Geometric Modeling for Computer Graphics

Mike Bailey
mjb@cs.oregonstate.edu

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License
What do we mean by “Modeling”?  

How we model geometry depends on what we would like to use the geometry for:

- Looking at its appearance
- Interacting with its shape?
- How does it interact with its environment?
- What is its surface area and volume?
- Will it need to be 3D-printed?
- Etc.
Explicitly Listing Geometry and Topology

Models can consist of thousands of vertices and faces – we need some way to list them efficiently

This is called a **Mesh**. If it’s in nice neat rows like this, it is called a **Regular Mesh**. If it’s not, it is called an **Irregular Mesh**, or oftentimes called a **Triangular Irregular Network**, or **TIN**.
Explicitly Listing Geometry and Topology

static GLfloat CubeVertices[][3] =
{
    { -1., -1., -1. },
    {  1., -1., -1. },
    { -1.,  1., -1. },
    {  1.,  1., -1. },
    { -1., -1.,  1. },
    {  1., -1.,  1. },
    { -1.,  1.,  1. },
    {  1.,  1.,  1. }
};

static GLfloat CubeColors[][3] =
{
    { 0., 0., 0. },
    { 1., 0., 0. },
    { 0., 1., 0. },
    { 1., 1., 0. },
    { 0., 0., 1. },
    { 1., 0., 1. },
    { 0., 1., 1. },
    { 1., 1., 1. }
};

static GLuint CubeQuadIndices[][4] =
{
    { 0, 2, 3, 1 },
    { 4, 5, 7, 6 },
    { 1, 3, 7, 5 },
    { 0, 4, 6, 2 },
    { 2, 6, 7, 3 },
    { 0, 1, 5, 4 }
};
The Cube Can Also Be Defined with Triangles

```c
GLuint CubeQuadIndices[][4] = 
{
    { 0, 2, 3, 1 },
    { 4, 5, 7, 6 },
    { 1, 3, 7, 5 },
    { 0, 4, 6, 2 },
    { 2, 6, 7, 3 },
    { 0, 1, 5, 4 }
};

GLuint CubeTriangleIndices[][3] = 
{
    { 0, 2, 3 },
    { 0, 3, 1 },
    { 4, 5, 7 },
    { 4, 7, 6 },
    { 1, 3, 7 },
    { 1, 7, 5 },
    { 0, 4, 6 },
    { 0, 6, 2 },
    { 2, 6, 7 },
    { 2, 7, 3 },
    { 0, 1, 5 },
    { 0, 5, 4 }
};
```
3D Printing uses an Irregular Triangular Mesh Data Format
3D Printing uses an Irregular Triangular Mesh Data Format
Go Beavs – mmmmmm! 😊
Meshes Can Be Smoothed
Editing the Vertices of a Mesh

- Original
- Pulling on a single Vertex
- Pulling on a Vertex with Proportional Editing Turned On

“Circle of Influence”
Remember Venn Diagrams (2D Boolean Operators) from High School?

Two Overlapping Shapes

Union: $A \cup B$

Intersection: $A \cap B$

Difference: $A - B$
Well, Welcome to Venn Diagrams in 3D

Two Overlapping Solids

Union: \( A \cup B \)

Intersection: \( A \cap B \)

Difference: \( A - B \)

This is often called **Constructive Solid Geometry**, or **CSG**
Geometric Modeling Using 3D Boolean Operators on Meshes

Two Overlapping Solids

Union: $A \cup B$

Intersection: $A \cap B$

Difference: $A - B$
Another Way to Edit Meshes: Volume Sculpting

This is often called a “Lattice” or a “Cage”.

Slip a simpler object (e.g., a subdivided cube) around some of the object’s vertices. As you sculpt the simpler object, all those object vertices get sculpted too.

A Small Amount of Input Change Results in a Large Amount of Output Change
Another way to Model: Curve Sculpting – Bézier Curve Sculpting

\[ P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2 (1-t) P_2 + t^3 P_3 \]

\[ 0 \leq t \leq 1. \]

where \( P \) represents \[ \{x, y, z\} \]
t goes from 0.0 to 1.0 in whatever increment you’d like

\[ 0. \leq t \leq 1. \]

You draw the curve as a series of lines

**GL_LINE_STRIP** is a good topology for this
Curve Sculpting – Bézier Curve Sculpting Example

\[ P_0, P_1, P_2, P_3 \]
Curve Sculpting – Bézier Curve Sculpting Example

Moving a single control point moves its entire curve

**A Small Amount of Input Change Results in a Large Amount of Output Change**
The Yellow 4-Point Bézier Curve
Another way to Model:
Curve Sculpting – Catmull-Rom Curve Sculpting

The Catmull-Rom curve consists of any number of points.
The first point influences how the curve starts.
The last point influences how the curve ends.
The overall curve goes smoothly through all other points.

To draw the curve, grab points 0, 1, 2, and 3, call them \( P_0, P_1, P_2, \) and \( P_3 \), and loop through the following equation, varying \( t \) from 0. to 1. in an increment of your own choosing:

\[
P(t) = 0.5 \times \left[ 2 \times P_1 + t \times (-P_0 + P_2) + t^2 (2 \times P_0 - 5 \times P_1 + 4P_2 - P_3) + t^3 (-P_0 + 3P_1 - 3P_2 + P_3) \right]
\]

\[
0 \leq t \leq 1.
\]

where \( P \) represents \( \{x, y, z\} \)

For each set of 4 points, this equation just draws the line between the second and third points. That’s why you keep having to use subsequent sets of 4 points.
Another way to Model:
Curve Sculpting – Catmull-Rom Curve Sculpting

For each set of 4 points, this equation just draws the line between the second and third points. That’s why you keep having to use subsequent sets of 4 points.

