

Creating great 3D figures with minimal effort

Jean Feydy
HeKA team, Inria Paris
Inserm, Université Paris-Cité

6th of November, 2024
Shape Seminar
MAP5, Paris

Who do you want to trust?

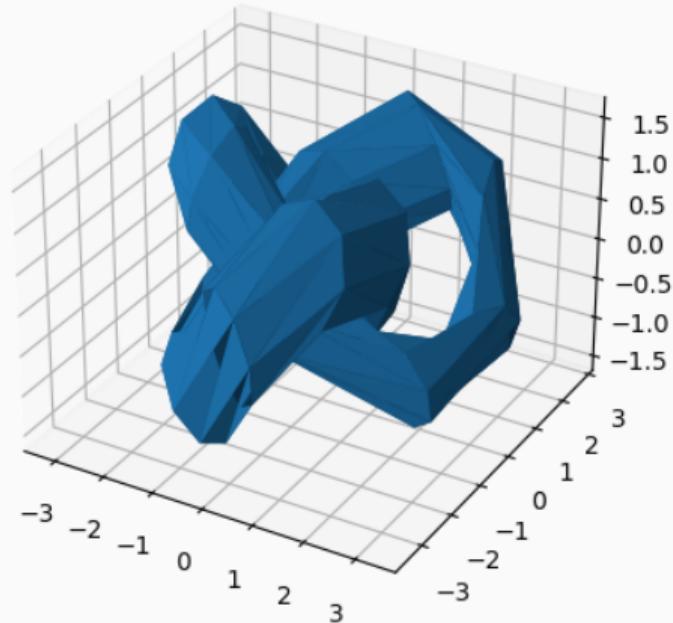


A triangle sandwich,
with soulless packaging.

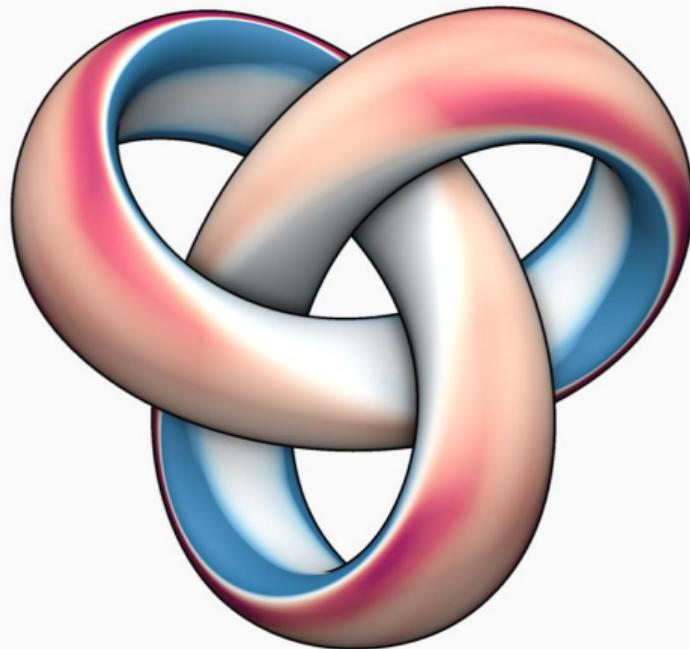


A fresh sandouiche,
are you salivating yet?

Who do you want to trust?



A trefoil knot,
with default Matplotlib settings.



The same data,
with 10 lines of PyVista code.

But Jean... I'm just a mathematician!

Creating great 3D figures takes some time but:

- **Can now be done in minutes** with PyVista, a transparent Matplotlib replacement.
- **Demonstrates** a familiarity with the **state-of-the-art** in geometry processing.
- **Sets you apart** from the **low-effort papers** that are flooding reviewers' mail boxes.

Keenan Crane CARNegie MELLon UNIVERSITY

[Home](#) [News](#) [Teaching](#) [Publications](#) [Code](#) [YouTube](#) [Twitter](#) [CV](#) [FAQ](#) [Misc](#)

Teaching

Computer Graphics (15-462/662) Discrete Differential Geometry (15-458/858)
Videos Code Webpage Videos Code [C++/JS] Webpage

Monte Carlo Methods (15-327/627/860, 21-387)
Notes Webpage

Publications

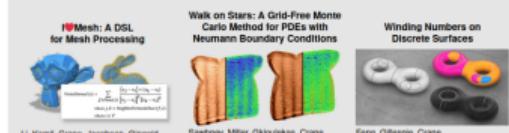
Differential Walk on Spheres Harmonic Functions Ray Tracing Repulsive Shells


Miller, Seemann, Crane, Gkoloukias ACM Trans. on Graph. (2024) Abstract PDF BibTeX Best Paper Award
Gillespie, Yang, Botsch, Crane ACM Trans. on Graph. (2024) Abstract PDF Project BibTeX
Sassouni, Schumacher, Rumpf, Crane ACM Trans. on Graph. (2024) Abstract PDF Project BibTeX Best Paper Award

Walkin' Robin: Walk on Stars with Robin Boundary Conditions A Heat Method for Generalized Signed Distance Minkowski Penalties: Robust Differentiable Constraint Enforcement for Vector Graphics


Miller, Sawhney, Crane, Gkoloukias ACM Trans. on Graph. (2024) Abstract PDF BibTeX Best Paper Award
Feng, Crane ACM Trans. on Graph. (2024) Abstract PDF Project BibTeX
Minarick, Estep, Ni, Crane SIGGRAPH 2024 Abstract PDF Project BibTeX

I²Mesh: A DSL for Mesh Processing Walk on Stars: A Grid-Free Monte Carlo Method for PDEs with Neumann Boundary Conditions Winding Numbers on Discrete Surfaces


Li, Kamit, Crane, Jacobson, Gingold Abstract PDF BibTeX
Sawhney, Miller, Gkoloukias, Crane Abstract PDF BibTeX
Feng, Gillespie, Crane Abstract PDF BibTeX

Thesis

Conformal Geometry Processing

Keenan Crane Caltech PhD Thesis (2013) Abstract PDF BibTeX

News

June 2024—3 SIGGRAPH Best Paper Awards Our group won two best paper awards and one honorable mention at SIGGRAPH 2024. The three best papers and 12 such awards (respectively) are selected from a pool of about 840 submissions.

May 2024—Hertz Fellowship Geometry Collective member Zia Mansouri is one of only 18 PhD students in the world to receive a Hertz Fellowship! Congrats, Zoll!

March 2024—5 Papers at SIGGRAPH We've had five papers conditionally accepted to SIGGRAPH 2024. Stay tuned for more information.

February 2024—Cell Representations Seminar I will give an invited talk at the *Allen Institute for Cell Science* on Interpretative Quantitative Cell Representations.

November 2023—The Aperiodical Had fun talking to *The Aperiodical* about the origins of my YouTube channel, and various other topics of mathematics / computer science.

October 2023—SIGGRAPH Technical Papers Committee I will serve on the technical papers committee for SIGGRAPH 2024, which takes place in Denver, Colorado.

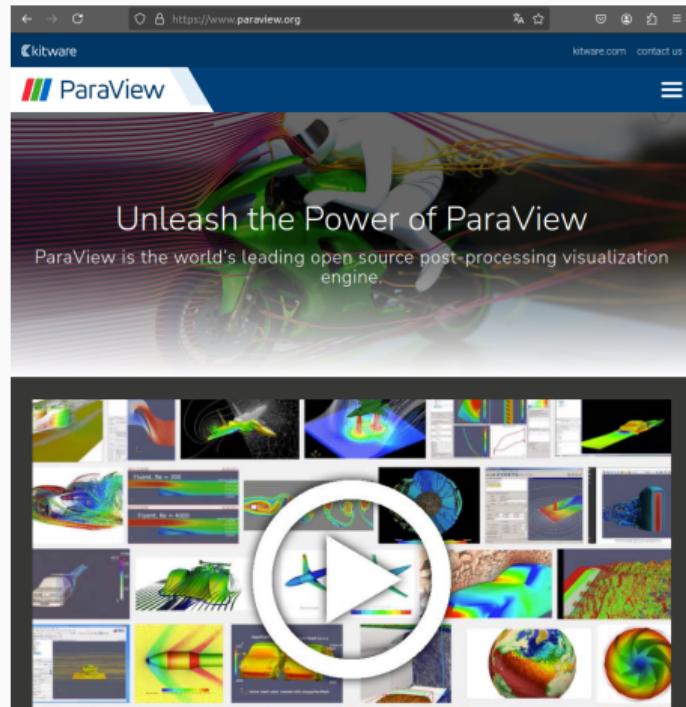
July 2023—ICD Keynote I will give a keynote at the *International Geometry Summit* in Genova, Italy, on "Walk on X" Monte-Carlo methods: the quest for a unified mathematical foundations and practical algorithms for nonconvex neural network minimization.

Keenan Crane **signals competence** with every single figure.

The Visualization ToolKit (VTK) – going strong since 1993.



PyVista – the pythonic API.



Paraview – the interactive GUI.

Today's talk – demystifying the graphics toolbox

1. Tips and tricks with **PyVista**.

Subdivision? Culling? SSAO? PBR?

2. Links to **advanced resources**.

The maths behind the simple functions.

3. Interactive **Paraview** demo.

With real data :-)

Opening a PyVista window – import pyvista as pv

```
pl = pv.Plotter(window_size=[800, 800])  
  
S = pv.PolyData(  
    [[x1, y1, z1], [x2, y2, z2], ...],  
    faces=[3, a, b, c, 3, d, e, f, ...],  
)  
pl.add_mesh(S)  
  
pl.camera_position = "xy"  
pl.camera.zoom(1.4)  
pl.enable_antialiasing("ssaa")  
pl.enable_parallel_projection()  
  
pl.show()  
pl.screenshot("knot.jpg")
```



A “**flat**” 3D rendering
of our trefoil knot.

Anti-aliasing matters, especially for small screenshots – 200x200 below



`pl.disable_anti_aliasing()`

One sample per pixel.



`pl.enable_anti_aliasing()`

Multiple samples per pixel.

Parallel projection – a bit better for scientific visualization?



Default 3D perspective.



pl.**enable_parallel_projection()**

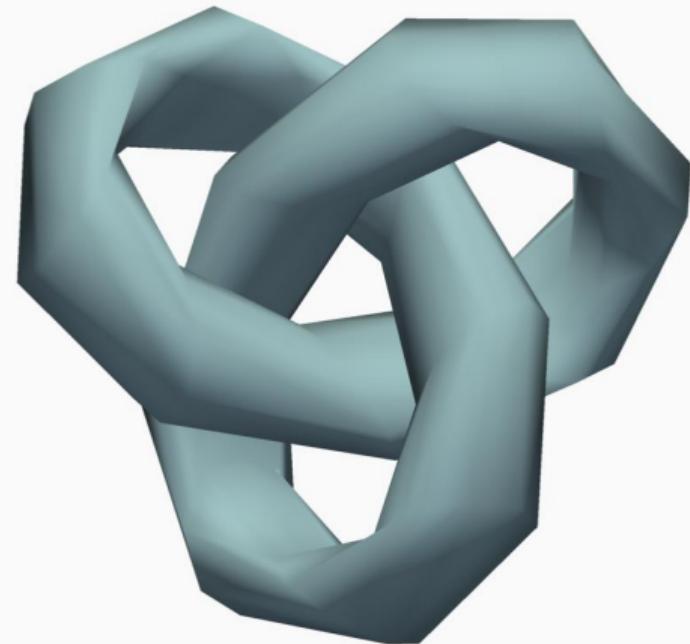
Smoothness

From flat to Gouraud shading with point normals



`pl.add_mesh(S)`

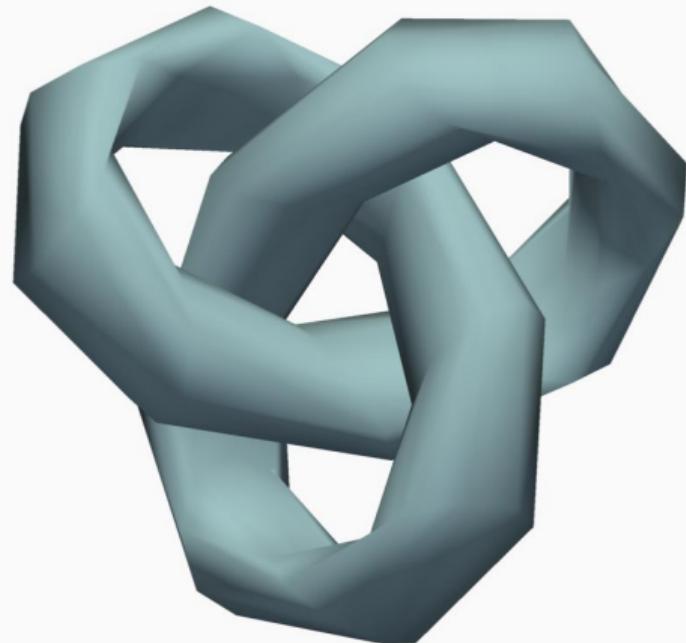
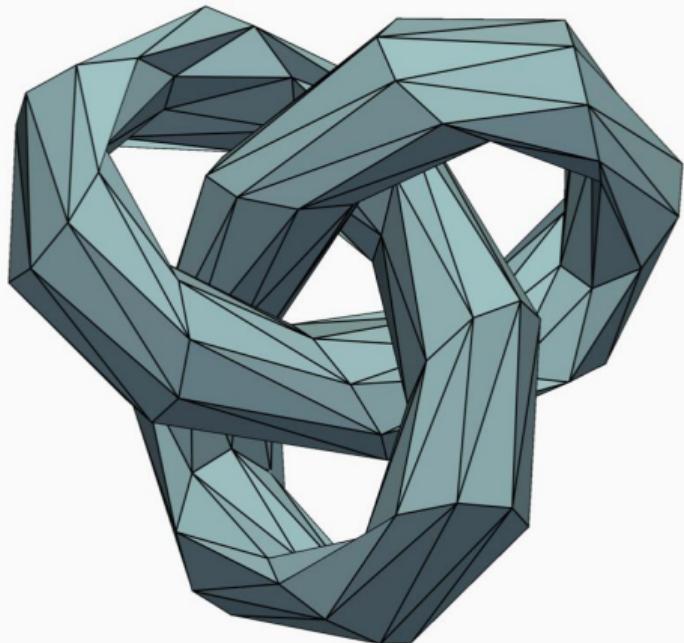
Raw data with flat shading.



