Introduction to OpenMesh (and Exercise 1)

Kyle Olszewski

OpenMesh

- From ACG at RWTH Aachen
- C++ library
- Implements **half-edge** data structure
- Integrated **basic geometric operations**
- 3-D model file reader/writer

Why OpenMesh ?

Flexible

- Random access to vertices, edges, and faces

- Arbitrary scalar types

- Arrays or lists as underlying kernels

Efficient in space and time

- Dynamic memory management for array-based meshes (not in CGAL)

- Extendable to specialized kernels for non-manifold meshes (not in CGAL)

Integrated geometric operations

```
OpenMesh::Vec3f x,y,n,crossproductXY;

...
l = (x-y).length();


n = x.normalize();
scalarProductXY = (x | y);
crossProductXY = x % y;

...
```

Mesh definition

```
#include <OpenMesh/Core/IO/MeshIO.hh>
#include <OpenMesh/Core/Mesh/Types/TriMesh_ArrayKernelT.hh>

typedef Openmesh::TriMesh_ArrayKernelT<>  Mesh;
```

name space

mesh type:
– triangle mesh
– array kernel
– default traits

Loading and writing a mesh

```
Mesh * myMesh;

OpenMesh::IO::Options readOptions;

OpenMesh::IO::read_mesh(*myMesh,"/path/to/bunny.off",readOptions)
```

reader/writer settings:
– enable vertex normals/colors / texture coordinates?
– enable face normals/colors?

Adding attributes

Mesh * myMesh;

OpenMesh::IO::Options readOptions;

OpenMesh::IO::read_mesh(*myMesh,"/path/to/bunny.off",readOptions)

```
if(!readOptions.check(OpenMesh::IO::Options::FaceNormal))
{
    myMesh->update_face_normals();
}


if(! readOptions.check(OpenMesh::IO::Options::VertexNormal))
{
    myMesh->update_vertex_normals();
}
```

Iterating over vertices

```
typedef Openmesh::TriMesh_ArrayKernelT<> Mesh;
Mesh * myMesh;
```

```
Mesh::VertexIter vIt,vBegin,vEnd;

vBegin = myMesh->vertices_begin();
vEnd = myMesh->vertices_end();

for( vIt = vBegin ; vIt != vEnd; ++vIt )
{
    doSomethingWithVertex(vIt.handle());
}
```
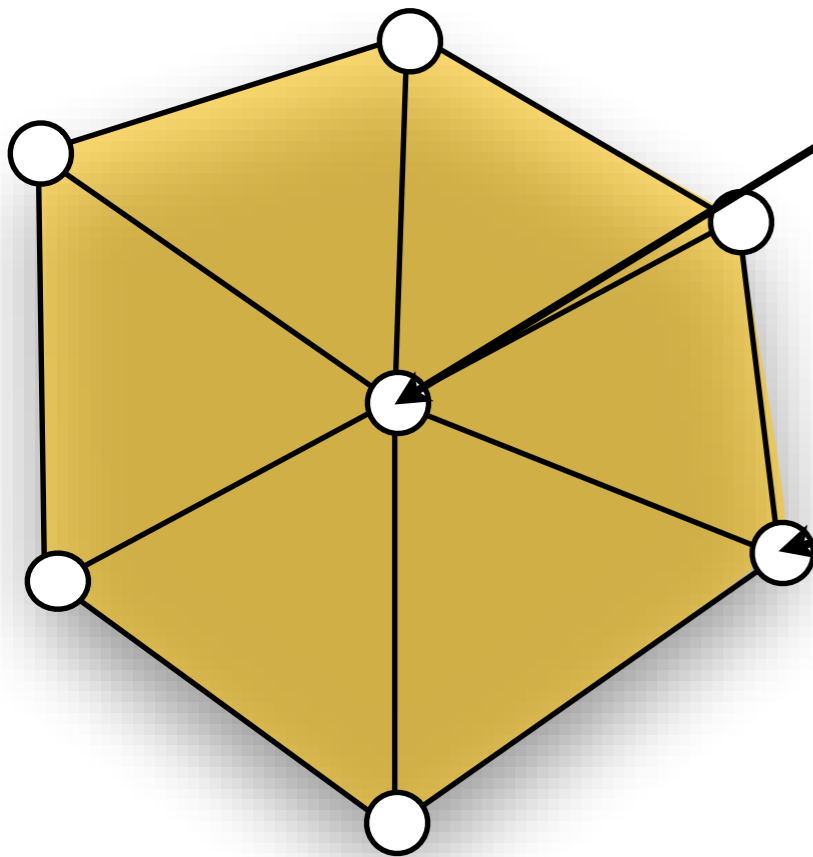
mesh processing

