

Shape Modeling and Geometry Processing

Exercise 1: libigl “Hello World”

Acknowledgments: Olga Diamanti, Julian Panetta



Libigl

- Experiment with the geometry processing library

The screenshot shows the libigl website. At the top is a blue header with the 'libigl' logo on the left, a search bar in the center, and the GitHub repository 'libigl/libigl' with 2.4k stars and 750 forks on the right. Below the header, the main content area features the title 'libigl - A simple C++ geometry processing library' with a pencil icon. Underneath the title are three status boxes: 'Build passing' (green), 'Nightly failing' (red), and 'Install with conda' (green). The central part of the page is a collage of circular images showing various 3D models and processing results, including a cow, a human head, a robot, and a grid of small images. To the left of the main content is a navigation menu with links: 'libigl', 'Home', 'Tutorial', 'Python Bindings', 'Compilation', 'Contributing', 'Misc', 'FAQ', and 'About'. To the right is a 'Table of contents' section listing: 'Tutorial', 'libigl Example Project', 'Coding Guidelines and Tips', 'Installation', 'Dependencies' (with sub-items: 'Optional Dependencies', 'Downloading Dependencies', 'GCC and the Optional CGAL Dependency', 'OpenMP and Windows'), 'Download', 'Known Issues', 'Unit Testing', 'How to Contribute', and 'License'.

<https://libigl.github.io>

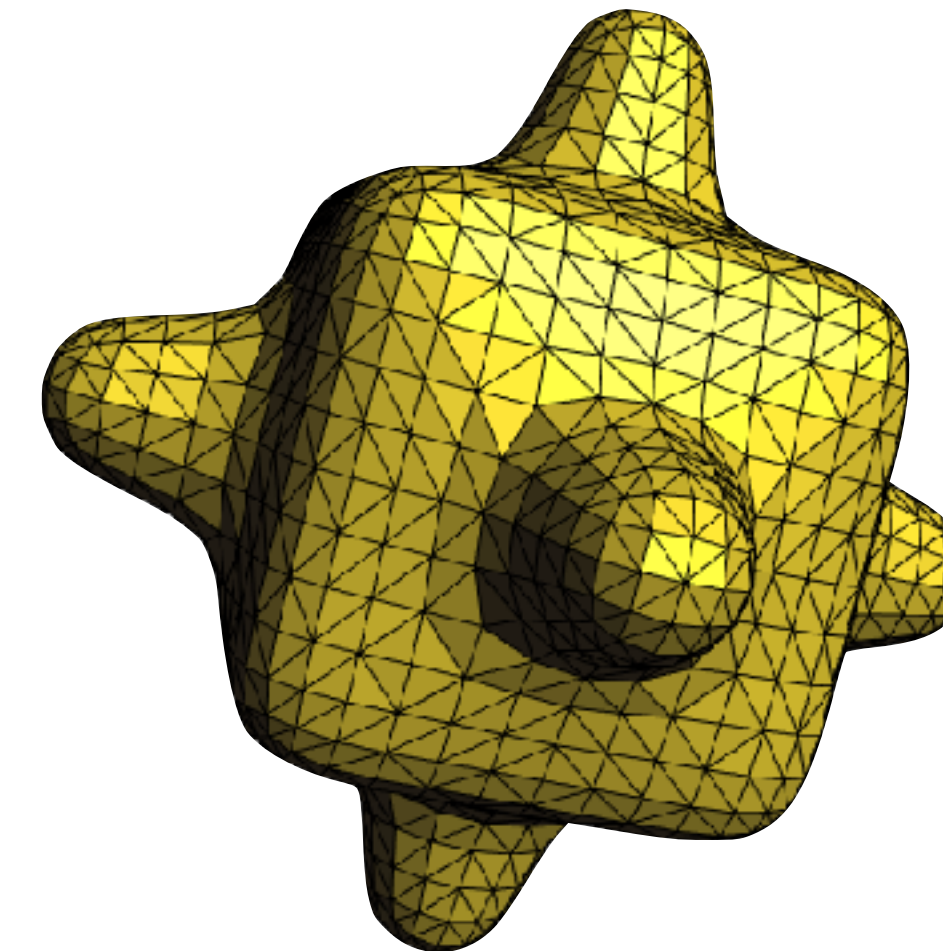


Read and visualize a mesh

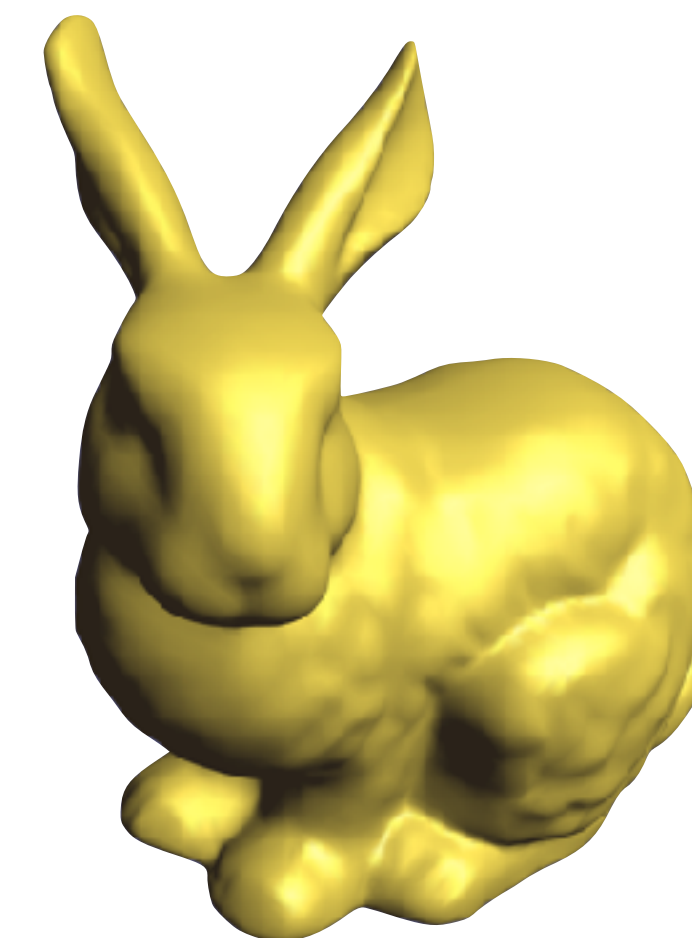
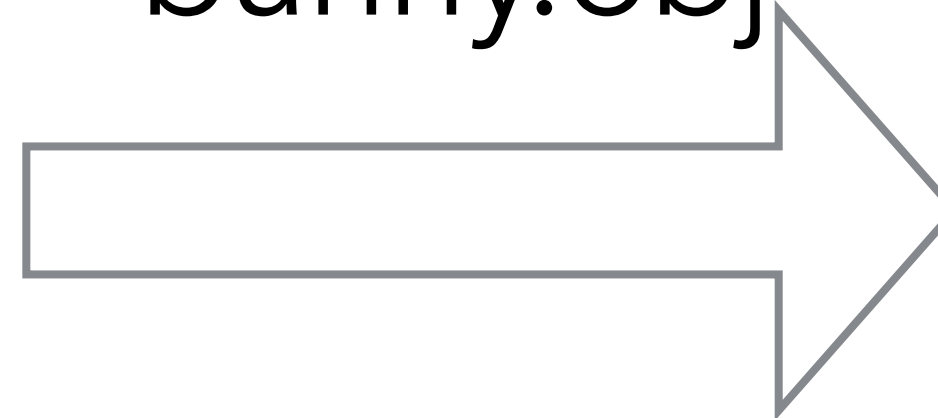
```
# Wavefront OBJ file
v 30.50959969 12.17459898 -15.84426970
v 30.49857998 11.87718728 -15.40759913
v 30.53679943 12.68500615 -14.82485356
v 30.67168999 11.71161003 -15.78844530
...
f 633/16706 11590/29979 4339/16704
f 11590/3161 633/16716 19901/16699
...
```

```
OFF
1250 2496 0
-2.09105 -2.09105 2.09105
-0.833333 -2.23958 2.23958
0.833333 -2.23958 2.23958
2.09105 -2.09105 2.09105
...
3 940 83 320
3 386 0 941
...
```