`pl.add_mesh(S, smooth_shading=True)`

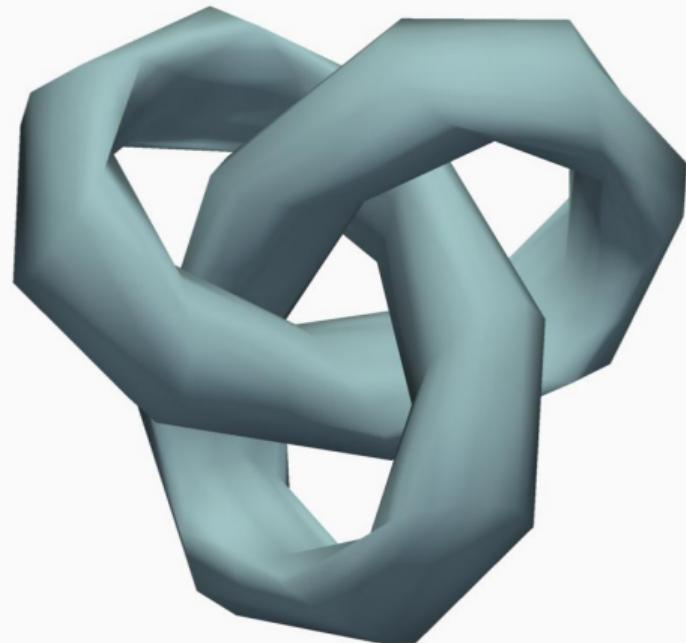
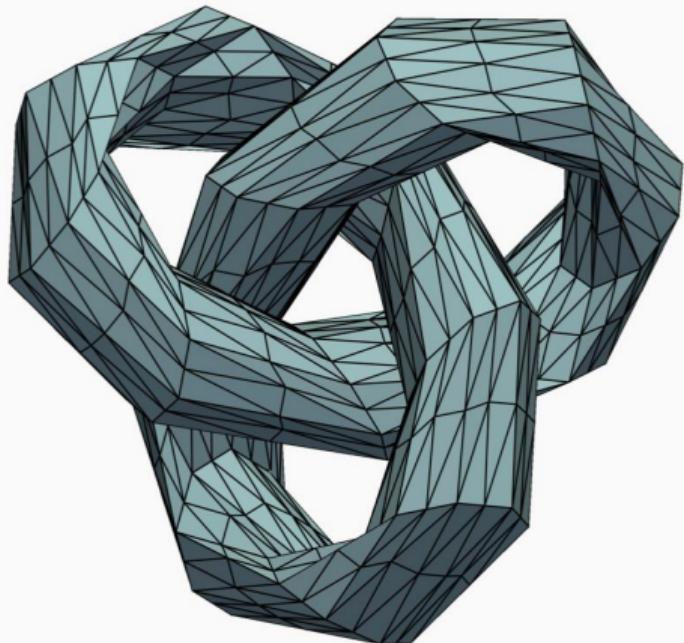
Interpolate point normals.

Getting smoother geometry: with linear subdivision?



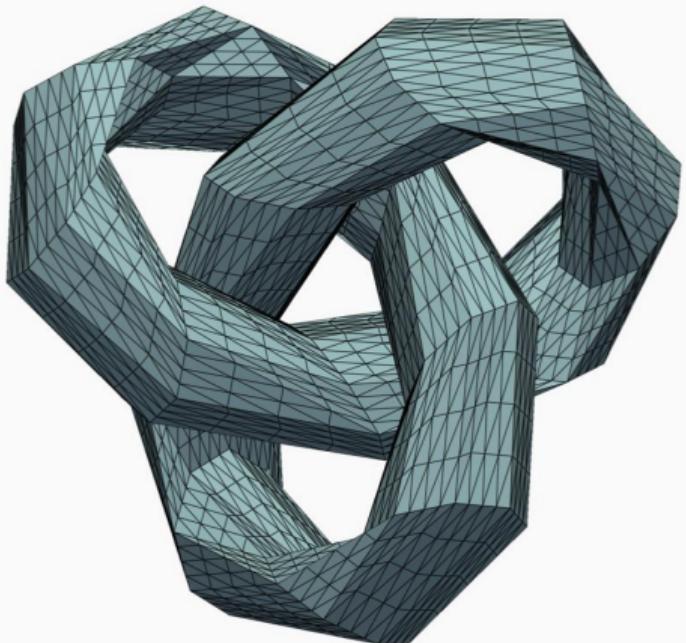
```
S = S.subdivide(subfilter="linear", nsub=0)
```

Getting smoother geometry: with linear subdivision?



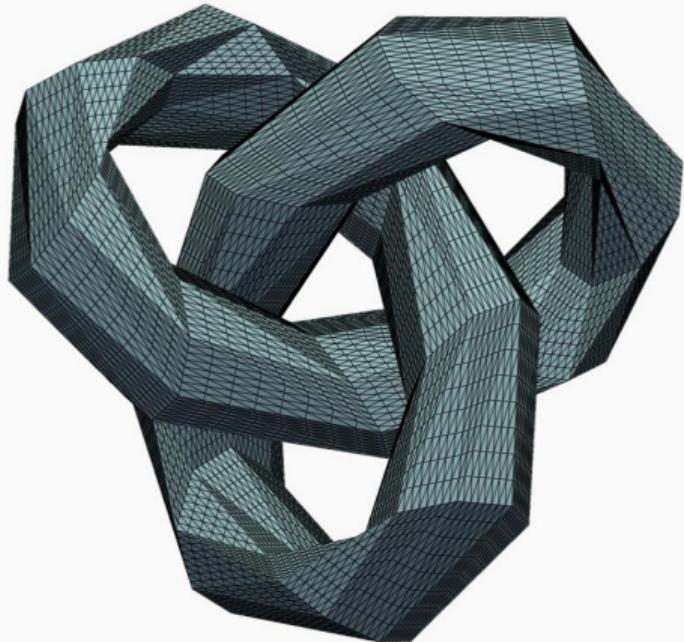
```
S = S.subdivide(subfilter="linear", nsub=1)
```

Getting smoother geometry: with linear subdivision?



```
S = S.subdivide(subfilter="linear", nsub=2)
```

Getting smoother geometry: with linear subdivision?



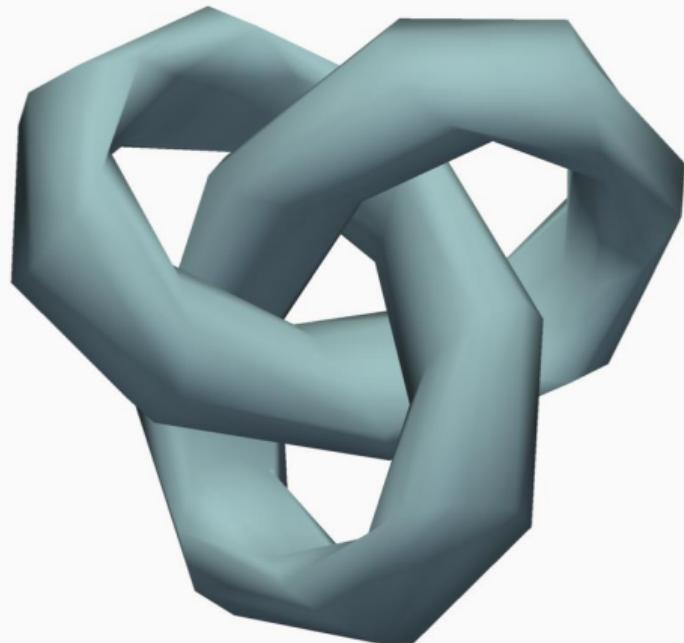
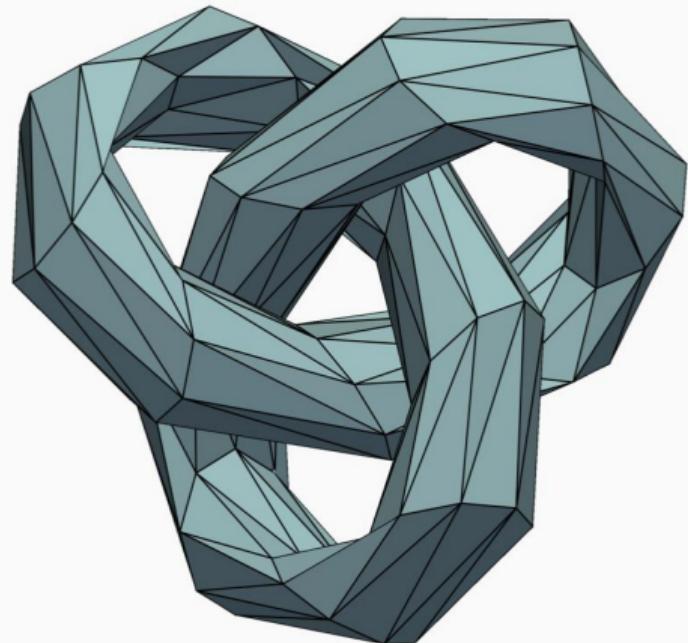
```
S = S.subdivide(subfilter="linear", nsub=3)
```

Getting smoother geometry: with linear subdivision?



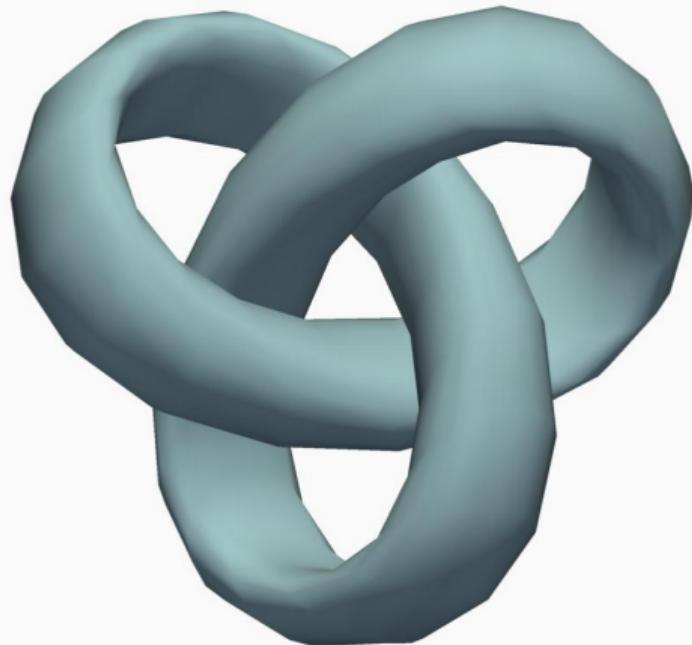
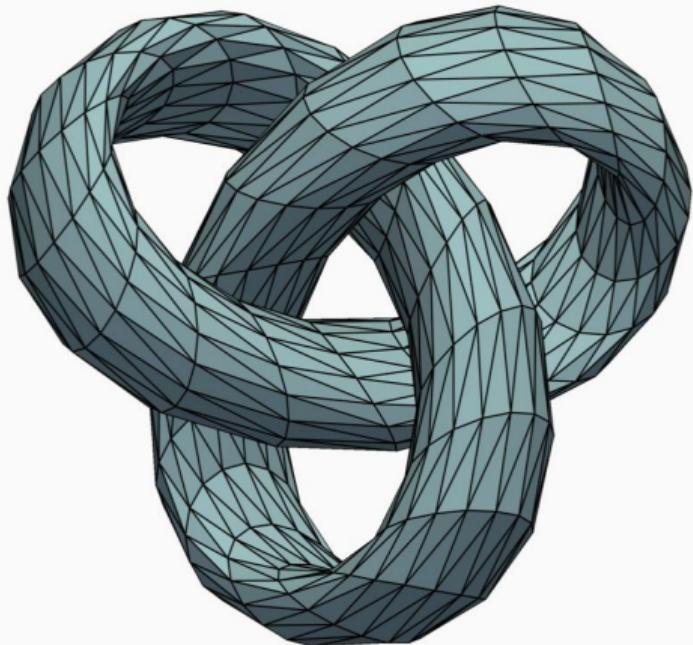
```
S = S.subdivide(subfilter="linear", nsub=4)
```

Getting smoother geometry: with loop subdivision!



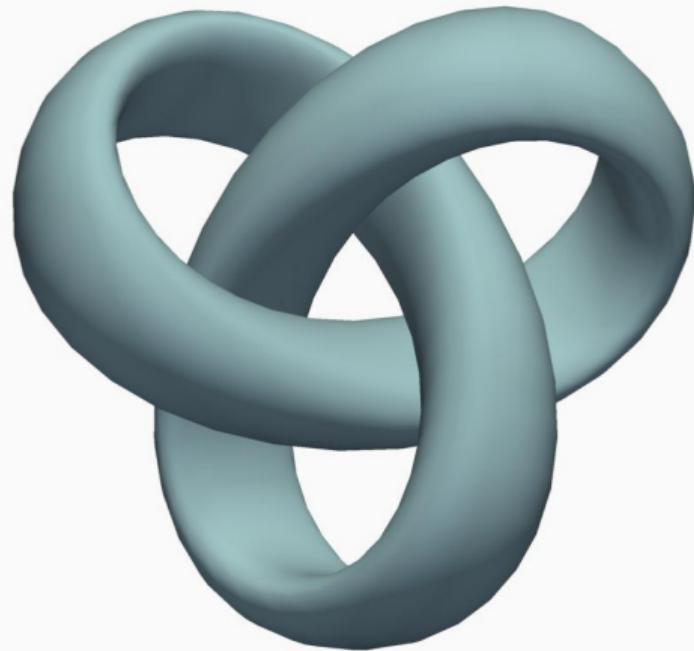
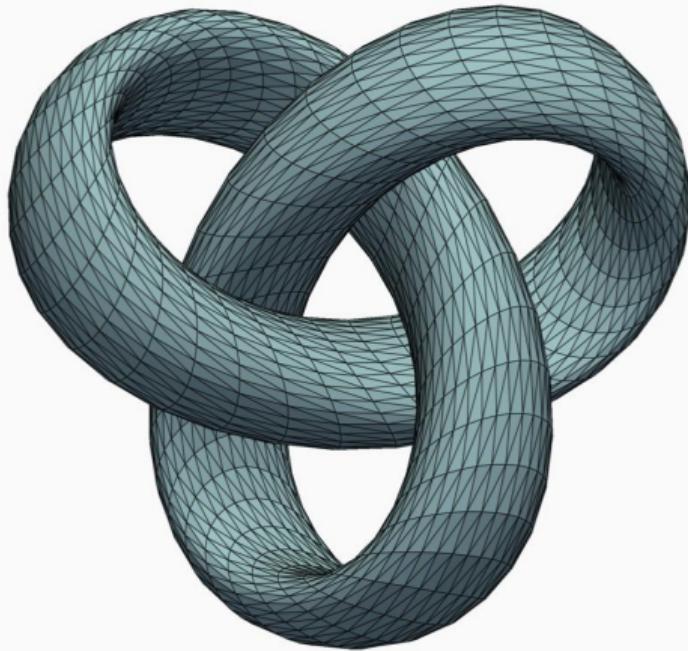
```
S = S.subdivide(subfilter="loop", nsub=0)
```

Getting smoother geometry: with loop subdivision!



