Shape Modeling and Geometry Processing

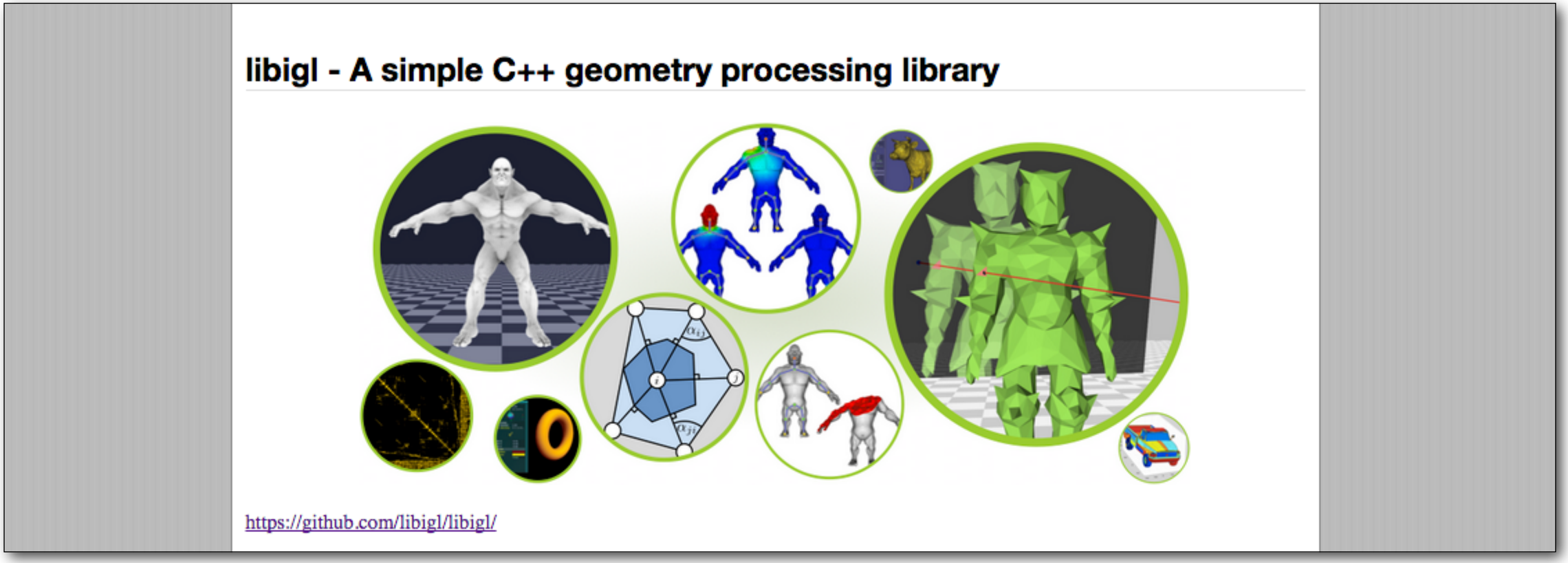
Exercise 1: libigl "Hello World"

Julian Panetta — fjp234@nyu.edu

Acknowledgments: Olga Diamanti

Exercise 1 - libigl “Hello World!”

- Experiment with the geometry processing library

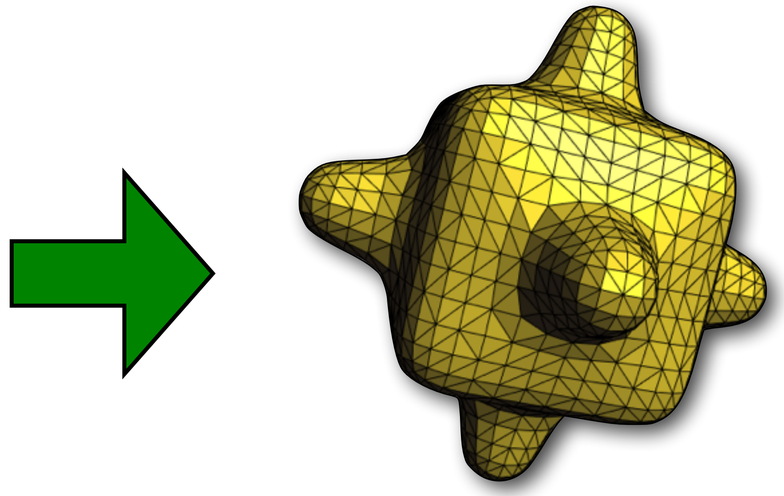


Exercise 1 - libigl “Hello World!”

- Read and visualize a mesh

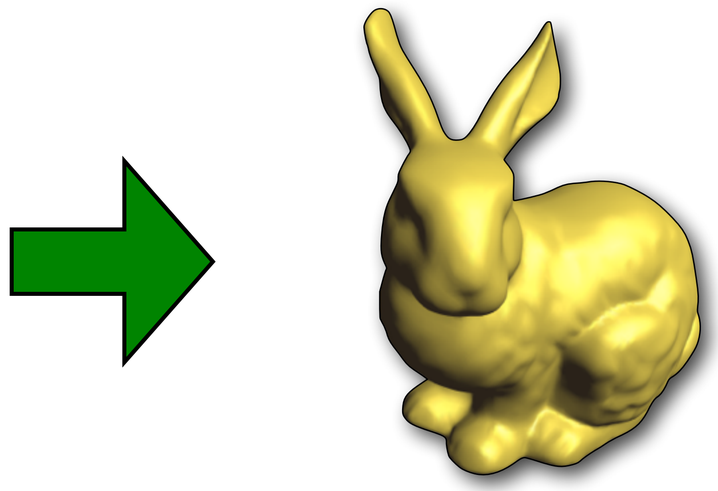
```
OFF
1250 2496 0
-2.09105 -2.09105 2.09105
-0.833333 -2.23958 2.23958
0.833333 -2.23958 2.23958
2.09105 -2.09105 2.09105
...
3 940 83 320
3 386 0 941
...
```

