Introduction to Geometric Modeling

CSE606: Computer Graphics
Jaya Sreevalsan Nair, IIIT Bangalore
January 15, 2025

Geometry of Curves and Surfaces

Curves - Representation

How can we define a circular arc: assume center at the origin and unit radius?

Curves - Representation

How can we define a quadrant of a circle: assume center at the origin and unit radius?

Implicit form:

$$x^2 + y^2 = 1$$

$$y = \sqrt{1 - x^2}, x \in (0, 1)$$

Curves - Representation

How can we define a quadrant of a circle: assume center at the origin and unit radius?

Implicit form:

$$x^2 + y^2 = 1$$

$$y = \sqrt{1 - x^2}, x \in (0, 1)$$

Parametric form:

$$x = \cos(\theta), y = \sin(\theta), \theta \in (0, \pi/2)$$

$$x = \frac{1 - t^2}{1 + t^2}$$

$$y = \frac{2t}{1 + t^2}$$

$$, t \in (0, 1)$$

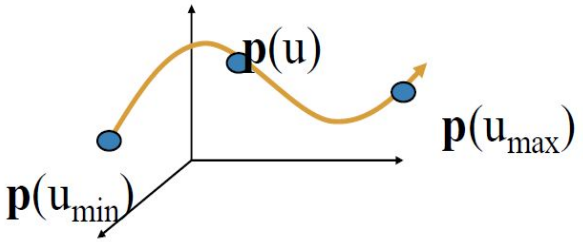
$$\textit{In general, } x = x_c + r\cos\theta, \; y = y_c + r\sin\theta, \; \theta_{min} \leq \theta \leq \theta_{max}$$

Curves - Evaluation

Need to be able to compute points on the curve in specified manner. e.g.,

- a set of uniformly spaced points on the curve
- points for each pixel covered by the curve

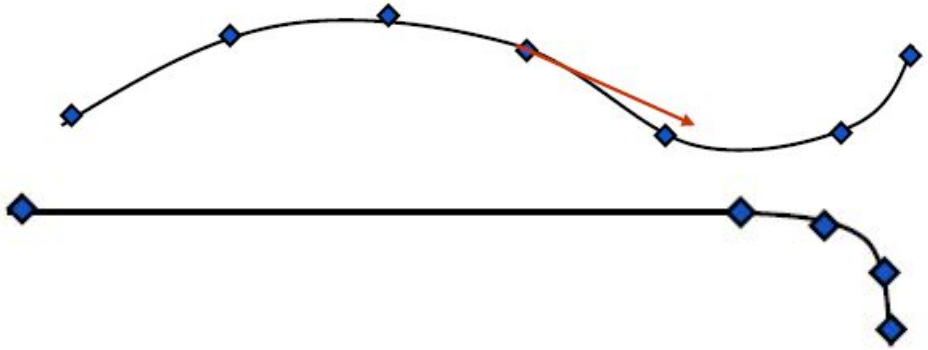
$$\mathbf{p}(u)=[x(u), y(u), z(u)]^T$$

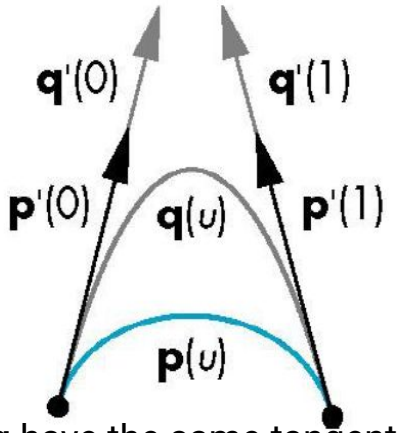


Parametric representation convenient for such computations: define the curve as a set of functions on a single parameter ***u***

Tangents and Derivatives

Often we also need to compute tangents and/or derivatives at points along a curve. Can be easily computed for parametric forms.





p and **q** have the same tangent (direction) but different first derivatives (magnitude)

E. Angel and D. Shreiner: Interactive Computer Graphics

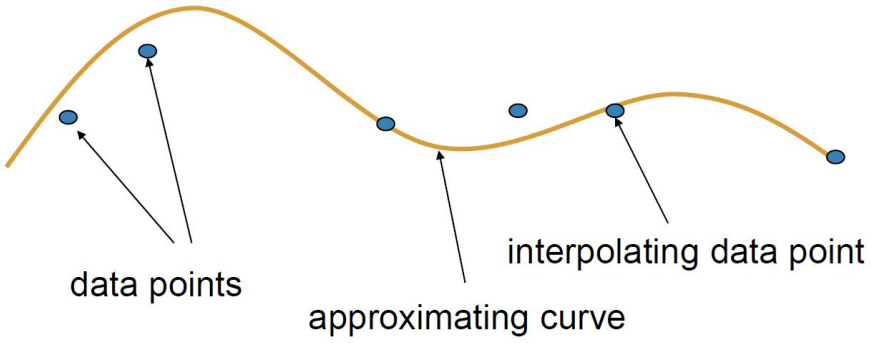
Assume the points marked are equally spaced in parametric space. Curve can be *tangent continuous* (***G*^{1}**) but not *derivative continuous* (***C*^{1}**).

Modeling with Curves - Interpolation vs Approximation

Assume we have a set of points. We have two options:

1. Interpolation: Compute a curve that exactly fits all these points
2. Approximation: Compute a curve that is “close” to all the given points, and meets a defined threshold.

Exact fits (i.e. interpolation) can generate curves with undesirable properties



E. Angel and D. Shreiner: Interactive Computer Graphics

Curves

Bezier Curves

Smooth approximating curves that are piecewise polynomials (typically quadratic or cubic), and ensure tangent continuity at joins.

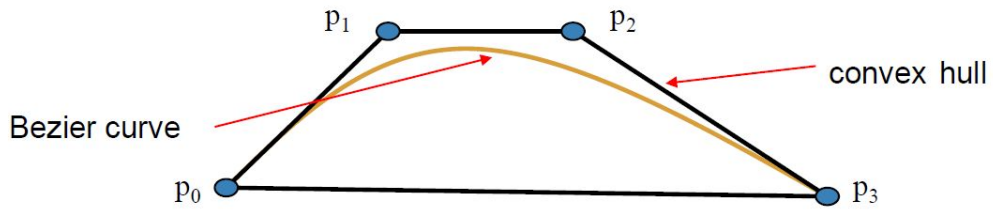
Given points p_0, p_1, p_2, p_3 :

Define a curve that fits p_0, p_3, and whose tangents (at the end points) match the directions p_0p_1 and p_2p_3

Simple parametric form:

Shape of curve can be controlled by moving one or more of the points. Provides good editing capability.

