

Spring 2017

CSCI 621: Advanced Digital Geometry Processing

Introduction to OpenMesh (and Exercise 1)



Yi Zhou

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

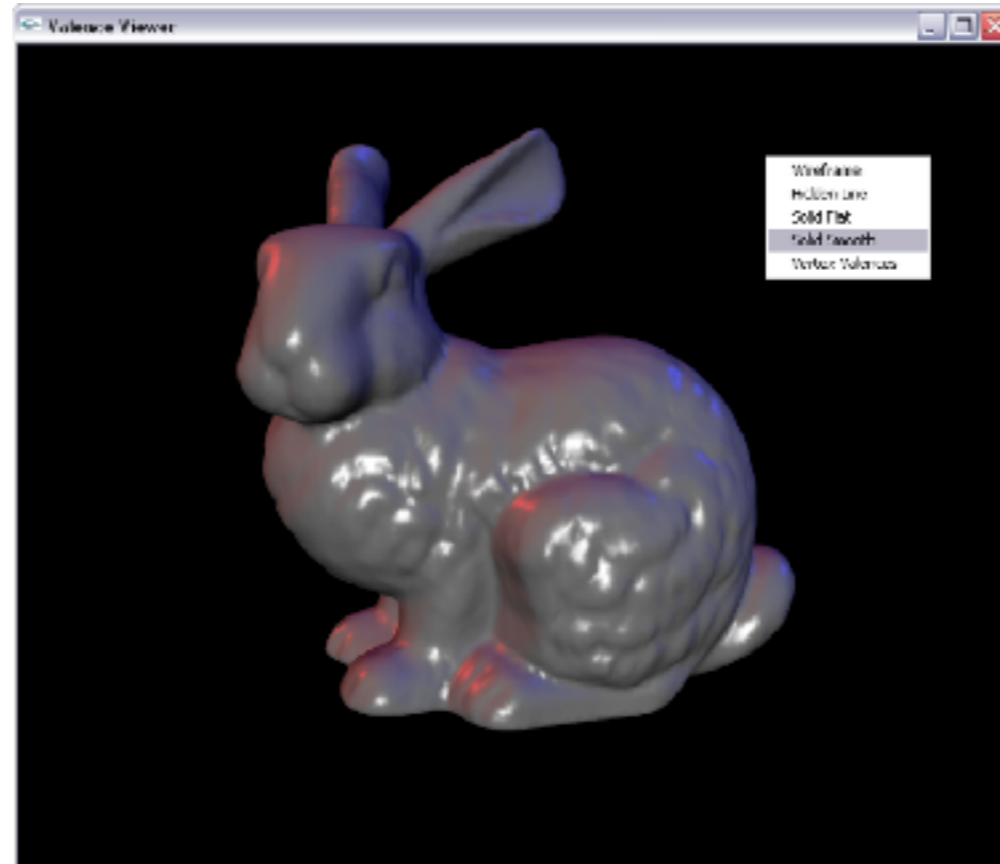
1.1 Installation and getting started

- 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

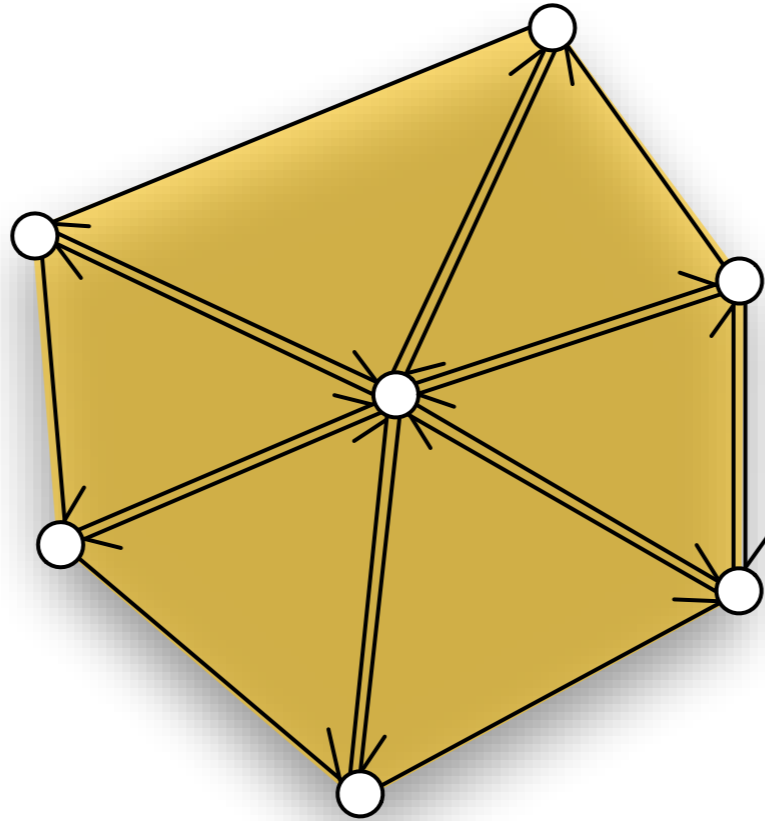
- Handout contains instructions for building starter code on each platform
- Must build OpenMesh from source on Linux / OS X
 - Use CMake as described in handout for building exercise code (followed by “sudo make install”)

