Tensor Product Surfaces
Prof. Dr. Markus Gross
Overview

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

- Let \( x(u) = \sum_{i=0}^{m} c_i F_i(u) \) be a 2D or 3D spatial curve given by the bases \( F_i \).

- Coefficients \( c_i \) are functions of a second parameter \( v \).

- \( c_i \)-curves as linear combinations of \( G_j \):
  \[
  c_i(v) = \sum_{j=0}^{v} \alpha_{i,j} G_j(v)
  \]

- \( \Rightarrow \) so-called tensor product surface \( x(u,v) \):
  \[
  x(u,v) = \sum_{i} c_i(v) F_i(u) = \sum_{i=0}^{m} \sum_{j=0}^{n} \alpha_{i,j} F_i(u) G_j(v)
  \]
“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 $B_i(u) \cdot B_j(v)$ can be constructed by

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

$$B_{i,j}^m(u,v) = B_i^m(u) \cdot B_j^m(v) \quad i,j = 0,\ldots,m$$
Tensor Product Surface
(Trace of a Curve in Space)
Bézier Patches

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

\[
b^m(u) = \sum_{i=0}^{m} b_i B_i^m(u)
\]

- Control points \( b_i \) as Bézier-curves of degree \( n \):

\[
b_i = b_i(v) = \sum_{j=0}^{n} b_{i,j} B_j^n(v)
\]

- Point \( b_{m,n}(u,v) \) on the surface with:

\[
b^{m,n}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{i,j} B_i^m(u) B_j^n(v)
\]
Bézier Patches

- Control net $b_{i,j}$
- Isoparameter curves of degree $m \cdot n$ for $\hat{v} = \text{const}$ ($\hat{u} = \text{const}$)

$$b_i(\hat{v}) = \sum_{j=0}^{n} b_{i,j} B_j^n(\hat{v}), \quad i = 0,..,m$$

⚠️ 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 $b_{ij}$, $0 \leq i, j \leq n$ and a parameter pair $(u,v)$
- Intermediate values in level $r$ of the algorithm computed by

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

$r = 1,..,n$ ; $i, j = 0,..,n-r$

with $b_{0,0}^{0,0} = b_{i,j}$

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

$\Rightarrow$ 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 $(u, v) = (0.5, 0.5)$:
  
  - $r = 1$:

\[
\begin{bmatrix}
0 & 2 & 4 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & 2 & 4 \\
2 & 2 & 2 \\
0 & 0 & 2 \\
0 & 2 & 4 \\
4 & 4 & 4 \\
0 & 4 & 4
\end{bmatrix}
\]
Bézier Patches

- \( r = 2 \):

\[
\begin{bmatrix}
1 \\
1 \\
0 \\
1 \\
3 \\
1
\end{bmatrix}
\begin{bmatrix}
3 \\
1 \\
0.5 \\
3 \\
3 \\
2.5
\end{bmatrix}
\]

- \( r = 3 \):

\[
\begin{bmatrix}
2 \\
2 \\
1
\end{bmatrix}
\]
OpenGL-Surfaces

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

```c
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 */
}
```
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:

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

• Solve interpolation problem

\[
m(u_i, v_j) = \begin{bmatrix} B_0(u_i) & \ldots & B_n(u_i) \end{bmatrix} \begin{bmatrix} b_{00} & \ldots & b_{0m} \\ \vdots & \ddots & \vdots \\ b_{n0} & \ldots & b_{nm} \end{bmatrix} \begin{bmatrix} B_0(v_j) \\ \vdots \\ B_m(v_j) \end{bmatrix}
\]

• Sample parametric function at \((u_i, v_j)\)

\[
I_{m,n}(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{i,j} B_i^m(u)B_j^n(v)
\]
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

\[ b_{m,n}^{u,v} = \begin{bmatrix} B_0^m(u) & \cdots & B_m^m(u) \end{bmatrix} \begin{bmatrix} b_{00} & \cdots & b_{0n} \\ \vdots & \ddots & \vdots \\ b_{m0} & \cdots & b_{mn} \end{bmatrix} \begin{bmatrix} B_0^n(v) \\ \vdots \\ B_n^n(v) \end{bmatrix} \]

- Matrix \( \{b_{ij}\} \) defines the control net of the surface

- Conversion into monomials

\[ b_{m,n}^{u,v} = \begin{bmatrix} u^0 & \cdots & u^m \end{bmatrix} M^T \begin{bmatrix} b_{00} & \cdots & b_{0n} \\ \vdots & \ddots & \vdots \\ b_{m0} & \cdots & b_{mn} \end{bmatrix} N \begin{bmatrix} v^0 \\ \vdots \\ v^n \end{bmatrix} \]
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

\[
M = N = \begin{bmatrix}
1 & -3 & 3 & 1 \\
0 & 3 & -6 & 3 \\
0 & 0 & 3 & -3 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
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

