CENG 789 – Digital Geometry Processing

04- Mesh Data Structure and 3D Graphics Programming

Prof. Dr. Yusuf Sahillioğlu

Computer Eng. Dept, MIDDLE EAST TECHNICAL UNIVERSITY, Turkey

Mesh Data Structure

- ✔ Polygon mesh: set of *polygons* embedded in 2D or 3D.
- ✔ Polygon mesh: set of *vertices, edges, and faces* embedded in 2D or 3D.
- ✔ Lets handle these vertices, edges, faces in a structured way, hence the mesh data structure.

Mesh Data Structure

- How to store geometry & connectivity of a mesh.

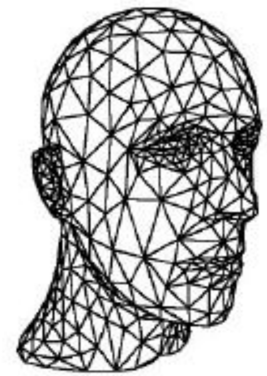
  3D vertex coordinates   Vertex adjacency

- Attributes also stored: normal, color, texture coords, labels, etc.

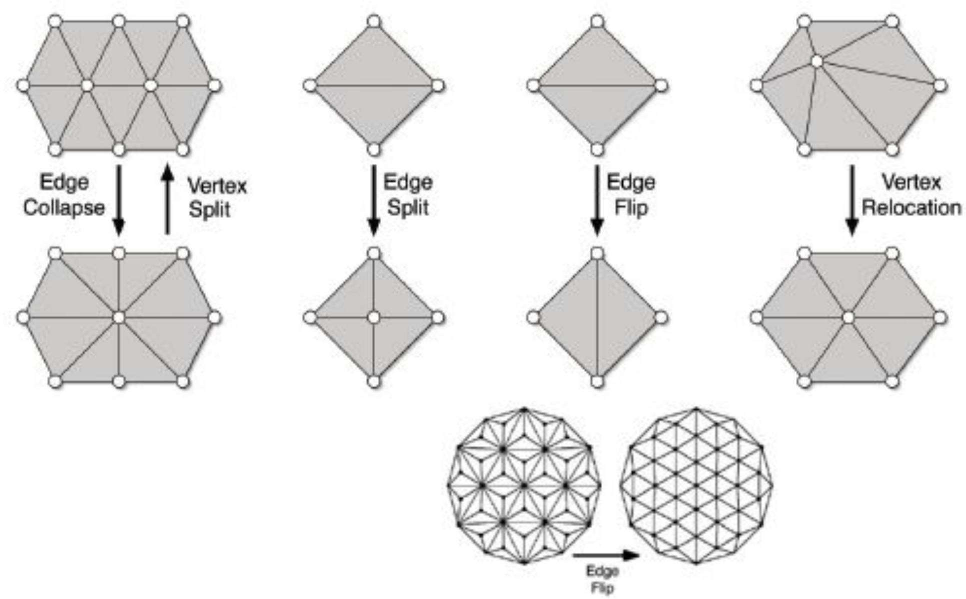
- Efficient algorithms on meshes to get:
  - All vertices/edges of a face.
  - All incident vertices/edges/faces of a vertex.

Mesh Data Structure

- ✔ Classical queries:
  - ✔ What are the vertices of face 77?
  - ✔ Is vertex 7 adjacent to vertex 17?
  - ✔ Which edges are incident to vertex 27?
  - ✔ Which faces are incident to vertex 27?

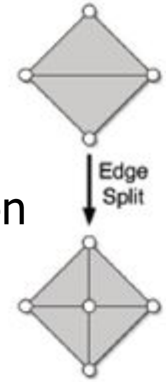


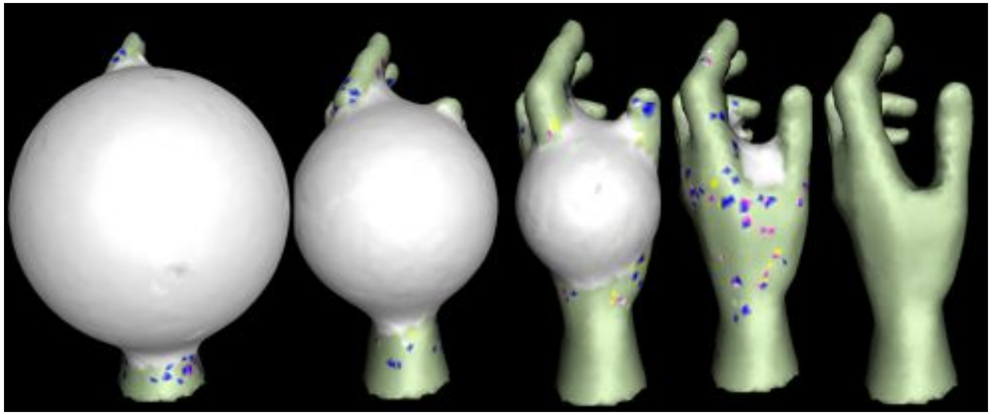
- ✔ Classical operations:
  - ✔ Remove/add vertex/face.
  - ✔ Split/collapse/flip edges.
  - ✔ Change vertex coordinates.
  - ✔ Topological vs. geometrical.



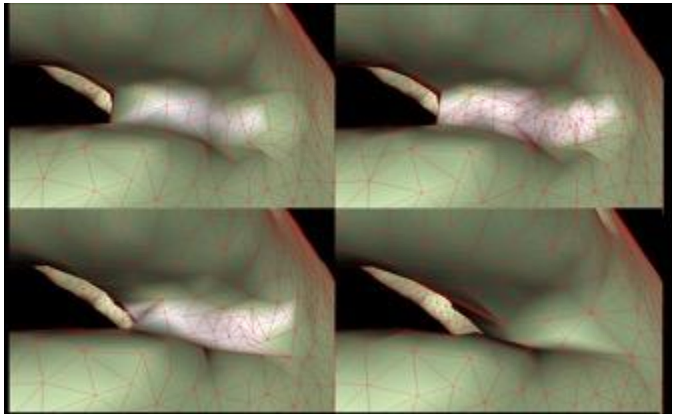
Mesh Data Structure

- ✔ Applications of edge split:
- ✔ Increase resolution to catch details in 3D reconstruction
  - ✔ Paper: Shape from silhouette using topology-adaptive mesh deformation



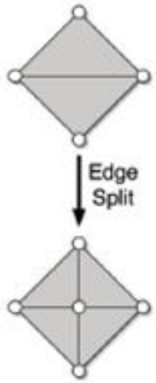


- ✔ Split short edge if midpoint is OUT:

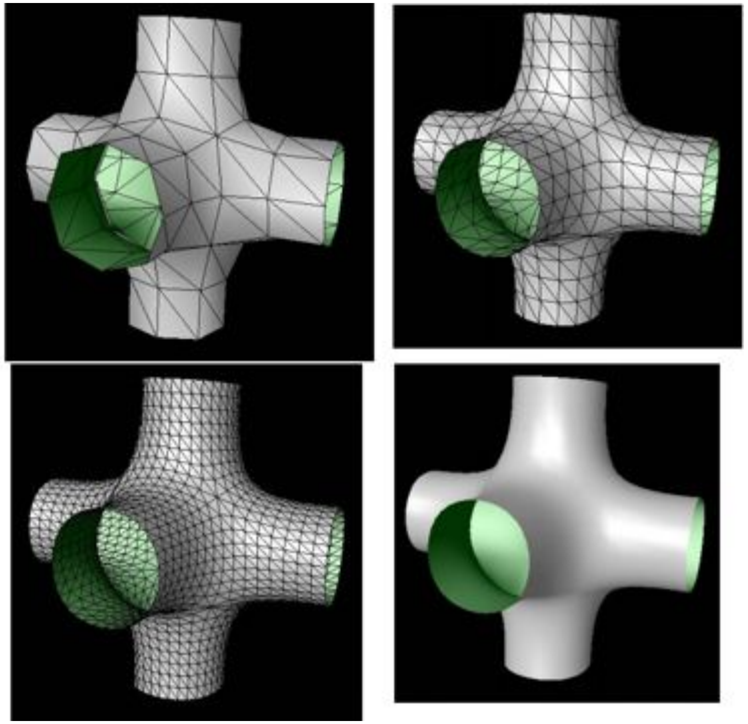


Mesh Data Structure

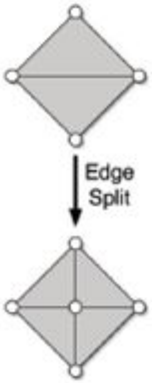
- ✔ Applications of edge split:
- ✔ Increase resolution for smoother surfaces: Subdivision Surfaces
  - ✔ Loop subdivision



- ✔ 32 (original) to 1628 vertices in 3 iterations:



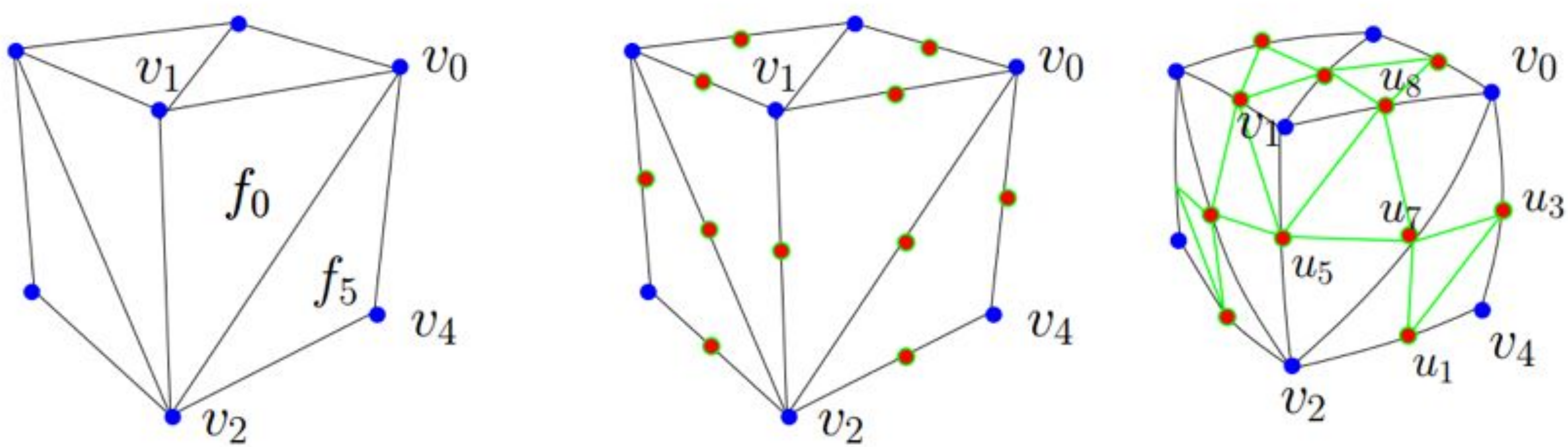
Mesh Data Structure



- ✔ Applications of edge split:
- ✔ Increase resolution for smoother surfaces: Subdivision Surfaces
  - ✔ Loop subdivision
  - ✔ Updating the topology (connectivity)

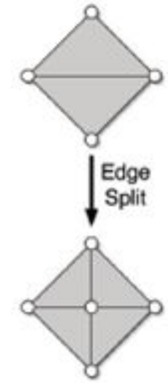
split all edges, by inserting a midpoint

subdivide each face into 4 triangles

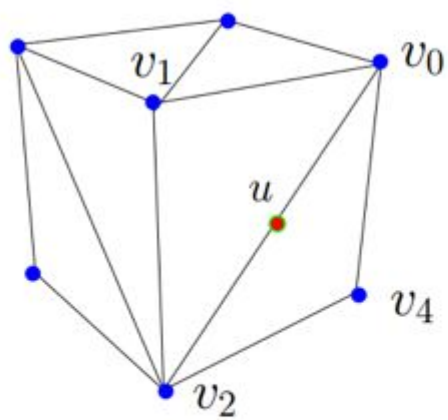


Mesh Data Structure

- ✔ Applications of edge split:
- ✔ Increase resolution for smoother surfaces: Subdivision Surfaces
  - ✔ Loop subdivision
  - ✔ Updating the geometry (coordinates)