bumpy_cube.off



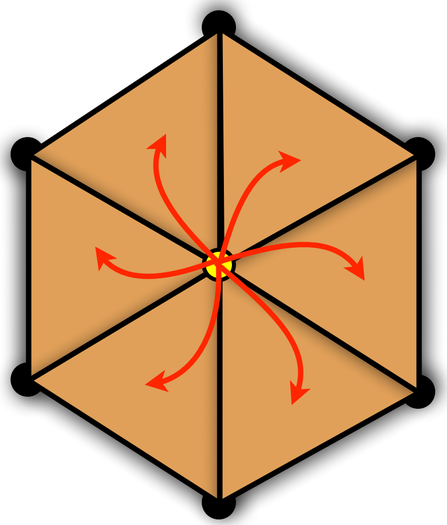
```
# Wavefront OBJ file
v 30.50959969 12.17459898 -15.84426970
v 30.49857998 11.87718728 -15.40759913
v 30.53679943 12.68500615 -14.82485356
v 30.67168999 11.71161003 -15.78844530
...
f  633/16706 11590/29979 4339/16704
f  11590/3161 633/16716 19901/16699
...
```

bunny.obj

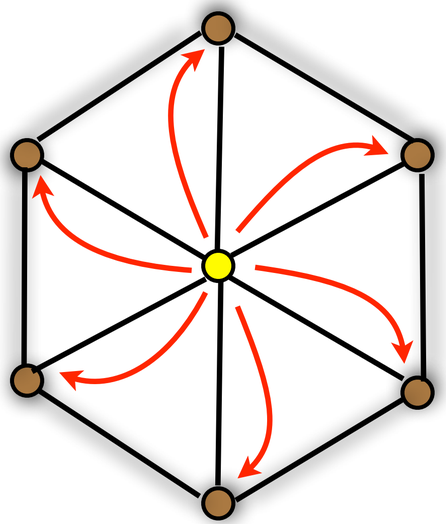


Exercise 1 - libigl “Hello World!”

- Perform simple neighborhood calculations



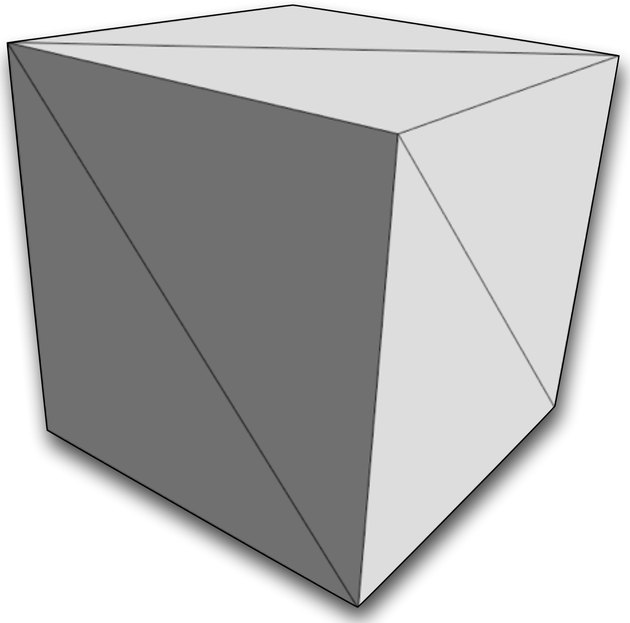
vertex-to-face



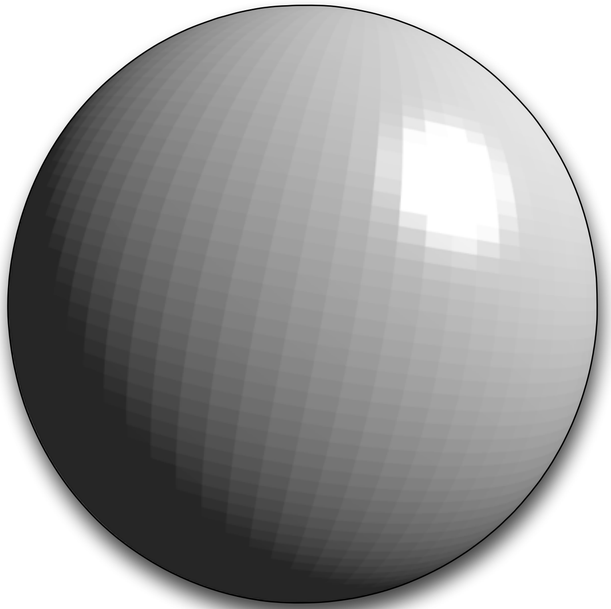
vertex-to-vertex

Exercise 1 - libigl “Hello World!”

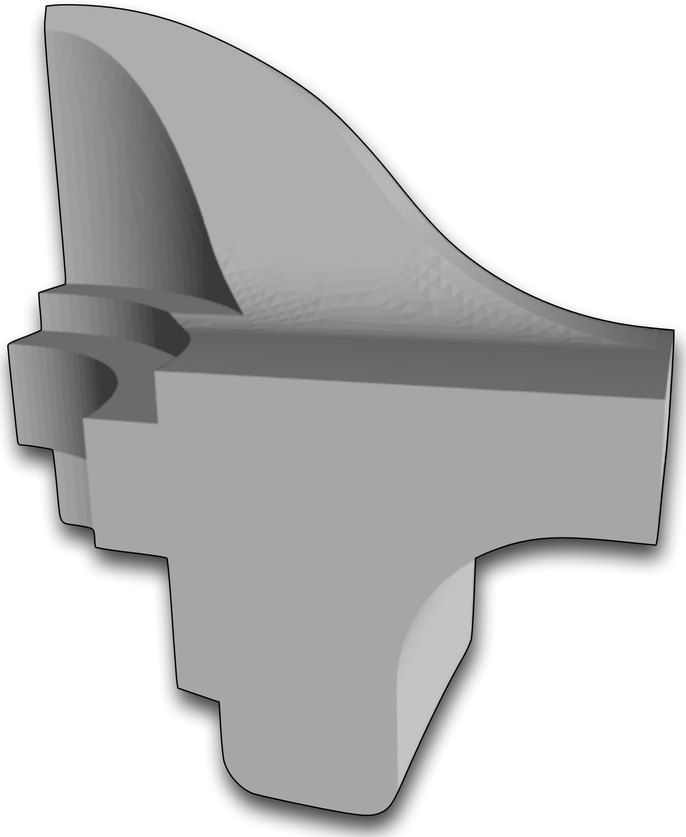
- Flat shading
  - Compute one normal per polygon



Creased surfaces render well.

