Geometric Modeling for Computer Graphics

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
- Will we need to interact with its shape?
- How does it interact with its environment?
- How does it interact with other objects?
- What is its surface area and volume?
- Will it need to be 3D-printed?
- Etc.
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

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

\[ “F \rightarrow FF+[+F-F-F][-F+F+F]” \]
**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+^F+vF]
```

+ rotate + about Z
- rotate - about Z
< rotate + about Y
> rotate – about Y
v rotate + about X
^ rotate – about X

---

**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

```c
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 GLboolean 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 }
};

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.}
};
```
3D Printing uses an Irregular Triangular Mesh Data Format
Meshes Can Be Smoothed
Mesh Vertices Can Be Edited

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

“Circle of Influence”

Meshes Can Be Sculpted

Original
“Clay Thumb” Sculpting
Sculpting Can Produce Extra Mesh Vertices
Remember Venn Diagrams (2D Boolean Operators) from High School?

Two Overlapping Shapes

Union: \( A \cup B \)

Intersection: \( A \cap B \)

Difference: \( A - B \)

I thought I left Venn Diagrams behind in High School!

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: $A - B$

Procedural Geometric Modeling Using TinkerCad/Codeblocks
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 \( \begin{cases} x \\ y \\ z \end{cases} \)
You draw the curve as a series of lines

**GL_LINE_STRIP** is a good topology for this.
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:

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

where \( P \) represents \( \begin{bmatrix} x \\ y \\ z \end{bmatrix} \)

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.

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

Wireframe
Surface
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
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**

**Displacement Textures as a Special Way to Model 3D Geometry**
Displacement Textures as a Special Way to Model 3D Geometry

#version 330 compatibility
uniform float uLightX, uLightY, uLightZ;
uniform float uHeightScale;
uniform float uSeaLevel;
uniform sampler2D uDispUnit;
uniform bool uDoElevations;

out vec2 vST;
out vec3 vN; // normal vector
out vec3 vL; // vector from point to light

void main()
{
  vec2 st = gl_MultiTexCoord0.st;
  vST = st;

  vec3 norm = normalize(gl_NormalMatrix * gl_Normal); // normal vector
  vN = norm;

  vec3 LightPos = normalize(vec3(uLightX, uLightY, uLightZ));

  vec4 ECposition = gl_ModelViewMatrix * gl_Vertex; // eye coordinate position
  vL = LightPos - ECposition.xyz; // vector from the point to the light position

  vec3 vert = gl_Vertex.xyz;
  if (uDoElevations)
  {
    float disp = texture(uDispUnit, st).r;
    disp -= uSeaLevel;
    disp *= uHeightScale;
    vert += normalize(gl_Normal) * disp;
  }
  gl_Position = gl_ModelViewProjectionMatrix * vec4(vert, 1.);
}
Displacement Textures as a Special Way to Model 3D Geometry

```glsl
#version 330 compatibility
uniform bool uDoBumpMapping;
uniform float uKa, uKd;
uniform float uHeightScale;
uniform float uNormalScale;
uniform sampler2D uColorUnit;
uniform sampler2D uDispUnit;

in vec2 vST;
in vec3 vN;
in vec3 vL;
define DELTA 0.01

void main()
{
    vec3 newColor = texture( uColorUnit, vST ).rgb;
    gl_FragColor = vec4( newColor, 1. );
    if( uDoBumpMapping )
    {
        . . . // see next slide
    }
}
```

moondisp.frag, I

```glsl
if( uDoBumpMapping )
{
    vec2 stp0 = vec2( DELTA, 0. );
    vec2 st0p = vec2( 0., DELTA );
    float west = texture2D( uDispUnit, vST-stp0 ).r;
    float east = texture2D( uDispUnit, vST+stp0 ).r;
    float south = texture2D( uDispUnit, vST-st0p ).r;
    float north = texture2D( uDispUnit, vST+st0p ).r;
    vec3 stangent = vec3( 2.*DELTA, 0., uNormalScale * ( east - west ) );
    vec3 ttangent = vec3( 0., 2.*DELTA, uNormalScale * ( north - south ) );
    vec3 Normal = normalize( cross( stangent, ttangent ) );
    vec3 Light = normalize(vL);
    vec3 ambient = uKa * newColor;
    float d = 0.;
    if( dot(Normal,Light) > 0. ) // only do diffuse if the light can see the point
    {
        d = dot(Normal,Light);
    }
    vec3 diffuse = uKd * d * newColor;
    gl_FragColor = vec4( ambient + diffuse, 1. );
}
```

moondisp.frag, II
Displacement Textures as a Special Way to Model 3D Geometry

- Lighting only, no displacements
- Displacements only, no lighting
- Displacements + Lighting

Note: as you can imagine, static images do not do this justice. Being able to dynamically rotate the Moon and change the height exaggeration and light position makes a big difference!

Modeling as an Initial Step in Simulation (Explosion)
Object Modeling Rules for 3D Printing

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

\[ F - E + V = 2 \]

*sometimes called the Euler-Poincaré formula

For a cube, \(6 - 12 + 8 = 2\)

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)
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