CSC 4356
Interactive Computer Graphics
Lecture 10: 3D Objects

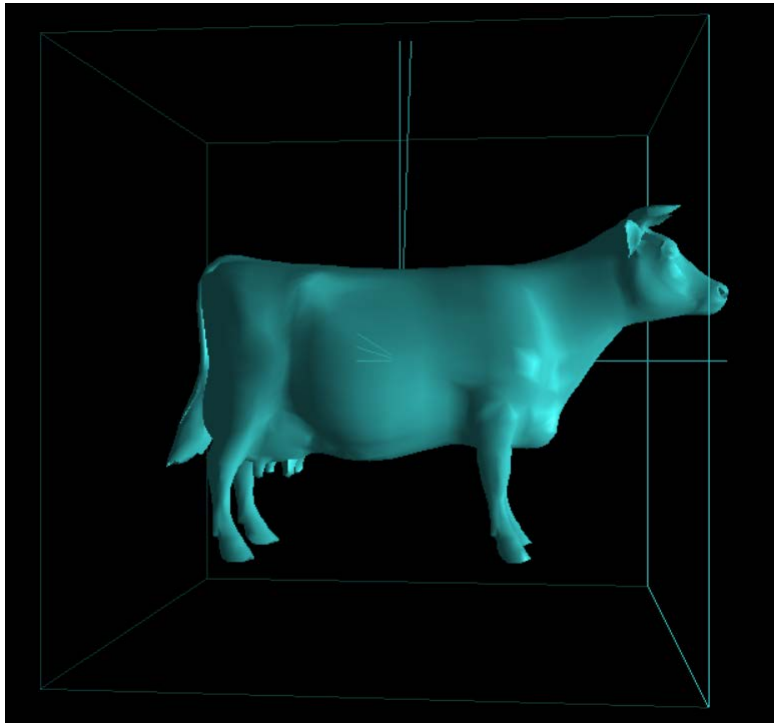
Jinwei Ye

http://www.csc.lsu.edu/~jye/CSC4356/

Tue & Thu: 10:30 - 11:50am
218 Tureaud Hall

Lecture 10: 3D Objects

- 3D object representation
- 3D data file format
- 3D data parsing



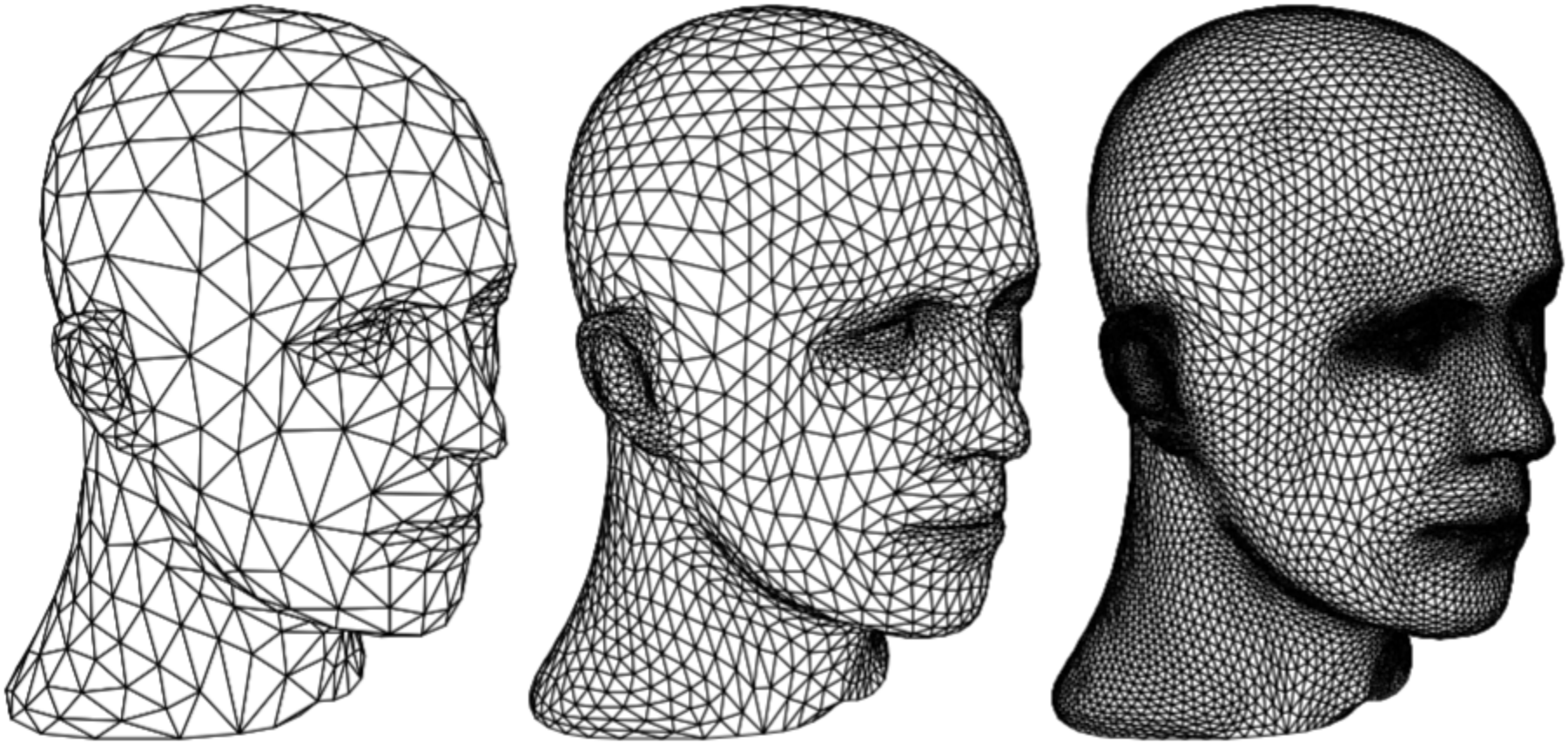
How To Specify 3D Objects?

