Building a 3D World

- Specifying 3D geometry
- An “OBJ”ect parser
- Setting up a scene

Lecture 7

CISC440/640

Spring 2015

COMPUTER GRAPHICS

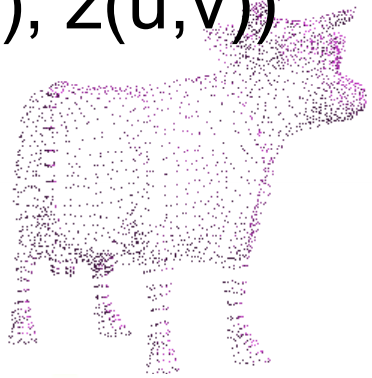
Department of Computer and Information Science

Welcome to 3D!

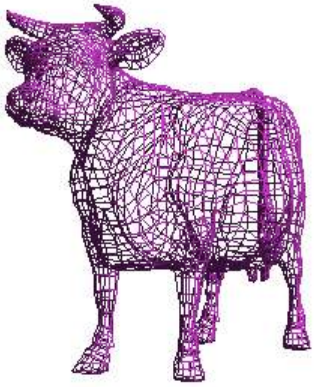
- But first, three ways to write a program
  - Minimal – the smallest possible fragment of working code that demonstrates the key concept
  - Efficient – the fastest possible code tuned for the best performance
  - Well Designed – Robust and maintainable
- Today, a tool for your bag of tricks
  - Useful for the next few lectures and your next project too!

Primitive 3D

- How do we specify 3D objects?
  - Simple mathematical functions, z = f(x,y)
  - Parametric functions, (x(u,v), y(u,v), z(u,v))
  - Implicit functions, f(x,y,z) = 0



- Build up from simple primitives
  - Point – nothing really to see
  - Lines – nearly see through
  - Planes – a surface



Simple Planes

- Surfaces modeled as connected planar facets
  - N (>3) vertices, each with 3 coordinates
  - Minimally a triangle

Specifying a Face

- 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

  Vertex[0] = [v0.x, v0.y, v0.z]

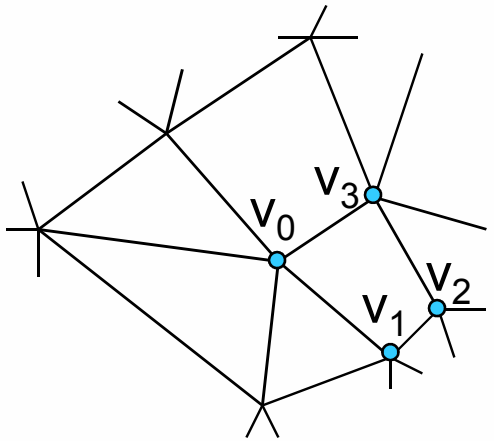
  Vertex[1] = [v1.x, v1.y, v1.z]

  Vertex[2] = [v2.x, v2.y, v2.z]

  :

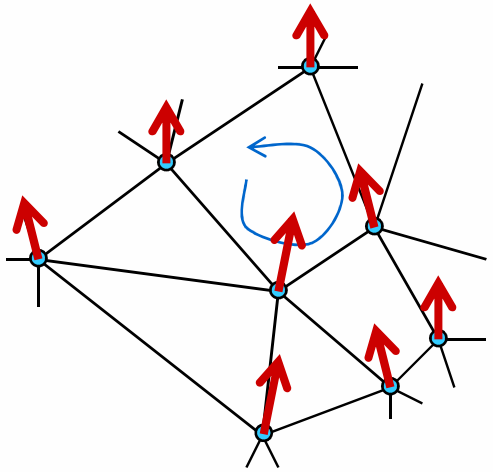
  Vertex[N] = [vN.x, vN.y, vN.z]

