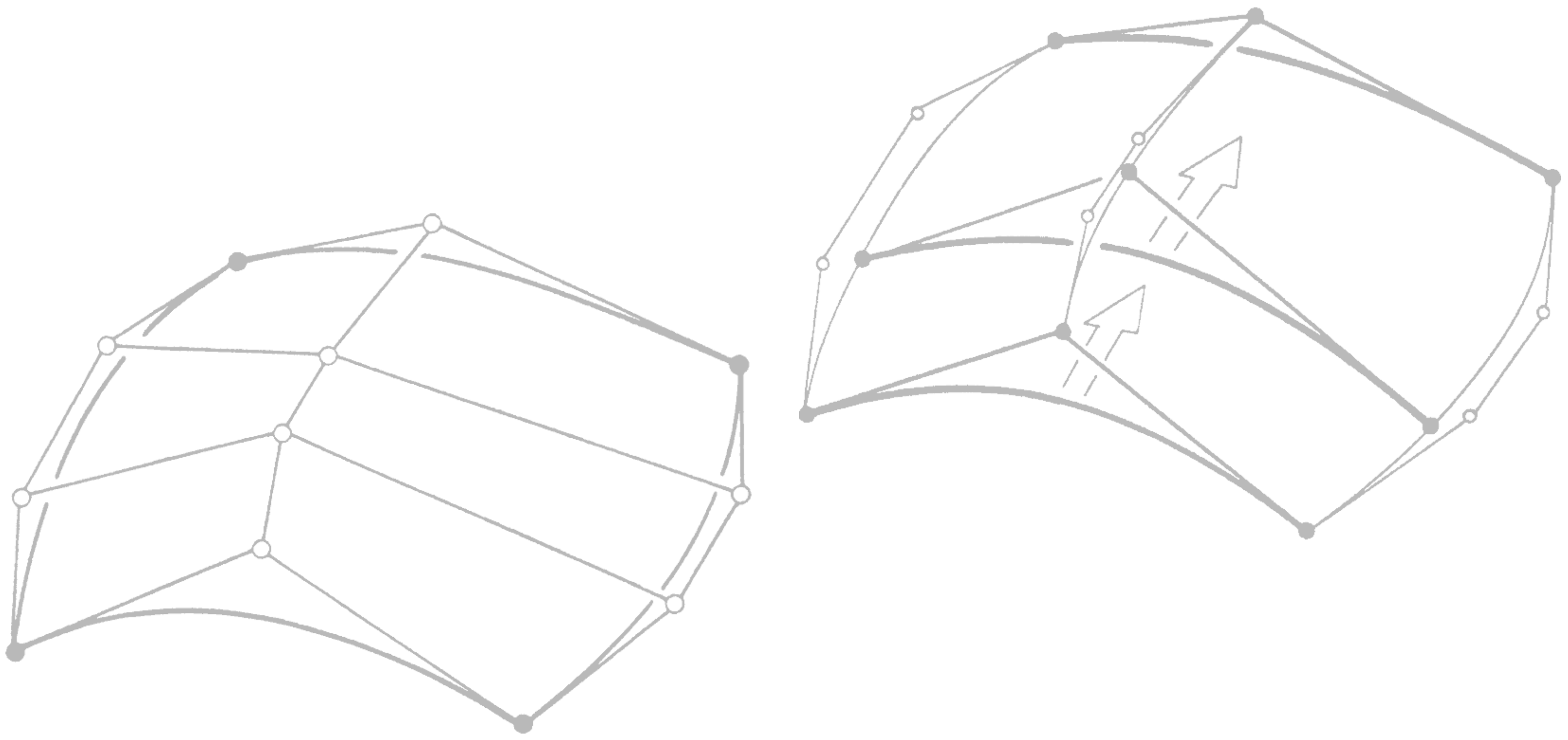
Tensor Product Surfaces

Prof. Dr. Markus Gross



ETH zürich

cgl

Overview

- Tensor Product Approach
- Surface Construction
- Bézier Patch
- 2D de Casteljau
- B-Spline Patch
- Derivatives

The Tensor Product Approach

- Let $\mathbf{x}(u) = \sum_{i=0}^{m} \mathbf{c}_i F_i(u)$ be a 2D or 3D spatial curve given by the bases F_i
- Coefficients $\mathbf{c}_i$ are functions of a second parameter v
- $\mathbf{c}_i$ -curves as linear combinations of G_j

$$\mathbf{c}_i(v) = \sum_{j=0}^{v} \boldsymbol{\alpha}_{i,j} G_j(v)$$

- $\Rightarrow$ so-called ***tensor product surface x(u,v)***:

$$\mathbf{x}(u,v) = \sum_i \mathbf{c}_i(v) F_i(u) = \sum_{i=0}^{m} \sum_{j=0}^{n} \boldsymbol{\alpha}_{i,j} F_i(u) G_j(v)$$

“Tensor Product”

The name “tensor product” is derived from the tensor product or outer product operator by which the 2D separable basis functions can be constructed.

We assume the function space V_1 to be spanned by $B_i(u)$.

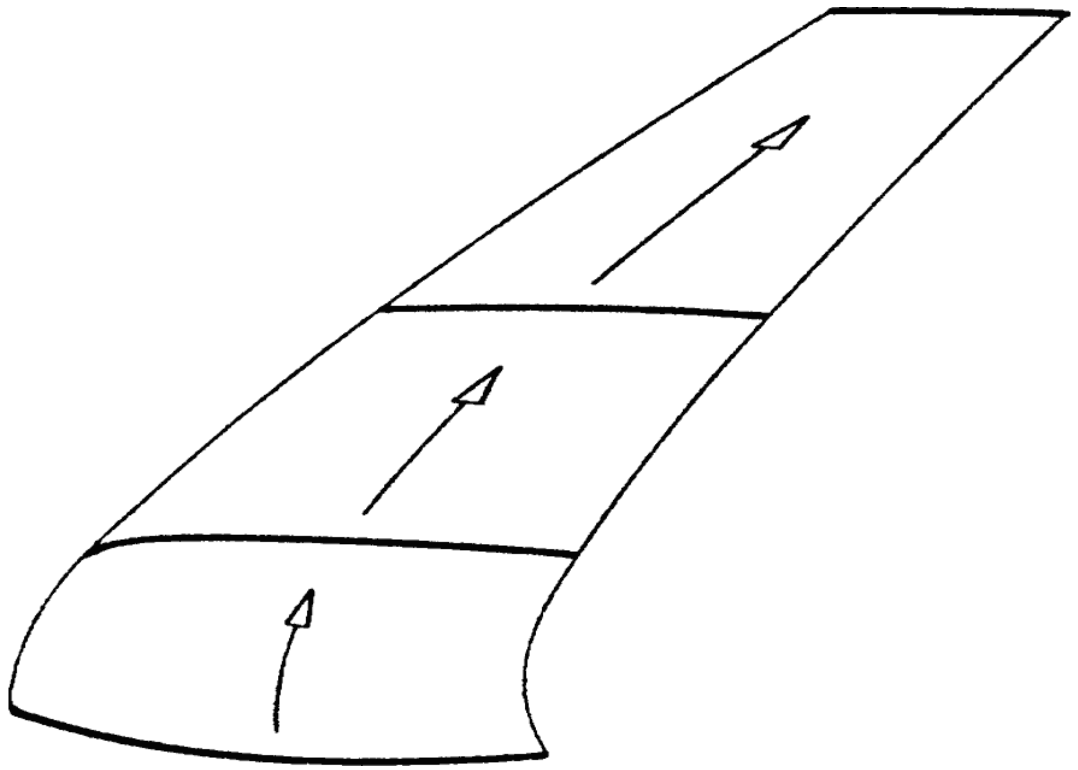
A 2D basis $B_i(u) \bullet B_j(v)$ can be constructed by

$$V_2 = V_1 \otimes V_1 \quad \text{with}$$

$$B_{i,j}^m(u,v) = B_i^m(u) \bullet B_j^m(v) \qquad i,j = 0,..,m$$

Tensor Product Surface

(Trace of a Curve in Space)



Bézier Patches

- Given a Bézier-curve $\boldsymbol{b}^m(u)$ of degree m with:

$$\boldsymbol{b}^{\boldsymbol{m}}(u)=\sum_{i=0}^{m}\boldsymbol{b}_{\boldsymbol{i}}B_i^m(u)$$

- Control points $\boldsymbol{b}_i$ as Bézier-curves of degree n:

$$\boldsymbol{b}_{\boldsymbol{i}}=\boldsymbol{b}_{\boldsymbol{i}}(v)=\sum_{j=0}^{n}\boldsymbol{b}_{\boldsymbol{i,j}}B_j^n(v)$$

- Point $\boldsymbol{b}_{m,n}(u,v)$ on the surface with:

$$\boldsymbol{b}^{\boldsymbol{m,n}}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}\boldsymbol{b}_{\boldsymbol{i,j}}B_i^m(u)B_j^n(v)$$