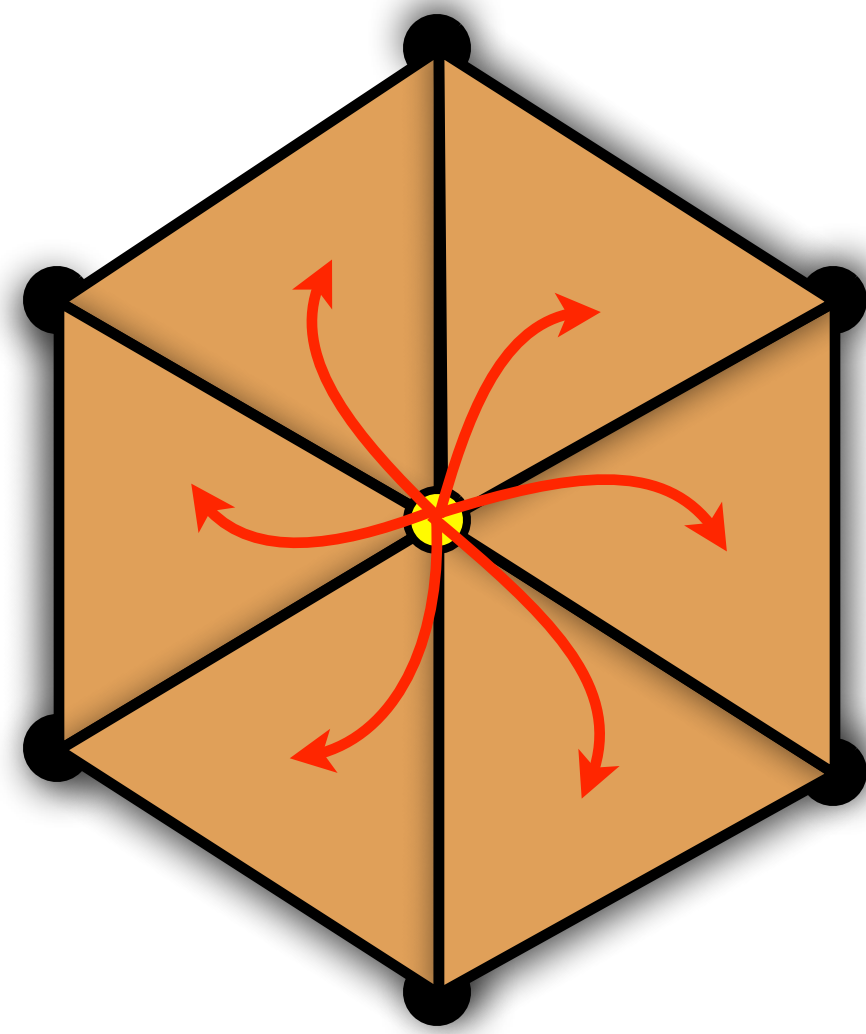
bumpy_cube.off



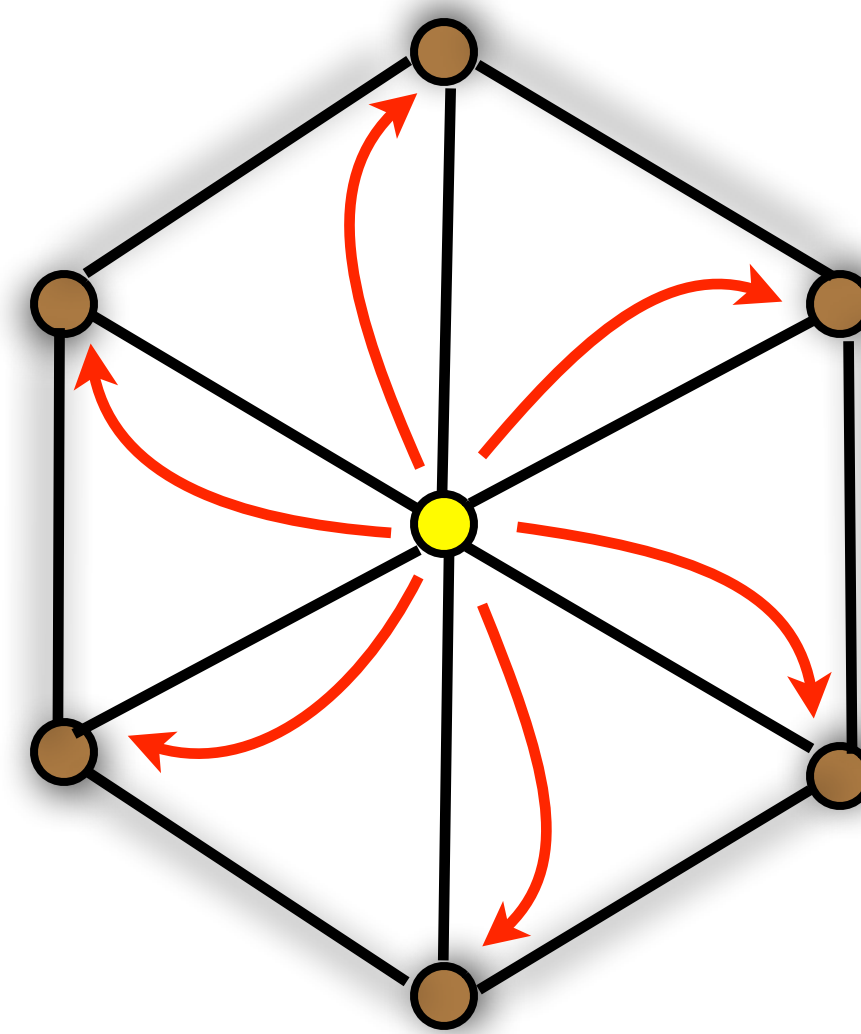
bunny.obj



Perform simple neighborhood calculations



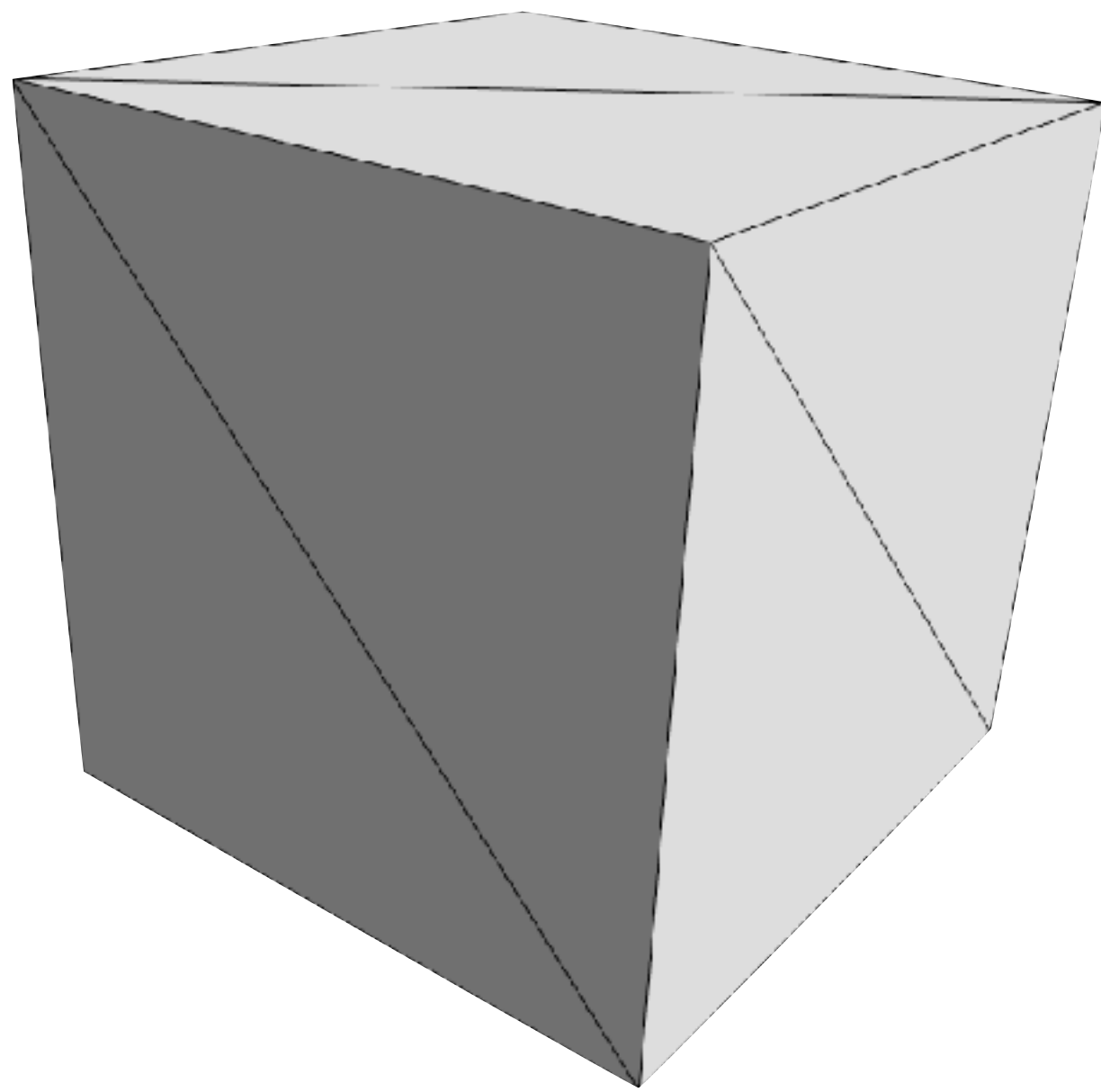
vertex-to-face



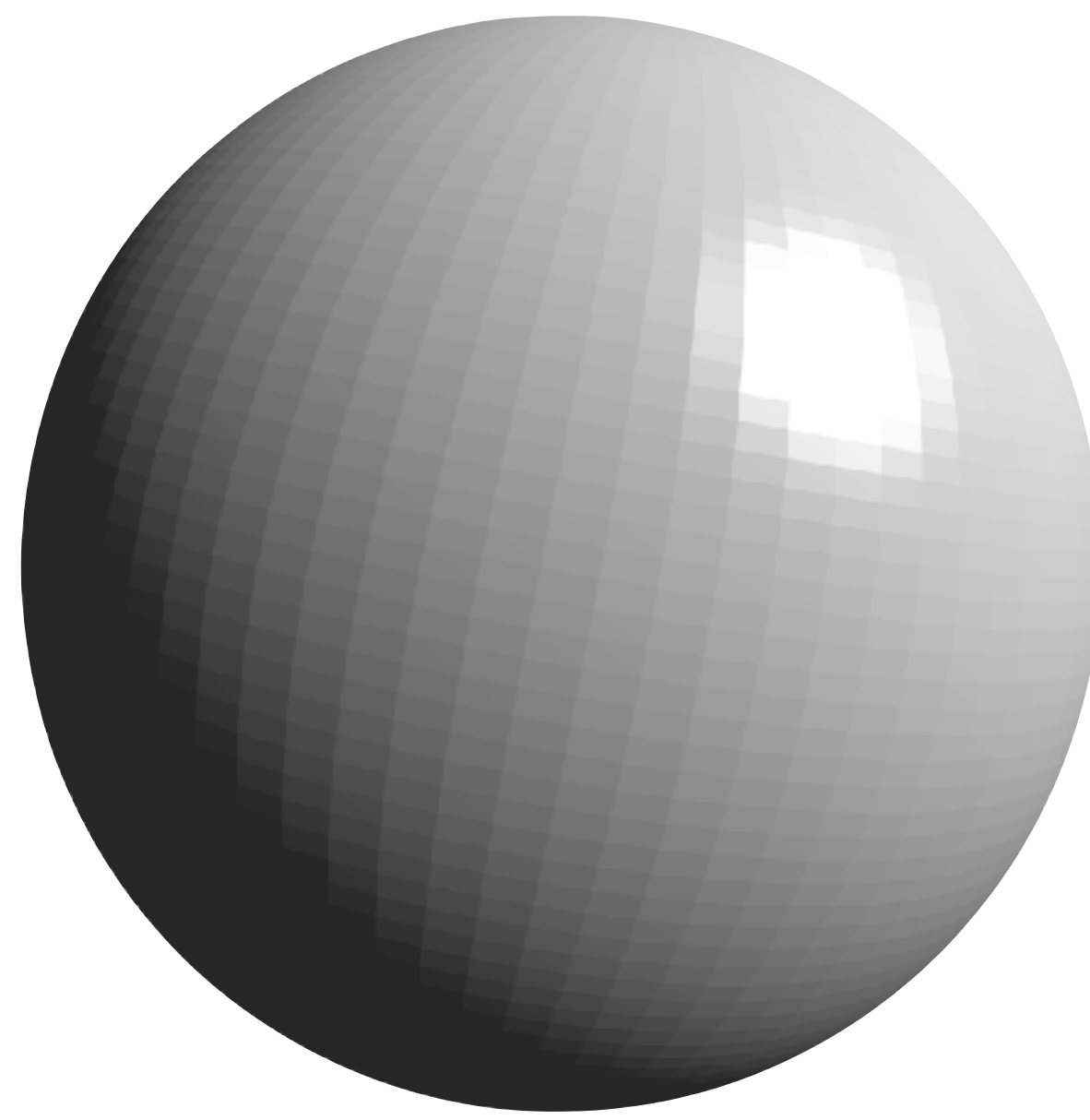