Demo: http://nurbscalculator.in/ (WebGL) and https://www.jasondavies.com/animated-bezier/



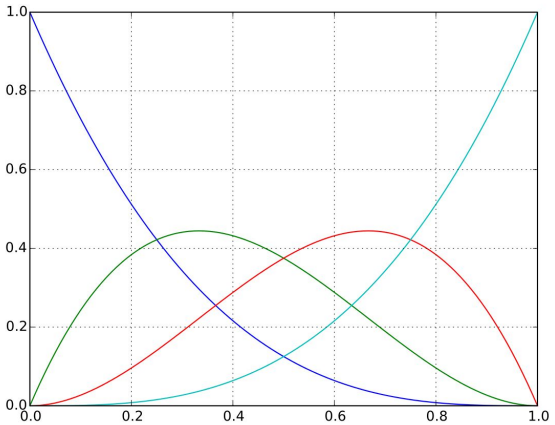
Bezier Curves - Blending Functions

Given a set of points, and a set of appropriate low degree polynomials - called *basis* or *blending functions*, we define the curve $\mathbf{p}$ as:

$$\mathrm{p}(u) = \sum_{k=0}^{d} b_k(u)\, p_k$$

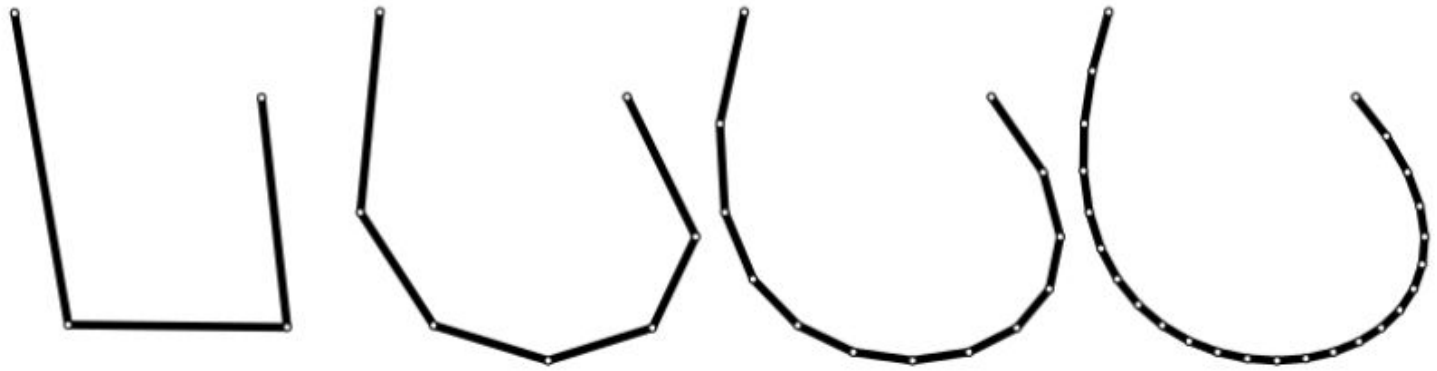
The weights at each parameter value sum up to 1 - this ensures the *convex hull property* - the curve will lie within the convex hull of the input points

B-splines and NURBS (non-uniform rational b-splines) extend this concept to provide greater control and flexibility



Rendering Curves

To draw a curve, need to identify the set of pixels that the curve maps to and set their color appropriately. Compute a *piecewise linear approximation* of the curve, and render the linear elements



SIGGRAPH 2000 Course Notes on Subdivision for Modeling and animation, D. Zorin and P. Shroeder

Rendering Curves

To draw a curve, need to identify the set of pixels that the curve maps to and set their color appropriately. Compute a *piecewise linear approximation* of the curve, and render the linear elements

Approach:

1. Generate a series of points that lie on the curve and that are “close enough” - as determined by the application
2. Map each such point to pixels in the viewing area
3. Identify pixels through interpolation (usually linear) between successive points
4. Compute the color for each such pixel

Steps 2 to 4 are computed in the graphics pipeline, including appropriate shaders

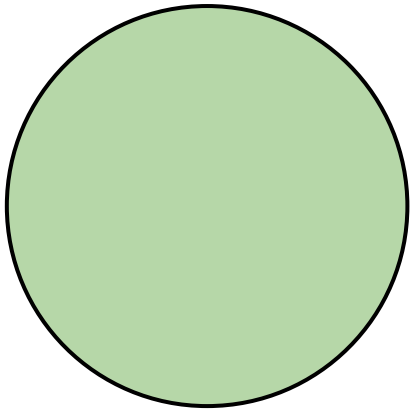
Surfaces

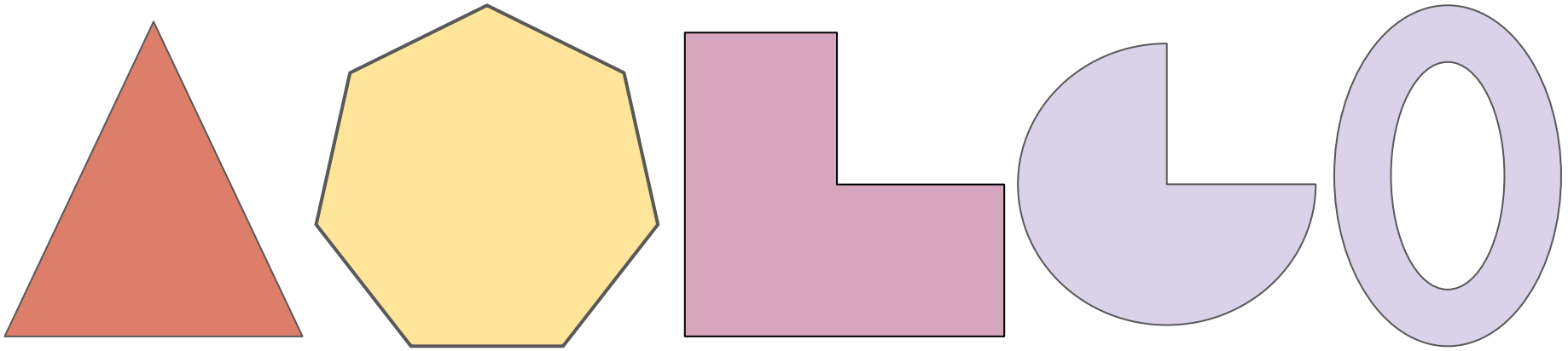
Regions in 2D

Closed or bounded regions in 2D: defined by their boundary and the region of 2D space that is *"inside"* the boundary. Boundaries can be of arbitrary geometry

E.g: circular disc defined by the circle that bounds it,

Polygonal region by the bounding polygon, etc.

