Building a 3D World

- Specifying 3D geometry
- An “OBJ”ect parser
- Setting up a scene
  - NO CLASS on Monday 2/2
  - Programming Assignment #1

Lecture 5

Comp 236

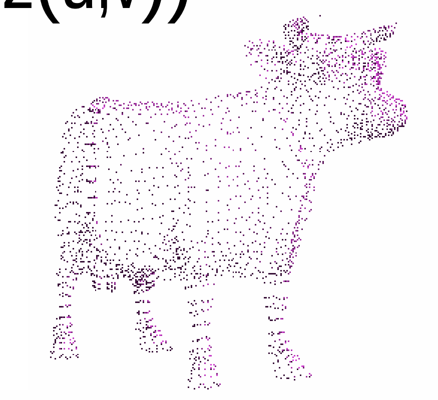
Spring 2004

Welcome to 3D!

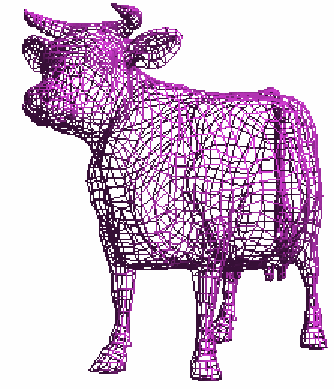
- But first, three ways to write a program
  - Minimal – the smallest possible fragment of working code that demonstrates the key concept
  - Efficient – the fastest possible code tuned for the best performance
  - Well Designed – Robust and maintainable
- Today, a tool for your bag of tricks
  - Useful 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, $f(x,y,z) = 0$

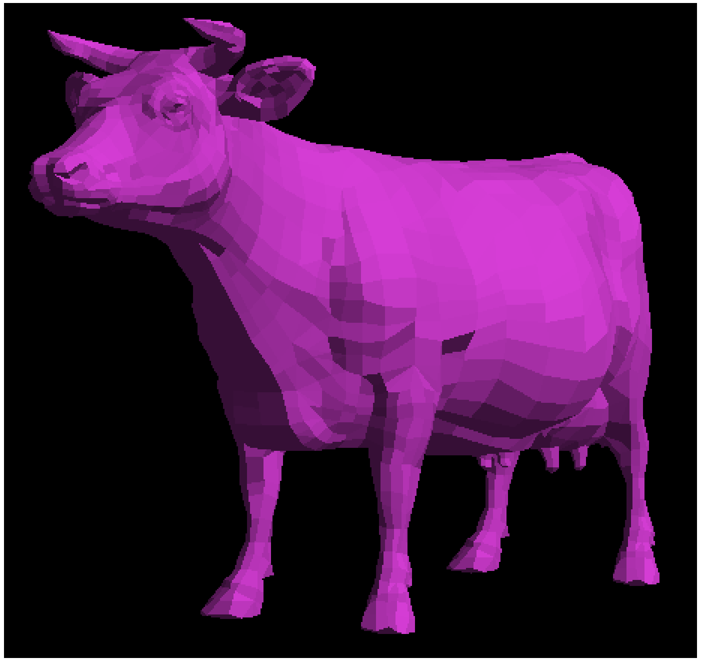


- Build up from simple primitives
  - Point – nothing really to see
  - Lines – nearly see through
  - Planes – a surface



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]

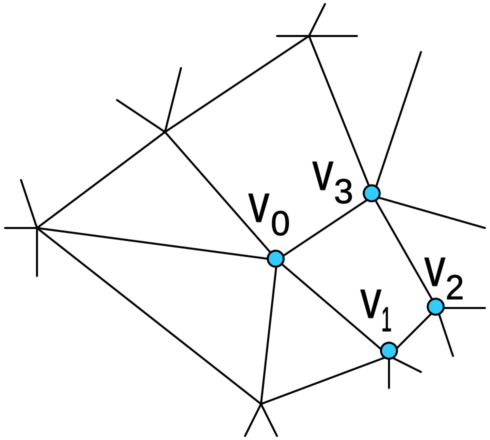
Vertex[1] = [v1.x, v1.y, v1.z]