- Mathematical equations
  - Explicit functions: z = f(x,y)
  - Implicit functions: f(x,y,z) = 0
  - Parametric functions: (x(u,v), y(u,v), z(u,v))
- Build up from simple primitives
  - Points: point cloud
  - Lines: wire frame
  - Planes: solid surface

Simple Planes

- Object surfaces are modeled as connect planar facets
  - N (>3) vertices, each with 3 coordinates
  - Minimally a triangle

Surface Subdivision



Face Specification

- Face or Facet

  Face: [v0.x, v0.y, v0.z] [v1.x, v1.y, v1.z] [v2.x, v2.y, v2.z] … [vN.x, vN.y, vN.z]

- Sharing vertices via indirection

  v0 = [v0.x, v0.y, v0.z]

  v1 = [v1.x, v1.y, v1.z]

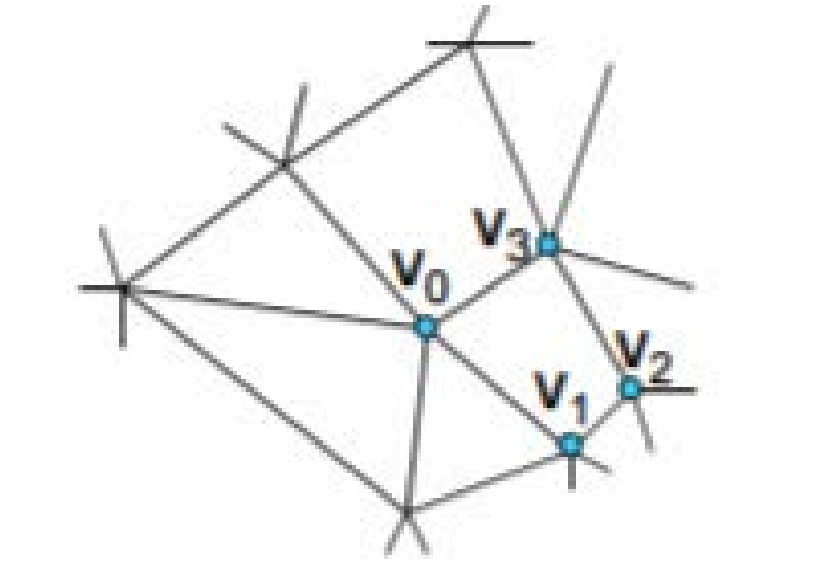
  v2 = [v2.x, v2.y, v2.z]

  :

  :

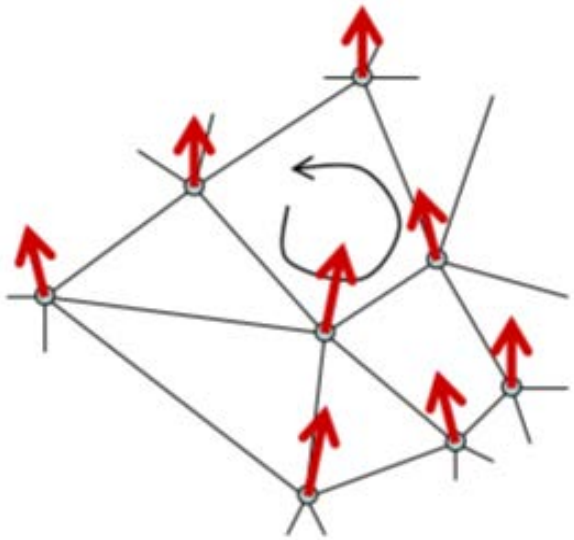
  v3 = [vN.x, vN.y, vN.z]

