CMT107 Visual Computing

IX.1 Curves

Xianfang Sun

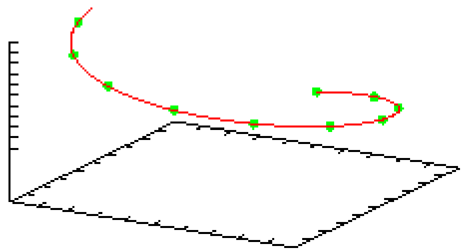
School of Computer Science & Informatics
Cardiff University

Overview

- Curve representations
  - Explicit representation
  - Implicit representation
- Parametric representation of curves
  - Piecewise polynomial curves (spline curves)
  - Bézier curves

Curves

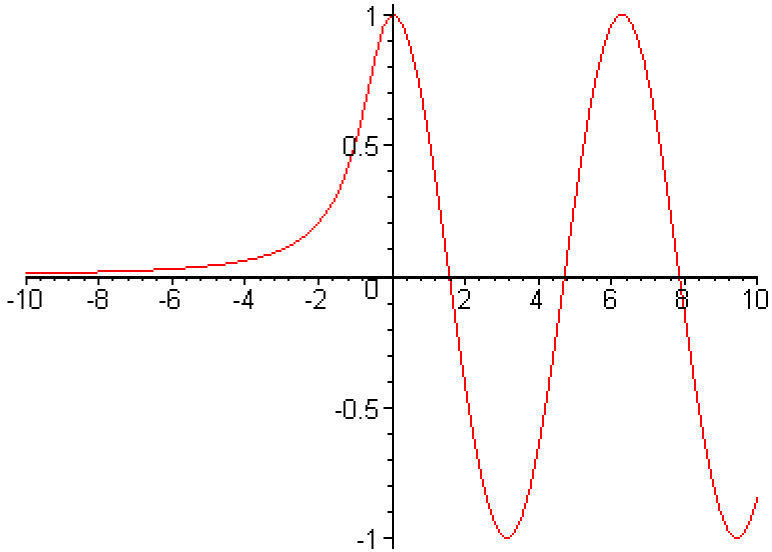
- A curve is a set of positions of a point moving with one degree of freedom



- Useful to describe shapes on a *higher level*
  - Not only straight lines or curved shapes approximated by short line segments
  - Simpler to create, edit and analyse
  - More accurate rendering and less storage (compared to linear approximation)

Explicit Representation

- *Explicit curve*: $y = f(x)$
  - Essentially a *function plot* over some interval $x \in [a, b]$



- Properties:
  - Simple to compute points and plot them
  - Simple to check whether a point lies on curve
  - Cannot represent closed or multi-valued curves:

    Only one y value for each x value (a function)

Implicit Representation

- Define curves *implicitly* as *solution of an equation system*
  - Straight line in 2D: $Ax + By + C = 0$
  - Circle of radius R in 2D: $x^2 + y^2 - R^2 = 0$
  - Conic section: $Ax^2 + 2Bxy + Cy^2 + Dx + Ey + F = 0$
  - Matrix/vector representation up to order two:

$$\boldsymbol{x}^T M \boldsymbol{x} + v^T \boldsymbol{x} + s = 0 \quad (\boldsymbol{x} = [x\ y]^T)$$

- In 3D, *two* equations are needed
  (*1 equation restricts 1 variable*, but there are 3 variables)
  - Straight line: $Ax + By + Cz + D = 0,$

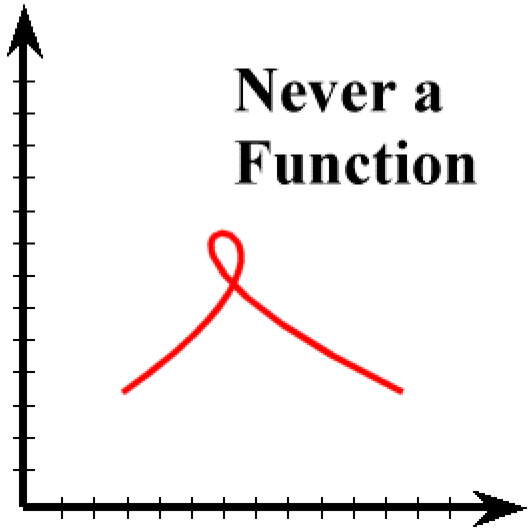
$$Ex + Fy + Gz + H = 0$$

  - A circle in x-y plane: $x^2 + y^2 = r^2,$

$$z = 0$$

Properties of Implicit Curves

- Mainly use polynomial or rational functions
- Coefficients determine geometric properties
- *Properties:*
  - Hard to render (have to solve non-linear equation system)
  - Simple to check whether a point lies on curve
  - Can represent closed or multi-valued curves



Parametric Curves

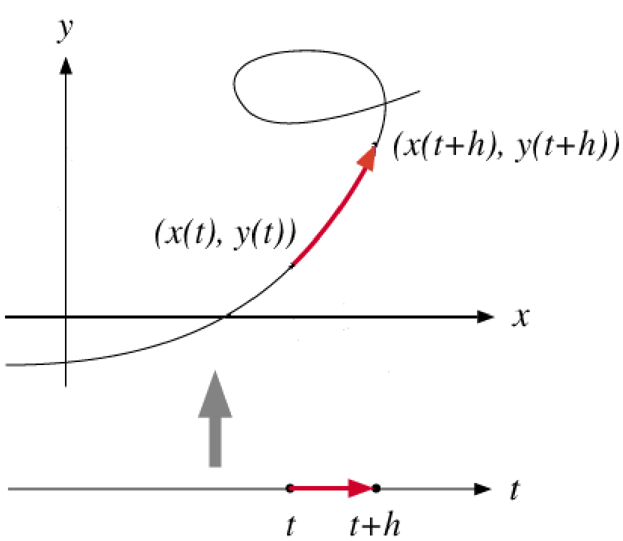
- Describe the position on the curve by a parameter $u \in R$

$$c(u) = \begin{pmatrix} x(u) \\ y(u) \\ z(u) \end{pmatrix}$$

  - $x(u)$, $y(u)$, $z(u)$ are usually polynomial or rational functions in u
  - $u \in [a, b]$, usually $u \in [0, 1]$
- Parameter function maps parameter to model coordinates
  - Parameter space: u (parameter domain)
  - Model space: x, y, z (Cartesian coordinates)

Properties of Parametric Curves

➢ *Properties*:
- Simple to render (evaluate parameter function)
- Hard to check whether a point lies on curve (must compute inverse mapping from (x, y, z) to u; involves solving non-linear equations)
- Can represent closed or multi-valued curves

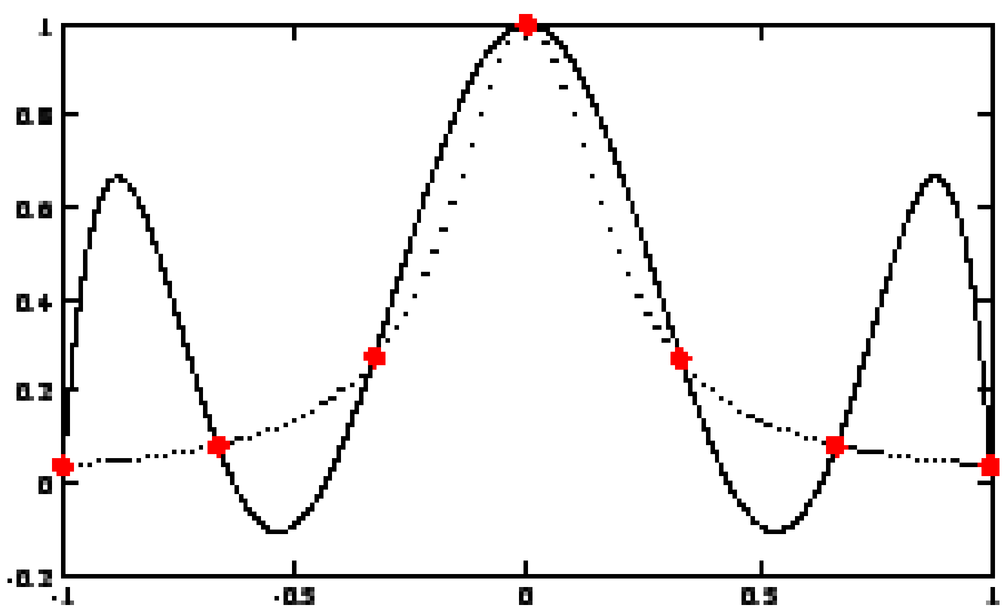


Parametric Polynomial Curves

- Describe coordinates by *polynomials*:

$$x(u) = \sum_{l=0}^{d} A_l u^l, \quad y(u) = \sum_{l=0}^{d} B_l u^l, \quad z(u) = \sum_{l=0}^{d} C_l u^l$$

- *Smooth* (infinitely differentiable)
- Higher order curves (say > 4) cause *numerical problems*
- Hard to control shape by *interpolation*



Bernstein Polynomials

- Bernstein *basis* polynomials

$$b_l^d(u) = \binom{d}{l} u^l (1-u)^{d-l},\ l = 0,1,\cdots,d.$$

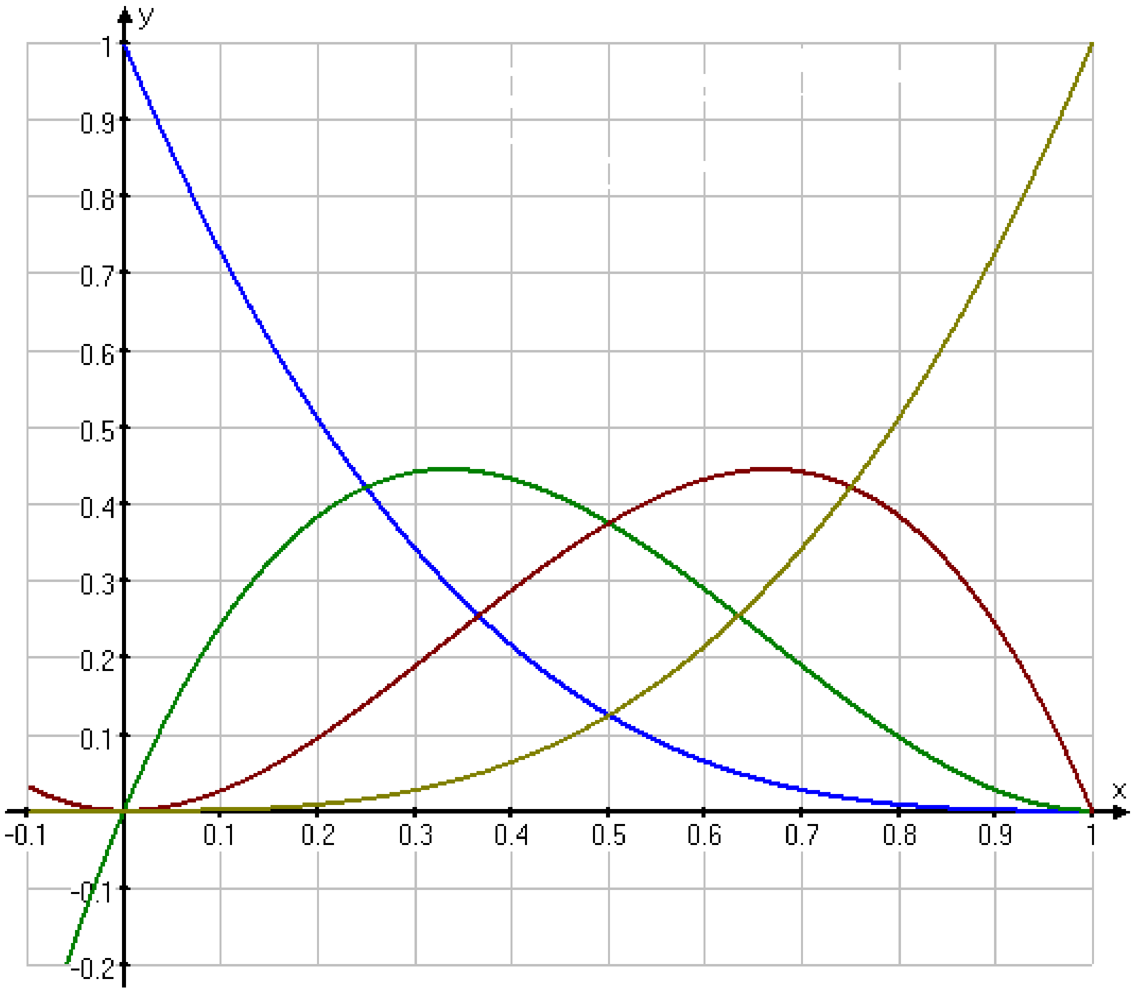
- $\binom{d}{l} = \dfrac{d!}{l!(d-l)!}$ is binomial coefficient.
- Property: $\sum_{l=0}^{d} b_l^d(u) = 1 \text{ for } u \in [0, 1]$

