Introduction to 3D Shape Representations

Richard (Hao) Zhang

CMPT 464/764: Geometric Modeling in Computer Graphics

Lecture 3
Outline

- Implicit reps
- Parametric reps
- Polygonal meshes
- Point-sampled geometry
- Volumes
- Projective reps
- Subdivision surfaces

Smooth curves and surfaces

Discrete representations

3D $\rightarrow$ 2D

Limit surface is smooth but mostly treated as discrete
Why smooth curves & surfaces in CG?

- Naturally, to model smooth shapes, e.g.,
  - Body of an automobile
  - Shape of cartoon characters (Shrek)
  - Motion curves in animation
- Compact, analytical representation
- Smoothness can often be guaranteed analytically
- Theory of smooth curves and surfaces is well-developed

[Zorin 01]
Why smooth curves & surfaces in CG?

- Study of smooth curves and surfaces
  - e.g., the notion of arc length, area, curvature, surface normal, tangents, parameterization, etc.

forms the basis behind geometric modeling and processing using other primitives,

- e.g., polygonal meshes, subdivision surfaces, point clouds

- Much work in image and mesh processing involves
discretization of the theory for the continuous and the smooth
1. Implicit representations

Shape = \{ \mathbf{x} \in \mathbb{R}^k \mid f(\mathbf{x}) = 0 \},

e.g., for a plane, \( f(\mathbf{x}) = \mathbf{n} \cdot (\mathbf{x} - \mathbf{p}) \)

- \( f \): inside-outside function
  - \( \mathbf{x} \) is inside the shape, if \( f(\mathbf{x}) < 0 \)
  - \( \mathbf{x} \) is outside the shape, if \( f(\mathbf{x}) > 0 \)

For this to work effectively, \( f \) should be easy to evaluate
Where are implicit representations used?

- Bresenham line drawing algorithm
- Back face culling
- Intersections tests in ray tracing or collision detection
- Intermediate representation in surface reconstruction
- *** Evolve a surface by evolving its 3D scalar field, governed by a PDE — level-set methods — topology changes automatically enforced

Conversion between implicit and parametric is not always easy

Evolution of a 2D curve
Rendering of an implicit form $f(x) = 0$

Convert to discrete forms, e.g., a mesh
- In 2D case, overlay a regular grid
- Assign signs to grid points depending on $f$
  - $f(x) < 0$: $x \leftarrow -$  
  - $f(x) > 0$: $x \leftarrow +$
- Visit one cell at a time
  1. Linearly interpolate along edge to determine point of intersection
  2. Connect points depending on sign at corners

Generalization to 3D: Marching cubes
2. Parametric curves & surfaces

- **2D planar curve segment:**
  \[(x(t), y(t)), \quad t \in [0, 1]\]

- **3D space curve segment:**
  \[(x(t), y(t), z(t)), \quad t \in [0, 1]\]

- **3D surface patch:**
  \[(x(u, v), y(u, v), z(u, v)), \quad u, v \in [0, 1]\]
Use of polynomials

- In computer graphics, we prefer parametric curves and surfaces defined by polynomials

  - **Approximation power**: Can approximate any continuous function to any accuracy (Weierstrass’s Theorem)
  - Can offer local control for shape design through the use of piecewise polynomials
  - All **derivatives and integrals** are available (infinitely smooth) and easy to compute
  - **Compact representation**
  - **Efficient evaluation** — e.g., Horner’s rule, forward differences — for rendering and other processing
Polynomial evaluation: Horner’s rule

Consider a cubic polynomial

\[ b(t) = at^3 + bt^2 + ct + d \]

Straightforward way to evaluate \( b(t) \) takes 6 multiplications and 3 additions

Horner’s Rule:

\[ b(t) = ((a \times t + b) \times t + c) \times t + d \]

It takes 3 additions and 3 multiplications. In general, \( n +’s \) and \( n *’s \) for polynomial with degree = \( n \).
Poly evaluation: Forward differences

- Can do much better if $b(t)$ is evaluated at *equally spaced parameter values*, i.e., $0$, $\Delta t$, $2\Delta t$, $\ldots$, $n\Delta t = 1$