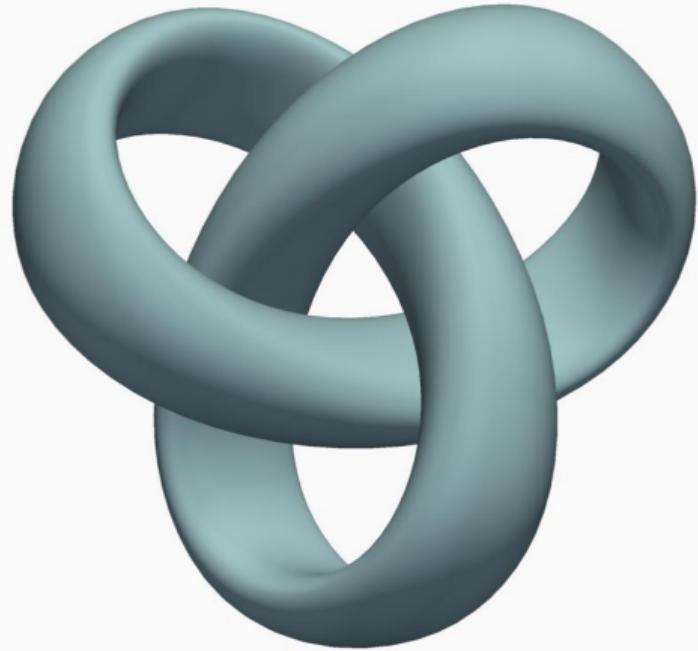
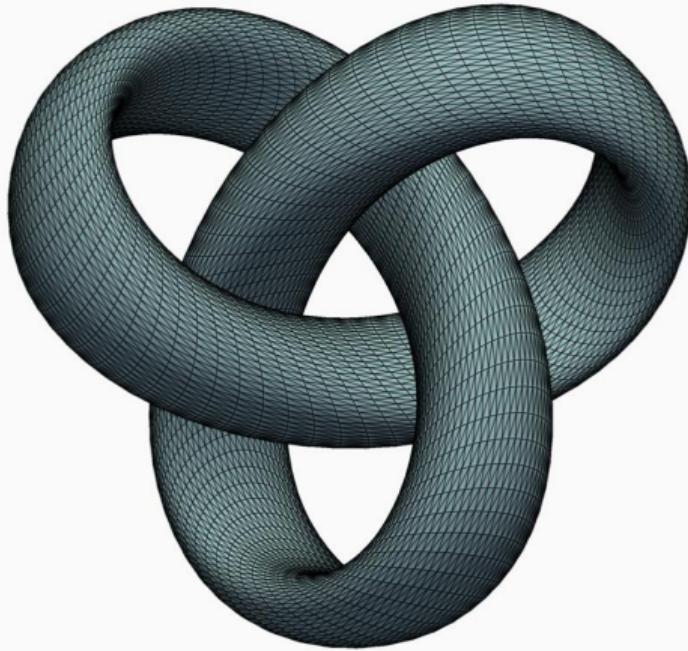
```
S = S.subdivide(subfilter="loop", nsub=1)
```

Getting smoother geometry: with loop subdivision!



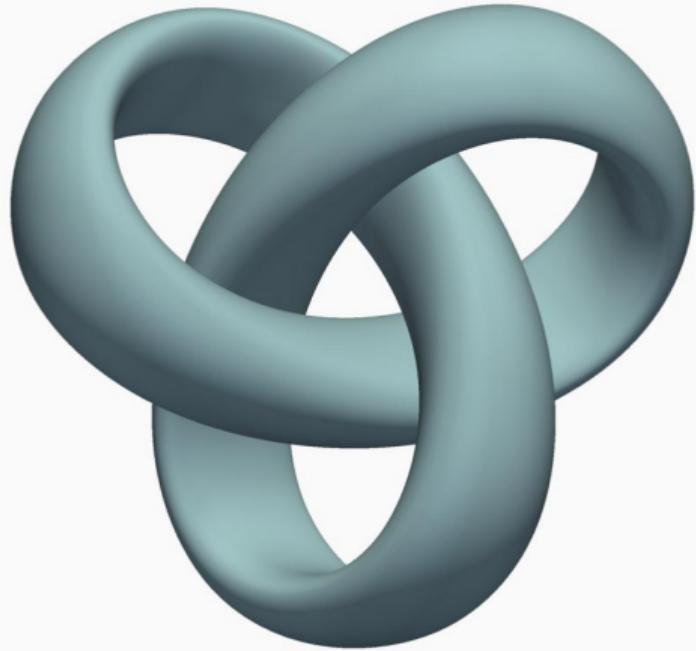
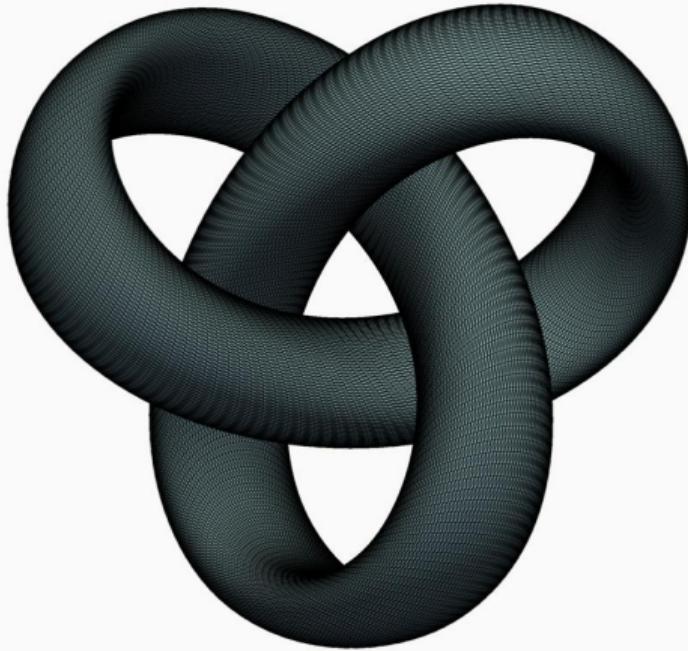
```
S = S.subdivide(subfilter="loop", nsub=2)
```

Getting smoother geometry: with loop subdivision!



```
S = S.subdivide(subfilter="loop", nsub=3)
```

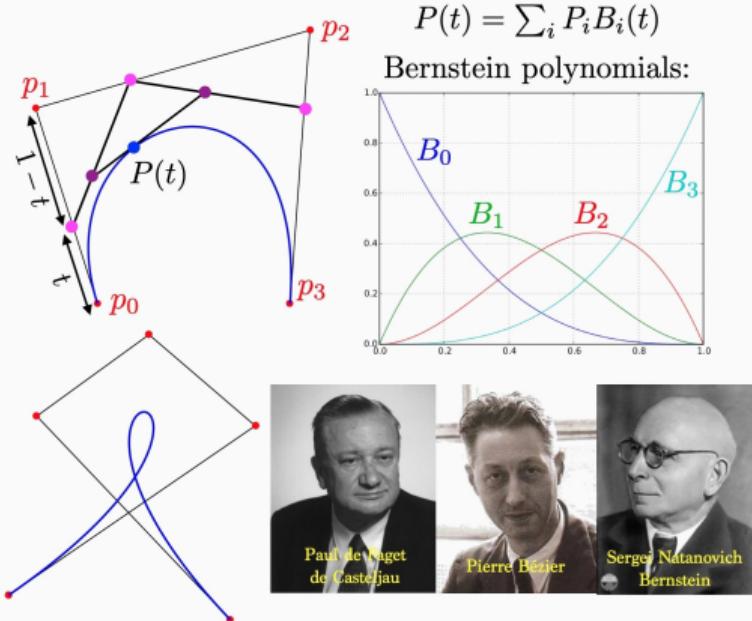
Getting smoother geometry: with loop subdivision!



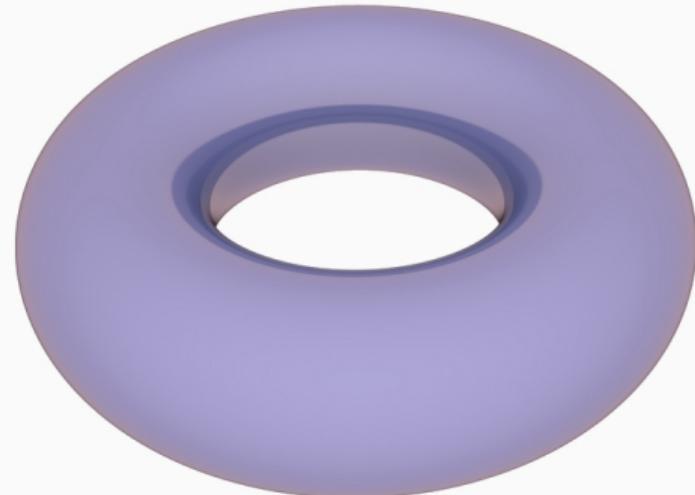
```
S = S.subdivide(subfilter="loop", nsub=4)
```

This is not “cheating” – why stick to linear interpolation for smooth data?

Interesting mathematics behind subdivision surfaces

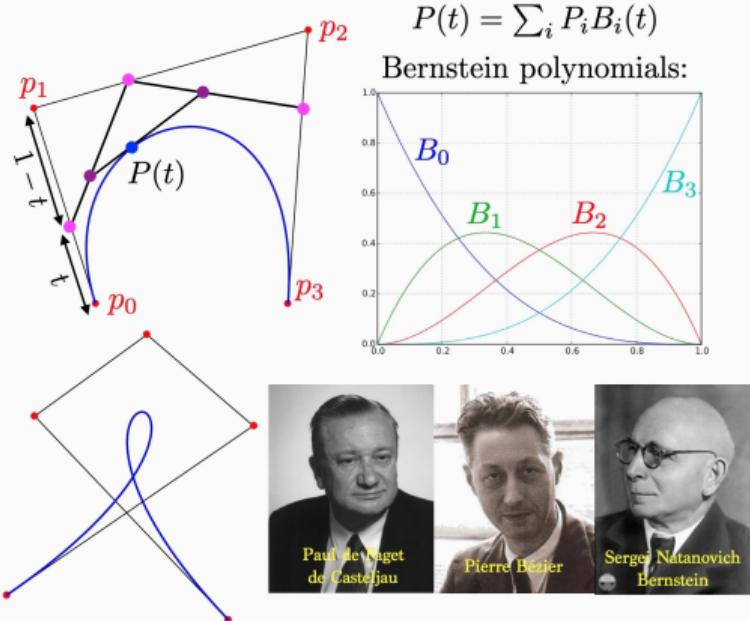


De Casteljau's algorithm,
courtesy of Gabriel Peyré.



The **flat torus** embedded in \mathbb{R}^3
by the Hevea project.

Interesting mathematics behind subdivision surfaces

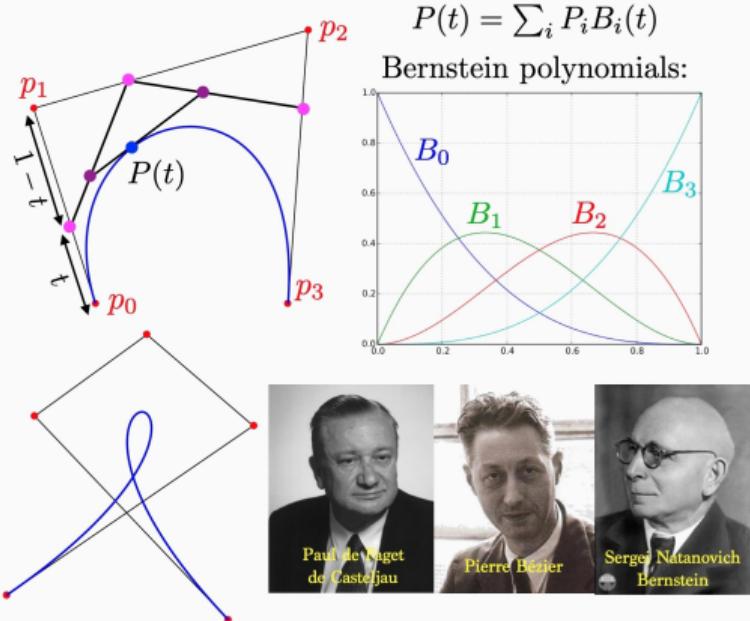


De Casteljau's algorithm,
courtesy of Gabriel Peyré.



The **flat torus** embedded in \mathbb{R}^3
by the Hevea project.

Interesting mathematics behind subdivision surfaces

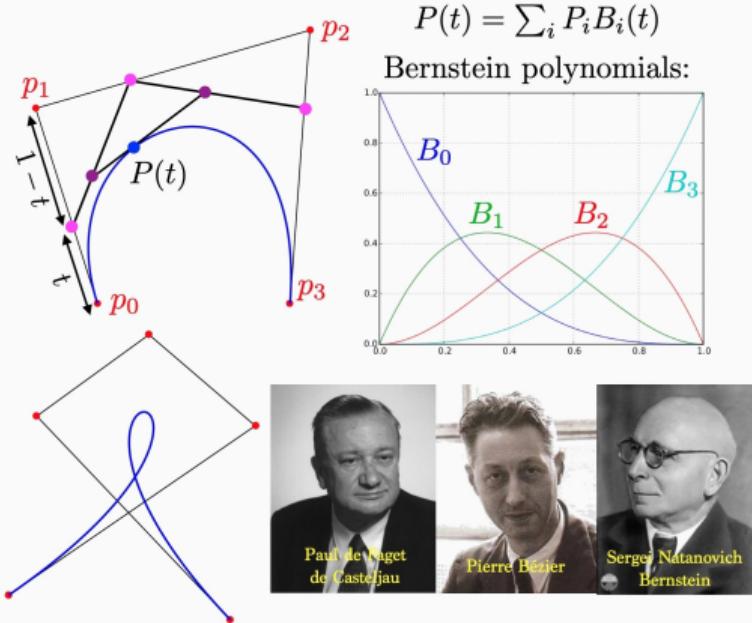


De Casteljau's algorithm,
courtesy of Gabriel Peyré.



The **flat torus** embedded in \mathbb{R}^3
by the Hevea project.

Interesting mathematics behind subdivision surfaces

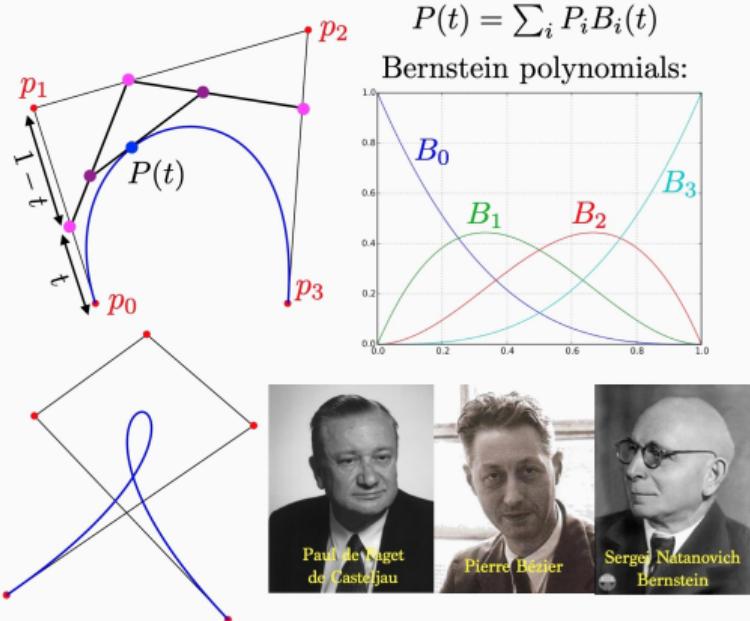


De Casteljau's algorithm,
courtesy of Gabriel Peyré.



The **flat torus** embedded in \mathbb{R}^3
by the Hevea project.