- A Bernstein polynomial is a linear combination of Bernstain basis polynomials

$$B(u) = \sum_{l=0}^{d} \beta_l b_l^d(u),\ u \in [0, 1].$$

Cubic Bernstein Basis Polynomials

- There are 4 cubic Bernstein basis polynomials

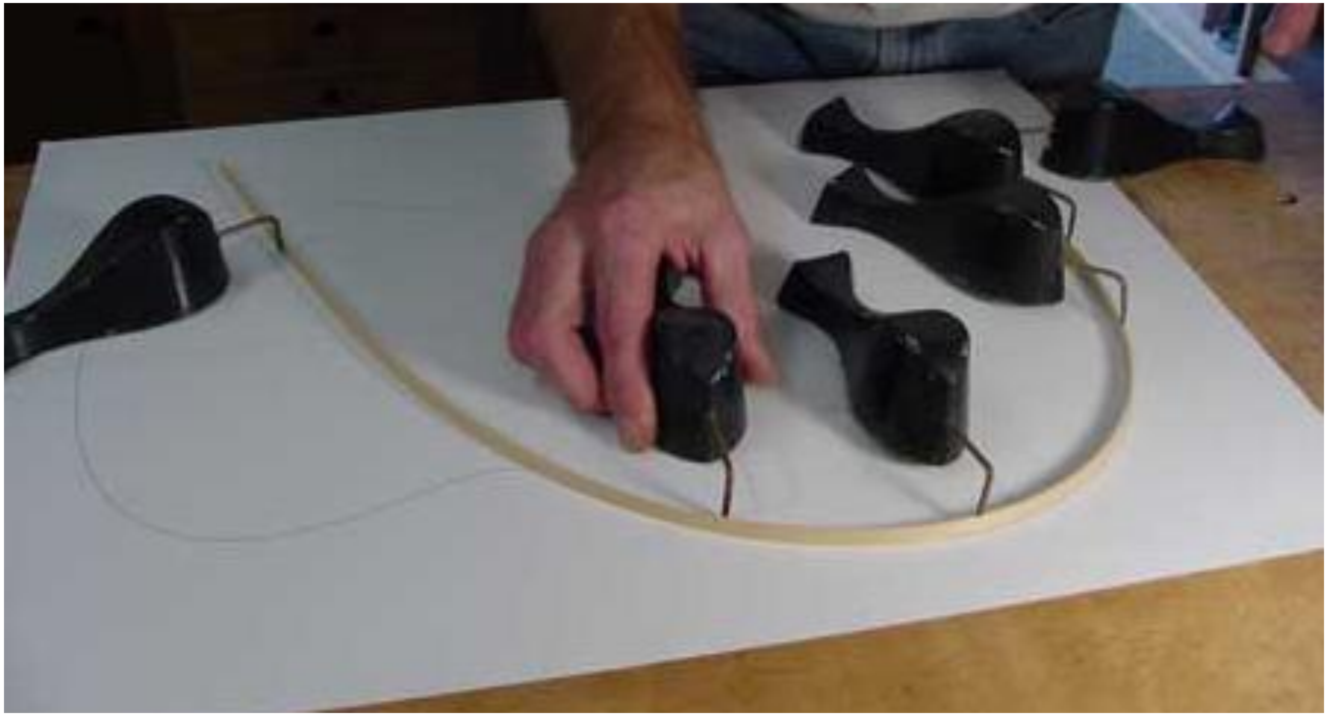


Piecewise Polynomial Curves

- Cut curve into *segments* and represent each segment as *polynomial* curve
- Can use *low-order polynomial* curves, e.g. cubic (order 3)
- But how to guarantee *smoothness at the joints*?
  - Continuity problem

Spline Curves

- In general, piecewise polynomial curves are called splines
  - Motivated by loftsman's spline
    - Long narrow strip of wood or plastic
    - Shaped by lead weights (called ducks)
  - Gives curves that are smooth or fair

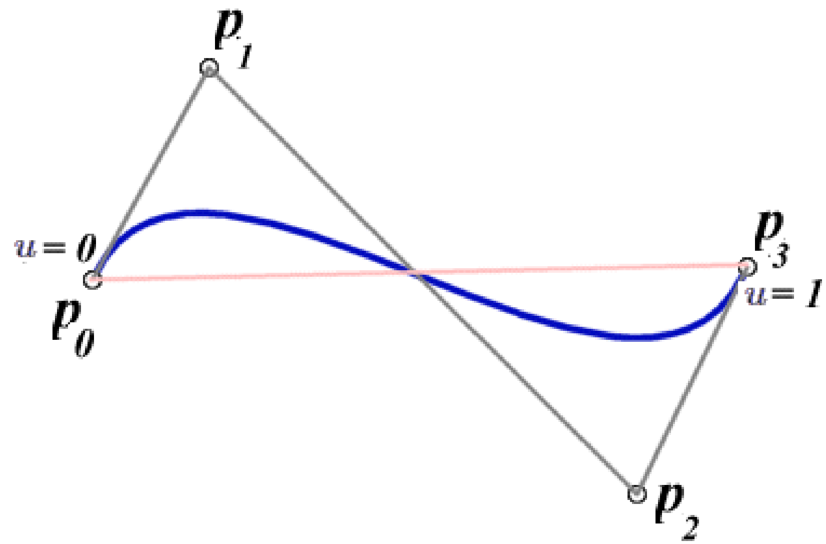


Bézier Curves

- Represent a polynomial segment as

$$Q(u) = \sum_{l=0}^{d} p_l b_l^d(u),\ u \in [0,1]$$

$$Q(u) = \sum_{l=0}^{d} p_l \binom{d}{l} u^l (1-u)^{d-l},\ u \in [0,1]$$



- Control points $p_l \in R^3$ or R^2 determine segment's shape
- $b_l^d(u)$: l^{th} *Bernstain basis polynomial* of degree d.

- Cubic Bézier curve ($d = 3$) has four control points
  - Note that $\sum_{l=0}^{d} b_l^d(u) = 1 \text{ for } u \in [0,1]$

    ➡ *Convex combination* of control points

Properties of Bézier Curves

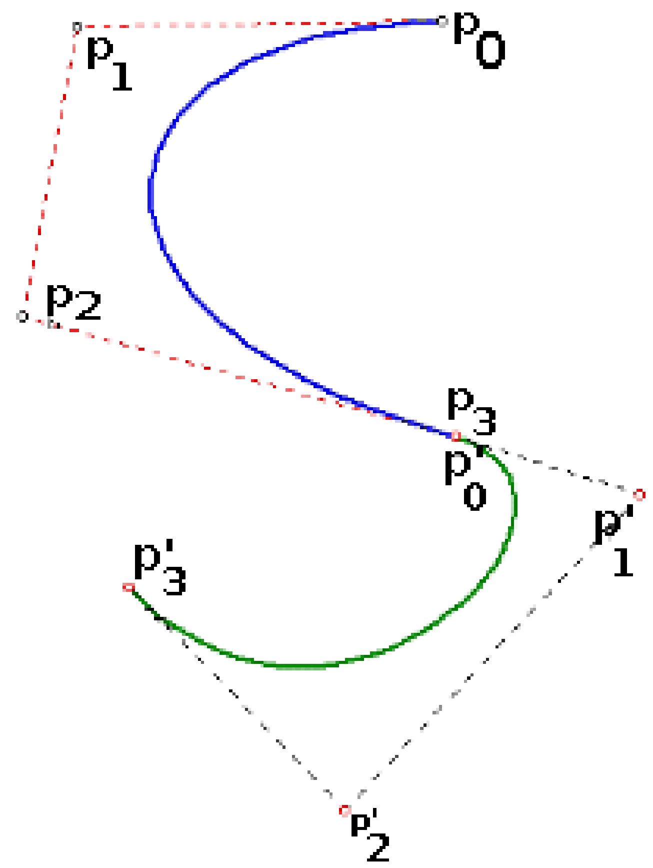
- *Convex hull*:
  - curve lies inside the convex hull of its control points
- *Endpoint interpolation:*

$$Q(0) = p_0$$
$$Q(1) = p_d$$

- *Tangents*

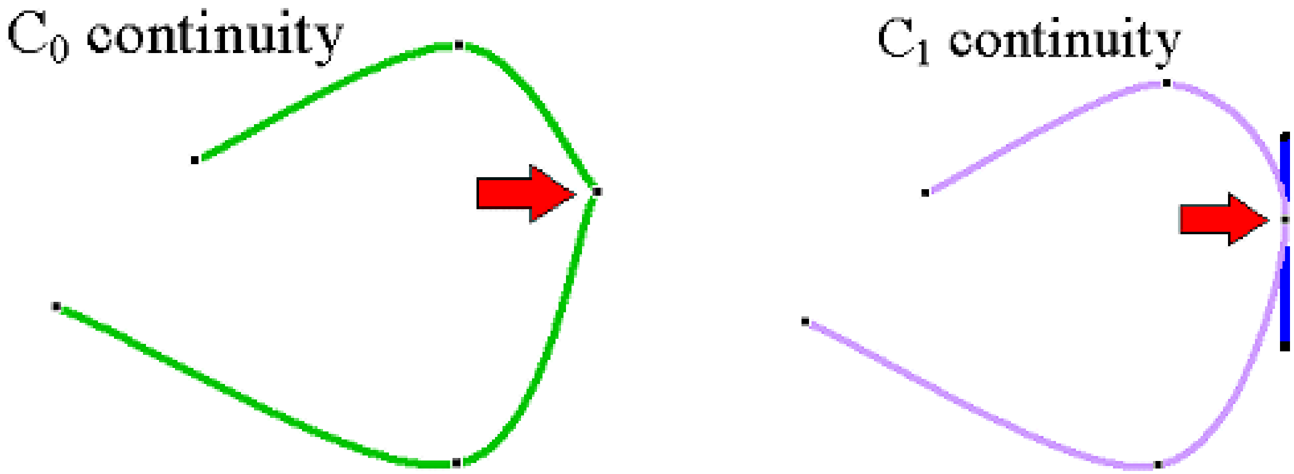
$$Q'(0) = d(p_1 - p_0)$$
$$Q'(1) = d(p_d - p_{d-1})$$

- *Symmetry*
  - $Q(u)$ defined by $p_0, \ldots, p_d$ is equal to $Q(1 - u)$ defined by $p_d, \ldots, p_0$



Smooth Bézier Curves

- *Smooth joint* between two Bézier curves of order d with control points $\{p_0, \ldots, p_d\}$, $\{p'_0, \ldots, p'_d\}$ respectively
  - C_0: same end-control-points at joints: $p_d = p'_0$ (due to end-point interpolation)
  - C_1: control points p_{d-1}, $p_d = p'_0$, p'_1 must be collinear (due to tangent property)



