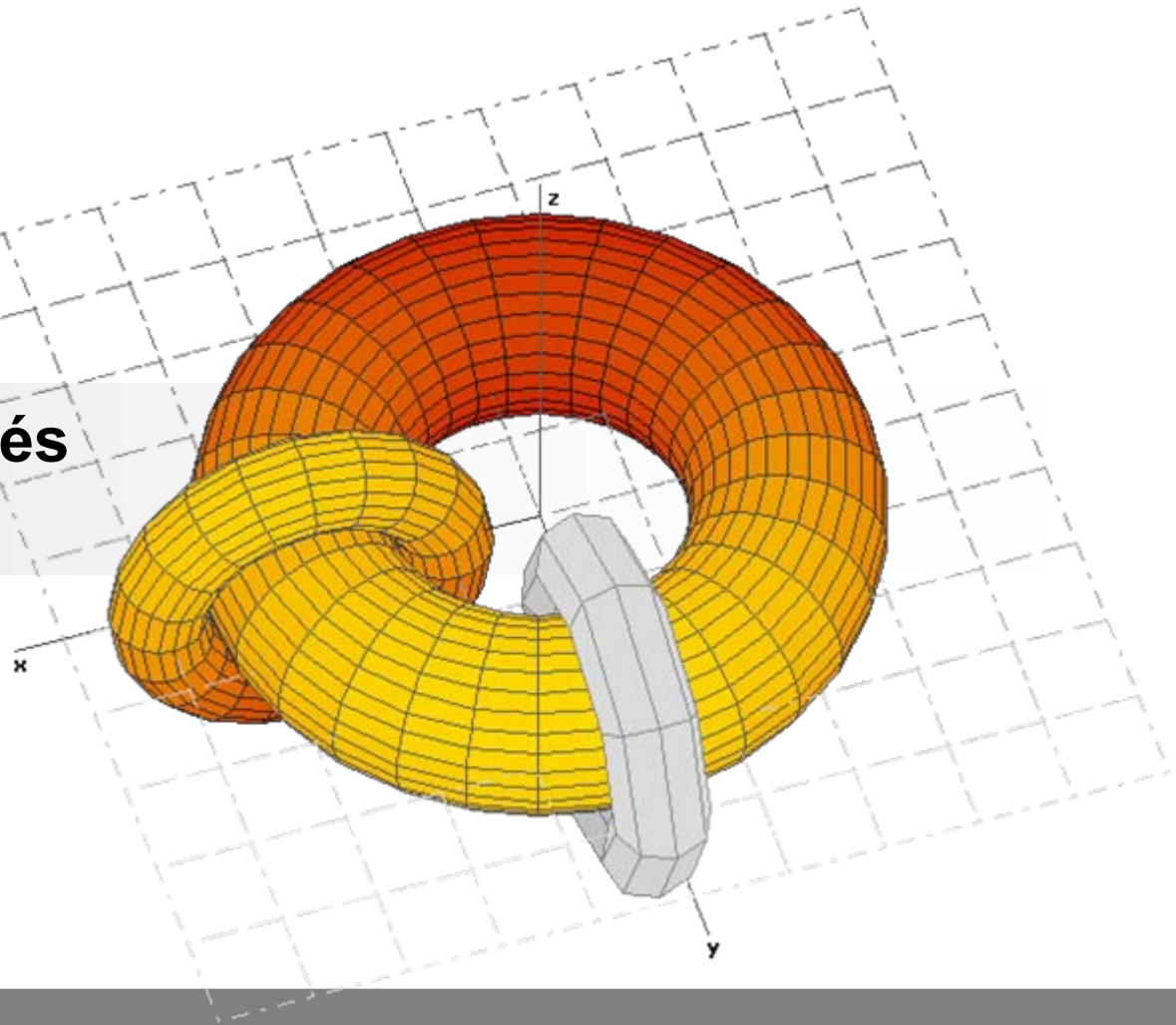
Farkas Gyula Szakkollégium

Bit- és számtologatók



Parametrikus görbék és felületek ábrázolása

2006. március 8., 22.

Róth Ágoston

vectors2D.h

```
class CSquare2D; // később jelenik meg a leírása
class CMesh2D;   // később jelenik meg a leírása

class CVector2D
{
            friend CSquare2D;
            friend CMesh2D;
            friend CVector2D operator * (const CVector2D &v, float lambda);
            friend CVector2D operator * (float lambda, const CVector2D &v);
private:
            float x, y;
public:
                        CVector2D(float x=0.0f,float y=0.0f);

            // módosító metódusok
            void        setX(float x);
            void        setY(float y);
            void        updateAll(float x, float y);

            // néhány operátor túlterhelése
            CVector2D   operator +  (const CVector2D &v);
            CVector2D&  operator += (const CVector2D &v);
            CVector2D   operator -  (const CVector2D &v);
            CVector2D&  operator -= (const CVector2D &v);
            float       operator *  (const CVector2D& v);
            CVector2D&  operator *= (float lambda);

            // vektort skáláz
            void        scale(float lambdax, float lambday);

            // vektort egységnyi hosszúságúra alakít
            void        normalize();

            // lekérdező metódusok
            float       getX() const;
            float       getY() const;
            float       norm() const;
};
```

vectors3D.h

```
class CSquare3D; // később jelenik meg a leírása
class CMesh3D;   // később jelenik meg a leírása
class CVector3D
{
            friend CSquare3D;
            friend CMesh3D;
            friend CVector3D operator * (const CVector3D &v, float lambda);
            friend CVector3D operator * (float lambda, const CVector3D &v);
private:
            float x, y, z;
public:
                        CVector3D(float x=0.0f, float y=0.0f, float z=0.0f);
            void        setX(float x);
            void        setY(float y);
            void        setZ(float z);
            void        updateAll(float x, float y, float z);

            CVector3D   operator +  (const CVector3D &v);
            CVector3D&  operator += (const CVector3D &v);
            CVector3D   operator -  (const CVector3D &v);
            float       operator *  (const CVector3D &v1, const CVector3D &v2);
            CVector3D&  operator *= (float lambda);
            CVector3D&  operator -= (const CVector3D &v);
            CVector3D   operator %  (const CVector3D &v); // vektoriális szorzat
            CVector3D&  operator %= (const CVector3D &v); // vektoriális szorzat

            void        scale(float lambdax, float lambday, float lambdaz);
            void        normalize();

            float       getX() const;
            float       getY() const;
            float       getZ() const;
            float       norm() const;

            CVector2D   orthoProjection(const CVector3D &lightDir);     // merőleges vetítés
            CVector2D   perspProjection(float eyeDistance);             // perspektívikus vetítés
            void        rotateLeft(float alfa);                         // forgatási transzformáció
            void        rotateRight(float alfa);                        // forgatási transzformáció
            bool        rotateUp(float alfa);                           // forgatási transzformáció
            bool        rotateDown(float alfa);                         // forgatási transzformáció
};
```

Nézőpont megváltoztatása

Balra

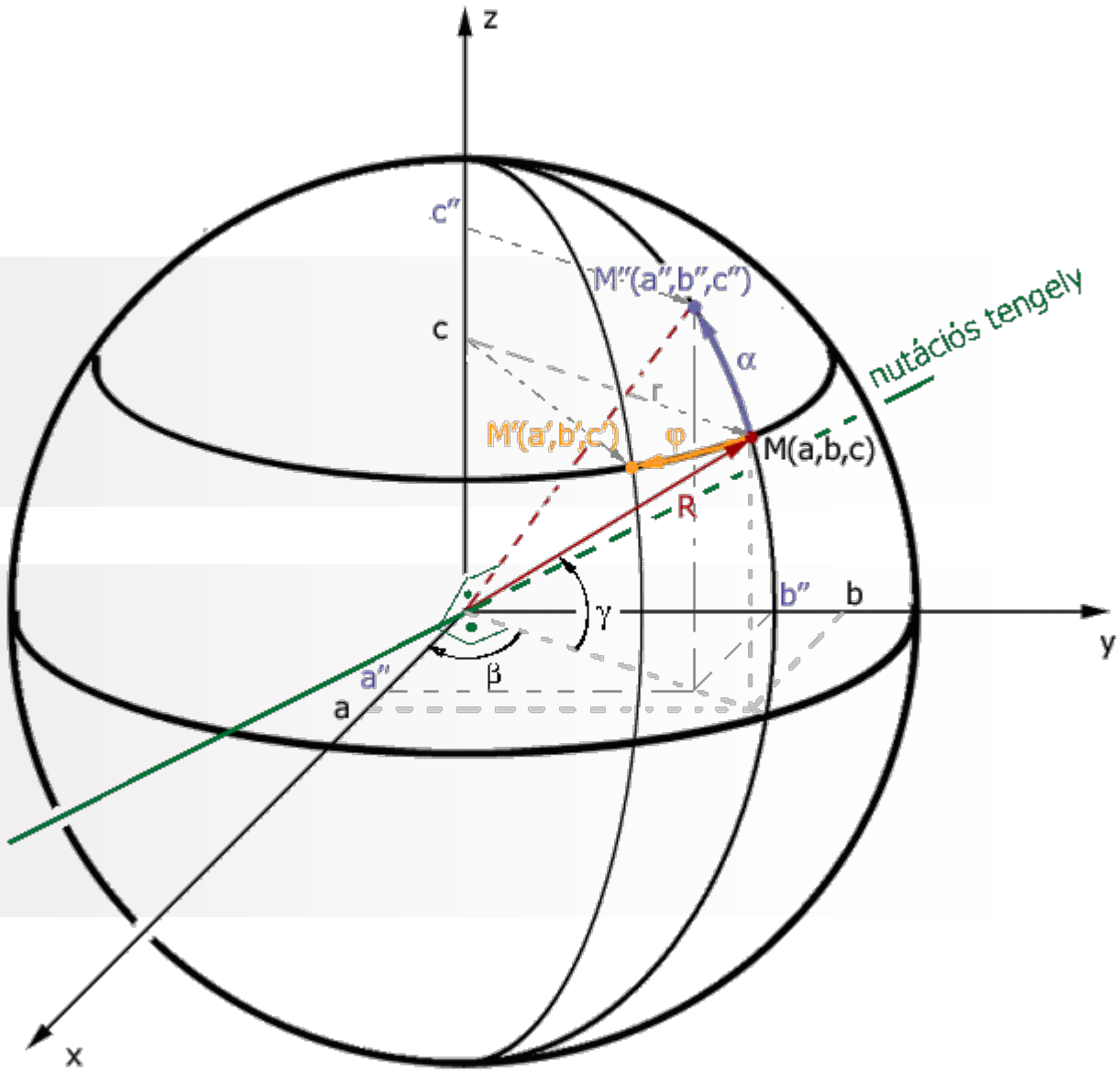
$$\begin{cases} a' = a\cos\varphi - b\sin\varphi \\ b' = a\sin\varphi + b\cos\varphi \\ c' = c \end{cases}$$