vertex-to-vertex

Flat shading

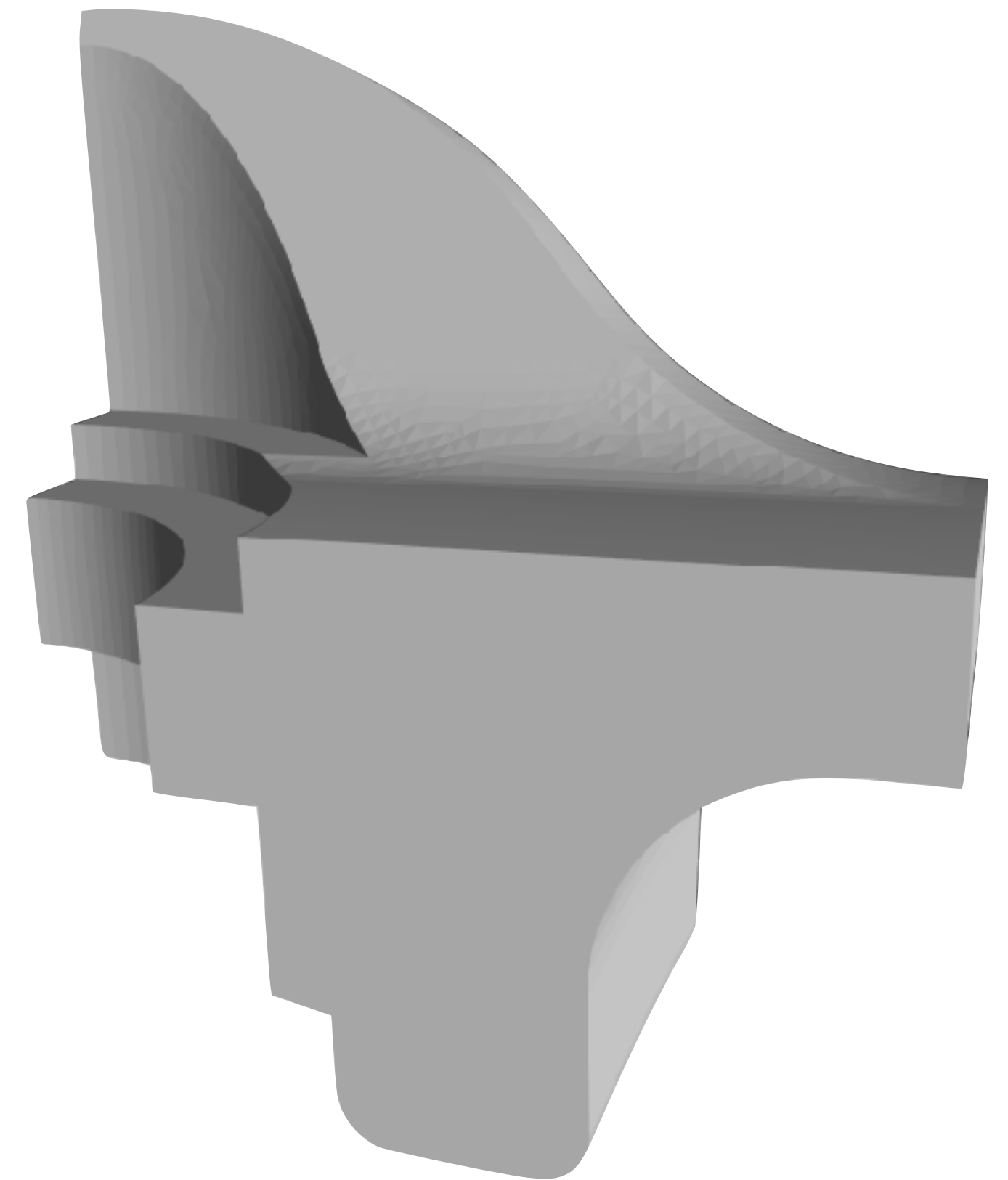
- Compute one normal per polygon



Creased surfaces render well.

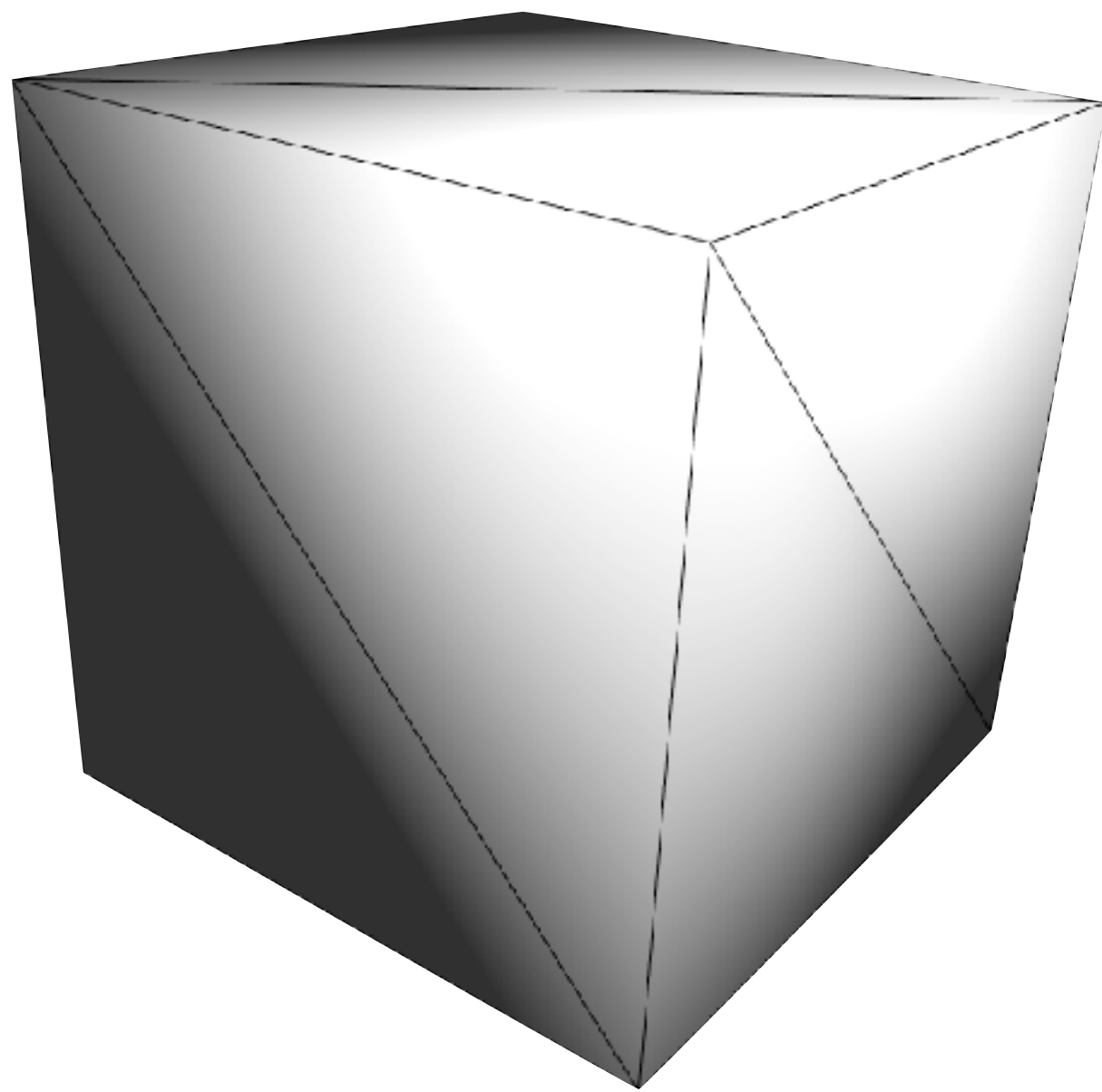


But discontinuous normals lead to poor results for smooth surfaces.

