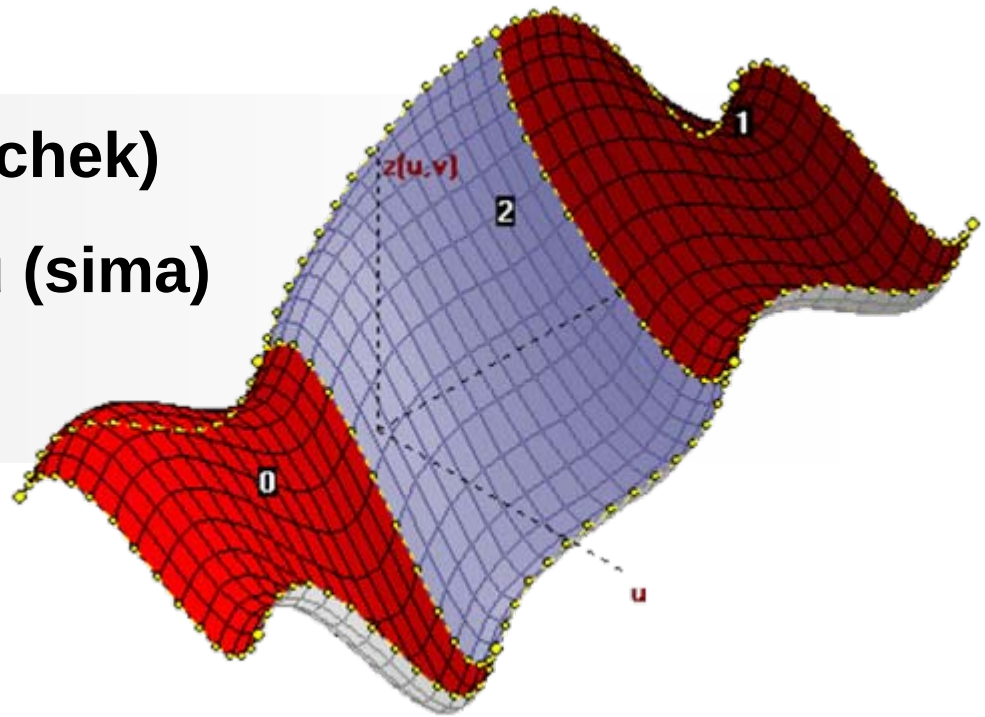
# Bikubikus felületi foltok (patchek) megjelenítése és $C^1$ osztályú (sima) illesztése



# Bikubikus felületi foltok általában

**Mátrix alak:**

$$\begin{cases} F:[0,1]\times[0,1]\to R^3 \\ F(u,v)=\begin{bmatrix} f_0(u) & f_1(u) & f_2(u) & f_3(u) \end{bmatrix}\cdot M\cdot \begin{bmatrix} f_0(v) \\ f_1(v) \\ f_2(v) \\ f_3(v) \end{bmatrix} \end{cases}$$

**ahol:**

- $f_0, f_1, f_2, f_3 : [0,1]\to R$ **ún. súlyfüggvények (harmadfokú polinomok), azaz** $\sum_{i=0}^{3} f_i(t)=1, \forall\, t\in[0,1]$

- $M=\begin{bmatrix} I_{00} & I_{01} & I_{02} & I_{03} \\ I_{10} & I_{11} & I_{12} & I_{13} \\ I_{20} & I_{21} & I_{22} & I_{23} \\ I_{30} & I_{31} & I_{32} & I_{33} \end{bmatrix}=\left[I_{kl}(x_{kl},y_{kl},z_{kl})\right]_{k,l=0,1,2,3}\in M_{4,4}(R^3)$ **a rendelkezésünkre álló információk (kontrollpontok, érintők, twist-vektorok, stb.) vektormátrixa**