Jobbra: $\varphi \leftrightarrow -\varphi$

Felfelé

$$\begin{cases} a'' = a\left(\cos\alpha - \frac{c}{r}\sin\alpha\right) \\ b'' = b\left(\cos\alpha - \frac{c}{r}\sin\alpha\right) \\ c'' = c\cos\alpha + r\sin\alpha \end{cases}$$

Lefelé: $\alpha \leftrightarrow -\alpha$



Merőleges vetítés

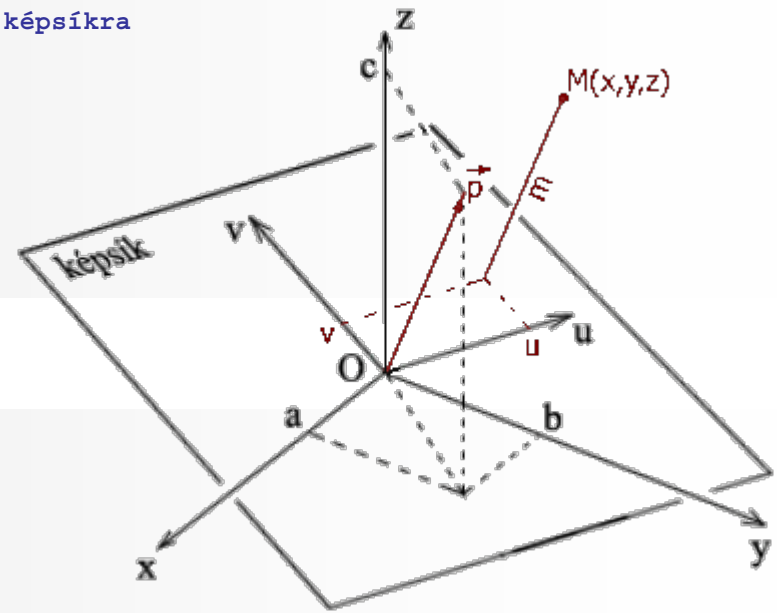
Egységnyi vetítő vektor, mely merőleges az Origón átmenő képsíkra

$\vec{p}(a,b,c) \in R^3$

$\|\vec{p}\| = a^2 + b^2 + c^2 = 1$

$a^2 + b^2 \neq 0$

- az összes többi fénysugár párhuzamos $\vec{p}$-vel



- képsíkbeli koordináták:

$\forall M(x,y,z) \in R^3$

$$\begin{cases} u = \dfrac{-bx + ay}{\sqrt{a^2 + b^2}} \\ v = \dfrac{-c(ax + by) + (a^2 + b^2)z}{\sqrt{a^2 + b^2}} \end{cases}$$

- képsíktól való távolság: $m = ax + by + cz = \vec{p} \cdot \overrightarrow{OM}$

Perspektívikus vetítés

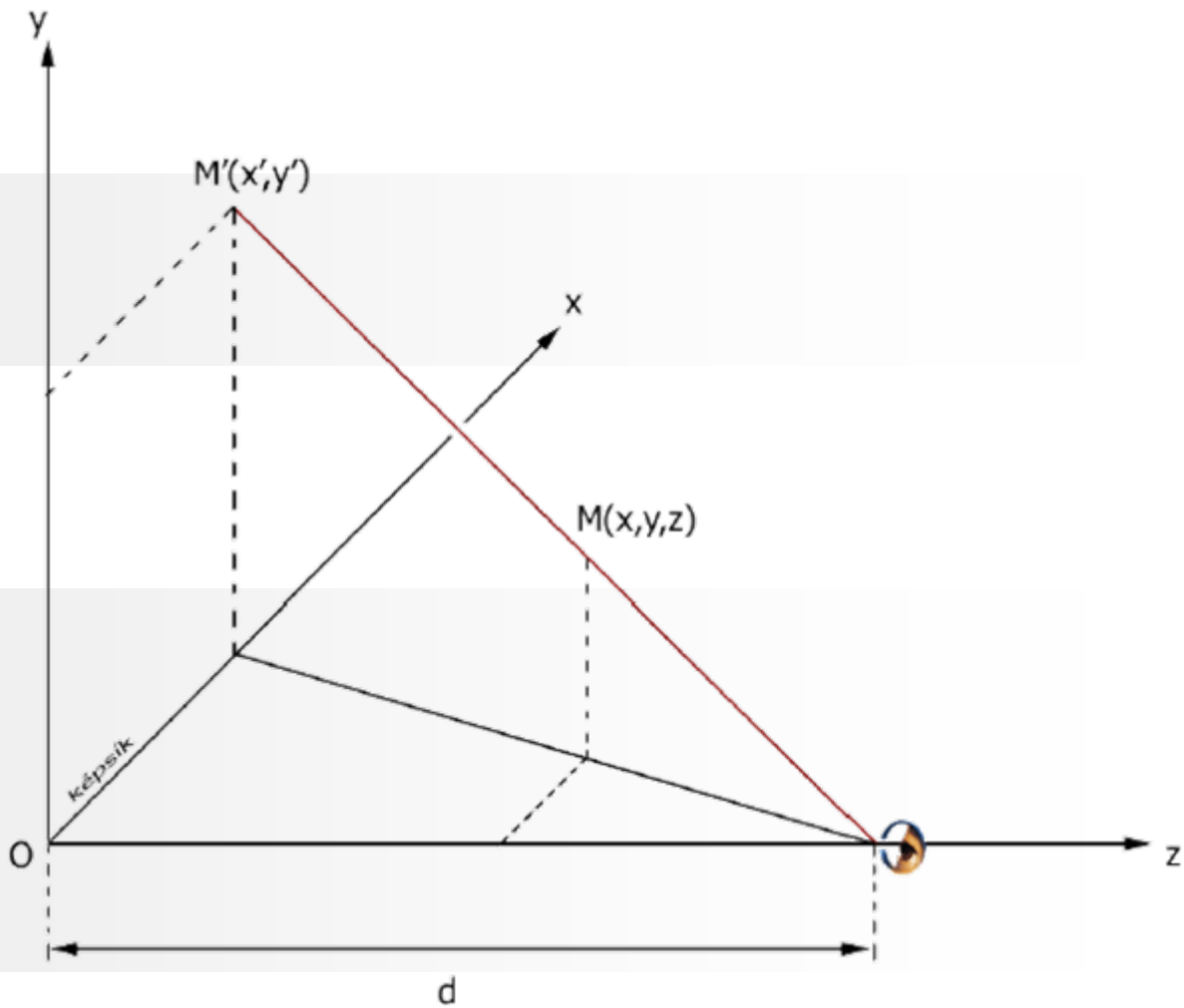
- **képsík:** xOy
- **megfigyelő:** Oz **tengelyen**
- **szemtávolság:** d

- **képsíkbeli koordináták:**

$$\forall M(x,y,z) \in R^3$$

$$\begin{cases} x' = \dfrac{xz}{d-z} \\ y' = \dfrac{yz}{d-z} \end{cases}$$

- **képsíktól való távolság:** z



Parametrikus felületek. Négyszögű háló

- **felület paraméterezése:**

$$F:[a,b]\times[c,d]\to R^3$$
$$F(u,v)=(fx(u,v),fy(u,v),fz(u,v)),(u,v)\in[a,b]\times[c,d]$$

