CENG 789 – Digital Geometry Processing

02- Mesh Data Structures

Asst. Prof. Yusuf Sahillioğlu

Computer Eng. Dept., Middle East Technical University, Turkey
Mesh Data Structures

- Polygon mesh: set of *polygons* embedded in 2D or 3D.

- Polygon mesh: set of *vertices, edges, and faces* embedded in 2D or 3D.

- Let's handle these vertices, edges, faces in a structured way!
Mesh Data Structures

✓ How to store geometry & connectivity of a mesh.

✓ 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 Structures

✓ 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.
Face-Based Data Structures

- **Face-Set data structure.** //aka polygon soup 'cos no connectivity info.

<table>
<thead>
<tr>
<th>Triangles</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_{11} \ y_{11} \ z_{11}$</td>
</tr>
<tr>
<td>$x_{21} \ y_{21} \ z_{21}$</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>$x_{F1} \ y_{F1} \ z_{F1}$</td>
</tr>
</tbody>
</table>

//vertices and associated data are replicated.

- Indexed face-set data structure (obj, off, ply formats). **Better!!!!**
**Edge-Based Data Structures**

- 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)
Open Inventor vs. OpenGL

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

```c
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();
    //Reset modelview matrix STATE
    glLoadIdentity(); //hope that current matrix mode is glMatrixMode
    //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();
}
```
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 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.
Let’s change SoCube with a more generic shape, which is a SoSeparator.
The code is dead simple:

```cpp
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;
}
```
The code is dead simple.

You can also set myCoords and faceSet values one-by-one:

```cpp
myCoors->point.set1Value(0,0,0,0);
myCoors->point.set1Value(1,5.1,-1.1,1.0);
myCoords->point.set1Value(2, 9.6,-1.1,0.4);
...
```

```cpp
faceSet->coordIndex.set1Value(0,0);
faceSet->coordIndex.set1Value(1,1);
faceSet->coordIndex.set1Value(2,4);
faceSet->coordIndex.set1Value(3,-1);
...
```
Dead simple full code to create and render a cube.

```cpp
#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;
}
```