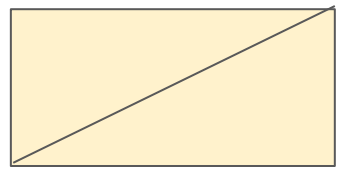


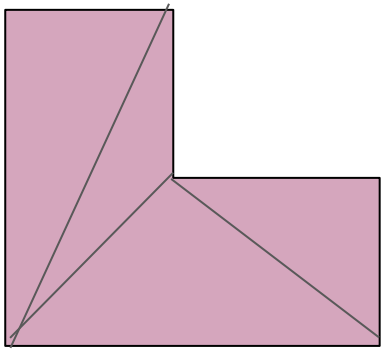


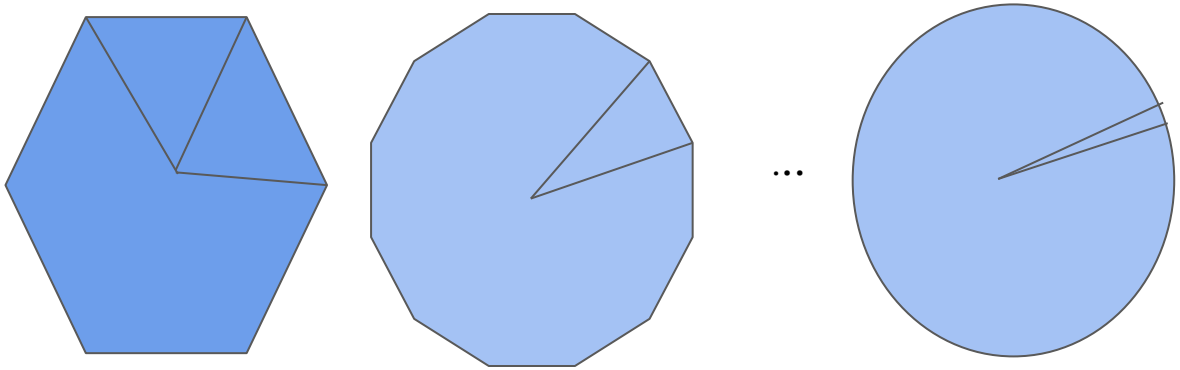
Rendering 2D Regions

1. Piecewise linear approximation of the boundary to the desired level of detail
2. Render all the points (pixels) that are *on* or *inside* the boundary. (rasterization)

To simplify graphics processing, triangulate (tessellate) the interior of the region, and render the triangles.







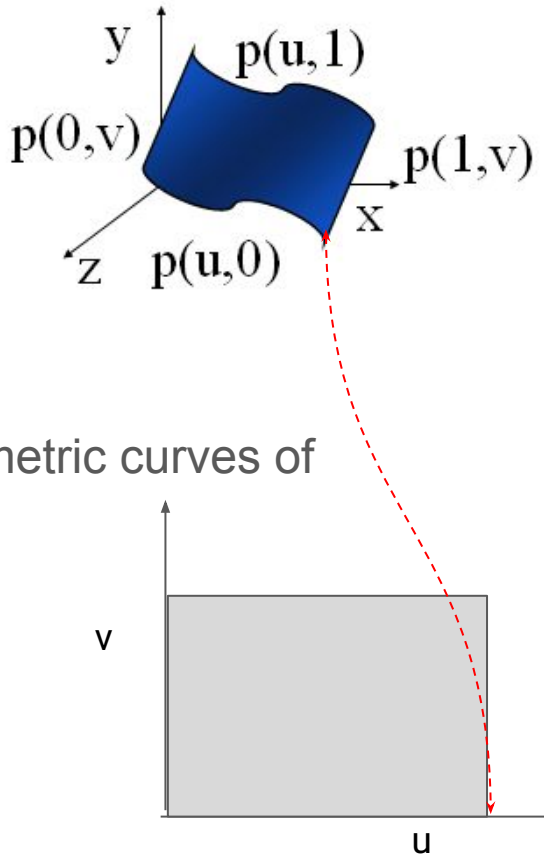
Circular disc rendered with boundary approximated by polygons of increasing number of sides

Surfaces in 3D

Parametric surfaces: A 2-dimensional object in 3D space

Defined as function of 2 parameters:

$$\mathbf{p}(u,v) = [x(u,v),\ y(u,v),\ z(u,v)]^T$$



Holding one of the parameters constant produces iso-parametric curves of the surface.

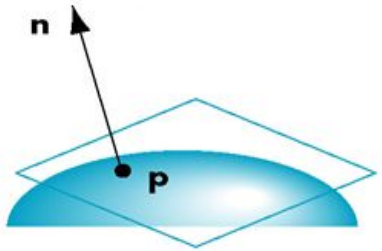
Trigonometric and polynomial maps commonly used.

Surfaces - Tangents and Normals

For parametric surfaces, tangents/tangent planes can be computed through *partial derivatives* w.r.t the parameter directions (essentially the derivative of the iso-parametric curves). This pair of vectors defines the *tangent plane*. The normal to this plane is the *normal* to the surface at that point.

$$\frac{\partial \mathbf{p}(u,v)}{\partial u} = \begin{bmatrix} \partial \mathrm{x}(u,v)/\partial u \\ \partial \mathrm{y}(u,v)/\partial u \\ \partial \mathrm{z}(u,v)/\partial u \end{bmatrix} \qquad \frac{\partial \mathbf{p}(u,v)}{\partial v} = \begin{bmatrix} \partial \mathrm{x}(u,v)/\partial v \\ \partial \mathrm{y}(u,v)/\partial v \\ \partial \mathrm{z}(u,v)/\partial v \end{bmatrix}$$

$$\mathbf{n} = \frac{\partial \mathbf{p}(u,v)}{\partial u} \times \frac{\partial \mathbf{p}(u,v)}{\partial v}$$



E. Angel and D. Shreiner: Interactive Computer Graphics

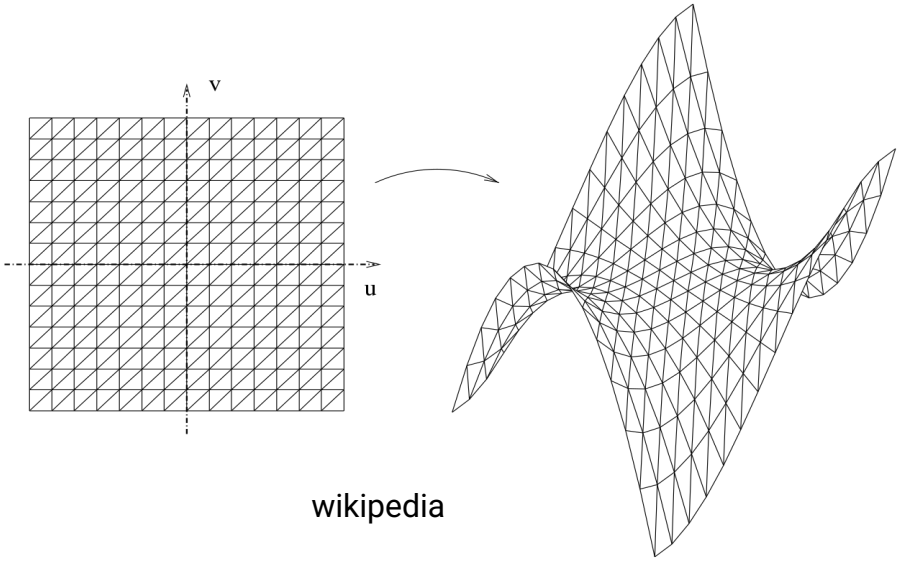
Rendering Parametric Surfaces

Compute a triangulation of parametric surfaces by computing a triangulation of the parametric space (easy, since this is a rectangular region).

Map the points of the triangulation to vertices on the surface and generate triangles - a piecewise linear approximation of the surface.

Render the individual triangles.