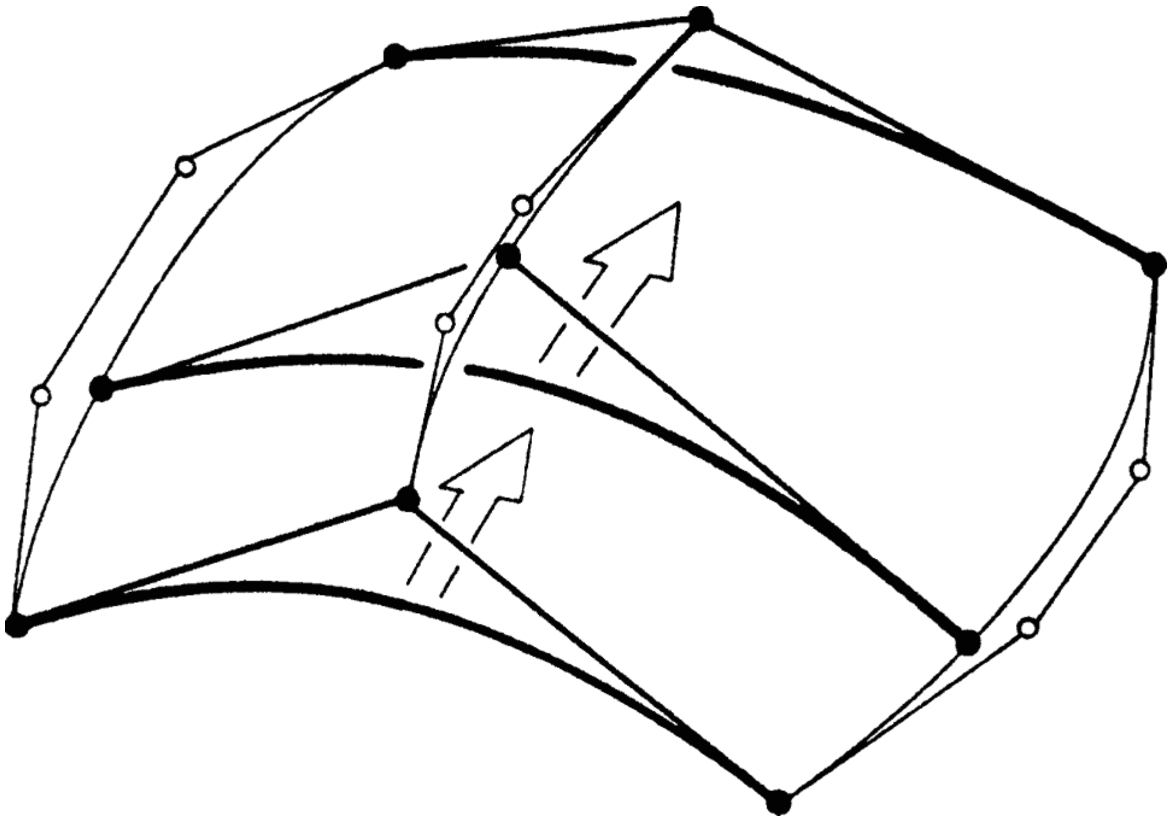
Bézier Patches

- Control net $\boldsymbol{b}_{i,j}$
- Isoparameter curves of degree *m* (*n*) for $\hat{v}$ = const ($\hat{u}$ = const)

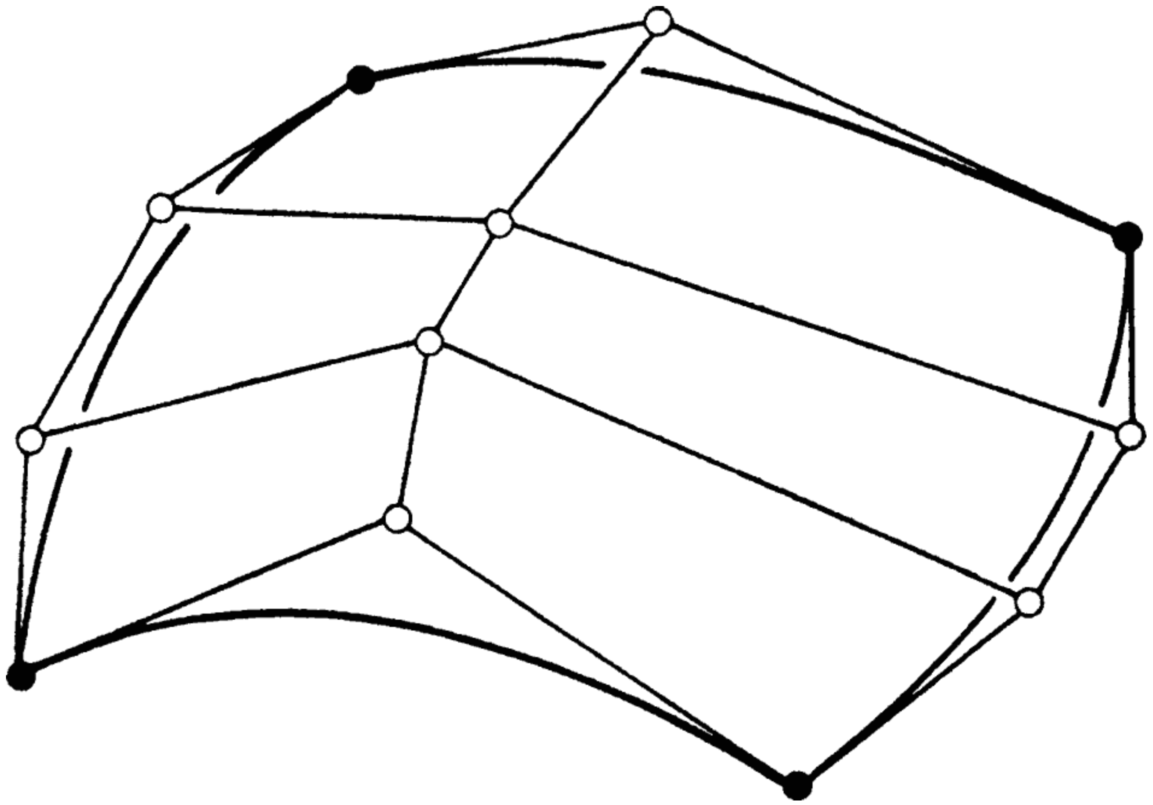
$$\boldsymbol{b}_{\boldsymbol{i}}(\hat{v}) = \sum_{j=0}^{n} \boldsymbol{b}_{\boldsymbol{i,j}} B_j^n(\hat{v}), \qquad i = 0,..,m$$

⚠ ***These curves follow straight lines in parameter space (u,v) and are parallel to the axes u and v***
General curves, such as along the patch diagonals are of degree n+m

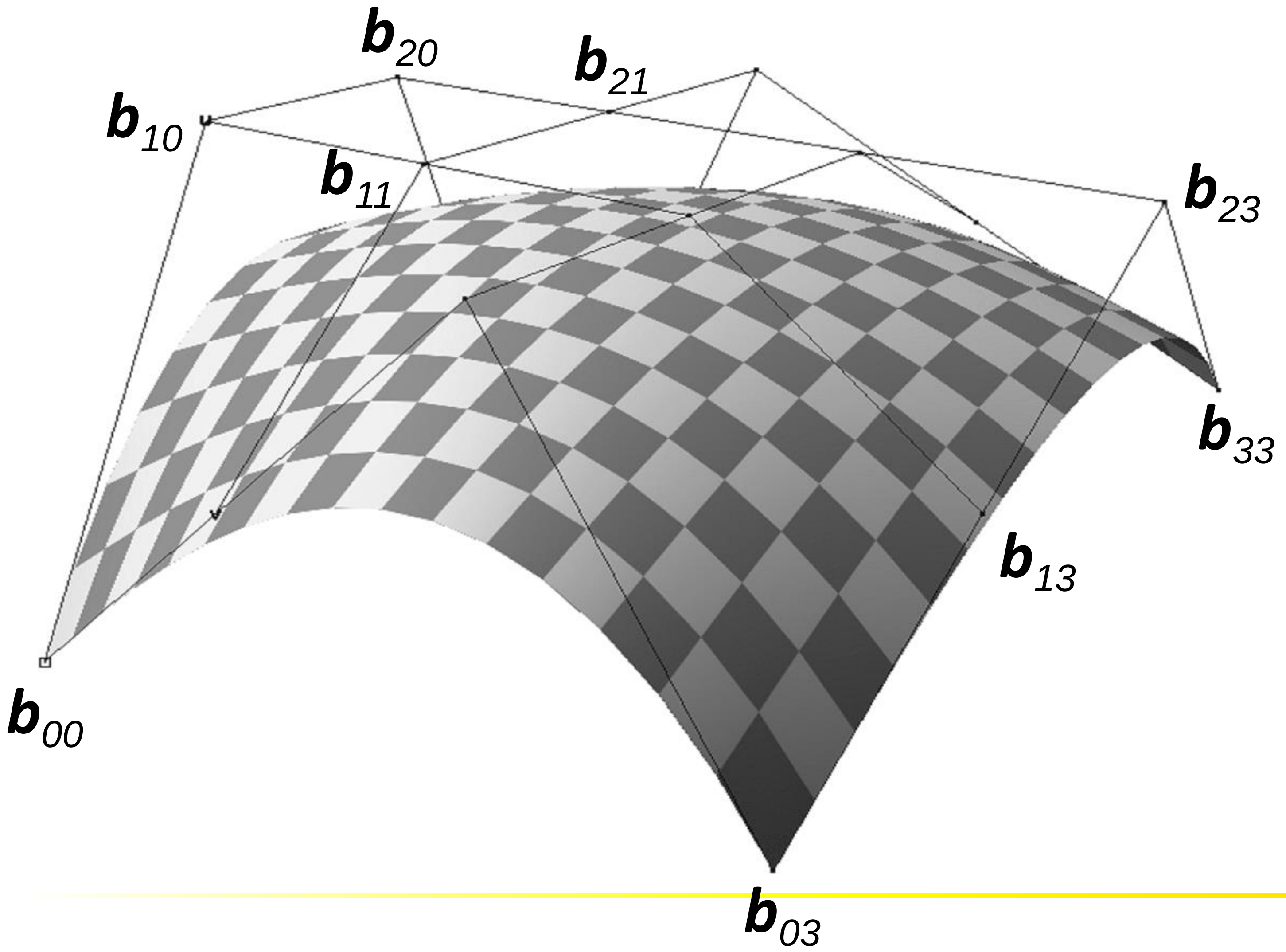
Tensor Product Bézier Patch



Tensor Product Bézier Patch



Example of a Bicubic Bézier Patch



Bézier Patches

- Bézier surfaces have similar properties as Bézier curves:
  - Affine invariance
  - Convex hull property
  - Variation diminishing property
  - ***Boundary curves***: The patch boundary curves are Bézier curves