- Idea: do the evaluation incrementally

- Use finite differences, note that $\Delta t$ is a constant, for cubic $b(t)$

\[
\Delta b(t) = b(t + \Delta t) - b(t) \quad \text{: degree-2 polynomial in } t
\]

\[
\Delta \Delta b(t) = \Delta b(t + \Delta t) - \Delta b(t) \quad \text{: degree-1 polynomial in } t
\]

\[
\Delta \Delta \Delta b(t) = \Delta \Delta b(t + \Delta t) - \Delta \Delta b(t) \quad \text{: degree-0 polynomial = constant!}
\]
Poly evaluation: Forward differences

Evaluation of cubic $b(t)$ reduces to evaluation of quadratic $Δb(t)$

Evaluation of quadratic $Δb(t)$ reduces to evaluation to linear $ΔΔb(t)$

Evaluation of linear $ΔΔb(t)$ reduces to evaluation of constant $ΔΔΔb(t)$
Poly evaluation: Incremental evaluation

\[
\begin{align*}
  b &:= b(0) \\
  db &:= \Delta b(0) \\
  ddb &:= \Delta \Delta b(0) \\
  dddd &:= \Delta \Delta \Delta b(0)
\end{align*}
\]

for \( i = 2 \) to \( n \) do

\[
\begin{align*}
  b &+ db; & \text{// this is } & b( i\Delta t ) \! \\
  db &+ ddb; \\
  ddb &+ dddd;
\end{align*}
\]

- Only three additions inside the loop – extremely fast!
- Number of multiplications does not depend on \( n \)
Among parametric polynomial curves …

- What degree to choose for the polynomials?
  - Degree 0 – 2: simple but not enough flexibility
  - High-degree: unnecessarily complex and easy to introduce undesirable wiggles — most objects have a fair shape
  - Most commonly used in graphics as well as CAGD: parametric cubic curves and surfaces
- Let us focus on parametric curves for now
- Generalization to surfaces later: tensor-product surfaces
Scattered point interpolation

- Consider an interpolation problem:

What is the function here?
Consider an interpolation problem:

High-degree polynomial interpolant: \textit{smooth but not fair}
Fairness vs. smoothness

- Smoothness of curves and surfaces:
  - **Local property**: often achieved by design
  - Related to existence and continuity of various derivatives,
    e.g., \(3x^{100} - 9x^2 + \ldots\) is infinitely smooth, is it “visually pleasing”?

- Fairness (often appears in CAGD literature)
  - **Global property**: achieved by some form of energy minimization
  - Related to the “energy” of a curve or surface
    e.g., \(3x^{100} - 9x^2 + \ldots\) has high bending energy — not visually pleasing
Remedy: piece-wise polynomials

- The same interpolation problem:
  - High-degree polynomial interpolant: smooth but not fair

- Piece-wise cubic interpolation
  - Each segment is cubic
  - Allow many (> 4) constraints

- Remedy: piece-wise polynomials
Parametric cubic segment

Consider a single piece: 

\[ x(t) = a_3t^3 + a_2t^2 + a_1t + a_0 \]
\[ y(t) = b_3t^3 + b_2t^2 + b_1t + b_0 \]
\[ z(t) = c_3t^3 + c_2t^2 + c_1t + c_0 \]

In matrix form:

\[
\begin{bmatrix}
x(t)
\end{bmatrix} =
\begin{bmatrix}
t^3 & t^2 & t & 1
\end{bmatrix}
\begin{bmatrix}
a_3 \\
a_2 \\
a_1 \\
a_0
\end{bmatrix}
\]

or \[ x(t) = TA \]

\( T \) is said to be the **monomial basis**
Derivatives and continuity

- 1\textsuperscript{st}-order derivative of \((x(t), y(t))\): \((x'(t), y'(t))\) – tangent

- 2\textsuperscript{nd}-order derivative: \((x''(t), y''(t))\) – related to curvature

- **Parametric continuity** of a curve (**smoothness of motion**):
  - \(C^0\) continuous: curve is joined or connected
  - \(C^1\): requires \(C^0\) & 1\textsuperscript{st}-order derivative is continuous
  - \(C^2\): requires \(C^0\) & \(C^1\) & 2\textsuperscript{nd}-order derivative is continuous
  - \(C^n\): requires \(C^0\) & … & \(C^{n-1}\) & \(n\)-th derivative continuous
Curvature of plane curve