**Komponensfüggvényekre lebontva:**

$$\begin{cases} x(u,v)=\sum_{k=0}^{3}\sum_{l=0}^{3} x_{kl}\cdot f_k(u)\cdot f_l(v) \\ y(u,v)=\sum_{k=0}^{3}\sum_{l=0}^{3} y_{kl}\cdot f_k(u)\cdot f_l(v), \forall\,(u,v)\in[0,1]\times[0,1] \\ z(u,v)=\sum_{k=0}^{3}\sum_{l=0}^{3} z_{kl}\cdot f_k(u)\cdot f_l(v) \end{cases}$$

# Bikubikus felületi foltok általában (folytatás)

**Parciális deriváltak egy adott pontban:**

$$\frac{\partial F}{\partial u}(u,v) = \begin{bmatrix} \frac{df_0}{du}(u) & \frac{df_1}{du}(u) & \frac{df_2}{du}(u) & \frac{df_3}{du}(u) \end{bmatrix} \cdot M \cdot \begin{bmatrix} f_0(v) \\ f_1(v) \\ f_2(v) \\ f_3(v) \end{bmatrix}$$

$$\frac{\partial F}{\partial v}(u,v) = \begin{bmatrix} f_0(u) & f_1(u) & f_2(u) & f_3(u) \end{bmatrix} \cdot M \cdot \begin{bmatrix} \frac{df_0}{dv}(v) \\ \frac{df_1}{dv}(v) \\ \frac{df_2}{dv}(v) \\ \frac{df_3}{dv}(v) \end{bmatrix}$$

**Normális egy adott pontban:**

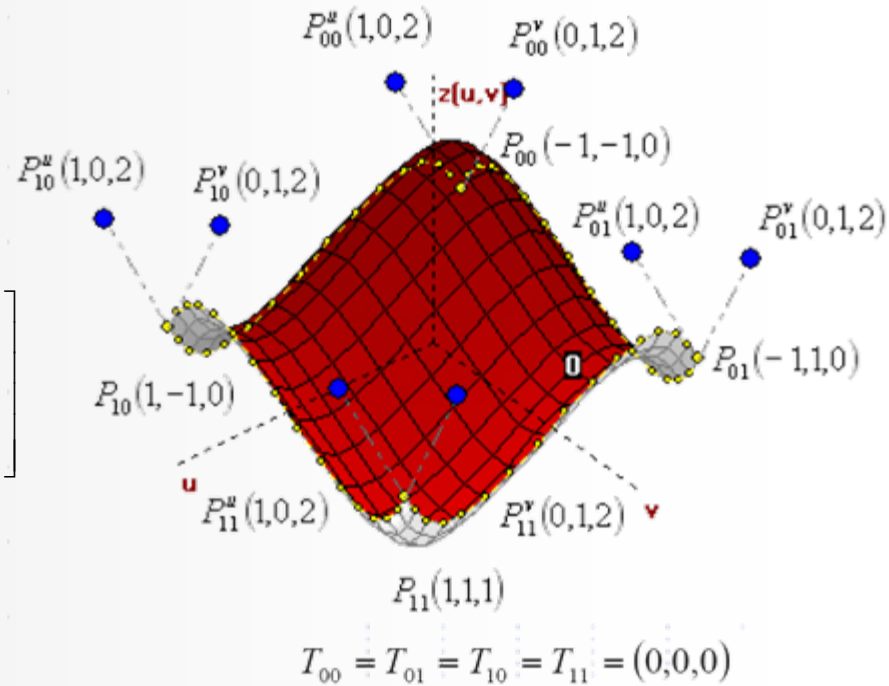
$$n(u,v) = \frac{\partial F}{\partial u}(u,v) \times \frac{\partial F}{\partial v}(u,v)$$