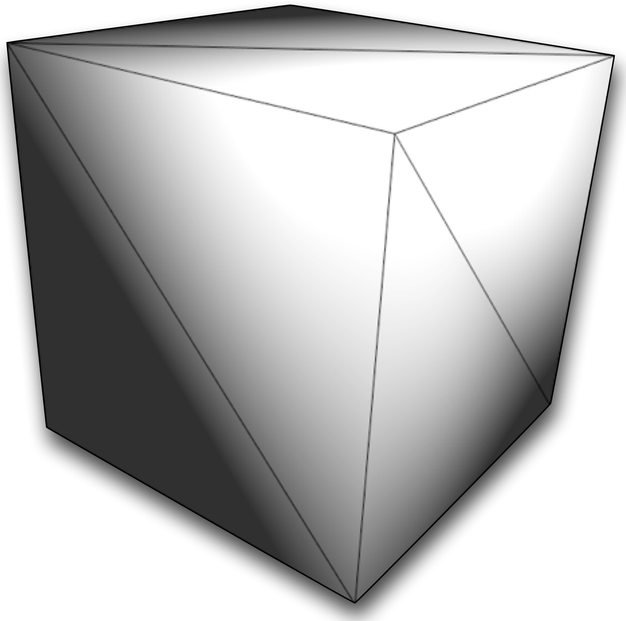


But discontinuous normals lead to poor results for smooth surfaces.



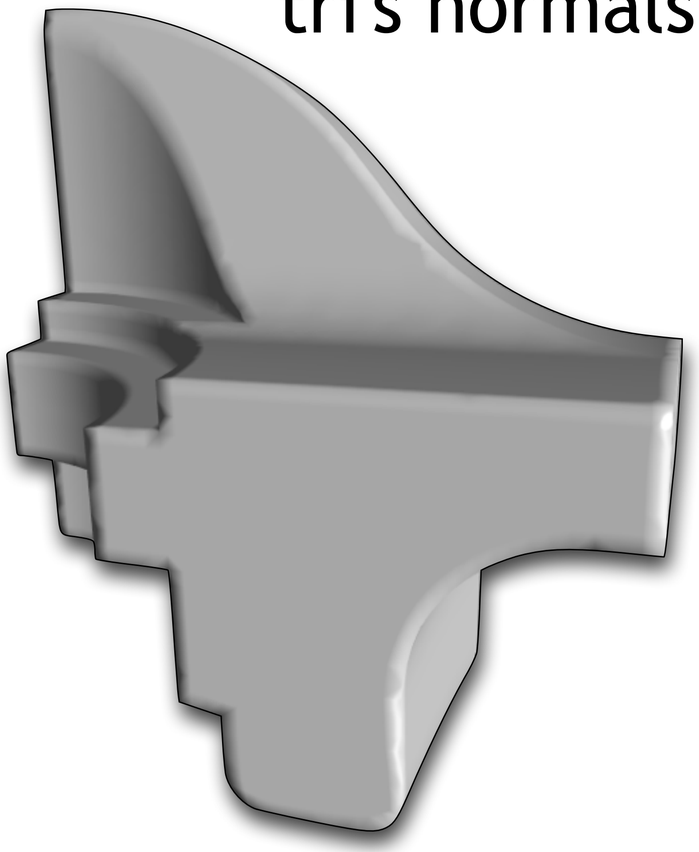
Exercise 1 - libigl “Hello World!”

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



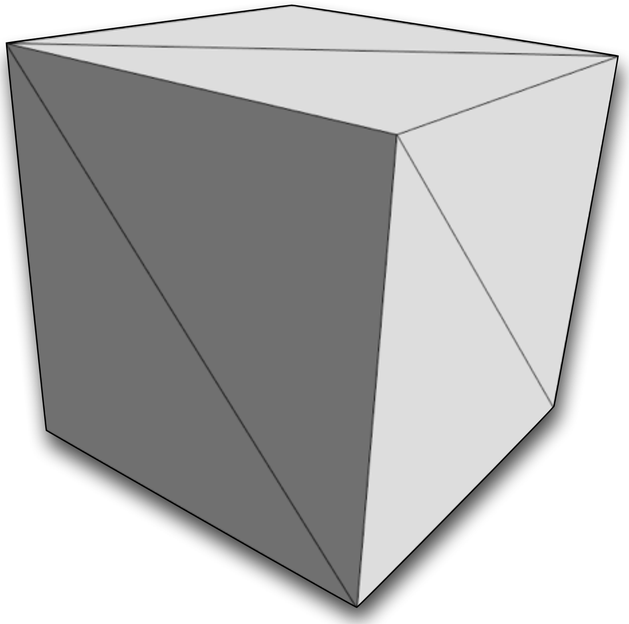
Creased surfaces look strange and burry.