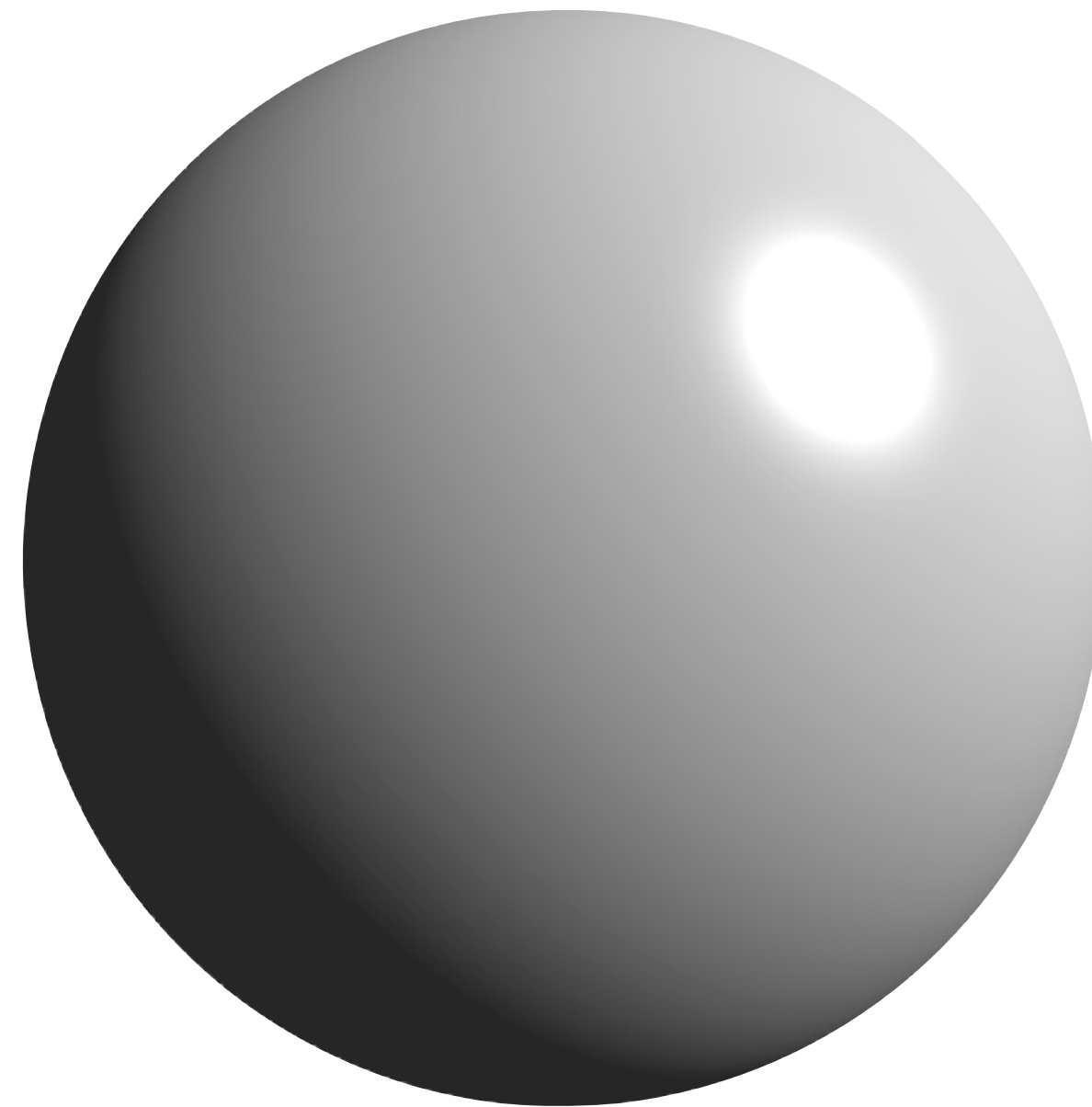


Smooth (Gouraud) Shading

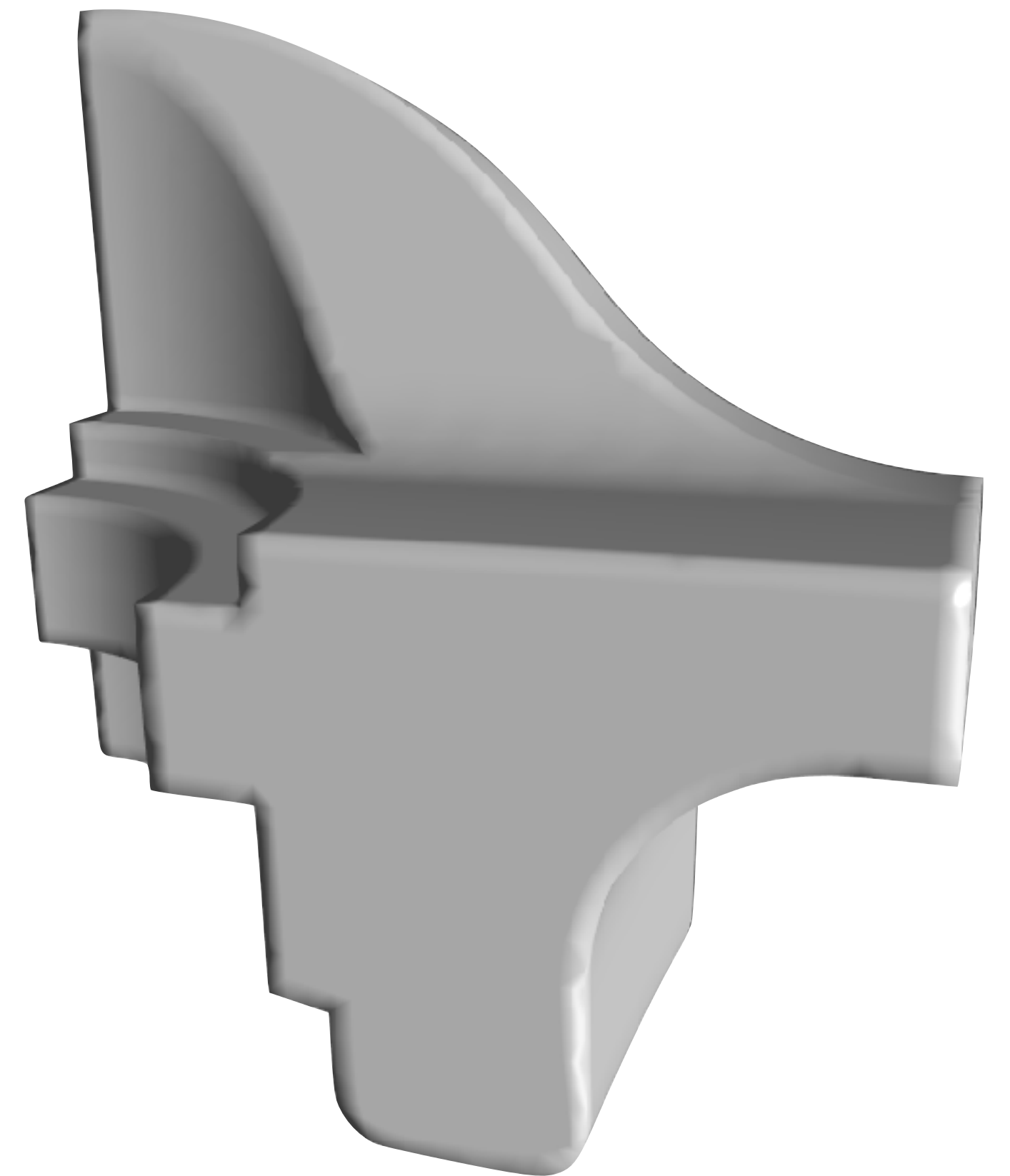
- One normal per vertex (average incident tri's normals)



Creased surfaces look strange
and burry.

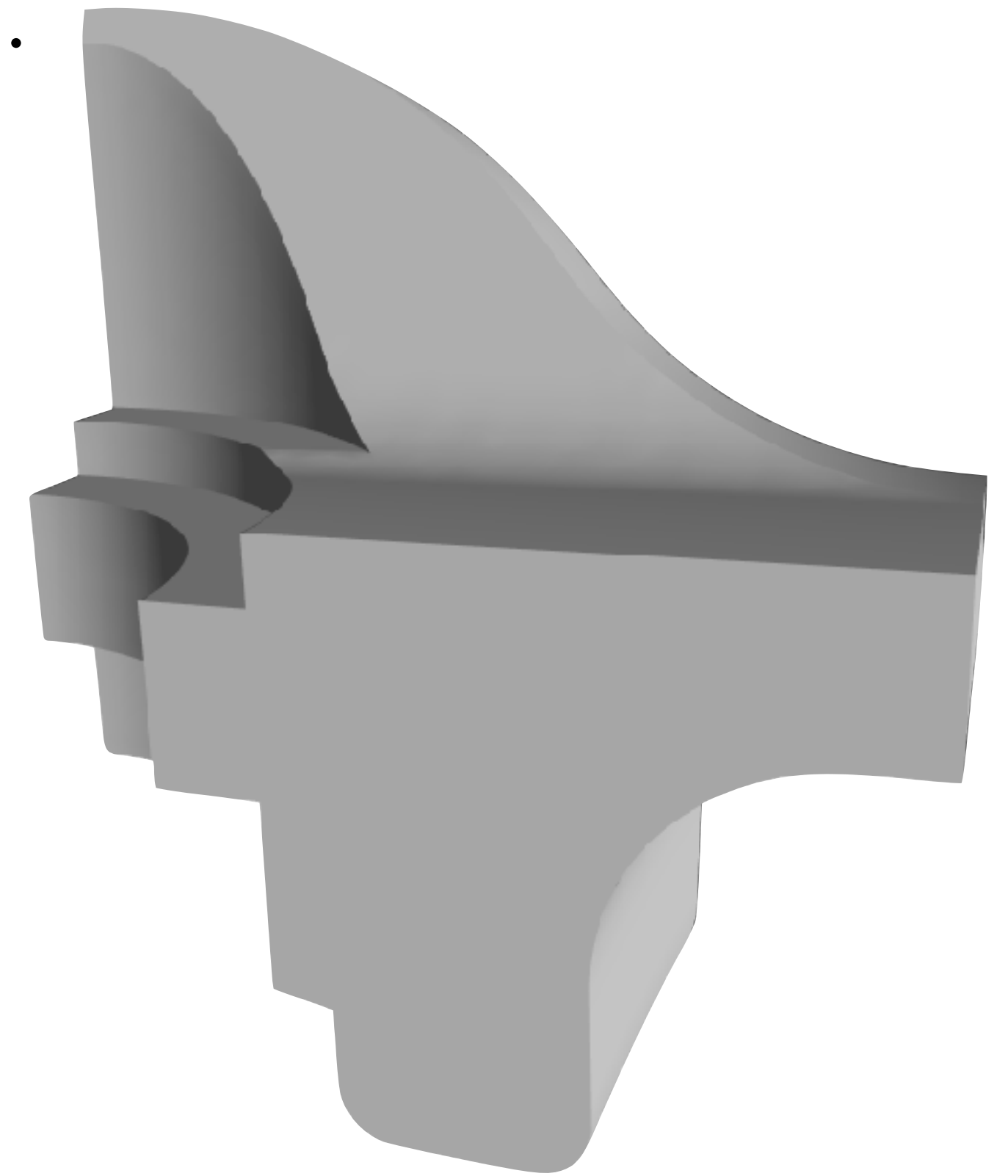
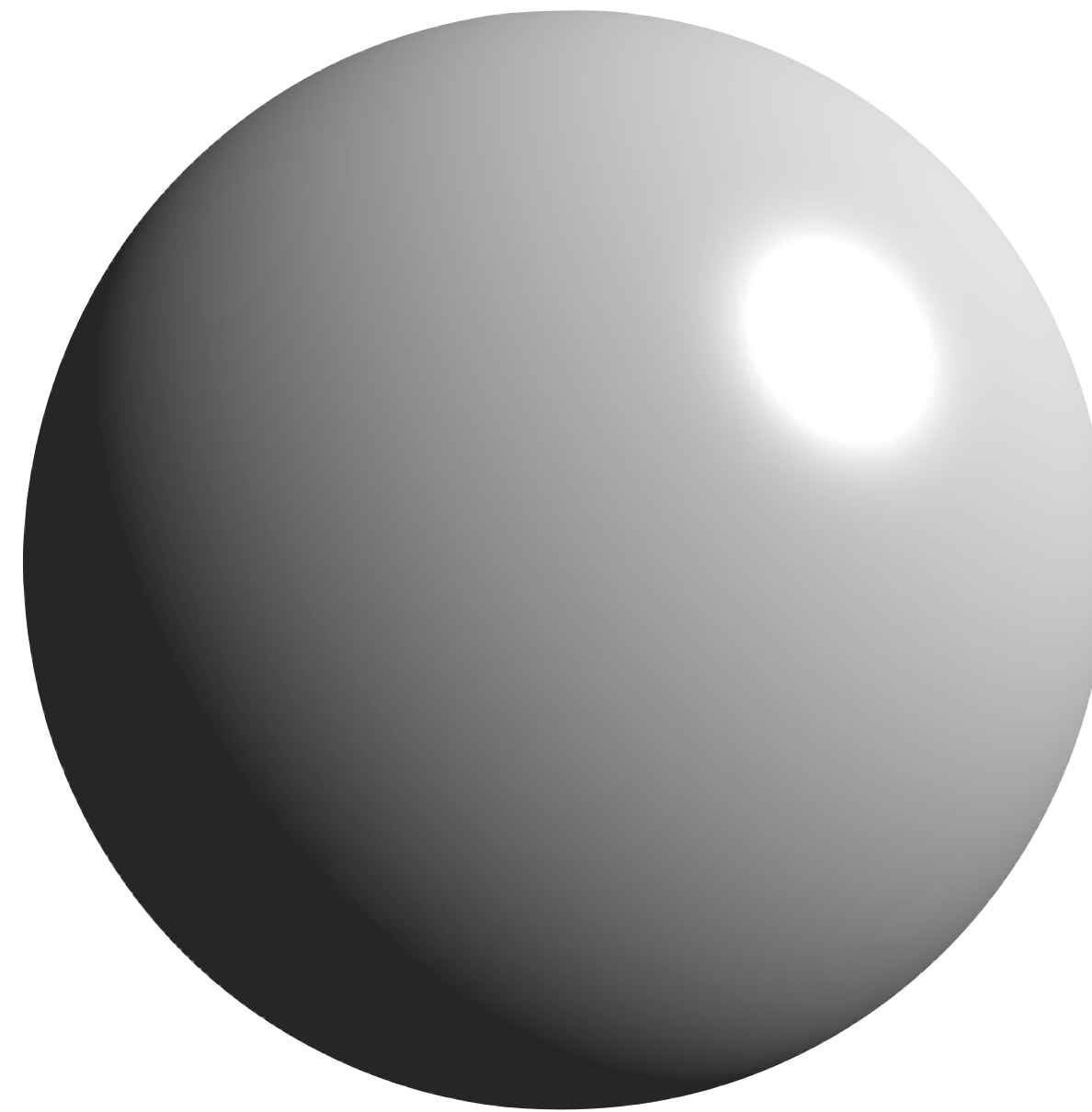
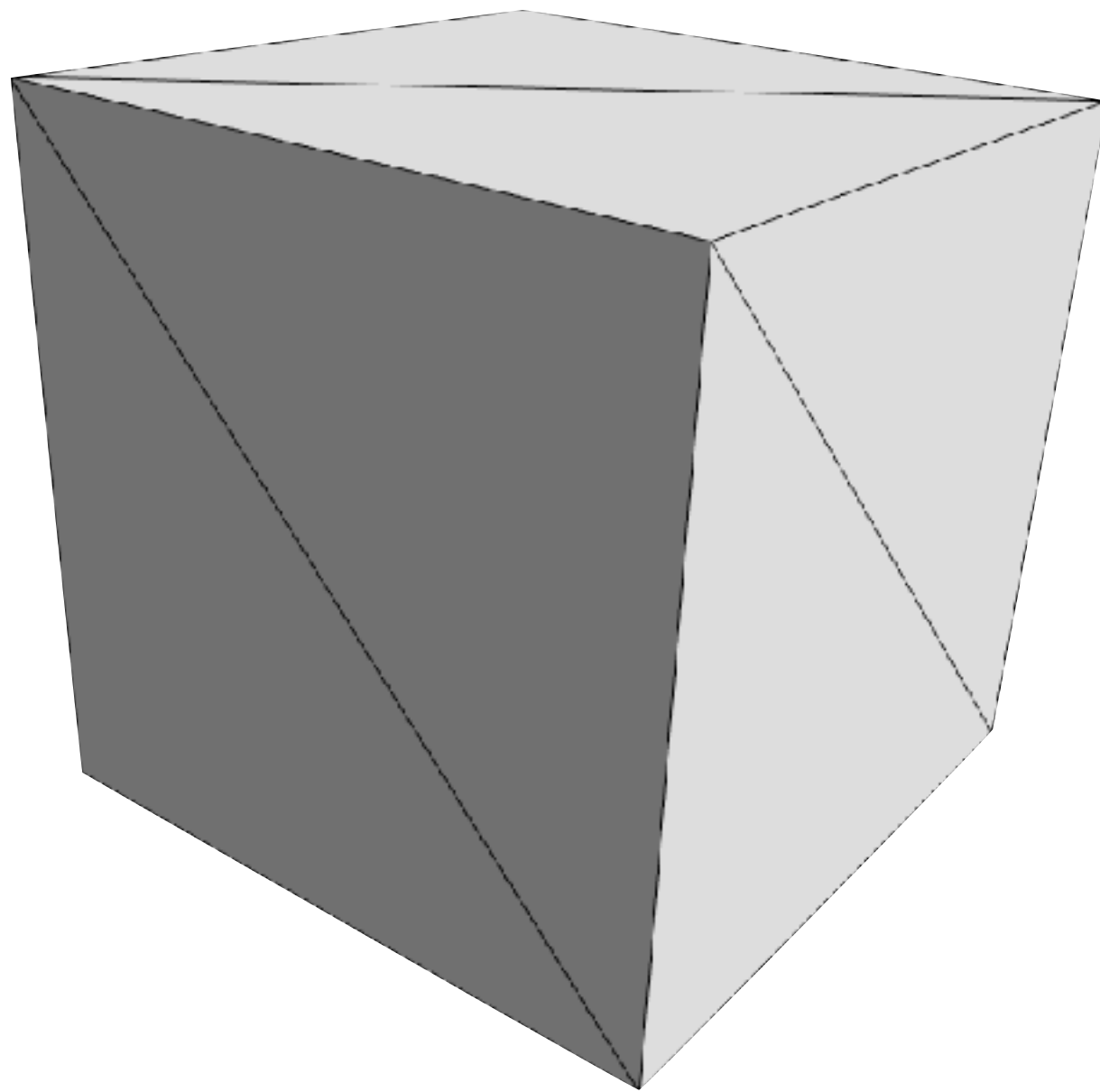