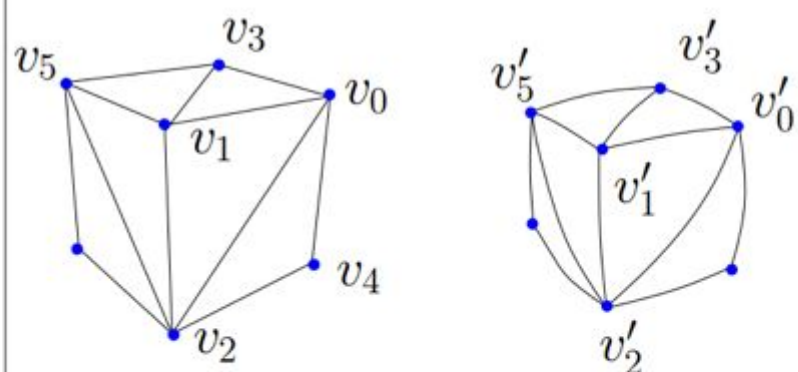


First compute edge points u_k



$$u = \tfrac{3}{8}v_0 + \tfrac{3}{8}v_2 + \tfrac{1}{8}v_1 + \tfrac{1}{8}v_4$$

Compute new locations v_i' of initial vertices



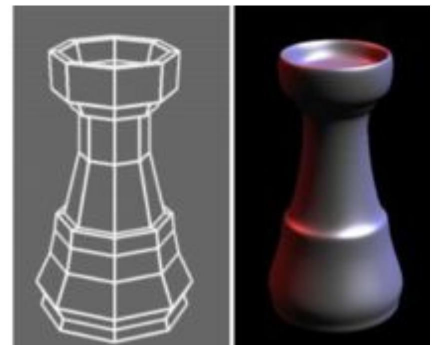
$$v_i' = (1 - \alpha d)v_i + \alpha \sum_{j=1}^{d} v_{i_j}$$

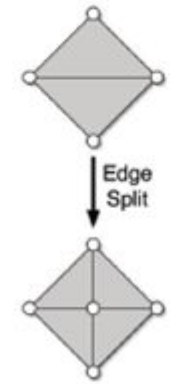
d is the *degree* of vertex v_i
v_{i_j} is the j-th neighbor of v_i

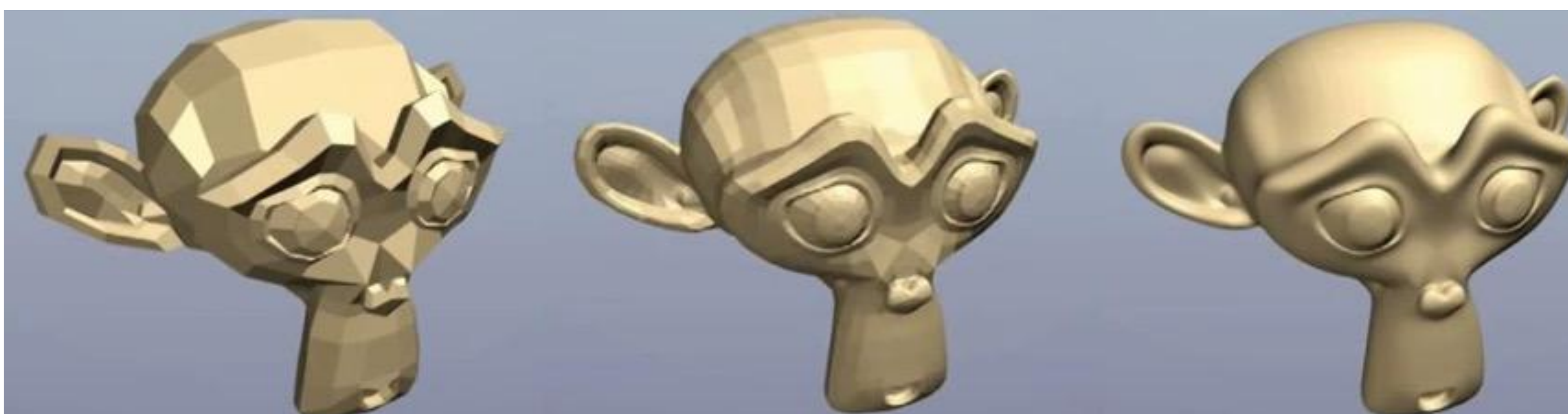
$\alpha = \frac{3}{16}$, if $d = 3$
$\alpha = \frac{3}{8d}$, if $d > 3$

Mesh Data Structure

- ✔ Applications of edge split:
- ✔ Increase resolution for smoother surfaces: Subdivision Surfaces
  - ✔ Loop subdivision (best for triangle meshes)
  - ✔ Catmull-Clark subdivision (quad meshes)
  - ✔ Butterfly subdivision
  - ✔ Doo-Sabin subdivision
  - ✔ $\sqrt{3}$-subdivision
- ✔ More on this in the Subdivison lecture later in semester.

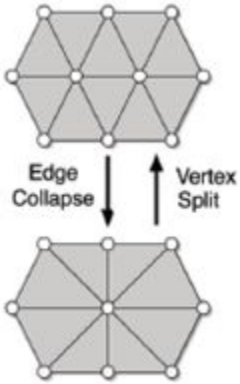


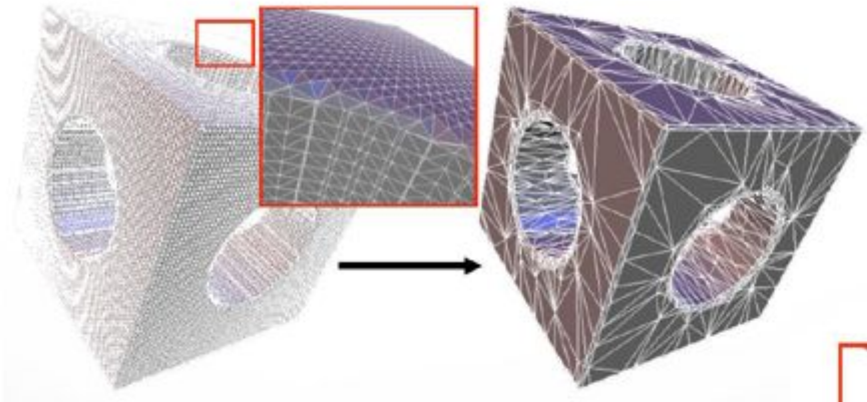


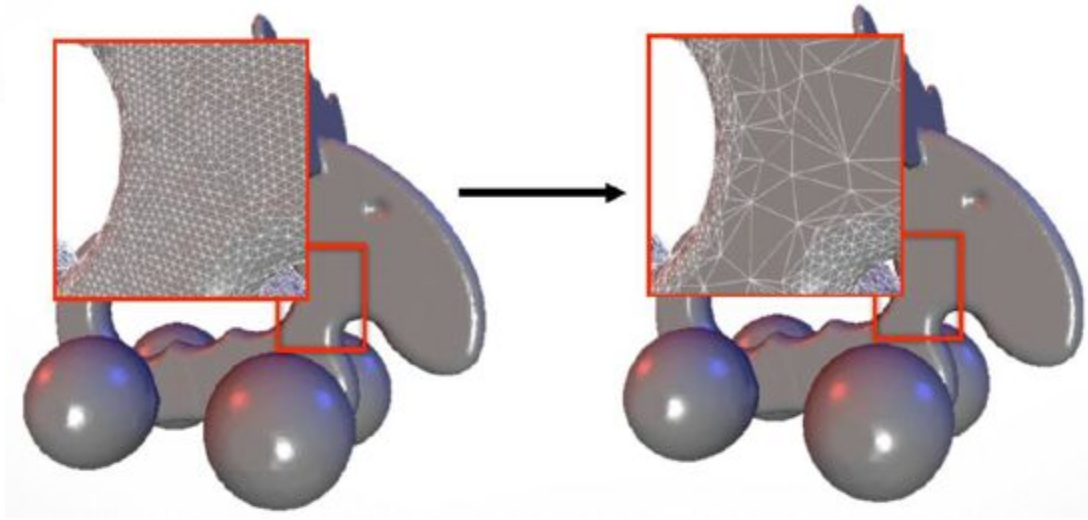


Mesh Data Structure

- ✔ Applications of edge collapse:
- ✔ Decrease resolution for efficiency
  - ✔ Detail-preserving