2D deCasteljau

- Points on the surface by recursive interpolation
- Given: Array of control points $\mathbf{b}_{ij}$, $0 \leq i, j \leq n$ and a parameter pair (u,v)
- Intermediate values in level r of the algorithm computed by

$$\mathbf{b}_{i,j}^{r,r} = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathbf{b}_{i,j}^{r-1,r-1} & \mathbf{b}_{i,j+1}^{r-1,r-1} \\ \mathbf{b}_{i+1,j}^{r-1,r-1} & \mathbf{b}_{i+1,j+1}^{r-1,r-1} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix}$$

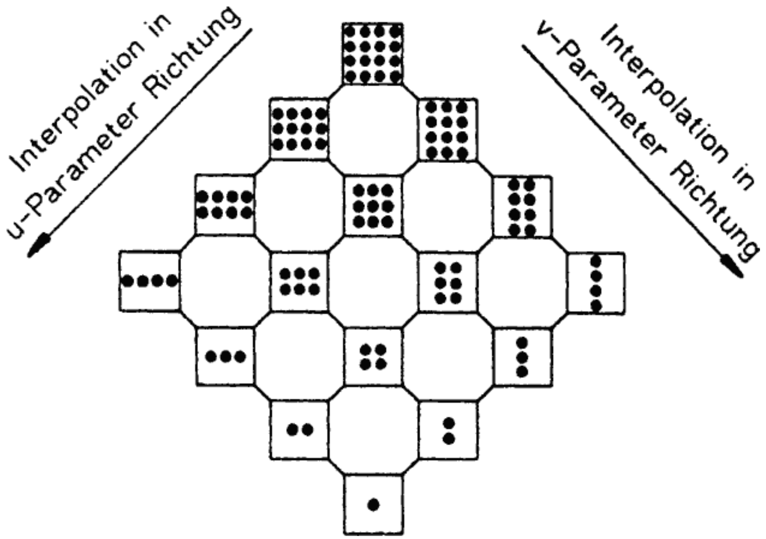
$$r = 1,\ldots,n \quad ; \quad i,j = 0,\ldots,n-r$$

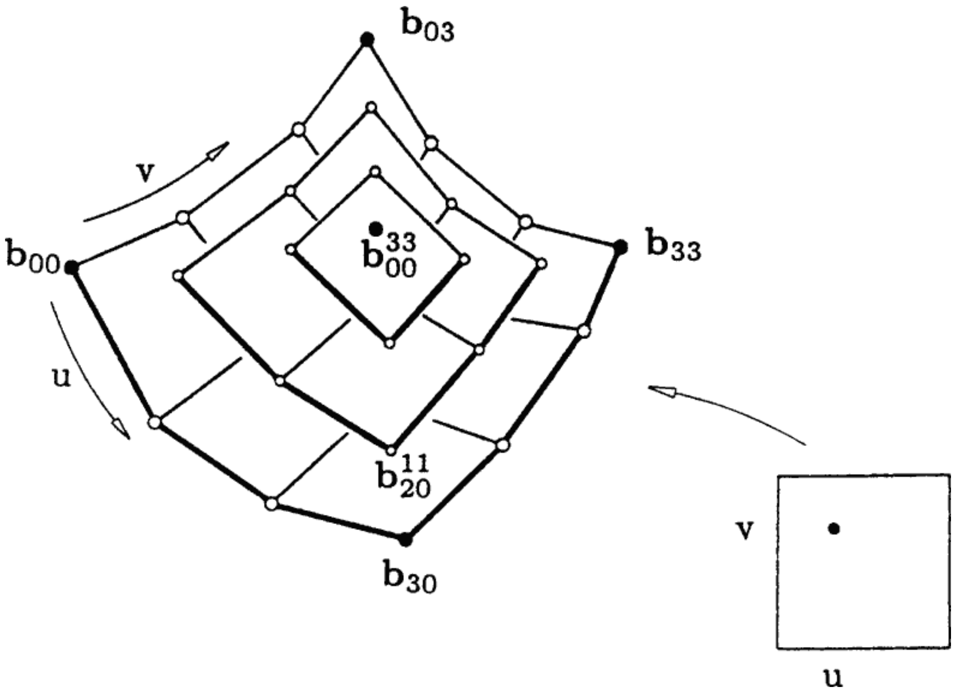
$$\text{with} \quad \mathbf{b}_{i,j}^{0,0} = \mathbf{b}_{i,j}$$

- $\mathbf{b}_{0,0}^{n,n}$ represents a point on the surface (u,v) of the Bézier patch $\mathbf{b}^{n,n}$

⇒ ***bilinear interpolation***

deCasteljau Algorithm





Bézier Patches

⚠ *If the number of control points differs in u- and v-direction we compute k = min(m,n) 2D interpolation steps and proceed with the 1D version of the algorithm*

Bézier Patches

- Example of the deCasteljau Algorithm for $(u,v) = (0.5, 0.5)$:
  - $r = 1$:

$$\begin{bmatrix}0\\0\\0\end{bmatrix} \begin{bmatrix}2\\0\\0\end{bmatrix} \begin{bmatrix}4\\0\\0\end{bmatrix}$$
$$\begin{bmatrix}0\\2\\0\end{bmatrix} \begin{bmatrix}2\\2\\0\end{bmatrix} \begin{bmatrix}4\\2\\2\end{bmatrix}$$
$$\begin{bmatrix}0\\4\\0\end{bmatrix} \begin{bmatrix}2\\4\\4\end{bmatrix} \begin{bmatrix}4\\4\\4\end{bmatrix}$$

Bézier Patches

– $r = 2$:

$$\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 3 \\ 1 \\ 0.5 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 3 \\ 3 \\ 2.5 \end{bmatrix}$$

– $r = 3$:

$$\begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

OpenGL-Surfaces

- Using `glMap2f()` and `glEvalMesh2f()`

```
void myinit(void) {
    glClearColor (0.0, 0.0, 0.0, 1.0);
    glEnable (GL_DEPTH_TEST);
    glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
           0, 1, 12, 4, &ctrlpoints[0][0][0]);
    glEnable(GL_MAP2_VERTEX_3);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_NORMALIZE);
    glMapGrid2f(100, 0.0, 1.0, 100, 0.0, 1.0);
    initlights(); /* for lighted version only */
}
```

OpenGL-Surfaces

```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT |
      GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glRotatef(85.0, 1.0, 1.0, 1.0);
    glEvalMesh2(GL_FILL, 0, 100, 0, 100);
    glPopMatrix();
    glFlush();
}
```

Warping as a 2D Parametric Function

- Given a matrix of vector valued landmark points:

$$\mathbf{m_{ij}} = \begin{pmatrix} x_{ij}(u_i, v_j) \\ y_{ij}(u_i, v_j) \end{pmatrix}$$