1.1 Installation and getting started



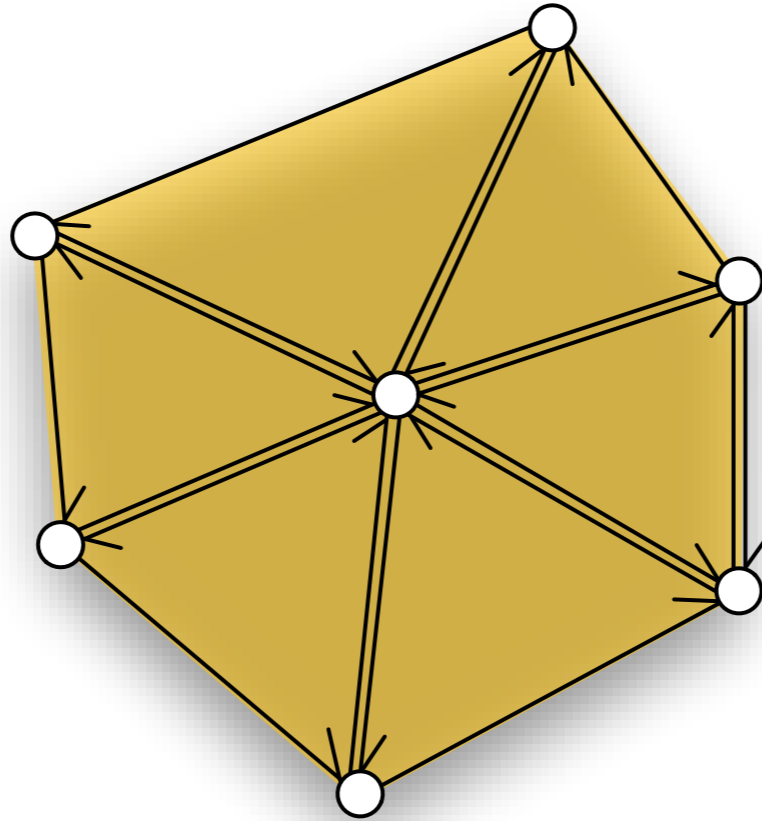
- Pass model to load on command line
 - e.g.: `./exercise1 bunny.off`
- Become familiar with project file organization and classes “GLUTViewer,” “MeshViewer” and “ValenceViewer”
- Learn how to use OpenMesh by reading first 4 sections of online OpenMesh tutorial