# Példák: Coons-Hermite

$$\begin{cases} f_0, f_1, f_2, f_3 : [0,1] \to R \\ f_0(t) = 2t^3 - 3t^2 + 1 \\ f_1(t) = -2t^3 + 3t^2 \\ f_2(t) = t^3 - 2t^2 + t \\ f_3(t) = t^3 - t^2 \end{cases}$$

$$\begin{cases} \frac{df_0}{dt}, \frac{df_1}{dt}, \frac{df_2}{dt}, \frac{df_3}{dt} : [0,1] \to R \\ \frac{df_0}{dt}(t) = 6t^2 - 6t \\ \frac{df_1}{dt}(t) = -6t^2 + 6t \\ \frac{df_2}{dt}(t) = 3t^2 - 4t + 1 \\ \frac{df_3}{dt}(t) = 3t^2 - 2t \end{cases}$$

$$M = \begin{bmatrix} P_{00} & P_{01} & P_{00}^v & P_{01}^v \\ P_{10} & P_{11} & P_{10}^v & P_{11}^v \\ P_{00}^u & P_{01}^u & T_{00} & T_{01} \\ P_{10}^u & P_{11}^u & T_{10} & T_{11} \end{bmatrix}$$



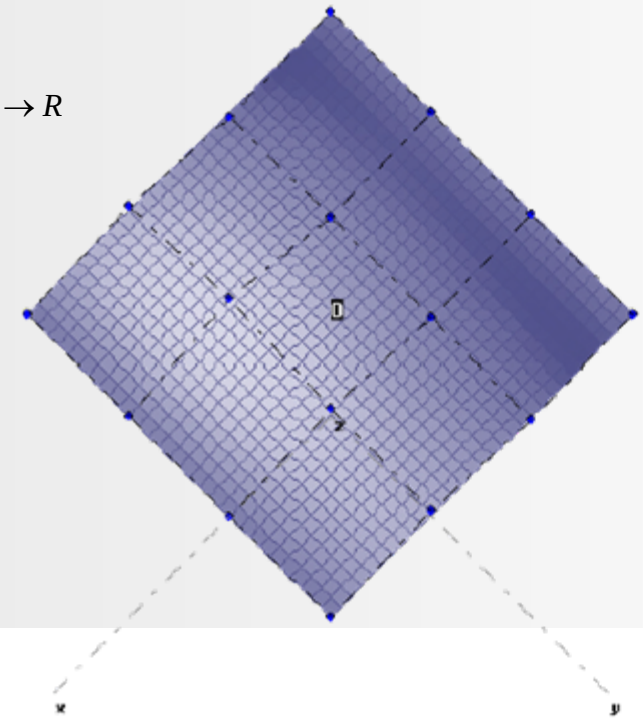

$$T_{00} = T_{01} = T_{10} = T_{11} = (0,0,0)$$

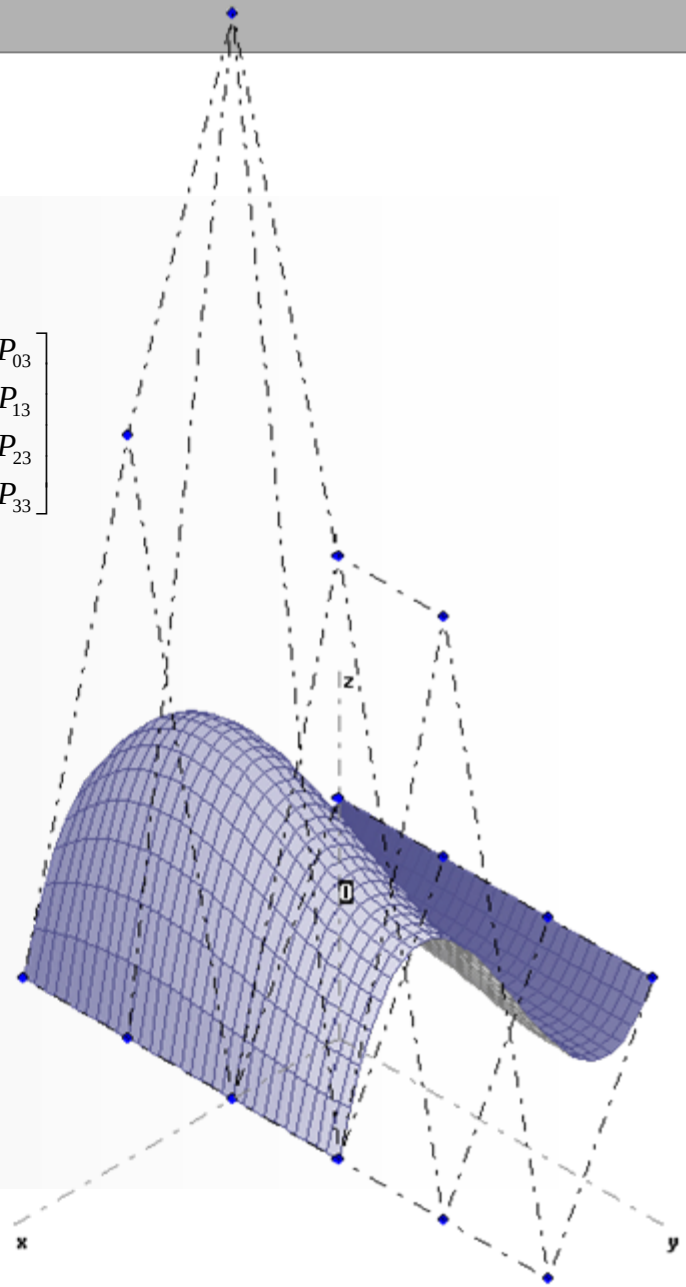
# Példák: Bézier

$$\begin{cases} f_0, f_1, f_2, f_3 : [0,1] \to R \\ f_0(t) = 1 - 3t + 3t^2 - t^3 \\ f_1(t) = 3t - 6t^2 + 3t^3 \\ f_2(t) = 3t^2 - 3t^3 \\ f_3(t) = t^3 \end{cases}$$

$$M = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$

$$\begin{cases} \frac{df_0}{dt}, \frac{df_1}{dt}, \frac{df_2}{dt}, \frac{df_3}{dt} : [0,1] \to R \\ \frac{df_0}{dt}(t) = -3 + 6t - 3t^2 \\ \frac{df_1}{dt}(t) = 3 - 12t + 9t^2 \\ \frac{df_2}{dt}(t) = 6t - 9t^2 \\ \frac{df_3}{dt}(t) = 3t^2 \end{cases}$$