Mesh::VertexIter → Mesh::FaceIter

vertices_begin() → faces_begin()

vertices_end() → faces_end()

Circulating over faces around a vertex


```
Mesh::VertexIter vIt,vBegin,vEnd;

vBegin = myMesh->vertices_begin();
vEnd = myMesh->vertices_end();

for( vIt = vBegin ; vIt != vEnd; ++vIt )
{
```

```
Mesh::VertexFaceIter vfIt,vfBegin;
vfBegin = myMesh->vf_iter(vIt);

for( vfIt = vfBegin ; vfIt ; ++vfIt)
{
    doSomethingWithFace(vfIt.handle());
}
```

```
}
```

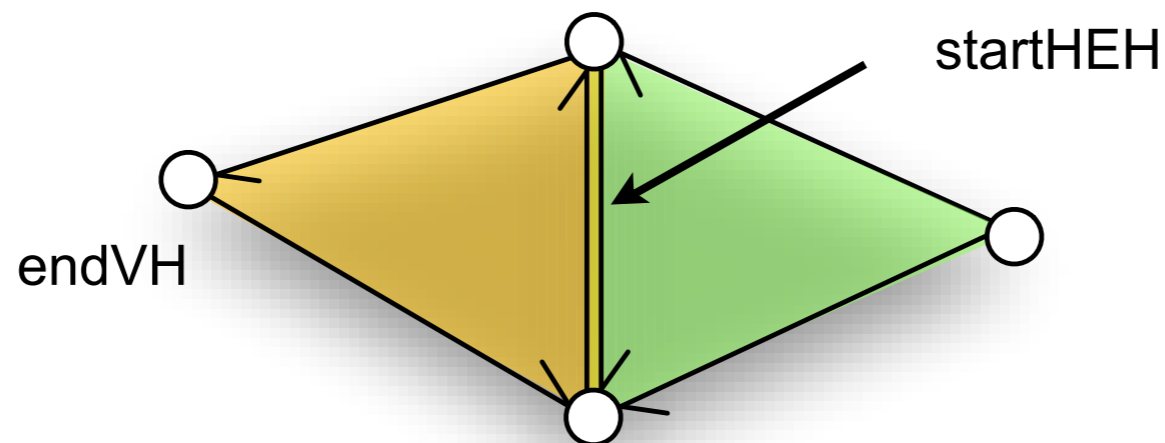returns false after a complete circulation round

Triangle geometry

```cpp
void analyzeTriangle(OpenMesh::FaceHandle & _fh)
{
    OpenMesh::Vec3f pointA,pointB,pointC;
    Mesh::ConstFaceVertexIter cfvIt;

    cfvIt = myMesh->cfv_iter(_fh);
    pointA = myMesh->point(cfvIt.handle());
    pointB = myMesh->point((++cfvIt).handle());
    pointC = myMesh->point((++cfvIt).handle());

    perimeter(pointA,pointB,pointC);
    area(pointA,pointB,pointC)
}
```

Neighborhood access in O(1)

startHEH

endVH

```
OpenMesh::VertexHandle endVH;
OpenMesh::HalfEdgeHandle startHEH,oppositeHEH,nextHEH;

startHEH = hehIt.handle();
```