  Face v0, v1, v2, … vN



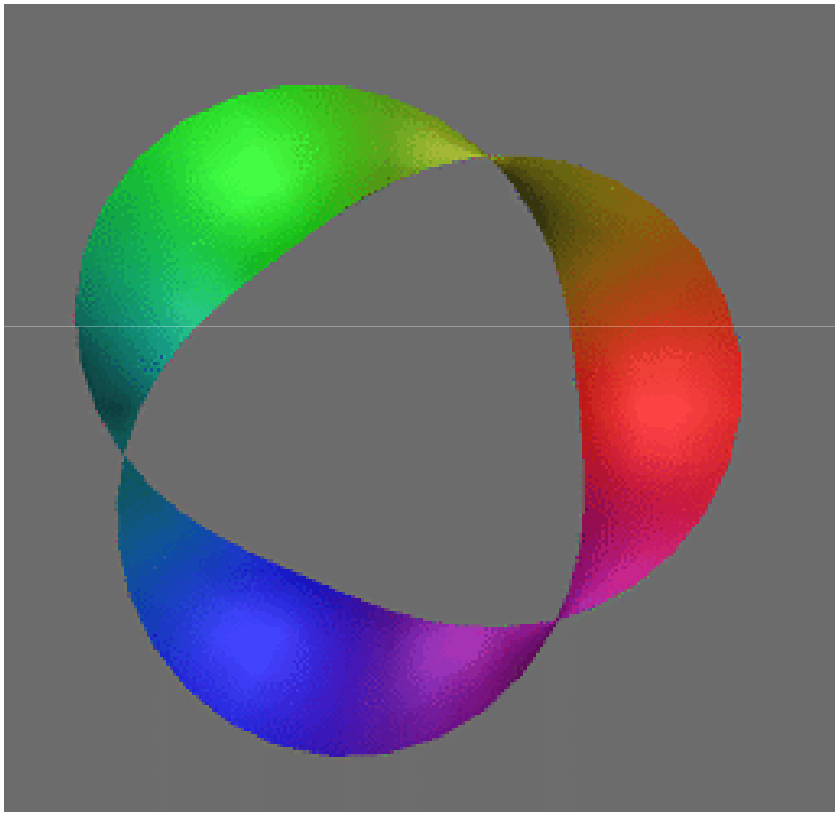
Vertex Specification

- 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
- Geometry Nearby
  - Often we'd like to "fake" a more complex shape than our true faceted (piecewise-planar) model
  - Required for lighting and shading in OpenGL



Do all smooth surfaces maintain consistent orientations?

- Mobius Strip

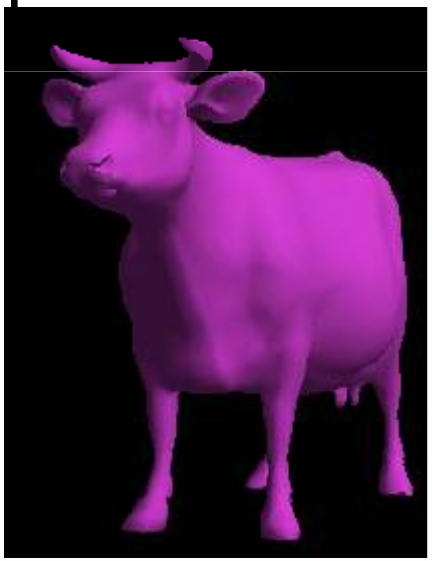


Smoothing things over

- Normals
  - First-Order Taylor-series approximation of surface
  - Normals provide derivative information
  - A unit-vector perpendicular to the actual surface a the specified vertex
  - 3 coordinates – 2 degrees of freedom

$$[n_x, n_y, n_z]$$

  - Normalized $\hat{n} = \frac{[n_x, n_y, n_z]}{\sqrt{n_x^2 + n_y^2 + n_z^2}}$



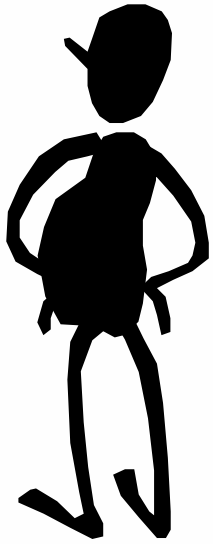
Drawing Faces in OpenGL

```
glBegin(GL_POLYGON);
foreach (Vertex v in Facet) {
  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();
```