- Extrinsic vs. intrinsic definitions
- Intrinsic curvature at a point $p$ on a plane curve:
  
  Osculating circle: limit circle passing through $p$ and its neighbors
  
  Unit of curvature: inverse distance

- Extrinsic curvature at $p$ of plane curve $(x(t), y(t))$
  
  \[ \kappa = \frac{d\theta}{ds} = \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}} = \frac{1}{R} \]

  where $\theta$ is the turning angle and $s$ is arc length
Continuity of piecewise curves

- A single polynomial segment is always $C^\infty$
- But we mostly deal with **piecewise** polynomial curves
- Key: what happens at the joints between segments
  - $C^0$: curve segments are connected
  - $C^1$: $C^0$ & 1\textsuperscript{st}-order derivatives agree at joints
  - $C^2$: $C^0$ & $C^1$ & 2\textsuperscript{nd}-order derivative agree at joints, etc.
- If parametric continuity not possible to enforce, can relax to
  - “Visual” smoothness: direction of tangents stays the same but magnitude (speed) may change
Geometric continuity

- geometric continuity
  - $G^0$ continuous: curve segments are connected (same as $C^0$)
  - $G^1$: $G^0$ & 1st-order derivatives are proportional at joints.
  - Note:
    - Proportional = same direction but may have different magnitudes
    - Weaker than $C^1$
  - $G^2$: $G^1$ & 2nd-order derivative proportional at joints
- Example: $p(t) = (3t, t^3)$ and $q(t) = (4t+3, 2t^2+4t+1)$ with $t \in [0, 1]$ for each. Is this $C^0$, $G^1$, and/or $C^1$?
  - $p(1)=q(0)=(3,1)$, so $G^0$; $p'(1)=(3,3)$ and $q'(0)=(4,4)$, so $G^1$ not $C^1$
Now on to curve design

- Do you say to yourself,

  “I want to design a cubic curve $a_3 t^3 + a_2 t^2 + a_1 t + a_0$ with $a_3 = 1$, $a_2 = -9$, $a_1 = 4$, and $a_0 = 21$”?
Curves with the right design constraints

- Want to design piecewise cubic polynomial curves that satisfy certain design constraints, e.g.,
  - Curve should pass certain points
  - Curve should have some given derivatives at specific points
  - Curve should be smooth: $G^1$, $C^1$, $C^2$, or …
  - Curve must be contained in certain area, or has at most this length, etc.
- Need to use proper basis functions to facilitate the design process
- Often, the basis used identifies the curve representation
Basis functions and control points

- Recall basis expansion: \( x(t) = P_1 b_1(t) + P_2 b_2(t) + P_3 b_3(t) + P_4 b_4(t) \)

- Monomial basis, \( \{1, t, t^2, t^3\} \): only one of many possible bases for cubic polynomials

- From a design point of view, want \( P_1, P_2, P_3, \) and \( P_4 \) to represent **observable** quantities (not so for monomial basis), e.g.,
  - Position: for interpolation
  - Derivatives: to control direction and smoothness, etc.

- \( P_1, P_2, P_3, \) and \( P_4 \) serve as **control points**

- Control points are blended by the basis functions \( b_1, b_2, b_3, \) and \( b_4 \)
Ex. 1: Cubic Hermite curves

- Defined by two points \((P_1, P_4)\) and two tangents \((R_1, R_4)\)
- Aim: Achieve \(C^1\) or \(G^1\) continuity
- Want cubic curve \(x(t), t \in [0, 1]\), such that
  \[
  \begin{align*}
  x(0) &= P_1 \\
  x(1) &= P_4 \\
  x'(0) &= R_1 \\
  x'(1) &= R_4
  \end{align*}
  \]
  \((y \text{ and } z \text{ are similar})\)
- Usage example: determining the trajectory of a ball in animation

Let us note that the control “points” \(P_1, P_4, R_1,\) and \(R_4\) are all observable quantities and they control the shape of the curve.
Cubic Hermite curves