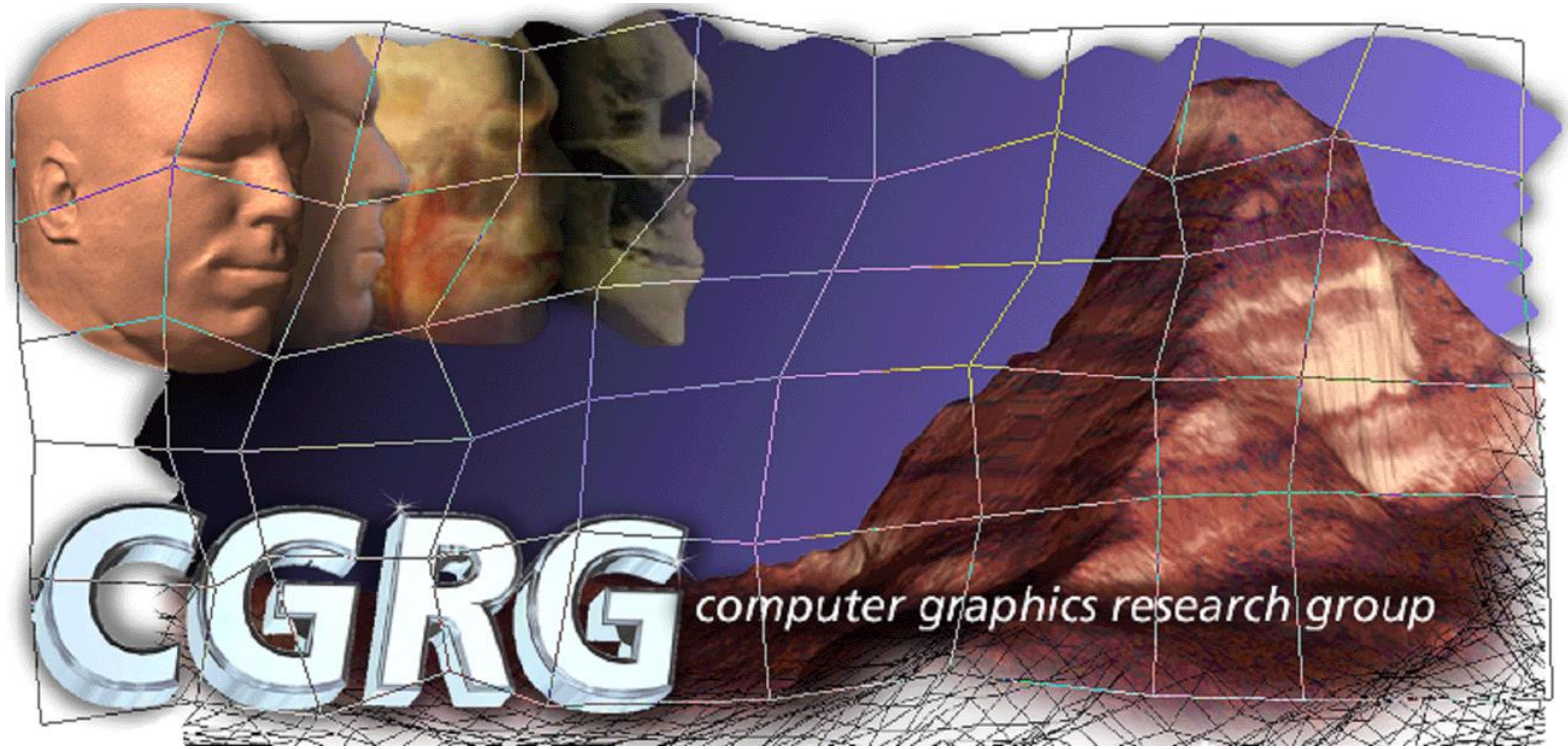
- Solve interpolation problem

$$\mathbf{m}(u_i, v_j) = \begin{bmatrix} B_0(u_i) & \dots & B_n(u_i) \end{bmatrix} \begin{bmatrix} \mathbf{b_{00}} & \dots & \mathbf{b_{0m}} \\ \vdots & & \vdots \\ \mathbf{b_{n0}} & \dots & \mathbf{b_{nm}} \end{bmatrix} \begin{bmatrix} B_0(v_j) \\ \vdots \\ B_m(v_j) \end{bmatrix}$$

- Sample parametric function at (u_i, v_j)

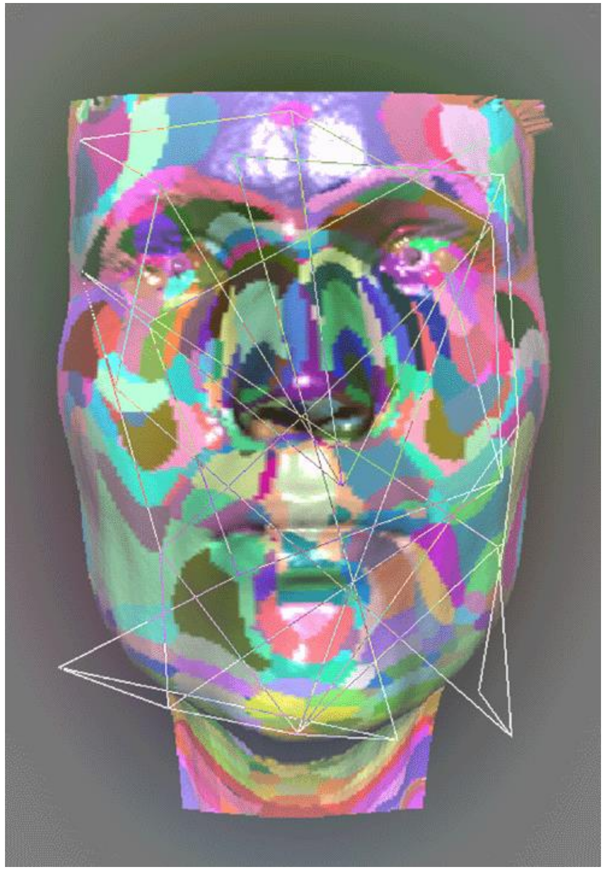
$$I^{m,n}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{b_{i,j}} B_i^m(u) B_j^n(v)$$

Warping as a 2D Parametric Function



Warping as a 2D Parametric Function

Warping as a 2D Parametric Function



Matrix Form

- Generalization of notions for curves

$$\mathbf{b}^{m,n}(u,v) = \begin{bmatrix} B_0^m(u) & \dots & B_m^m(u) \end{bmatrix} \begin{bmatrix} \mathbf{b}_{00} & \dots & \mathbf{b}_{0n} \\ \vdots & & \vdots \\ \mathbf{b}_{m0} & \dots & \mathbf{b}_{mn} \end{bmatrix} \begin{bmatrix} B_0^n(v) \\ \vdots \\ B_n^n(v) \end{bmatrix}$$

- Matrix $\{\mathbf{b}_{ij}\}$ defines the control net of the surface
- Conversion into monomials

$$\mathbf{b}^{m,n}(u,v) = \begin{bmatrix} u^0 & \dots & u^m \end{bmatrix} \mathbf{M}^T \begin{bmatrix} \mathbf{b}_{00} & \dots & \mathbf{b}_{0n} \\ \vdots & & \vdots \\ \mathbf{b}_{m0} & \dots & \mathbf{b}_{mn} \end{bmatrix} \mathbf{N} \begin{bmatrix} v^0 \\ \vdots \\ v^n \end{bmatrix}$$

Matrix Form

- Matrices ***M*** and ***N*** by

$$m_{ij} = (-1)^{j-i} \binom{m}{j} \binom{j}{i}$$

$$n_{ij} = (-1)^{j-i} \binom{n}{j} \binom{j}{i}$$

- Example: Bicubics

$$\mathbf{M} = \mathbf{N} = \begin{bmatrix} 1 & -3 & 3 & 1 \\ 0 & 3 & -6 & 3 \\ 0 & 0 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Derivatives

- Patch derivative computation is important for
  - Continuity between piecewise patches
  - Surface normal
- Similar to curve with partial derivatives in *u*- and *v*-direction
- We distinguish between $\frac{\partial}{\partial u}, \frac{\partial}{\partial v}, \frac{\partial^2}{\partial u \, \partial v}$

Derivatives – Computation

- Exploit separability

$$\frac{\partial}{\partial u}\mathbf{b}^{m,n}(u,v)=\sum_{j=0}^{n}\left[\frac{\partial}{\partial u}\sum_{i=0}^{m}\mathbf{b}_{i,j}B_i^m(u)\right]B_j^n(v)$$

- Use equation for curves

$$\frac{\partial}{\partial u}\mathbf{b}^{m,n}(u,v)=m\sum_{j=0}^{n}\sum_{i=0}^{m-1}\Delta^{1,0}\mathbf{b}_{i,j}B_i^{m-1}(u)B_j^n(v)$$

- Generalized forward difference operator $\Delta^{r,s}$:
  r-times in u- and s- times in v-direction

$$\Delta^{1,0}\mathbf{b}_{i,j}=\mathbf{b}_{i+1,j}-\mathbf{b}_{i,j}\qquad \Delta^{0,1}\mathbf{b}_{i,j}=\mathbf{b}_{i,j+1}-\mathbf{b}_{i,j}$$

Derivatives – Computation

- In *v*-direction

$$\frac{\partial}{\partial v}\mathbf{b}^{m,n}(u,v) = n\sum_{i=0}^{m}\sum_{j=0}^{n-1}\Delta^{0,1}\mathbf{b}_{i,j}\,B_j^{n-1}(v)B_i^m(u)$$

- In general

$$\frac{\partial^r}{\partial u^r}\mathbf{b}^{m,n}(u,v) = \frac{m!}{(m-r)!}\sum_{j=0}^{n}\sum_{i=0}^{m-r}\Delta^{r,0}\mathbf{b}_{i,j}\,B_i^{m-r}(u)B_j^n(v)$$

$$\frac{\partial^s}{\partial v^s}\mathbf{b}^{m,n}(u,v) = \frac{n!}{(n-s)!}\sum_{i=0}^{m}\sum_{j=0}^{n-s}\Delta^{0,s}\mathbf{b}_{i,j}\,B_j^{n-s}(v)B_i^m(u)$$

$$\Delta^{r,0}\mathbf{b}_{i,j} = \Delta^{r-1,0}\mathbf{b}_{i+1,j} - \Delta^{r-1,0}\mathbf{b}_{i,j}$$

$$\Delta^{0,s}\mathbf{b}_{i,j} = \Delta^{0,s-1}\mathbf{b}_{i,j+1} - \Delta^{0,s-1}\mathbf{b}_{i,j}$$

Derivatives – Computation

- Mixed terms of partial derivatives:

$$\frac{\partial^{r+s}}{\partial u^r \, \partial v^s} \mathbf{b}^{\mathbf{m,n}}(u,v) = \frac{m!\, n!}{(m-r)!(n-s)!} \sum_{i=0}^{m-r} \sum_{j=0}^{n-s} \Delta^{r,s} \mathbf{b}_{\mathbf{i,j}} B_i^{m-r}(u) B_j^{n-s}(v)$$

- Vector valued surface in $\boldsymbol{R}^3$
- Cross-boundary derivatives are fundamental

$$\left.\frac{\partial}{\partial u}\right|_{u=0} \qquad \frac{\partial^r}{\partial u^r} \mathbf{b}^{\mathbf{m,n}}(0,v) = \frac{m!}{(m-r)!} \sum_{j=0}^{n} \Delta^{r,0} \mathbf{b}_{0,\mathbf{j}} B_j^n(v)$$

- r^{th} order derivatives at the patch boundaries depend $r+1$ rows (columns) of control points