mesh topology
involved

```
oppositeHEH = myMesh->opposite_halfedge_handle(startHEH);
nextHEH = myMesh->next_halfedge_handle(oppositeHEH);
endVH =  myMesh->to_vertex_handle(nextHEH);
```
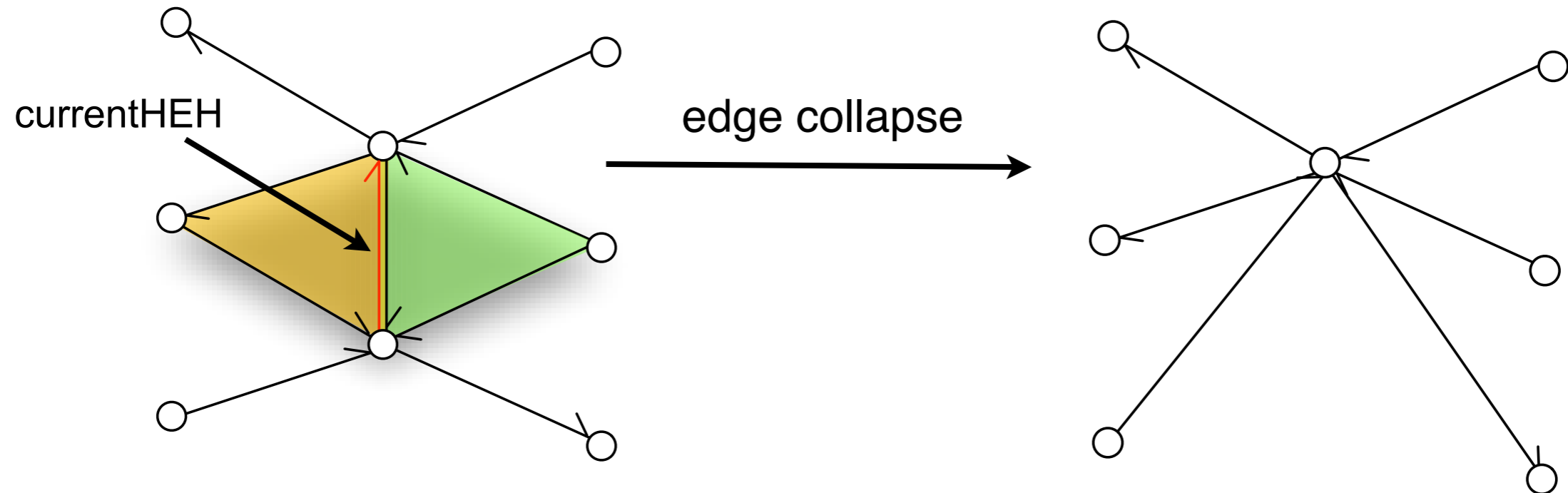
Modifying the geometry

```
for( vIt = vBegin ; vIt != vEnd; ++vIt )
{
  scale(vIt.handle(),2.0);
}
```

```
void scale(OpenMesh::VertexHandle & _vh,double _alpha)
{

    OpenMesh::Vec3f newCoordinate;

    newCoordinate = myMesh->point(_vh);

    myMesh->set_point(_vh, newCoordinate * _alpha);

}
```

Changing the topology

currentHEH

edge collapse

```
myMesh->request_vertex_status();
myMesh->request_edge_status();
myMesh->request_face_status();
```

```
OpenMesh::HalfedgeHandle currentHEH = heIt.handle();
```

```
myMesh->collapse(currentHEH);
myMesh->garbage_collection();
```

Customizing the Mesh

- Face type with predefined array kernel

  ```
  typedef Openmesh::TriMesh_ArrayKernelT<> Mesh;
  typedef Openmesh::PolyMesh_ArrayKernelT<> Mesh;
  ```

- Traits

  predefined attributes:
  – normals / colors
  – coordinate types: 2-D, 3-D, ..., nD
  – scalar types: float, double, ...

  custom attributes: centerOfGravity, ...

Traits – static customization

```cpp
#include <OpenMesh/Core/IO/MeshIO.hh>
#include <OpenMesh/Core/Mesh/Types/TriMesh_ArrayKernelT.hh>
```

```cpp
struct myMeshTraits : public OpenMesh::DefaultTraits
{
    typedef OpenMesh::Vec4f Color;

    VertexAttributes (
            OpenMesh::Attributes::Normal |
            OpenMesh::Attributes::Color);

    FaceAttributes (
            OpenMesh::Attributes::Normal |
            OpenMesh::Attributes::Color);

}
```

```cpp
typedef Openmesh::TriMesh_ArrayKernelT<myMeshTraits> Mesh;
```

Dynamic customization of predefined attributes

```cpp
typedef Openmesh::TriMesh_ArrayKernelT<> Mesh;

Mesh * myMesh;

... // load file into myMesh

myMesh->request_vertex_normals();
myMesh->request_vertex_colors();
myMesh->request_face_normals();

...

myMesh->set_color(currentVH,Mesh::Color(0,0,255));

blueColor = myMesh->color(currentVH);
```