1.2 Vertex valence of a triangle mesh



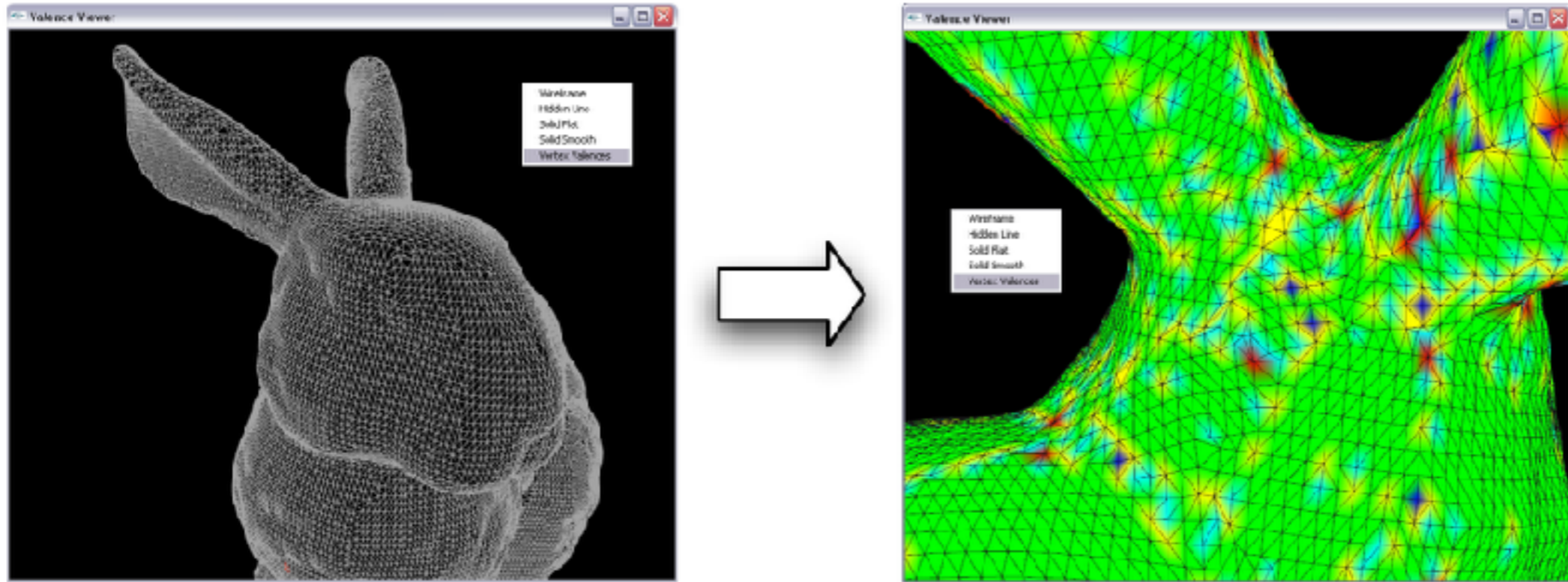
- Review: The valence $v(x)$ of a vertex x in a triangle mesh is the number of vertices in its one-ring neighborhood
 - Each vertex in neighborhood is connected by an edge to x

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. 7, 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)

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 v_It, v_Begin, v_End;
```

```
v_Begin = myMesh->vertices_begin();  
v_End = myMesh->vertices_end();
```

```
for( v_It = vBegin ; v_It != vEnd; ++v_It )  
{  
    doSomethingWithVertex(v_It.handle());  
}
```



mesh processing

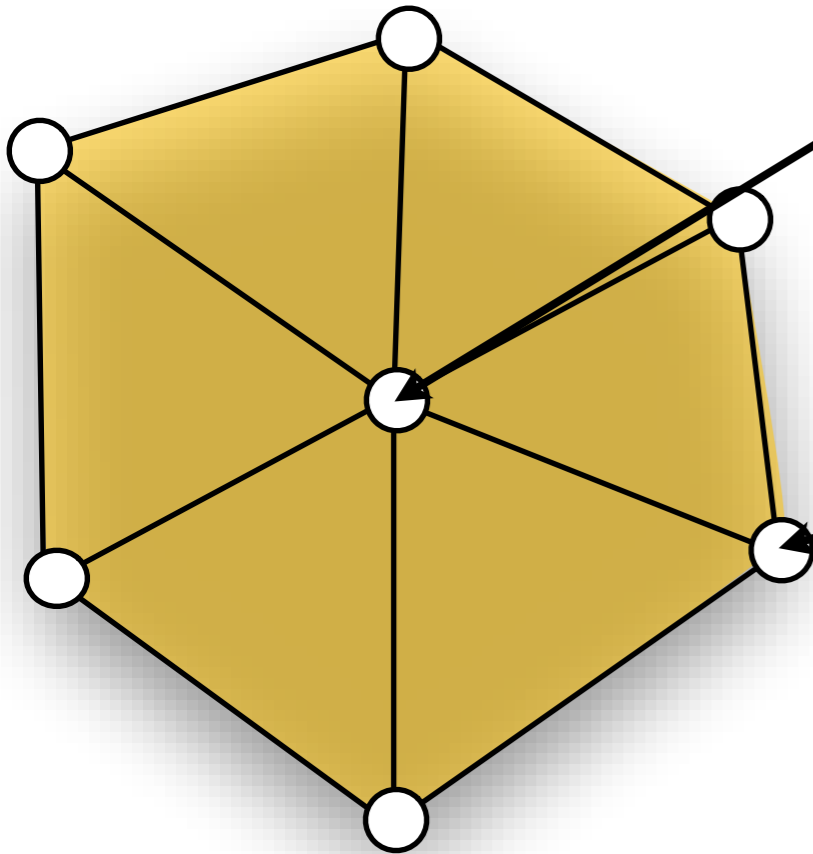
Iterating over faces

Mesh::VertexIter → Mesh::FaceIter

vertices_begin() → faces_begin()

vertices_end() → faces_end()

Circulating over faces around a vertex