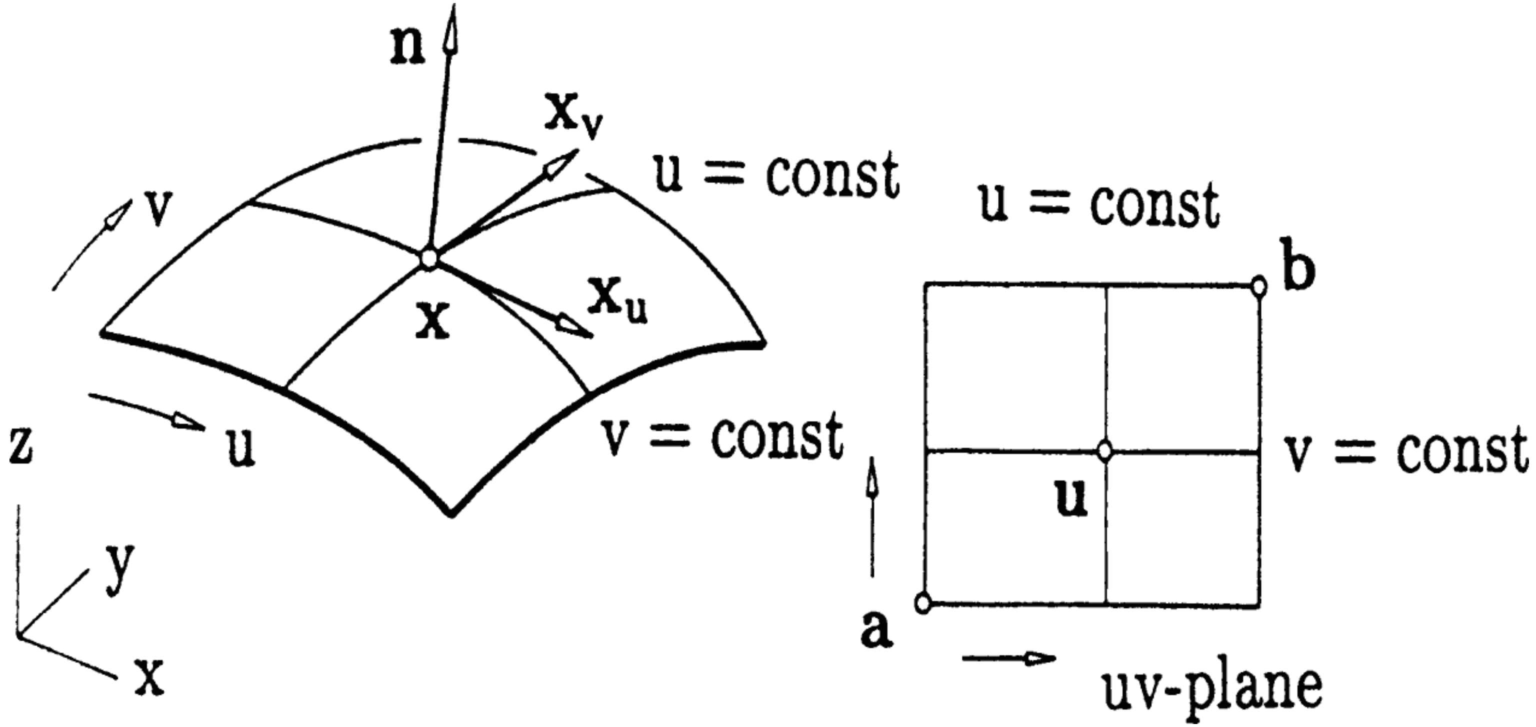
Normal Vector

- Defined as cross product of partial derivatives in *u* and *v*

$$\boldsymbol{n}(u,v)=\frac{\frac{\partial}{\partial u}\boldsymbol{b}^{m,n}(u,v)\times\frac{\partial}{\partial v}\boldsymbol{b}^{m,n}(u,v)}{\left\|\frac{\partial}{\partial u}\boldsymbol{b}^{m,n}(u,v)\times\frac{\partial}{\partial v}\boldsymbol{b}^{m,n}(u,v)\right\|}$$

- Orthogonal to tangential plane at (*u*,*v*)

Tangential Plane and Surface Normal



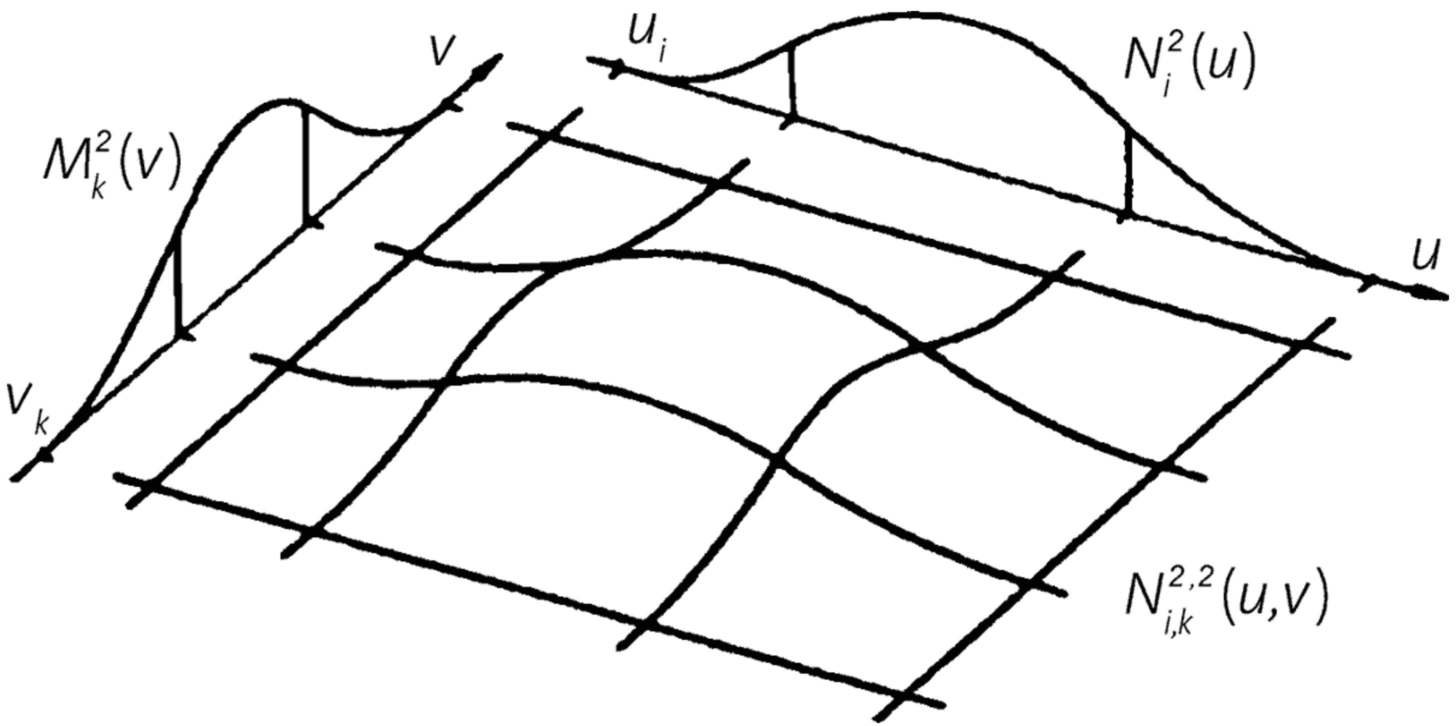
B-Spline Patches

- Fundamental importance in surface modelling
- Most advanced modelling and animation systems are based on NURBS
- Tensor product surface given by 1D bases $M_j^m(v)$ and $N_i^n(u)$ for the knots u_i and v_k
- B-Spline surface $\boldsymbol{x}(u,v)$ defined by

$$\boldsymbol{x}(u,v)=\sum_{i=0}^{k}\sum_{j=0}^{h}\boldsymbol{d}_{\boldsymbol{i,j}}\,M_j^n(v)N_i^m(u)$$

d_{ij}: de Boor Points

Biquadratic B-Spline Basis



B-Spline Patches

- Isoparameter lines (v = const.) form B-Spline curves with deBoor points of type

$$\boldsymbol{d}_{\boldsymbol{i}}(v) = \sum_{j=0}^{h} \boldsymbol{d}_{i,j} \boldsymbol{M}_{j}^{m}(v)$$

- Changing a de Boor point $\boldsymbol{d}_{i,j}$ influences surface in interval $u \in [u_i, u_{i+n+1}]$, $v \in [v_j, v_{j+m+1}]$
- Conversely, patch $u \in [u_i, u_{i+1}]$, $v \in [v_j, v_{j+1}]$ given by de Boor points $\boldsymbol{d}_{i-n,j-m}, \ldots, \boldsymbol{d}_{i,j}$
- Bézier points by multiple knot insertion
- 2D deBoor algorithm

Rational B-Spline Patches (NURBS)

- In analogy to rational curves

$$\mathbf{s}(u,v) = \frac{\sum_{i=0}^{k}\sum_{j=0}^{h} w_{i,j}\,\mathbf{d}_{\mathbf{i,j}}\,N_i^m(u)N_j^n(v)}{\sum_{i=0}^{k}\sum_{j=0}^{h} w_{i,j}\,N_i^m(u)N_j^n(v)}$$

- Weights w_{ij} as an additional degree of freedom

NURB Surfaces

- Rational Surfaces are ***not*** tensor product surfaces, since bases are non-separable of type