Dynamic customization of custom attributes

```
OpenMesh::FPropHandleT<bool> marked;
myMesh->add_property(marked);

for(fIt = fBegin; fIt != fEnd; ++fIt)
{
    if(shouldMark(fIt))
        myMesh->property(marked,fIt) = true;
    else
        myMesh->property(marked,fIt) = false;
}

for(fIt = fBegin; fIt != fEnd; ++fIt)
{
    if(myMesh->property(marked,fIt))
        doSomething(fIt);
}
```

Three important links

www.openmesh.org → Overview
www.openmesh.org → Tutorial
www.openmesh.org → Documentation

→ Classes

→ Class Members

Further readings

- Documention: http://www.openmesh.org/

- OpenMesh – a generic and efficient polygon mesh data structure [Botsch et al. 2002]

Exercise 1

- Introduction to working with OpenMesh
- Code provided to load/render mesh
- You will modify it to calculate/visualize valence of mesh vertices
- Can use Windows, Linux or OS X
- 3 parts:
  - 1.1 Installation and getting started
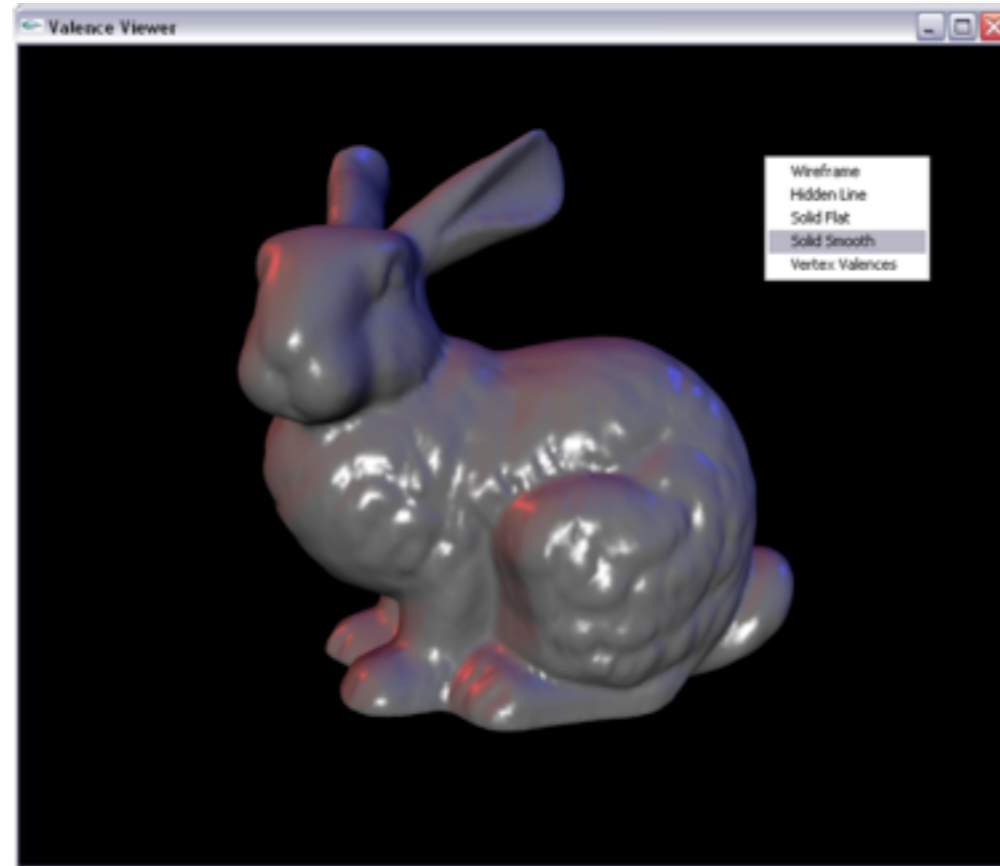 - 1.2 Vertex valence of a triangle mesh
  - 1.3 Color visualization