➡ Continuity conditions create restrictions on control points

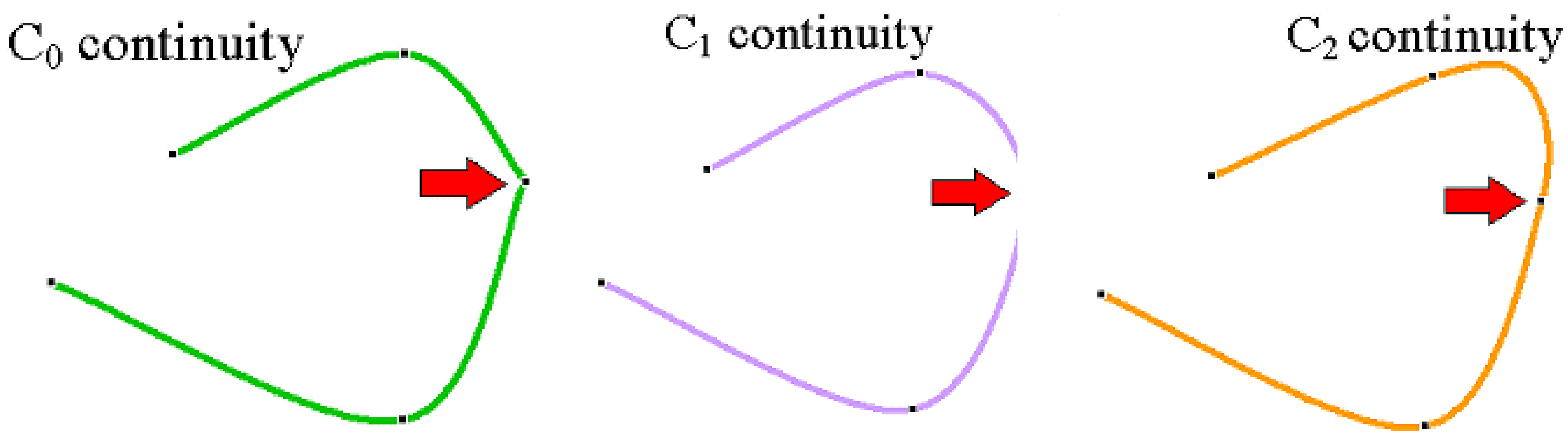
Parametric/Geometric Continuity

- Parametric continuity:
  - C^0: curves are joined
  - C^1: first derivatives are equal at the joint points
  - C^2: first *and* second derivatives are equal

    …
  - C^n: first *through* n^{th} derivatives are equal
- Geometric continuity:
  - G^0: The curves touch at the joint points
  - G^1: The curves also share a common tangent direction at the joint points (first derivatives are *proportional*)
  - G^2: The curves also share a common centre of curvature at the joint points (first and second derivatives are *proportional*)

Smoothness / Continuity

- Curve should be *smooth* to some order at joints
- Different types of *continuity at joints*
- *Geometric continuity*: from the geometric viewpoint
- *Parametric continuity*: for parametric curves



- Parametric continuity of order *n* implies geometric continuity of order n, but not vice versa.

Summary

- What is the implicit and explicit representation of a curve? What are the advantages and disadvantages of these representations?
- What are piecewise parametric polynomial curves (splines)? What is the advantage of this representation? What is the main problem?
- What are Bézier Curves and how are they defined? What properties do they have?
- What is the major problem when using piecewise polynomial curves? What conditions do the control points of a Bézier Curve have to fulfil in order to get C_0/C_1 continuous curves?

CMT107 Visual Computing

IX.2 Freeform Surfaces

Xianfang Sun

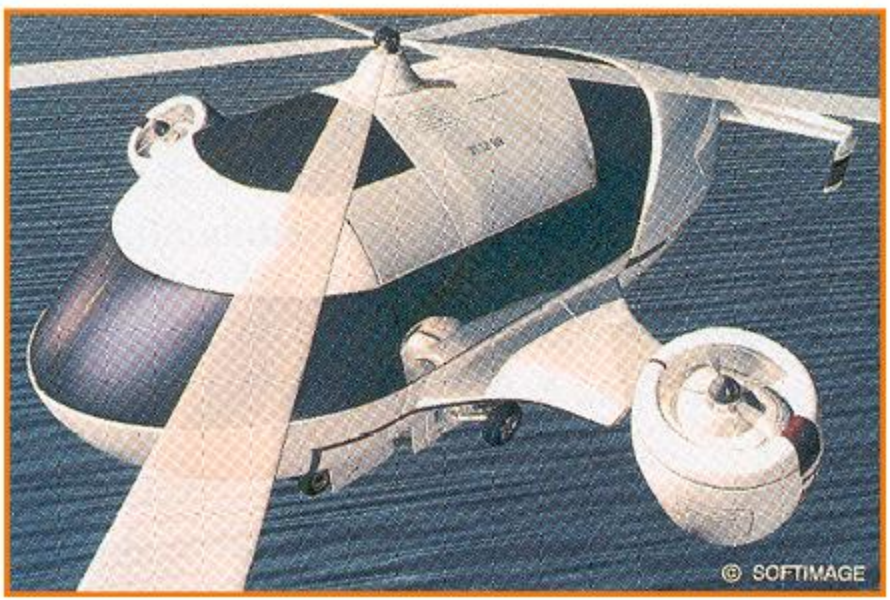
School of Computer Science & Informatics
Cardiff University

Overview

- Surface representations
- Parametric surfaces
- Piecewise polynomial surfaces
  - Tensor product splines
- Subdivision surfaces
  - Loop subdivision
  - Doo-Sabin subdivision
  - Catmull-Clark subdivision

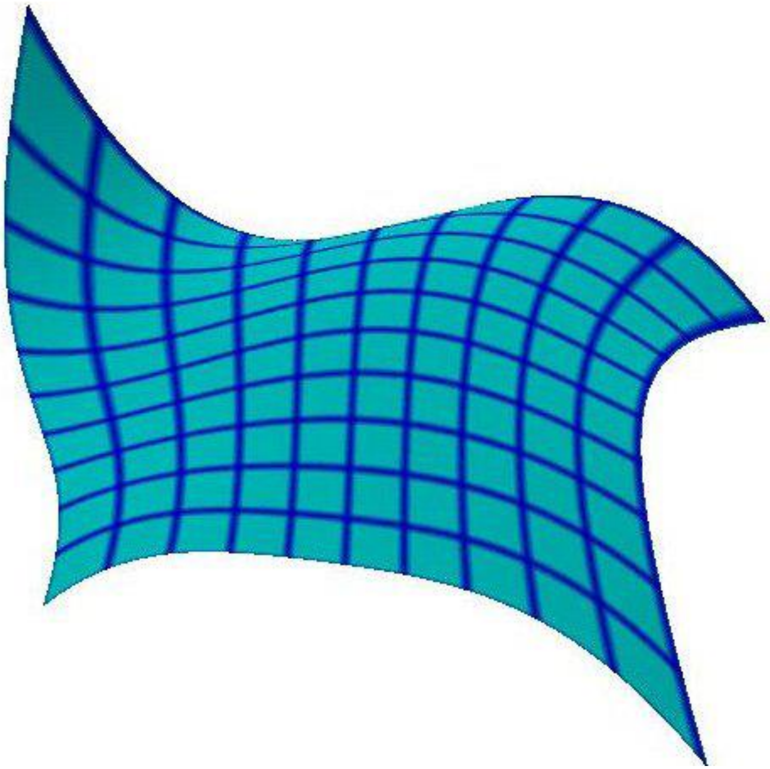
Surfaces

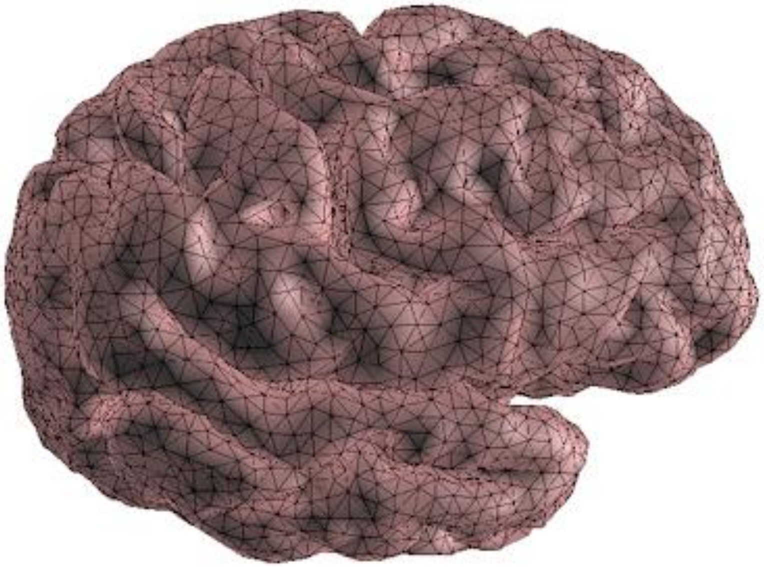
- We require general surface shapes (something better than polygonal meshes)
  - *Exact* boundary representation for some objects
  - *Create*, *edit* and *analyse* shapes



Explicit Surfaces

- A surface is a set of positions of a point moving with two degrees of freedom





- Explicit and implicit representation similar to curve
  - Explicit: $z = f(x, y)$ for $(x, y) \in R^2$

Implicit Surfaces

- Surface defined as solution of an equation system:

$$f(x, y, z) = 0$$

  - Usually one equation in 3D
- Example: linear equation (plane)

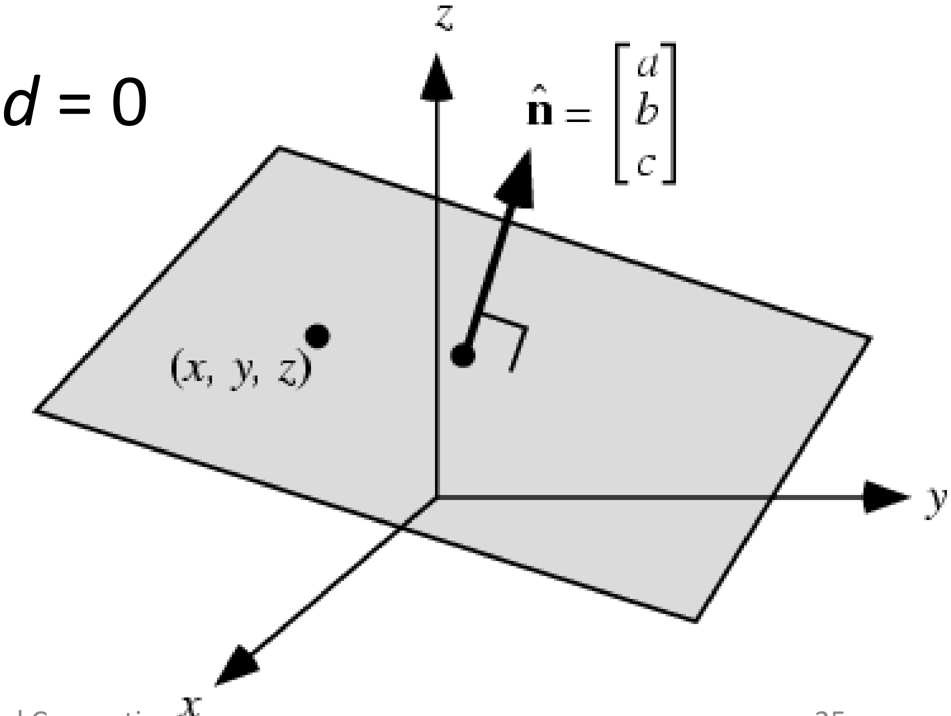
$$ax + by + cz + d = 0$$

  - Using vectors:

$$n^{\mathrm{T}}\mathbf{x} + d = 0$$

— n: unit plane normal

— d: distance from origin



Implicit Quadrics

- [Quadrics (quadratic surfaces)]

$$ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + gx + hy + jz + k = 0$$

- Matrix representation:

$$\boldsymbol{x}^{\mathrm{T}} M \boldsymbol{x} + v^{\mathrm{T}} \boldsymbol{x} + s = 0$$

- Sphere / Ellipsoid:

$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} + \frac{z^2}{r_z^2} - 1 = 0$$

- Cylinder (elliptic):