- **értelmezés tartomámybeli osztópontok:**

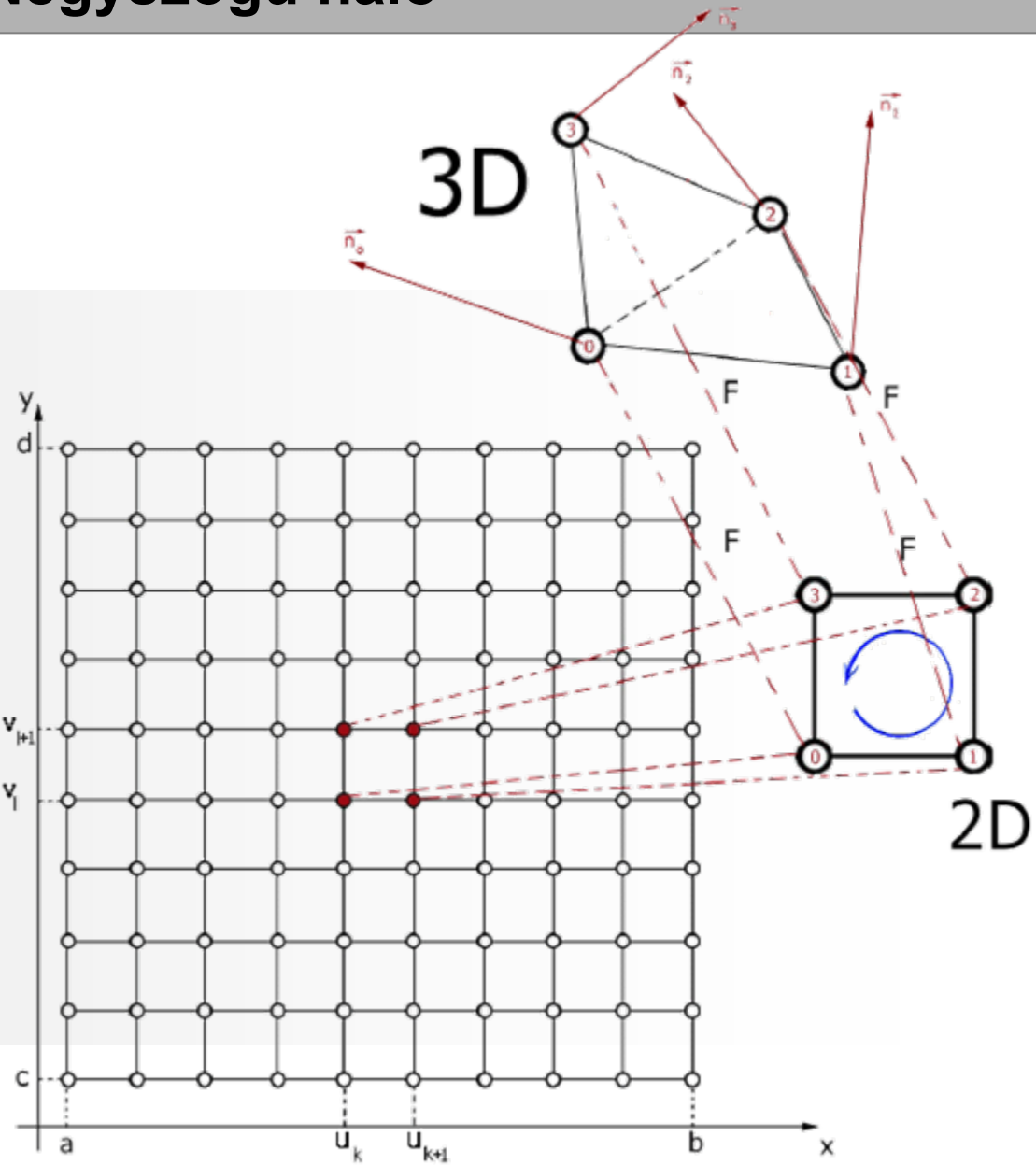
$$du=\frac{b-a}{m},dv=\frac{d-c}{n}$$
$$u_k=a+k\cdot du,k=0,1,\ldots,m$$
$$v_l=c+l\cdot dv,l=0,1,\ldots,n$$

- **2D-os rácshálót alkotó negyszögek:**

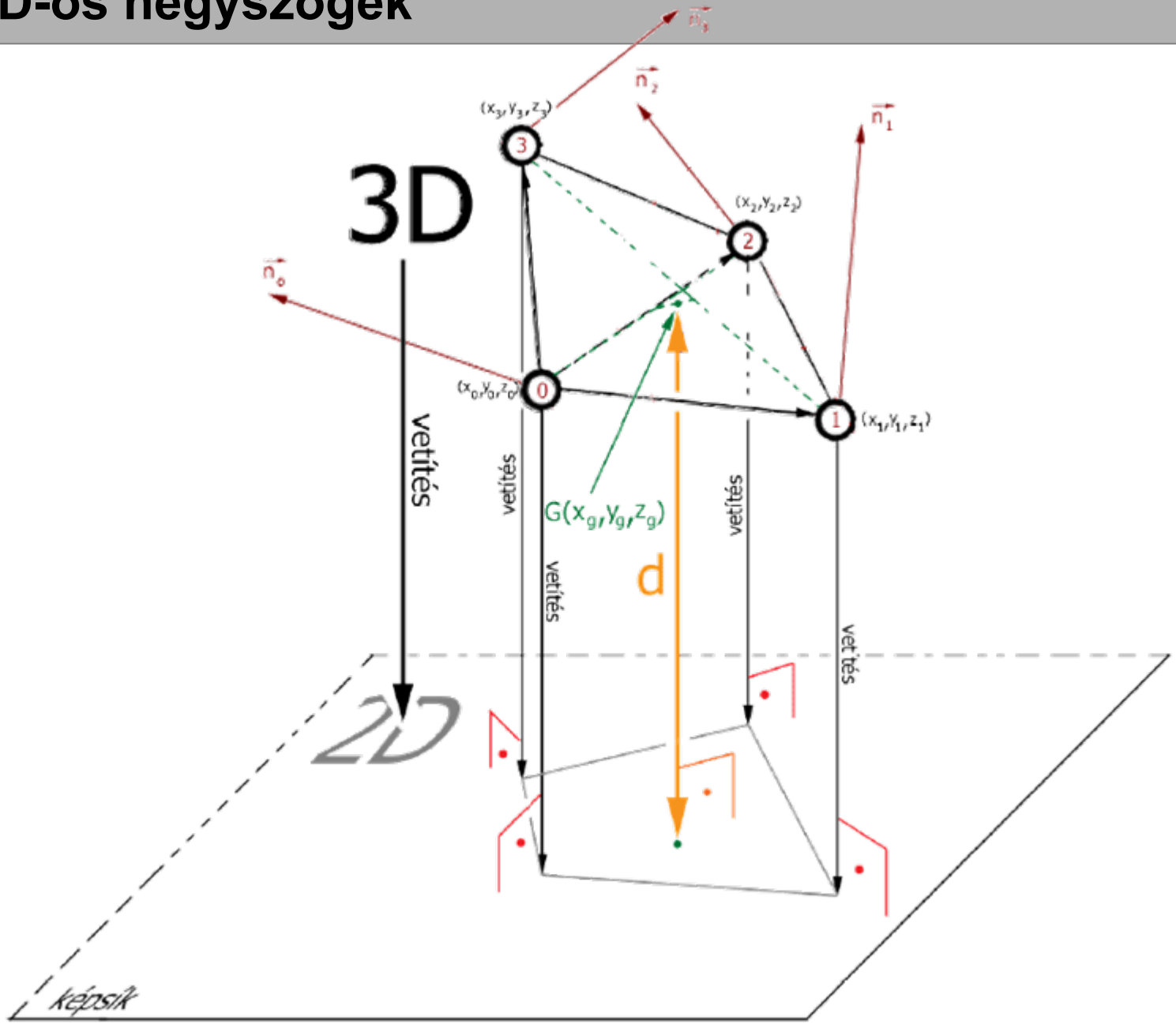
$$[(u_k,v_l),(u_{k+1},v_l),(u_{k+1},v_{l+1}),(u_k,v_l)]$$
$$k=0,1,\ldots,m-1$$
$$l=0,1,\ldots,n-1$$

- **3D-os rácshálót alkotó negyszögek:**

$$[F(u_k,v_l),F(u_{k+1},v_l),F(u_{k+1},v_{l+1}),F(u_k,v_l)]$$
$$k=0,1,\ldots,m-1$$
$$l=0,1,\ldots,n-1$$



2D/3D-os négyszögek



squares2D.h

```
class CSquare3D;
class CMesh2D;

class CSquare2D
{
            friend CMesh2D;
            friend CSquare3D;
private:
            CVector2D   *corner;
            bool        visible;
            bool        transparent;
            TColor      transparentColor;
public:
            // alapértelmezett konstruktor
            CSquare2D();

            // másoló konstruktor
            CSquare2D(const CSquare2D &s);

            // hozzárendelő operátor
            CSquare2D& operator = (const CSquare2D &s);

            //módosító metódusok
            void updateCorner(int iCorner, float x, float y);
            void updateCorner(int iCorner, const CVector2D& v);

            //destruktor
            ~CSquare2D();
};
```

squares3D.h

```
class CMesh3D;

class CSquare3D
{
            friend CMesh3D;
private:
            CVector3D   *corner;
            CVector3D   *normal;
            float       d;
            bool        transparent;
            TColor      transparentColor;
public:
                        CSquare3D();
                        CSquare3D(const CSquare3D &s);
            CSquare3D&  operator = (const CSquare3D &s);

            float       getDistance() const;

            void        updateCorner(int iCorner, float x, float y, float z);
            void        updateCorner(int iCorner, const CVector3D& v);

            void        updateNormal(int iCorner, float x, float y, float z);
            void        updateNormal(int iCorner, const CVector3D& n);

            CSquare2D   orthoProjection(const CVector3D &lightDir);

            ~CSquare3D();
};

//a qsort függvényhez szükséges egy összehasonlító függvény (a d alapján rendezzük a szemeket növekvő sorrendbe)
int compare3DSquares(const void * s1, const void * s2);
```

mesh2D.h

```
class CMesh3D;

class CMesh2D
{
            friend CMesh3D;
private:
            int nrMesh;
            CSquare2D *mesh;
public:

                        CMesh2D(int nrMesh=0);
                        CMesh2D(const CMesh2D &m2D);
            CMesh2D&    operator = (const CMesh2D &m2D);

            void        updateCorner(int iMesh, int jCorner, float x, float y);
            void        updateCorner(int iMesh, int jCorner, const CVector2D& v);

            void        setVisibility(int iMesh, bool visible);
            void        setTransparency(int iMesh, bool transparent);

            void        Draw(TCanvas *canvas, int x0, int y0, float zoom,
                             const TColor &backColor1, const TColor &backColor2,
                             const TColor &frontColor1, const TColor &frontColor2);

            virtual     ~CMesh2D(); // fontos, hogy virtuális legyen, mert származtatni fogunk ebből az osztályból
};
```