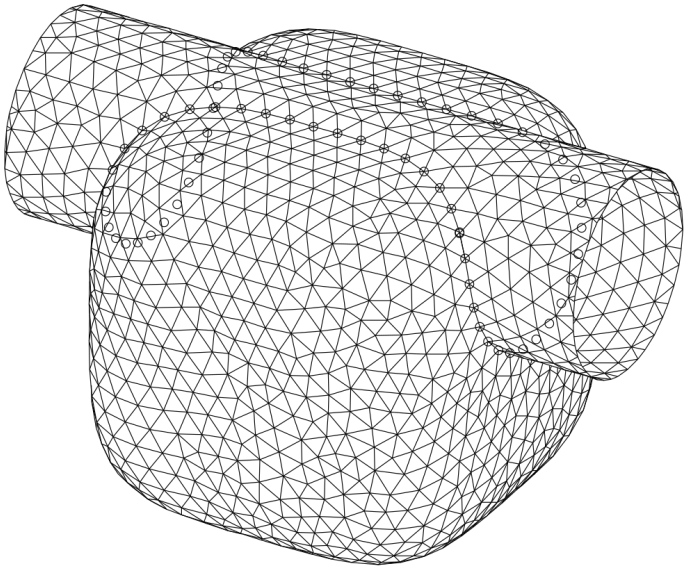
Density of triangulation determined by size of triangles required on the 3D surface.



wikipedia

Rendering Parametric Surfaces

Triangulation of parametric surfaces



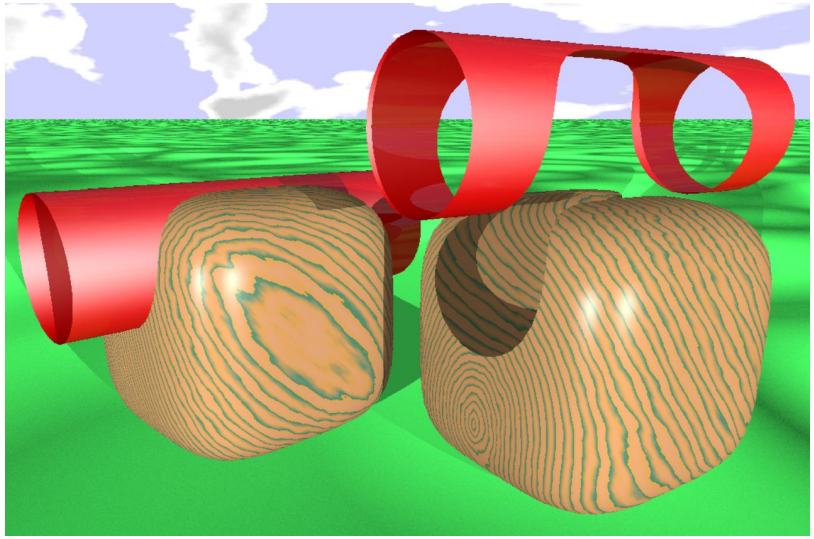


Image: Wikipedia. By Ag2gaeh

Surfaces with Irregular Boundaries

How would these be modelled and rendered?

Image source: Softimage

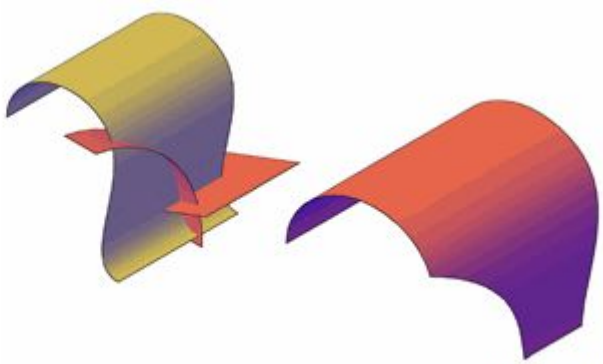
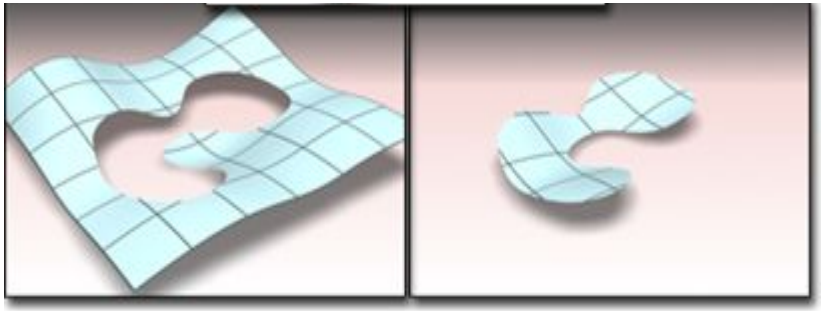


Image source: Autodesk: AutoCAD 2013

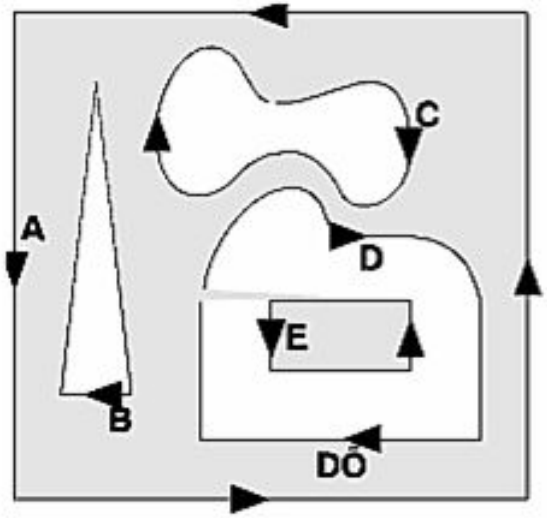


Trimmed Surfaces

Boundary of a surface can be seen as a set of connected curves - referred to as loops.

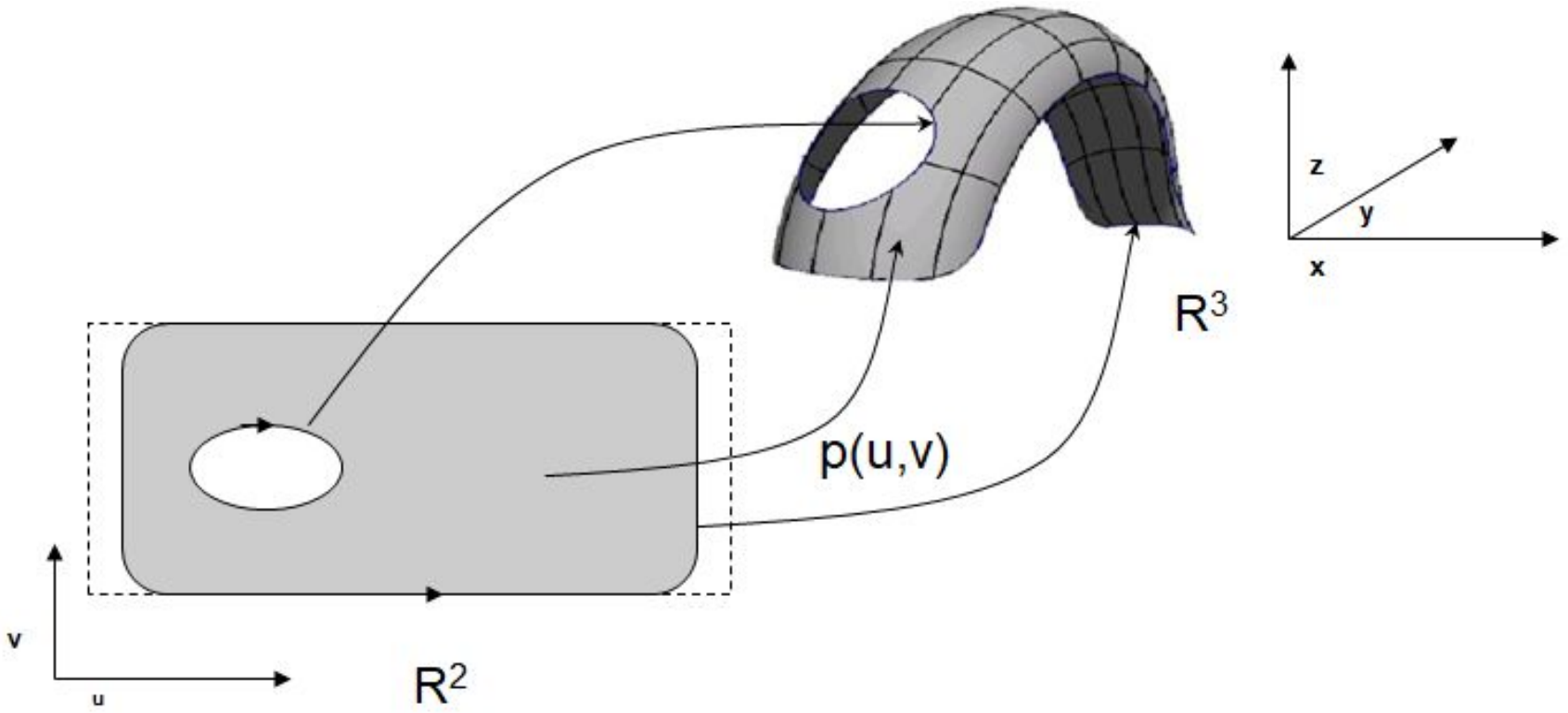
Based on the orientation of the loops, we can infer the “inside”, or the region of interest.