Interesting mathematics behind subdivision surfaces



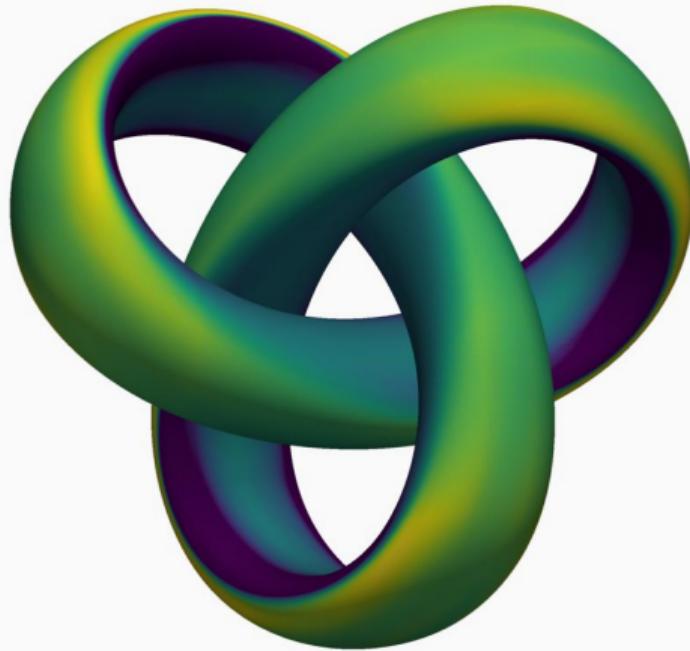
De Casteljau's algorithm,
courtesy of Gabriel Peyré.



The **flat torus** embedded in \mathbb{R}^3
by the Hevea project.

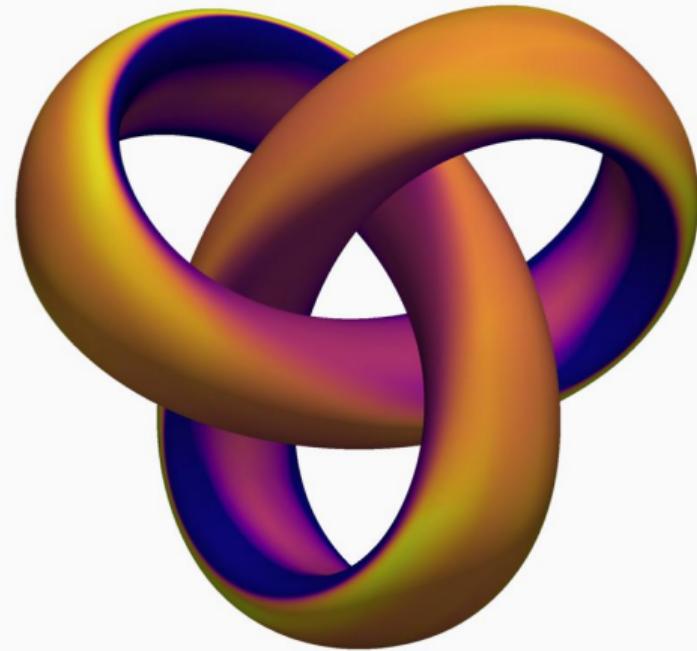
Colors

Adding some colors



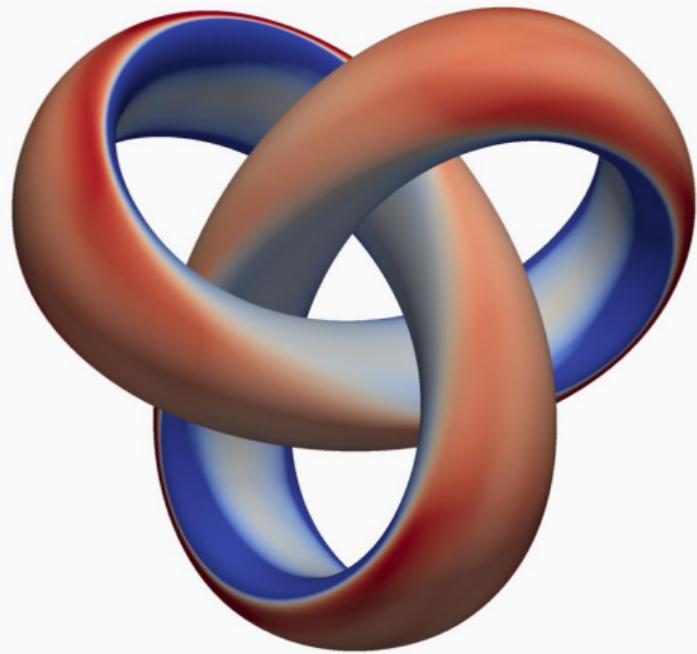
cmap="viridis"

```
pl.add_mesh(S, scalars=S.curvature("gaussian"), cmap=cmap, clim=(-1, 1))
```



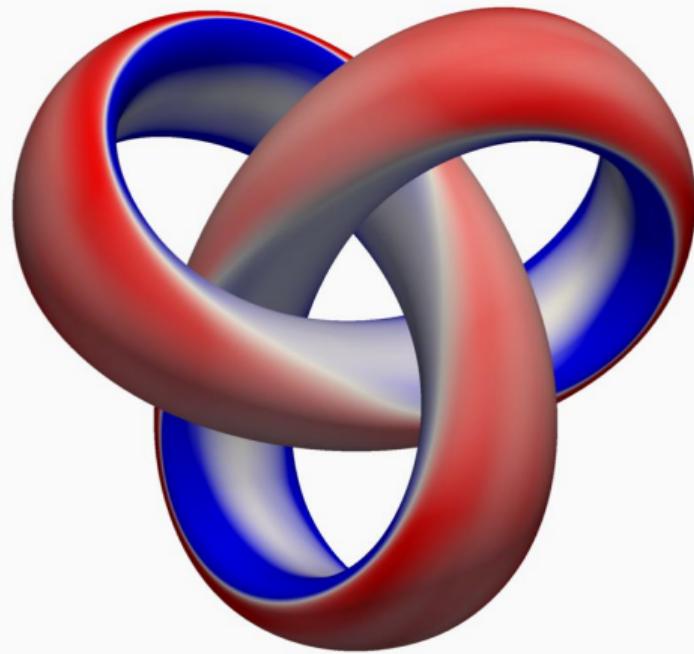
cmap="plasma"

Adding some colors: beware of perceptual artifacts



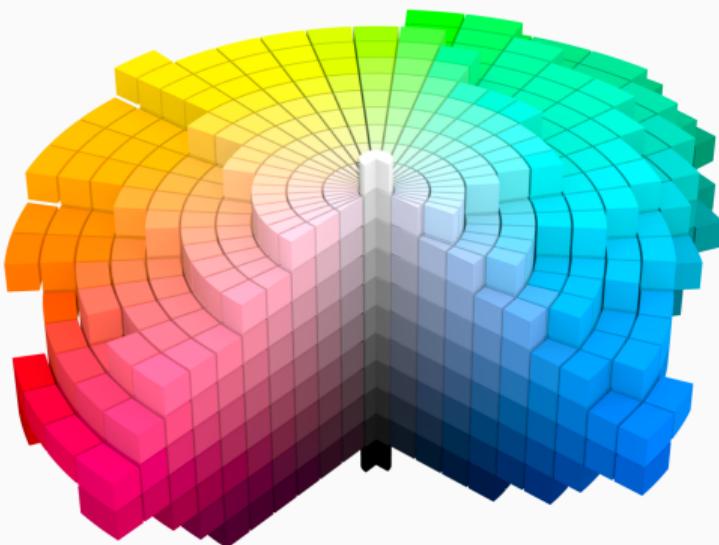
cmap="coolwarm"

```
pl.add_mesh(S, scalars=S.curvature("gaussian"), cmap=cmap, clim=(-1, 1))
```



cmap="bwr"

Choosing a colormap



The Munsell **perceptual** color space,
from the Chromatone center.



Using Matplotlib > Colors > Choosing...

Choosing Colormaps in Matplotlib

Matplotlib has a number of built-in colormaps accessible via [matplotlib.colormaps](#). There are also external libraries that have many extra colormaps, which can be viewed in the [Third-party colormaps](#) section of the Matplotlib documentation. Here we briefly discuss how to choose between the many options. For help on creating your own colormaps, see [Creating Colormaps in Matplotlib](#).

To get a list of all registered colormaps, you can do:

```
from matplotlib import colormaps  
list(colormaps)
```

Overview

The idea behind choosing a good colormap is to find a good representation in 3D colorspace for your data set. The best colormap for any given data set depends on many things including:

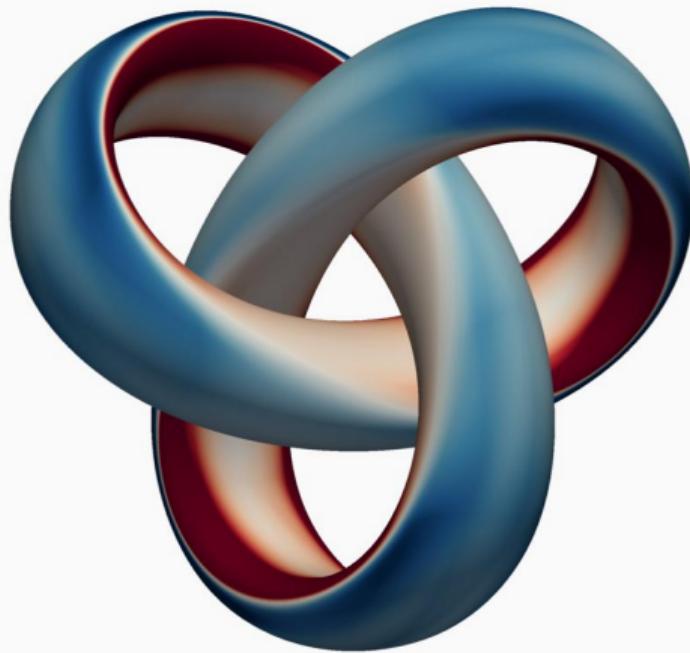
- Whether representing form or metric data ([\[Ware\]](#))
- Your knowledge of the data set (e.g., is there a critical value from which the other values deviate?)
- If there is an intuitive color scheme for the parameter you are plotting
- If there is a standard in the field the audience may be expecting

For many applications, a perceptually uniform colormap is the best choice; i.e. a colormap in which equal steps in data are perceived as equal steps in the color space. Researchers have found that the human brain perceives changes in the lightness parameter as changes in the data much better than, for example, changes in hue. Therefore, colormaps which have monotonically increasing lightness through the colormap will be better interpreted by the viewer. Wonderful examples of perceptually uniform colormaps can be found in the [Third-party colormaps](#) section as well.

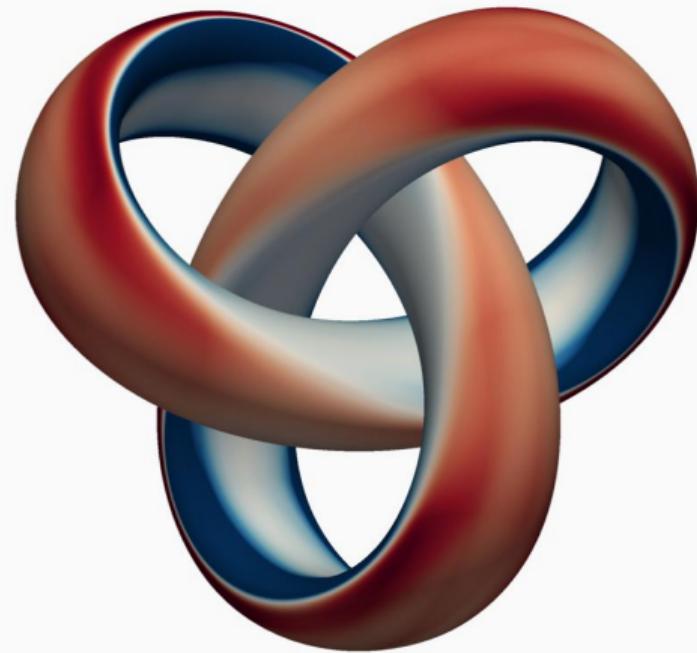
Color can be represented in 3D space in various ways. One way to represent color is using CIELAB. In CIELAB, color space is represented by lightness, L^* ; red-green, a^* ; and yellow-blue, b^* . The lightness parameter L^* can then be used to learn more about how the matplotlib colormaps will be perceived by viewers.

The **Matplotlib tutorial**
on the subject.

Adding some colors: append "_r" to reverse colormaps



cmap="RdBu"

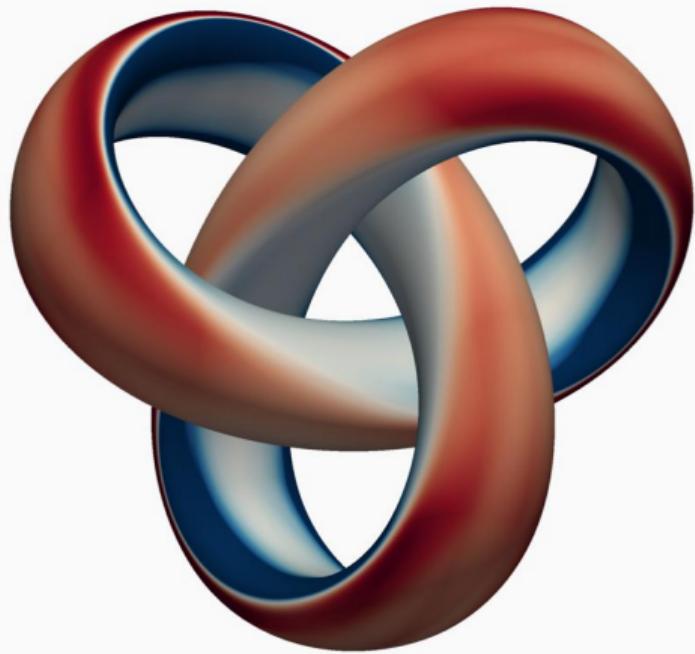


cmap="RdBu_r"

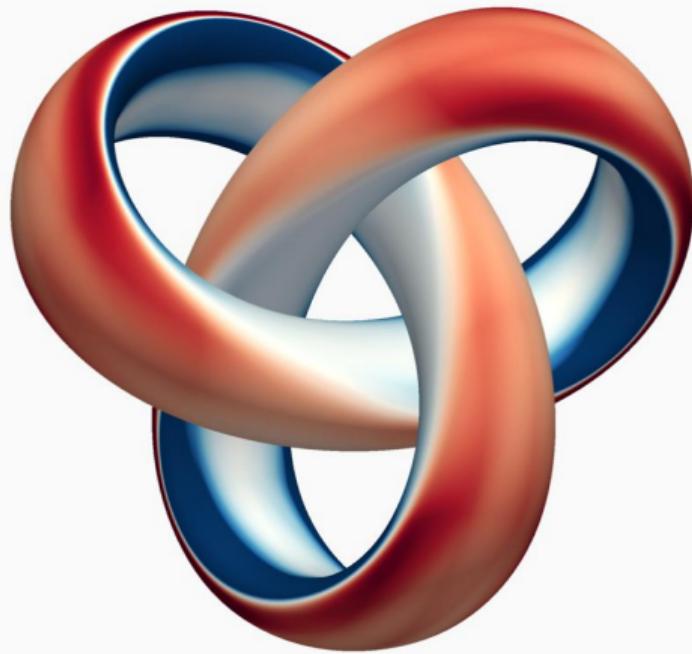
```
pl.add_mesh(S, scalars=S.curvature("gaussian"), cmap=cmap, clim=(-1, 1))
```