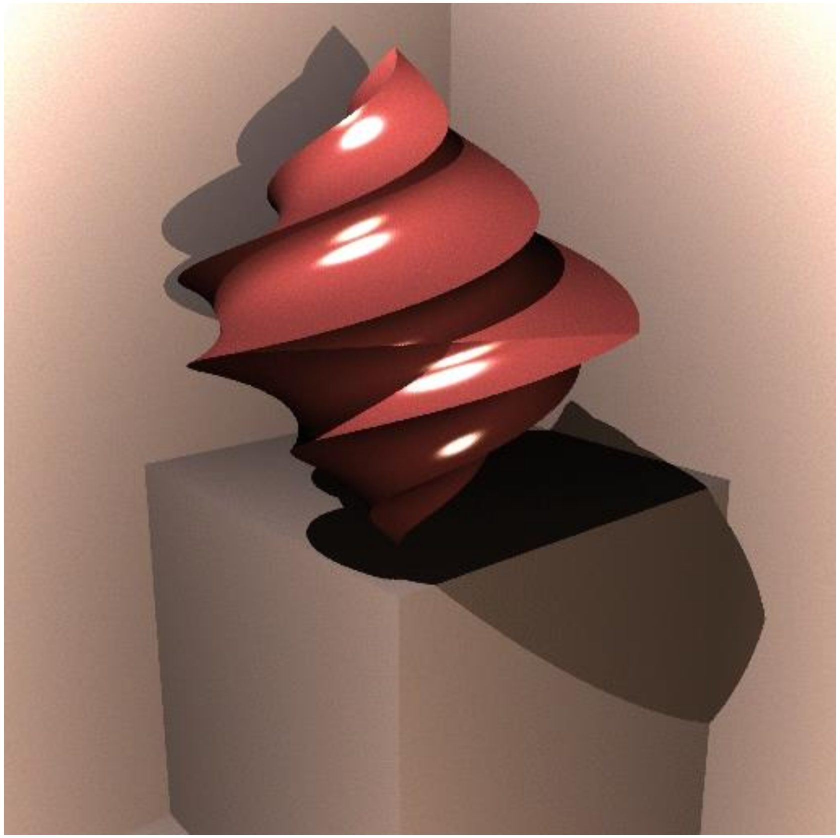
$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} - 1 = 0$$

- Cone (elliptic):

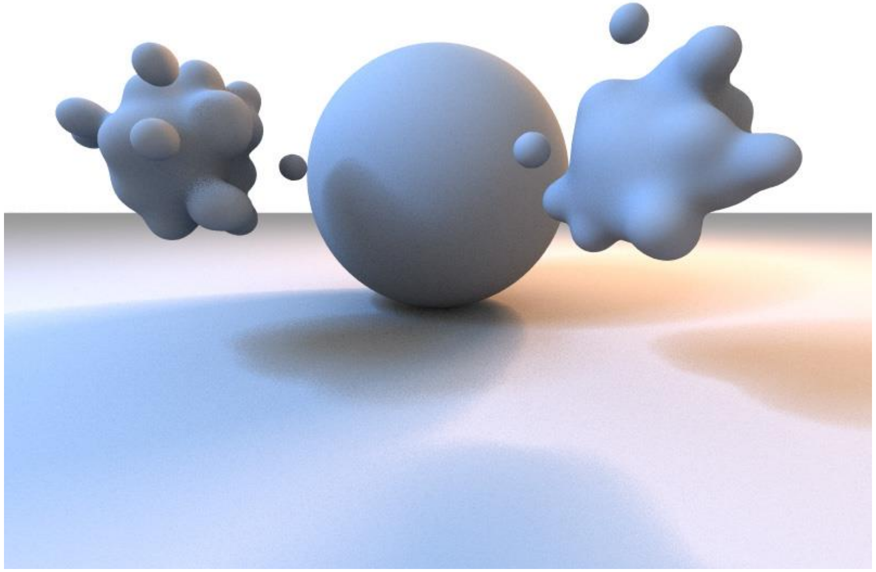
$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} - z^2 = 0$$

Properties of Implicit Surfaces

- Simple to test if point is on surface
- Simple to intersect two surfaces
- Hard to render
- Hard to describe complex shapes



Mathematical Functions / Sets



Blobby Models

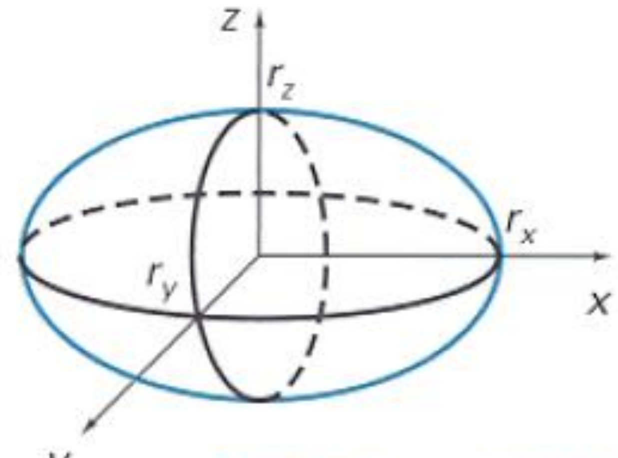
Parametric Surfaces

- Describe points on surface by *parametric functions*

$$\mathbf{s}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix}$$

  - Maps 2D (u,v) parameter domain to 3D (x,y,z) model space

- Example: ellipsoid
  - $x(u, v) = r_x \cos u \cos v$
    $y(u, v) = r_y \cos u \sin v$
    $z(u, v) = r_z \sin u$
    $(u, v) \in [-\pi/2, \pi/2] \times [0, 2\pi]$



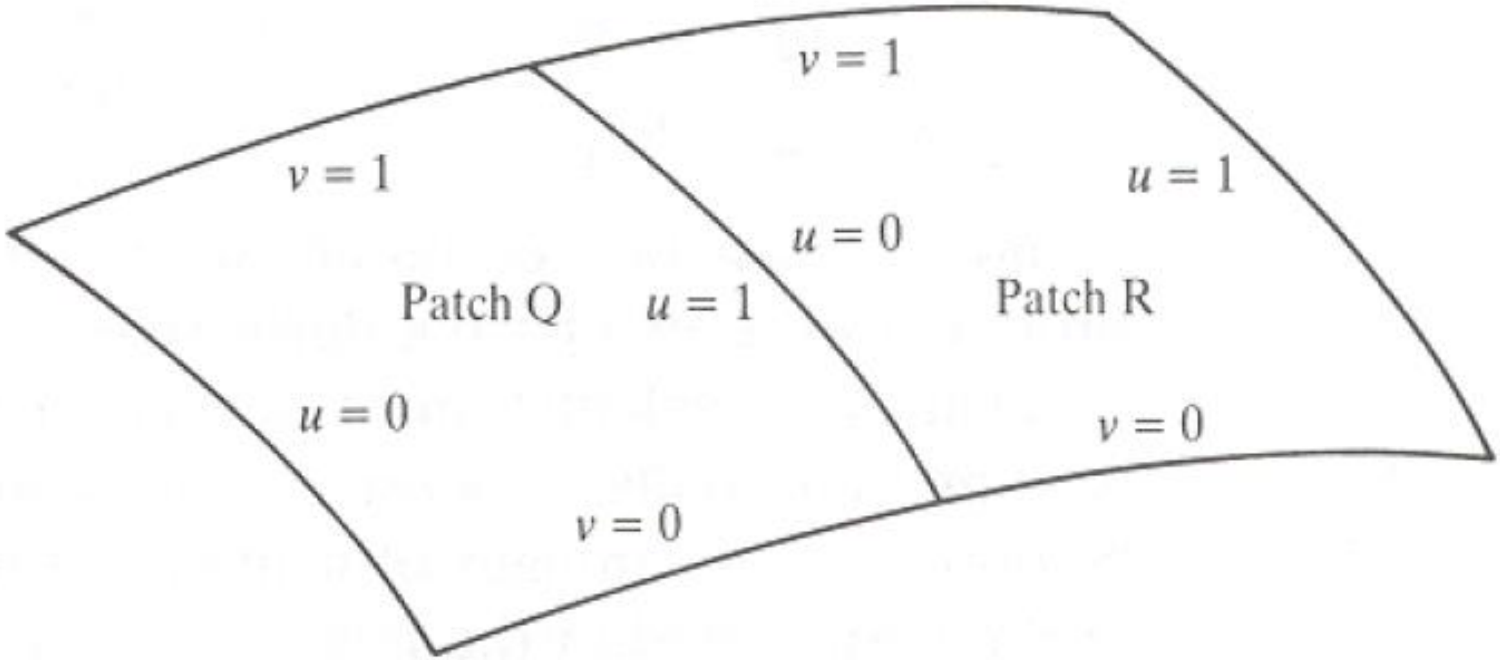
H&B Figure 10.10

Properties of Parametric Surfaces

- *Properties* similar to parametric curves
  - Simple to render points
  - Hard to test if point is on surface, compute intersections, etc.
- Hard to represent whole surface by single polynomial function
  - Use *piecewise polynomial surfaces*
  - Surface is cut into *patches*
  - *Smoothness* / *continuity* problem when joining patches

Piecewise Polynomial Surface

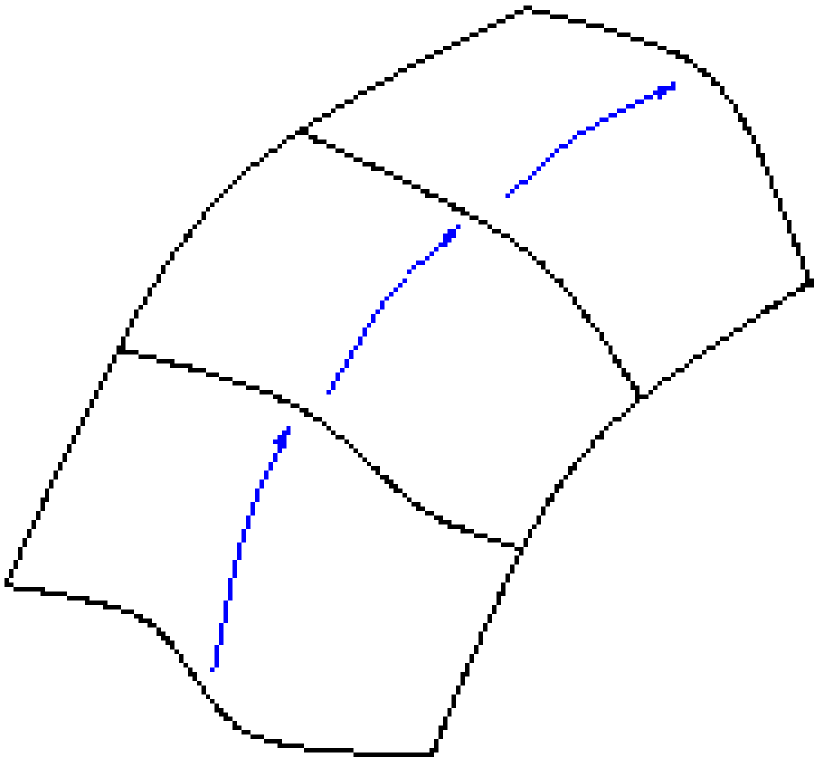
- *Spline surface*: piecewise polynomial surface patches



- Use spline curve approach with two degrees of freedom
  - Each patch is defined by a set of *control points*

Tensor Product

- Intuitively, a surface is a curve which *moves* through space while it changes its shape



- Mathematically this is the *tensor product* of two curves

Tensor Product Surfaces

- Surface patch as a curve moving through space
  - Assume this curve is at any time $v \in [0, 1]$ a Bézier curve

$$c^v(u) = \sum_{l=0}^{\alpha} P_l(v) B_l^{\alpha}(u)$$

  - The control points $P_l(v)$ lie on curves as well, assume these are also Bézier curves

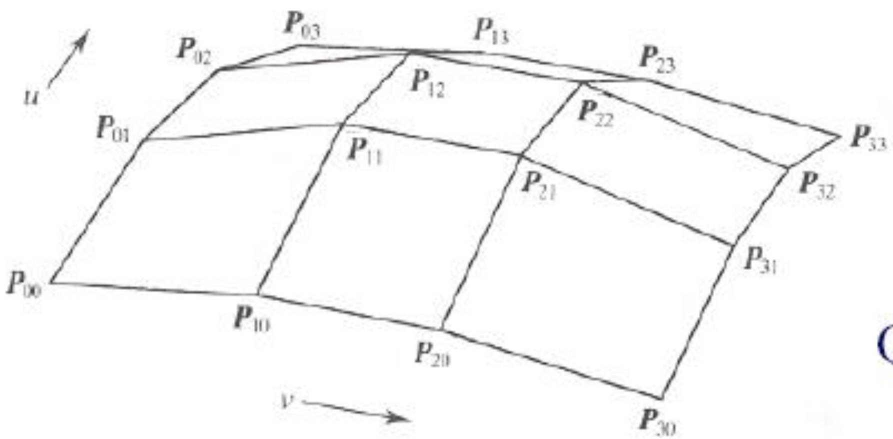
$$P_l(v) = \sum_{k=0}^{\beta} P_{l,k} B_k^{\beta}(v)$$