A heavyweight vertex model:

All information about a vertex is stored

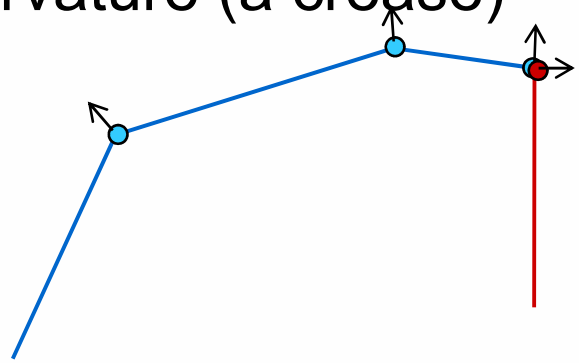
Redundancy- Generally, a vertex "position" is shared by at least 3 faces



Many vertex properties are often "face" features, (i.e. normals, texture, color)

Decoupling Vertex and Face Features

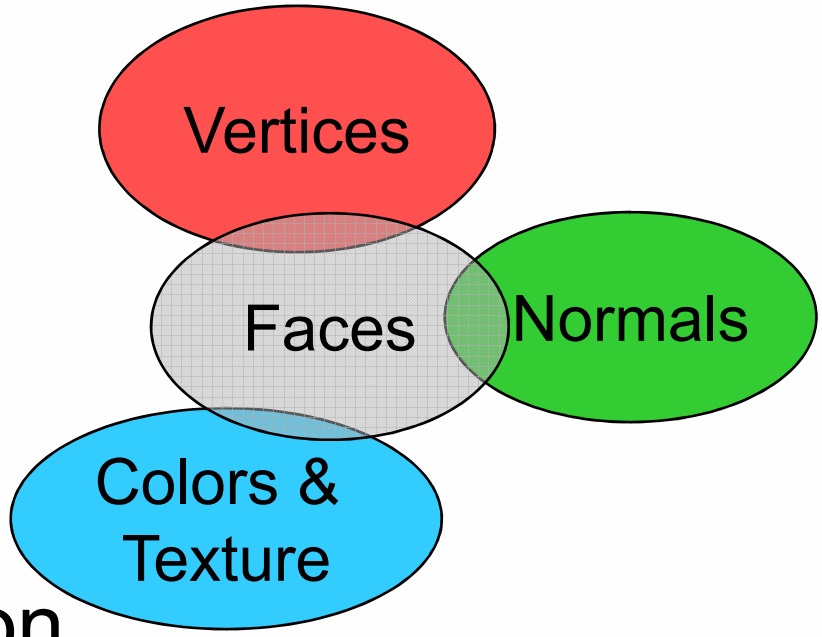
- Case for:
  - Most of the time vertices will be consistent
  - There are exceptions, however
  - Where the surface changes materials
  - Or has a high curvature (a crease)



- This is possible with ‘Heavyweight’ vertices, but less efficient

Polygon Soup

- A collection of
  - Vertices
  - Normals
  - Colors
- Connected by “facets”
- File format specification



Not Invented Here

- 3D object file formats
  - Typical Textbooks invent something
  - MAX – Studio Max
  - DXF – AutoCAD supports 2-D and 3-D, binary
  - 3ds – 3D studio, very flexible, binary
  - obj – Wavefront OBJ format
    - Widely supported
    - ASCII – Human readable (and writeable)
    - Minimal support for shading
    - VRML – Basically a clone

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 Varieties

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)

A polygonal facet. The arguments are indexes into the arrays of vertex positions, texture coordinates, and normals respectively. A number may be omitted if, for example, texture coordinates are not being defined in the model. There is no maximum number of vertices that a single polygon may contain. The .obj file specification says that 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
  - 1-based
  - 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
f 1 3 4 2
f 5 6 8 7
f 1 2 6 5
f 3 7 8 4
f 1 5 7 3
f 2 4 8 6
```

OBJ sources

- Avalon – Viewpoint (http://avalon.viewpoint.com/) old standards
- 3D Café – (http://www.3dcafe.com/asp/meshes.asp) Nice thumbnail index
- Others
- Most modeling programs will export .OBJ files
- Most rendering packages will read in .OBJ files

Code: 3D Vertex

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

Normal

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

    Normal( ) {
    }

    Normal(double xval, double yval, double zval) {
        setCoordinates(xval, yval, zval);
    }

    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;
    }
};
```