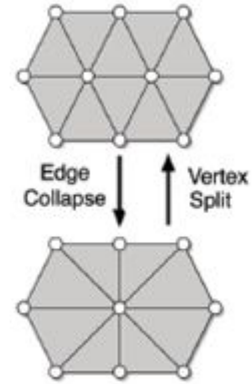


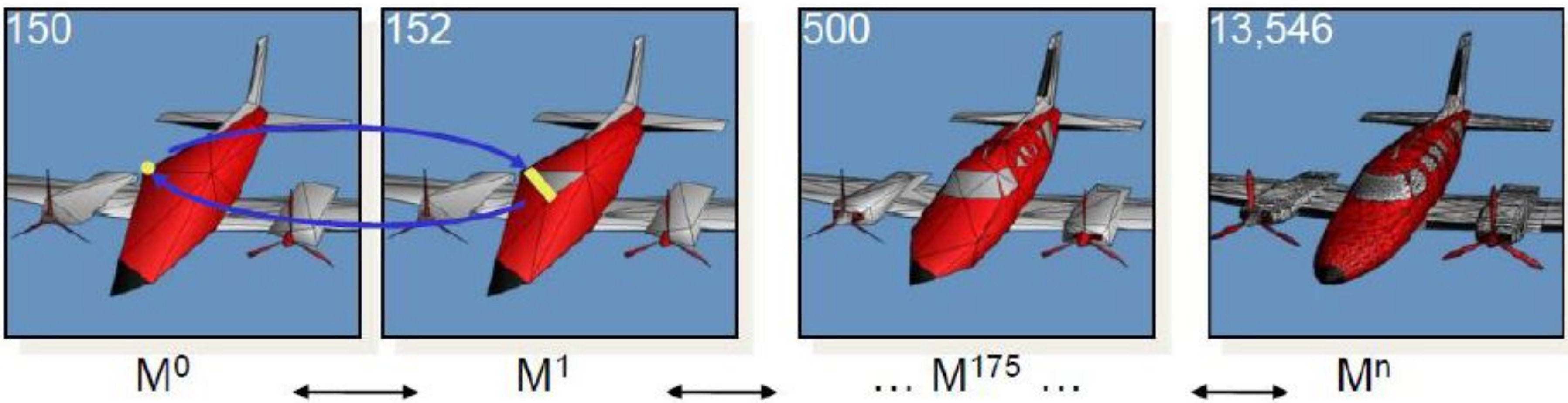




Mesh Data Structure

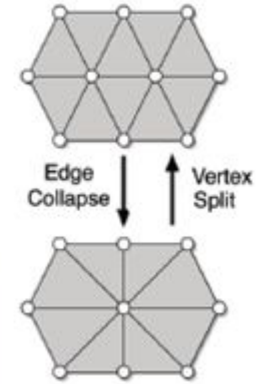
- ✔ Applications of edge collapse:
- ✔ Decrease resolution for efficiency
  - ✔ Detail-oblivious (level-of-detail)