- \( x(t) = TA = a_3t^3 + a_2t^2 + a_1t + a_0 \), where \( T = [t^3 \ t^2 \ t \ 1] \) and \( A = [a_3 \ a_2 \ a_1 \ a_0]^T \). We want

\[
\begin{align*}
x(0) &= P_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}A \\
x(1) &= P_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}A \\
x'(0) &= R_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}A \\
x'(1) &= R_4 = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix}A
\end{align*}
\]

- So \( G = BA \) and thus \( A = B^{-1}G \)
- It follows that \( x(t) = TA = TB^{-1}G = HG \)
Cubic Hermite curves

- How to interpret this: \( x(t) = TA = TB^{-1}G = HG \)
  - \( G \): vector of observables or control points
  - \( H \): vector of cubic Hermite basis (blending) functions

\[
H = [2t^3 - 3t^2 + 1, -2t^3 + 3t^2, t^3 - 2t^2 + t, t^3 - t^2]
\]

- For any \( G \), use \( H \) to blend four control points to get curve \( x(t) \)
- The matrix \( M_{\text{hermite}} = B^{-1} \) is really a change of basis matrix: changes the monomial basis \( T \) into the Hermite basis \( H \)
- Hermite curves are completely determined by \( M_{\text{hermite}} \)
The cubic Hermite matrix

\[ M_{\text{Hermite}} = \begin{bmatrix}
2 & -2 & 1 & 1 \\
-3 & 3 & -2 & -1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix} \]

- The Hermite change of basis matrix or its basis identifies the **Hermite representation** of cubic parametric curves.
- Any cubic parametric curve can be specified in Hermite form.
Piecewise Hermite curves

- Can obviously enforce $C^1$ or $G^1$ continuity at the joints
- Each segment parameterized over $[0, 1]$ as usual

\[ R_1 = kR'_1 \]
\[ P_4 = P'_1 \]
Ex. 2: Cubic Bézier curve

- Defined by four control points $P_0, P_1, P_2,$ and $P_3$
  
  \[
  x(0) = P_0 \\
  x(1) = P_3 \\
  x'(0) = 3(P_1 - P_0) \\
  x'(1) = 3(P_3 - P_2)
  \]

- **Convex hull property**: Bézier curve lies within the convex hull of the four control points – good control

- Convex hull of a set of points on the plane: tightest convex polygon enclosing the set – why would it be useful in graphics?
Convex hull property

- A cubic curve satisfies the convex hull property if it lies within the convex hull of its four control points.

- Convex hull property is satisfied if and only if the basis polynomials $b_1(t), b_2(t), b_3(t), b_4(t)$ satisfy:
  
  1. $0 \leq b_1(t), b_2(t), b_3(t), b_4(t) \leq 1$ for $t \in [0, 1]$, and
  2. $b_1(t) + b_2(t) + b_3(t) + b_4(t) = 1$

- Then each point of the curve is a convex combination of the control points.

- The basis $b_i(t)$ form a partition of unity.
Cubic Bezier change-of-basis matrix

\[
M_{\text{Bezier}} = \begin{bmatrix}
-1 & 3 & -3 & 1 \\
3 & -6 & 3 & 0 \\
-3 & 3 & 0 & 0 \\
1 & 0 & 0 & 0
\end{bmatrix}
\]

Symmetric matrix!

Exercise: derive the Bezier change of basis matrix
Bézier basis functions

\[ B_0(t) = (1 - t)^3, \ B_1(t) = 3t(1 - t)^2, \]
\[ B_2(t) = 3t^2(1 - t), \ B_3(t) = t^3 \]

- Well known as the **Bernstein Polynomials** of degree 3
- Bernstein polynomials of degree \( n \)
- We have (a recursion)

\[ B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i} \]

\[ B_i^n(t) = (1 - t)B_i^{n-1}(t) + tB_{i-1}^{n-1}(t) \]

- Partition of unity easy to see: \( \sum_i B_i(t) = [t + (1 - t)]^n \)
Piecewise Bézier curves

- How to ensure $C^1$ or $G^1$ for piecewise Bézier curves?
- Each segment is parameterized over $[0, 1]$ as usual

A constraint: not perfectly flexible
Ex. 3: Cubic B-spline curve

- Each cubic B-spline segment is specified by four control points
- Has the convex hull property
- No interpolation in general
- **Big advantage: C\(^2\) continuous**
- The cubic B-spline change of basis matrix

\[ M_{B\text{-spline}} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \]
Piecewise cubic B-splines