1.1 Installation and getting started

- Install dependencies:
  - GLUT: http://freeglut.sourceforge.net/
  - CMake: http://www.cmake.org/download/
  - OpenMesh: http://www.openmesh.org/download/
- Download/Unpack Exercise1.zip and handout (Exercise1.pdf) from Blackboard
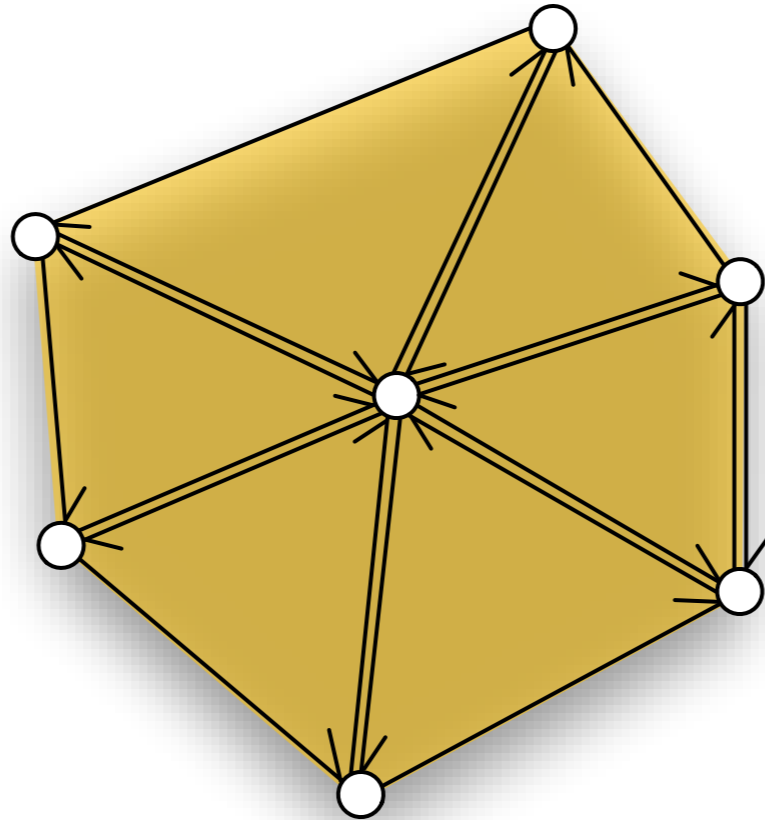
1.1 Installation and getting started

- Handout contains instructions for building starter code on each platform
- Must build OpenMesh from source on Linux / OS X
  - Use CMake as described in handout for building exercise code (followed by "sudo make install")
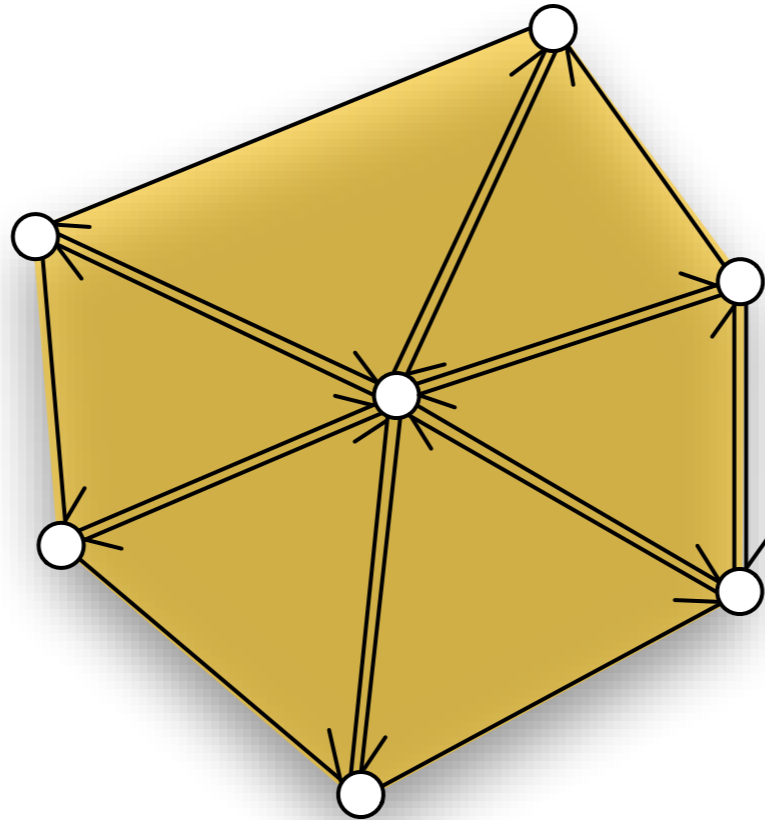
- Pass model to load on command line