  Face v0, v1, v2, … vN



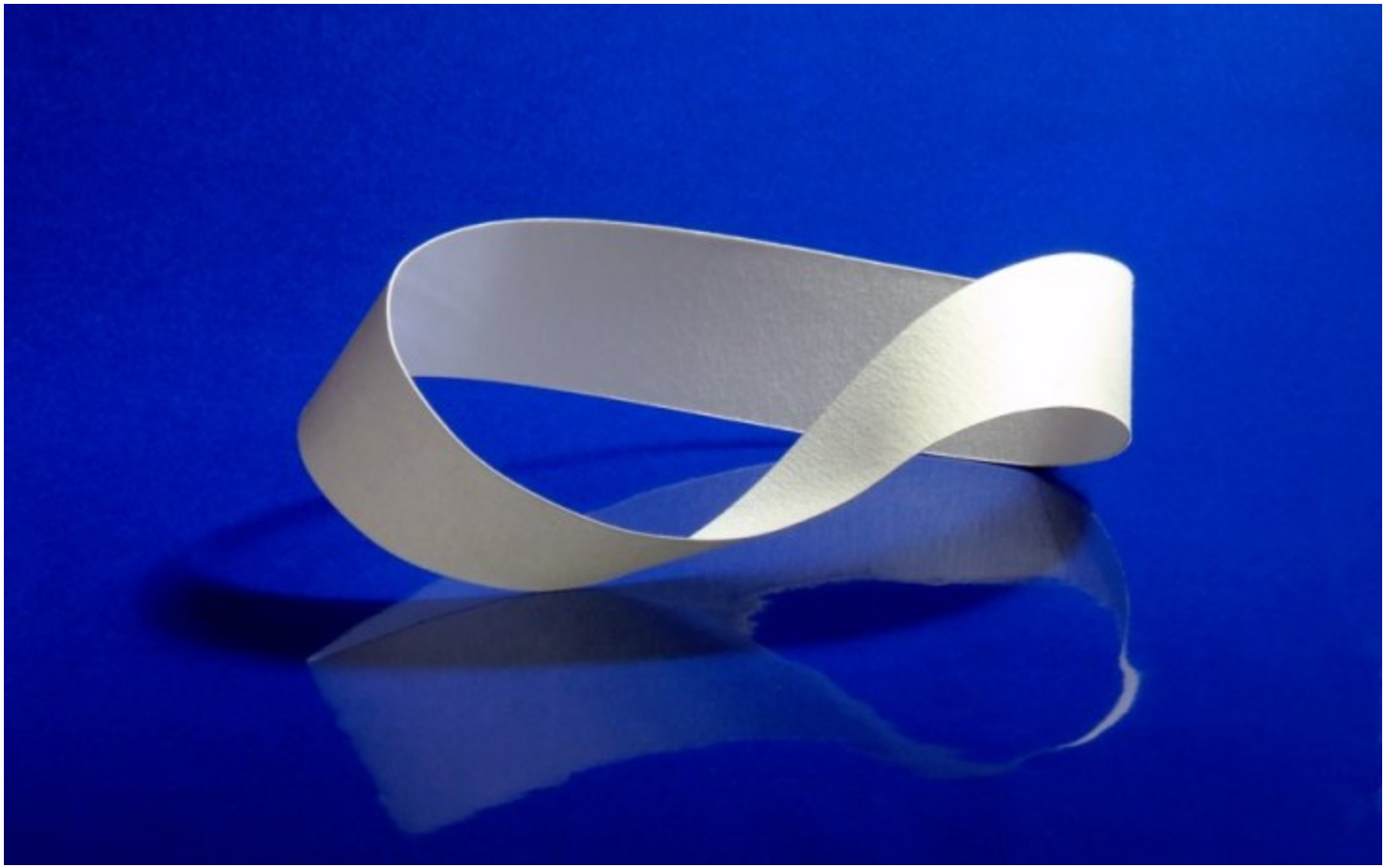
Vertex Specification

- Where
  - Geometric coordinates [x, y, z]
- What Color
  - Color values [r, g, b]
  - Texture Coordinates [u, v]
- Orientation
  - Inside vs. Outside
  - Encoded implicitly in ordering or specify by normal



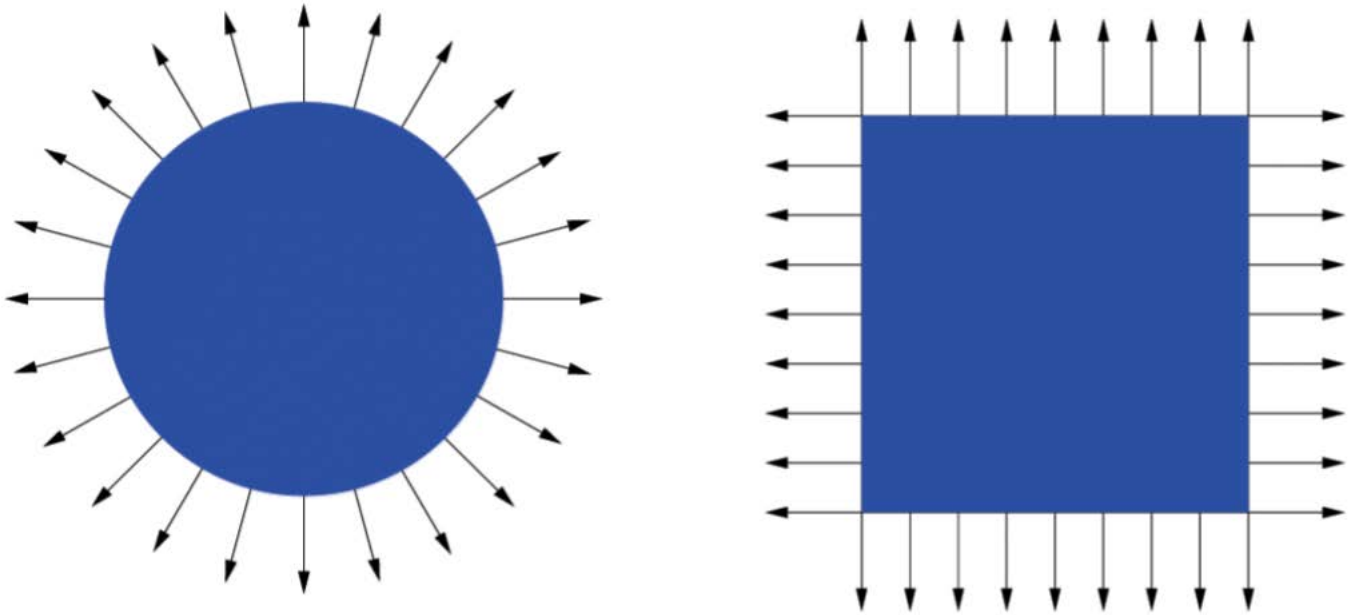
Do all smooth surfaces maintain consistence orientation?