mesh2D.cpp – részlet

```
void CMesh2D::Draw(TCanvas *canvas, int x0, int y0, float zoom,
                   const TColor &backColor1, const TColor &backColor2,
                   const TColor &frontColor1, const TColor &frontColor2)
{
            int br1=GetRValue(backColor1),bg1=GetGValue(backColor1),bb1=GetBValue(backColor1),
                br2=GetRValue(backColor2),bg2=GetGValue(backColor2),bb2=GetBValue(backColor2),
                fr1=GetRValue(frontColor1),fg1=GetGValue(frontColor1),fb1=GetBValue(frontColor1),
                fr2=GetRValue(frontColor2),fg2=GetGValue(frontColor2),fb2=GetBValue(frontColor2);
            float dbr=(float)(br2-br1)/(float)(nrMesh-1),
                  dbg=(float)(bg2-bg1)/(float)(nrMesh-1),
                  dbb=(float)(bb2-bb1)/(float)(nrMesh-1),
                  dfr=(float)(fr2-fr1)/(float)(nrMesh-1),
                  dfg=(float)(fg2-fg1)/(float)(nrMesh-1),
                  dfb=(float)(fb2-fb1)/(float)(nrMesh-1);

            TPoint *p;
            p= new TPoint[4];
            for (int i=0;i<nrMesh;i++)
            {
              for (int j=0;j<4;j++)
              {
                p[j].x=floor((float)x0+(float)zoom*mesh[i].corner[j].x);
                p[j].y=floor((float)y0-(float)zoom*mesh[i].corner[j].y);
              }
              if (mesh[i].visible)
              {
                if (mesh[i].transparent)
                {
                  canvas->Pen->Style=psDashDot;
                  canvas->Pen->Color=mesh[i].transparentColor;
                  canvas->Brush->Style=bsClear;
                }
                else
                {
                  canvas->Pen->Style=psSolid;
                  canvas->Pen->Color=RGB(floor(fr1+i*dfr)-75,floor(fg1+i*dfg)-75,floor(fb1+i*dfb)-75);
                  canvas->Brush->Color=RGB(floor(fr1+i*dfr),floor(fg1+i*dfg),floor(fb1+i*dfb));
                }
                canvas->Polygon(p,3);
              }
```

mesh2D.cpp – részlet (folytatás)

```
            else //if (!mesh[i].visible)
            {
               if (mesh[i].transparent)
               {
                  canvas->Pen->Style=psDashDot;
                  canvas->Pen->Color=mesh[i].transparentColor;
                  canvas->Brush->Style=bsClear;
               }
               else
               {
                  canvas->Pen->Style=psSolid;
                  canvas->Pen->Color=RGB(floor(i*100.0f/(nrMesh-1)),
                                         floor(i*100.0f/(nrMesh-1)),
                                         floor(i*100.0f/(nrMesh-1)));
                  canvas->Brush->Color=RGB(floor(br1+i*dbr),floor(bg1+i*dbg),floor(bb1+i*dbb));
                }
                canvas->Polygon(p,3);
            }
         }
         delete[] p;
}//for
```

mesh3D.h

```
class CMesh3D
{
private:
            int nrMesh;
            CSquare3D *mesh;
public:
                        CMesh3D(int nrMesh=0);
                        CMesh3D(const CMesh3D &m3D);
            CMesh3D&    operator = (const CMesh3D &m3D);

            CMesh3D     operator + (const CMesh3D &m3D);

            void        updateCorner(int iMesh, int jCorner,
                                     float x, float y, float z);
            void        updateCorner(int iMesh, int jCorner, const CVector3D& v);

            void        updateNormal(int iMesh, int jCorner,
                                     float x, float y, float z);
            void        updateNormal(int iMesh, int jCorner, const CVector3D& n);

            void        makeTransparent(TColor transparentColor);

            CMesh2D     orthoProjection(const CVector3D &lightDir);

            virtual     ~CMesh3D(); // fontos, hogy virtuális legyen, mert származtatni fogunk ebből az osztályból
};
```

mesh3D.cpp – részlet

```
CMesh2D CMesh3D::orthoProjection(const CVector3D &lightDir)
{
            CMesh2D     result(nrMesh);
            CVector3D G,N;

            // súlypontok távolsága a képsíktól
            for(int i=0;i<nrMesh;i++)
            {
                        G.updateAll(0.0f,0.0f,0.0f);
                        for(int j=0;j<4;j++) G+=mesh[i].corner[j];
                        mesh[i].d=G*lightDir;
            }

            // szemek növekvő sorrendbe rendezése a kapott távolságértékek alapján
            qsort(mesh,nrMesh,sizeof(CSquare3D),compare3DSquares);

            // szemek vetítése
            for(int i=0;i<nrMesh;i++)
            {
                        N.updateAll(0.0f,0.0f,0.0f);
                        for(int j=0;j<4;j++)
                                    N+=mesh[i].normal[j];
                        result.mesh[i]=mesh[i].orthoProjection(lightDir);

                        // láthatóság eldöntése
                        // a szemek rendezése nélkül, a láthatóság eldöntése nem ad helyes eredményt
                        if (N*lightDir>0.0f)
                                    result.setVisibility(i,true);
                        else
                                    result.setVisibility(i,false);
                        // 'áttetszőség' beállítása
                        result.setTransparency(i,mesh[i].transparent);
            }
            return result;
}
```

parametricsurfaces.h

```
typedef float (*fptr) (float, float);

class CParametricSurface3D: public CMesh3D
{
private:
            float a, b, c, d; // értelmezési tartomány
            fptr  fx, fy, fz; // koordinátafüggvények
            int   m, n;       // (m-1)x(n-1)-es racsháló
public:
            CParametricSurface3D(float a, float b, float c, float d,
                                 fptr fx, fptr fy, fptr fz,
                                 int m, int n);
};
```

parametricsurfaces.cpp

```
CVector3D diffu(fptr fx,fptr fy, fptr fz, float u, float v)
{
            float h=1e-5;
            return CVector3D((fx(u+h,v)-fx(u,v))/h,(fy(u+h,v)-fy(u,v))/h,(fz(u+h,v)-fz(u,v))/h);
}

CVector3D diffv(fptr fx,fptr fy, fptr fz, float u, float v)
{
            float h=1e-5;
            return CVector3D((fx(u,v+h)-fx(u,v))/h,(fy(u,v+h)-fy(u,v))/h,(fz(u,v+h)-fz(u,v))/h);
}

CParametricSurface3D::CParametricSurface3D(float a, float b, float c, float d,
                                           fptr fx,fptr fy,fptr fz,
                                           int m,int n):

CMesh3D(m*n),
a(a),b(b),c(c),d(d),fx(fx),fy(fy),fz(fz),m(m),n(n)
{
            float du=(b-a)/m, dv=(d-c)/n;
            float u[4],v[4];
            int act=0;
            for(unsigned int i=0;i<m;i++)
            {
                        u[0]=u[3]=a+i*du;
                        u[1]=u[2]=u[0]+du;
                        for(unsigned int j=0;j<n;j++)
                        {
                                    v[0]=v[1]=c+j*dv;
                                    v[2]=v[3]=v[0]+dv;
                                    for(unsigned int k=0;k<4;k++)
                                    {
                                          updateCorner(act,k,fx(u[k],v[k]),fy(u[k],v[k]),fz(u[k],v[k]));
                                          updateNormal(act,k,diffu(fx,fy,fz,u[k],v[k])%diffv(fx,fy,fz,u[k],v[k]));
                                    }
                                    act++;
                        }
            }
}
```

Domborzatjellegű (Euler-Monge előállítású) felületek

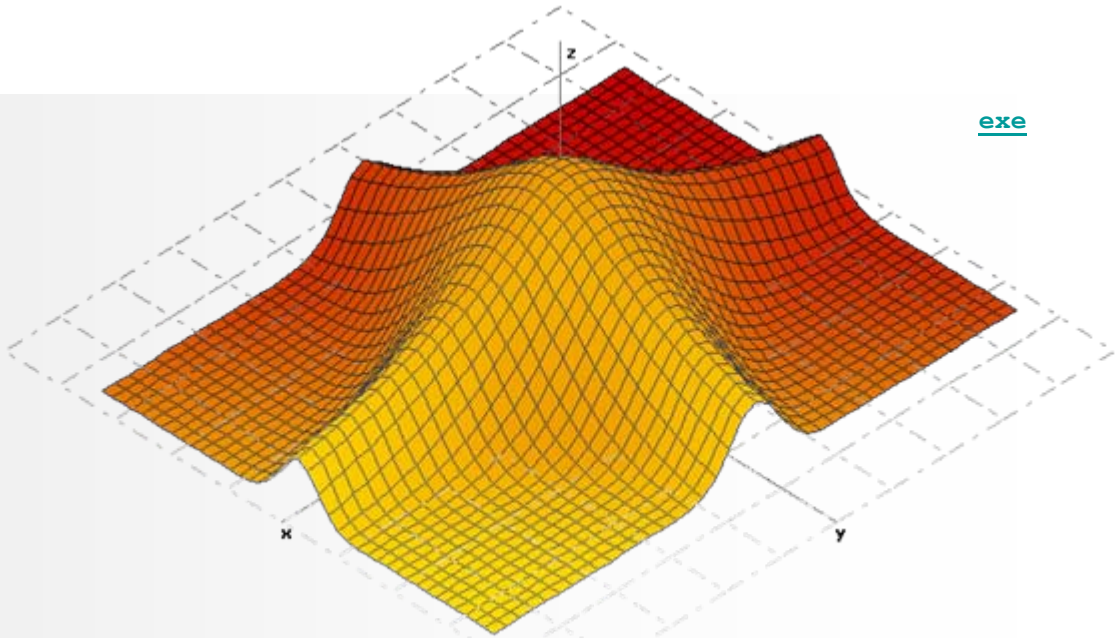
- **felület paraméterezése matematikailag:**