Lighting and material models

Adjust lighting: ambient glow



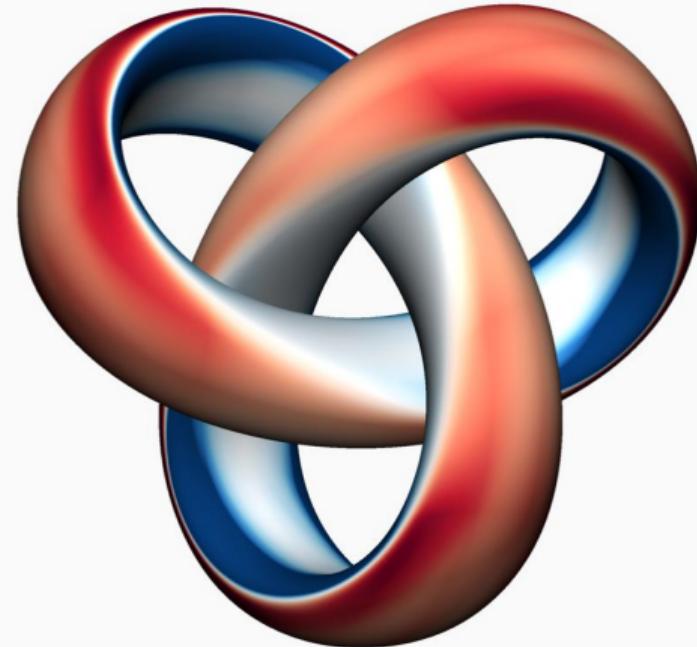
```
pl.add_mesh(S, ambient=0., ...)
```



```
pl.add_mesh(S, ambient=0.2, ...)
```

Adjust lighting: custom set with `pv.Light(...)`

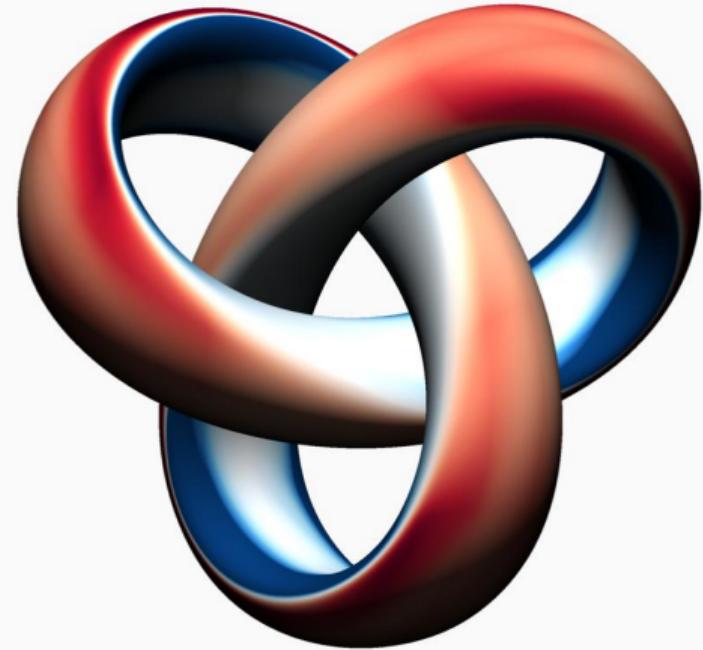
```
pl = pv.Plotter(  
    window_size=[800, 800],  
    lighting="none"  
)  
pl.add_mesh(S)  
  
light = pv.Light(intensity=0.7)  
light.set_direction_angle(40, 90)  
pl.add_light(light)  
  
light = pv.Light(  
    light_type="headlight",  
    intensity=0.7  
)  
pl.add_light(light)
```



Our light kit.

Adjust lighting: custom set with `pv.Light(...)`

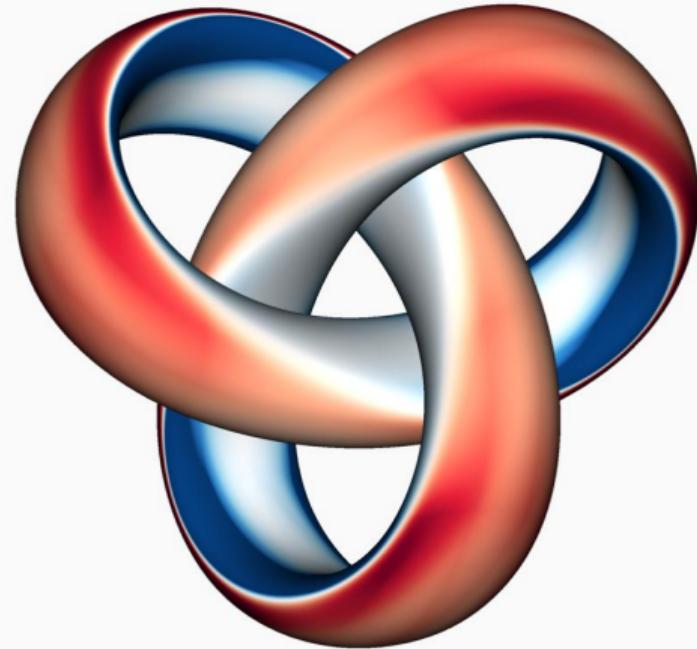
```
pl = pv.Plotter(  
    window_size=[800, 800],  
    lighting="none"  
)  
pl.add_mesh(S)  
  
light = pv.Light(intensity=1.2)  
light.set_direction_angle(40, 90)  
pl.add_light(light)  
  
light = pv.Light(  
    light_type="headlight",  
    intensity=0.2  
)  
pl.add_light(light)
```



Less headlight.

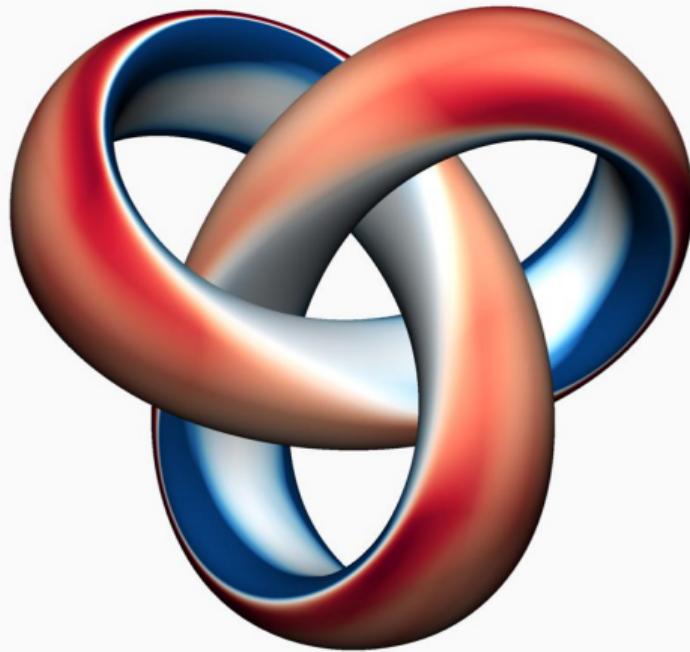
Adjust lighting: custom set with `pv.Light(...)`

```
pl = pv.Plotter(  
    window_size=[800, 800],  
    lighting="none"  
)  
pl.add_mesh(S)  
  
light = pv.Light(intensity=0.2)  
light.set_direction_angle(40, 90)  
pl.add_light(light)  
  
light = pv.Light(  
    light_type="headlight",  
    intensity=1.2  
)  
pl.add_light(light)
```

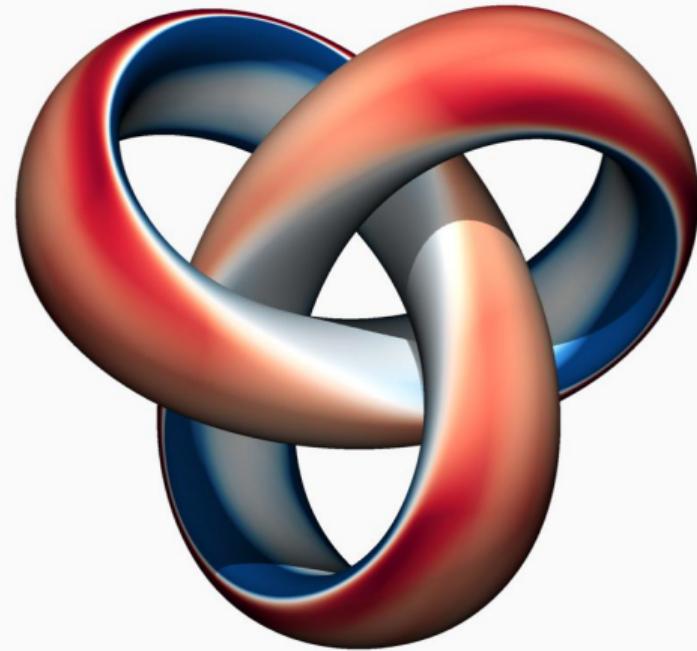


More headlight.

Adjust lighting: include shadows!

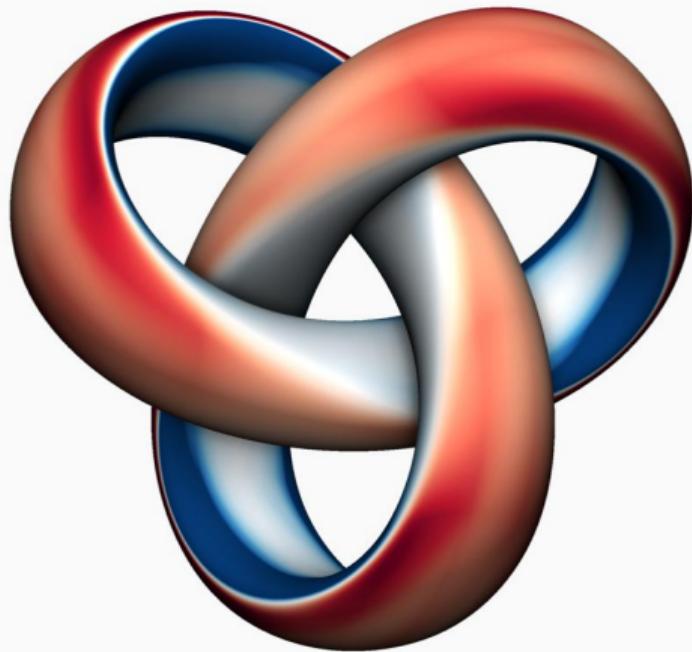


Default rendering.
Hard to read.



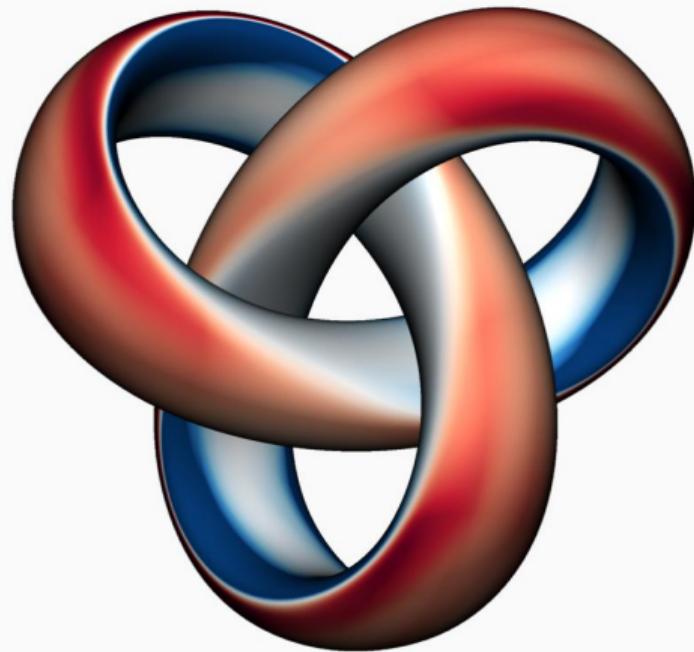
`pl.enable_shadows()`
Realistic shadow casting.

Adjust lighting: pseudo-shadows



`pl.enable_ssao(radius=1)`

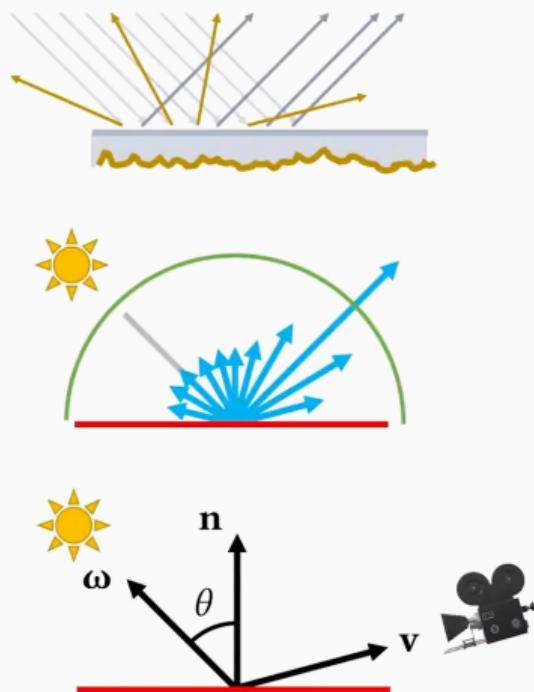
Screen Space Ambient Occlusion.



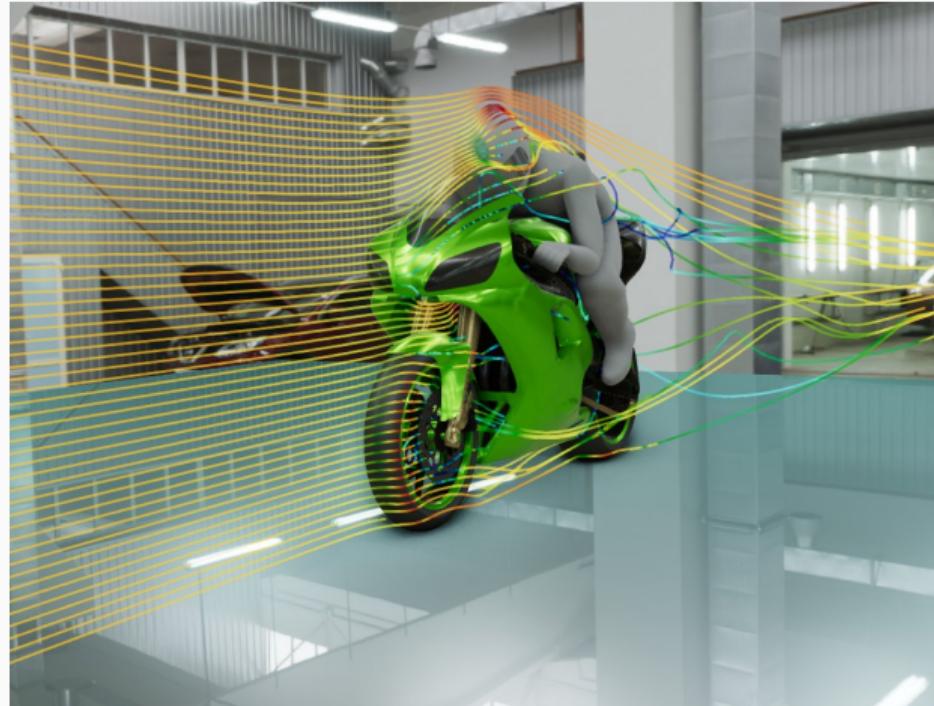
`pl.enable_eye_dome_lighting()`

Highlighting contours from the inside. 31

Adjust the material model with Physics Based Rendering (PBR)

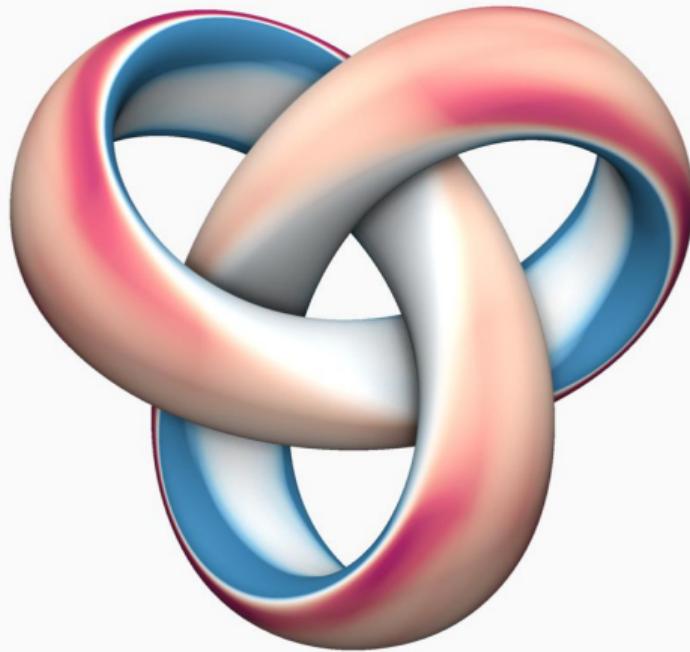


Microfacets and the rendering equation, by **Cem Yuksel**.