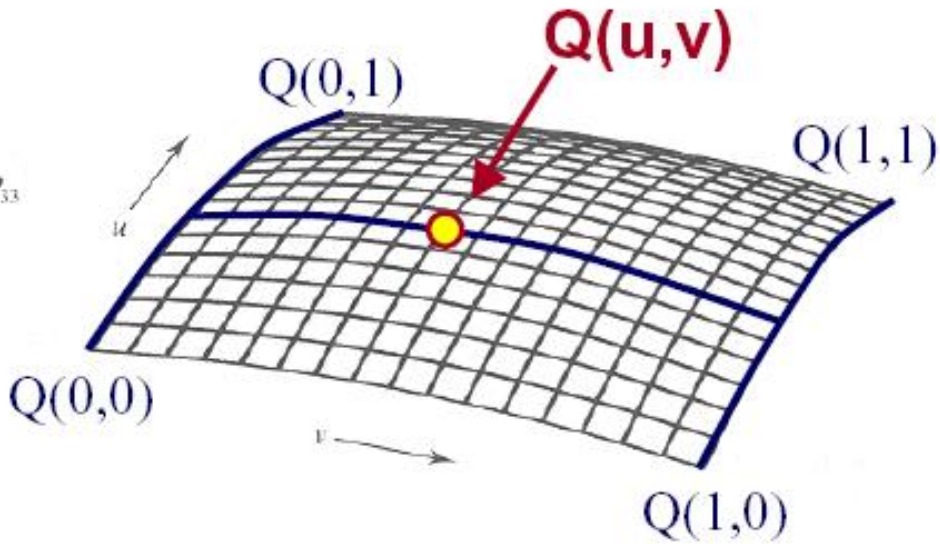
- Combing both gives the formula for a *Bézier surface patch*

$$Q(u, v) = \sum_{l=0}^{\alpha} \sum_{k=0}^{\beta} P_{l,k} B_l^{\alpha}(u) B_k^{\beta}(v)$$

Bézier Surface Patches

- Point $Q(u, v)$ on the patch is the tensor product of Bézier curves defined by the *control points* $P_{l,k}$

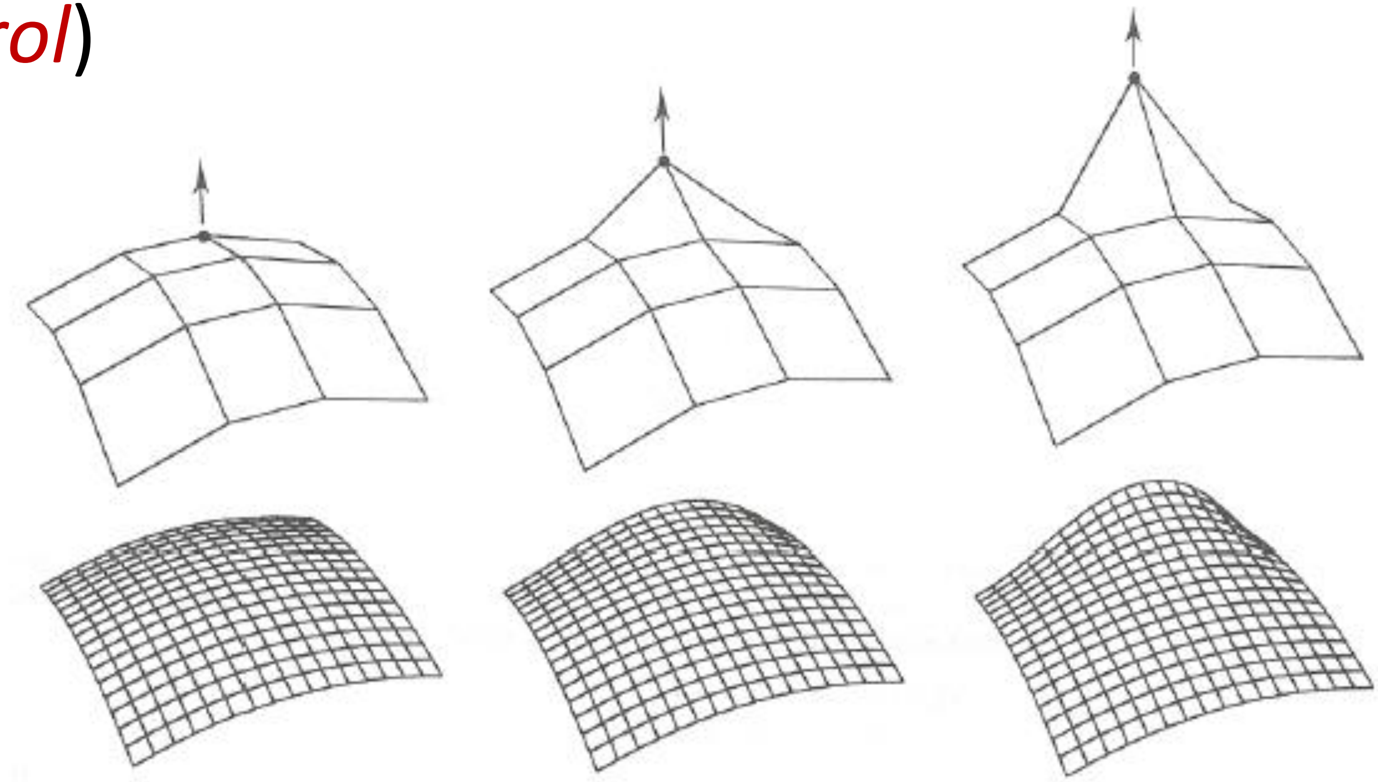




- *Order* of surface is given by order of curves α, β (e.g. bicubic: $\alpha = \beta = 3$)

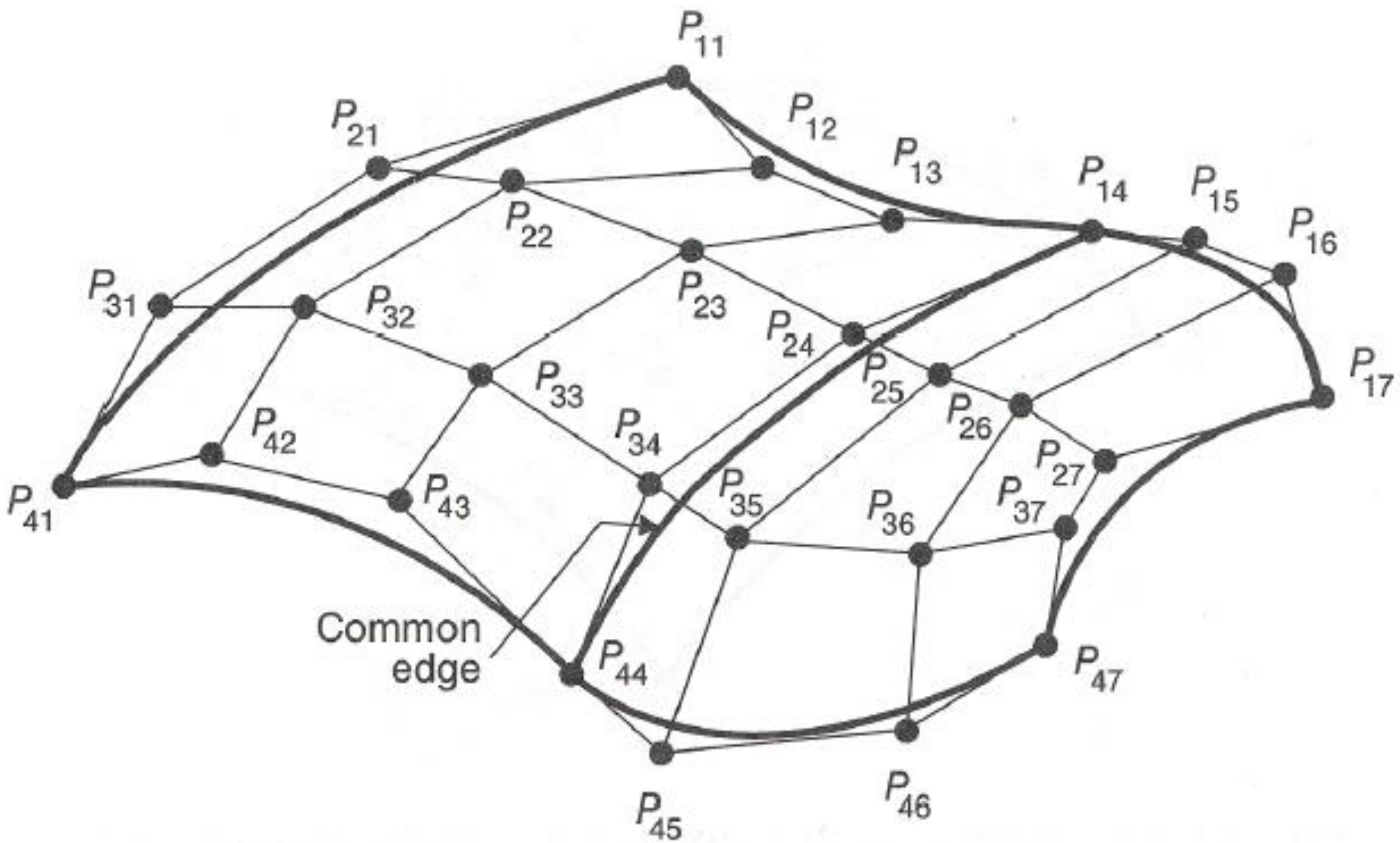
Properties of Bézier Patches

- *Interpolates* four corner control points
- Lies inside *convex hull* of control points
- Changing control points has only local effect (*local control*)



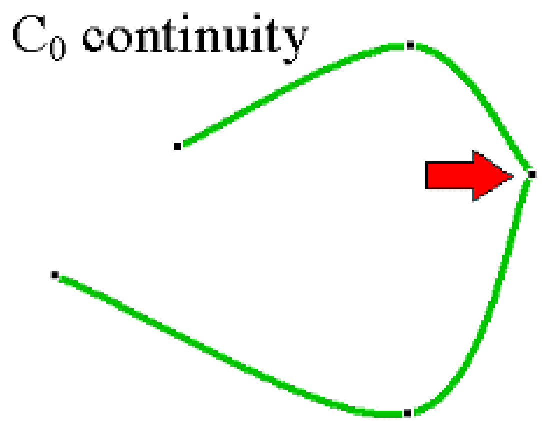
Smooth Bézier Surfaces

- *Continuity* / *smoothness* constraints similar to Bézier splines

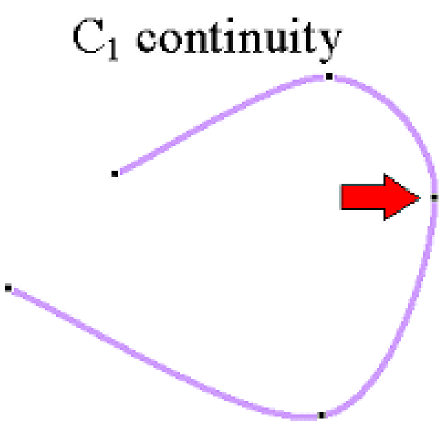


C^0 and C^1 Bézier Surfaces

- C^0 requires *aligning boundary curves*



- C^1 requires aligning boundary curves and derivatives

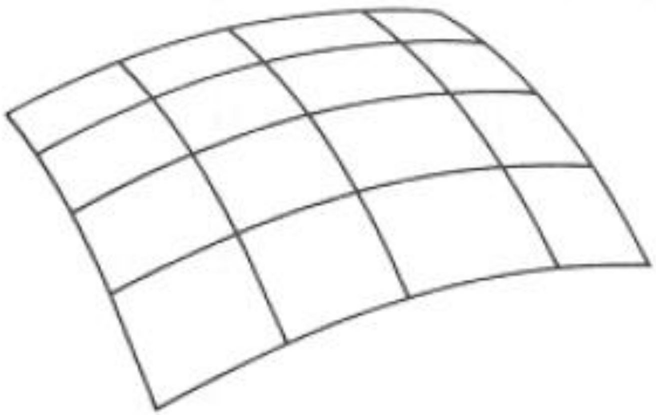


Drawing Bézier Surfaces

- Simple approach:

  loop through *uniformly* spaced increments of *u* and *v*

$$
\begin{array}{l}
\text{for (int } l = 0;\ l < l_{\max};\ ++l) \ \{ \\
\quad \text{double } u = u_{\min} + l * u_{\text{step}}; \\
\quad \text{for (int } k = 0;\ k < k_{\max};\ ++k) \ \{ \\
\quad\quad \text{double } v = v_{\min} + k * v_{\text{step}}; \\
\quad\quad \text{DrawQuadrilateral (...);} \\
\quad \} \\
\}
\end{array}
$$

