Geometric Modeling 1
Look around the room. To make a 3D model of a room requires modeling every single object you can see. Leaving out smaller objects (clutter) makes the room seem sterile and unrealistic (e.g. similar to an IKEA furniture store)
The same is true for outdoor scenes...
Movies...
Movies...
Games...
Games...
Mesh Construction
Mesh Construction

- A polygonal mesh is represented on the computer using various primitives: vertices, edges, and faces (triangles are the most common)

- **Problem:** Manually specifying every vertex, edge, and face is cumbersome (even if it is slightly faster than chipping a model out of stone...)

- **Solution:** software packages can help automate some vertex placement (but it can still take a long time)

- **Process:**
  - Model a low resolution mesh “by hand”
  - Refine that mesh with an *automatic* method/algorithm
  - Edit the refined mesh with a semi-*automated* method/algorithm
  - Refine more, Edit more, rinse, repeat, etc. etc.
Refinement via subdivision
Subdivision

- Given a coarse input mesh, generate a finer output mesh via subdivision.
- For a given mesh, various smooth limit surfaces exist.
  - The exact smooth limit surfaces depend on the subdivision algorithm used.
- After just a few refinements, additional changes become too small to see/matter.
Figure 2.1: Example of subdivision for curves in the plane. On the left 4 points connected with straight line segments. To the right of it a refined version: 3 new points have been inserted "inbetween" the old points and again a piecewise linear curve connecting them is drawn. After two more steps of subdivision the curve starts to become rather smooth.
Subdivision Surfaces
LOOP subdivision
Smooth Subdivision Surfaces Based on Triangles

by

Charles Terrell Loop

A thesis submitted to the faculty of the University of Utah in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mathematics
The University of Utah
August 1987

1878 citations...
QUESTION #1 (short and long)

What is the highest cited paper (according to google scholar) ever written by one of the Stanford CS graphics professors (see lecture 1)?

<look up after class, and email the answer to the usual place...>

1878 citations...
Loop Subdivision

• Subdivide each triangle into 4 triangles
• Move vertices to new positions
• Repeat the above two steps until you get the desired resolution
• Generates a $C^2$ continuous limit surface almost everywhere
  • except at some extraordinary vertices where the limit is $C^1$ continuous
Subdivide Each Triangle into 4 Triangles
Subdivide Each Triangle into 4 Triangles
Move the Vertices

- Compute perturbed locations for new vertices (black)
- Move the original vertices too (grey)
Move the Vertices

- Compute perturbed positions of new vertices (black) using a weighted average of the four adjacent original vertices (grey).
- Change the original vertex (grey) positions using a weighted average of six adjacent original vertices (grey).
- Repeat until converged (or a few times).
Extraordinary Points

- Most vertices are regular (degree 6), but not all
- If a mesh is topologically equivalent to a sphere, not all the vertices can have degree 6
Extraordinary Points

- Find weights for extraordinary points that generate a smooth surface (tangent plane continuous)
- Want the surface normal to be continuous
- Complex math problem...

Warren weights

$$\beta = \begin{cases} 
3 \\
\frac{3}{8n} \\
\frac{3}{16}
\end{cases} \quad \text{for } n > 3$$

$$\beta = \begin{cases} 
\frac{3}{8n} \\
\frac{3}{16}
\end{cases} \quad \text{for } n = 3$$
An Example...
Starting Mesh
Add New Vertices
New Vertex & Stencil
Move New Vertex
Original Vertex & Stencil
Move Original Vertex
Extraordinary Vertex & Stencil
Move Extraordinary Vertex
Subdivided Surface
Subdivide Again
And Again
And Again
Limit surface
Mesh editing
Mesh Editing

• Artist manipulates a few control points or mesh points and the system *automatically* deforms the mesh
  • E.g., twist, bend, stretch, etc.
  • Fast, intuitive, preserves details
  • Widely used in CAD software such as Blender, Maya, etc.

• Spline (e.g. B-spline, NURBS) mesh editing uses control points
• Laplacian mesh editing allows one to directly move mesh points
B-Spline

- The $i$-th B-spline basis function of order $j$ is recursively defined as

$$N_{i,0}(u) = \begin{cases} 1, & \text{if } u_i \leq u < u_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$N_{i,j}(u) = \frac{u - u_i}{u_{i+j} - u_i} N_{i,j-1}(u) + \frac{u_{i+j+1} - u}{u_{i+j+1} - u_{i+1}} N_{i+1,j-1}(u)$$

where $\{u_0, u_1, \ldots u_m\}$ are the knots

Basis functions with different orders ($p = 0, 1, 2$) for knots $\{0, 1, 2, 3\}$

- Then the B-spline is defined as:

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i$$

where $P_0, P_1, \ldots, P_n$ are the 2D/3D control points

- $n + 1 = m - p$ determines how many control points are needed, based on the number of intervals $m$ and the order $p$

- For example: $2 + 1 = 6 - 3$ (for the figure to the right)
B-Spline

- Use **knots** to subdivide the **parametric domain** of the curve into intervals
- Use **control points** to locally control the **shape** of the curve
  - e.g., interval $s_0$ is controlled by control points $P_0, P_1, P_2, P_3$
NURBS Curve

• NURBS (Non-Uniform Rational B-Spline) curve is defined by:

\[ C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \]

where \( N_{i,p} \) is a B-spline basis function, \( P_i \) is a control point, and \( w_i \) is the weight of \( P_i \).