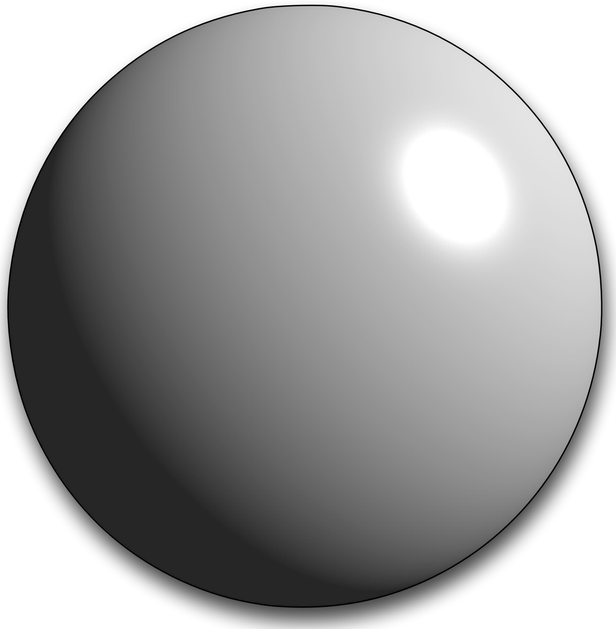
Smooth surfaces look nice.

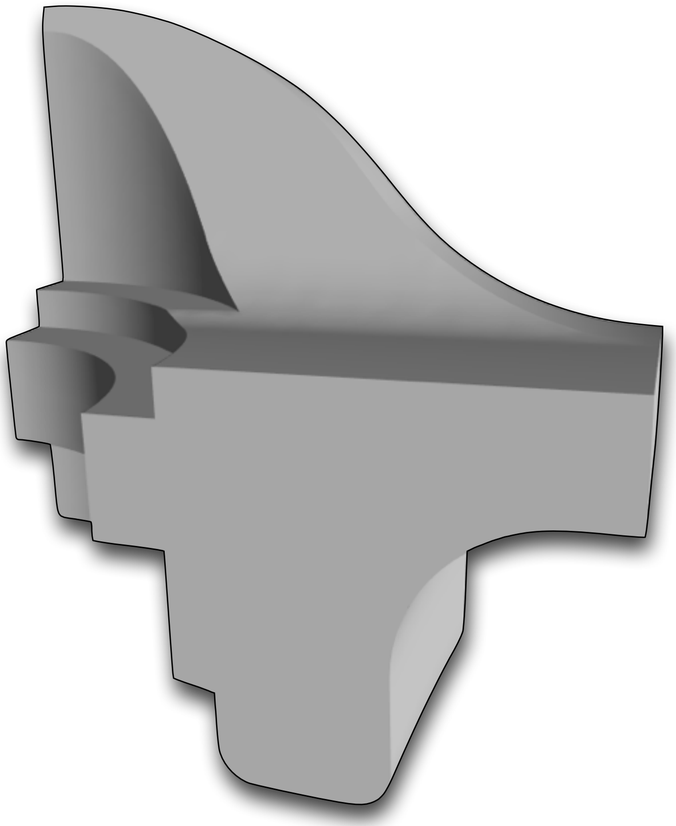


Exercise 1 - libigl “Hello World!”

- Per-corner Shading: find a nice balance
  - Compute three separate normals for each triangle (one per corner)
  - Average normals with “smoothly incident neighbors,” but preserve discontinuities across sharp edges.