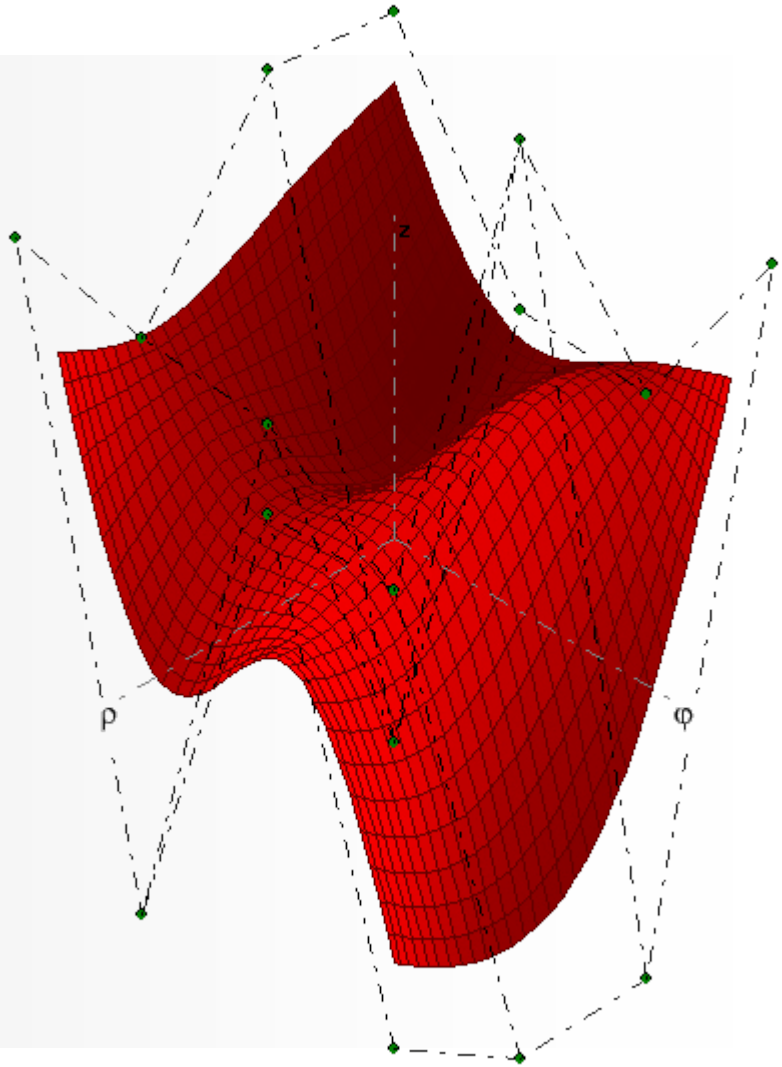



# Példák: B-spline

$$\begin{cases} f_0, f_1, f_2, f_3 : [0,1] \to R \\ f_0(t) = \dfrac{(1-t)^3}{6} \\ f_1(t) = \dfrac{3(1-t)^2 t + 3(1-t) + 1}{6} \\ f_2(t) = \dfrac{3(1-t)t^2 + 3t + 1}{6} \\ f_3(t) = \dfrac{t^3}{6} \end{cases}$$

$$\begin{cases} \dfrac{df_0}{dt}, \dfrac{df_1}{dt}, \dfrac{df_2}{dt}, \dfrac{df_3}{dt} : [0,1] \to R \\ \dfrac{df_0}{dt}(t) = -\dfrac{(1-t)^2}{2} \\ \dfrac{df_1}{dt}(t) = \dfrac{-2(1-t)t + (1-t)^2 - 1}{2} \\ \dfrac{df_2}{dt}(t) = \dfrac{-t^2 + 2(1-t)t + 1}{2} \\ \dfrac{df_3}{dt}(t) = \dfrac{t^2}{2} \end{cases}$$

$$M = \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix}$$



# matrix.h

```
template <class CElement>
class CMatrix
{
            friend CMatrix<CVector3D> operator * (const CMatrix<double> &R, const CMatrix<CVector3D> &V);
            friend CMatrix<CVector3D> operator * (const CMatrix<CVector3D> &V, const CMatrix<double> &R);
private:
            unsigned int row,column;
            CElement **e;
public:
            CMatrix(unsigned row=4,unsigned column=4);
            CMatrix(const CMatrix<CElement> &M);

            CMatrix<CElement>&      operator = (const CMatrix<CElement> &M);
            CMatrix<CElement>       operator + (const CMatrix<CElement> &M);
            CMatrix<CElement>       operator - (const CMatrix<CElement> &M);
            CMatrix<double>         operator * (const CMatrix<CElement> &M);


            void updateElement(const unsigned int &iRow, const unsigned int &iColumn, const CElement &element);

            CElement getElement(const unsigned int &iRow, const unsigned int &iColumn) const;
            unsigned int getRow() const;
            unsigned int getColumn() const;

            ~CMatrix();
};
```

# somepatch.h

```
#include "matrix.h"
#include "meshes.h"

class CBicubicPatch
{
private:
            CMatrix<CVector3D> M;
            unsigned int ordinal;
public:

            CBicubicPatch( const CMatrix<CVector3D> &M, const unsigned int &ordinal);
            CBicubicPatch( const CBicubicBSplinePatch &patch);

            CBicubicPatch& operator = (const CBicubicBSplinePatch &patch);

            void updateControlpoint( const unsigned int &iRow, const unsigned int &iColumn,const CVector3D &v);
            void updateControlnet( const CMatrix<CVector3D> &M);

            CVector3D F( const double &u, const double &v) const;
            CVector3D diffFu( const double &u, const double &v) const;
            CVector3D diffFv( const double &u, const double &v) const;

            CMesh3D generateSurface( const unsigned int &m, const unsigned int &n) const;

            ~CBicubicPatch();
};
```