  - e.g.: *./exercise1 bunny.off*

- Become familiar with project file organization and classes "GLUTViewer," "MeshViewer" and "ValenceViewer"

- Learn how to use OpenMesh by reading first 4 sections of online OpenMesh tutorial
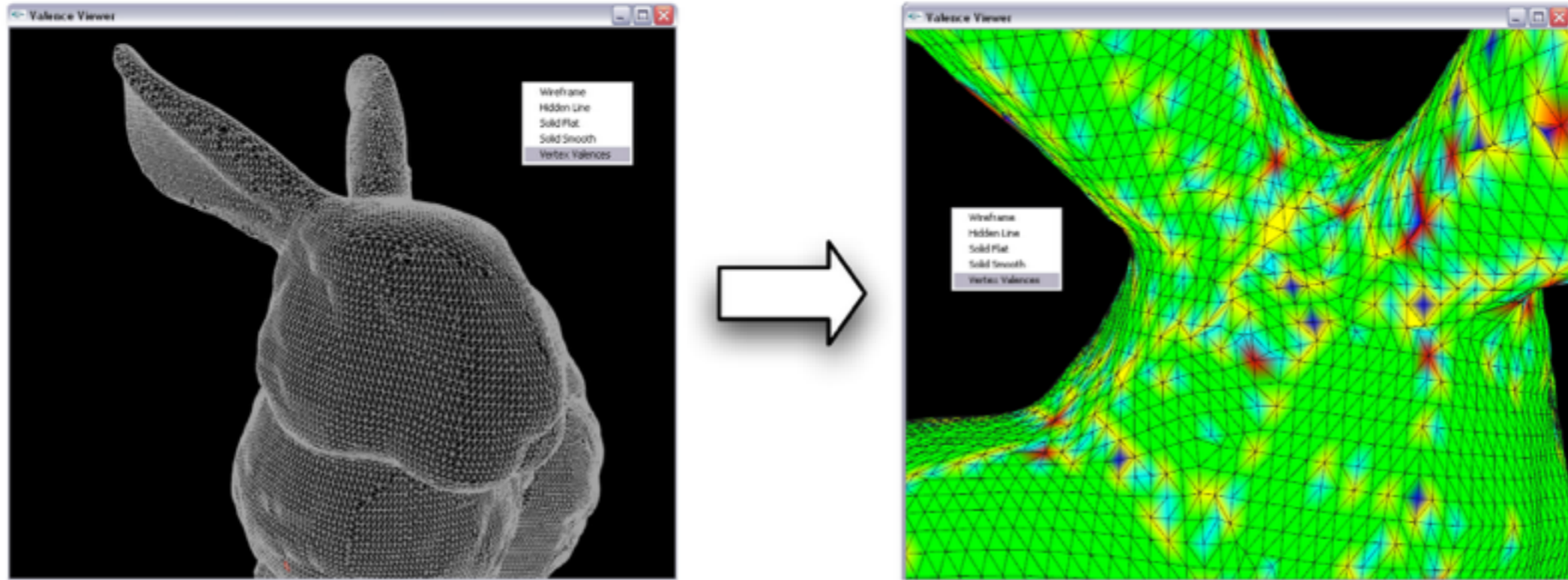
- Review: The valence *v(x)* of a vertex *x* in a triangle mesh is the number of vertices in its one-ring neighborhood
    - Each vertex in neighborhood is connected by an edge to *x*

1.2 Vertex valence of a triangle mesh

- ValenceViewer::calc_valences() called once before render loop starts

- Add code to this method to compute valence of each vertex in mesh "mesh_" (member of superclass "MeshViewer")

- Store valences in custom attribute you must define for each vertex

1.3 Color visualization

- Define function mapping of each valence number to RGB value used as vertex color

- Implement mapping in ValenceViewer::color_coding() (called right after calc_valences())

- Describe your mapping in readme.txt submitted with assignment

- Use *predefined* attributes for color (don't define your own)

Submission

- Deadline: **Mon. Feb. 2, 11:59 PM**
- Submit via Blackboard:
  - CMake script and ALL source files (even those you didn't need to change)
  - readme.txt:
    - Describe how you solved each exercise, using same exercise numbers (1.1 - 1.3) and titles as in handout
    - Describe problems you encountered
- Upload file named Exercise1-YourName.zip on Blackboard (make sure to click "Submit" afterwards)