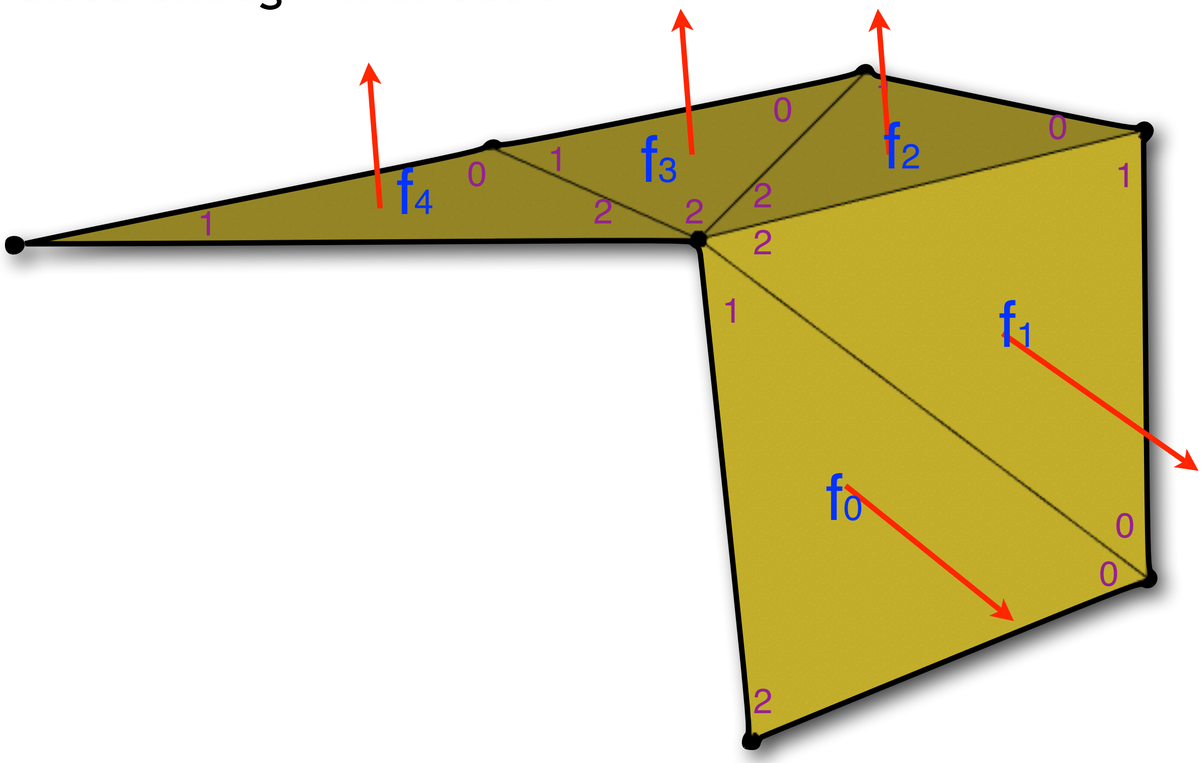






Corner normals

- 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) )
```

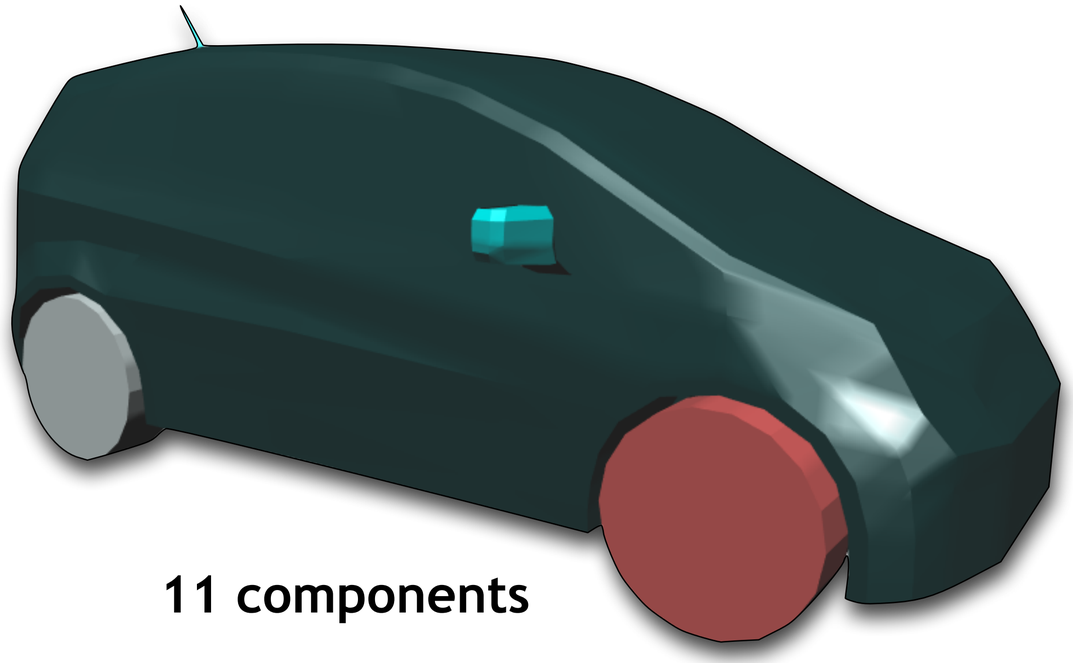
```
corner_normal(f0*3+0)
corner_normal(f0*3+1)
corner_normal(f0*3+2)
corner_normal(f1*3+0)
corner_normal(f1*3+1)
corner_normal(f1*3+2)
        ...
corner_normal(f4*3+0)
corner_normal(f4*3+1)
corner_normal(f4*3+2)