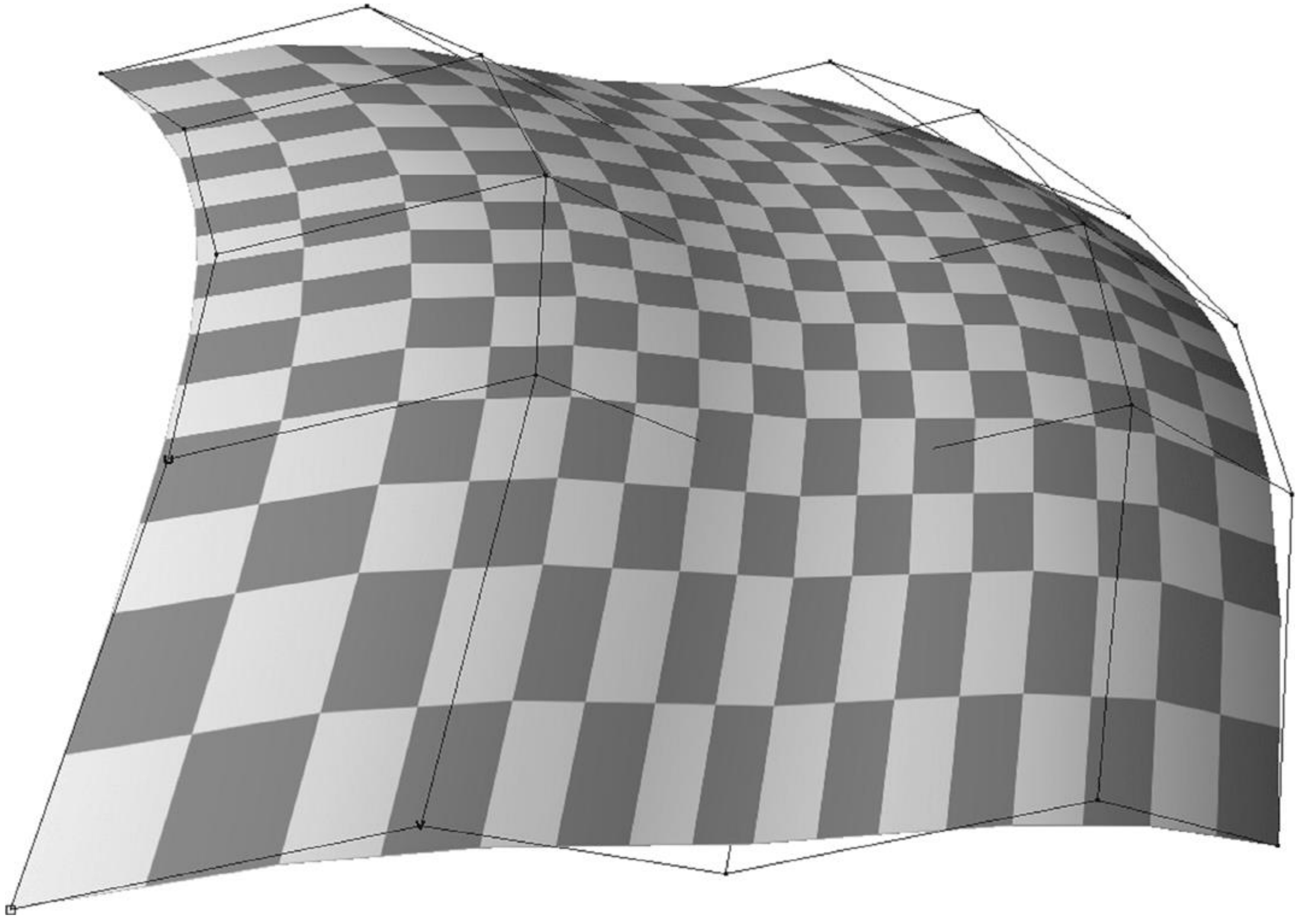
$$F_{i,j}(u,v) = \frac{w_{i,j} N_i^m(u) N_j^n(v)}{\sum_{i=0}^{k} \sum_{j=0}^{h} w_{i,j} N_i^m(u) N_j^n(v)}$$

⚠ ***Recall that we compute all algorithms in 4D and project back to 3D using homogeneous coordinates***

Tensor product algorithms operate in u and in v direction separately

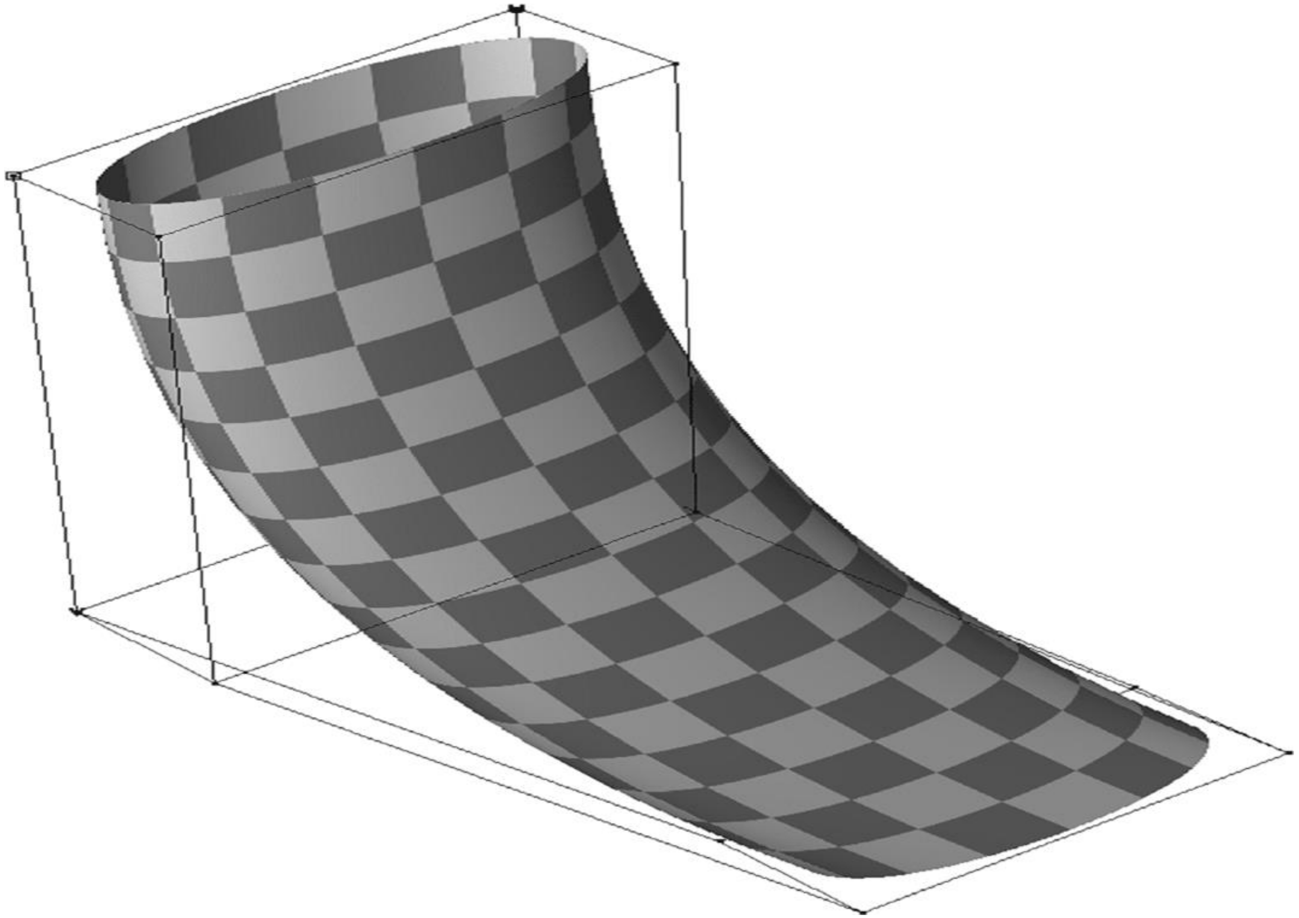
B-Spline Surface

(degree m = 3, non-periodic knot vector)