To draw the curve, grab points 0, 1, 2, and 3, call them \( P_0, P_1, P_2, \) and \( P_3 \), and loop through the equation, varying \( t \) from 0. to 1. in an increment of your own choosing.

Then, grab points 1, 2, 3, and 4, call them \( P_0, P_1, P_2, \) and \( P_3 \), and loop through the same equation.

Then, grab points 2, 3, 4, and 5, call them \( P_0, P_1, P_2, \) and \( P_3 \), and loop through the same equation.

And so on...

A Small Amount of Input Change Results in a Large Amount of Output Change
The Yellow 6-Point Catmull-Rom Curve
Another way to Model: Bézier Surface Sculpting

Moving a single point moves its entire surface

A Small Amount of Input Change Results in a Large Amount of Output Change
Surface Equations can also be used for Analysis

Showing Contour Lines

Showing Curvature
Another Way to Model: Metaball Objects
Metaball Objects

The cool thing is that, if you move them close enough together, they will “glom” into a single object.
Metaball Objects Can Be Turned into Meshes for Later Editing
Voxelization as a Special Way to Model 3D Geometry

Randy Rauwendaal
Displacement Textures as a Special Way to Model 3D Geometry

Vertex-described Object + Image Texture = Displacement Texture

= Moon Image
Displacement Textures as a Special Way to Model 3D Geometry

Image Texture

Displacement Texture
(light = high, dark = low)
Displacement Textures as a Special Way to Model 3D Geometry

```cpp
uniform float uScale;
uniform sampler2D uDispUnit;

globalfloat uScale;
globalsampler2D uDispUnit;

texture( uDispUnit, st ).r;

// in half-meters, relative to a radius of 1,727,400 meters

disp *= uScale;

vec3 vert = gl_Vertex.xyz;
vert += norm * disp;

vec2 st = gl_MultiTexCoord0.st;
vST = st; // to send to fragment shader

vec3 norm = normalize( gl_Normal);
vNormal= normalize( gl_NormalMatrix * gl_Normal);

void

main( )

{

    vec2 st = gl_MultiTexCoord0.st;
    vST = st; // to send to fragment shader

    vec3 norm = normalize( gl_Normal );
    vNormal= normalize( gl_NormalMatrix * gl_Normal );

    float disp = texture( uDispUnit, st ).r;
          // in half-meters, relative to a radius of 1,727,400 meters
    disp *= uScale;

    vec3 vert = gl_Vertex.xyz;
    vert += norm * disp;

    gl_Position = gl_ModelViewProjectionMatrix * vec4( vert, 1. );
}
```
Displacement Textures as a Special Way to Model 3D Geometry

```glsl
#version 330 compatibility

uniform float uLightX, uLightY, uLightZ;
uniform float uKd;
uniform sampler2D uColorUnit;

in vec2 vST;
in vec3 vNormal;

void main( )
{
    vec3 light = normalize( vec3( uLightX, uLightY, uLightZ ) );
    float intensity = uKd * abs( dot( vNormal, light ) );
    intensity += (1.-uKd); // ambient
    vec3 newcolor = texture( uColorUnit, vST).rgb;
    gl_FragColor = vec4( newcolor*intensity, 1. );
}
```

`moondisp.frag`
L-Systems as a Special Way to Model 3D Geometry

Introduced and developed in 1968 by Aristid Lindenmayer, L-systems are a way to apply grammar rules for generating fractal (self-similar) geometric shapes. For example, take the string:

“FF+[+F-F-F]-[-F+F+F]”

- F move forward one step
- + turn right
- - turn left
- [ push state
- ] pop state
L-Systems as a Special Way to Model 3D Geometry

But the real fun comes when you call that string recursively. For every F, replicate that string but with smaller geometry:

“F → FF+[+F-F-F]-[-F+F+F]”
L-Systems as a Special Way to Model 3D Geometry

And, of course we can introduce more grammar to swing it into 3D

```
F → FF+[+F->F][-F+^F+vF]
```

+ rotate + about Z
- rotate - about Z
< rotate + about Y
> rotate – about Y
v rotate + about X
^ rotate – about X
Modeling as an Initial Step in Simulation (Explosion)
The object must be a legal solid. It must have a definite inside and a definite outside. It can’t have any missing face pieces.

“Definite inside and outside” is sometimes called “Two-manifold” or “Watertight”
The Simplified Euler's Formula* for Legal Solids

*sometimes called the Euler-Poincaré formula

\[ F - E + V = 2 \]

- **F** Faces
- **E** Edges
- **V** Vertices

The full formula is:

\[ F - E + V - L = 2( B - G ) \]

- **F** Faces
- **E** Edges
- **V** Vertices
- **L** Inner Loops (within faces)
- **B** Bodies
- **G** Genus (number of through-holes)

For a cube, \( 6 - 12 + 8 = 2 \)
Object Modeling Rules for 3D Printing

Objects cannot pass through other objects. If you want two shapes together, do a Boolean union on them so that they become one complete object.

Overlapped in 3D -- bad

Boolean union -- good