corner_normals array
```

stack all corner normals for a face sequentially for all faces

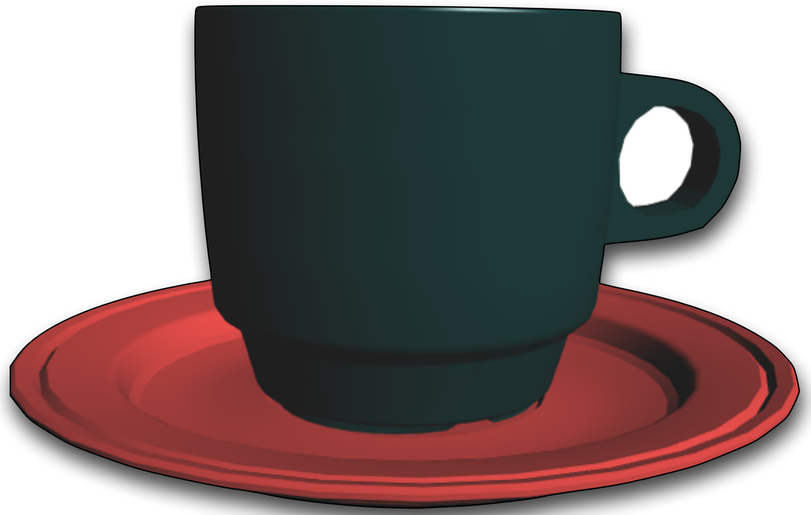
`corner_normals(i*3+j) =`
corner normal at corner j of face i (for triangle faces)

Exercise 1 - libigl “Hello World!”

- Connected Components



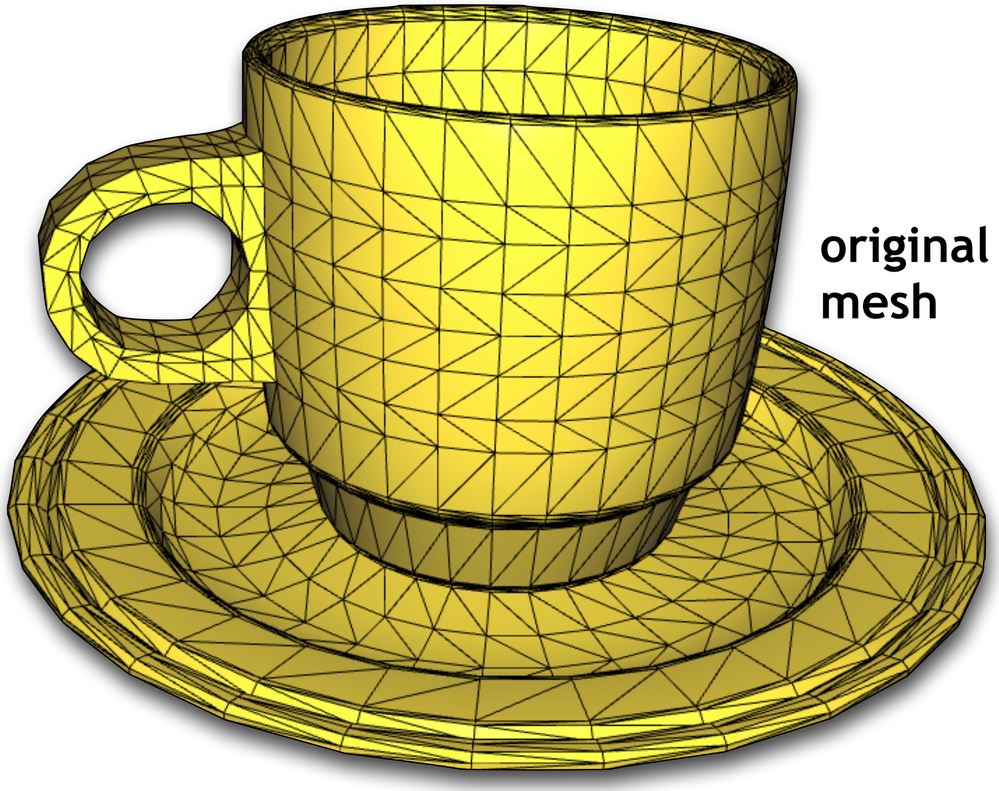
11 components



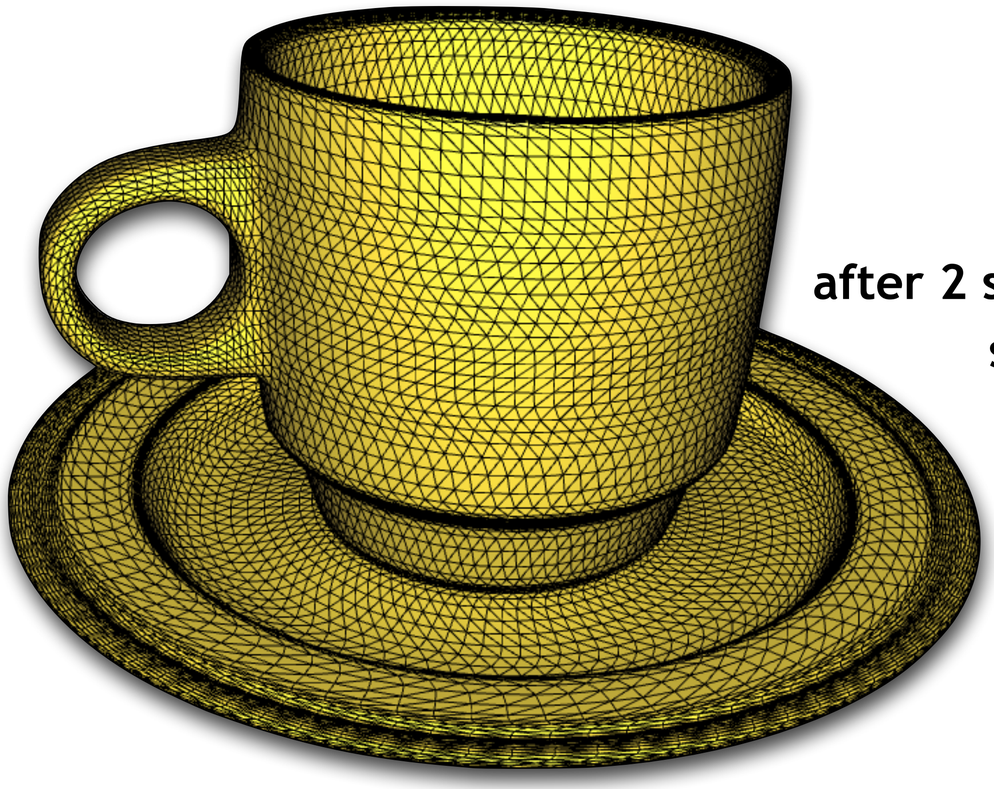
2 components

Exercise 1 - libigl “Hello World!”

- Sqrt(3) Subdivision



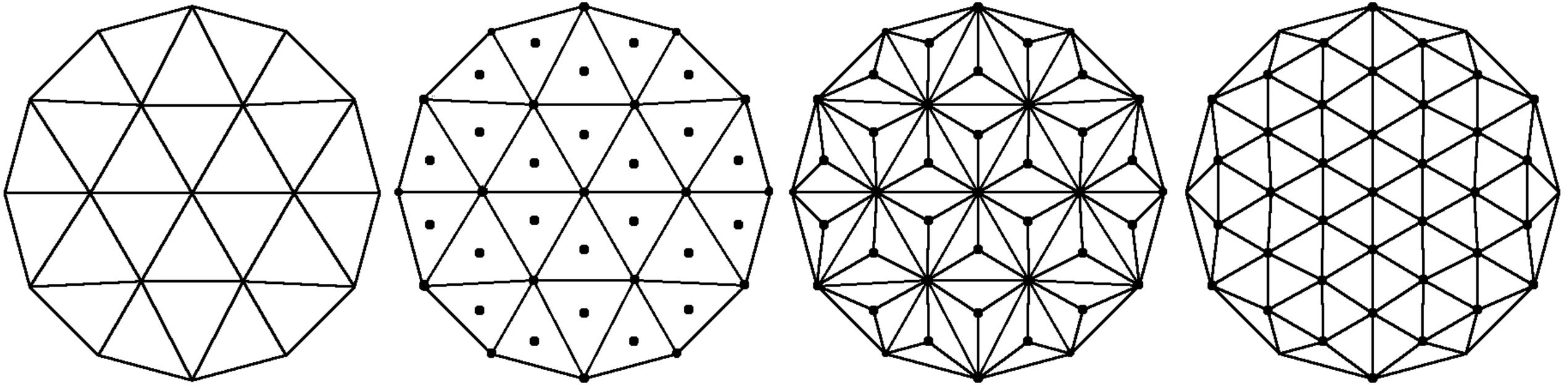
original mesh



after 2 subd. steps

Exercise 1 - libigl “Hello World!”

- Sqrt(3) Subdivision



original mesh

add vertices at face midpoints

connect new vertices to face corners

flip original edges

move old vertices by averaging in their one-ring

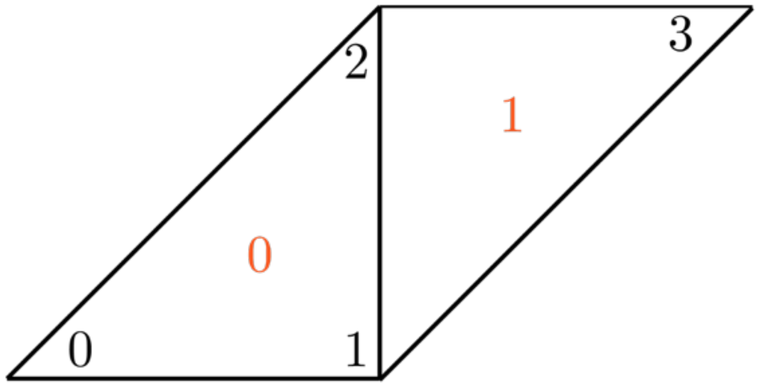
Eigen

- *Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms.*
  - *from* http://eigen.tuxfamily.org/index.php?title=Main_Page
- Header-only library
  - No compilation required!
- Tutorials:
  - http://eigen.tuxfamily.org/dox/TutorialMatrixClass.html
  - http://eigen.tuxfamily.org/dox/QuickRefPage.html

Mesh Representation with Eigen

- An Eigen matrix `Eigen::Matrix< type, #rows, #cols>`

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