• Increasing \( w_i \) pulls the curve closer to the corresponding control point \( P_i \), while decreasing \( w_i \) pushes the curve farther from \( P_i \).

NURBS Surface

• Extending the idea from curves to surfaces, a NURBS surface is defined as

\[ S(u, v) = \frac{\sum_{i=0}^{m} \sum_{j=0}^{n} N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j}}{\sum_{i=0}^{m} \sum_{j=0}^{n} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \]

• Now we have a 2D array of control points \( P_{i,j} \) with the associated basis function \( N_{i,p}(u)N_{j,q}(v) \).

http://www.3dmax-tutorials.com/CV_Surface.html
Subdividing splines
Subdividing Spline Curves

- A spline curve can be subdivided by adding more knots and control points
- First, insert new knots without changing the shape of the curve
  - Whenever inserting a knot, add a control point as well
  - The positions of the control points need to be updated to preserve the shape of the curve
- E.g., given a spline curve of degree \( p \) with knots \( \{ u_0, u_1, ..., u_m \} \) and control points \( P_0, P_1, ..., P_n \), we insert a new knot \( \hat{u} \) between \( u_k \) and \( u_{k+1} \), and add one more control point obtaining the new set of control points \( P_0^*, P_1^*, ..., P_{n+1}^* \). The positions of these control points are computed using linear interpolation:

\[
P_i^* = (1 - \alpha_i) P_{i-1} + \alpha_i P_i
\]

where

\[
\alpha_i = \begin{cases} 
1 & i < k - p + 1 \\
0 & i > k \\
\frac{\hat{u} - u_i}{u_{i+p} - u_i} & k - p + 1 \leq i \leq k
\end{cases}
\]
Original Curve
Insert a Knot
Add/Adjust Control Points

\[ \text{Add/Adjust Control Points} \]
Laplacian Mesh editing
Differential Coordinates

• For each vertex, we represent its differential coordinate by the difference between its position \( x_i \) and the average position of its neighbors:

\[
d_i = L(x_i) = x_i - \frac{1}{n_i} \sum_{j \in N_i} x_j
\]

• Differential coordinates approximate the local shape
  • The direction of \( d_i \) approximates the normal
  • The magnitude of \( d_i \) approximates the mean curvature

• A mesh/model can be described by a vector of differential coordinates of all its vertices \( D = \{d_i\} \)

• \( D \) can be calculated by multiplying a constant coefficient sparse matrix \( L \) (each vertex only interacts with its local neighbors) with a position vector \( X \) (a vector of positions of all the vertices \( X = \{x_i\} \))
  • i.e., \( D = LX \)
• User selects some control points $x_i, i = \{1, \ldots, n\}$ in the region of interest (ROI), and sets the target position $p_i$ for each control point

• Solves for $x_i^*$ for all vertices in the region of interest in a least squares sense

$$LX^* = D$$ with (soft) constraints $x_i^* = p_i$ for $i \in \{1, \ldots, n\}$

• Equivalent to minimizing the least squares error

$$E(X^*) = \sum_{i \in \text{ROI}} \left\| d_i - L(x_i^*) \right\|^2 + \sum_{i=1}^{n} \left\| x_i^* - p_i \right\|^2$$

• Since differential coordinates are sensitive to local deformations, we need to compute an appropriate transformation matrix $T_i(x_i^*)$ for each vertex obtaining the modified error function

$$E'(X^*) = \sum_{i \in \text{ROI}} \left\| T_i(x_i^*)d_i - L(x_i^*) \right\|^2 + \sum_{i=1}^{n} \left\| x_i^* - p_i \right\|^2$$

• $T_i$ is expressed as a function of $X^*$ by solving the following least squares problem analytically

$$\min_{T_i} \left( \left\| T_i x_i - x_i^* \right\|^2 + \sum_{j \in N_i} \left\| T_i x_j - x_j^* \right\|^2 \right)$$
Advice for Geometry Modeling

• **Decide what geometry you want in your scene**
  - Reference pictures from last year’s class to get an idea of what your model should look like
  - Have your final scene in mind when modeling your object, stops you from wasting time on modeling something that you won’t use

• **Manually build your geometry**
  - Different modeling programs can differ drastically in their UI, try them all to find one that you like, and watch a modeling tutorial online before you start
  - Export your model into an .obj file so that the basic .obj reader provided in the example code can parse them

• **Use downloaded geometry from the Internet**
  - If your model has multiple components with different attributes (vertices, faces, normals, uv coordinates, material names, etc.), you may need to extend the basic .obj reader in example code to read all those attributes you need
  - If your downloaded mesh has flaws (e.g., holes, non-manifold parts, etc), try to use Meshlab to fix them
  - For those models without sharp features, subdivision can always make them look better