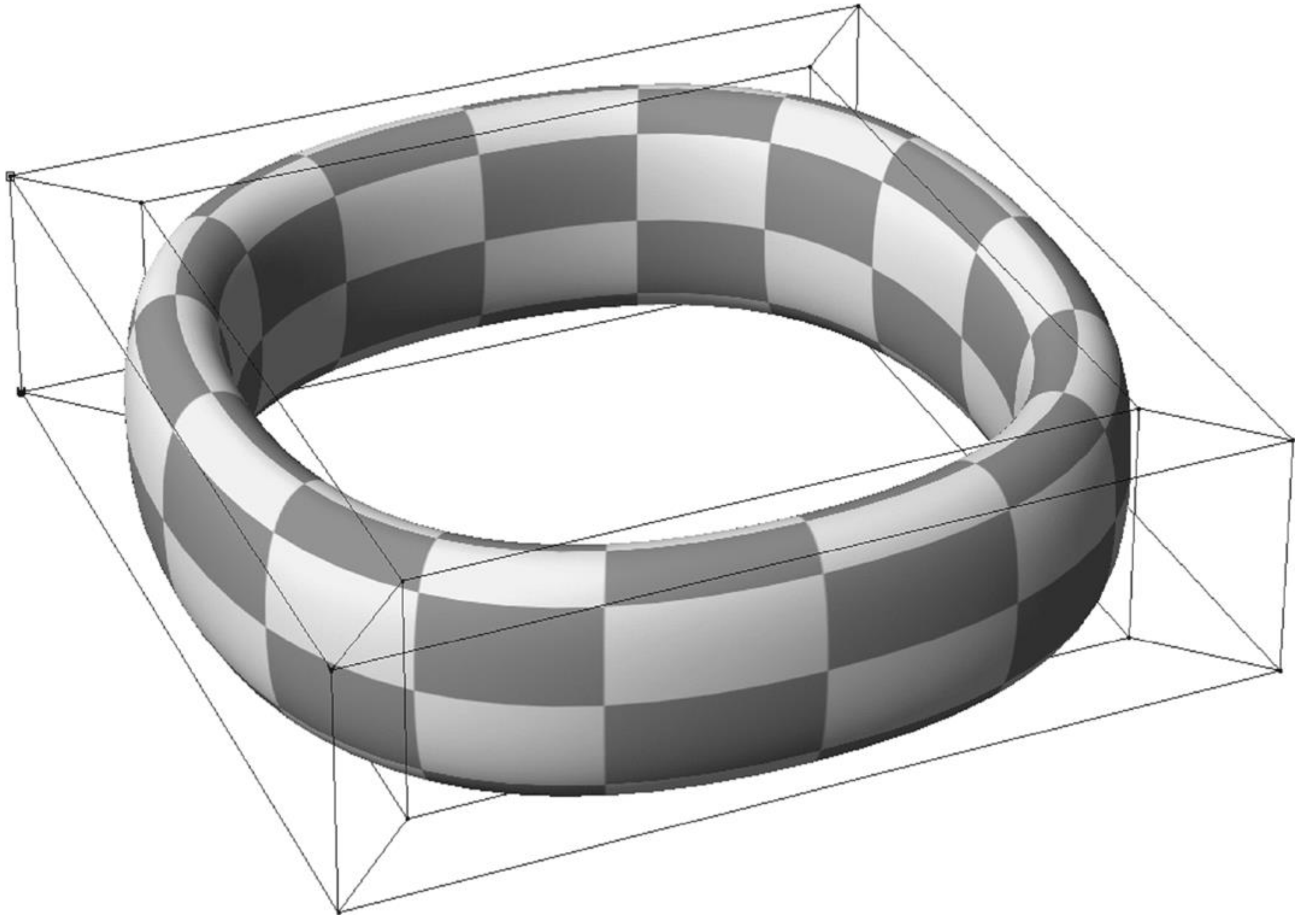
B-Spline Surface

(degree m = 2, knot vector periodic in u-direction)



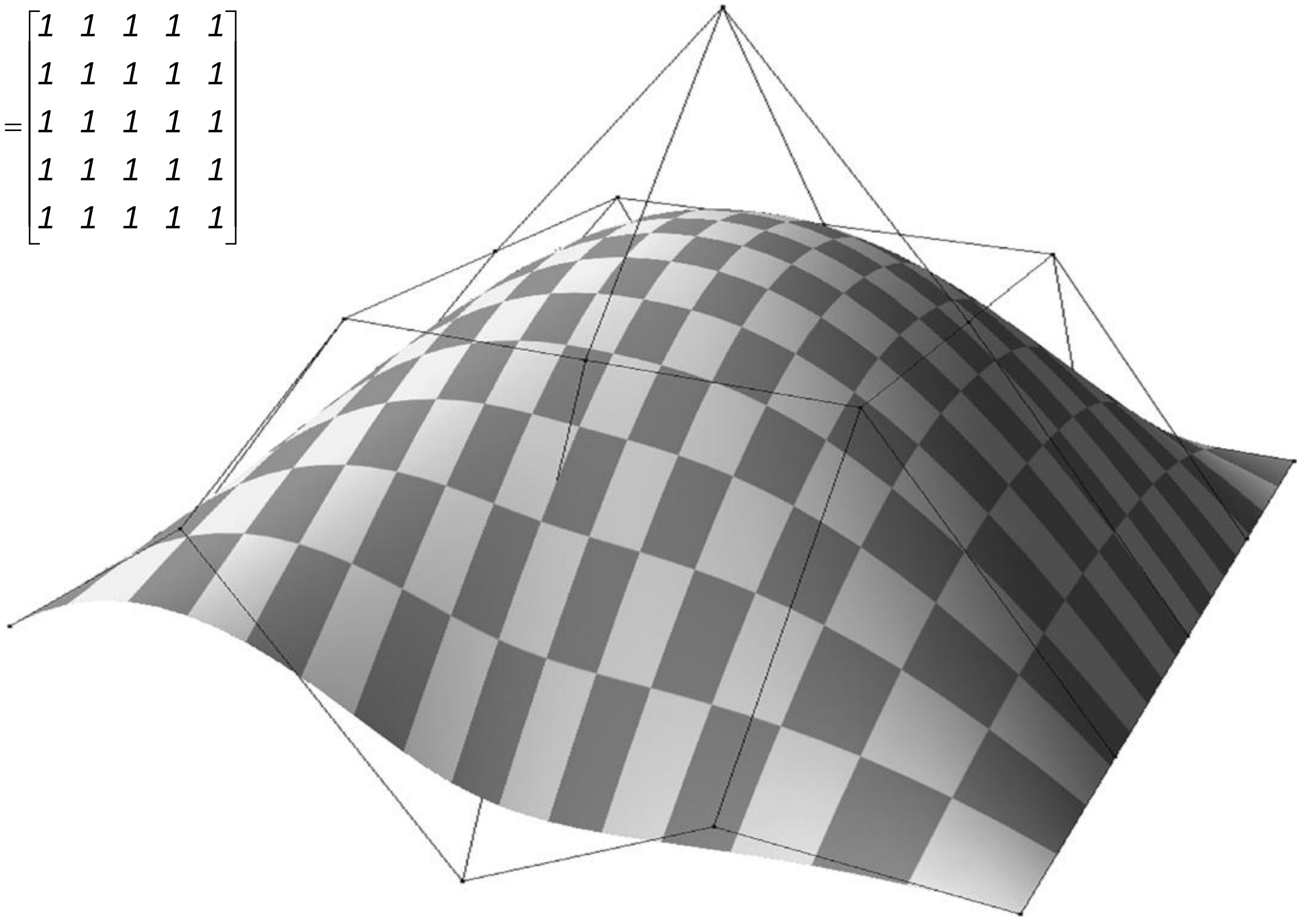
B-Spline Surface

(degree m = 2, knot vector periodic in u and v)