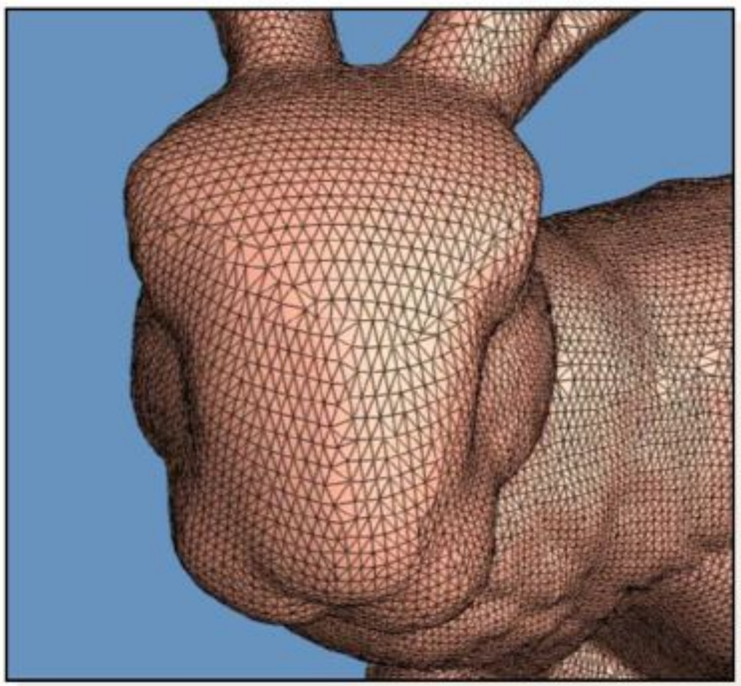


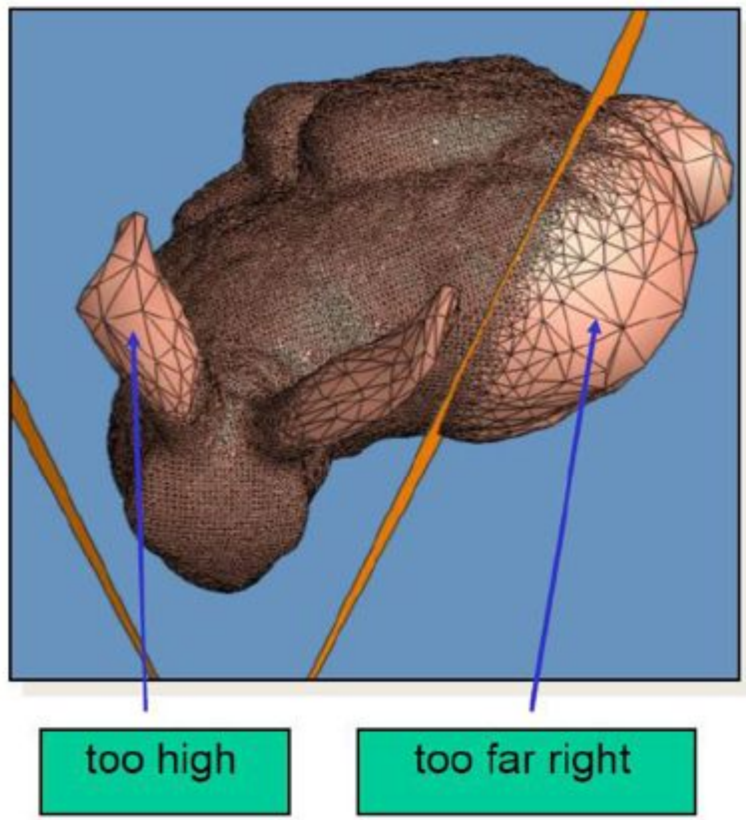


Mesh Data Structure

- ✔ Applications of edge collapse:
- ✔ Decrease resolution for efficiency
  - ✔ Detail-oblivious (view-dependent rendering)

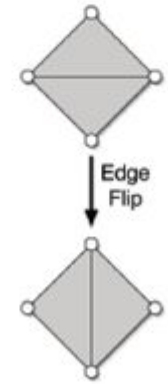


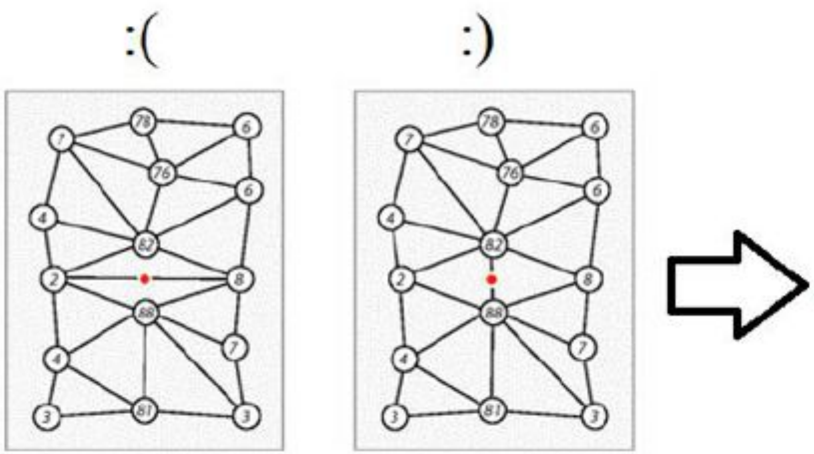


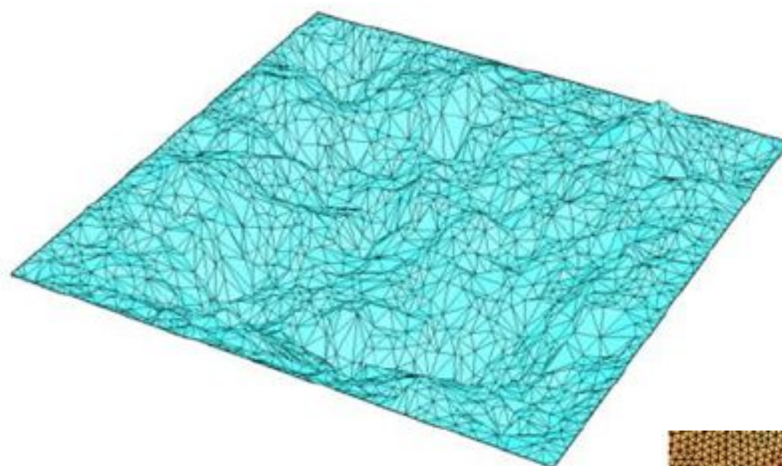


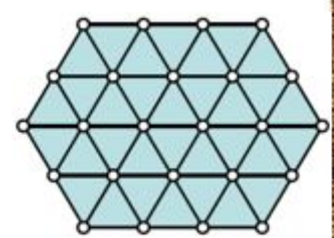
Mesh Data Structure

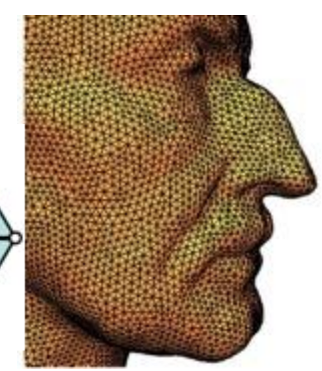
- ✔ Applications of edge flip:
- ✔ Better triangulations, e.g., w/ less skinny triangles
- ✔ Finite element modeling, simulations, terrain construction











Face-Based Data Structures

- ✔ One way to implement a mesh data structure is through faces.
- ✔ Face-Set data structure. //aka polygon soup ‘cos no connectivity info.

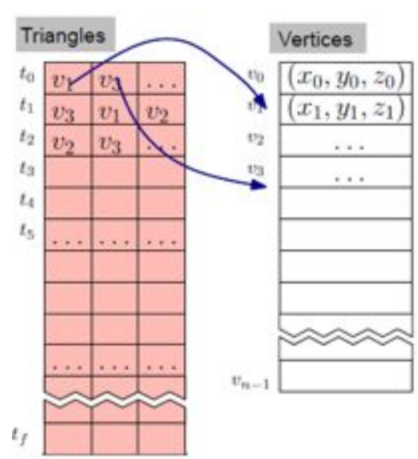
Triangles		
x_{11} y_{11} z_{11}	x_{12} y_{12} z_{12}	x_{13} y_{13} z_{13}
x_{21} y_{21} z_{21}	x_{22} y_{22} z_{22}	x_{23} y_{23} z_{23}
...	...	...
x_{F1} y_{F1} z_{F1}	x_{F2} y_{F2} z_{F2}	x_{F3} y_{F3} z_{F3}

a are replicated.