```
Eigen::Matrix<double, Eigen::Dynamic, 3> V;
Eigen::Matrix<int,    Eigen::Dynamic, 3> F;
```

Eigen: Initialization and Element Access

- Constructors

```
Eigen::Vector4d  v4;              //typedef for Eigen::Matrix<double, 4, 1>
Eigen::Vector4d  v3(x, y, z, w);
Eigen::VectorXf  v5;              //typedef for Eigen::Matrix<double, Eigen::Dynamic, 1>
Eigen::Matrix4f  m1;              //typedef for Eigen::Matrix<float, 4, 4>
m1.setZero();                     //also: setOnes, setConstant, setRandom
Eigen::MatrixXf  m5;              //typedef for Eigen::Matrix<float, Eigen::Dynamic, Eigen::Dynamic>
Eigen::MatrixXf  m6(nb_rows, nb_columns);
Eigen::MatrixXd  m7 = Eigen::MatrixXd::Zero(nb_rows, nb_columns);//also: Ones, Constant, Random
Eigen::MatrixXf  m8 = Eigen::Matrix3f::Identity();
```

- Comma Initializer (row-major order)

```
Eigen::Vector3f  v1;         v1 << x, y, z;
Eigen::MatrixXd  m1(3,4);    m1 << 1,  2,  3,  4,
                                   5,  6,  7,  8,
                                   9, 10, 11, 12;
```

- Element Access

```
matrix(i,j)       vector(i)        vector.x()
                  vector[i]        vector.y()
                                   vector.z()
```

with range checking

```
vector.coeff(i)           matrix.coeff(i,j)
vector.coeffRef(i)        matrix.coeffRef(i,j)
```

without range checking

Eigen 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 http://eigen.tuxfamily.org/dox/QuickRefPage.html

Libigl

- https://github.com/libigl/libigl.git

- *Open source C++ library for geometry processing*
  - Minimal dependencies
  - Header-only
  - No complex data types