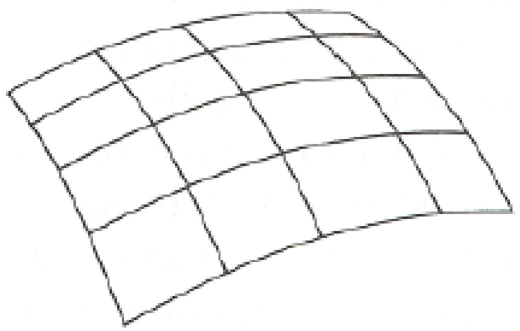


- Note, Bézier surfaces always have *quadrilateral* structure

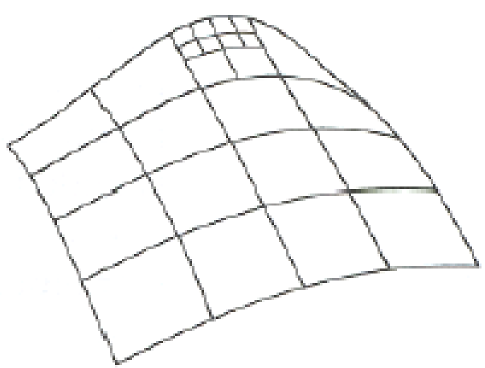
Drawing Bézier Surfaces

- Better approach:
  use *adaptive subdivision*

```
DrawSurface (surface) {
  if flat(surface, epsilon) {
    DrawQuadrilateral (surface);
  } else {
    SubdivideSurface (surface,...);
    DrawSurface (surfaceLL);
    DrawSurface (surfaceLR);
    DrawSurface (surfaceRL);
    DrawSurface (surfaceRR);
  }
}
```



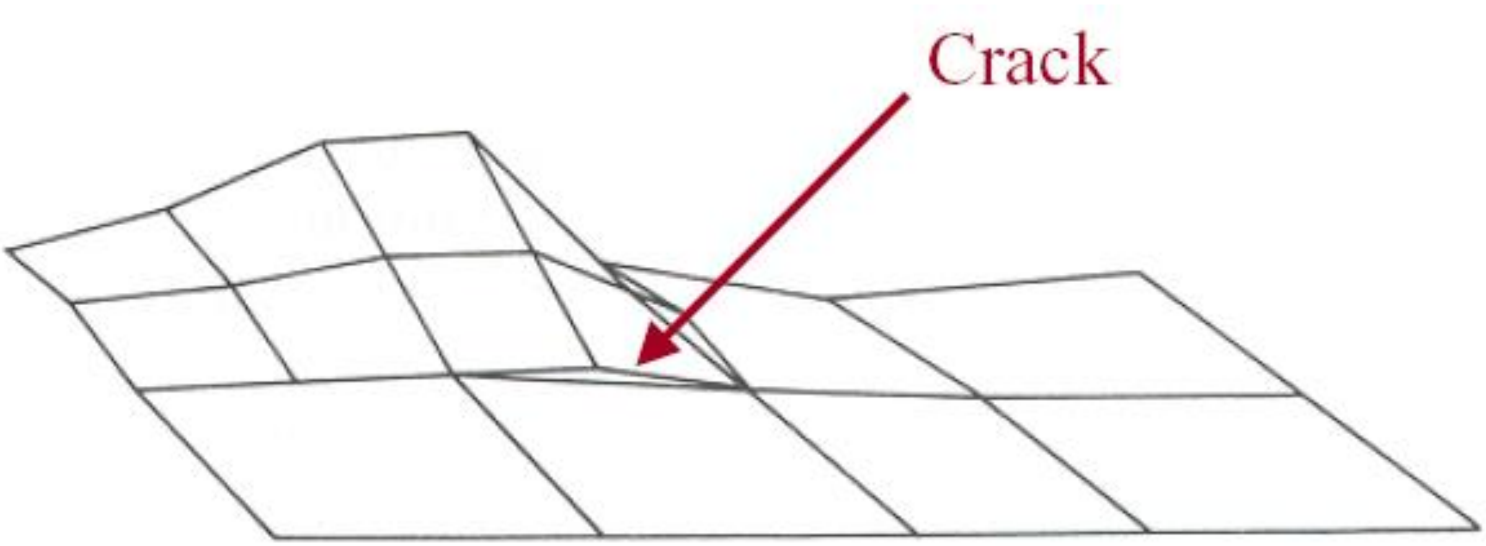
Uniform subdivision



Adaptive subdivision

Drawing Bézier Surfaces

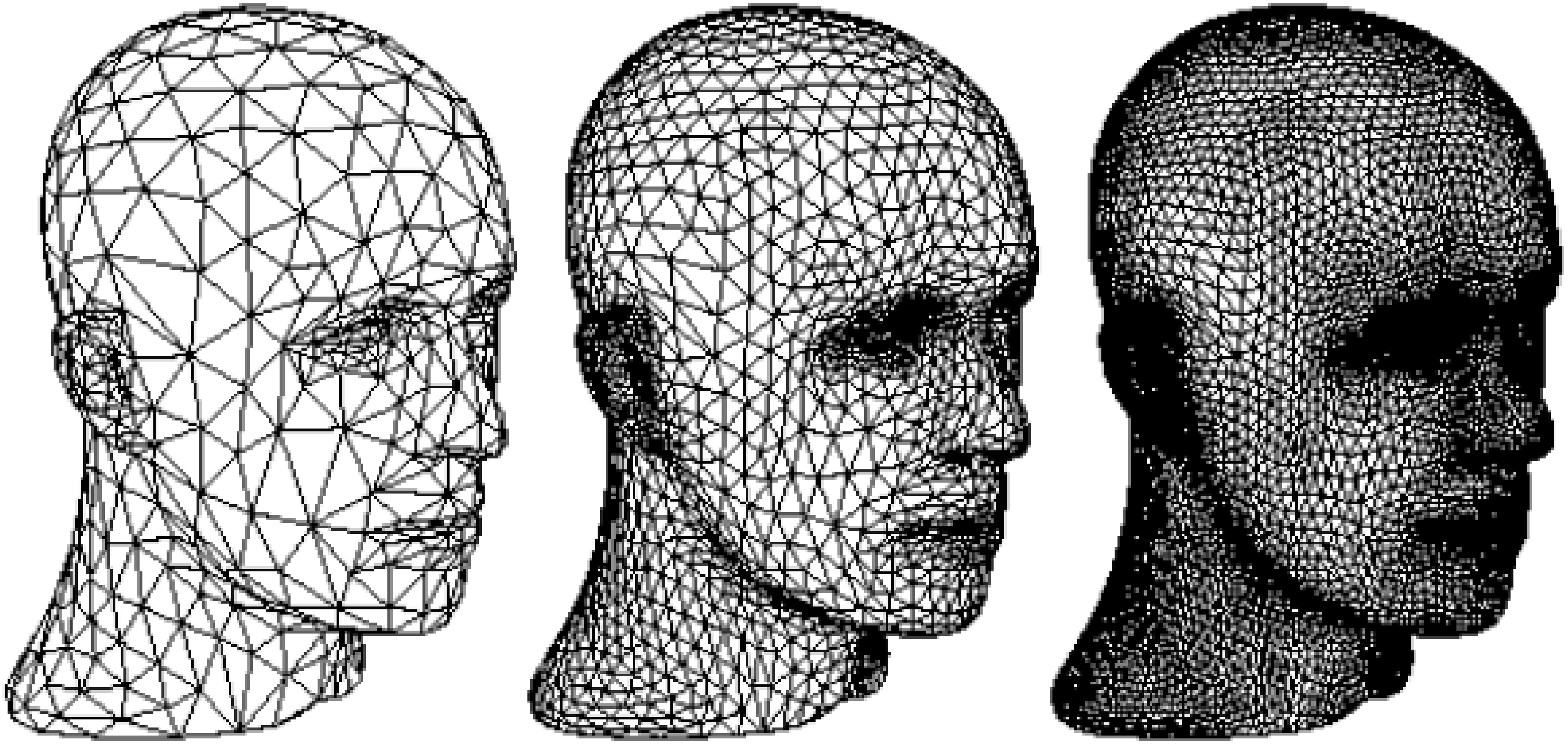
- Problem of adaptive subdivision:
  - *Cracks* at boundaries between patches at different subdivision levels



  - Avoid cracks by *adding extra vertices* and *triangulating* quadrilaterals with neighbours at finer level

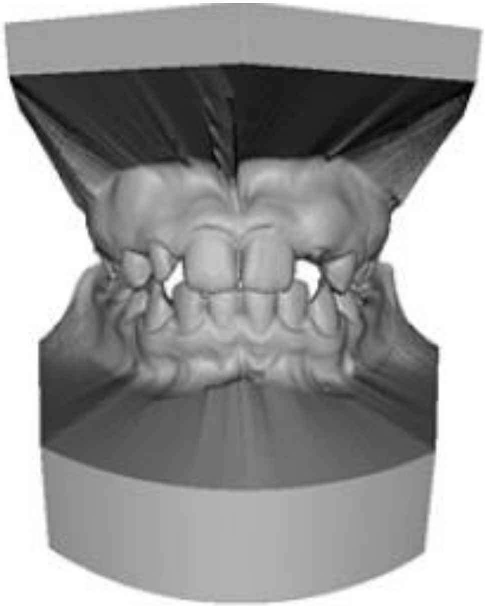
Subdivision Surfaces

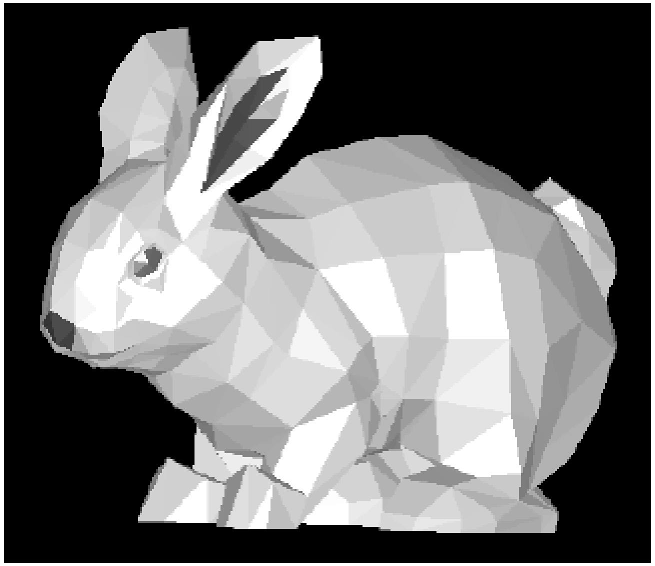
- Idea of *subdivision surfaces*
  - Define a smooth surface as the limit of a sequence of successive refinements of a mesh

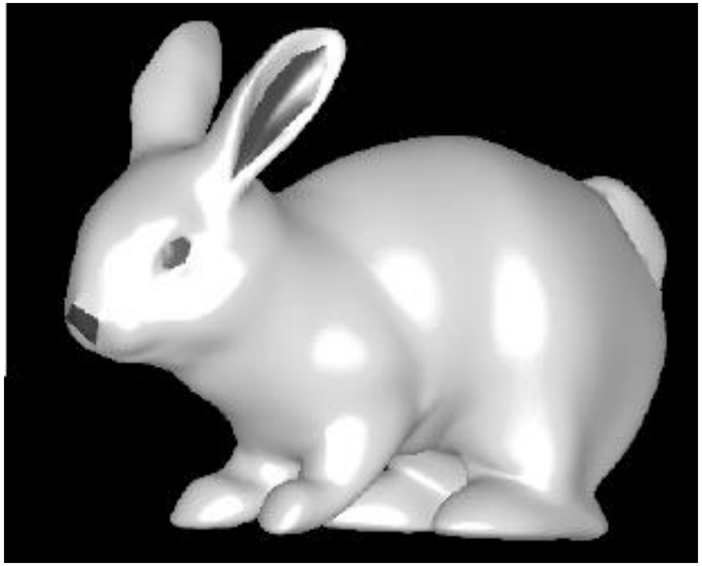


Why Subdivision?