- Mobius Strip - unorientable



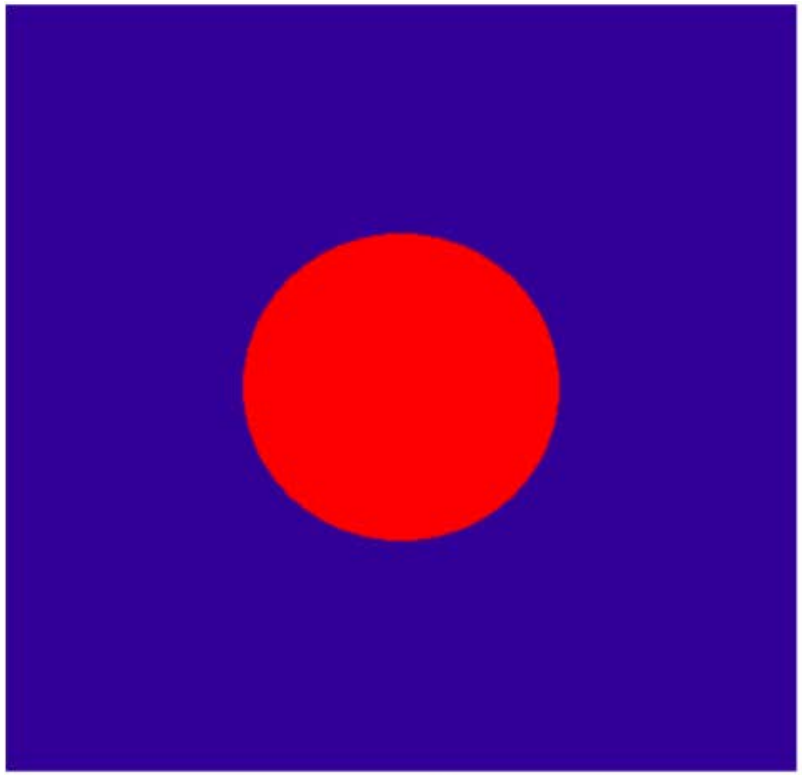
Normal Vector

- Normal is a unit vector perpendicular to the surface

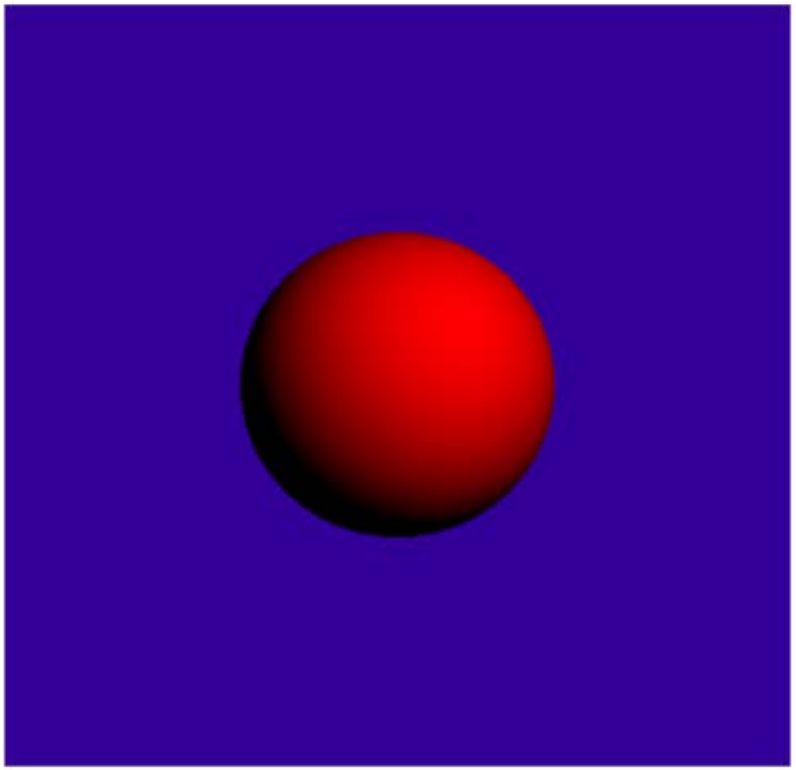


Why Normal Important?

- Make things look 3D!



object color only



Diffuse Shading

How to compute normal?

- Given surface function z = f(x,y)
  - Compute from the derivative ([dx,dy,dz]):

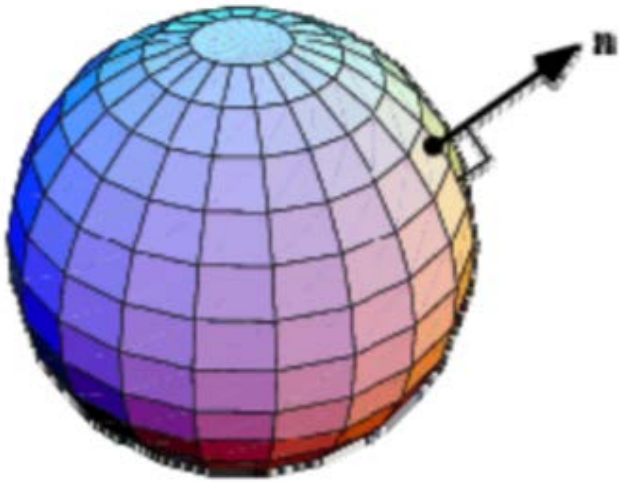
$$\vec{n} = [\frac{dz}{dx}, \frac{dz}{dy}, -1]$$