```
Mesh::VertexIter v_It, v_Begin, v_End;
```

```
v_Begin = myMesh->vertices_begin();  
v_End = myMesh->vertices_end();
```

```
for( v_It = v_Begin ; v_It != v_End; ++v_It )  
{
```

```
Mesh::VertexFaceIter vf_It, vf_Begin;  
vf_Begin = myMesh->vfinder(v_It);
```

```
for( vf_It = vf_Begin ; vf_It ; ++vf_It )  
{  
    doSomethingWithFace(vf_It.handle());  
}
```

```
}
```

returns false after a complete circulation round

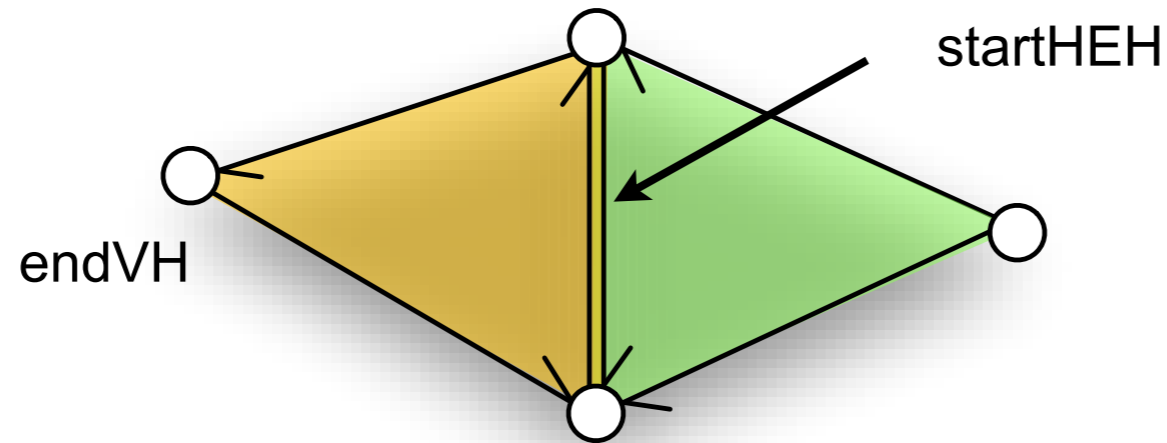
Triangle geometry

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

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

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

Neighborhood access in $O(1)$



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