Smooth surfaces look nice.



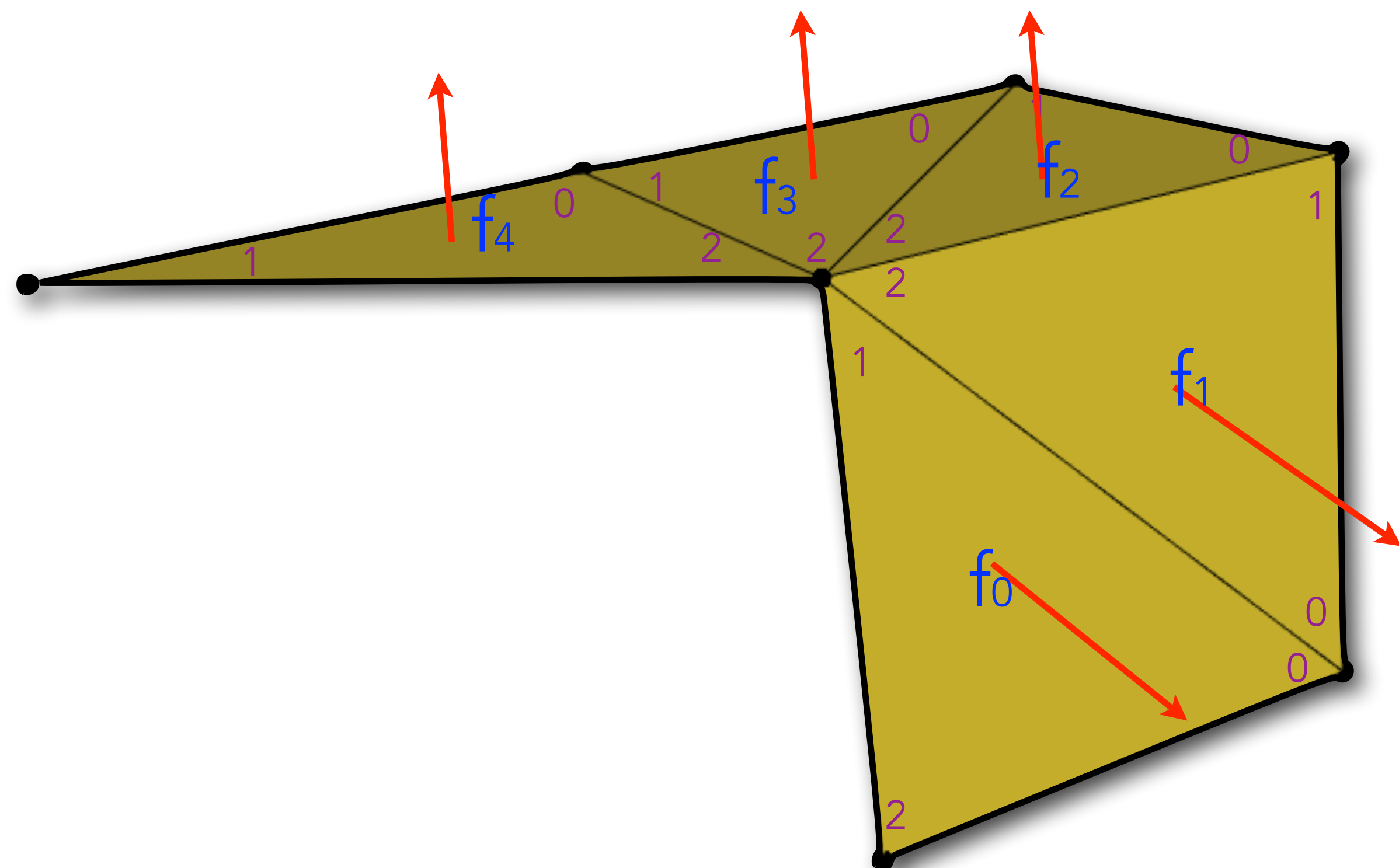
Per-corner Shading: find a nice balance

- Compute 3 separate normals for each tri (one per corner)
- Average normals with “smoothly incident neighbors,” but preserve discontinuities across sharp edges.



Corner normals

- For each corner, average adjacent face normals if they're close enough in direction



```
corner_normals(f4*3+2) =
corner_normals(f3*3+2) =
corner_normals(f2*3+2) =
average(face_normals(f2), face_normals(f3), face_normals(f4))
corner_normals(f0*3+1) =
corner_normals(f1*3+2) =
average(face_normals(f0), face_normals(f1))
```

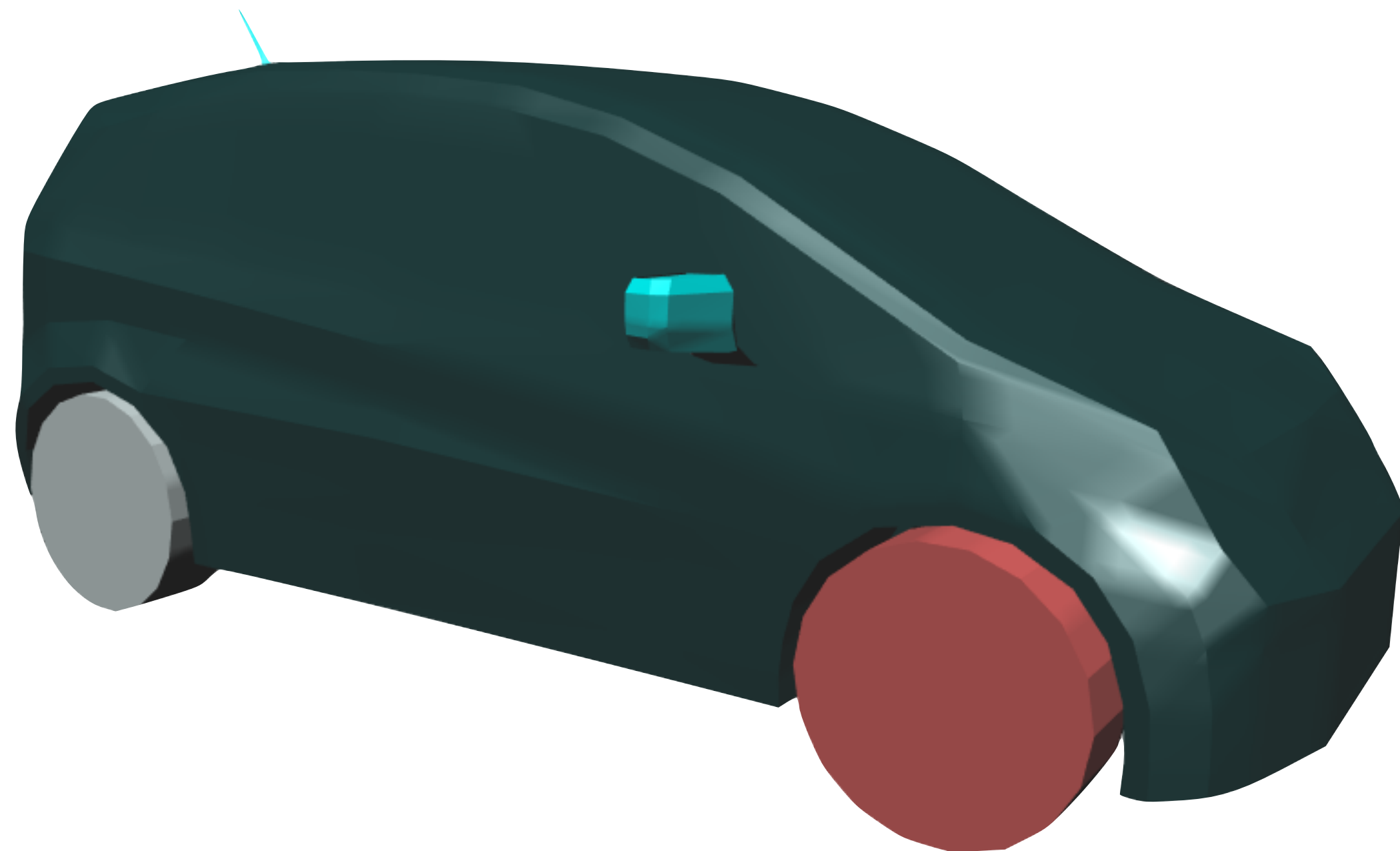
```
corner_normal(f0*3+0)
corner_normal(f0*3+1)
corner_normal(f0*3+2)
corner_normal(f1*3+0)
corner_normal(f1*3+1)
corner_normal(f1*3+2)
...
corner_normal(f4*3+0)
corner_normal(f4*3+1)
corner_normal(f4*3+2)
```