Loops can be defined in the parameter space of the surface or on the surface itself



GL Programming Red Book

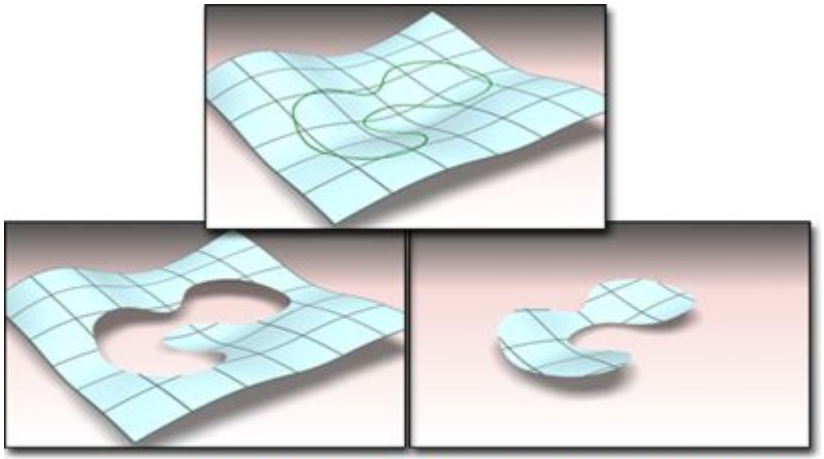
Trimming in Parameter Space



Surface Curves

Curves defined on the surface:

Surface Curves - can also be used to trim.



NURBS Surface Trimming, Autodesk 3DSMAX 2014

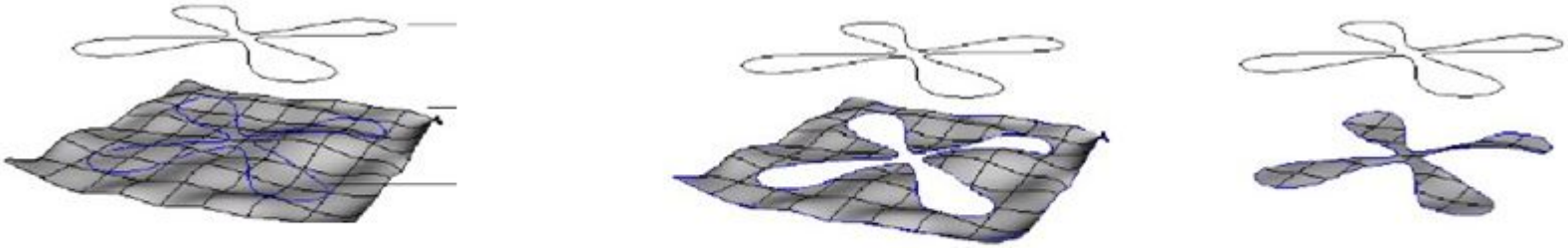
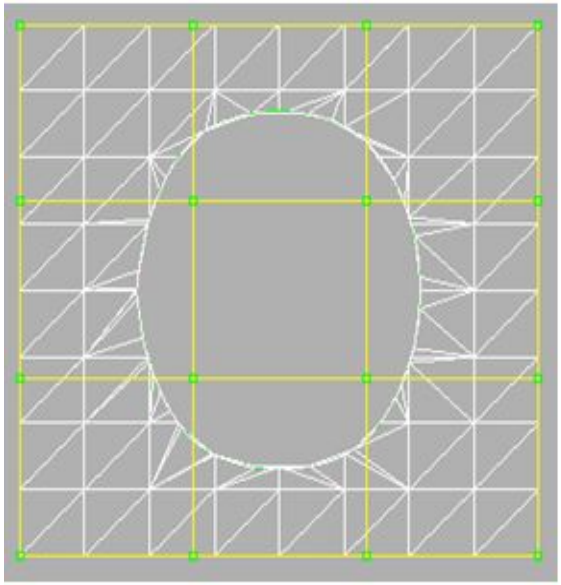


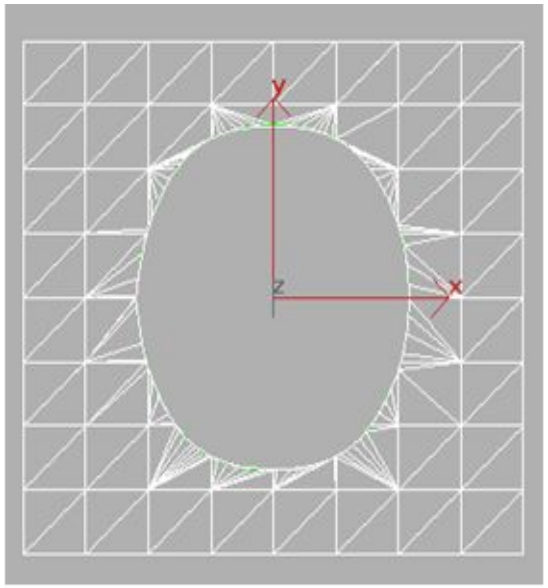
Image source: Softimage

Rendering Trimmed Surfaces

Compute the edges of the trimming curve. Use these as edges of the triangulation of the “interior” of the surface. Render these triangles as usual.

Shape, size, and density of triangles can influence quality and speed of rendering





Images: http://www.webreference.com/3d/lesson68/part3.html

Mesh Surfaces

Mesh Surfaces

While parametric surfaces provide a simple mathematical formulation and simplify evaluation, they are difficult to use for complex - and real-life - shapes.

Beziers, NURBS, etc are not convenient for modelling such objects, as well as clothing, etc.