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

:

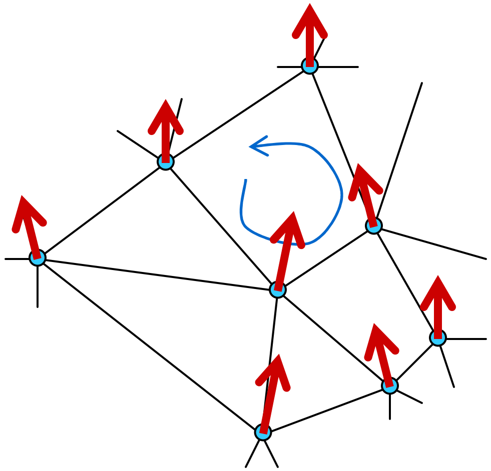
Vertex[N] = [vN.x, vN.y, vN.z]

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



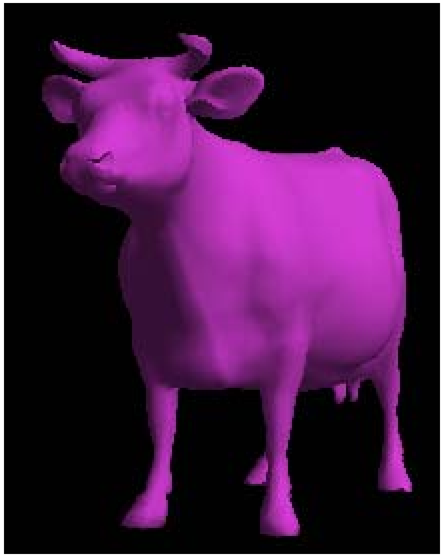
Smoothing things over

- 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

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



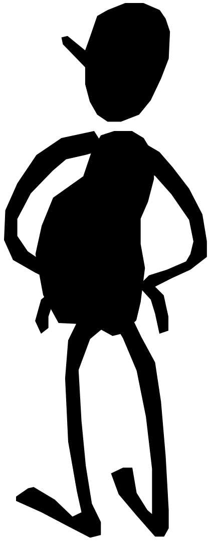
Drawing Faces in OpenGL

```
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

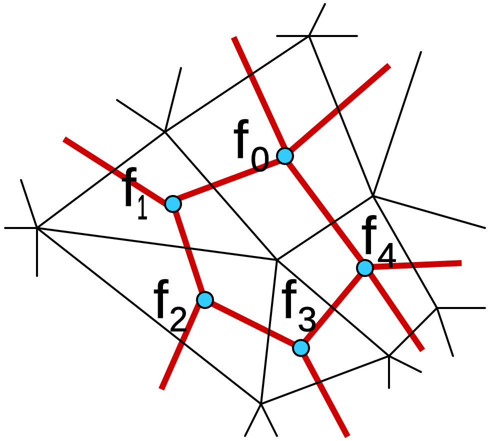
Redundancy- Generally, a vertex "position" is shared by at least 3 faces



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

Aside: Dual Graph

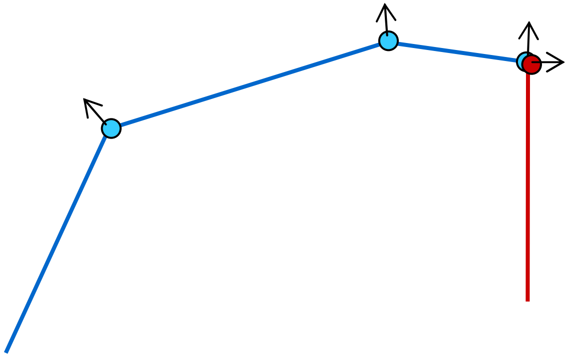
- Why do I say that a vertex is generally shared by 3 or more faces?
- Constructing a “dual” graph representation of our mesh.
  - Replace each face with a point
  - Insert an edge between every pair of faces that share an edge
  - Where are vertices in this “dual” representation?



- In what cases are there fewer than 3 faces sharing a vertex?

Decoupling Vertex and Face Features

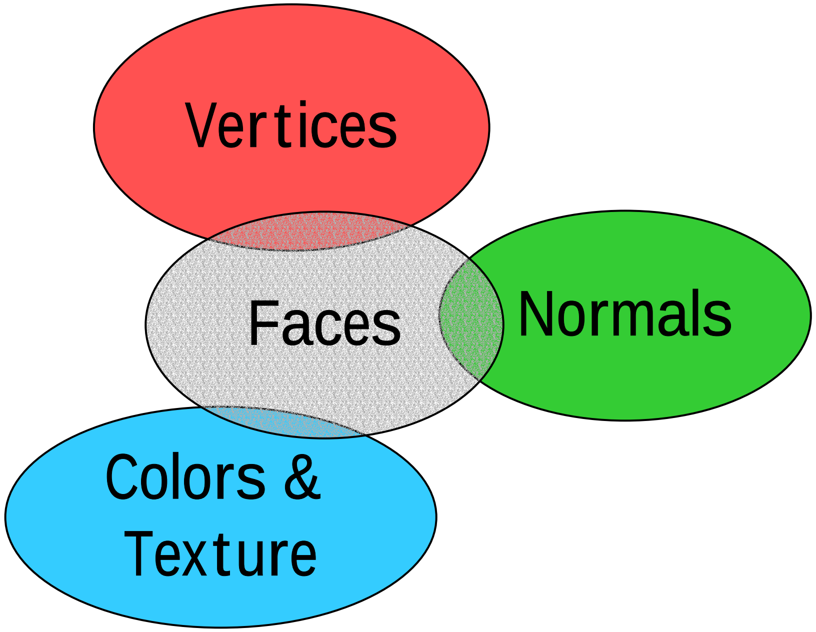
- Case for:
  - Most of the time vertices will be consistent
  - There are exceptions, however
  - Where the surface changes materials
  - Or has a high curvature (a crease)



- 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