- Given three vertices of a surface:

$$\vec{n} = (v_1 - v_0) \times (v_2 - v_0)$$

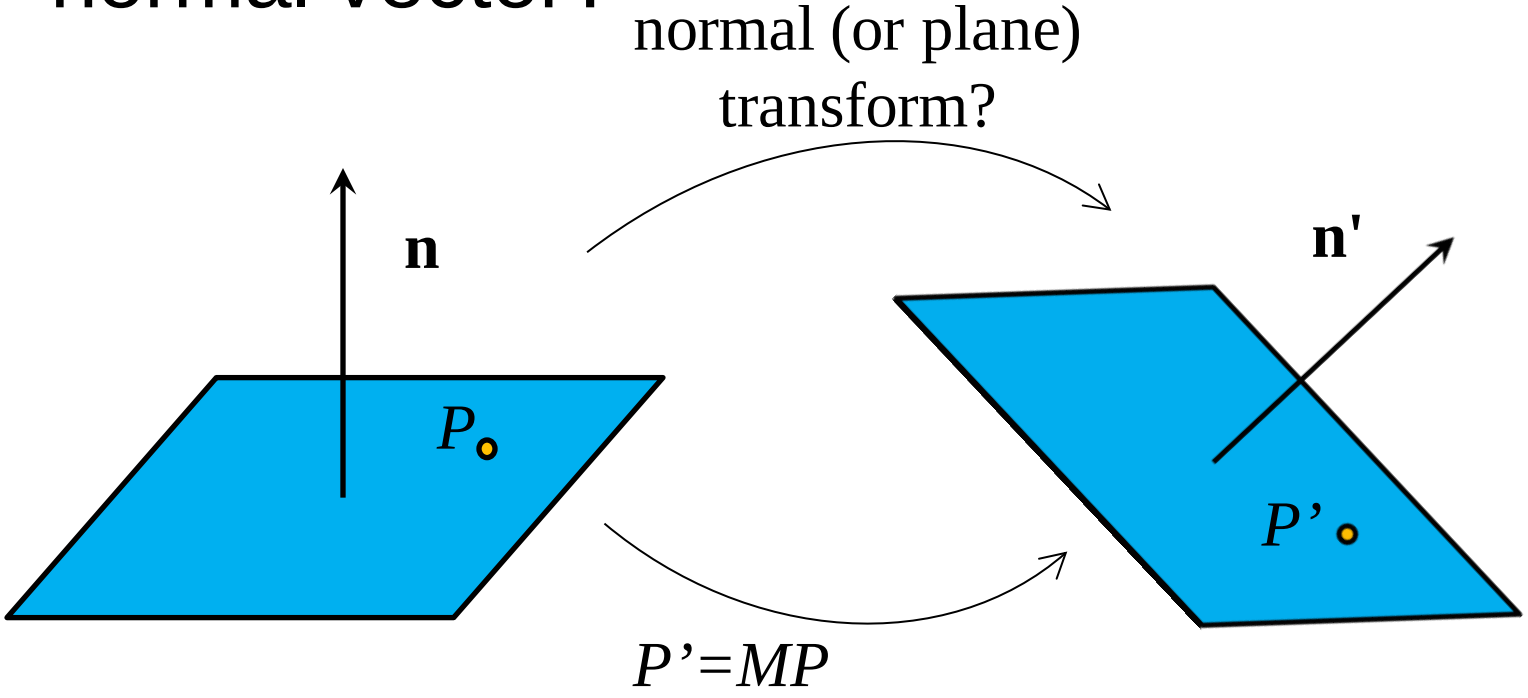
- Normalized

$$\hat{n} = \frac{\vec{n}}{\sqrt{n_x^2 + n_y^2 + n_z^2}}$$



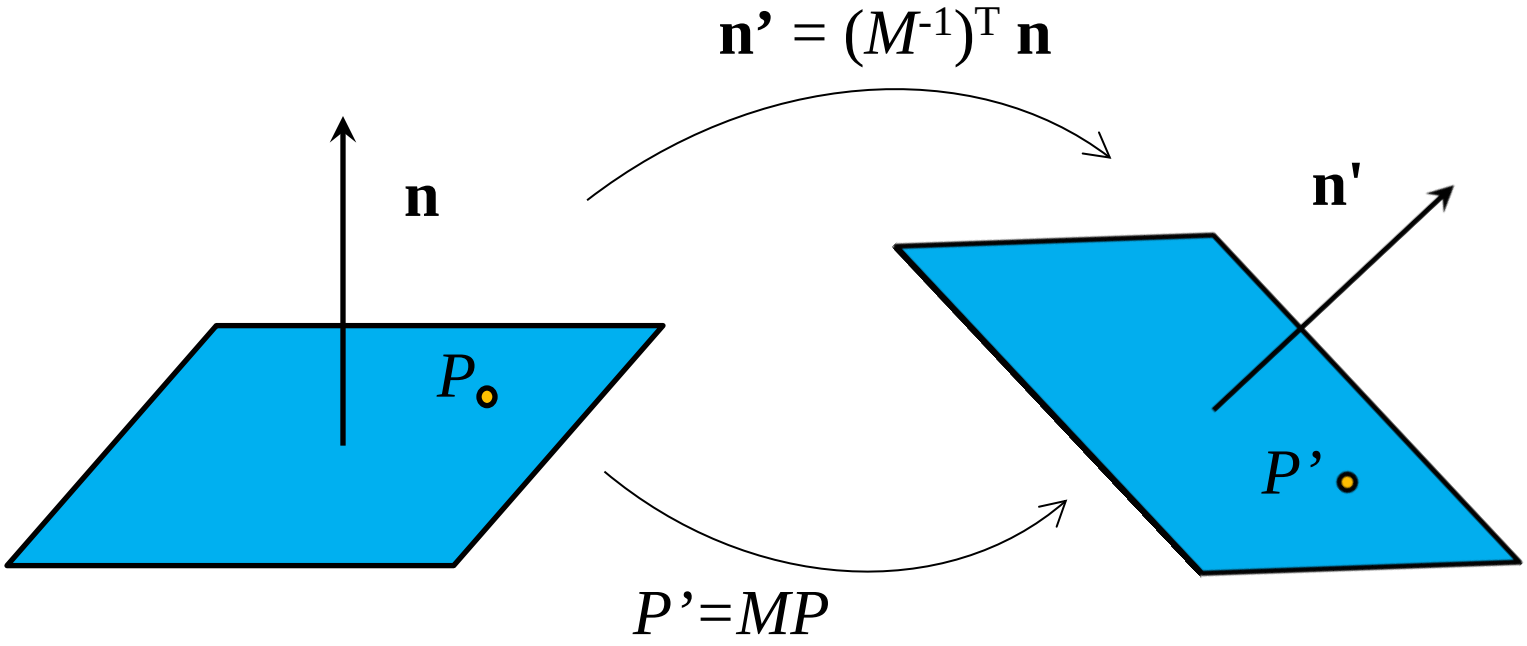
How to transform normal?