Texels

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

        Texel( ) {
        }

        Texel(double uval, double vval) {
                setCoordinates(uval, vval);
        }

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

Faces

```
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);
    }
```

Very simple code

Faces continued

```
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
        int *newN = new int[current_max
        int *newT = new int[current_max
        for ( int i = 0; i < vIndex; i+
            newV[i] = vList[i];
            newN[i] = nList[i];
            newT[i] = tList[i];
        }
```

Mostly simple code:

Only trick:
vList, nList, and tList are Dynamic arrays

WavefrontOBJ class

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

        Normal ** n;
        int nIndex;

        Texel ** t;
        int tIndex;

        Face ** f;
        int fIndex;

        bool isFlat;
        GLuint mode;

        const int DEFAULT_SIZE = 16;
```

Here's the soup bowl:

More Dynamic arrays

WavefrontOBJ constructor

```
WavefrontObj(string filename) {
        vIndex = -1;
        nIndex = -1;
        tIndex = -1;
        fIndex = -1;

        isFlat = false;
        mode = GL_POLYGON;

        char * line = new char[200];
        char wspace[] = {' ', '\t'};
        char separator[] = {'/'};
        char * tokens;
        ifstream file (filename);
        while ( !file.eof() ) {
            file.getline( line, 199 );          // first, strip off comments
            if ( line[0] == '#' )
                        continue;
            else if ( !strcmp( line, "" ) )
                        continue;
            else {
                        //parse the line….
            }
            ….
```

Setup for a simple parser

WavefrontOBJ constructor (cont)

```
strcpy( line_back, line ); // strtok destroys line.
token = strtok( line, wspace);
if ( !strcmp( token, "v" ) ) {
    x = atof( strtok( NULL, wspace ) );
    y = atof( strtok( NULL, wspace ) );
    z = atof( strtok( NULL, wspace ) );
    addVertex( x, y, z );
}
else if ( !strcmp( token, "vn" ) ) {
    x = atof( strtok( NULL, wspace ) );
    y = atof( strtok( NULL, wspace ) );
    z = atof( strtok( NULL, wspace ) );
    addNormal( x, y, z );
}
else if ( !strcmp( token, "vt" ) ) {
    tex_u = atof( strtok( NULL, wspace ) );
    tex_v = atof( strtok( NULL, wspace ) );
    addTexel(tex_u, tex_v );
}
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;
```

Parser

```
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() and AddTexel() are similar

WavefrontOBJ addFace()

```
Face *addFace( ) {
    if (fIndex < 0) {
        f = new Face*[DEFAULT_SIZE];
        fIndex = 0;
        current_max_faces = DEFAULT_SIZE;
    }
    f[fIndex] = new Face();
    fIndex += 1;
    if (fIndex == current_max_faces) {
        current_max_faces = 2*fIndex;
        Face **newF = new Face*[current_max_faces];
        for ( int i = 0; i < fIndex; i++ )
            newF[i] = f[i];
        delete [] f;
        f = newF;
    }
    return f[fIndex - 1];
}
```

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]], v[currentFace.vList[2]]);
                        glNormal3d(norm.x, norm.y, norm.z);
                    }
                } else if ((i = currentFace.nList[vertex]) >= 0) {
                    glNormal3d(n[i].x, n[i].y, n[i].z);
                } else if (vertex == 0) {
                    Normal norm = faceNormal(v[currentFace.vList[0]], v[currentFace.vList[1]], v[currentFace.vList[2]]);
                    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];
                glVertex3d(v[i].x, v[i].y, v[i].z);
            }
            glEnd();
        }
    }
```