$$\begin{cases} z:[a,b]\times[c,d]\to R \\ z=z(u,v) \end{cases}$$

- **programban:**

```
float EulerMongeX(float u, float v)
{
    return u;
}

float EulerMongeY(float u, float v)
{
    return v;
}

float EulerMongeZ(float u, float v)
{
    return z(u,v);
}

//...
CVector3D lightDir(sqrt(1.0/3.0), sqrt(1.0/3.0), sqrt(1.0,3.0)); // egységnyi vetítősugár
float a,b,c,d; // értelmezési tartomány
unsigned int m=30, n=30; // osztópontok száma u- illetve v-irányban
CParametricSurface3D EulerMonge(a, b, c, d, EulerMongeX, EulerMongeY, EulerMongeZ, m, n);
CMesh2D m2D=EulerMonge.orthoProjection(lightDir);
//...
m2D.Draw(...);
```

exe

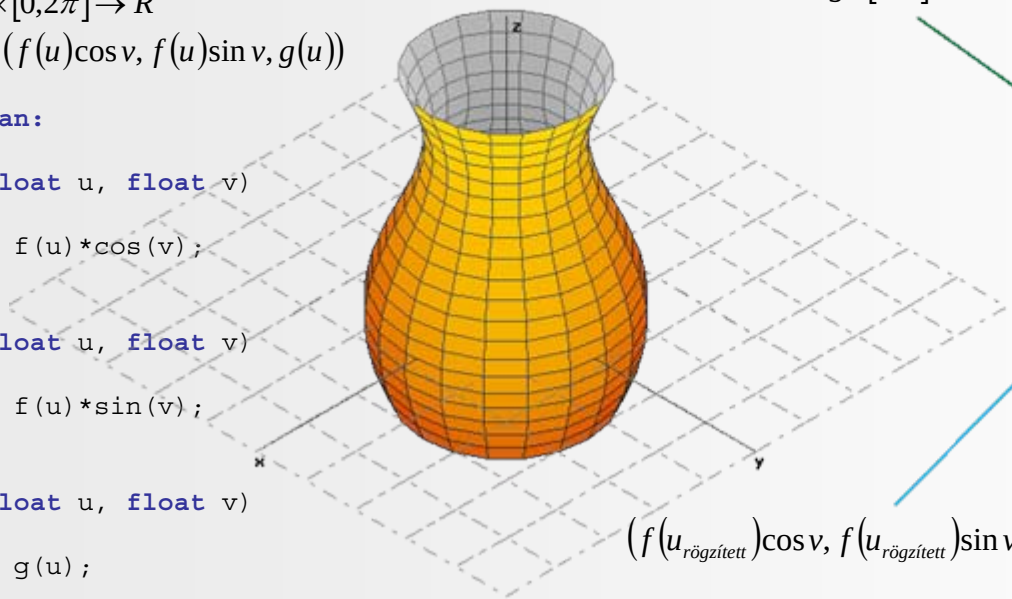


$$\begin{cases} z:[-3.2,3.2]\times[-2.4,2.4]\to R \\ z(u,v)=\dfrac{e^{-u^2v^2}}{\sqrt{u^2+v^2+1}} \end{cases}$$

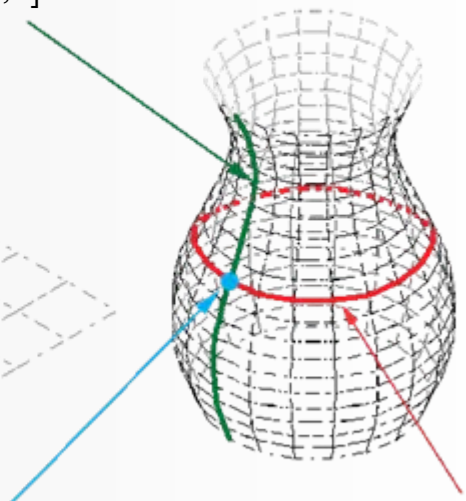
Forgásfelületek

- **felület paraméterezése matematikailag:**

$$\begin{cases} F:[a,b]\times[0,2\pi]\to R^3 \\ F(u,v)=(f(u)\cos v,\ f(u)\sin v,\ g(u)) \end{cases}$$



$$g:[a,b]\to R$$



$$\left(f\left(u_{rögzített}\right)\cos v,\ f\left(u_{rögzített}\right)\sin v,\ g\left(u_{rögzített}\right)\right)$$

$$(fx,\ fy):[0,2\pi]\to R^2$$

$$\begin{cases} fx\left(u_{rögzített},v\right)=f\left(u_{rögzített}\right)\cdot\cos v \\ fy\left(u_{rögzített},v\right)=f\left(u_{rögzített}\right)\cdot\sin v \end{cases}$$

- **programban:**

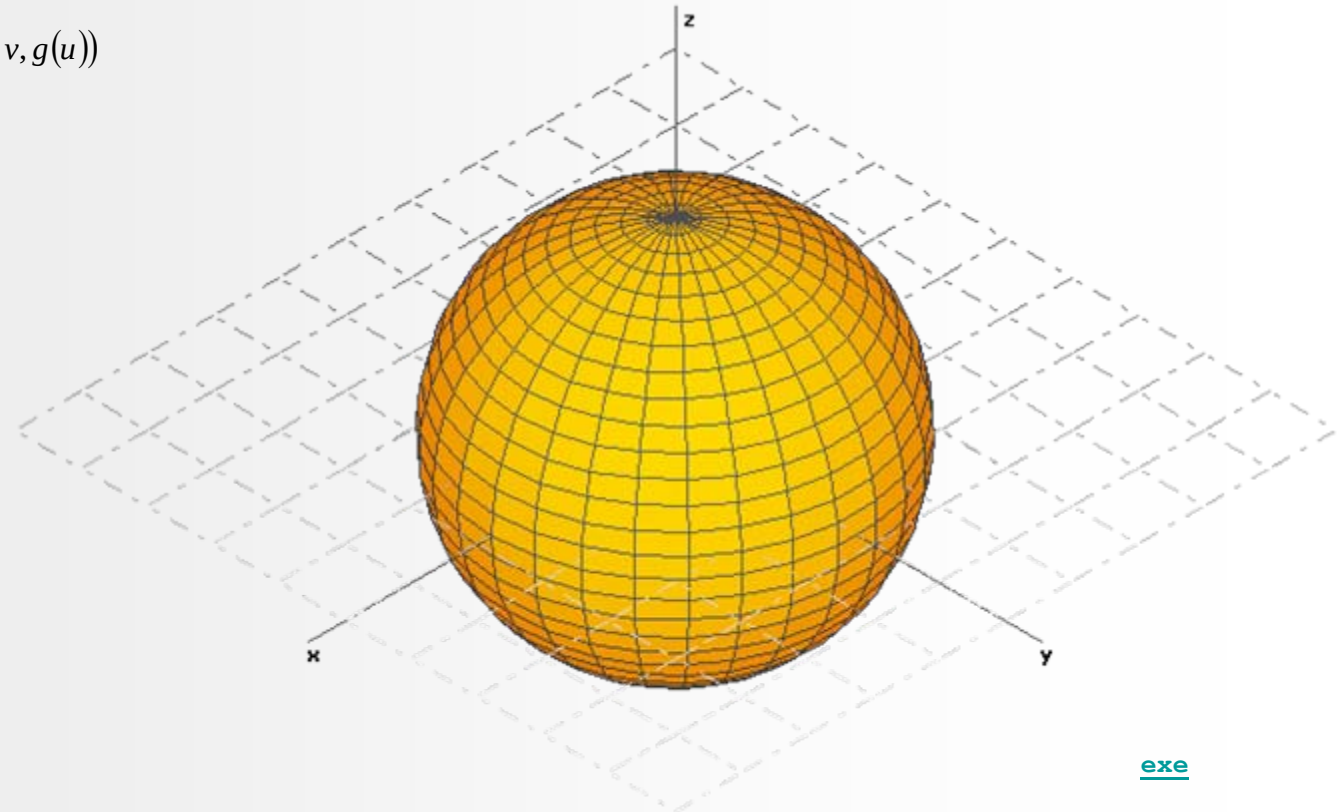
```
float fx(float u, float v)
{
    return f(u)*cos(v);
}

float fy(float u, float v)
{
    return f(u)*sin(v);
}

float fz(float u, float v)
{
    return g(u);
}

//...
CVector3D lightDir(sqrt(1.0/3.0), sqrt(1.0/3.0), sqrt(1.0/3.0)); // egységnyi vetítősugár
float a,b,c=0.0,d=2*pi;  // értelmezési tartomány
unsigned int m=30, n=30; // osztópontok száma u- illetve v-irányban
CParametricSurface3D Forgasfelulet(a, b, c, d, fx, fy, fz, m, n);
CMesh2D m2D=Forgasfelulet.orthoProjection(lightDir);
//...
m2D.Draw(...);
```