```
startHEH = hehlt.handle();
```

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

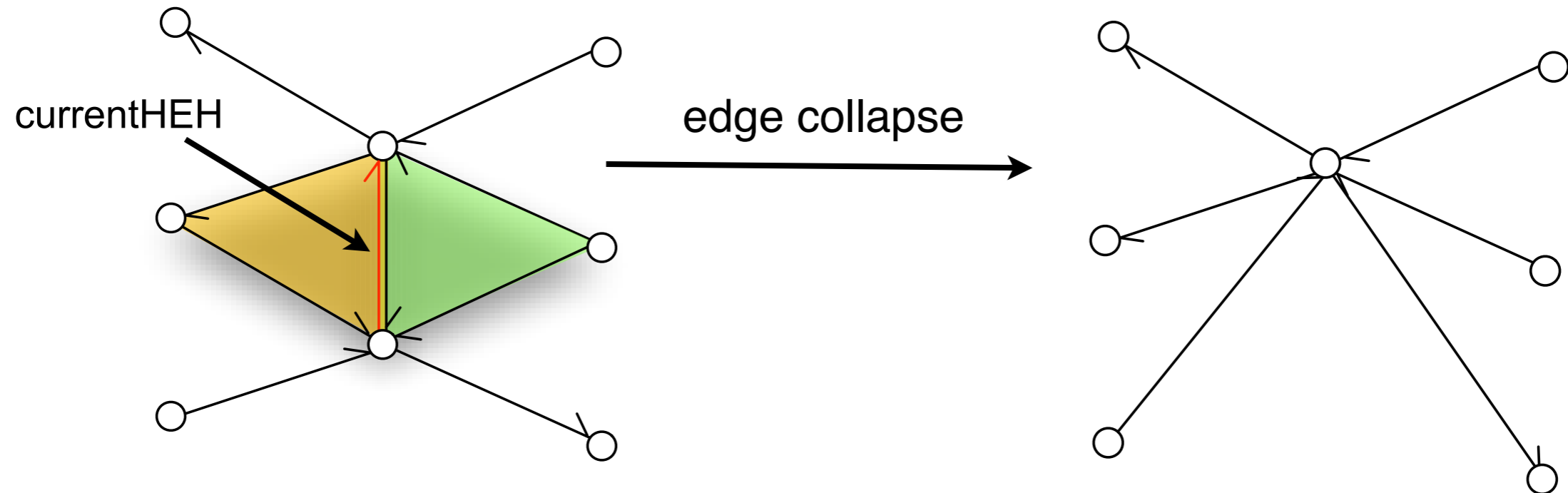
mesh topology
involved

Modifying the geometry

```
for( vlt = vBegin ; vlt != vEnd; ++vlt )  
{  
    scale(vlt.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



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

```
OpenMesh::HalfedgeHandle currentHEH = helt.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

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

```
struct myMeshTraits : public OpenMesh::DefaultTraits  
{  
    typedef OpenMesh::Vec4f Color;  
  
    VertexAttributes (  
        OpenMesh::Attributes::Normal |  
        OpenMesh::Attributes::Color);  
  
    FaceAttributes (  
        OpenMesh::Attributes::Normal |  
        OpenMesh::Attributes::Color);  
  