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

- Use equation for curves

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

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

\[
\Delta^{1,0}_{i,j} = b_{i+1,j} - b_{i,j} \quad \Delta^{0,1}_{i,j} = b_{i,j+1} - b_{i,j}
\]
Derivatives – Computation

• In \( \nu \)-direction

\[
\frac{\partial}{\partial \nu} b^{m,n}(u, \nu) = n \sum_{i=0}^{m} \sum_{j=0}^{n-1} \Delta^{0,1} b_{i,j} B_{j}^{n-1}(\nu) B_{i}^{m}(u)
\]

• In general

\[
\frac{\partial^{r}}{\partial u^{r}} b^{m,n}(u, \nu) = \frac{m!}{(m-r)!} \sum_{j=0}^{n} \sum_{i=0}^{m-r} \Delta^{r,0} b_{i,j} B_{i}^{m-r}(u) B_{j}^{n}(\nu)
\]

\[
\frac{\partial^{s}}{\partial \nu^{s}} b^{m,n}(u, \nu) = \frac{n!}{(n-s)!} \sum_{i=0}^{m} \sum_{j=0}^{n-s} \Delta^{0,s} b_{i,j} B_{j}^{n-s}(\nu) B_{i}^{m}(u)
\]

\[
\Delta^{r,0} b_{i,j} = \Delta^{r-1,0} b_{i+1,j} - \Delta^{r-1,0} b_{i,j}
\]

\[
\Delta^{0,s} b_{i,j} = \Delta^{0,s-1} b_{i,j+1} - \Delta^{0,s-1} b_{i,j}
\]
Derivatives – Computation

• Mixed terms of partial derivatives:

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

• Vector valued surface in \( \mathbb{R}^3 \)

• Cross-boundary derivatives are fundamental

\[
\left. \frac{\partial}{\partial u} \right|_{u=0} \quad \left. \frac{\partial^r}{\partial u^r} b^{m,n}(0,v) = \frac{m!}{(m-r)!} \sum_{j=0}^{n} \Delta^{r,0} b_0^j B_j^n(v) \right.
\]

• \( r^{th} \) 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 $v$

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

• 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 $x(u,v)$ defined by

$$x(u,v) = \sum_{i=0}^{k} \sum_{j=0}^{h} d_{i,j} M_{j}^{m}(v) N_{i}^{n}(u)$$

$d_{ij}$: de Boor Points
Biquadratic B-Spline Basis
B-Spline Patches

• Isoparameter lines ($v = \text{const.}$) form B-Spline curves with deBoor points of type

$$d_i(v) = \sum_{j=0}^{h} d_{i,j} M_j^m(v)$$

• Changing a de Boor point $d_{i,j}$ 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 $d_{i-n,j-m}, \ldots, d_{i,j}$

• Bézier points by multiple knot insertion

• 2D deBoor algorithm
Rational B-Spline Patches (NURBS)

• In analogy to rational curves

\[
s(u,v) = \frac{\sum_{i=0}^{k} \sum_{j=0}^{h} w_{i,j} d_{i,j} N_i^m(u) N_j^n(v)}{\sum_{i=0}^{k} \sum_{j=0}^{h} w_{i,j} N_i^m(u) N_j^n(v)}
\]

• 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

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

⚠️ 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

$$w = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \\
1 & 1 & 10 & 1 & 1 \\
1 & 1 & 11 & 1 & 1 \\
1 & 1 & 11 & 1 & 1 \\
\end{bmatrix}$$
NURB Surface

\[ w = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 30 & 30 & 30 & 1 \\ 1 & 30 & 1 & 30 & 1 \\ 1 & 30 & 30 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \]
NURB Surface

\[ w = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\
1 & 0.1 & 0.1 & 0.1 & 1 \\
1 & 0.1 & 1 & 0.1 & 1 \\
1 & 0.1 & 0.1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 \end{bmatrix} \]
GLUnurbsObj *theNurb;

theNurb = gluNewNurbsRenderer();

gluNurbsProperty(theNurb,
    GLU_SAMPLING_TOLERANCE, 25.0);
gluNurbsProperty(theNurb, GLU_DISPLAY_MODE,
    GLU_FILL);
OpenGL NURBS

```c
void gluBeginSurface(GLuNurb* 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);
}

void gluEndSurface(GLuNurb* theNurb) {
}
```
The Tensor Product Approach

• 2D basis functions can be separated along the parameters $u$ and $v$
• Examples:
  – Monomials: $x(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \alpha_{i,j} u^i v^j$
  – Lagrange-Polynomials: $x(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} p_{i,j} L_i^m(u) J_j^n(v)$

$u_i$ and $v_j$ define parameter lines – $L_i^m(u)$ and $J_j^n(v)$ Lagrange-Polynomials
• Surface defined by $(n+1)(m+1)$ points $p_{i,j}$
16 Point Lagrange Patch (interpolating)