Sajátságos forgásfelületek

- Gömb

$$\begin{cases} F:[0,\pi]\times[0,2\pi]\to R^3 \\ F(u,v)=(f(u)\cos v,\ f(u)\sin v,\ g(u)) \end{cases}$$

$$\begin{cases} f(u)=r\sin u \\ g(u)=r\cos u \end{cases}$$



exe

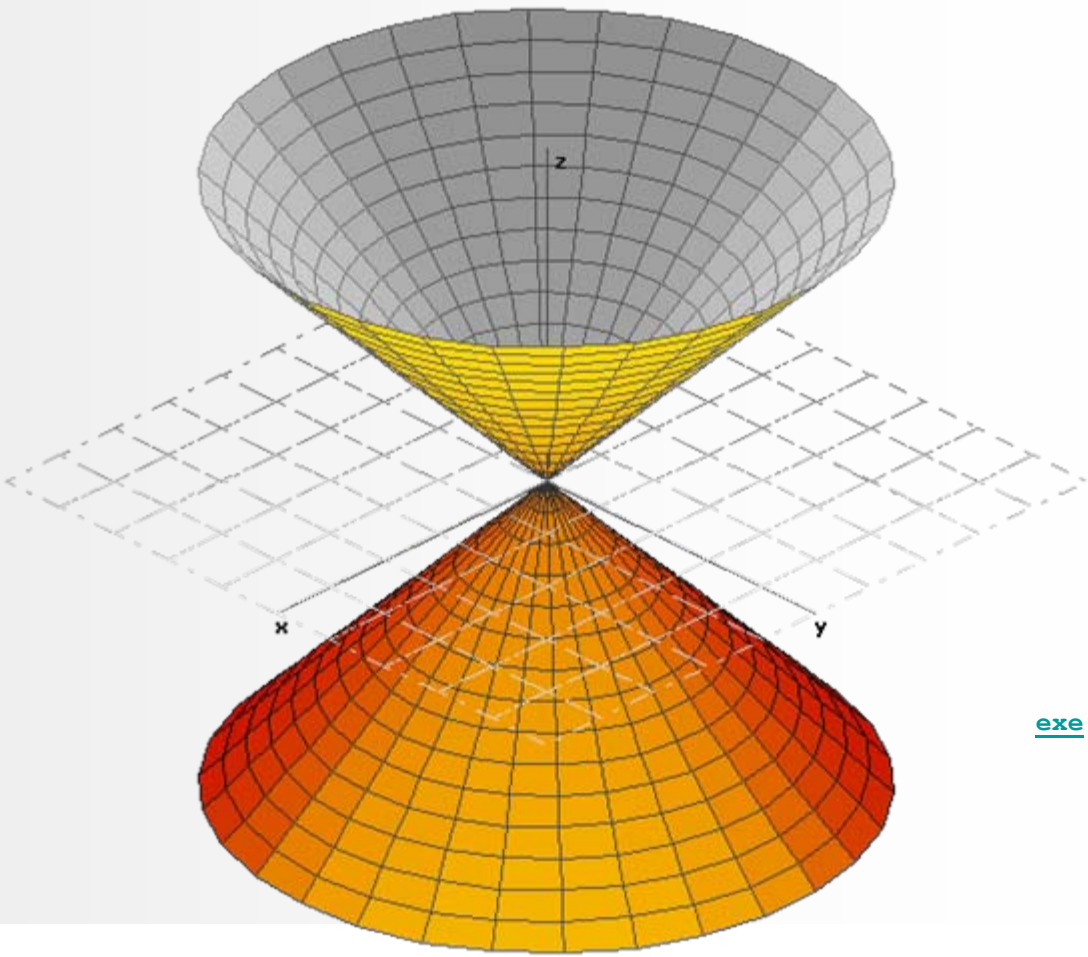
$r=1$

Sajátságos forgásfelületek

- **Kúp**

$$\begin{cases} F:[-c,c]\times[0,2\pi]\to R^3 \\ F(u,v)=(f(u)\cos v,\ f(u)\sin v,\ g(u)) \end{cases}$$

$$\begin{cases} f(u)=u \\ g(u)=\lambda\cdot u \end{cases},\ \lambda\in R^*$$



exe

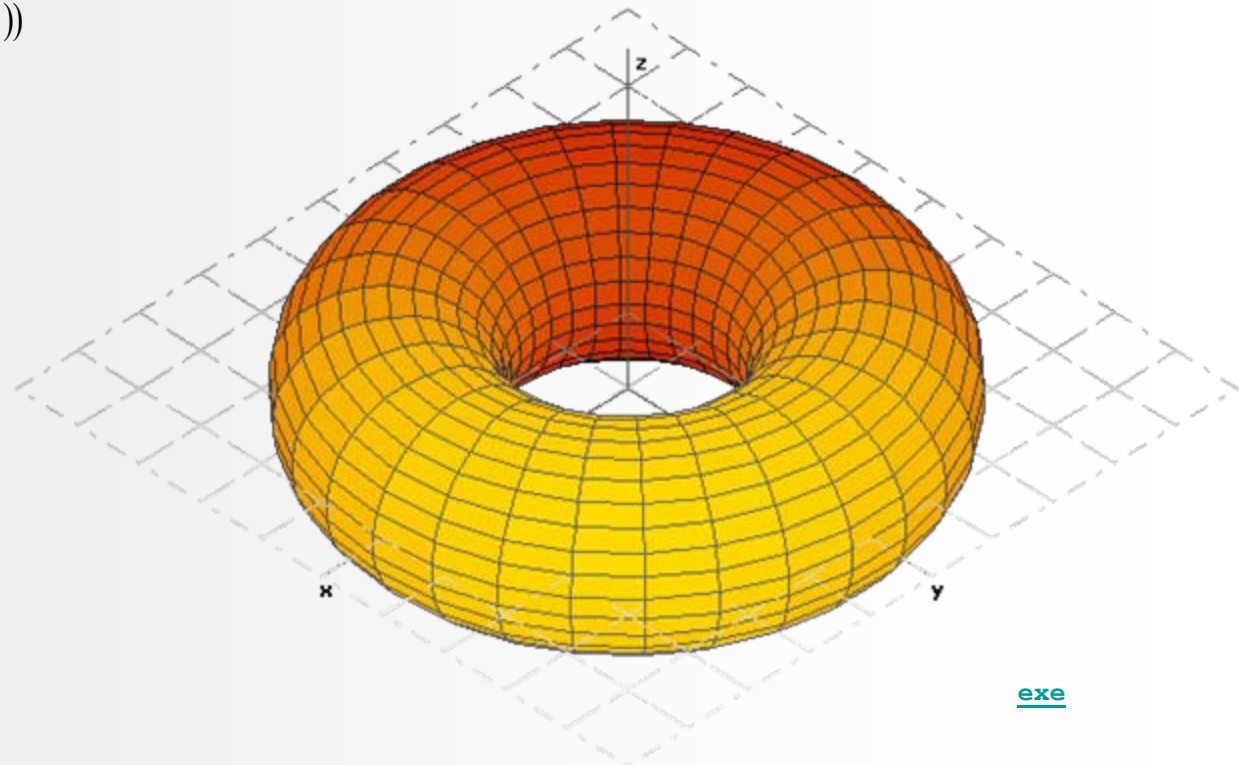
$c=2,\ \lambda=1$

Sajátságos forgásfelületek

- **Tórusz**

$$\begin{cases} F:[0,2\pi]\times[0,2\pi]\to R^3 \\ F(u,v)=(f(u)\cos v,\ f(u)\sin v,\ g(u)) \end{cases}$$