stack all corner normals
for a face sequentially
for all faces

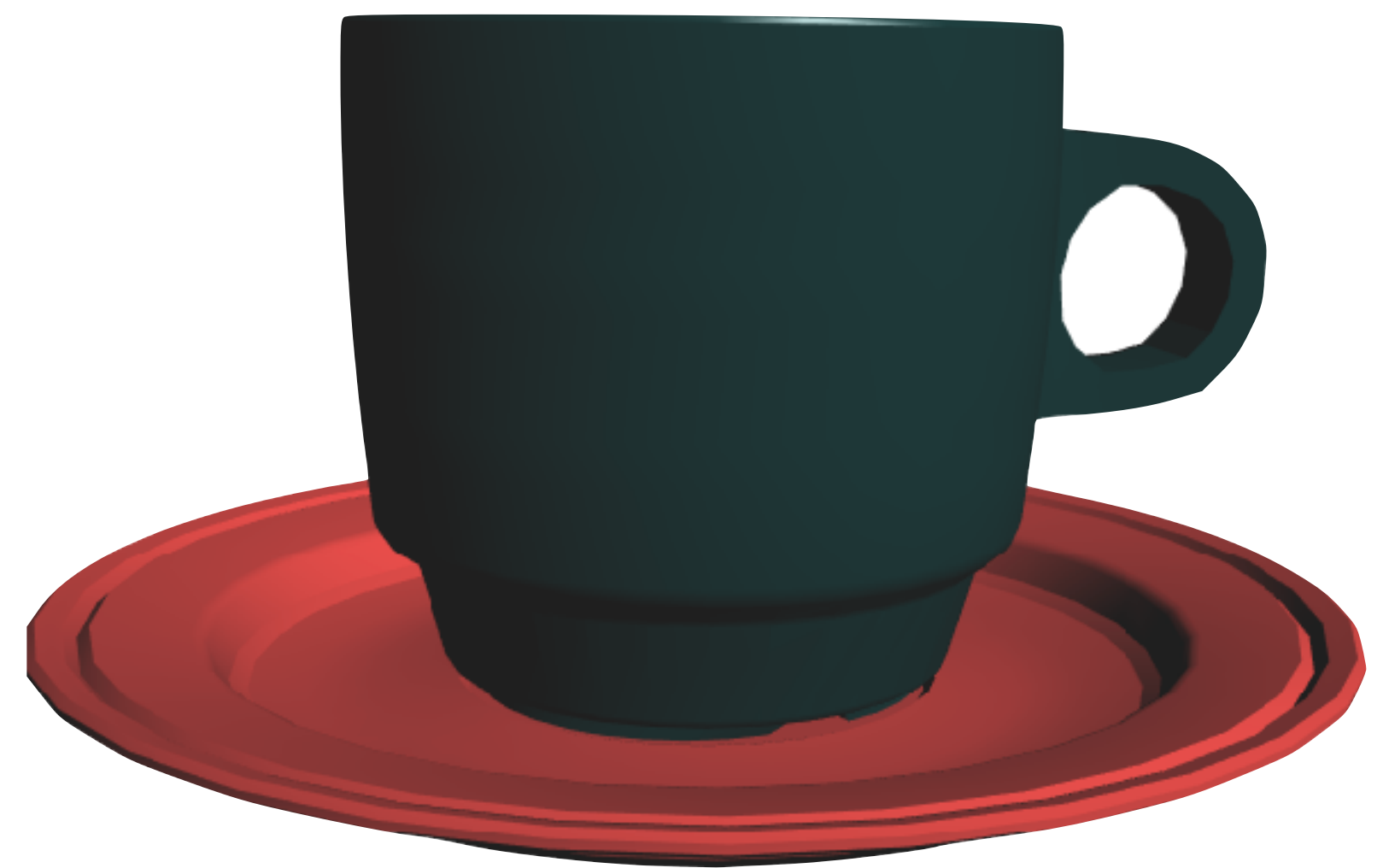
```
corner_normals(i*3+j) =
corner normal at corner j of face i (for triangle faces)
```