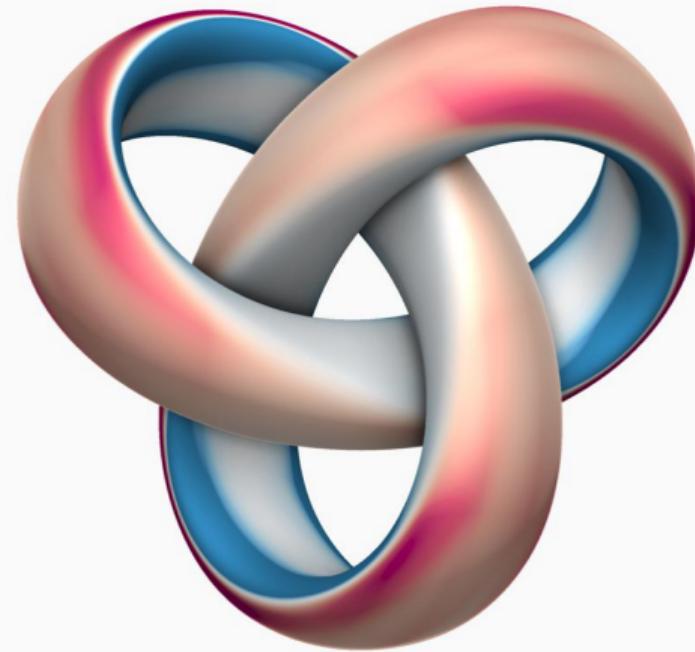
VTK supports PBR since 2019.
Made in **Lyon!**

Adjust the material model with Physics Based Rendering (PBR)



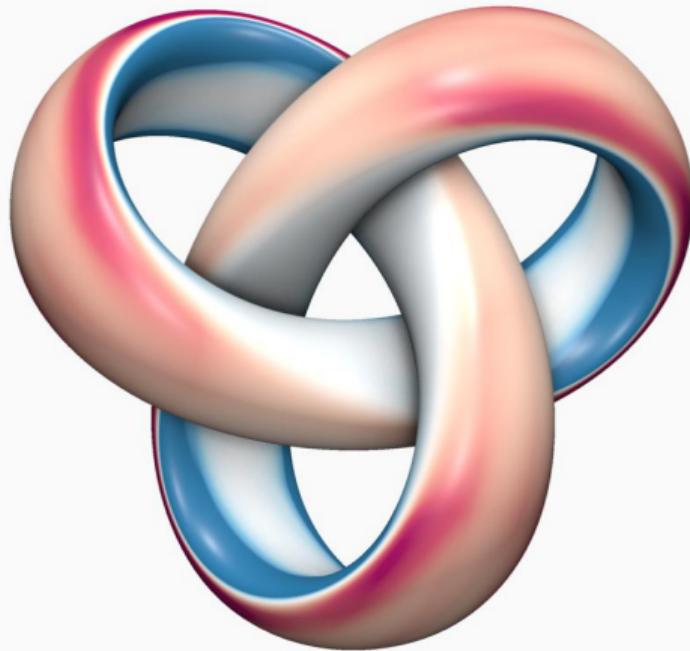
metallic=0, roughness=0.8

```
pl.add_mesh(S, pbr=True, metallic=m, roughness=r, ...)
```



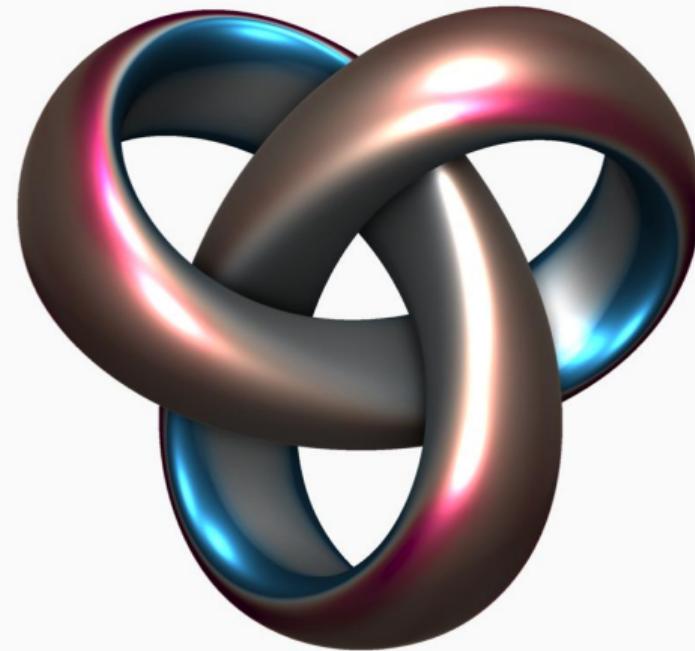
metallic=1, roughness=0.8

Adjust the material model with Physics Based Rendering (PBR)



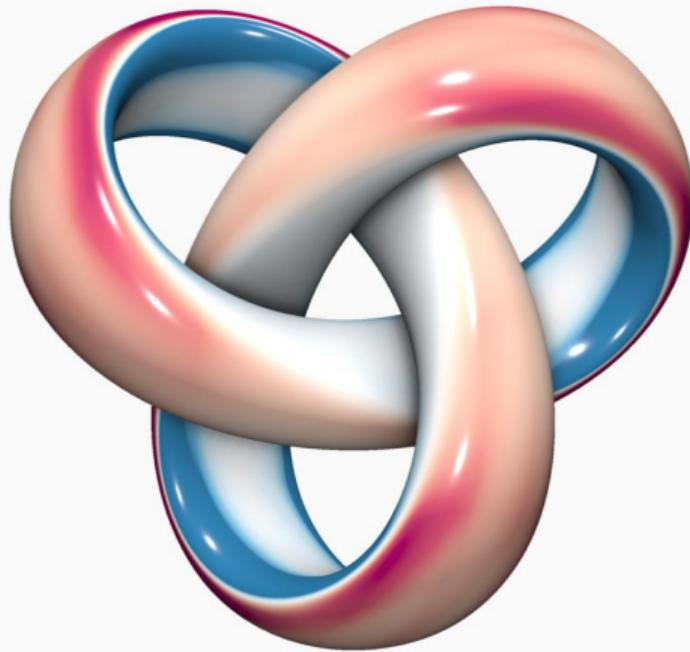
metallic=0, roughness=0.4

```
pl.add_mesh(S, pbr=True, metallic=m, roughness=r, ...)
```



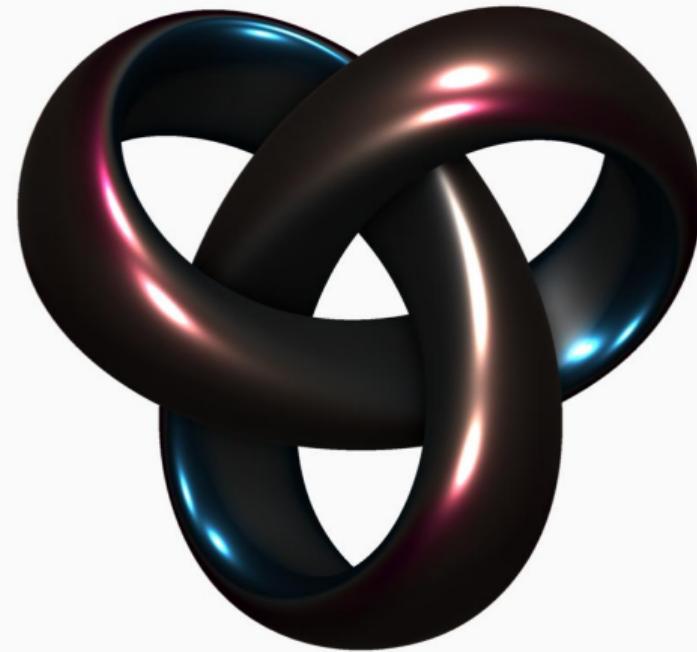
metallic=1, roughness=0.4

Adjust the material model with Physics Based Rendering (PBR)



metallic=0, roughness=0.2

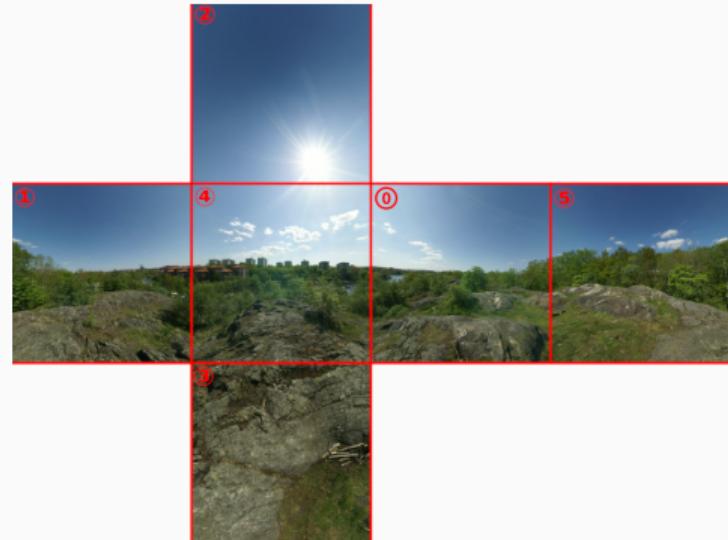
```
pl.add_mesh(S, pbr=True, metallic=m, roughness=r, ...)
```



metallic=1, roughness=0.2

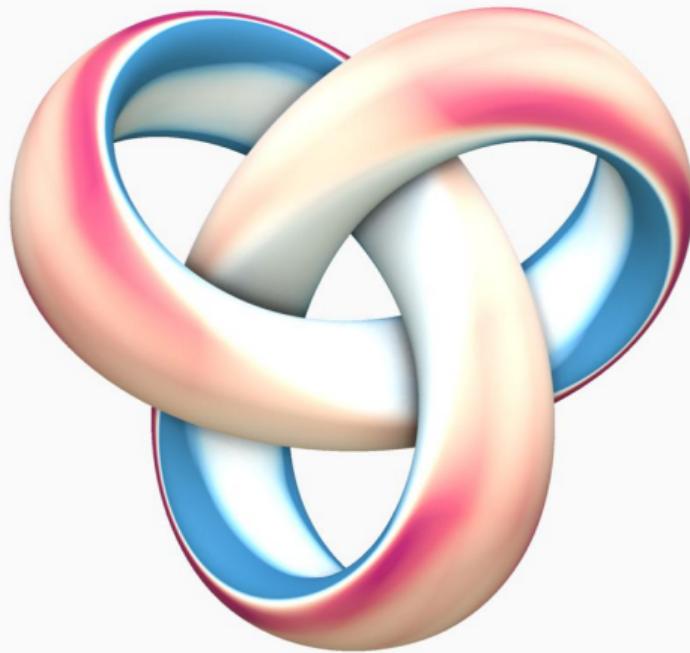
Add a cubemap – and some more lights

```
cubemap =  
pv.examples.download_sky_box_cube_map()  
pl.set_environment_texture(cubemap)  
  
if show_background:  
    pl.add_actor(cubemap.to_skybox())  
  
light = pv.Light(intensity=1.0)  
light.set_direction_angle(40, 90)  
for _ in range(3):  
    pl.add_light(light)  
  
...  
pl.show()
```



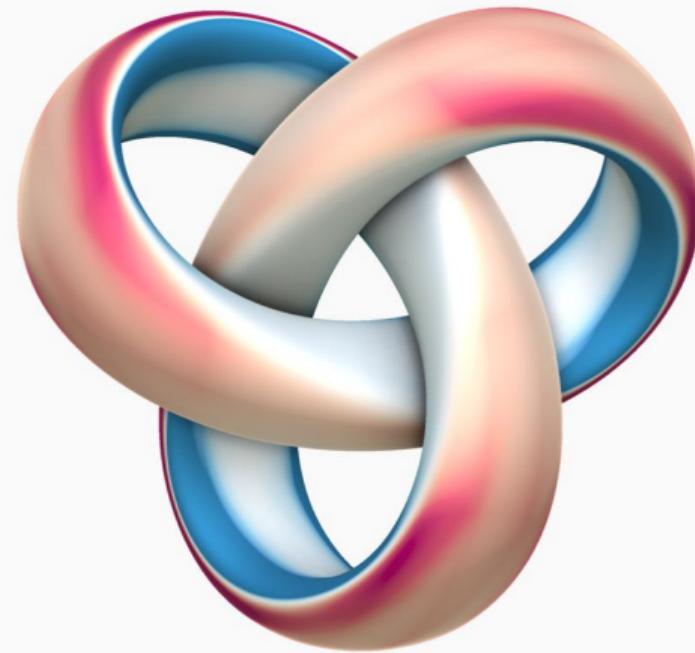
A **cubemap** encodes the **environment** lighting using **6 square images**.