$$\begin{cases} f(u)=R+r\sin u \\ g(u)=r\cos u \end{cases}$$



exe

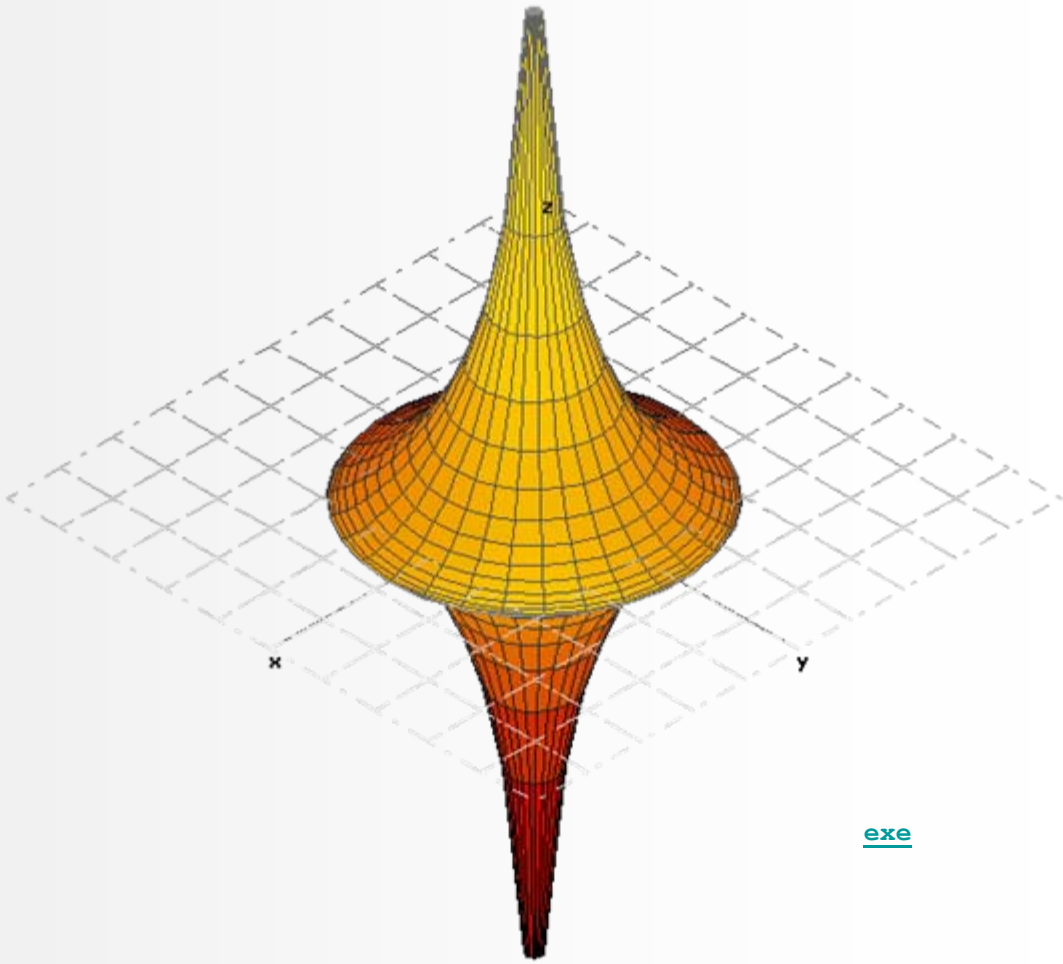
$R=2, r=1$

Sajátságos forgásfelületek

- **Pszeudoszféra**

$$\begin{cases} F:(0,\pi)\times[0,2\pi]\to R^3 \\ F(u,v)=(f(u)\cos v,\ f(u)\sin v,\ g(u)) \end{cases}$$

$$\begin{cases} f(u)=r\sin u \\ g(u)=r\left[\ln\left(\tan\frac{u}{2}\right)+\cos u\right] \end{cases}$$



exe

$r=1$

S hogy miért nem jó az egész elképzelés?

Soha ne higyjetek egy hosszúhajú huligánnak! :)

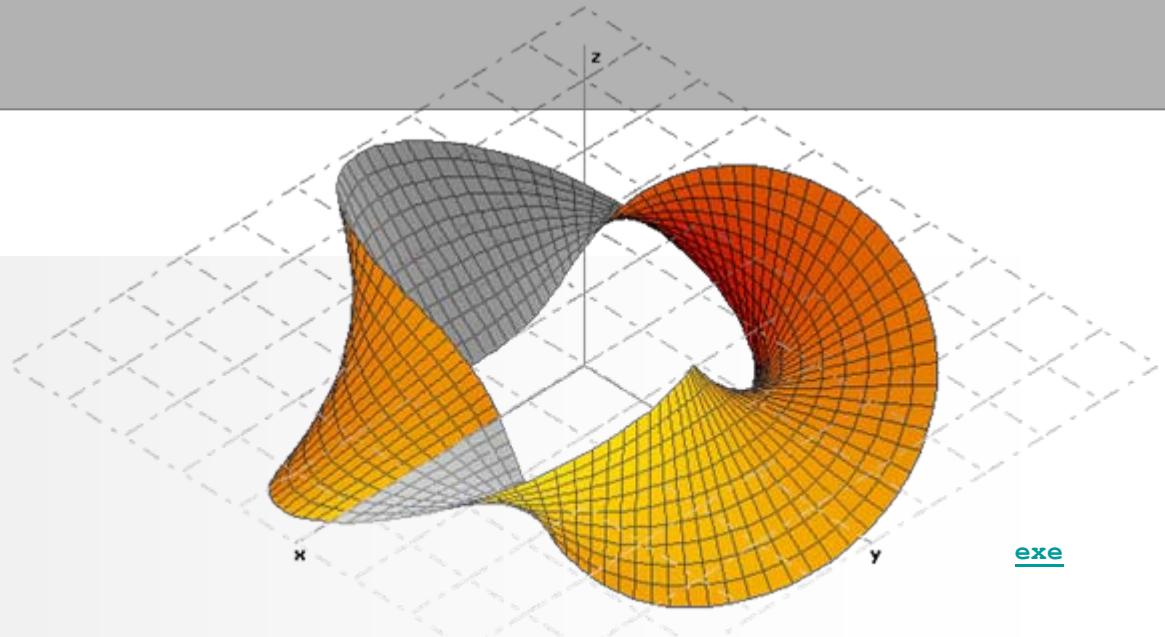
Egyoldalú felületek

- Sajátságosan, Moebius-féle szalagok
- felület paraméterezése matematikailag:

$$\begin{cases} F:[0,1]\times[0,2\pi]\to R^3 \\ F(u,v)=(f(u)\cos v,\ f(u)\sin v,\ g(u)) \end{cases}$$

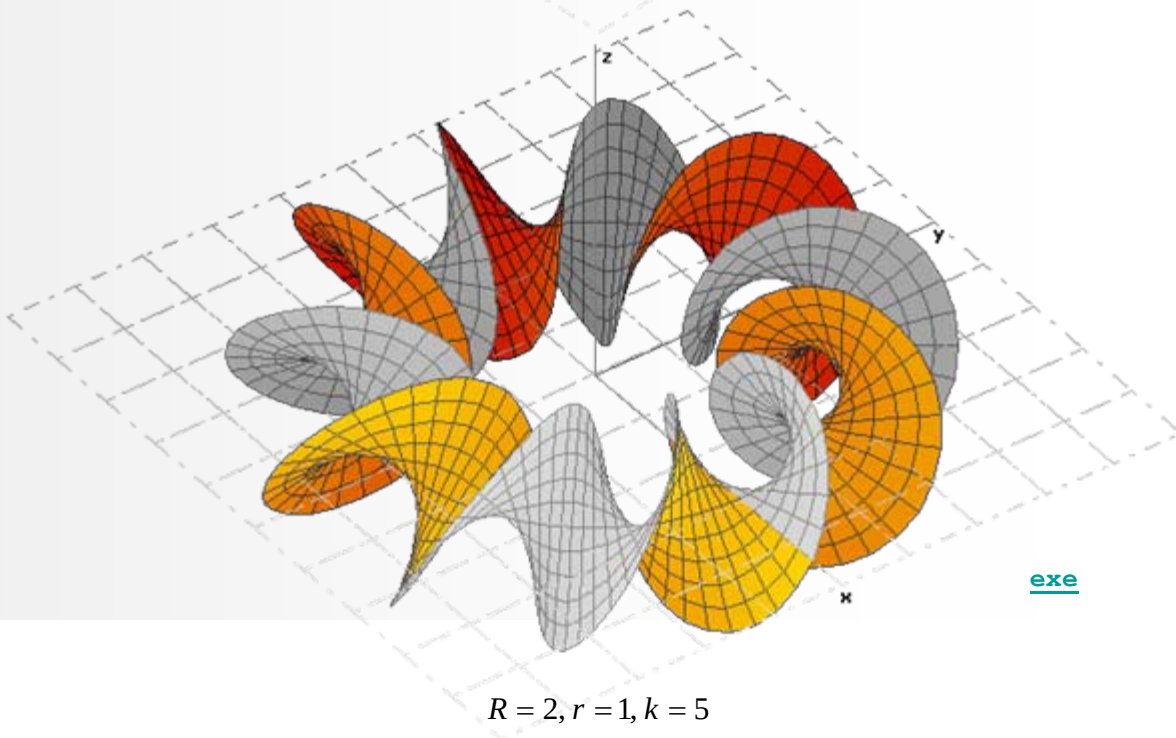
$$\begin{cases} f(u)=R+r(1-2u)\cos\left(\dfrac{2k+1}{2}v\right) \\ g(u)=r(1-2u)\sin\left(\dfrac{2k+1}{2}v\right) \end{cases}$$

k - csavarások száma



exe

$R=2, r=1, k=1$



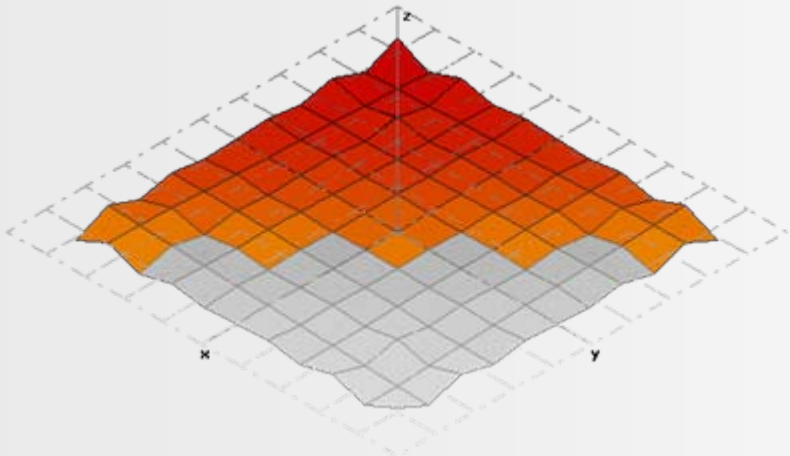
exe

$R=2, r=1, k=5$

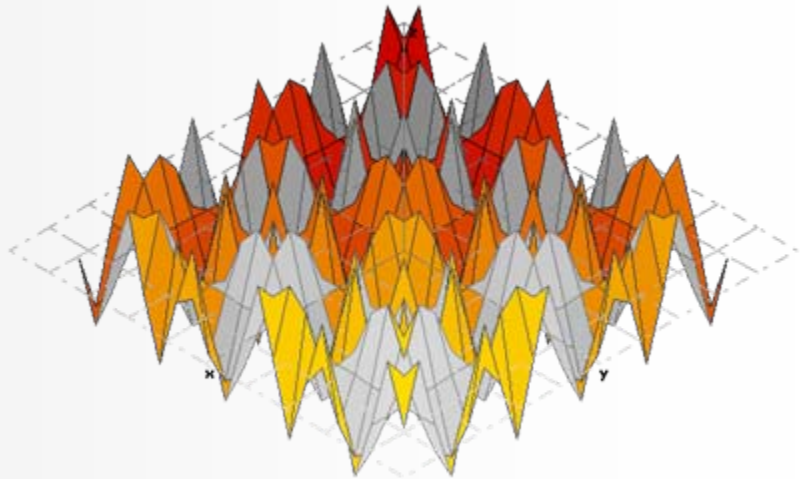
Nem mindig kapunk helyes takarási sorrendet