Connected Components

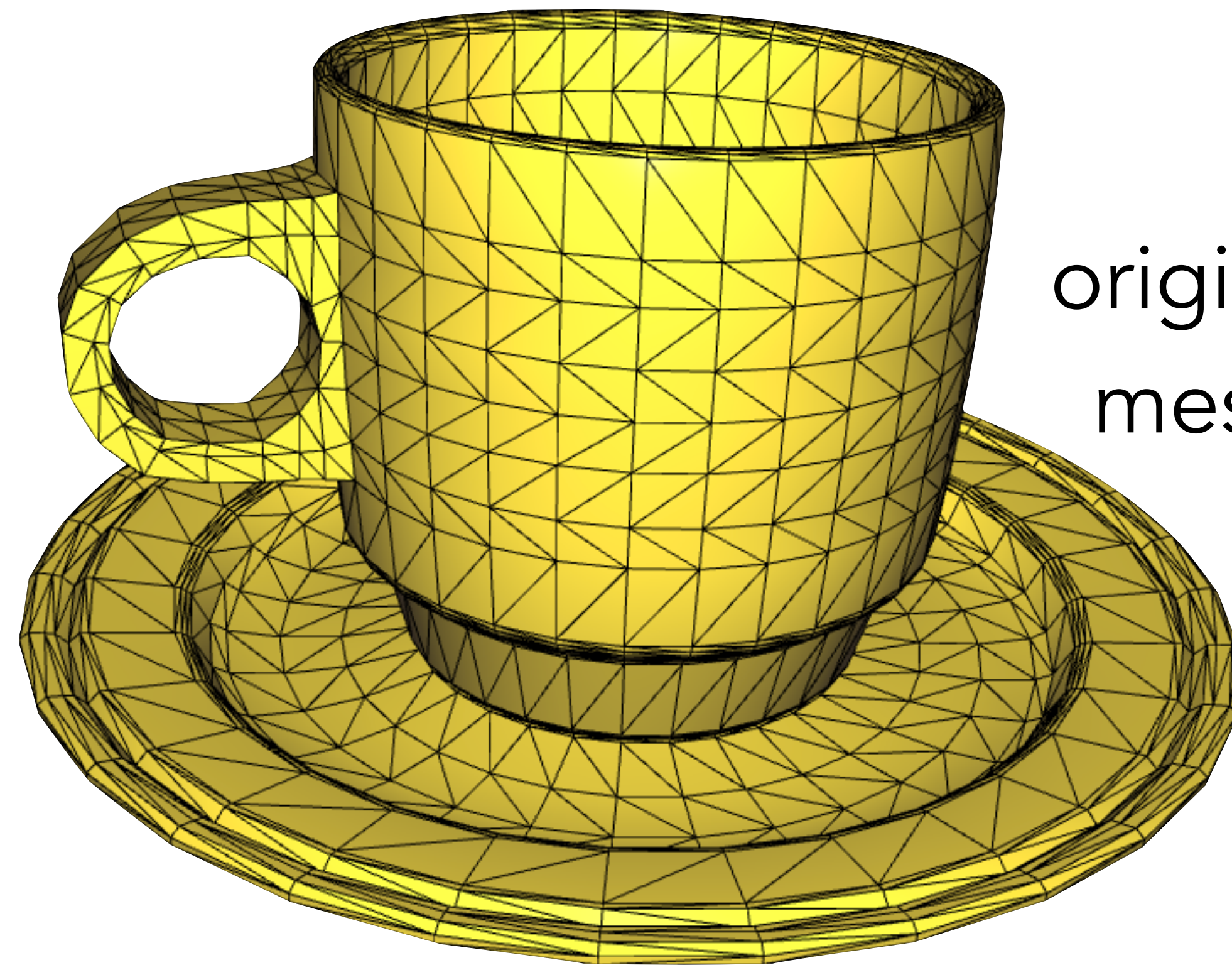


11 components

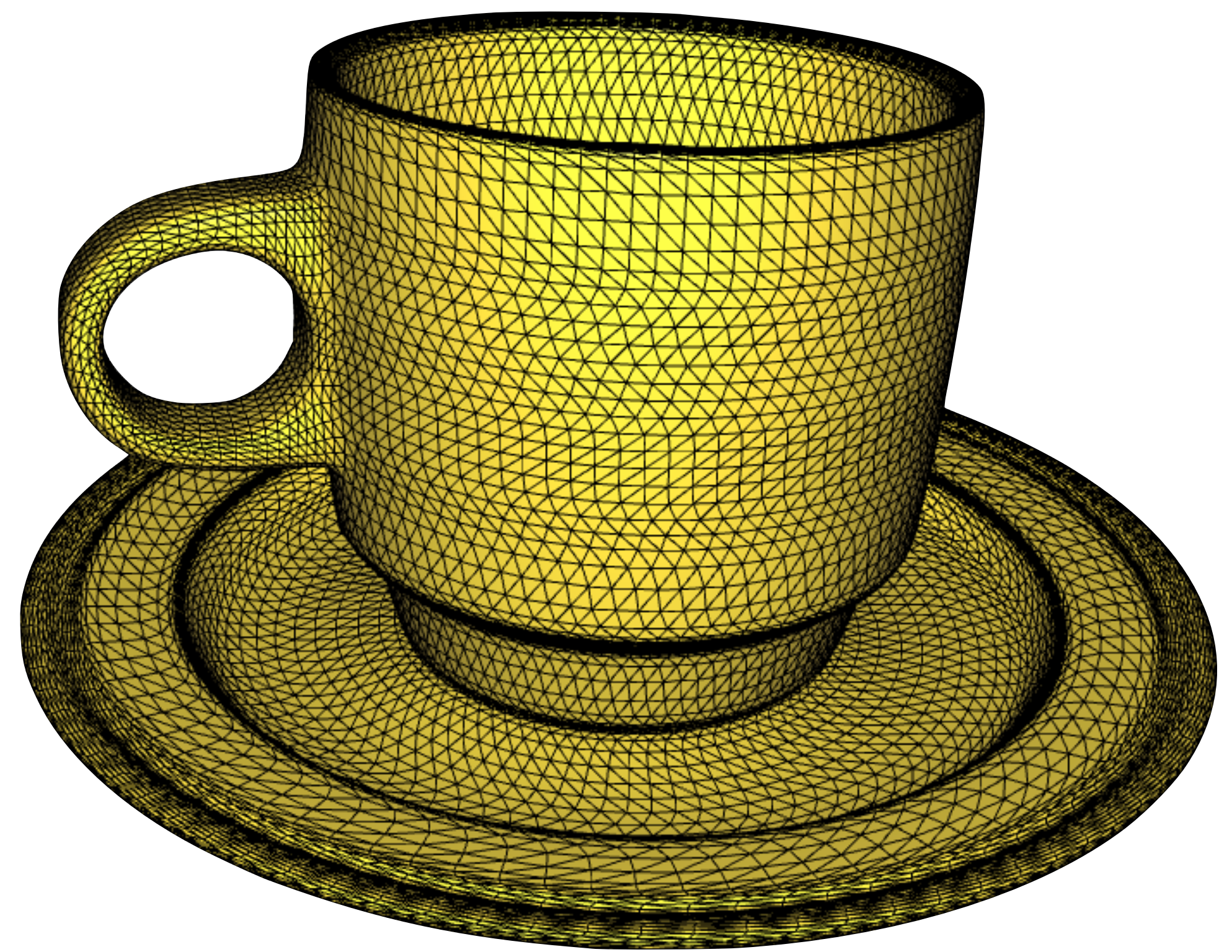


2 components

Sqrt(3) Subdivision

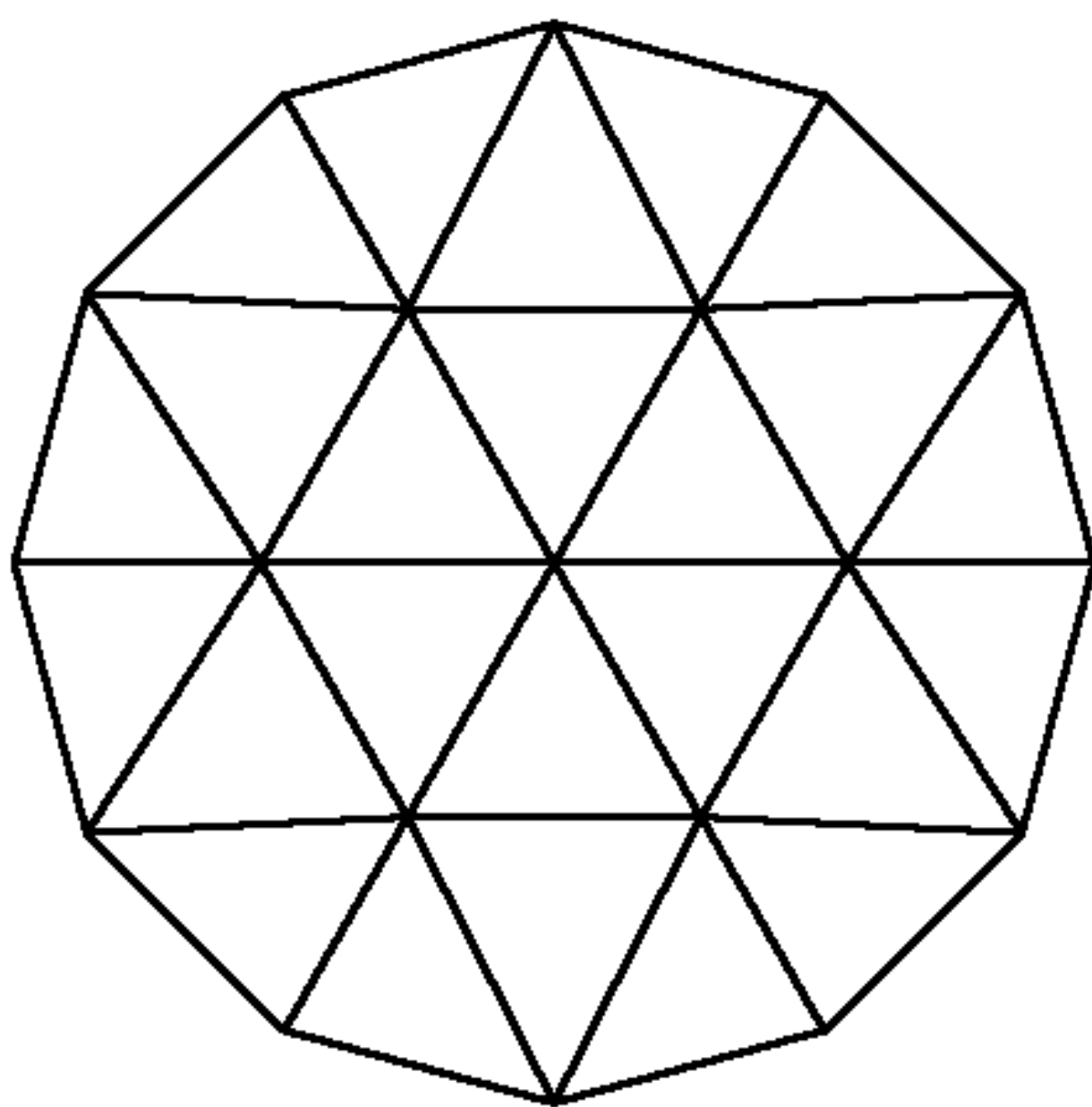


original
mesh

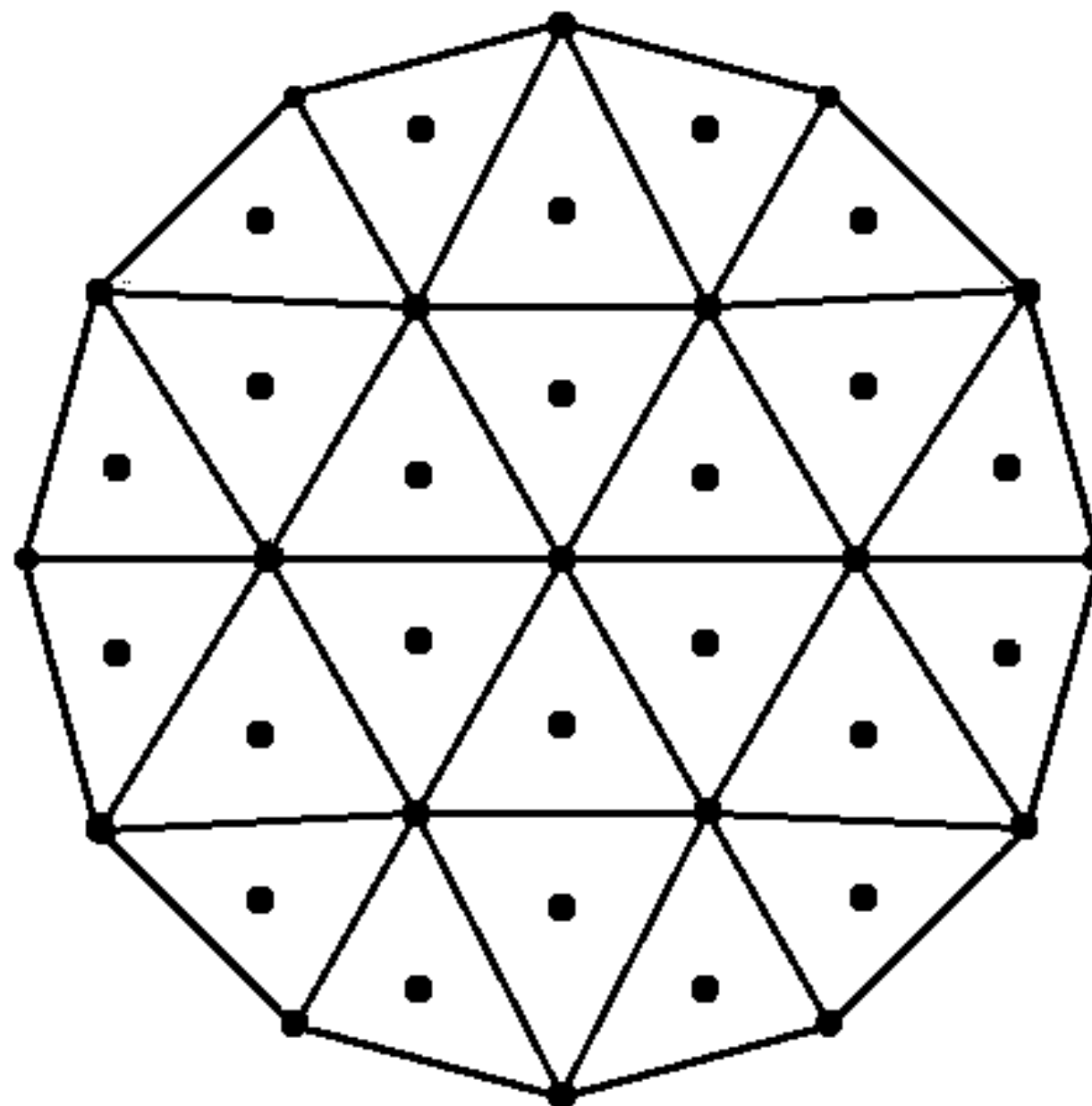


after 2 subd. steps

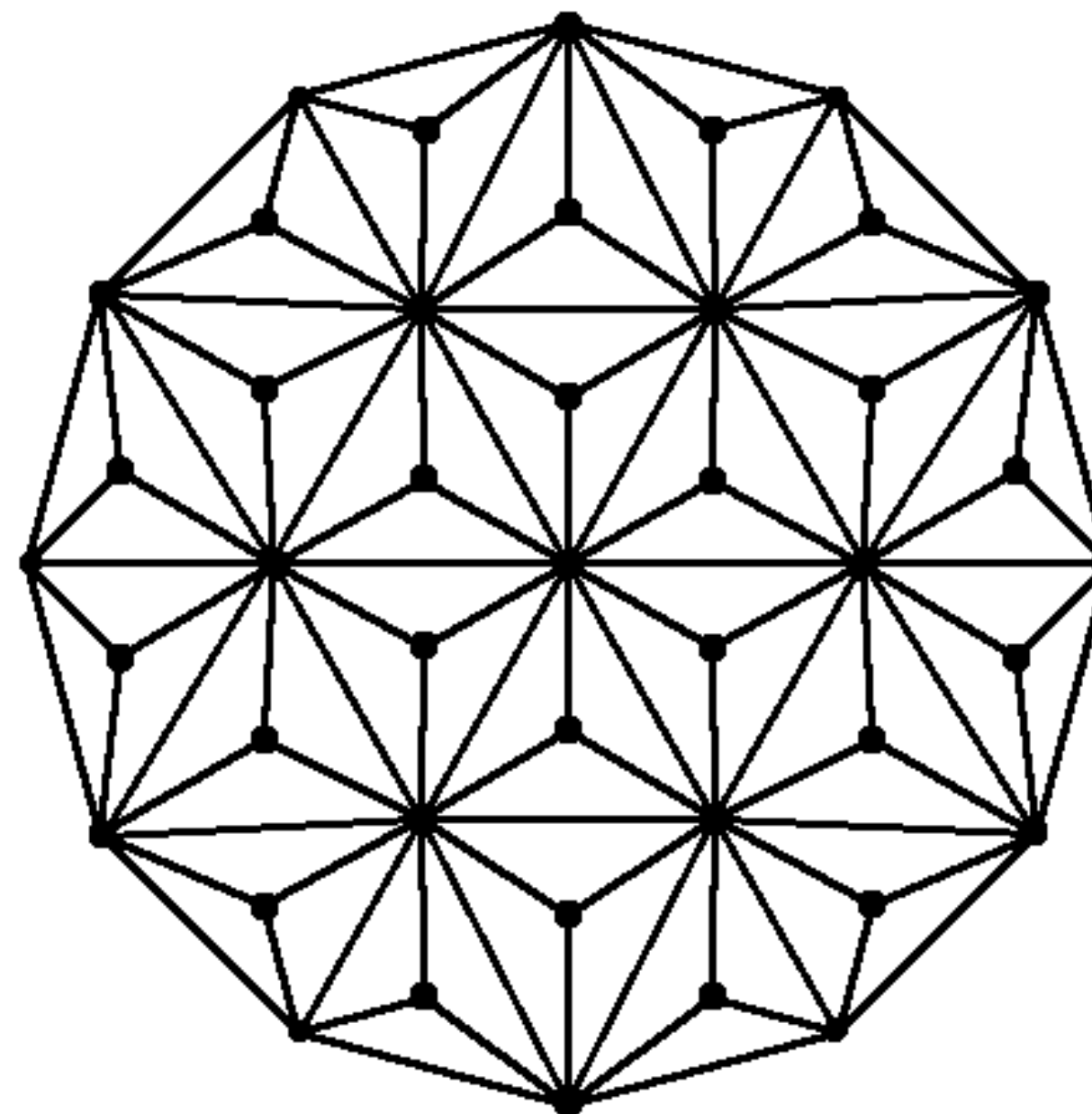
Sqrt(3) Subdivision



original mesh

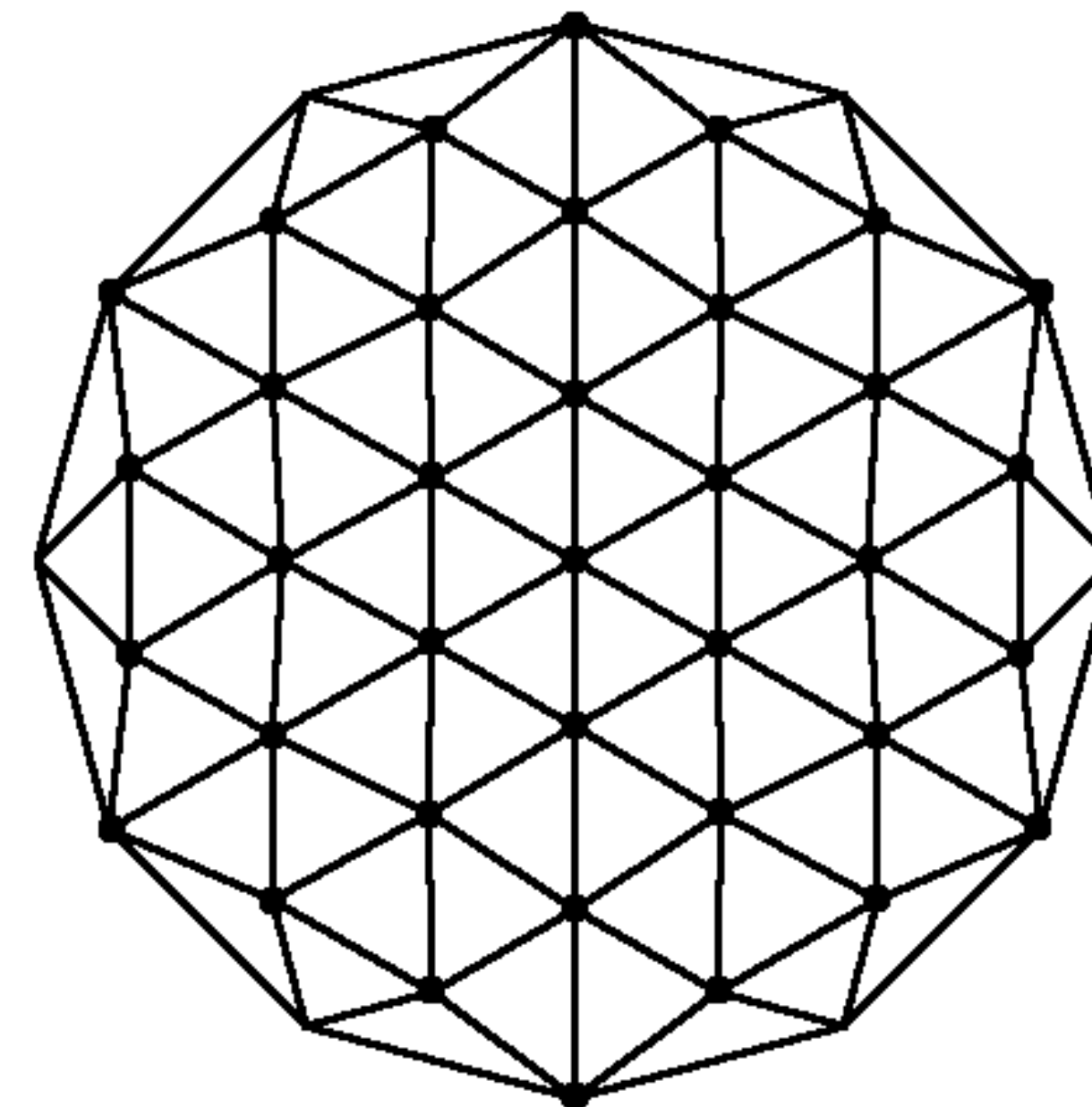


add vertices at face
midpoints



connect new vertices to
face corners

move old vertices
by averaging in
their one-ring



flip original edges

NumPy and SciPy

- NumPy is the fundamental package for scientific computing with Python. It supports matrices, vectors
 - <https://numpy.org>
- SymPy is a Python ecosystem of software for mathematics, science, and engineering. In particular it contains numerical solvers, and sparse matrices.
 - <https://www.scipy.org>

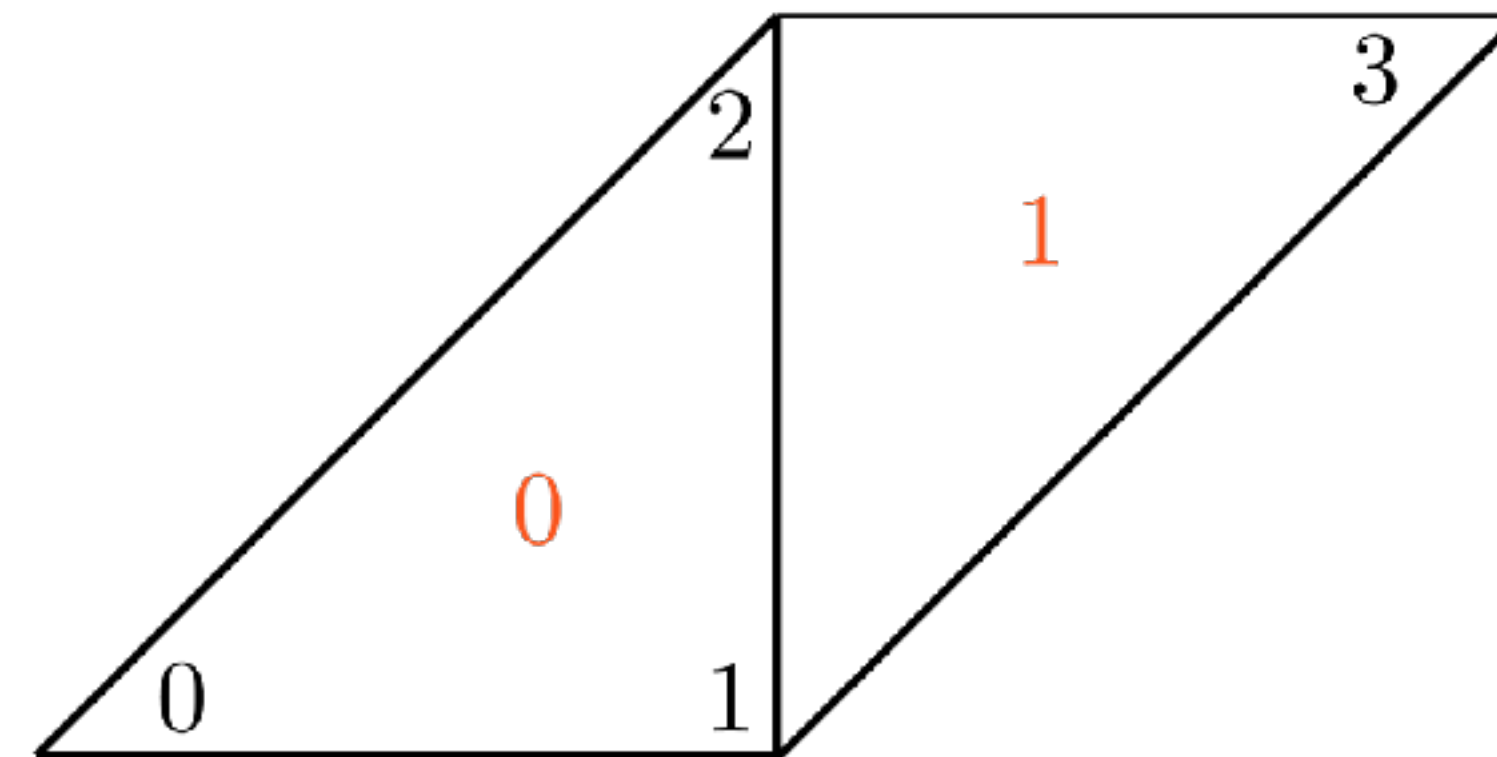
Mesh Representation with NumPy

An numpy matrix

```
numpy.array(..., dtype=...)
```

$$V = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 1 & 0 \end{pmatrix}$$

$$F = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 3 & 2 \end{pmatrix}$$



- Everything needed to display the mesh

```
V = numpy.array(..., dtype=numpy.double)
F = numpy.array(..., dtype=numpy.int32)
```

NumPy: Initialization and Element Access

Initialization

<code>m1 = numpy.zeros((rows, cols))</code>	<code>#numpy.double numpy matrix</code>
<code>v1 = numpy.zeros(rows)</code>	<code>#numpy.double numpy vector</code>
<code>v2 = numpy.array([x, y, z, w])</code>	<code>#initialize with default values</code>
<code>m2 = numpy.zeros((rows, cols), dtype=numpy.int64)</code>	<code>#numpy.int64 numpy matrix</code>

Element Access

<code>matrix[i,j]</code> <code>vector[i]</code>
--

NumPy Quickstart