- ✔ Indexed face-set data structure (obj, off, ply formats). ***Better!!!!***

Vertices
x_1 y_1 z_1
...
x_V y_V z_V

Triangles
i_{11} i_{12} i_{13}
...
...
...
...
i_{F1} i_{F2} i_{F3}



Face-Based Data Structures

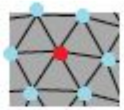
- ✔ My base code implements Indexed face-set data structure
- ✔ See http://www.ceng.metu.edu.tr/~ys/ceng789-dgp/MyDemo.zip

```
struct Triangle
{
    int idx; //tris[idx] is this triangle
    int v1i, v2i, v3i; //triangle formed by verts[v1i]-verts[v2i]-verts[v3i]
    float area, * normal; //direction

    Triangle(int i, int a, int b, int c) : idx(i), v1i(a), v2i(b), v3i(c) {};
};

struct Vertex
{
    int idx; //verts[idx]
    float* coords, //coords[0] ~ x coord, ..
         * normal; //direction

    vector< int > triList;
    vector< int > edgeList;
    vector< int > vertList;

    Vertex(int i, float* c) : idx(i), coords(c) {};
};

struct Edge
{
    int idx; //edges[idx]
    int v1i, v2i;
    float length;

    Edge(int i, int a, int b, float l) : idx(i), v1i(a), v2i(b), length(l) {};
};
```

```
class Mesh
{
public:
    vector< Triangle* > tris;
    vector< Vertex* > verts;
    vector< Edge* > edges;

    void loadOff(char* fName);
    void createCube(float sl);
private:
    void addVertex(float* c);
    void addTriangle(int v1i, int v2i, int v3i);
    void addEdge(int v1i, int v2i);
    bool makeVertsNeighbors(int v, int w);
};
```

Edge-Based Data Structures

- ✔ Another way to implement a mesh data structure is through edges.
- ✔ For explicit storage of edges.
  - ✔ Enables efficient one-ring enumeration.



- ✔ Can be done with slight modifications to Indexed face-set.
  - ✔ Define an Edge struct.
  - ✔ In addition to coordinates, vertices have refs to Vertexes, Edges, Faces.
  - ✔ Begin coding to demonstrate this and introduce Open Inventor.
    - ✔ Read Open Inventor Mentor for detailed Open Inventor programming.

- ✔ Ready-to-use mesh processing libraries and software:
  - ✔ CGAL (lib)
  - ✔ OpenMesh (lib)
  - ✔ MeshLab (sw)

3D Graphics Programming

- We prefer a high-level object-oriented approach to 3D graphics development.
- We also prefer a programming API that brings its own 3D viewer (with trackball navigation and everything).
- Coin3D, an independent implementation of Open Inventor, fits best.
  - https://youtu.be/lK7aoc1AO8w
  - Inventor Mentor: http://www-evasion.imag.fr/Membres/Francois.Faure/doc/inventorMentor/sgi_html/
- The Visualization Toolkit (VTK) is a popular alternative.
  - https://youtu.be/IgvbhyDh8r0
  - Manual: https://www.researchgate.net/profile/William_Schroeder3/publication/200034772_The_Visualization_Toolkit_An_Object-Oriented_Approach_To_3D_Graphics/links/57dfcfa708ae1dcfea865e57/The-Visualization-Toolkit-An-Object-Oriented-Approach-To-3D-Graphics.pdf
- Other alternatives are
  - libigl: http://libigl.github.io/libigl
  - OpenMesh: http://www.openmesh.org
  - OpenSceneGraph: https://youtu.be/1l5PAVCj2iY

Open Inventor vs. OpenGL

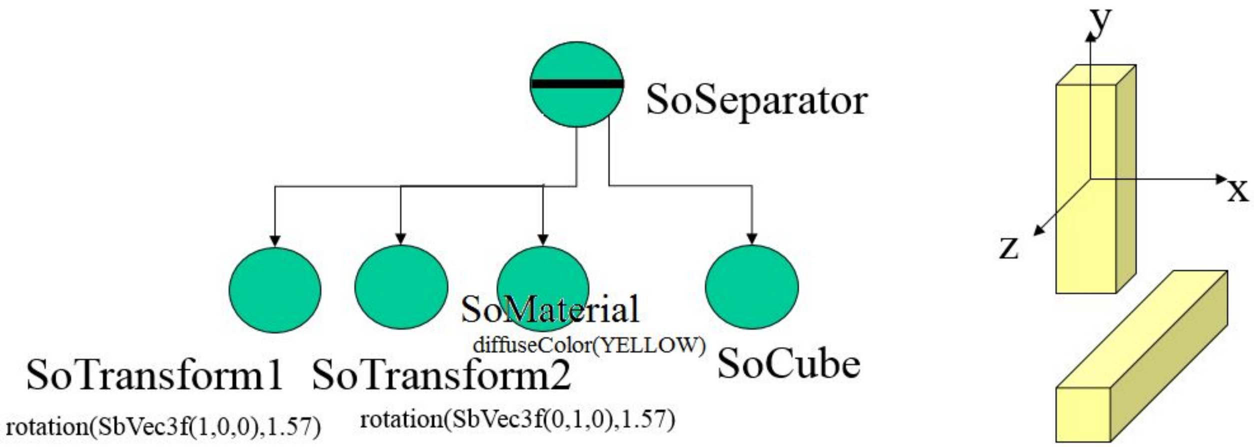
- ✔ OpenGL is not object-oriented. It is state-based, unintuitive.

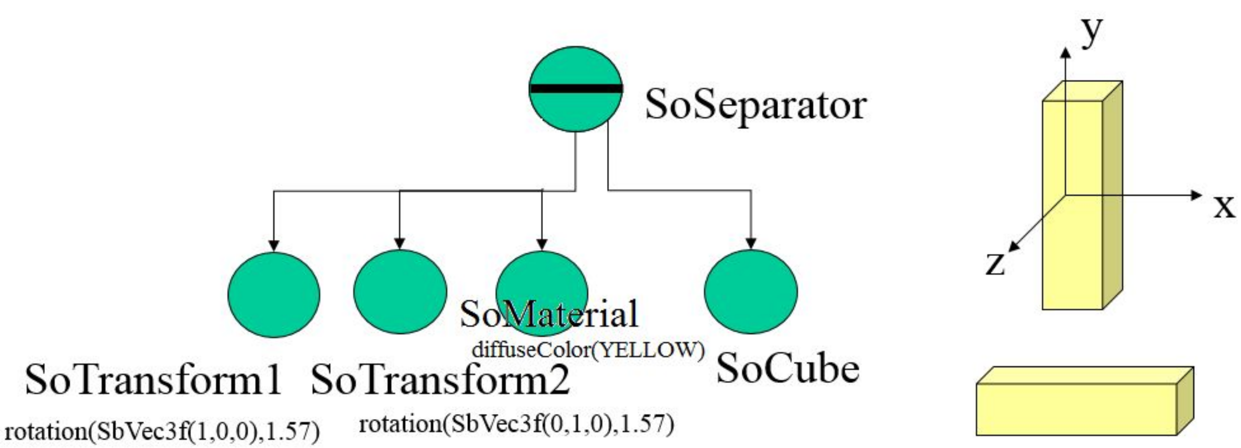
```
void render()
{
    //Clear color buffer
    glClear( GL_COLOR_BUFFER_BIT );
    //////////////////// object # 1 ////////////////////////////
    //Reset modelview matrix STATE
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();
    //Move to center of the screen
    glTranslatef( SCREEN_WIDTH / 2.f, SCREEN_HEIGHT / 2.f, 0.f );
    //Set color to cyan and this applies to everything that follows,
    //i.e., state-based, unintuitive, not object-oriented
    glColor3f( 0.f, 1.f, 1.f );
    glBegin( GL_QUADS );
        glVertex2f( -50.f, -50.f );
        glVertex2f(  50.f, -50.f );
        glVertex2f(  50.f,  50.f );
        glVertex2f( -50.f,  50.f );
    glEnd();
```

```
    //////////////////// object # 2 ////////////////////////////
    //Reset modelview matrix STATE
    glLoadIdentity(); //hope that current matrix mode is glMatrixM
                      //to be sure set it explicitly: glMatrixMode
    //Move right 1.5 units and into the screen 7.0
    glTranslatef(1.5f,0.0f,-7.0f);
    //change color STATE to green
    glColor3f(0.0f,1.0f,0.0f);
    glBegin(GL_QUADS);
        glVertex3f( 1.0f, 1.0f,-1.0f);
        glVertex3f(-1.0f, 1.0f,-1.0f);
        glVertex3f(-1.0f, 1.0f, 1.0f);
        glVertex3f( 1.0f, 1.0f, 1.0f);

        glVertex3f( 1.0f,-1.0f, 1.0f);
        glVertex3f(-1.0f,-1.0f, 1.0f);
        glVertex3f(-1.0f,-1.0f,-1.0f);
        glVertex3f( 1.0f,-1.0f,-1.0f);
    glEnd();
    //Update screen
    glutSwapBuffers();
} //end of render()
```

