl1_fc.TypeId}')
print(f'Face ParameterRange: {cyl1_fc.ParameterRange}')

print(f'Surface TypeId: {cyl1_sf.TypeId}')
print(f'Surface bound: {cyl1_sf.bounds()}')
```

Now we have generated a proper TopoShape, and we have specified cylinder's height, you will easily see the result in **Report View**.

```
Face TypeId: Part::TopoShape
Face ParameterRange: (0.0, 6.283185307179586, 0.0, 10.0)
Surface TypeId: Part::GeomCylinder
Surface bounds: (0.0, 6.283185307179586, -2e+100, 2e+100)
```

The code above could seem strange until we know that for every cylinder `Faces[0]` is the cylindrical face.

With `cyl1_fc.ParameterRange` we will obtain something similar to what we have obtained using `bounds()` when we have used the geometric primitive;

Note that in this case we have used `1.0` as V parameter, like with the geometric primitive, but `VMin` is 0 and `VMax` is equal to the height stated in `Part.makeCylinder`, so V interval is between `0 ... 10`.

Data about what parametrizations is used could be found in the OCCT documentation, as example in:

`https://dev.opencascade.org/doc/occt-7.6.0/refman/html/index.html`

Under:

Module ModelingData » Toolkit TKG3d » Package Geom » Geom_CylindricalSurface

You will find in the graph at the right the links to the geometric primitives, if you as example click on **Geom_CylindricalSurface**, and in the page that will be opened click on **More...** in blue, you will find under **Detailed Description**

...

This class defines the infinite cylindrical surface.
Every cylindrical surface is set by the following equation:
`S(U,V) = Location + R*cos(U)*XAxis + R*sin(U)*YAxis + V*ZAxis`

That is the equation presented above, with some thing added to take care of cylinder Axis's orientation.

1.5 Mapping something on a Face

Now that hopefully some theory behind the concept of UV space has been explained, let's see some real life applications.

One of the things to be aware is that 2D geometries must be "mapped" on a 3D surface to generate a TopoShape.

Let's start creating a simple 2D line.

To start a remark, we will use the concept of 2D Vector, if you see common code you will easily find this line that define a shortcut for the function:

```
V2d = FreeCAD.Base.Vector2d
```

In **FreeCAD** Part module we have almost all the primitive present in OCCT under `Part.Geom2D`.

Start a new code with code below.

```
c_rad = 10
c_hei = 30

cyl1_ts = Part.makeCylinder(c_rad, c_hei)
cyl1_fc = cyl1_ts.Faces[0]
cyl1_sf = cyl1_fc.Surface
cyl1_pr = cyl1_fc.ParameterRange

print(f'Face ParameterRange: {cyl1_pr}')

Uper = cyl1_pr[1]

turns = 3
line2d_1 = Part.Geom2d.Line2dSegment(V2d(0, 0), V2d(Uper * turns,
    ↪  c_hei))
```

Code above will remain the same across all the experiments.

Now with the code below, we visualize some things on the 3d view.

```
Part.show(cyl1_ts, "cylinder")

Part.show(line2d_1.toShape(), "line2d")
```

It simply show the cylinder and a line, this is a 2d line "mapped" in 3D using `toShape()` not a new thing, I hope.

Let's modify the code above, as follows

```
# Part.show(cyl1_ts, "cylinder")

cyl1_gp = Part.Cylinder()
cyl1_gp.Radius = c_rad

linep = Part.show(line2d_1.toShape(cyl1_gp), "proj2d")
```

At a first glance, there are not so much difference in writing:

1. `Part.show()` in this case is making a projection.
2. we have specified a cylinder **Radius**, but "apparently" we have not specified an height.
3. Something is appeared on the cylinder surface, (with some artifacts).

point 1 Even if the line is very similar to the preceding use, in this case we have specified as argument of `toShape` a geometric primitive, so we are doing a mapping.

point 2 As we have used a geometric primitive, but projected a line that as some data on it:

```
line2d_1 = Part.Geom2d.Line2dSegment(V2d(0, 0), V2d(Uper * turns,
                    ↪  c_hei))
```

and precisely:

```
V2d(Uper * turns, c_hei)
```

That set the endpoint of the line.

It has a precise Y coordinate, that is equal to `cyl_ts` height. and has even a X coordinate that is a multiple of `Uper` it is projected on the geometric primitive that has an infinite V interval.

point 3 We have visualized the projected line, that is coplanar with the surface, if we hide the cylinder we will easily see that is a sort of coil.

As exercise, you could play with `cyl1_gp.Radius` and `turns`, just to see what happens.

1.5.1 Conclusion

This is not exaustive article about projections, as it is simply projecting an edge on a surface to have in returns an edge that could be easily adding to the code this line:

```
print(f'linep shapetype: {linep.Shape.ShapeType}')
```

This edge could be used as example as a path for a sweep, maybe to make a thread or a coil.

1.6 Afterword

I need some interaction with users, so your suggestions are welcomed, you could contact me on **FreeCAD** forum using my nickname **onekk**, or using this forum post:

https://forum.freecadweb.org/viewtopic.php?f=36&t=74950

And I need some encouragement, as the time spent in trying to explain things, and making a decent graphic typeset is not negligible.

You could "help development" using the ways explained in this site:

https://github.com/onekk/freecad-doc

Thanks to all for every help and feedback.

Carlo D. onekk