Further, we sometimes have data of the points on a surface (e.g. from scanning) and need to compute a surface that “fits” these points

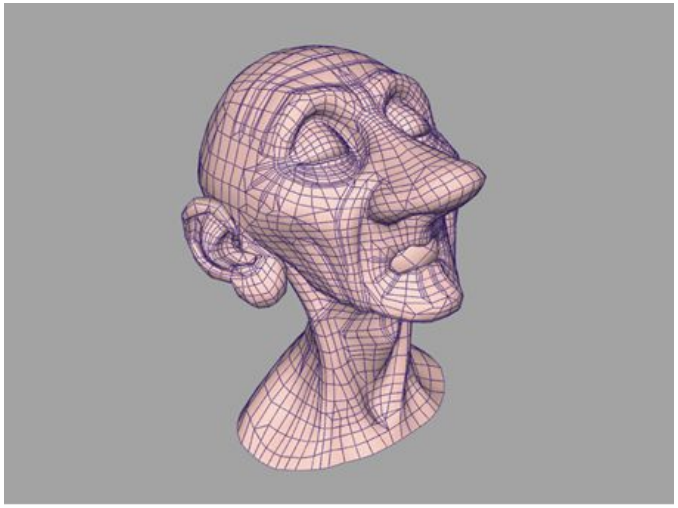


Image from: *Subdivision Surfaces in Character Animation*, Tony DeRose, Michael Kass, Tien Truong; Pixar Animation Studios

Mesh Surfaces

Model such objects directly as a 2D mesh:

- Surface defined by set of vertices, and connectivity between vertices that define triangles or quadrilaterals.

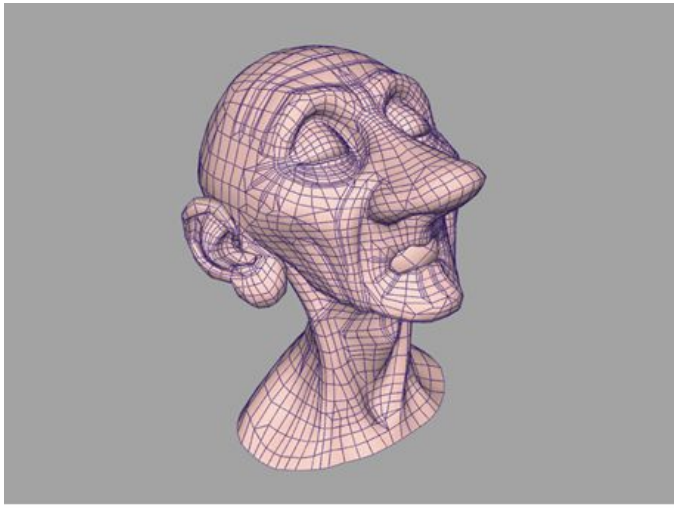


Image from: *Subdivision Surfaces in Character Animation*, Tony DeRose, Michael Kass, Tien Truong; Pixar Animation Studios

Mesh Surfaces

- Represent a curve or surface as connected sets of piecewise linear elements
  - Line segments (polylines) for curves (1D)
  - Polygon elements for surfaces (2D)
- Triangles and quadrilaterals most commonly used for surface meshes.
  - Other polygons reduced to some combination of the basic types.
- A preferred mechanism for modeling “natural” (i.e. not “manufactured”) surfaces, which typically do not have convenient analytical formulations.
- Special data structures such as half-edge, winged-edge, lath, etc used to enable efficient processing of large meshes
- Geometric properties of the object are derived from the mesh representation directly or through refinement of the mesh (e.g. using subdivision schemes)

Mesh Surfaces

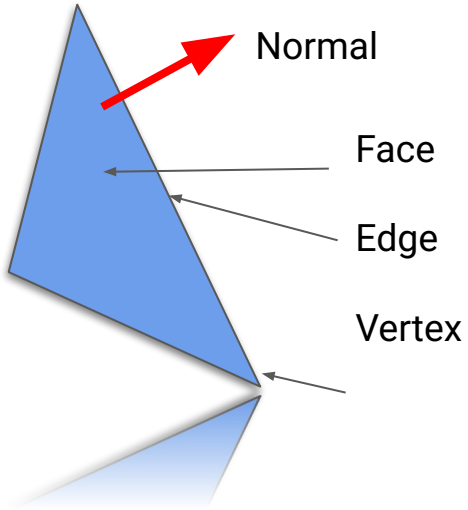
Mesh representation - vertices and connectivity - provides information on the shape of the object.

Vertices - Edges - Faces

* *Depending on the application, edges may not be explicitly represented*

Additional computations needed to compute properties such as surface normal and curvature

1. Normal at a vertex can be computed as a weighted average of the normals of the mesh elements (triangles) at that vertex
2. Use refinement techniques to compute better approximations of the mesh (e.g. subdivision schemes) and use these to compute normals etc



Rendering Mesh Surfaces

OpenGL/WebGL provide mechanisms to render triangles as groups of vertices. Thus, generate a vertex array from coordinates of the vertices and render in WebGL using **WebGLRenderingContext.drawElements()** with gl.TRIANGLES as the primitive mode. Also, TRIANGLE_STRIP, TRIANGLE_FAN

Options:

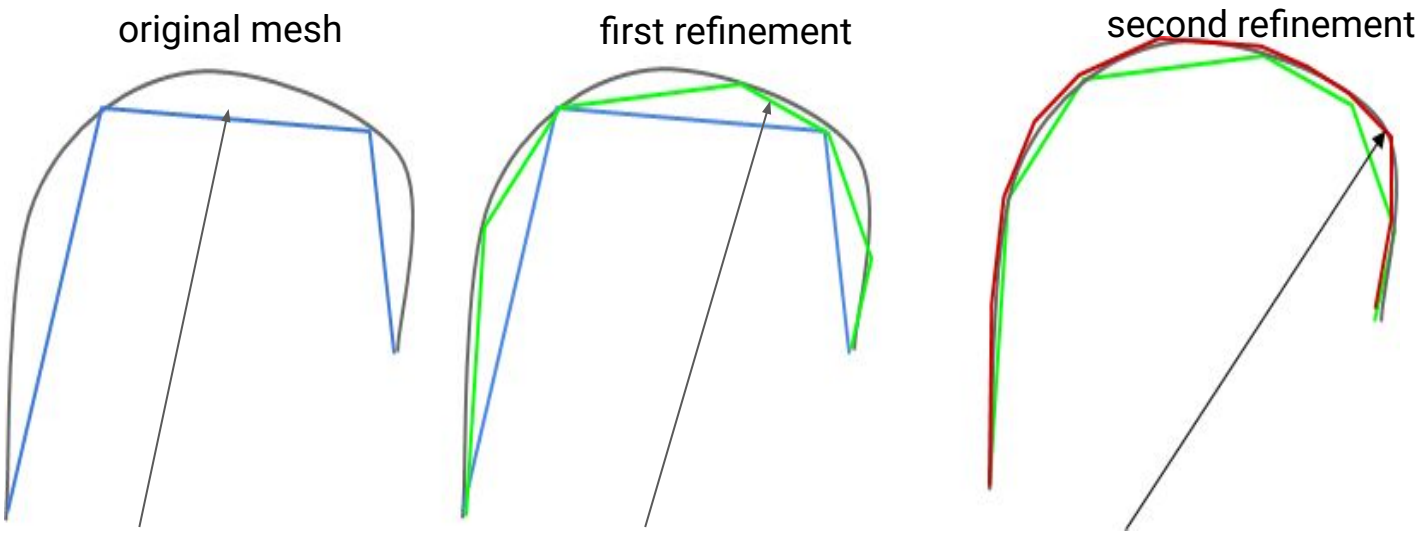
1. List the vertices for each triangle separately. That is, repeat vertices for each triangle that they bound. Provides flexibility in color and other computations
2. Have one copy of each vertex in the vertex array. Elements array provides triples of indices into the array. Optimizes on number of vertex computations

Vertex normals can be computed for the vertices of the model and sent in as attributes, to be used in the shader

Subdivision of Meshes

Subdivision defines a smooth curve (or surface) as the limit of a sequence of successive refinements. At each step, new vertices are generated and their coordinates computed based on local/neighbourhood geometry

Each refinement step produces a better piece-wise linear approximation of the “limit surface” – the hypothetical curve (or surface) obtained after an infinite number of subdivision steps.