- Given a plane, and a transformation M that transforms every point on the plane, what should be the transformation apply on the normal vector?



How to transform normal?

- Given a plane, and a transformation M that transforms every point on the plane, what should be the transformation apply on the normal vector?



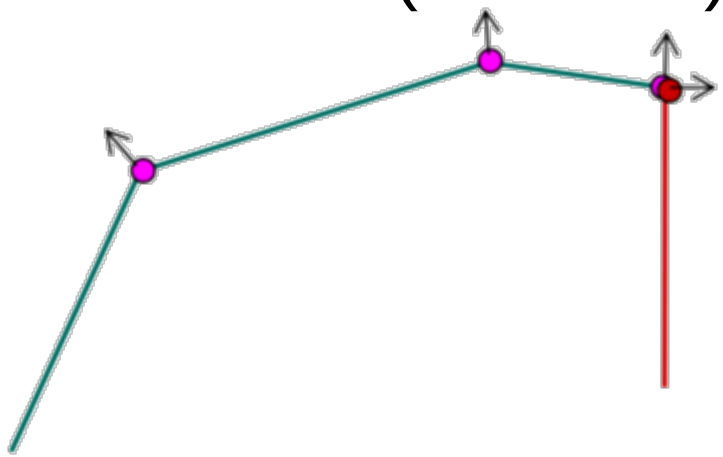
Drawing Faces in OpenGL

```
glBegin(GL_POLYGON);
   foreach (Vertex v in Face) {
        glColor4d(v.red, v.green, v.blue, v.alpha);
        glNormal3d(v.norm.x, v.norm.y, v.norm.z);
        glTexCoord2d(v.texture.u, v.texture.v);
        glVertex3d(v.x, v.y, v.z);
    }
glEnd();
```

- Heavy-weight model
  - All information about a vertex is stored
- Redundant
  - Vertex positions often shared by at least 3 faces
  - Vertex attributes are often face attributes
    e.g. normal, texture, color

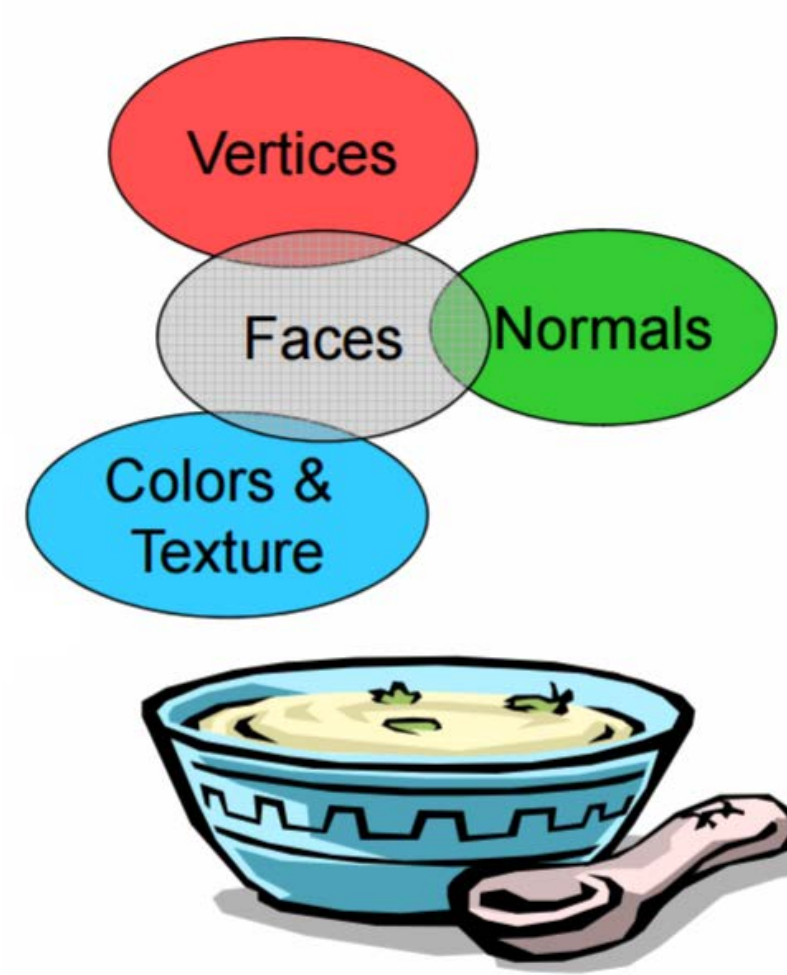
Decoupling Vertex and Face Attributes via Indirection

- Works for many cases
  - Used with vertex array or vertex buffer objects in OpenGL
- Exceptions:
  - Regions where the surface changes materials
  - Regions of high curvature (a crease)



Polygon Soup

- A collection of
  - Vertices
  - Normals
  - Colors & Textures
- Connected by “facets”