Adjust the material model with Physics Based Rendering (PBR)



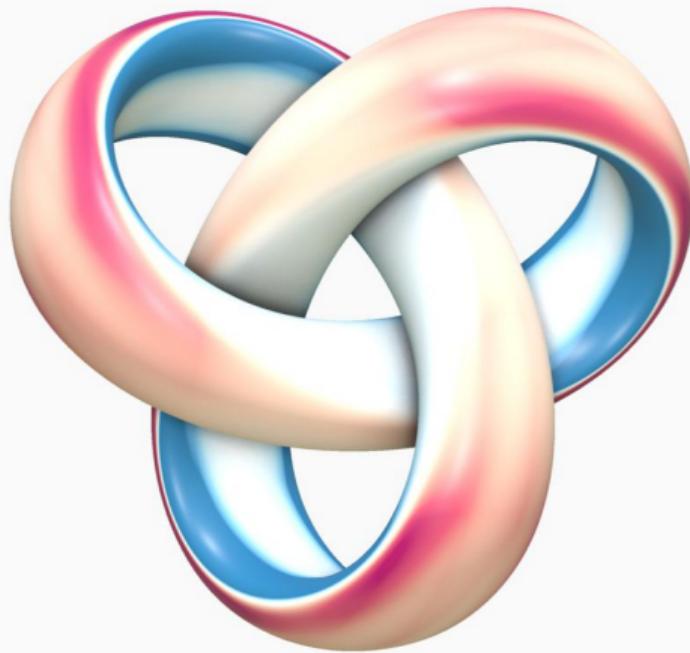
metallic=0.5, roughness=0.8

```
pl.add_mesh(S, pbr=True, metallic=m, roughness=r, ...)
```



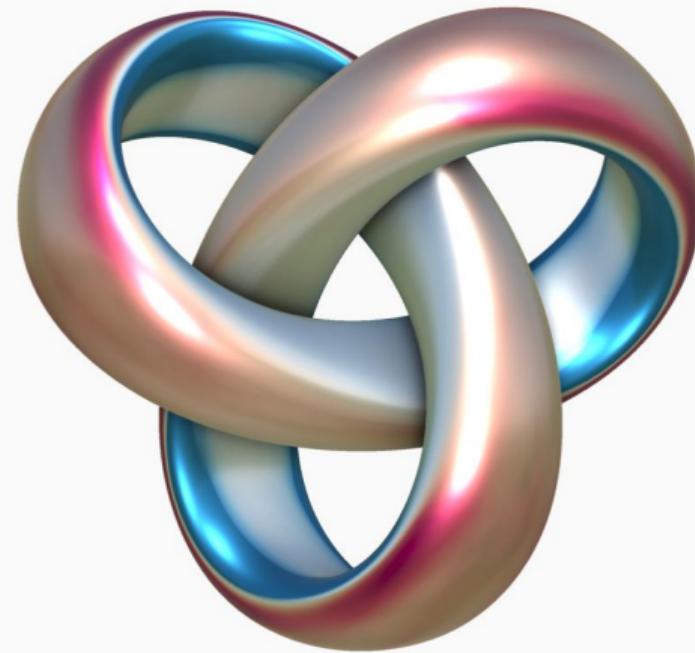
metallic=1, roughness=0.8

Adjust the material model with Physics Based Rendering (PBR)



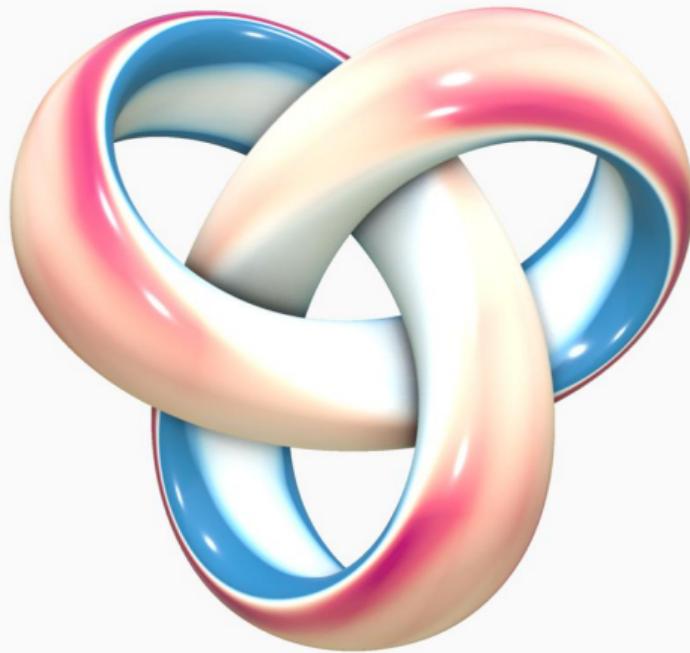
metallic=0.5, roughness=0.4

```
pl.add_mesh(S, pbr=True, metallic=m, roughness=r, ...)
```



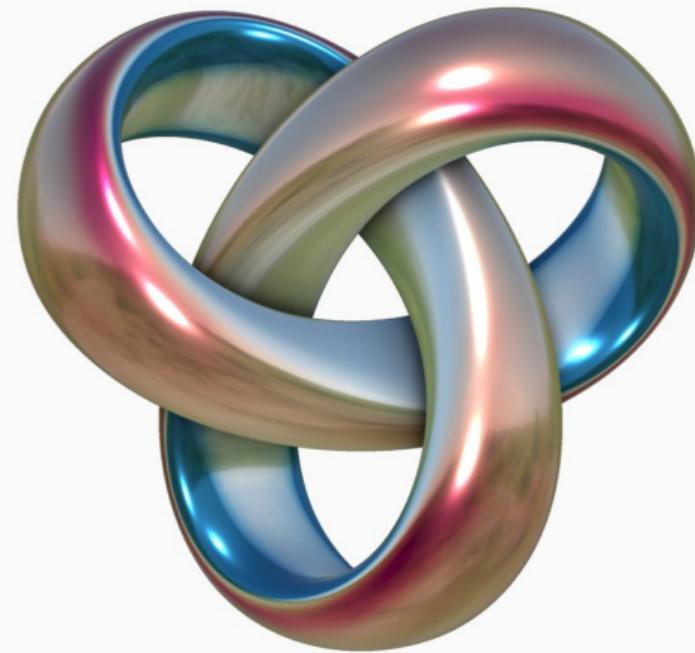
metallic=1, roughness=0.4

Adjust the material model with Physics Based Rendering (PBR)



metallic=0.5, roughness=0.2

```
pl.add_mesh(S, pbr=True, metallic=m, roughness=r, ...)
```



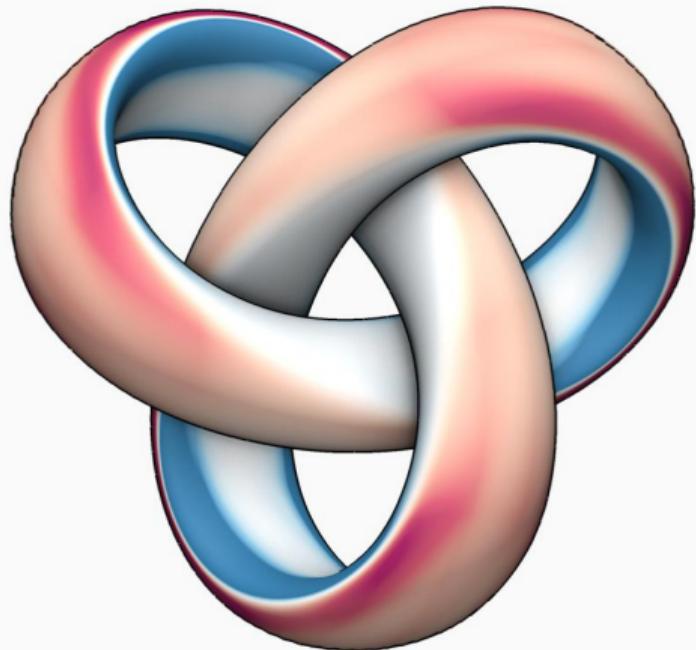
metallic=1, roughness=0.2

Silhouettes and animations

Highlight the silhouette with a simple filtering step

Extract the **mesh edges** that look “tangent” and draw them with a **bold marker**.

```
pl.add_mesh(  
    s,  
    silhouette=dict(  
        color="black",  
        line_width=10.0,  
    ),  
    ...  
)
```



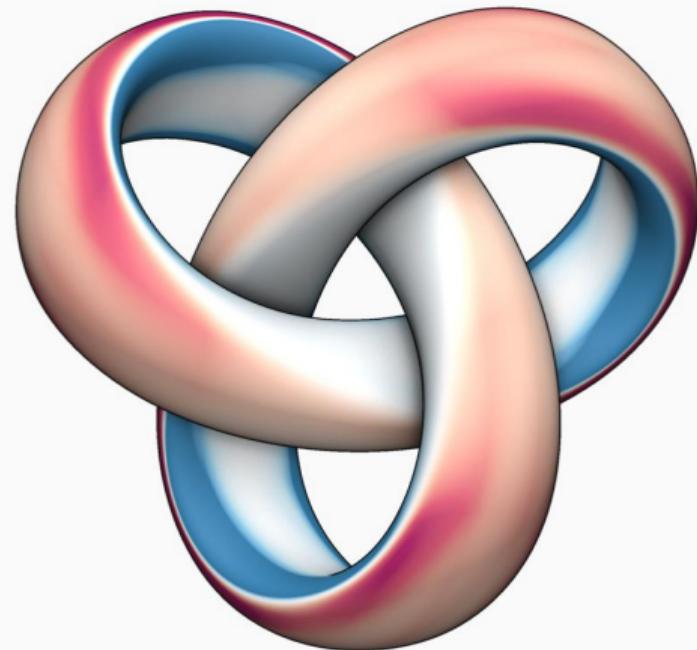
Fast and simple, but crenelated.

Highlight the silhouette with a black shell

Inflate the mesh, paint it black,
and **cull the triangles** facing the camera.

```
S["sw"] = w * np.ones(surface.n_points)  
shell = S.warp_by_scalar(scalars="sw")
```

```
pl.add_mesh(S, ...)  
pl.add_mesh(  
    shell,  
    color="black",  
    culling="front",  
    interpolation="pbr",  
    roughness=1,  
)
```



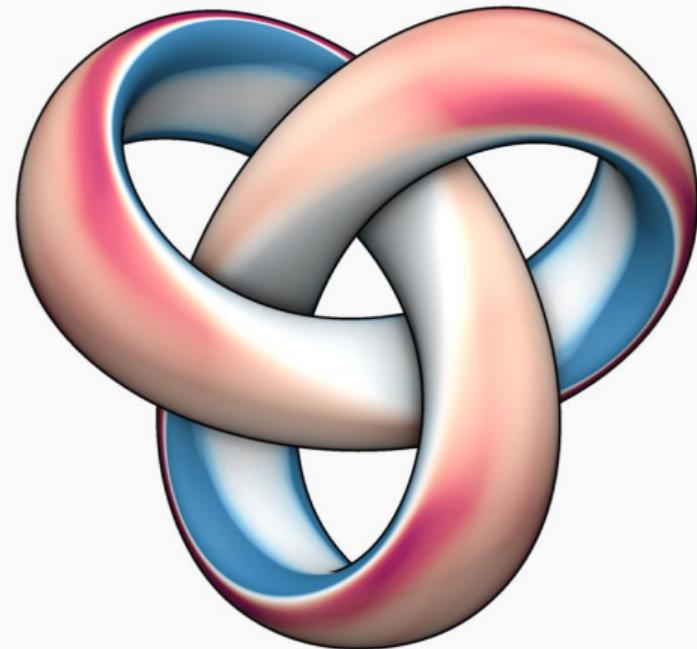
w = 0.01

Highlight the silhouette with a black shell

Inflate the mesh, paint it black,
and **cull the triangles** facing the camera.

```
S["sw"] = w * np.ones(surface.n_points)  
shell = S.warp_by_scalar(scalars="sw")
```

```
pl.add_mesh(S, ...)  
pl.add_mesh(  
    shell,  
    color="black",  
    culling="front",  
    interpolation="pbr",  
    roughness=1,  
)
```



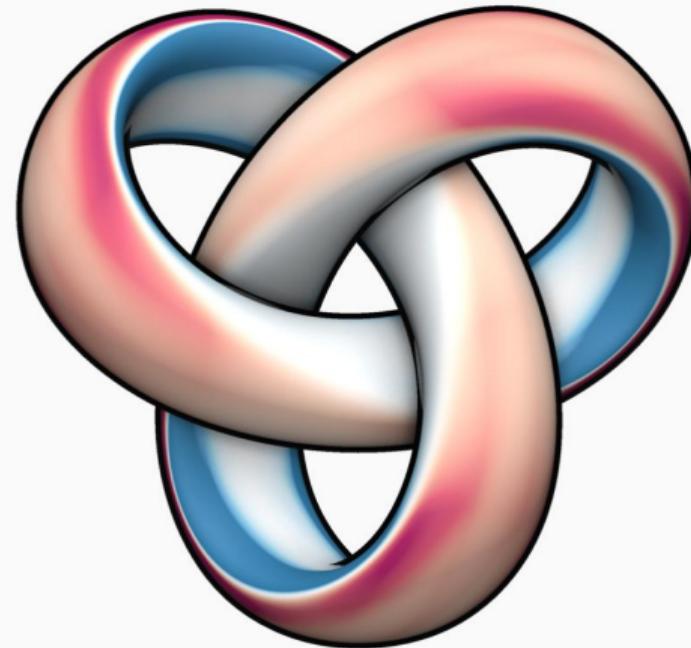
w = 0.02

Highlight the silhouette with a black shell

Inflate the mesh, paint it black,
and **cull the triangles** facing the camera.

```
S["sw"] = w * np.ones(surface.n_points)  
shell = S.warp_by_scalar(scalars="sw")
```

```
pl.add_mesh(S, ...)  
pl.add_mesh(  
    shell,  
    color="black",  
    culling="front",  
    interpolation="pbr",  
    roughness=1,  
)
```

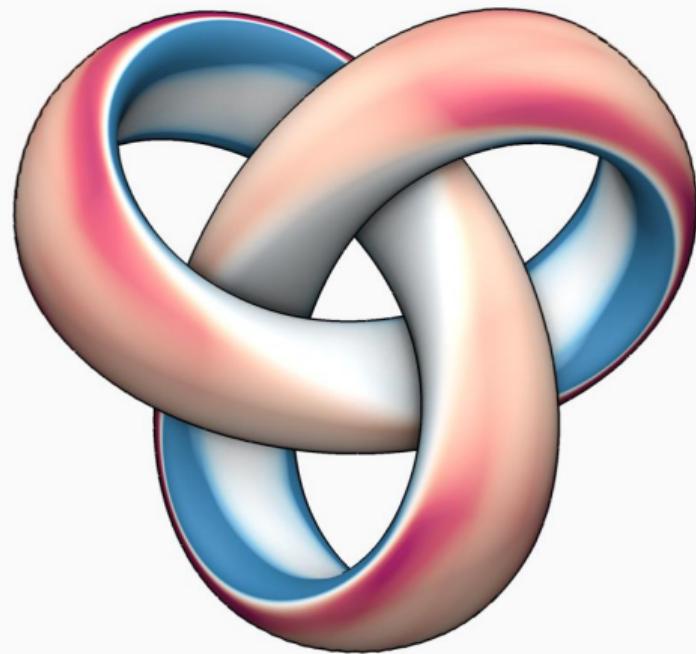


w = 0.05

Create movies!

Save screenshots and fuse them with **ffmpeg**
or update point features and **write frames**.