- **próbáljuk meg ábrázolni a következő felületet különböző felbontású rácshálókat használva**

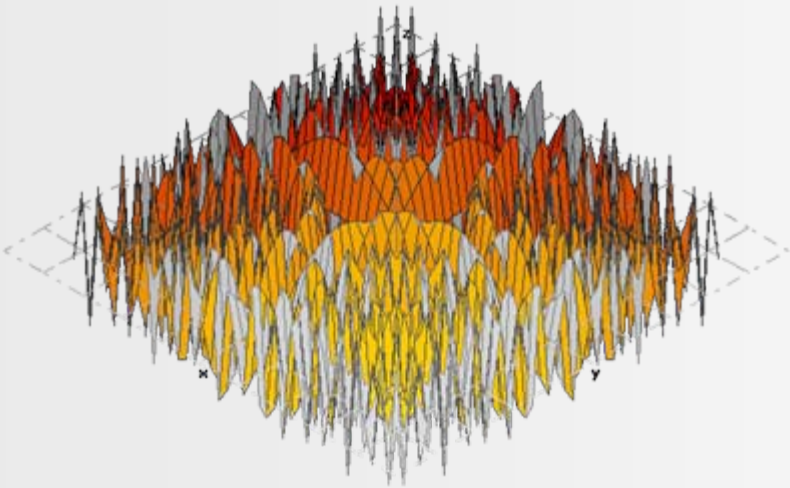
$$\begin{cases} z:[-3,3]\times[-3,3]\to R \\ z=\sin(10\cdot u\cdot v) \end{cases}$$



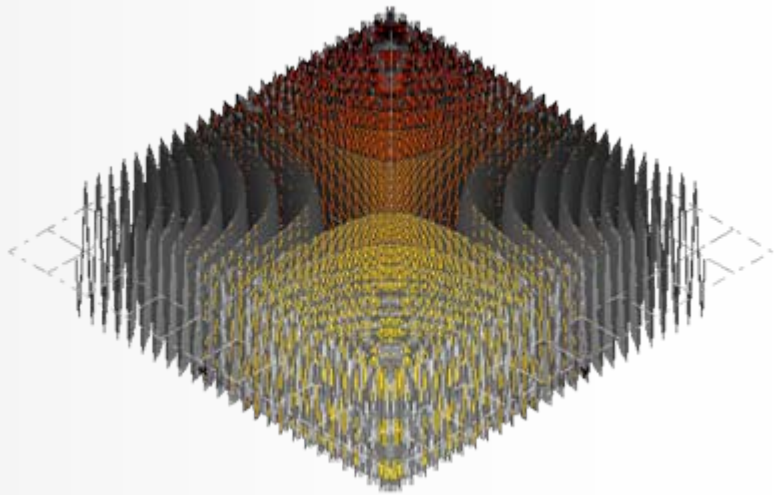
$m=n=10$



$m=n=20$

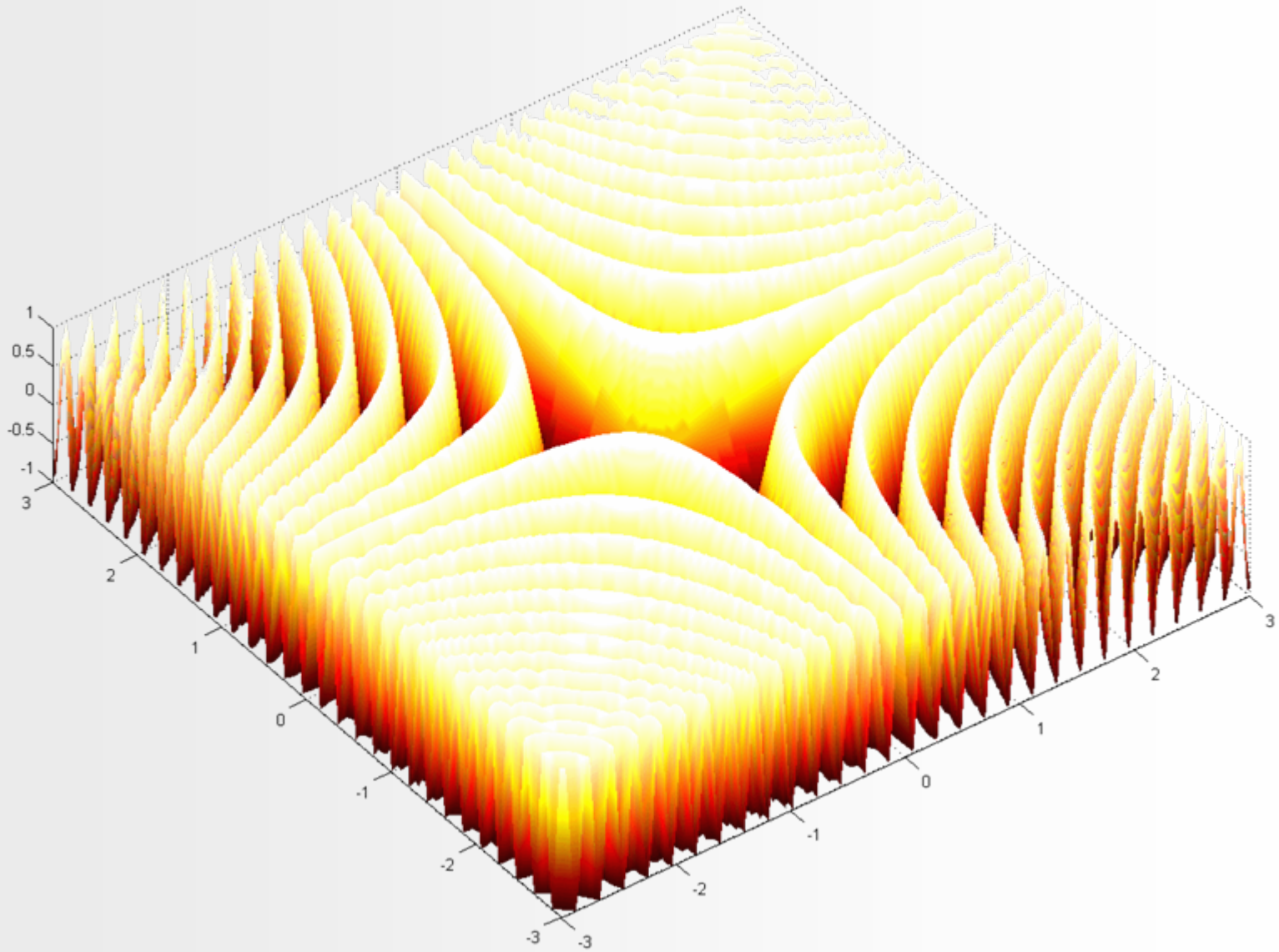


$m=n=40$



$m=n=200$

Az előbbi felület OpenGL alatt



Mire jó az operator + a CMesh3D osztályban?

```
float planeXOYx(float u, float v)
{
            return u;
}

float planeXOYy(float u, float v)
{
            return v;
}

float planeXOYz(float u, float v)
{
            return 0.0f;
}
```

```
float Torus1x(float u, float v)
{
            return (5+2*sin(u))*cos(v);
}

float Torus1y(float u, float v)
{
            return (5+2*sin(u))*sin(v);
}

float Torus1z(float u, float v)
{
            return -2*cos(u);
}
```

```
float Torus2x(float u, float v)
{
            return 5.0f+(3.0f+1.0f*sin(u))*cos(v);
}

float Torus2y(float u, float v)
{
            return 1.0f*cos(u);
}

float Torus2z(float u, float v)
{
            return (3.0f+1.0f*sin(u))*sin(v);
}
```

```
float Torus3x(float u, float v)
{
            return 1.0f*cos(u);
}

float Torus3y(float u, float v)
{
            return 5.0f+(3.0f+1.0f*sin(u))*cos(v);
}

float Torus3z(float u, float v)
{
            return (3.0f+1.0f*sin(u))*sin(v);
}
```

Mire jó az operator + a CMesh3D osztályban? (folytatás)

```
CParametricSurface3D planeXOY(-11.0f,11.0f,-11.0f,11.0f,planeXOYx,planeXOYy,planeXOYz,10,10);
CParametricSurface3D Torus1(0.0f,2.0f*pi,0.0f,2.0f*pi,Torus1x,Torus1y,Torus1z,30,30);
CParametricSurface3D Torus2(0.0f,2.0f*pi,0.0f,2.0f*pi,Torus2x,Torus2y,Torus2z,20,20);
CParametricSurface3D Torus3(0.0f,2.0f*pi,0.0f,2.0f*pi,Torus3x,Torus3y,Torus3z,8,8);
```

```
planeXOY.makeTransparent(clGray);

//...

CVector3D   lightDir( sqrt(1.0/3.0), sqrt(1.0/3.0), sqrt(1.0/3.0));
CMesh3D     m3D=planeXOY+Torus1+Torus2+Torus3;
CMesh2D     m2D=m3D.orthoProjection(lightDir);

//...

m2D.Draw(...);
```