Subdivision Surfaces

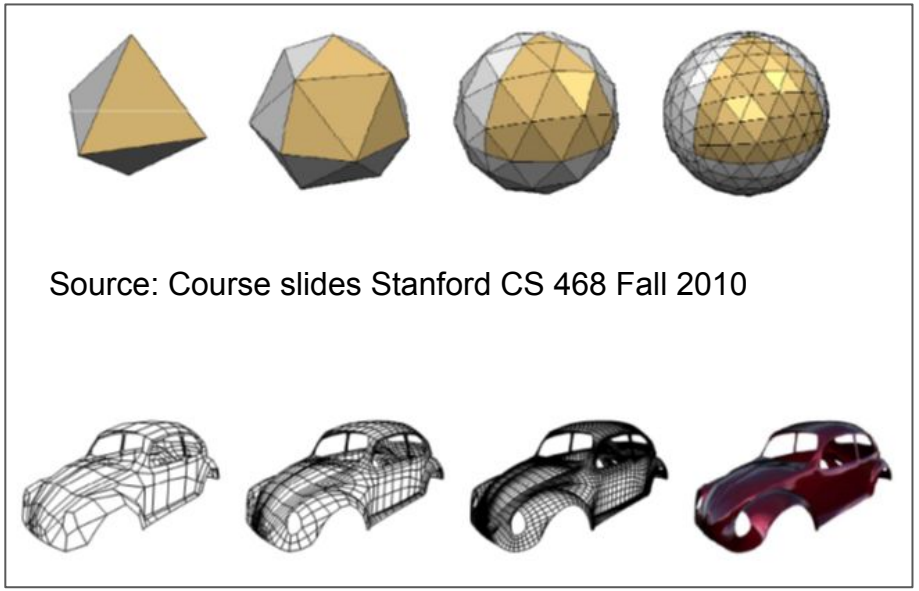
Successive refinement of meshes:

1. Add vertices in the interior of triangles.
2. Update coordinates of new vertices

Coordinates of new vertices computed through a weighted average of neighbour vertices. Existing vertices may also be “smoothed”

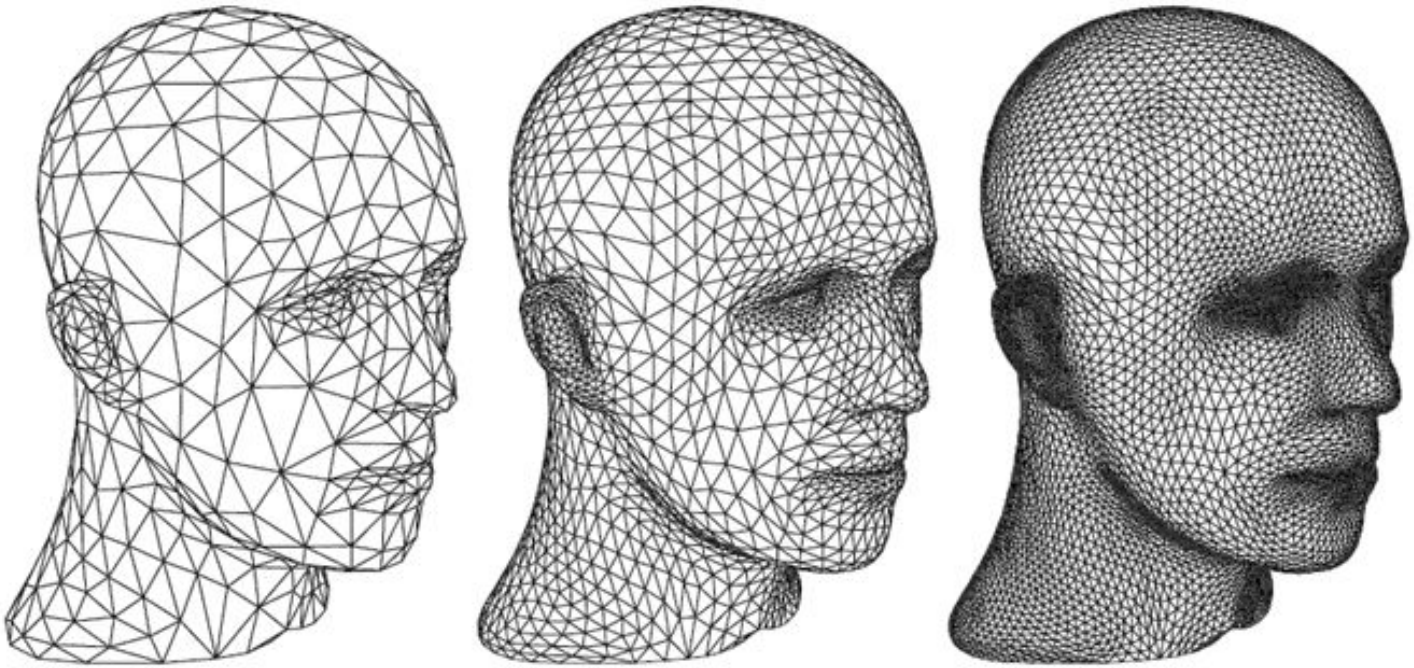
The smoothing depends on subdivision technique being used (Examples: Loop, Catmull-Clark)

Converges to a smooth surface



Source: Course slides Stanford CS 468 Fall 2010

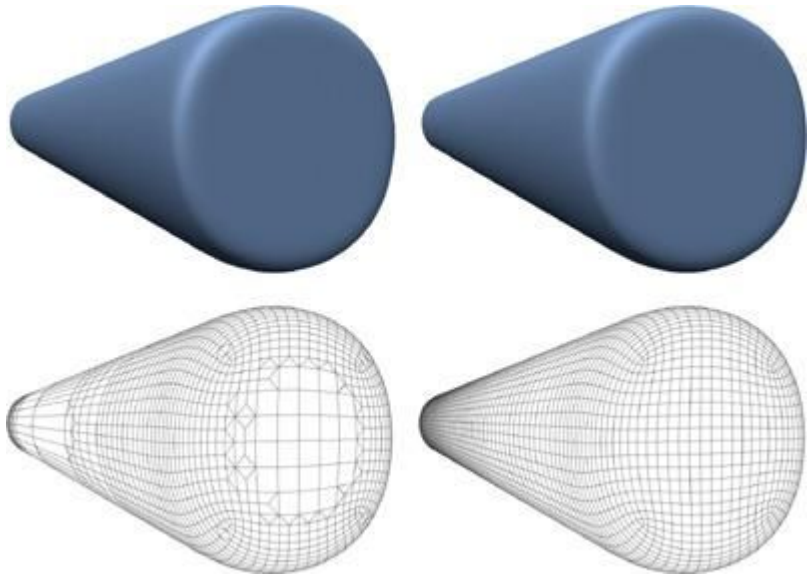
Refinement of Surface Meshes



The second and third images are produced by subdividing each triangle of the previous image into 4 triangles using a specific rule (Loop subdivision). SIGGRAPH 2000 Course Notes on Subdivision for Modeling and animation, D. Zorin and P. Shroeder