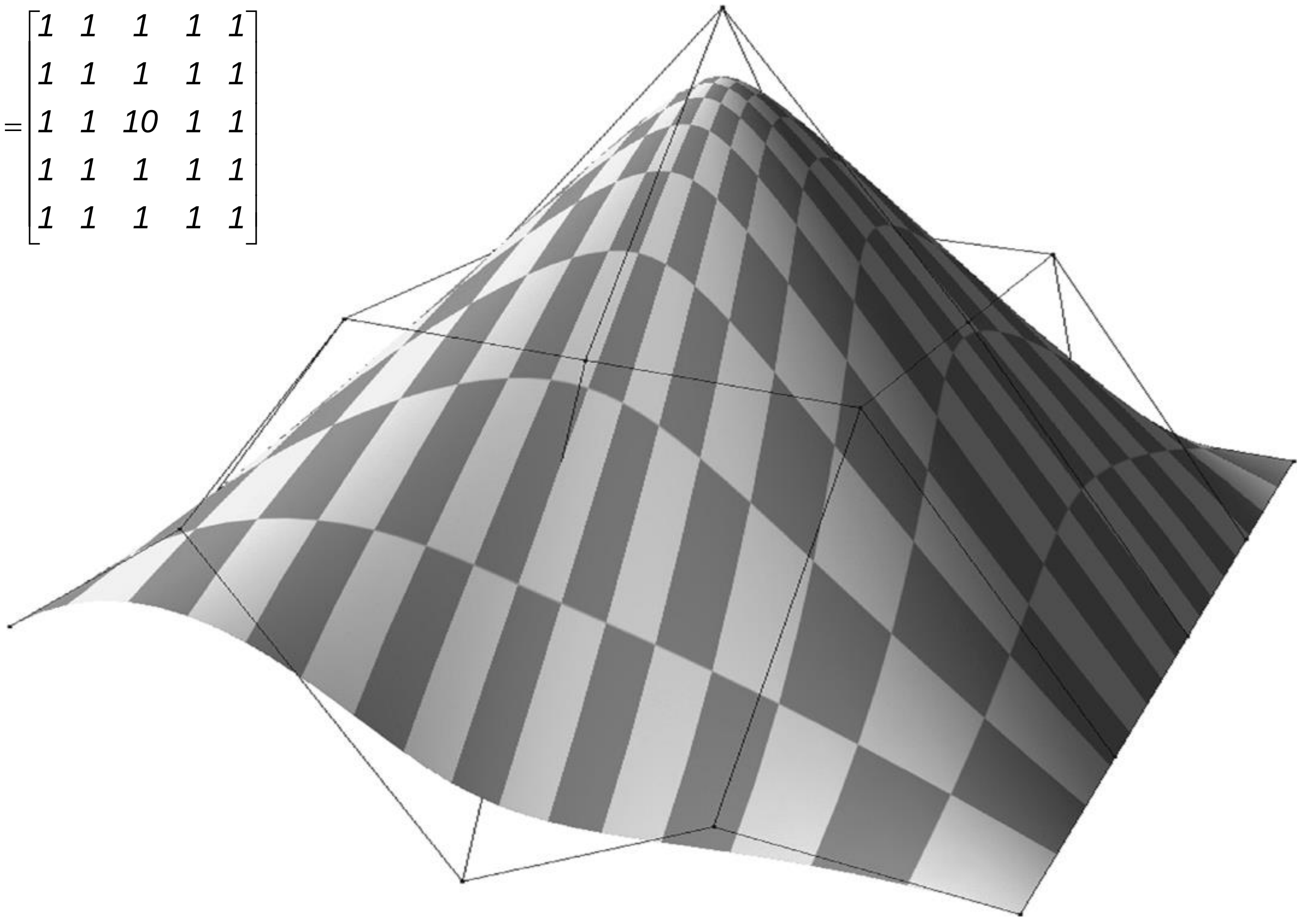
NURB Surface

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



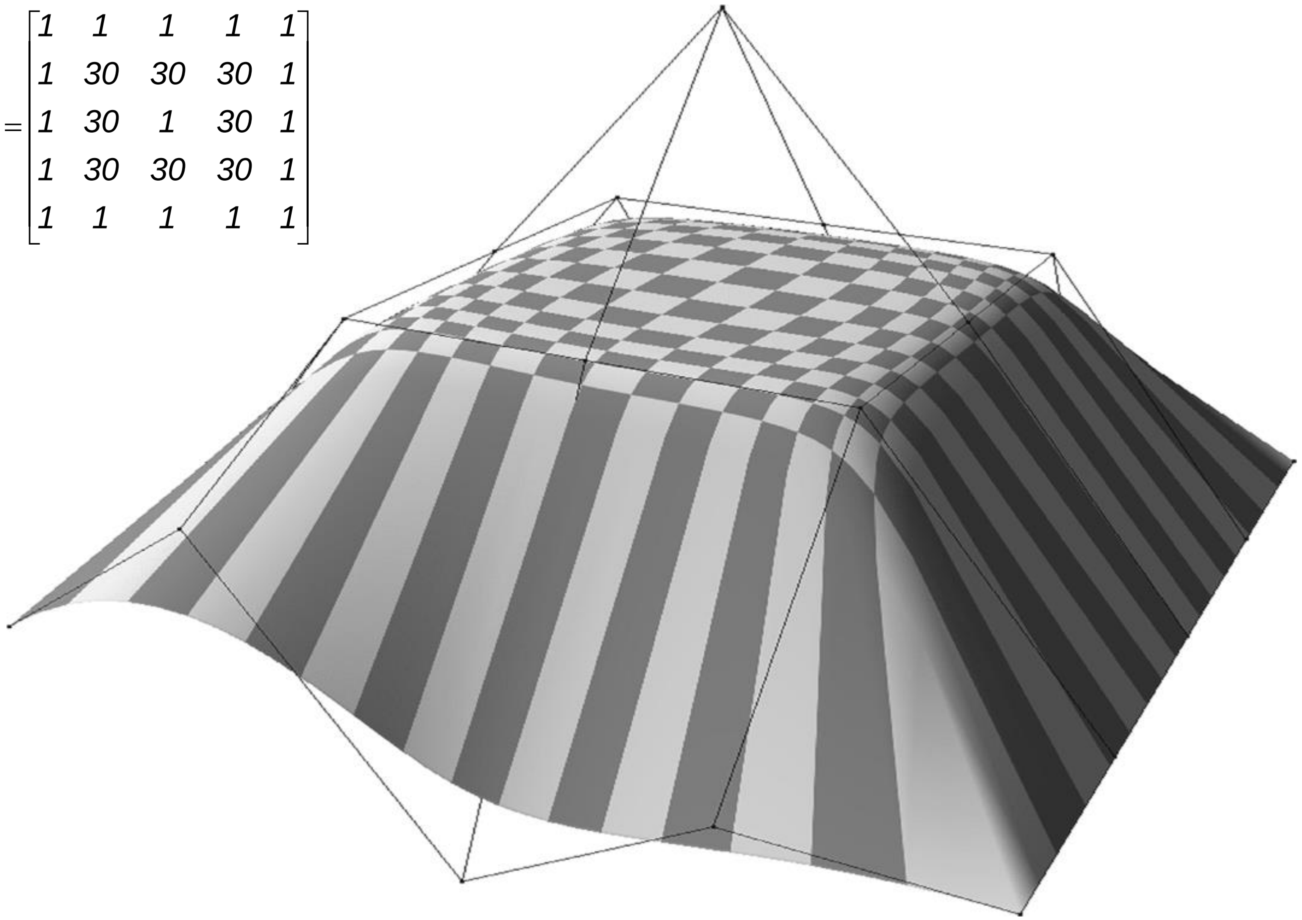
NURB Surface

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 10 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



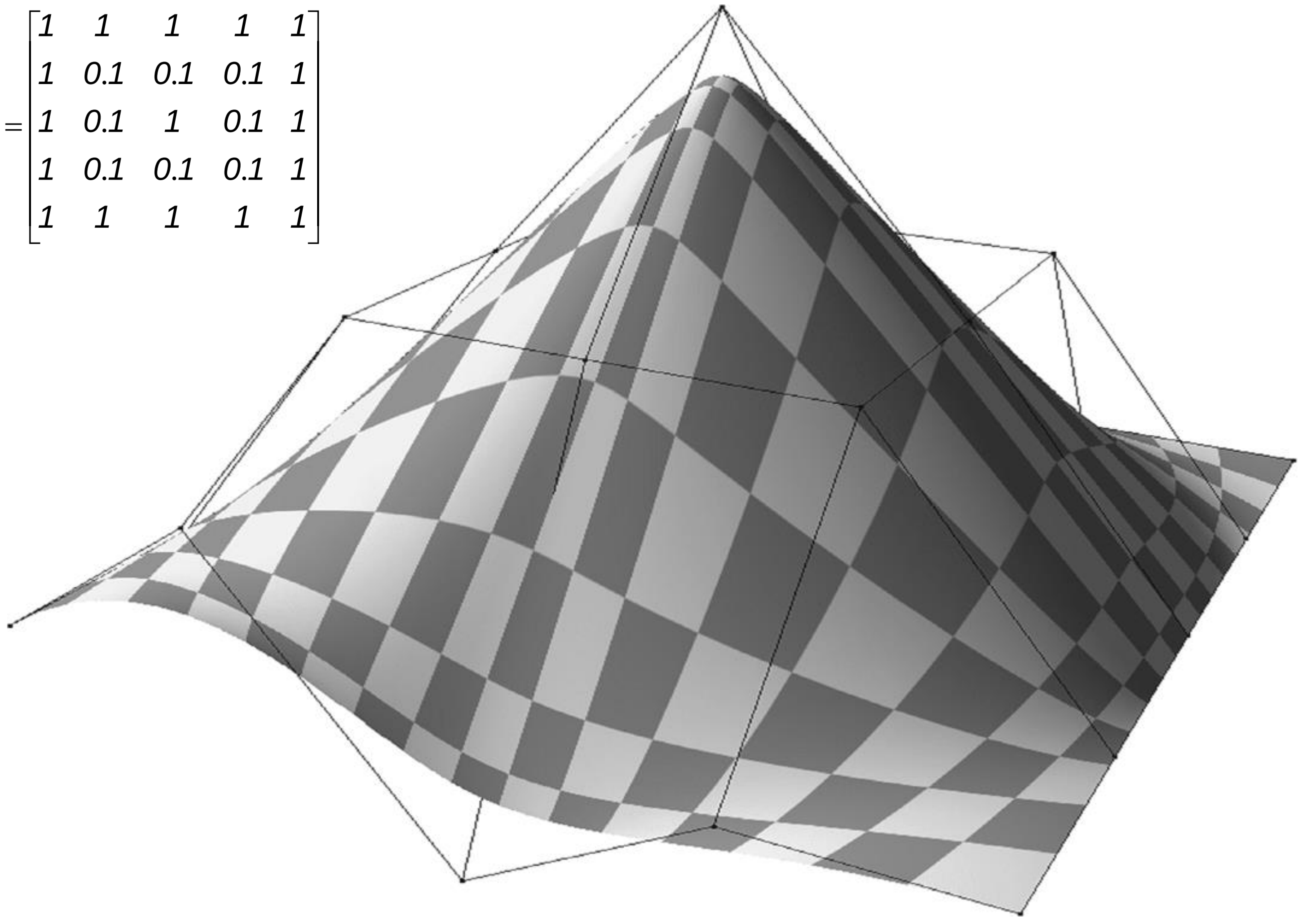
NURB Surface

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 30 & 30 & 30 & 1 \\ 1 & 30 & 1 & 30 & 1 \\ 1 & 30 & 30 & 30 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



NURB Surface

$$W = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0.1 & 0.1 & 0.1 & 1 \\ 1 & 0.1 & 1 & 0.1 & 1 \\ 1 & 0.1 & 0.1 & 0.1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$



OpenGL NURBS

```
GLUnurbsObj *theNurb;
.
.
.
theNurb = gluNewNurbsRenderer();
.
    gluNurbsProperty(theNurb,
        GLU_SAMPLING_TOLERANCE, 25.0);
    gluNurbsProperty(theNurb, GLU_DISPLAY_MODE,
        GLU_FILL);
```

OpenGL NURBS

```
gluBeginSurface(theNurb);
   gluNurbsSurface(theNurb,
          S_NUMKNOTS, sknots,
          T_NUMKNOTS, tknots,
          4 * T_NUMPOINTS,
          4,
          &ctlpoints[0][0][0],
          S_ORDER, T_ORDER,
          GL_MAP2_VERTEX_4);
   gluEndSurface(theNurb);
```

The Tensor Product Approach

- 2D basis functions can be separated along the parameters *u* and *v*
- Examples:
  - Monomials: $$\mathbf{x}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}\boldsymbol{\alpha}_{\mathbf{i,j}}\,u^{i}\,v^{j}$$
  - Lagrange-Polynomials: $$\mathbf{x}(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}\mathbf{p}_{\mathbf{i,j}}\,L_{i}^{m}(u)\,J_{j}^{n}(v)$$

  u_i and v_j define parameter lines – $L_i^m(u)$ and $L_j^n(v)$ Lagrange-Polynomials
- Surface defined by $(n+1)(m+1)$ points $\mathbf{p}_{i,j}$

16 Point Lagrange Patch (interpolating)