3D File Formats

- MAX – Studio Max
- DXF – AutoCAD
  - Supports 2-D and 3-D; binary
- 3DS – 3D studio
  - Flexible; binary
- VRML – Virtual reality modeling language
  - ASCII
- OBJ – Wavefront OBJ format
  - ASCII
  - Human readable
  - Extremely simple
  - Widely supported

OBJ Basics

- The most common Wavefront obj file tokens are listed below:

  # some text

  Rest of line is a comment

  v float float float

  A single vertex's geometric position in space. The first vertex listed in the file has index 1, and subsequent vertices are numbered sequentially.

  vn float float float

  A normal. The first normal in the file is index 1, and subsequent normals are numbered sequentially.

  vt float float

  A texture coordinate. The first texture coordinate in the file is index 1, and subsequent textures are numbered sequentially.

OBJ Face Specification

f int int int ... (vertex only)

or

f int/int int/int int/int . . . (vertex & texel)

or

f int/int/int int/int/int int/int/int … (vertex, texel, &normal)

or

f int//int int//int int//int … (vertex & normal)

- The arguments are 1-based indexes into the arrays of vertex positions, texture coordinates, and normals respectively.
- There is no maximum number of vertices
- Each face must be flat and convex.

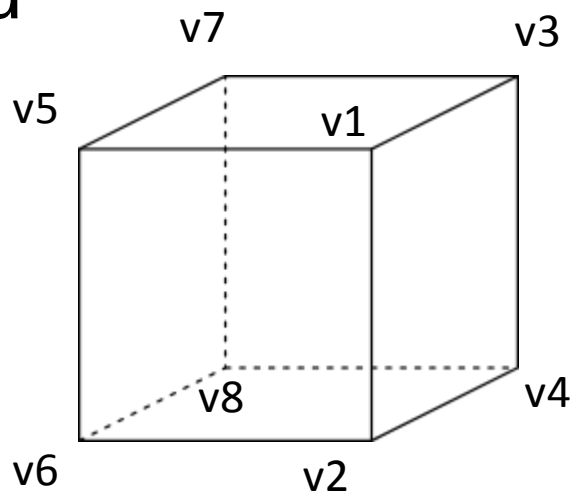
OBJ Extras

- **g string**
  - group specification where string label indicates the following primitives within the same group. This is really the only hint you get for coloring

- **s int**
  - smoothing group specification where int ID indicates the following primitives are smooth (the vertices can share common normals). Used if normals must be estimated.

OBJ Example

- Vertices followed by faces
  - Faces reference previous vertices by integer index
  - Co-planarity of vertices is assumed




```
# A simple
cube
v 1 1 1
v 1 1 -1
v 1 -1 1
v 1 -1 -1
v -1 1 1
v -1 1 -1
v -1 -1 1
v -1 -1 1
v -1 -1 -1

f 1 2 4 3
f 5 7 8 6
f 1 5 6 2
f 3 4 8 7
f 1 3 7 5
f 2 6 8 4
```

OBJ Sources

- free3d.com
- www.3dcafe.com
- Others
- Most modeling programs will export .obj files
- Most rendering packages will read in .obj files

Code: Vertex

```
class Vertex {
public:
    double x, y, z;

    Vertex( double x, double y,
double z ) {
        setCoordinates( x, y, z );
    }
    void setCoordinates( double xval,
double yval, double zval ) {
        x = xval; y = yval; z = zval;
    }
};
```

Normal

```
class Normal {
public:
    double x, y, z;

    Normal ( double x, double y, double z ) {
        setCoordinates( x, y, z );
    }
    void setCoordinates(double xval, double yval,
double zval) {
        double l = sqrt(xval*xval+yval*yval+zval*zval);
        if (l != 0.0)
            l = 1.0 / l;
        x = l*xval;
        y = l*yval;
        z = l*zval;
    }
};
```

Texel

```
class Texel {
public:
    double u, v;

    Texel(double u, double v) {
        setCoordinates(u, v);
    }
    void setCoordinates(double uval, double vval)
{
        u = uval;
        v = vval;
    }
};
```

Face

```
class Face {
public:
    int *vList;
    int *nList;
    int *tList;
    int vIndex;
    const int DEFAULT_SIZE;
    int current_max_size;

    Face(): DEFAULT_SIZE(4) {
        vIndex = -1;
    }
    void addVertex(int v) {
        // make indices zero referenced
        add(v-1, -1, -1);
    }
    void addVertexTexel(int v, int t) {
        add(v-1, -1, t-1);
    }
    void addVertexNormal(int v, int n) {
        add(v-1, n-1, -1);
    }
    void addVertexNormalTexel(int v, int n, int t) {
        add(v-1, n-1, t-1);
    }
```

Face (Cont.)

```
void add(int v, int n, int t) {
        if (vIndex < 0) {
            vList = new int[DEFAULT_SIZE];
            nList = new int[DEFAULT_SIZE];
            tList = new int[DEFAULT_SIZE];
            current_max_size = DEFAULT_SIZE;
            vIndex = 0;
        }
        vList[vIndex] = v;
        nList[vIndex] = n;
        tList[vIndex] = t;
        vIndex += 1;
        if (vIndex == current_max_size) {
            current_max_size = 2*vIndex;
            int *newV = new int[current_max_size];
            int *newN = new int[current_max_size];
            int *newT = new int[current_max_size];
            for ( int i = 0; i < vIndex; i++ ) {
                newV[i] = vList[i];
                newN[i] = nList[i];
                newT[i] = tList[i];
            }
            delete [] vList;
            delete [] nList;
            delete [] tList;
            vList = newV;
            nList = newN;
            tList = newT;
        }
    }
};
```

vList, nList and tList are dynamic arrays

WavefrontOBJ Class

```
class WavefrontObj  {
public:
    Vertex **v;
    int vIndex;
    int current_max_verticies;

    Normal **n;
    int nIndex;
    int current_max_normals;

    Texel **t;
    int tIndex;
    int current_max_texels;

    Face **f;
    int fIndex;
    int current_max_faces;

    bool isFlat;
    GLuint mode;

    const int DEFAULT_SIZE = 16;
```