Adaptive Subdivision

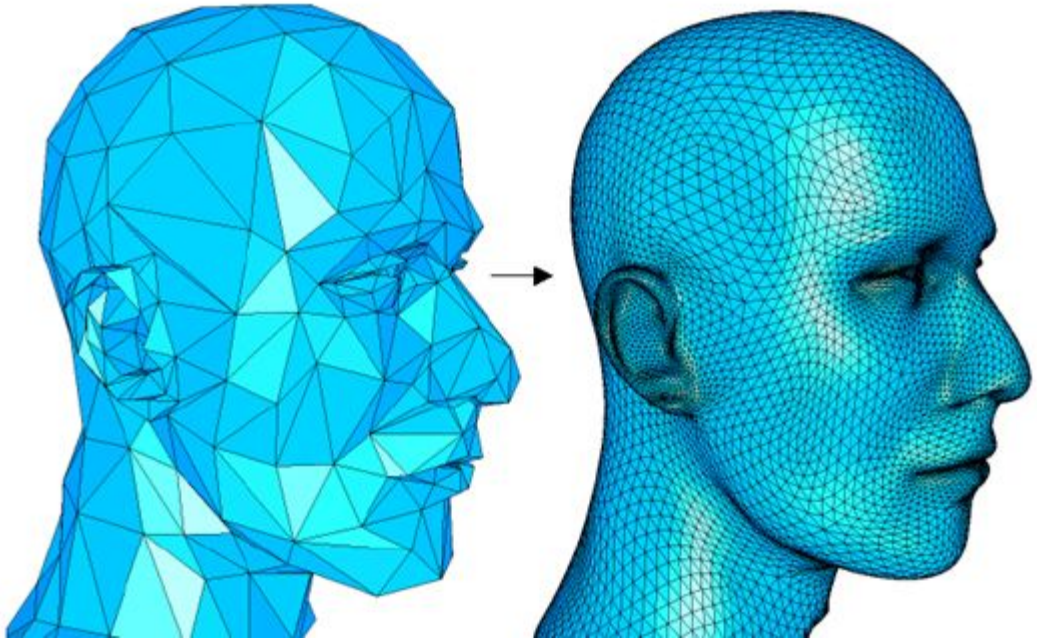
Note: subdivision need not be uniform. Certain triangles could be refined much finer than others, based on curvature or other features. Reduces size of mesh



Adaptive vs Uniform tessellation

(Source: Nvidia GPU Gems)

View dependent Adaptive Subdivision



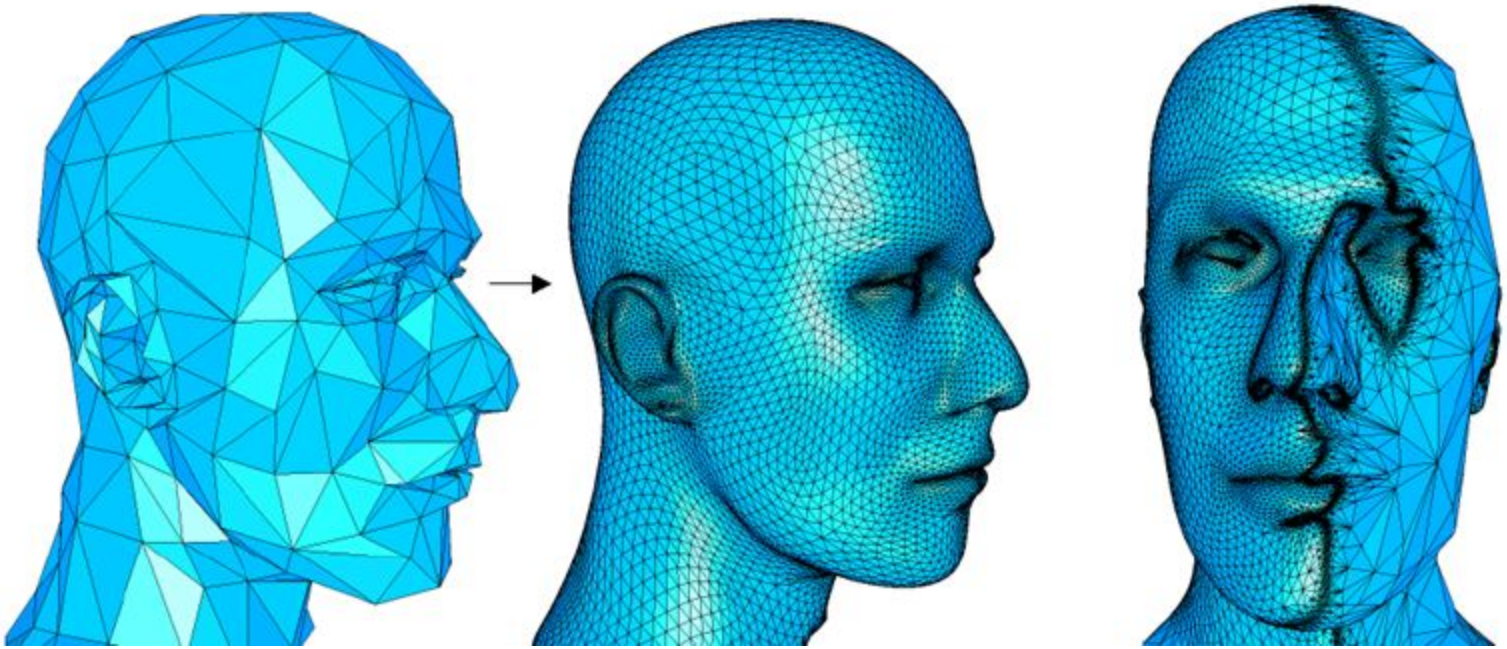
Mesh density needs to be greater where:

- significant features
- higher curvature
- region is visible
- along silhouettes
- higher pixel area on screen

From: Efficient View-Dependent Refinement of 3D Meshes using $\sqrt{3}$ – Subdivision

Pierre Alliez, Nathalie Laurent, Henri Sanson, Francis Schmitt

View dependent Adaptive Subdivision



From: Efficient View-Dependent Refinement of 3D Meshes using √3 – Subdivision

Pierre Alliez, Nathalie Laurent, Henri Sanson, Francis Schmitt

Topics Covered Today

- Geometry of Curves and Surfaces
  - Representation and Evaluation
- Curves
  - Bezier Curves
  - Rendering
- Surfaces
  - Shapes (Regions) in 2D
  - Parametric Surfaces
  - Trimmed Surfaces
  - Rendering
- Mesh Surfaces
  - Representation
  - Rendering in WebGL
  - Subdivision Surfaces