}
```

```
typedef OpenMesh::TriMesh_ArrayKernelT<myMeshTraits> Mesh;
```


Dynamic customization of predefined attributes

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

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

Three important links

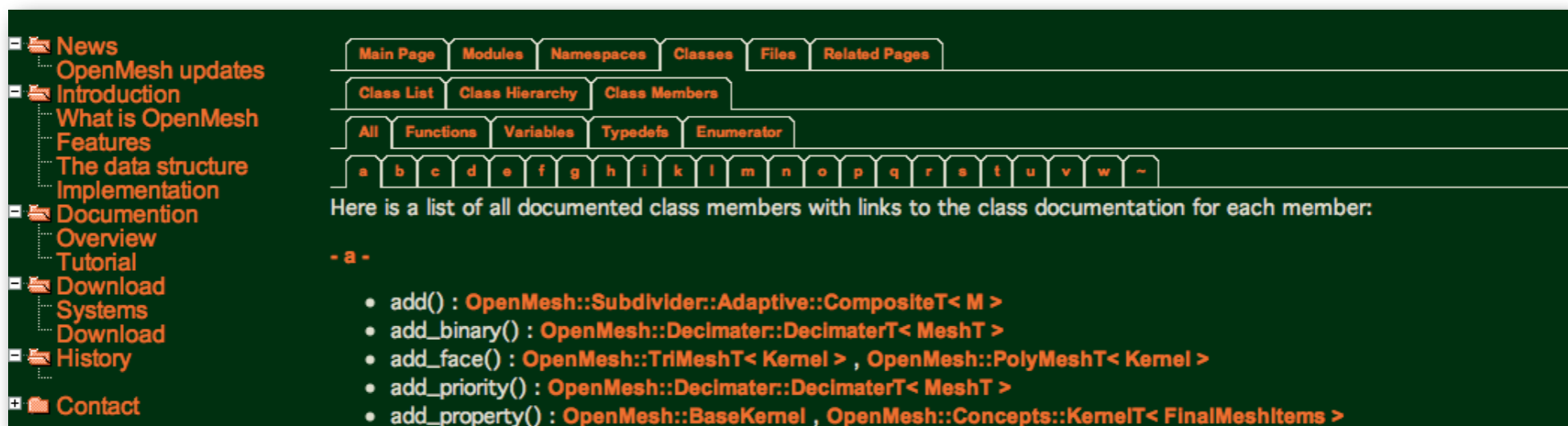
www.openmesh.org → Overview

www.openmesh.org → Tutorial

www.openmesh.org → Documentation

→ Classes

→ Class Members



The screenshot shows the OpenMesh website documentation page. On the left is a navigation menu with items like News, Introduction, Features, Documentation, and Download. The main content area has several navigation tabs: Main Page, Modules, Namespaces, Classes, Files, and Related Pages. Below these are tabs for Class List, Class Hierarchy, and Class Members. Further down are tabs for All, Functions, Variables, Typedefs, and Enumerator. A letter index bar contains