- Two consecutive segments share three control points
- \( m \) control points \( \rightarrow m - 3 \) segments
- Exercise: Prove \( C^2 \) continuity for cubic B-splines
- Exercise: what if control points repeat?
From curves to surfaces

- One easy way: sweep a curve whose control points also trace out some curves, e.g., bilinear interpolation
- Fit the simplest surface between four points
- Sweep a straight line and each point on the line traces a straight line
- An example of a ruled surface
- Also an example of tensor-product surfaces
3. Tensor-product (TP) surfaces

- The curve to sweep:
  \[ p(u) = \sum_{i=0}^{m} a_i A_i(u) \]

- Control point \( a_i \) goes through a curve
  \[ a_i = q_i(v) = \sum_{j=0}^{n} P_{ij} B_j(v) \]

- The resulting surface is a tensor-product surface
  \[ p(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} A_i(u) B_j(v) = A(u)^T P B(v) \]

- Surface is controlled by the grid of control points \( P_{ij} \)
3. Tensor-product (TP) surfaces

\[ p(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{ij} A_i(u) B_j(v) = A(u)^T \mathbf{P} \mathbf{B}(v) \]

\[
\begin{bmatrix}
A_0(u) & A_1(u) & A_2(u) & A_3(u)
\end{bmatrix}
\begin{bmatrix}
B_0(v) \\
B_1(v) \\
B_2(v) \\
B_3(v)
\end{bmatrix}
\]
Ex. 1: TP cubic Bézier patch

\[ p(u, v) = \sum_{i=0}^{3} \sum_{j=0}^{3} P_{ij} B_i^3(u) B_j^3(v) = B(u)^T P B(v) \]

- \( B_i(u) B_j(v) \)'s form a basis for bi-cubic polynomials
- Partition of unity — patch lies within the convex hull of 16 control points \( P_{ij} \)
- Patch can be seen as generated by sweeping a Bézier curve where each point on the curve sweeps out a Bézier curve
- The four corner vertices are interpolated
- The tangent plane at each corner interpolates the corner vertex and its two immediate neighbors
Cubic Bézier surface patch

In matrix form:

\[ p(u, v) = B(u)^T P B(v) = u^T M_{\text{Bezier}} P M_{\text{Bezier}}^T v, \quad P = [P_{ij}] \]
Ex. 2: TP cubic B-spline surfaces

- Analogues extension of B-spline curves

- $N(u)$ and $N(v)$ are the cubic B-spline bases

- 16 control points per bicubic B-spline patch

- Satisfy the convex hull property

$$p(u, v) = \sum_{i=0}^{m} \sum_{k=0}^{n} P_{ik} N_{m,i}(u)N_{n,k}(v)$$
Smoothness of Bézier surface

- $G^1$ continuity: make 2-sets of 4 control points on either side of an edge collinear [Faux 79]
- $C^1$ continuity: collinear and equally spaced
Smoothness of B-spline surface

- $C^2$ continuity is achieved if adjacent patches share control points
Curvature of surfaces

- **Regular point** on a surface
  - Consider all curves lying in the surface through the point
  - Point is regular if tangent vectors of all these curves lie in the same plane — the **tangent plane**

- Surface **normal** at regular point: normal to tangent plane

- Intersection between surface and a plane through the normal is called a **normal section**

- **Principal curvatures**: maximum ($\kappa_1$) and minimum ($\kappa_2$) curvatures of the normal sections
Curvature of surfaces

- **Gaussian curvature:** \( K_1 K_2 \)

- **Mean curvature:** \( \frac{K_1 + K_2}{2} \)

- For a regular point, the two **principal (curvature) directions are perpendicular**

- **Elliptic, hyperbolic, parabolic, umbilical points**

For an elliptic point, both principal curvatures are positive; for a hyperbolic point, they are both negative; for a parabolic point, one of the principal curvatures is zero; for an umbilical point, both principal curvatures are equal.