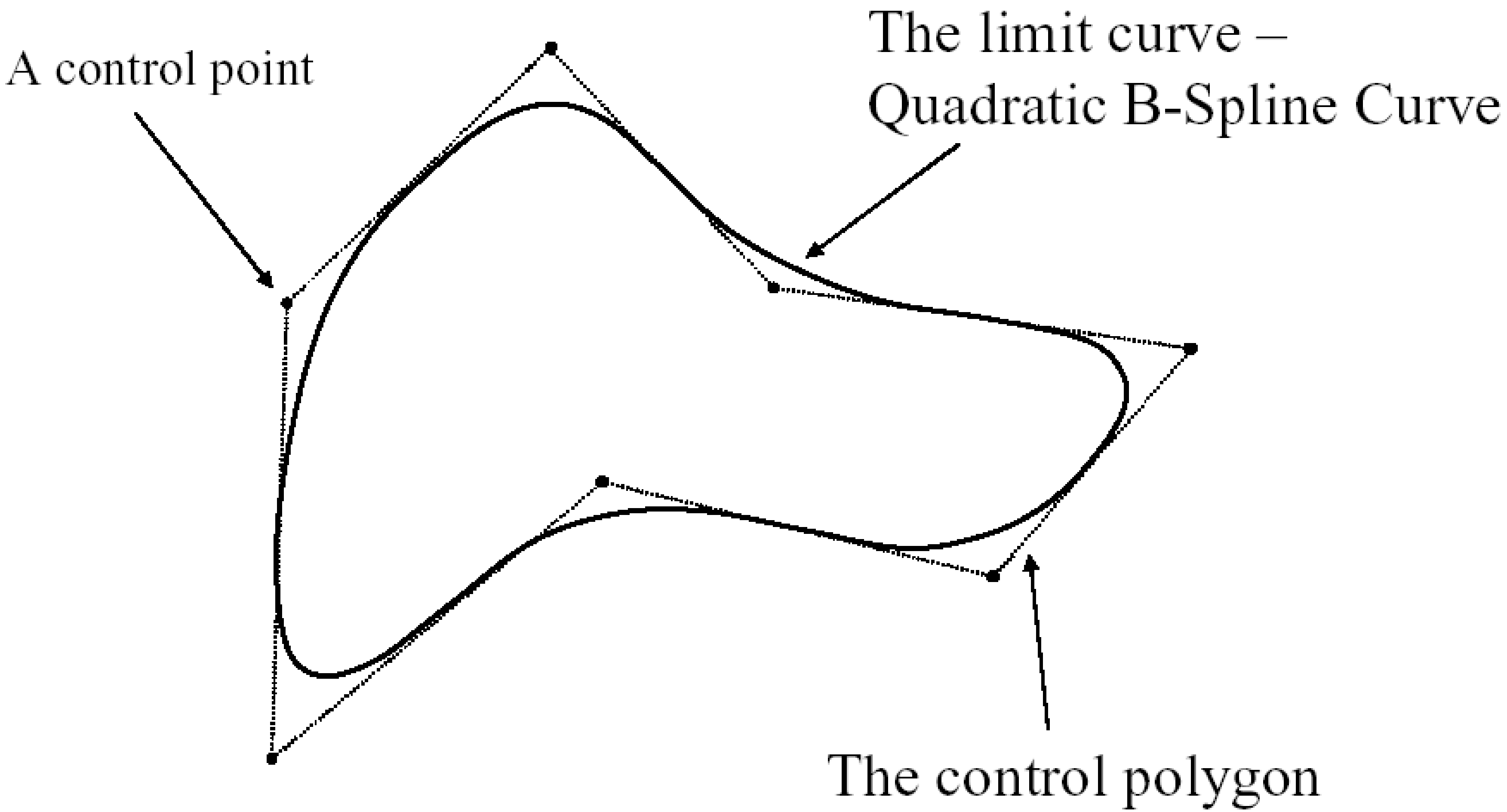
- Level of Detail
- Compression
- Smoothing







Cutting Corners – Curves



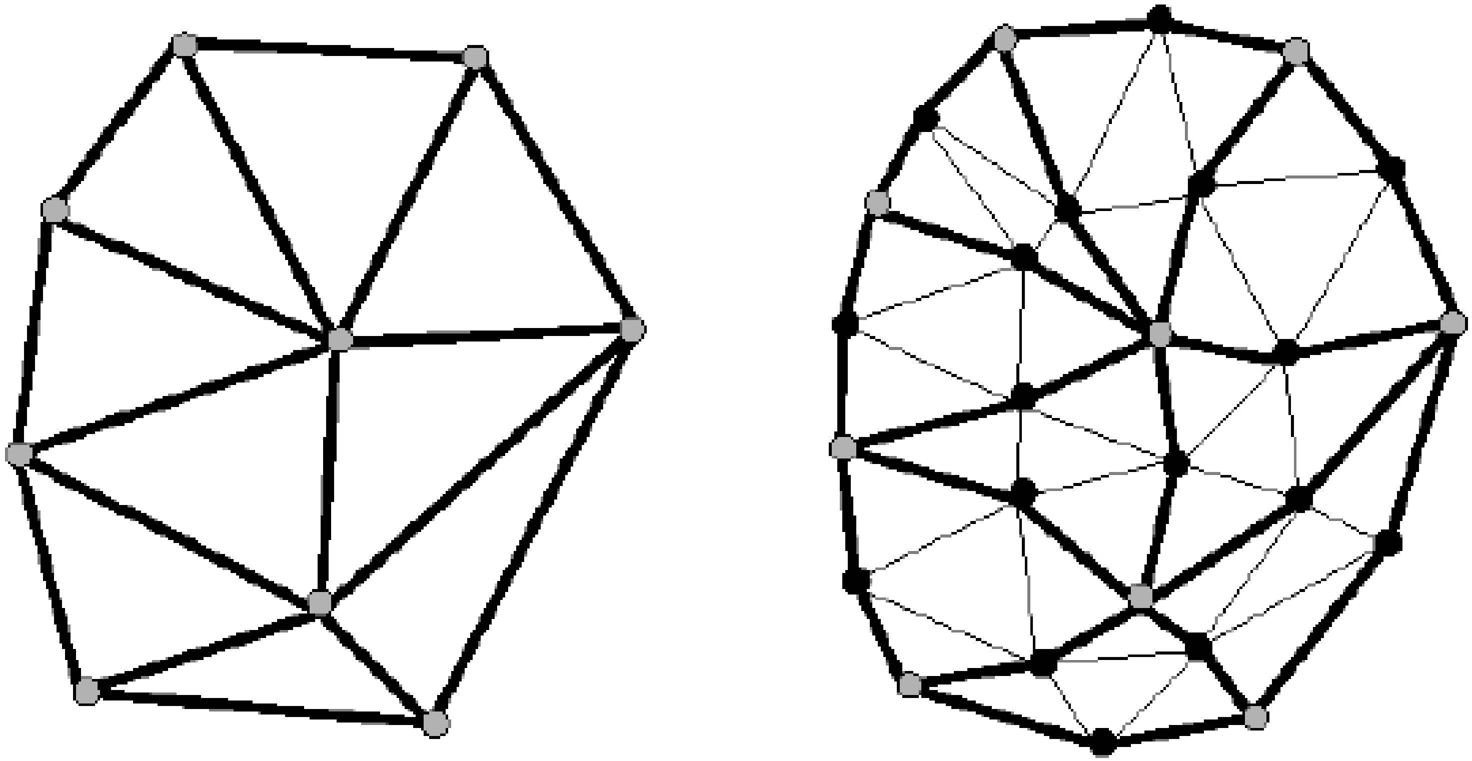
Center for Graphics and Geometric Computing, Technion

Surface Subdivision

- Start with a *control mesh*
- Per iteration construct *refined* mesh by inserting vertices
- Mesh sequence should converge to a *limit surface*
- Subdivision scheme defined by two elements
  - Generate *topology* of the new mesh
  - Compute vertex locations in new mesh
    - *Vertex point*: new location of old vertex
    - *Edge point*: location of new vertex on old edge
    - *Face point*: location of new vertex on old face

Loop Subdivision

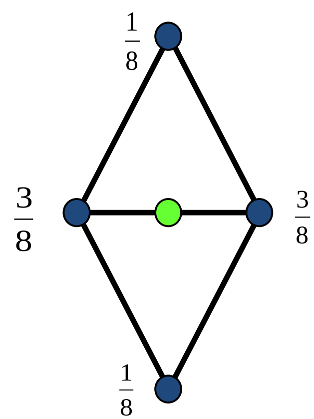
- Loop subdivision scheme:
  - Refine each triangle into 4 triangles by splitting each edge and connecting new vertices



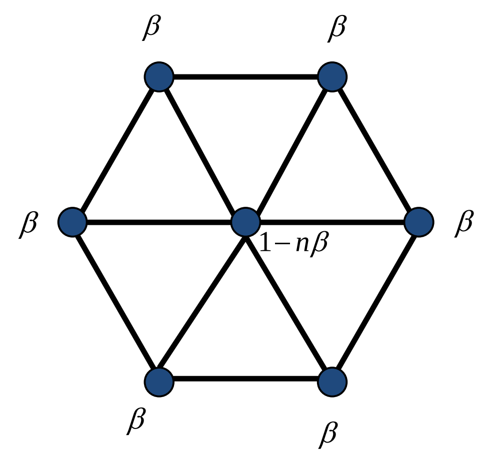
Loop Subdivision

- Computing locations of new vertices
  - *Weighted average* of original vertices in neighbourhood

Edge point



Vertex point



$$\beta = \frac{1}{n}\left(\frac{5}{8} - \left(\frac{3}{8} + \frac{2}{8}\cos\left(\frac{2\pi}{n}\right)\right)^2\right)$$

- No face points

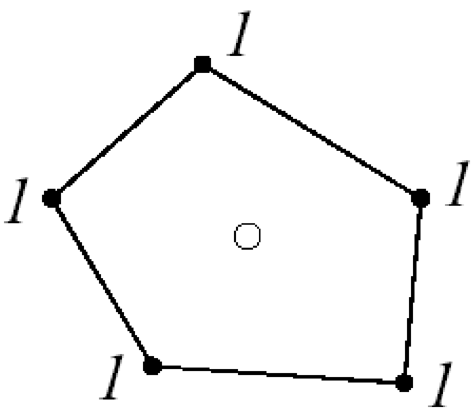
Catmull-Clark Subdivision

- Mesh is the control mesh of a *B-Spline surface*
  - Refined mesh is also a control mesh of a B-Spline Surface
- Incremental construction
  - Calculate face points
  - Calculate edge points using face points
  - Calculate vertex points using face and edge points
  - Connect vertices