# somepatch.cpp (részlet)

```
CVector3D CBicubicPatch::F( const double &u, const double &v)
{
            CMatrix < double > B3u(1,4), B3v(4,1);

            double u2=u*u, u3=u2*u, wu =1.0f-u, wu2=wu*wu,wu3=wu2*wu;

            B3u.updateElement(0,0,wu3/6.0f);
            B3u.updateElement(0,1,(3.0f*wu2*u+3.0f*wu+1.0f)/6.0f);
            B3u.updateElement(0,2,(3.0f*wu*u2+3.0f*u+1.0f)/6.0f);
            B3u.updateElement(0,3,u3/6.0f);

            double v2=v*v,v3=v2*v, wv=1.0f-v, wv2=wv*wv, wv3=wv2*wv;

            B3v.updateElement(0,0,wv3/6.0f);
            B3v.updateElement(1,0,(3.0f*wv2*v+3.0f*wv+1.0f)/6.0f);
            B3v.updateElement(2,0,(3.0f*vw*t2+3.0f*v+1.0f)/6.0f);
            B3v.updateElement(3,0,v3/6.0f);

            return (B3u*M*B3v).getElement(0,0);
}
```

# somepatch.cpp (részlet, folytatás)

```
CVector3D CBicubicBSplinePatch::diffFu( const double &u, const double &v)
{
            CMatrix< double > dB3u(1,4), B3v(4,1);

            double u2=u*u, wu=1.0f-u, wu2=wu*wu;
            dB3u.updateElement(0,0,-wu2/2.0f);
            dB3u.updateElement(0,1,(-2.0f*wu*u+wu2-1.0f)/2.0f);
            dB3u.updateElement(0,2,(-u2+2.0f*wu*u+1.0f)/2.0f);
            dB3u.updateElement(0,3,u2/2.0f);

            double v2=v*v, v3=v2*v, wv=1.0f-v, wv2=wv*wv, wv3=wv2*wv;
            B3v.updateElement(0,0,wv3/6.0f);
            B3v.updateElement(1,0,(3*wv2*v+3*wv+1)/6.0f);
            B3v.updateElement(2,0,(3*wv*v2+3*v+1)/6.0f);
            B3v.updateElement(3,0,v3/6.0f);

            return (dB3u*M*B3v).getElement(0,0);
}

CVector3D CBicubicBSplinePatch::diffFv( const double &u, const double &v)
{
            CMatrix< double > B3u(1,4), dB3v(4,1);

            double u2=u*u, u3=u2*u, wu =1.0f-u, wu2=wu*wu,wu3=wu2*wu;
            B3u.updateElement(0,0,wu3/6.0f);
            B3u.updateElement(0,1,(3.0f*wu2*u+3.0f*wu+1.0f)/6.0f);
            B3u.updateElement(0,2,(3.0f*wu*u2+3.0f*u+1.0f)/6.0f);
            B3u.updateElement(0,3,u3/6.0f);

            double v2=v*v, wv=1.0f-v, wv2=wv*wv;
            dB3v.updateElement(0,0,-wv2/2.0f);
            dB3v.updateElement(1,0,(-2.0f*wv*v+wv2-1.0f)/2.0f);
            dB3v.updateElement(2,0,(-v2+2.0f*wv*v+1.0f)/2.0f);
            dB3v.updateElement(3,0,v2/2.0f);

            return (B3u*M*dB3v).getElement(0,0);
}
```

# somepatch.cpp (részlet, folytatás)

```
CMesh3D CBicubicPatch::generateSurface( const unsigned int &m, const unsigned int &n)
{
            CMesh3D result(m*n);
            double du=1.0f/m,dv=1.0f/n;
            double u[4],v[4];
            int act=0;
            for ( unsigned int i=0;i<m;i++)
            {
                        u[0]=u[3]=i*du;
                        u[1]=u[2]=u[0]+du;
                        for (unsigned int j=0;j<n;j++)
                        {
                                    v[0]=v[1]=j*dv;
                                    v[2]=v[3]=v[0]+dv;
                                    for ( unsigned int k=0;k<4;k++)
                                    {
                                           result.updateCorner(act,k,F(u[k],v[k]));
                                           result.updateNormal(act,k,diffFu(u[k],v[k])%diffFv(u[k],v[k]));
                                    }
                                    act++;
                        }
            }
            return result;
}
```