- Most element-wise and matrix operations supported
 - element-wise addition, subtraction, multiplication
 - multiplication by scalar
 - matrix-matrix multiplication
 - transposition, adjoint
 - norm, normalization
 - dot product
 - cross product (3d vectors only)
 - sub-matrix manipulation
 - trigonometric functions
 -

See <https://numpy.org/doc/stable/user/quickstart.html>

Python Libigl

- <https://github.com/libigl/libigl.git>
- <https://libigl.github.io/libigl-python-bindings/>
- Open source C++/Python library for geometry processing
 - No complex data types, only numpy

```
V, F = igl.read_triangle_mesh("../shared/cube.off")
```

The meshplot Viewer

- Very basic UI options
- Rotate (left click and drag)
Translate (right click and drag)
Zoom (scroll)
- Texture/normals
- Some material/color options
- Integrated in Jupyter
- <https://skoch9.github.io/meshplot/>

```
mp.plot(v, f)
```



"Hello Viewer"

```
mp.plot(v, f)
```

```
import igl
import meshplot

V, F = igl.read_triangle_mesh("bunny.off")
meshplot.plot(V, F)
```



Python Setup for Assignment 1

- Anaconda is a package manager used in particular for Python
- For the course you will need some libraries
- Anaconda (or Miniconda) can be installed from <https://docs.conda.io/en/latest/miniconda.html>
- We suggest to install them through conda

Conda Setup

- In a terminal (or conda terminal) type

```
conda create -n gp
```

```
conda activate gp
```

```
conda config --add channels conda-forge
```

```
conda install numpy
```

```
conda install scipy
```

```
conda install igl
```

```
conda install meshplot
```

```
conda install notebook
```

Creates a new virtual environment called gp

Activates the environment, all changes will affect only the gp environment

Add a new channel, all libraries are on conda-forge

Installs the necessary packages