```
public abstract class Drawable : GL {
    public abstract void Draw( );
}

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

    public Vertex( ) { }

    public Vertex(double x, double y, double z) {
        setCoordinates(x, y, z);
    }

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

Normal

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

    public Normal( ) {
    }

    public Normal(double x, double y, double z) {
        setCoordinates(x, y, z);
    }

    public void setCoordinates(double xval, double yval, double zval) {
        double l = Math.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

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

    public Texel( ) {
    }

    public Texel(double u, double v) {
        setCoordinates(u, v);
    }

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

Faces

```
public class Face {
    public int [] vList;
    public int [] nList;
    public int [] tList;
    public int vIndex;
    private const int DEFAULT_SIZE = 4;

    public Face() {
            vIndex = -1;
    }

    public void addVertex(int v) {
      // make indices zero referenced
      add(v-1, -1, -1);
    }

    public void addVertexTexel(int v, int t) {
      add(v-1, -1, t-1);
    }

    public void addVertexNormal(int v, int n) {
      add(v-1, n-1, -1);
    }

    public void addVertexNormalTexel(int v, int n, int t) {
      add(v-1, n-1, t-1);
    }
```

Very simple code

Faces continued

```
    private 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];
        vIndex = 0;
      }
      vList[vIndex] = v;
      nList[vIndex] = n;
      tList[vIndex] = t;
      vIndex += 1;
      if (vIndex == vList.Length) {
        int newLength = 2*vIndex;
        int [] newV = new int[newLength];
        int [] newN = new int[newLength];
        int [] newT = new int[newLength];
        vList.CopyTo(newV, 0);
        nList.CopyTo(newN, 0);
        tList.CopyTo(newT, 0);
        vList = newV;
        nList = newN;
        tList = newT;
      }
    }

}
```

Mostly simple code:

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

WavefrontOBJ class

```
public class WavefrontObj : Drawable {
	public Vertex [] v;
	public int vIndex;

	public Normal [] n;
	public int nIndex;

	public Texel [] t;
	public int tIndex;

	public Face [] f;
	public int fIndex;

	public bool isFlat;
	public uint mode;

	private const int DEFAULT_SIZE = 16;
```

Here’s the soup bowl:

More Dynamic arrays

WavefrontOBJ constructor

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

    isFlat = false;
    mode = GL_POLYGON;

    string line;
    char [] wspace = { , \t};
    char [] separator = {/};
    string [] tokens;
    string [] indices;

    StreamReader file = new StreamReader(filename);
    while ((line = file.ReadLine()) != null) {
        // first, strip off comments
        int comment = line.IndexOf(#);
        if (comment >= 0) {
            line = line.Substring(0, comment);
        }
        tokens = line.Split(wspace);
```

Setup for a simple parser

WavefrontOBJ constructor (cont)

```
switch (tokens[0]) {
        case "v" :
           addVertex(Double.Parse(tokens[1]), Double.Parse(tokens[2]), Double.Parse(tokens[3]));
           break;
        case "vn" :
           addNormal(Double.Parse(tokens[1]), Double.Parse(tokens[2]), Double.Parse(tokens[3]));
           break;
        case "vt" :
           addTexel(Double.Parse(tokens[1]), Double.Parse(tokens[2]));
           break;
        case "f" :
        // SEE NEXT SLIDE
           break;
        case "g":   // group
           break;
        case "s":   // smoothing group
           break;
        case "":    // blank line
           break;
        default :
           System.Console.WriteLine(line);
           break;
   }
 }
```

Parsing code

WavefrontOBJ constructor (cont)

Face code

```
case "f" :
    Face f = addFace();
    for (int i = 1; i < tokens.Length; i++) {
        indices = tokens[i].Split(separator);
        switch (indices.Length) {
            case 1:
                f.addVertex(int.Parse(indices[0]));
                break;
            case 2:
                f.addVertexTexel(int.Parse(indices[0]), int.Parse(indices[1]));
                break;
            case 3:
                if (indices[1] != "") {
                    f.addVertexNormalTexel(int.Parse(indices[0]), int.Parse(indices[1]), int.Parse(indices[2]));
                } else {
                    f.addVertexNormal(int.Parse(indices[0]), int.Parse(indices[2]));
                }
                break;
        }
    }
    break;
```

WavefrontOBJ addVertex()

```
public void addVertex(double x, double y, double z) {
    Vertex vert = new Vertex(x, y, z);
    addVertex(vert);
}

public void addVertex(Vertex vert) {
    if (vIndex < 0) {
        v = new Vertex[DEFAULT_SIZE];
        vIndex = 0;
    }
    v[vIndex] = vert;
    vIndex += 1;
    if (vIndex == v.Length) {
        int newLength = 2*vIndex;
        Vertex [] newV = new Vertex[newLength];
        v.CopyTo(newV, 0);
        v = newV;
    }
}
```

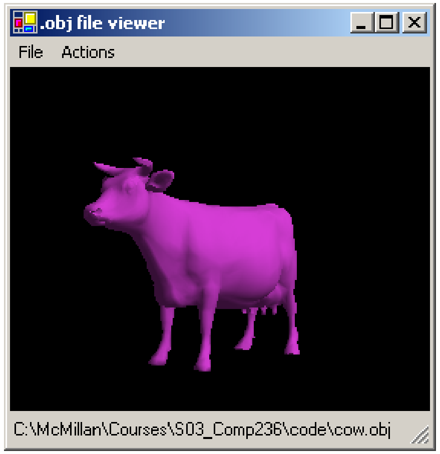
AddNormal() and AddTexel() are similar

WavefrontOBJ addFace()

```
public Face addFace( ) {
        if (fIndex < 0) {
            f = new Face[DEFAULT_SIZE];
            fIndex = 0;
        }
        f[fIndex] = new Face();
        fIndex += 1;
        if (fIndex == f.Length) {
            int newLength = 2*fIndex;
            Face [] newF = new Face[newLength];
            f.CopyTo(newF, 0);
            f = newF;
        }
        return f[fIndex - 1];
    }
```

WavefrontOBJ Draw()

```
public override 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();
    }
}
```



C:\McMillan\Courses\S03_Comp236\code\cow.obj

Next Time

- 3-D Transformation Principles
- Vector Spaces
- Affine Spaces
- 3-D Transfigurations