Open Inventor vs. OpenGL

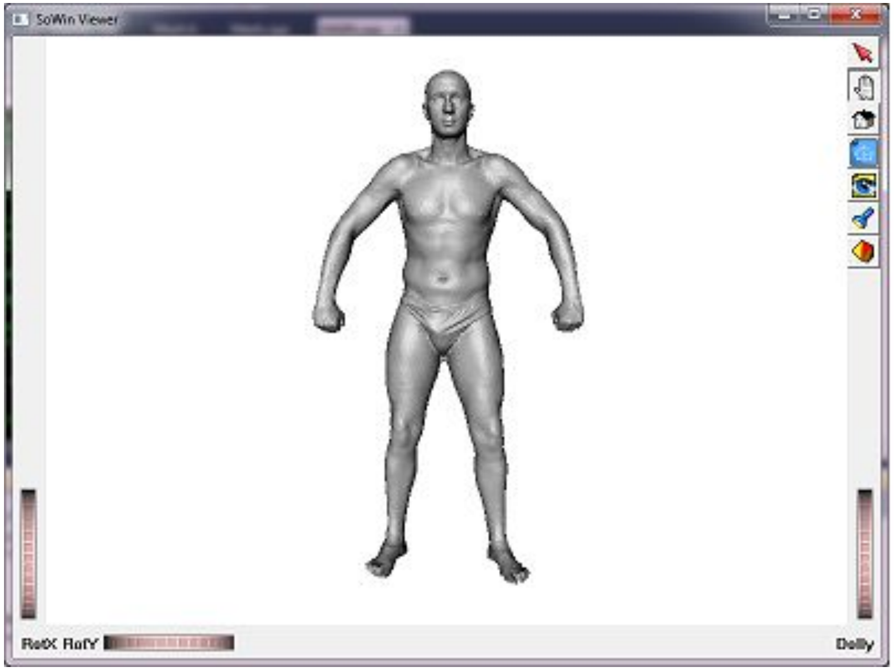
- Open Inventor is object-oriented. Everything on screen is an object (of type SoSeparator) with its own fields/attributes.