Catmull-Clark Subdivision

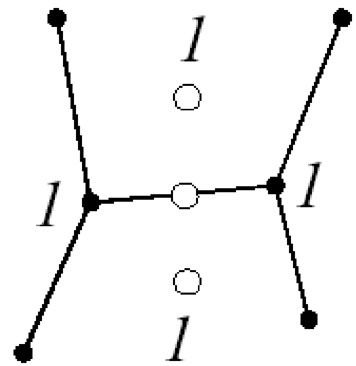
Step 1

First, all the face points are calculated



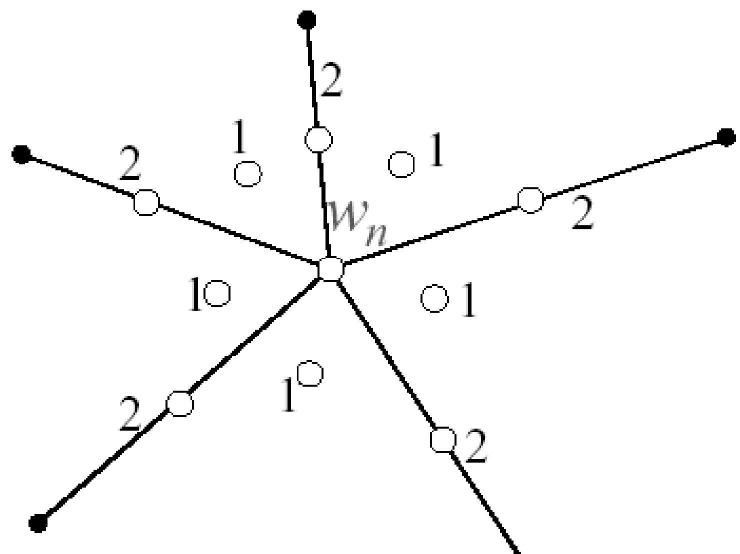
Step 2

Then the edge points are calculated using the values of the face points and the original vertices



Step 3

Last, the vertex points are calculated using the values of the face and edge points and the original vertex

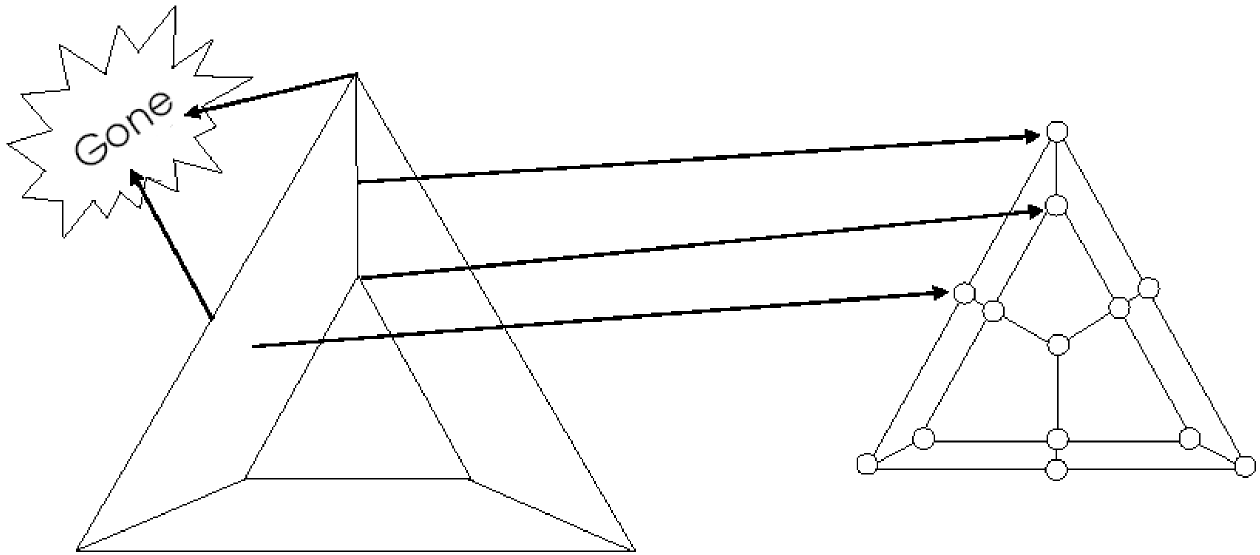


n - the vertex valence

Center for Graphics and Geometric Computing, Technion

Catmull-Clark Subdivision

- Connecting new vertices:
  - Connect each new face point to edge points of the edges defining the old face
  - Connect each new vertex point to new edge points of all old edges incident on the old vertex point



Center for Graphics and Geometric Computing, Technion

Catmull-Clark Subdivision

- Face Point

$$f = \frac{1}{m}\sum_{i=1}^{m} p_i$$

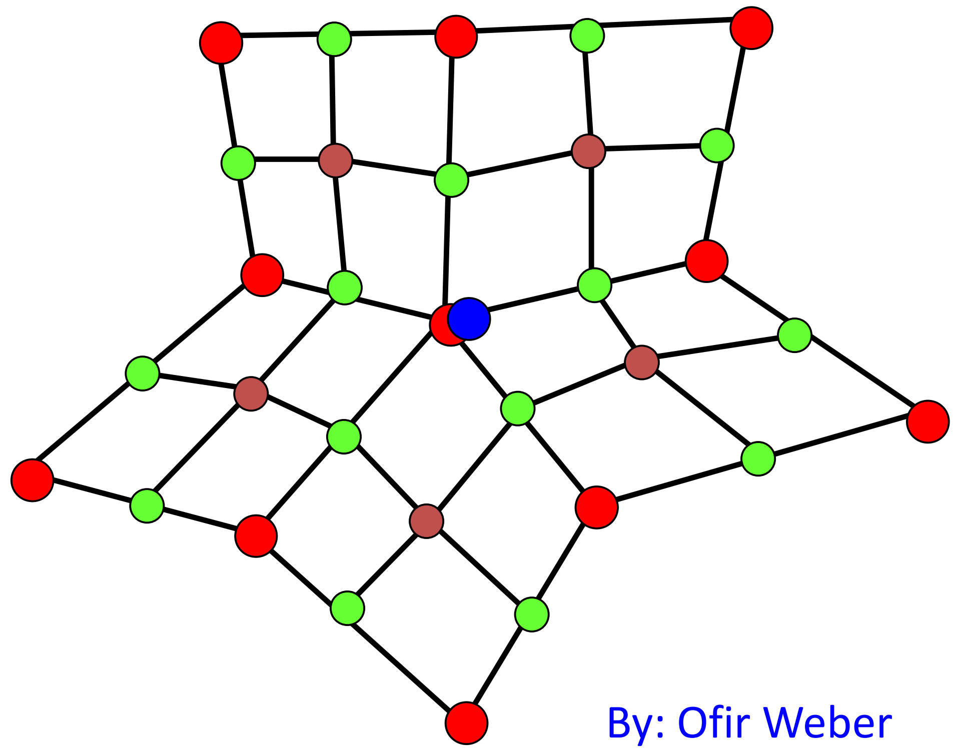
- Edge Point

$$e = \frac{p_1 + p_2 + f_1 + f_2}{4}$$

- Vertex Point

$$v = \frac{Q}{n} + \frac{2R}{n} + \frac{p\,(n-3)}{n}$$

$$\boldsymbol{v} = \frac{1}{\boldsymbol{n}^2}\sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{f}_{\boldsymbol{i}} + \frac{1}{\boldsymbol{n}^2}\sum_{\boldsymbol{i}=1}^{\boldsymbol{n}} \boldsymbol{p}_{\boldsymbol{i}} + \frac{\boldsymbol{n}-2}{\boldsymbol{n}}\boldsymbol{p}$$

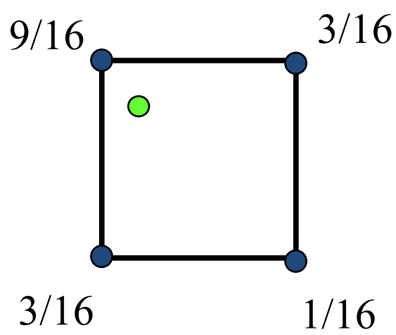


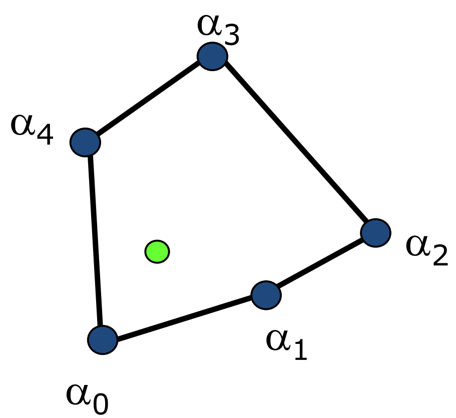
By: Ofir Weber

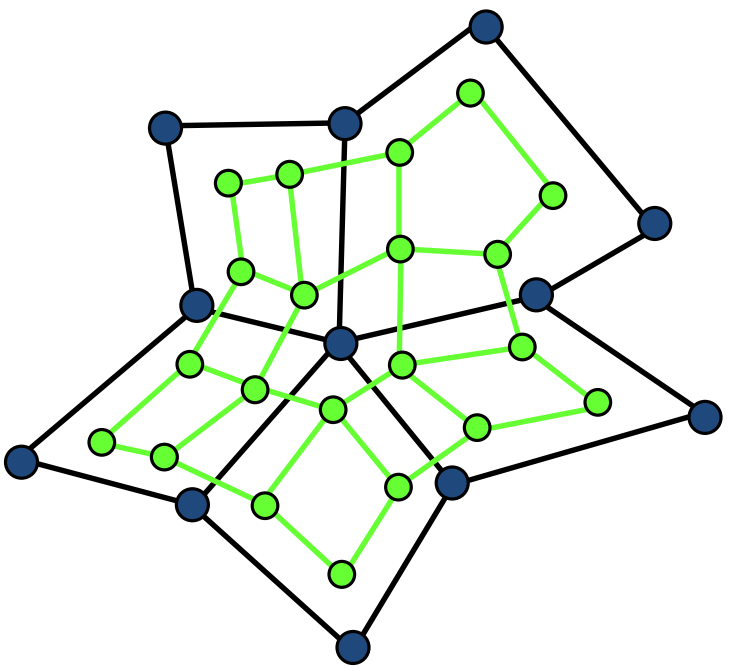
- *Q* – Average of face points
- *R* – Average of midpoints
- *p* – old vertex

Doo-Sabin Subdivision

Masks:





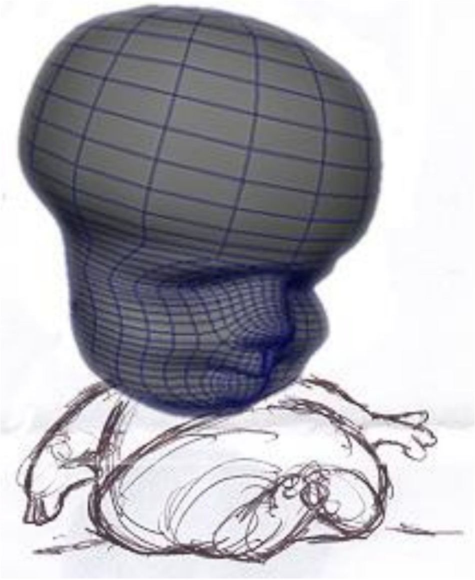


$$\alpha_i = \frac{\delta_{i,0}}{4} + \frac{3 + 2\cos\left(2i\pi / n\right)}{4n}$$

$$p = \sum_{i=0}^{n-1} \alpha_i p_i$$

Properties of Subdivision Surfaces

- *Advantages*
  - Simple methods for describing complex surfaces
  - Multi-resolution evaluation and manipulation
  - Arbitrary topology of control mesh (not only quadrilateral)
  - Limit surface is smooth
- *Disadvantages*
  - No obvious parametrisation
  - Hard to find intersections



Summary

- What are parametric surfaces? What are their advantages and disadvantages?
- What are spline surfaces? What are their advantages and disadvantages? What is the major problem when defining surfaces "piecewise"?
- What is the principle of a tensor product surface? What are Bézier surfaces? What conditions do the control points of C^0/C^1 continuous Bézier surfaces have to fulfil?
- What is the principle of subdivision surfaces? What are their advantages / disadvantages?
- How do Loop, Catmull-Clark, Doo-Sabin subdivision schemes work?