Here’s the soup bowl

WavefrontOBJ Constructor

Setup for a simple paser

```
WavefrontObj(char *filename) : DEFAULT_SIZE(16) {
    vIndex = -1;
    nIndex = -1;
    tIndex = -1;
    fIndex = -1;

    isFlat = false;
    mode = GL_POLYGON;

    char *line = new char[200];
    char *line_back = new char[200];
    char wspace[] = {' ', '\t'};
    char separator[] = {'/'};
    char *token;

    ifstream file(filename);
    if ( !file ) {
       cerr <<"Cannot open file: " <<filename <<" exiting." <<endl;
       exit ( -1 );
    }
    while ( !file.eof() ) {
       file.getline( line, 199 );
       // first, strip off comments
       if ( line[0] == '#' )
          continue;
       else if ( !strcmp( line, "" ) )
          continue;
       else {
                //parse the line…
```

WavefrontOBJ Constructor

Setup for a simple parser

```
WavefrontObj(char *filename) : DEFAULT_SIZE(16)
{
        vIndex = -1;
        nIndex = -1;
        tIndex = -1;
        fIndex = -1;

        isFlat = false;
        mode = GL_POLYGON;

        char *line = new char[200];
        char *line_back = new char[200];
        char wspace[] = {' ', '\t'};
        char separator[] = {'/'};
        char *token;

        ifstream file(filename);
        if ( !file ) {
            cerr <<"Cannot open file: "
<<filename <<" exiting." <<endl;
            exit ( -1 );
        }
        while ( !file.eof() ) {
            file.getline( line, 199 );
            // first, strip off comments
            if ( line[0] == '#' )
                continue;
            else if ( !strcmp( line, "" ) )
                continue;
            else {
```

WavefrontOBJ Constructor

```
else if ( !strcmp( token, "f" ) ) {
    Face *f = addFace();
    for (char *p = strtok( NULL, wspace ); p; p = strtok( NULL, wspace ) ) {
        indices[0] = -1;
        indices[1] = -1;
        indices[2] = -1;
        int i = 0;
        for ( int j = 0 ; j < strlen( p ); j++ ) {
            if ( p[j] != '/' ) {
                if ( indices[i] == -1 )
                    indices[i] = 0;
                indices[i] *= 10;
                char c[2];
                c[0] = p[j];
                c[1] = '\0';
                indices[i] += atoi( c );
            }
            else {
                i++;
            }
        }

        if ( (indices[1] == -1) & (indices[2] == -1) ) {
            f->addVertex(indices[0]);                                    // num//
        }
        else  if ( indices[2] == -1 ) {
            f->addVertexTexel(indices[0], indices[1]);                   // num/num/
        }
        else if ( indices[1] == -1 ) {
            f->addVertexNormal( indices[0], indices[2] );                // num//num
        }
        else {
            f->addVertexNormalTexel(indices[0], indices[1], indices[2]); // num/num/num
        }
    }
```

WavefrontOBJ addVertex()

```
void addVertex(Vertex *vert) {
    if (vIndex < 0) {
        v = new Vertex*[DEFAULT_SIZE];
        vIndex = 0;
        current_max_verticies = DEFAULT_SIZE;
    }
    v[vIndex] = vert;
    vIndex += 1;
    if (vIndex == current_max_verticies) {
        current_max_verticies = 2*vIndex;
        Vertex **newV = new Vertex*[current_max_verticies];
        for ( int i = 0; i < vIndex; i++ )
            newV[i] = v[i];
        delete [] v;
        v = newV;
    }
}
```

addNormal(), addTexel(), and addFace() are silimar:

Creating dynamic arrays

WavefrontOBJ Draw()

```
void Draw() {
    int face, vertex, i;
    for (face = 0; face < fIndex; face++) {
        Face *currentFace = f[face];
        glBegin(mode);
        for (vertex = 0; vertex < currentFace->vIndex; vertex++) {
            if (isFlat) {
                if (vertex == 0) {
                    Normal *norm = faceNormal(v[currentFace->vList[0]], v[currentFace->vList[1]]
                    glNormal3d(norm->x, norm->y, norm->z);
                }
            }
            else if ((i = currentFace->nList[vertex]) >= 0) {
                if ( i < nIndex )
                    glNormal3d(n[i]->x, n[i]->y, n[i]->z);
                else  {
//                      cerr <<"Error i = " <<i <<" nIndex = " <<nIndex <<endl;
                }
            }
            else if (vertex == 0) {
                Normal *norm = faceNormal(v[currentFace->vList[0]], v[currentFace->vList[1]], v[
                currentFace->nList[0] = nIndex;
                addNormal(norm);
                glNormal3d(norm->x, norm->y, norm->z);
            }
            if ((i = currentFace->tList[vertex]) >= 0) {
                glTexCoord2d(t[i]->u, t[i]->v);
            }
            i = currentFace->vList[vertex];
            if ( i < vIndex )
                glVertex3d(v[i]->x, v[i]->y, v[i]->z);
            else {
//                  cerr <<"Error i = " <<i <<" vIndex = " <<vIndex <<endl;
            }
        }
        glEnd();
    }
}
```