• **Procedural modeling**
  - Using an external library for noise generation can save you a lot of time

• **Display your geometry in OpenGL**
  - Make sure the clockwise/counterclockwise order of the triangle vertex indices are consistent with OpenGL (OpenGL uses the counterclockwise order)
  - Make sure the coordinate system used for your model is consistent with the one used in OpenGL. Transform your model if the two coordinate systems are different, e.g., you need to flip y and z axis of your model to correctly show it in OpenGL if you use 3DSMAX.
  - Make sure that the normals are consistent in your object so that it can be shaded properly in OpenGL
Useful Resources

- There are many modeling programs that are designed to help you create and modify models
  - Blender (free)
  - Autodesk Maya (free for students)
  - Autodesk 3DSMax (free for students)
  - SketchUp (free)
  - Meshlab (free)
  - Other modeling programs include Modo, AutoCAD, Lightwave, Bryce, Hexagon, etc.
  - Many 3D modeling programs also include other features such as animation and simulation.

- Model Database
  - Aim@Shape (models used in graphics academies, e.g., Stanford bunny, dragon, armadillo, etc)
  - Archive3D (everyday objects, e.g., desks, chairs, sofa, etc)
  - GrabCAD (mechanical objects, e.g., robots, planes, cars, warships, etc)
  - TF3DM (well categorized free models)
  - TurboSquid (largest model database in the world, but only part of them are free)
  - 3D Warehouse (architecture, e.g. buildings, bridges, furniture, etc.)

- Procedural Modeling
  - Libnoise (a C++ library to generate coherent noise)
  - Noise Texture Generator (generate your noise texture online)
EXAMPLES...
Blender, Maya, Meshlab

• Creates all the meshes with modeling software

• Extends the obj loader given in sample code to read vertex locations, normals, faces, and texture coordinates from files made in Maya and Blender

Adam Young and Winnie Lin, CS148 2013
Blender, Loop Subdivision, Fluid Simulator

• Modeled all the meshes with low resolution in Blender, and then loop-subdivide them for smoother looking
• Generate the icings of the donuts using a fluid simulator
• Generate normal map from the texture using CrazyBump
Artistic Scene Layout with Mixed Downloaded and Manually Generated Meshes

- Find the cloth and other mesh objects available online
- Manually generate mesh for the diamond
- Put everything together artistically, create a scene that resembles traditional still life oil paintings
Perlin Noise, Procedural Mesh/Texturing Generation

- Procedurally generate terrain with a 2D Perlin Noise function followed by 3 iterations of Loop subdivision
- Partition the terrain mesh into grass, snow, and water regions based on height
- Model and animate clouds with a 3D Perlin noise function
Procedural and Parametric Mesh Generation

• The boat is parametrically calculated from a number of values; the length, width, height, thickness, quadratic width, etc.

• The bridge is also parametrically calculated from the given values length, width, thickness, gap size and number of boards.

• The waves in the sea are procedurally generated based on the function \( y = 4\sin(2x + \epsilon)\cos(2z) \) where \( \epsilon \) is a noise function.
L-system, Plant Growing Animation

- Plants are modeled as an L-system, stored as a list of segments and drawn as truncated cones
- Rains, seeds, and fruits are modeled as particle sprites
Maya, TF3DM, environment map, billboards

- The toy monster was modeled in Maya from geometric primitives
- The bridged and helicopter were downloaded from TF3DM
- The UVs and textures for the bridge were fixed in Maya
- The lights are billboards, the sky is an environment map, the water was a plane with a normal map

Marianna Neubauer, CS148 2014
Fractal Geometry: Raytracing Quaternion Julia Sets

- Calculate the ray intersection point (in quaternion space) by iterating over $z_{n+1} = z_n^2 + c$.
- $z_n$ represents the origin of the ray and $z_{n+1}$ represents the ray intersection point.
- $c$ is a quaternion that defines the shape of the Julia Set.
- Caustic effects were accomplished with bidirectional path tracing.
- The D&D die was from TurboSquid.

Marianna Neubauer, CS148 2014
Blender, Microsoft Word and Paint

- Models are created, modified, and subdivided in Blender (wolf is downloaded online)
- Careful scaling and rotation of objects to make the scene realistic and seamless
- Textures are done using Blender, Word, and Paint
TurboSquid, Blender, Normal mapping, Multiple lights

- Canoe model is downloaded from TurboSquid, and candles and lily pads are created in Blender
- Normal mapping is applied on the water surface and image textures are used for scene objects
- 11 light sources are used to light up the scene, with alpha blending for transparency effect
Glow

- Random geometry generation to create the blocks
- All emissive objects in the scene are drawn to a glowmap (framebuffer object), and blurred using fragment shader

Charlie Yang, CS148 2014
Procedurally generated terrain, Noise, Haze, Moving camera

- Procedurally generated terrain using the diamond-squares algorithm
- Parts of the terrain were then sequentially smoothed into rolling hills using a derivation of a box blurring technique
- Terrains are painted using height and noise from multiple sine functions
- Haze effect is implemented in fragment shader and moving camera is used to make scene more realistic