letters from a to w and a tilde symbol. Below the index bar, the text reads: "Here is a list of all documented class members with links to the class documentation for each member:". Under the heading "- a -", there is a list of class members:

- [add\(\) : OpenMesh::Subdivider::Adaptive::CompositeT< M >](#)
- [add_binary\(\) : OpenMesh::Decimater::DecimaterT< MeshT >](#)
- [add_face\(\) : OpenMesh::TriMeshT< Kernel > , OpenMesh::PolyMeshT< Kernel >](#)
- [add_priority\(\) : OpenMesh::Decimater::DecimaterT< MeshT >](#)
- [add_property\(\) : OpenMesh::BaseKernel , OpenMesh::Concepts::KernelT< FinalMeshItems >](#)

Further readings

- Documentation: <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

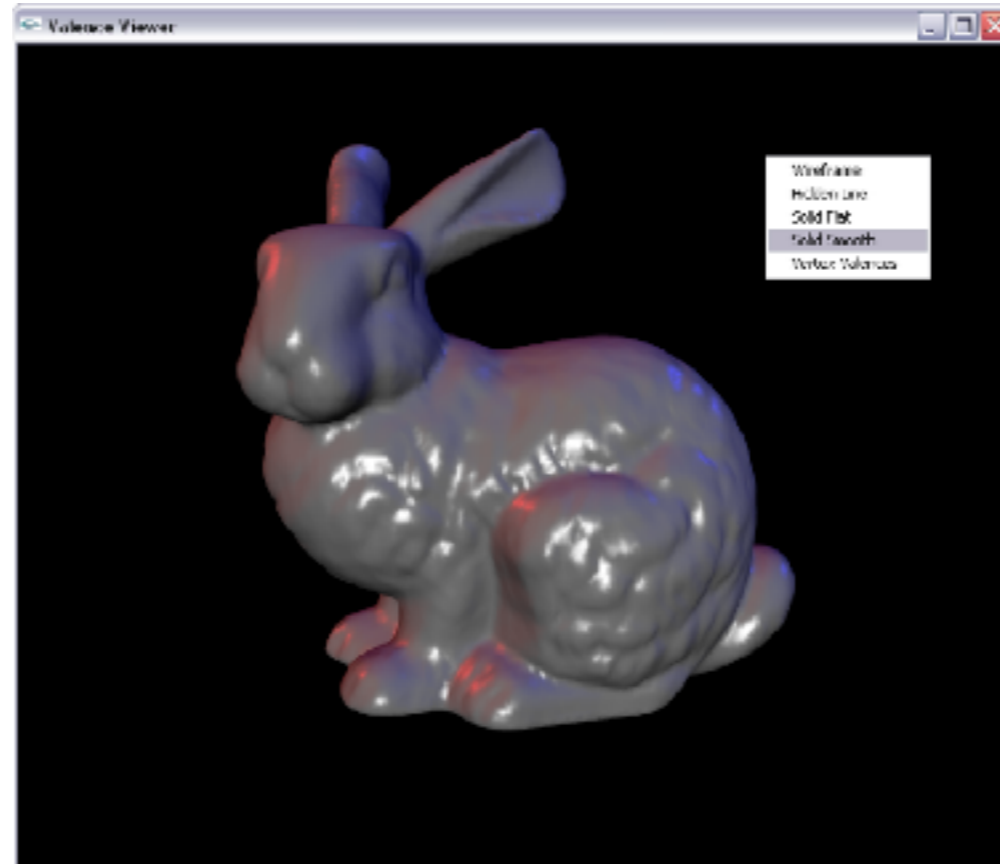
1.1 Installation and getting started

- 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

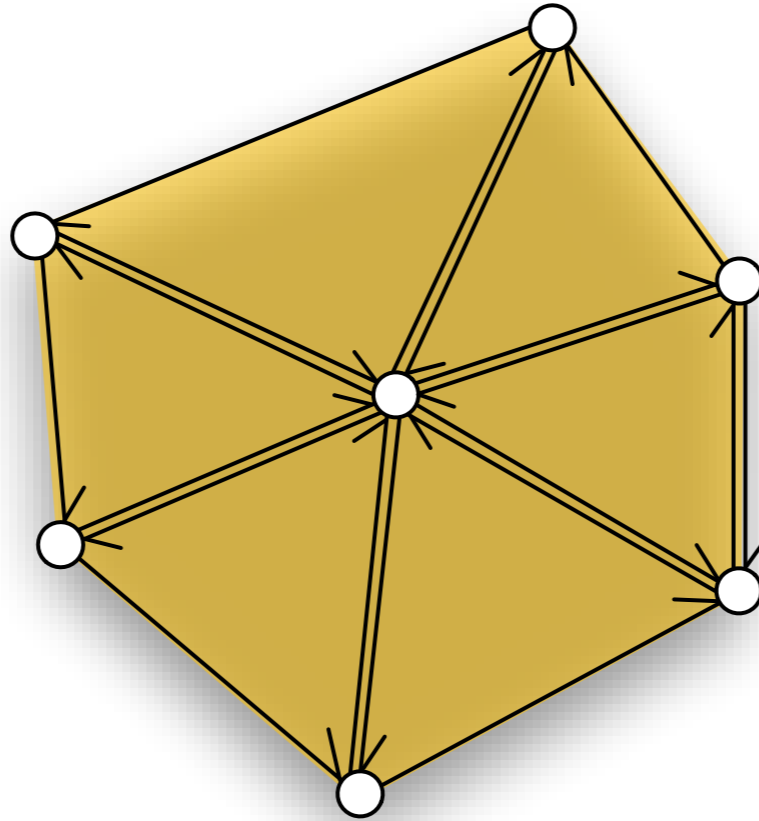