all the same?!
Mini summary: what to take away

- Rendering of implicit forms: Marching squares/cubes
- Fast polynomial evaluations: Horner’s rules
- Fairness vs. smoothness – need for piecewise representations
- Curvatures: along a curve or on a surface
- Derivation of COB matrix from observable constraints
- Hermite, Bezier, and B-splines
- Parametric vs. geometric continuity
- Tensor-produce surfaces: bi-cubic Bezier and B-spline surfaces
Shape modeling using TP surfaces

- A complex shape is divided into patches and each patch is represented by a TP surface

- Pros
  - Compact representations (just the control points)
  - Analytical measures, e.g., normal, curvature

- Cons
  - Extreme care needed to maintain continuity across patches, especially during animation
  - Hard to cut through a patch, e.g., for editing
  - When rendering, still need to covert patches to discrete representations, e.g., triangles or quads
4. Polygonal meshes

- Polygonal mesh: composed of a set of polygons pasted along their edges – triangles most common
- Some issues: vertices, edges, faces, **manifold or not**, boundaries, valences, normals, curvatures, etc.
- A more general “mess”: triangle/polygon **soup**
Meshes vs. TP: Pros and cons

- Primitive
- Ease to model objects with arbitrary shape, topology, and details – it gets truly realistic
- One piece and seamless
- Hardware accelerated rendering
- Model complexity: need compression and/or level-of-detail modeling
- Non-functional: need parameterization
- Connectivity is **artificial** and can be **irregular**
5. point-based reps (PBR)

- A 3D surface model is represented using a set of **points near the surface**
- There is no (explicit) connectivity information between the points
- Typically need \( k \)NN – **\( k \) nearest neighbors** – during processing
- Colors and normals (how to estimate?) can also be specified for rendering
A historical note

- “As the visual complexity of computer generated scenes continue to increase, the use of classical modeling primitives (polygons) as display primitives becomes less appealing.”
  
  Levoy and Whitted, “The Use of Points as a Display Primitive”, 1985

- Use of points traces back to modeling of smoke, fire, and cloud around the late 70’s [Csuri et al. 79, Blinn 82]
Renewed interests: twice!

- PBR was witnessing a revival around 2002 - 2012
- Points are directly available via laser scanning
  - Substantial advances in 3D digitizing and laser scanning and acquisition technology
  - High quality points (color and texture) easily obtained
  - Cheap scanners (< $3,000) available now
3D digitizer and scanner

- InSpeck
- FastScan ($23K)
- NextEngine (< $3K)
- Cyberware
- Roland DGA LPX-250 ($10K)
Point-based rep (PBR) for complexity

- Complexity of geometry will continue to grow
- Overhead associated with storing triangle meshes becomes less justified when size of triangles become much less than a pixel
  - Either go for **higher-order** surface descriptions (compact)
  - Or **simpler primitives** such as points (also compact!)
- Points enable interactive rendering, compact storage, LOD, editing, etc. : A LOT of research works around 2002-2012
Second revival: deep learning

- **PointNet and PointNET++, since 2016/17**

- Deep neural network to **encode and aggregate point features** for shape recognition, segmentation, etc.

- Points offer more **flexible shape representations**, but so far the networks can deal with quite **low resolutions**, e.g., 2,048 points
6. Volumetric representations

- Even more frequently adopted for deep learning of 3D shapes
- Embed 3D shape in regular volumetric grid: 3D shape = set of all voxels that lie on or inside shape
- Closely related to implicit reps and support similar operations
- Closely related to image reps: it is a 3D image
6. Summary and other reps

- Unlike speech, images, or video, there is no universally accepted rep or encoding for 3D shapes.
- Low-level reps: mainly for rendering and viz.

Mesh: set of triangles  Volume: set of voxels  Point cloud: set of points
Higher-level representations

Procedural: e.g., CSG  Structural: set of parts  Hierarchical organization of parts
Most neural networks (NNs) for these reps, e.g., MVCNN, PointNET, O-CNN, etc., are designed to carry out discriminative analyses. Not many would allow 3D shape generation or synthesis.
Exercises

- Read Section 3.4 of [Zorin 00], Subdivision Course Notes, on definition of smoothness of surfaces

- Design a curve of your own, e.g., interpolate two end points and interpolate position and tangent at midpoint – compute C.O.B. matrix

- Identify the curve …
  - de Casteljau algorithm