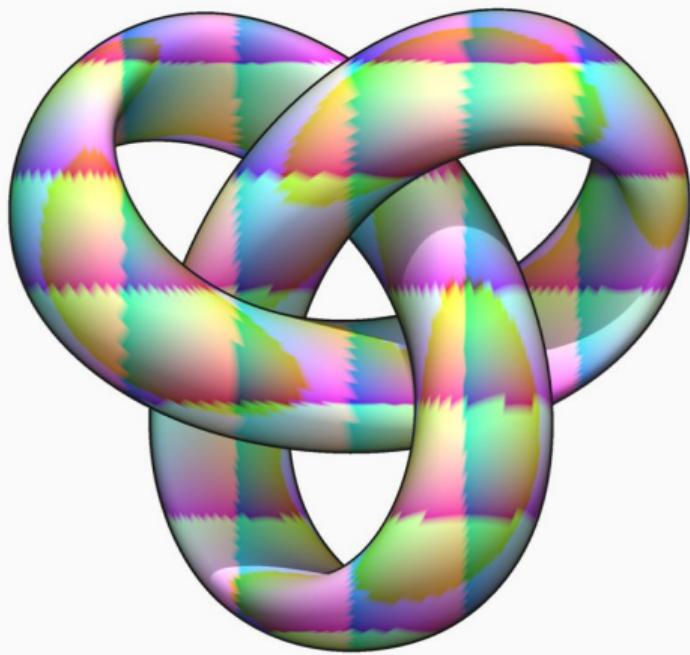
```
pl.open_movie("flow.mp4", quality=10)  
  
for frame in range(1000):  
    S.points += ...  
    S["signal"] = ...  
    pl.write_frame()  
  
pl.close()
```



Mean curvature flow.

Textures

Define custom RGB(A) vertex colors

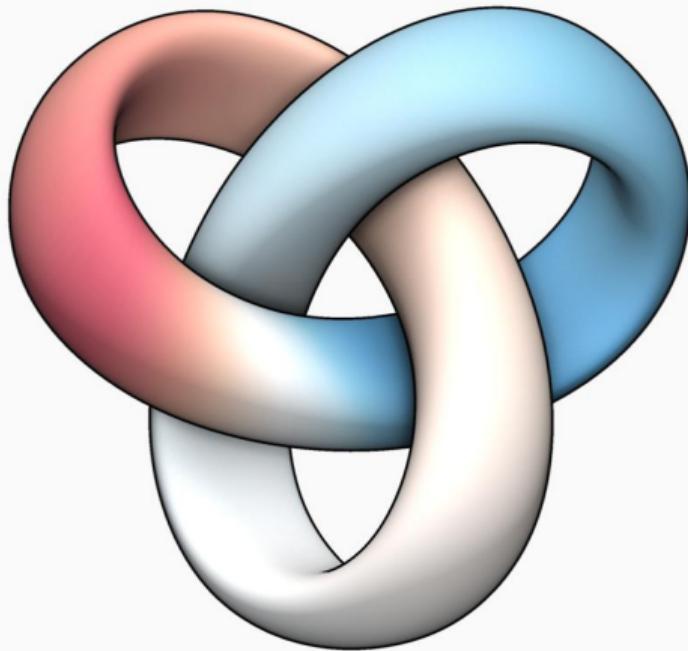


RGB = **S.points**

```
pl.add_mesh(S, rgb=True, scalars=RGB, ...)
```

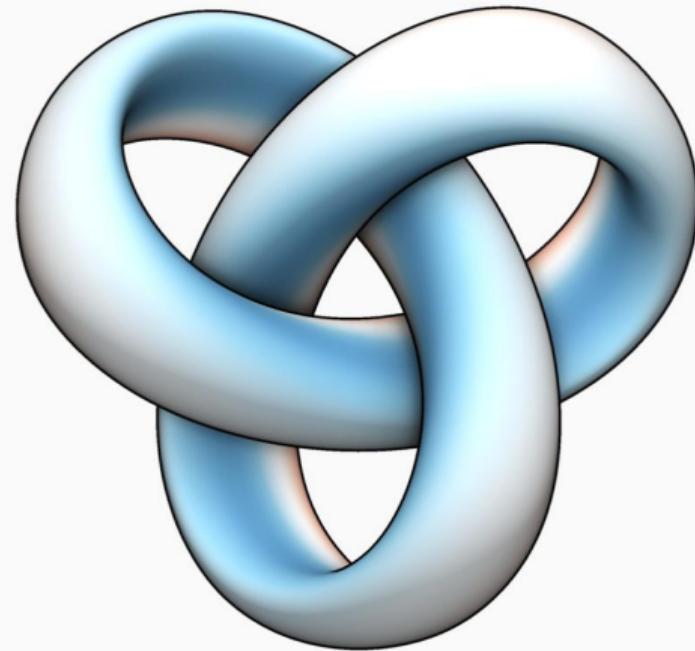
RGB = (**S.point_normals** + 1) / 2

Define custom UV coordinates in [0,1] and fetch RGBA from a texture image



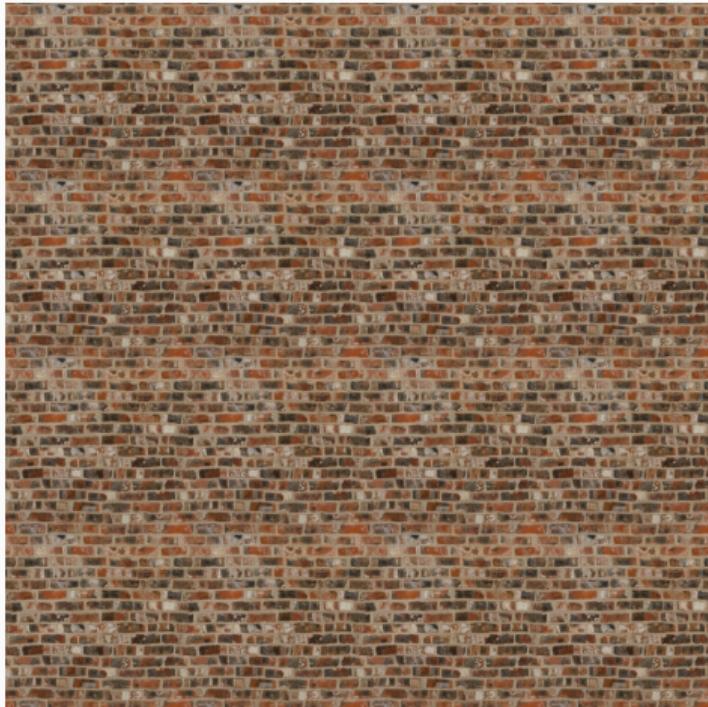
U coordinates.

`S.active_texture_coordinates = UV.reshape(S.n_points, 2)`



V coordinates.

Define custom UV coordinates in [0,1] and fetch RGBA from a texture image



Texture image.

```
pl.add_mesh(S, texture=pv.read_texture("bricks.jpg"), ...)
```



Textured mesh.

Going further with glTF models – use textures for material properties

The screenshot shows the homepage of Poly Haven. At the top, there's a navigation bar with links for Assets, Add-on, Gallery, Support Us, Blog, About/Contact, and a language switcher. Below the header, there are three main sections: "HDRIs" featuring a green field scene, "Textures" featuring a wooden texture, and "Models" featuring a baseball. Each section has a "Browse" button. A large banner at the bottom left says "100% Free" and includes a note about CC0 licensing.

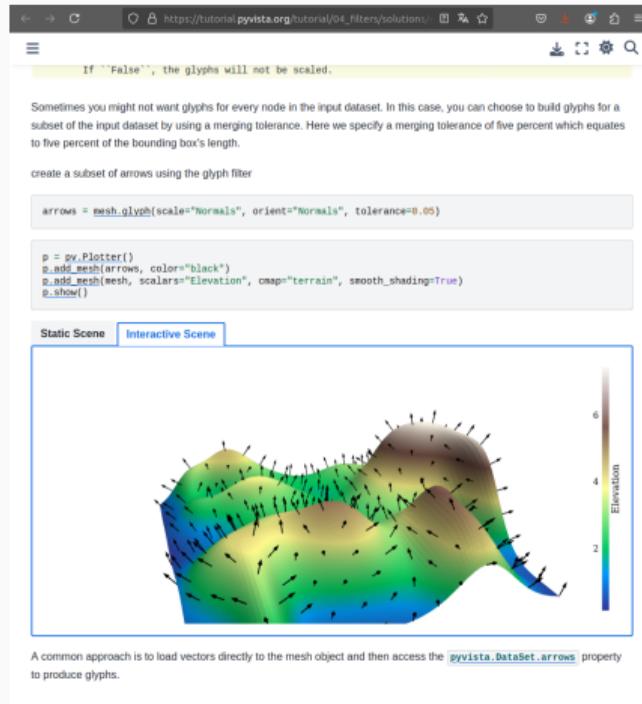
Download assets from
e.g. **PolyHaven**.

The screenshot shows a PyVista documentation page. It features a large image of a red Iron Man helmet with glowing blue eyes. Above the image is some sample Python code for importing a glTF file. Below the image is a caption stating, "You can also directly read in gltf files and extract the underlying mesh."

```
pl = pv.Plotter()
pl.import_gltf('helmet.gltf')
pl.set_environment_texture(texture)
pl.camera.zoom(1.7)
pl.show()
```

Use them with PyVista!
pl.`import_gltf`(...)

Going further – the PyVista documentation is great



If ``False'', the glyphs will not be scaled.

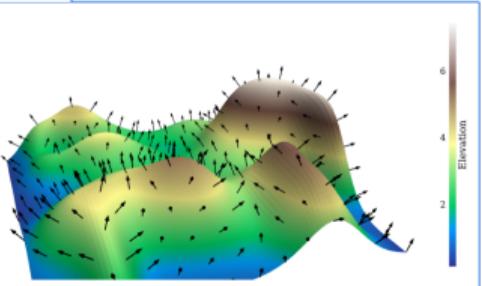
Sometimes you might not want glyphs for every node in the input dataset. In this case, you can choose to build glyphs for a subset of the input dataset by using a merging tolerance. Here we specify a merging tolerance of five percent which equates to five percent of the bounding box's length.

create a subset of arrows using the glyph filter

```
arrows = mesh.glyph(scale="Normals", orient="Normals", tolerance=0.05)
```

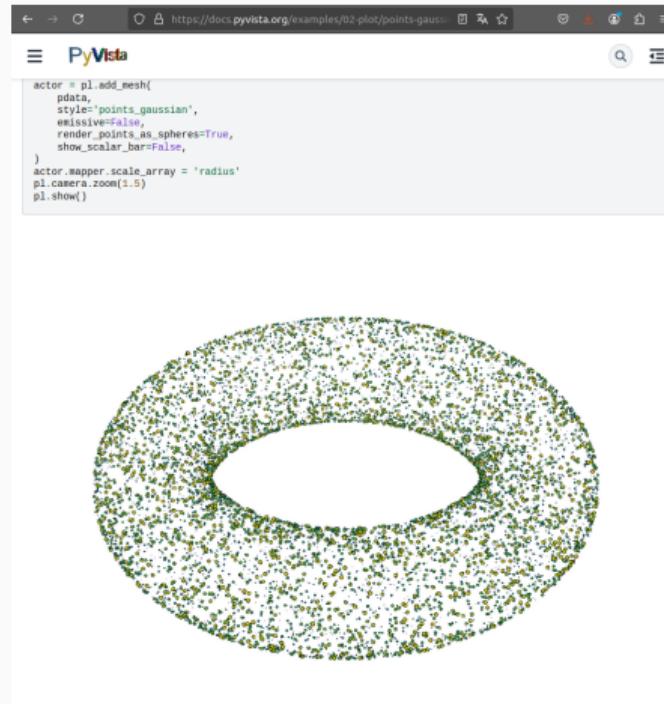
```
p = pv.Plotter()
p.add_mesharrows, color="black")
p.add_mesh(mesh, scalars="Elevation", cmap="terrain", smooth_shading=True)
p.show()
```

Static Scene Interactive Scene



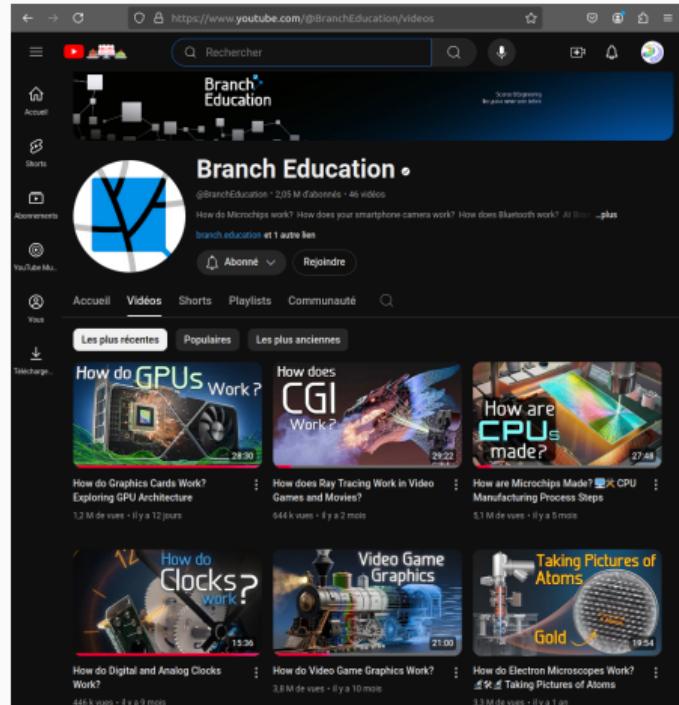
A common approach is to load vectors directly to the mesh object and then access the `pyvista.DataSet.arrows` property to produce glyphs.

Use **glyphs** to display **arrows**.

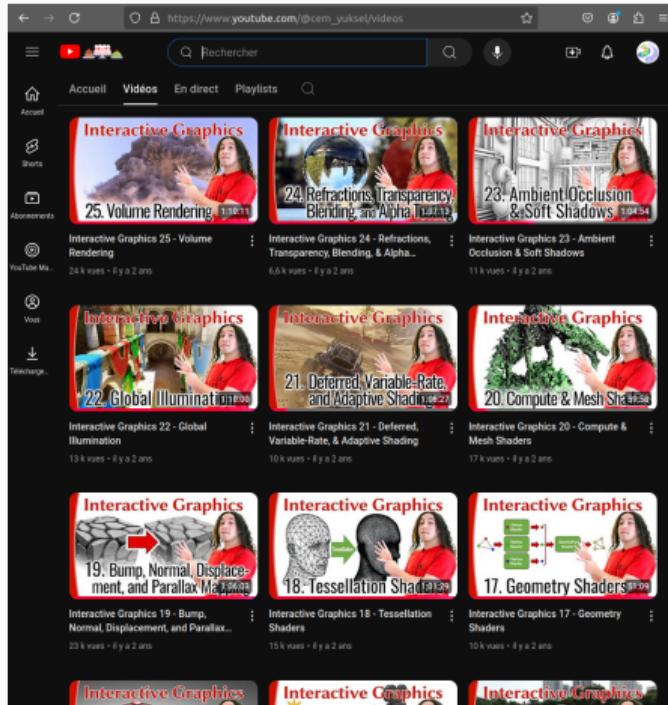


Use **point Gaussians**
for **non-uniform** point clouds.

Going further – two fantastic YouTube channels



Branch Education's deep dives
into **modern hardware**.



Cem Yuksel's lectures about
(interactive) **computer graphics**.

Time for a Paraview demo?

References i