- Handout contains instructions for building starter code on each platform
- Must build OpenMesh from source on Linux / OS X
 - Use CMake as described in handout for building exercise code (followed by “sudo make install”)

1.1 Installation and getting started



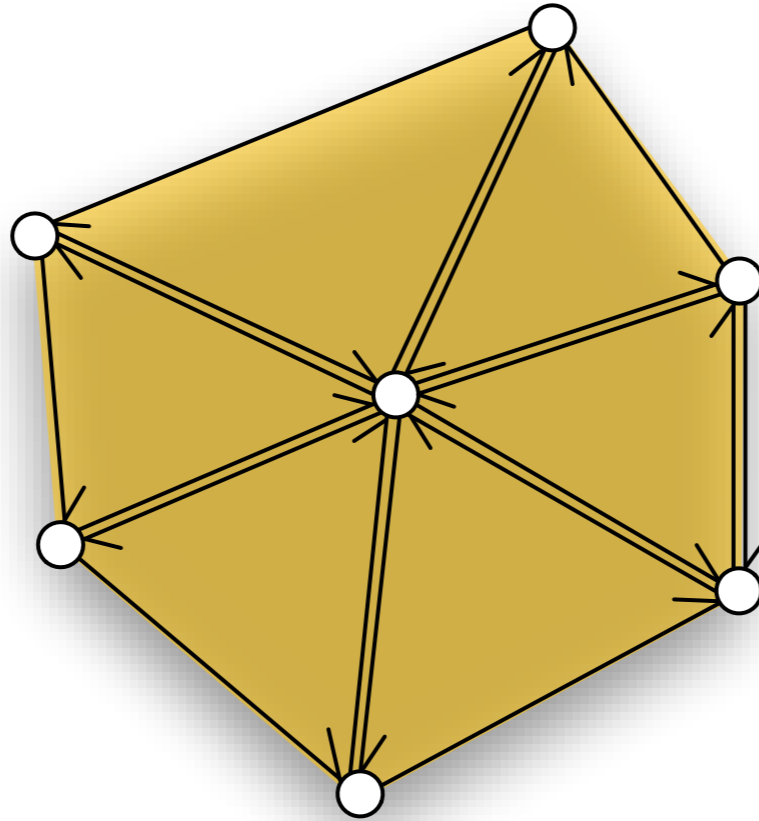
- Pass model to load on command line
 - e.g.: `./exercise1 bunny.off`
- Become familiar with project file organization and classes “GLUTViewer,” “MeshViewer” and “ValenceViewer”
- Learn how to use OpenMesh by reading first 4 sections of online OpenMesh tutorial

1.2 Vertex valence of a triangle mesh



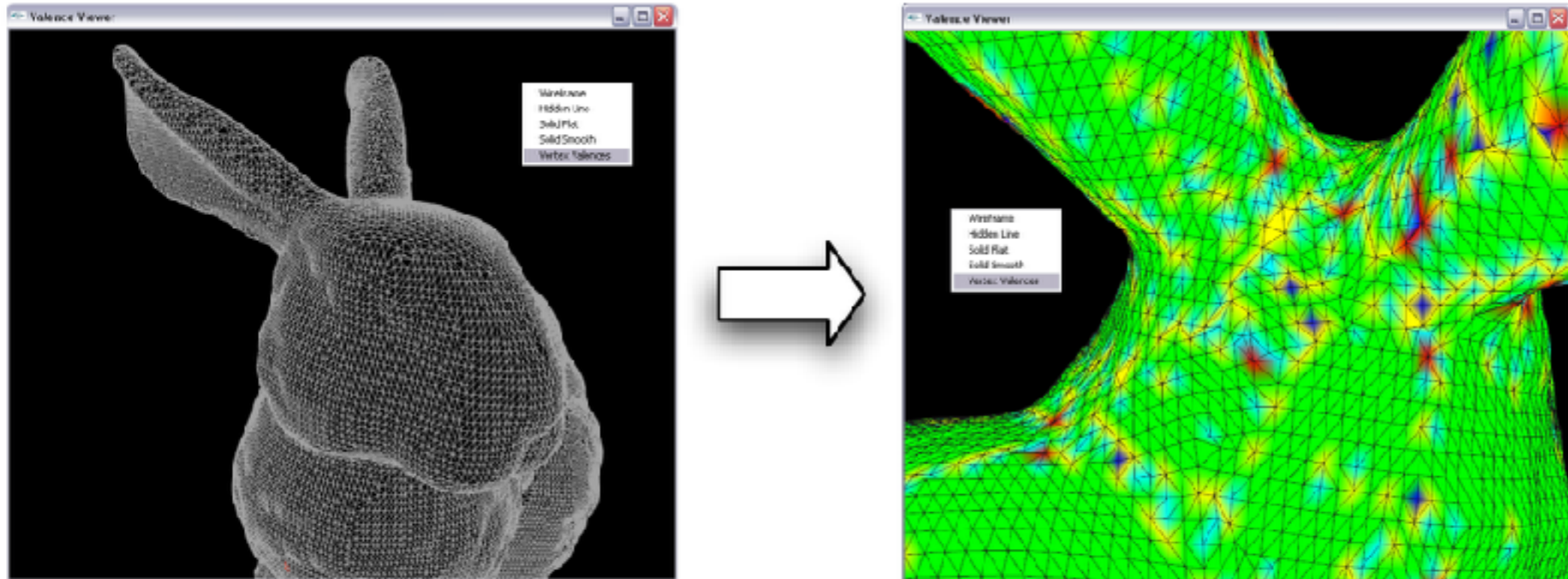
- Review: The valence $v(x)$ of a vertex x in a triangle mesh is the number of vertices in its one-ring neighborhood
 - Each vertex in neighborhood is connected by an edge to x

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)