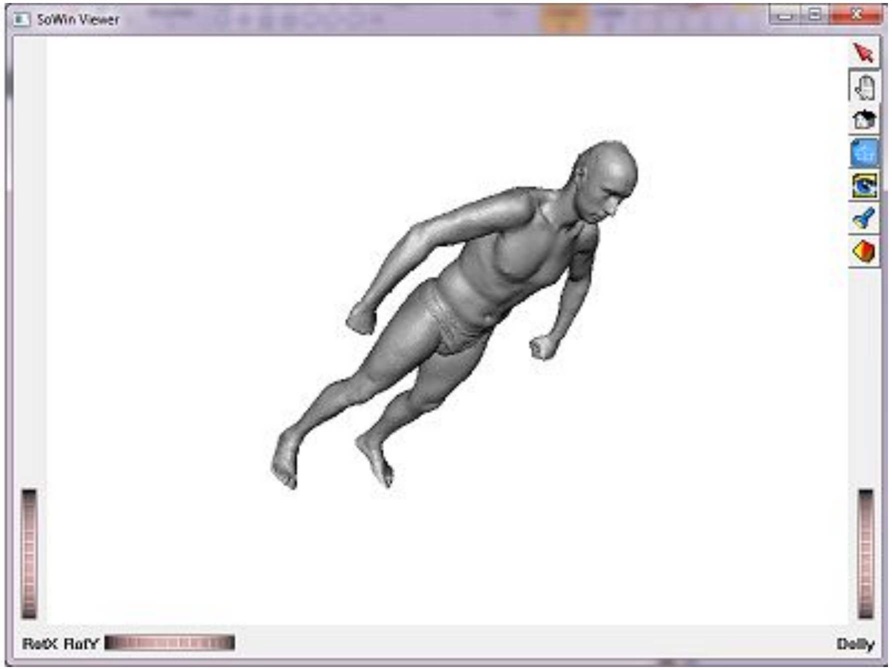




Open Inventor vs. OpenGL

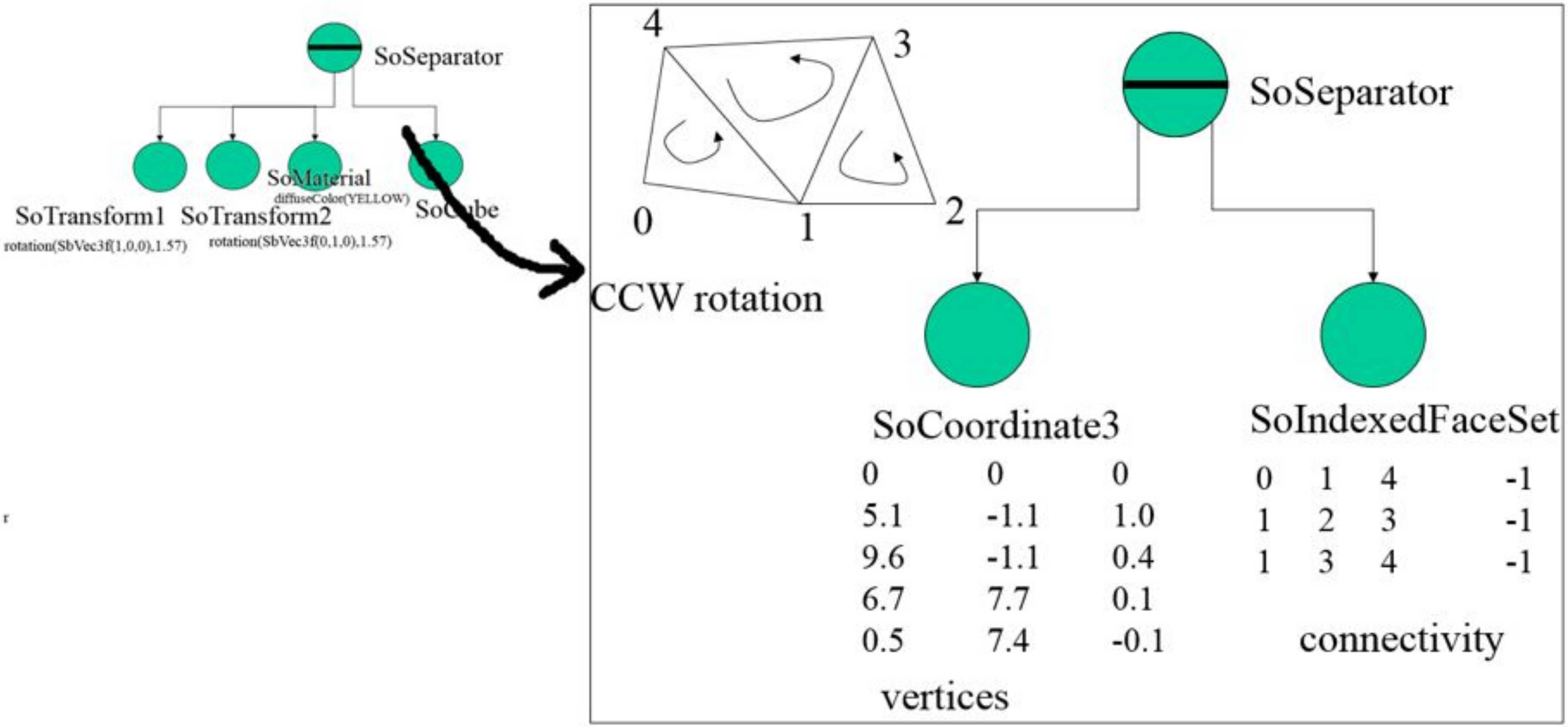
- ✔ OpenGL uses a primitive viewer, glut, where you need to implement your own trackball navigation, camera location/lookups, render modes, etc.
- ✔ Open Inventor uses an advanced viewer (SoWin Windows, SoXt Unix) with built-in trackball navigation, camera handling, render modes, etc.
  - ✔ https://youtu.be/lK7aoc1AO8w





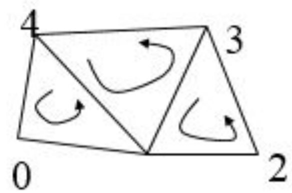
Open Inventor Programming

- ✔ Let's change SoCube with a more generic shape, which is a SoSeparator.



Open Inventor Programming

- ✔ The code is dead simple:



Get this info from MyMesh.cpp for a more structured code!

```
SoSeparator *makeMyObject( ) {
        float myVerts[5][3] = {{0.0, 0.0, 0.0}, {5.1, -1.1, 1.0}, ...};
        int myIndex[12] = {0, 1, 4, -1, ...};

        SoSeparator *myObject = new SoSeparator;
        SoCoordinate3 *myCoords = new SoCoordinate3;
        myCoords->point.setValues(0, 5, myVerts);
        myObject ->addChild(myCoords);
        SoIndexedFaceSet *faceSet = new SoIndexedFaceSet;
        faceSet->coordIndex.setValues(0, 12, myIndex);
         myObject ->addChild(faceSet);
        return myObject;
}
```

Open Inventor Programming

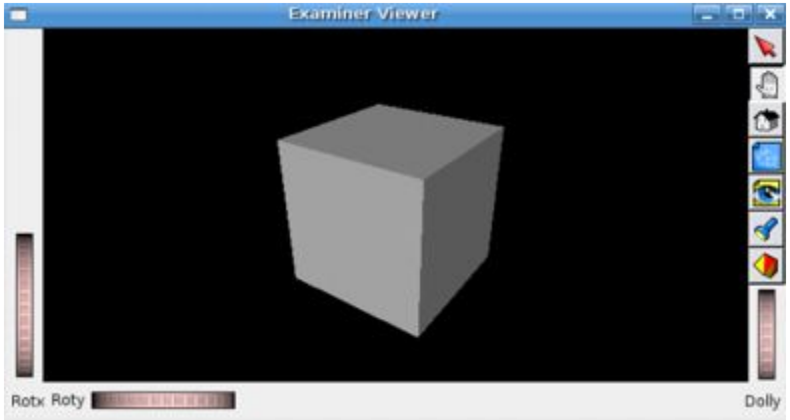
- ✔ The code is dead simple.
  - ✔ You can also set myCoords and faceSet values one-by-one:

```
                                index
myCoords->point.set1Value(0,0,0,0);
myCoords->point.set1Value(1,5.1,-1.1,1.0);
myCoords->point.set1Value(2, 9.6,-1.1,0.4);
...
                                   index
faceSet->coordIndex.set1Value(0,0);
faceSet->coordIndex.set1Value(1,1);
faceSet->coordIndex.set1Value(2,4);
faceSet->coordIndex.set1Value(3,-1);
...
```

Open Inventor Programming

- ✔ Dead simple full code to create and render a cube.

```
#include <Inventor/Qt/SoQt.h>
#include <Inventor/Qt/viewers/SoQtExaminerViewer.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoCube.h>

int main(int argc, char ** argv)
{
  QWidget * mainwin = SoQt::init(argc, argv, argv[0]);

  SoSeparator * root = new SoSeparator;
  root->ref();
  SoCube *cube = new SoCube;
  root->addChild(cube);

  SoQtExaminerViewer * eviewer = new
SoQtExaminerViewer(mainwin);
  eviewer->setSceneGraph(root);
  eviewer->show();
  SoQt::show(mainwin);
  SoQt::mainLoop();

  root->unref();
  delete eviewer;
  return 0;
}
```



Potential Project Topics

- ✔ Implement all schemes in Slide 9 to generate subdivision surfaces.
- ✔ Implement paper: Progressive Meshes (Slide 11).