```
Eigen::MatrixXd V;
Eigen::MatrixXi F;
```

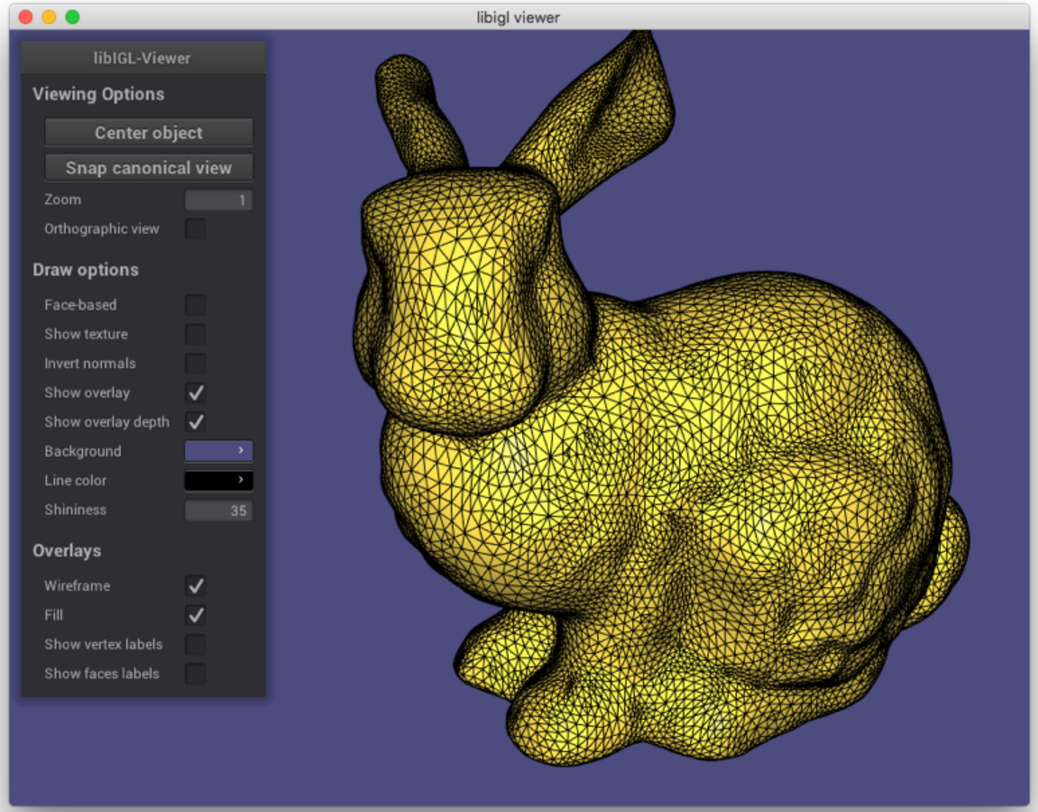
```
igl::readOFF("../shared/cube.off",
                V, F);
```

Libigl

- Dependencies: all included as submodules!
  - Eigen, GLFW, NanoGUI, etc.
- Tutorials explaining the main functionality
  - Compilation instructions: see assignment sheet
- May need to pull changes periodically!

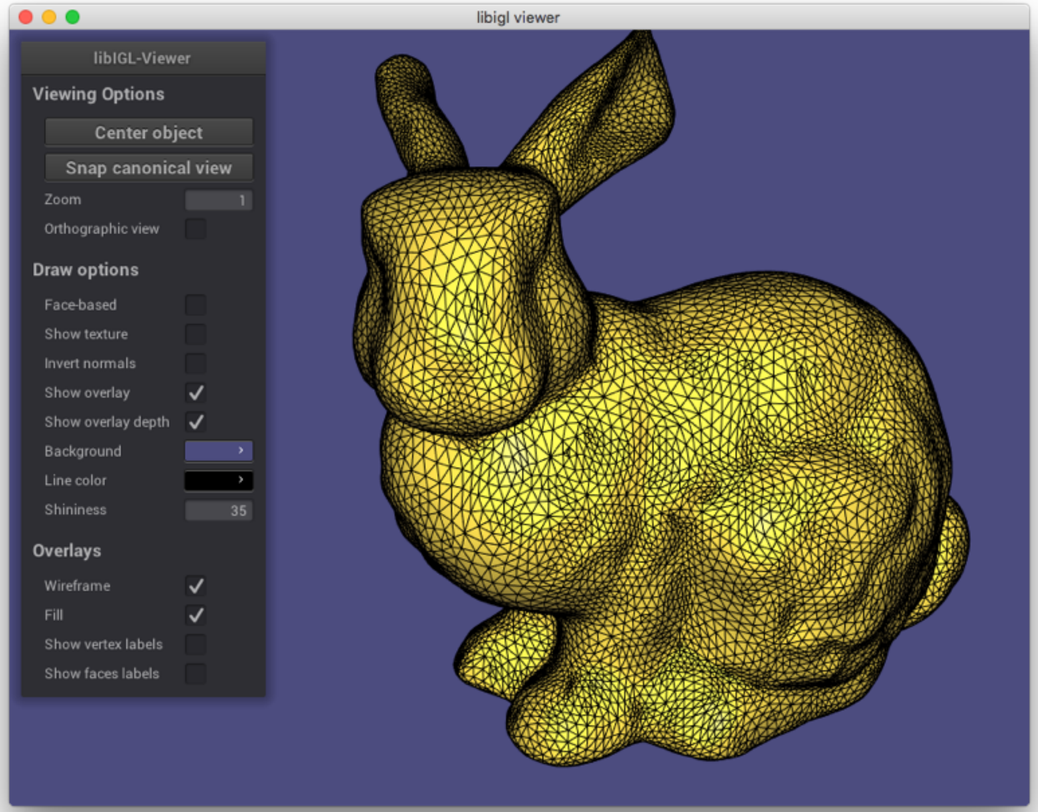
The libigl Viewer

- Display mesh
- Very basic UI options
  - Rotate (left click and drag)
    Translate (right click and drag)
    Zoom (scroll/‘a’-’s’)
  - Texture/normals
  - Some material/color options
  - Show vertex/face id
- Other options available in code



“Hello Viewer”

```
#include <igl/readOFF.h>
#include <igl/viewer/Viewer.h>

Eigen::MatrixXd V;
Eigen::MatrixXi F;

int main(int argc, char *argv[])
{
  // Load a mesh in OFF format
  igl::readOFF(“bunny.off”, V, F);

  // Plot the mesh
  igl::viewer::Viewer viewer;
  viewer.data.set_mesh(V, F);
  viewer.launch();
}
```



Adding Functionality to the Viewer

- Custom callbacks for keyboard/mouse interactions supported
  - See tutorial 103_Events
- Also supported: Face/Vertex Colors, Overlays (points/lines)
  - See tutorials 104_Colors, 105_Overlays
- Read $LIBIGL_ROOT/tutorial/tutorial.html

CMake Project for Assignment 1

- Compiles a single main.cpp launching the viewer
  - Assignment tasks are to be implemented as key interactions
  - ‘1’ - ‘2’ : neighborhood relations
  - ‘3’ - ‘5’ : shading
  - ‘6’: connected components
  - ‘7’: subdivision
- Compile

```
mkdir build;
cd build;
cmake -DCMAKE_BUILD_TYPE=Release ../; make
```

- Run

```
./ex1 bin <some mesh file>
```

Need Help?

- GitHub issue tracker: please direct questions here first!
- Recitation: Wednesday 3:00PM-4:00PM, Location TBA
- Mailing list: csci_ga_3033_018_sp17@cs.nyu.edu
- Office hours
  - ***Thursday, 4:00 PM- 5:00PM, 60 First Ave, Room 522 (Temporary)***
  - ***Also by appointment; send request to fjp234@nyu.edu***
- Bug reports/suggestions also welcome!