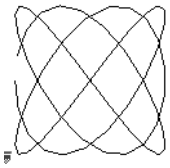
Formelsammlung: GDV

Bezeichnung	Formel

Gefülltes regelmäßiges n-Eck	`/**`
	` * Zeichnet ein gefülltes n-Eck.`
	` * x0, y0 : Koordinaten des Mittelpunkts.`
	` * r0 : Radius des Außenkreises.`
	` * nn : Anzahl der Ecken.`
	` * (nn > 0 = Spitze oben, nn < 0 = Spitze unten)`
	` */`
	`void draw_nEck(double x0, double y0, double r0, int nn)`
	`{`
	`  double x, y, u;`
	`  int k;`
	
	`  glBegin( GL_POLYGON );`
	`  for ( k=0; k<abs(nn); ++k )`
	`  {`
	`  u = PI * 2 * k / nn ;`
	
	`  if ( nn < 0 )`
	`  u += PI;`
	
	`  x = x0 + r0 * sin(u);`
	`  y = y0 + r0 * cos(u);`
	
	`  glVertex2d(x, y);`
	`  }`
	`  glEnd();`
	`}`

Lissajous-Figuren
 	
w=PI, m=3, n=4,
uMin=0.0, uMax=2*PI, du=0.1
 	
ergibt folgende Figur:
 
	`//-------------------------------------------------------`
	`// Lissajous-Figuren zeichnen`
	`//-------------------------------------------------------void`
	`drawLissajous( GLdouble w, GLdouble m, GLdouble n,`
	`  GLdouble uMin, GLdouble uMax,`
	`  GLdouble du )`
	`{`
	`  GLdouble u, x, y;`
	
	`  glBegin( GL_LINE_STRIP );`
	`  for ( u=uMin; u<uMax; u+=du )`
	`  {`
	`  glColor3d(u, 1.0-u, 0.0);`
	`  x = cos( m*u + w );`
	`  y = sin( n*u - w );`
	
	`  // Umrechnung nach links oben`
	`  x = x/4 - 0.5;`
	`  y = y/4 + 0.5;`
	
	`  glVertex2d(x, y);`
	`  }`
	`  glEnd();`
	`}`

Statische Lissajous-Figuren	`void drawLissajous(GLdouble x0, GLdouble y0, double f )`
	`{`
	`  GLdouble x, y, u, t;`
	
	`  glBegin( GL_LINES );`
	`  for ( u = 0.0; u < 1.001; u += 0.01 )`
	`  {`
	`  glColor3d( u, 1.0-u, 0.0 );`
	`  t = u * f;  x = x0 + cos(t);`
	`  y = y0 + sin(t+t);`
	`  glVertex2d( x, y );`
	`  }`
	`  glEnd();`
	`}`

Gitternetz

```
typedef struct tagR2 { R1 x; R1 y; } R2;

void drawNetz(R2 P1, R2 P2, R2 Q1, R2 Q2)
{
  GLdouble Px, Py, Qx, Qy, t;

  // Zwischenlinien
  glBegin( GL_LINES );
    for ( t = 0.0; t < 1.0001; t += 0.05 )
    {
      glColor3d( 1.0-t, t, 0.0 );
      Px = P1.x + ( P2.x - P1.x ) * t;
      Py = P1.y + ( P2.y - P1.y ) * t;
      Qx = Q1.x + ( Q2.x - Q1.x ) * ( 1.0 - t );
      Qy = Q1.y + ( Q2.y - Q1.y ) * ( 1.0 - t );

      glVertex2d( Px, Py );
      glVertex2d( Qx, Qy );
    }
  glEnd();
}
```

Kugel

```
//-------------------------------------------------------
// drawSphere(): Zeichnet ein Kreuz.
// mode   = GL_QUADS oder GL_LINE_STRIP
// r      = Radius
// nnu    = Anzahl der w1-Intervalle
// nnv    = Anzahl der w2-Intervalle
//-------------------------------------------------------
void drawSphere( int mode, double r, GLint nnu, GLint nnv )
{
  double  u1 = 0.0; double u2 = PI + PI; // Winkelgrz. w1
  double  v1 = 0.0; double v2 = PI;      // Winkelgrz. w2
  double  u  = 0.0,
          v  = 0.0,
          du, dv, xx, yy, zz, xn, yn, zn;
  int     i, j;

  du = (u2 - u1) / nnu;
  dv = (v2 - v1) / nnv;

  glEnable(GL_NORMALIZE);
  glBegin(mode);
    for ( j=0; j<nnv; j++)
    {
      for ( i=0; i<nnu; i++)
      {
        //Kugelkoordinaten:
        xn = sin(v)*cos(u); // u1 = 0.0; u2 = PI + PI;
        yn = sin(v)*sin(u); // v1 = 0.0; v2 = PI;
        zn = cos(v);        // Schrittw.  dv = (v2-v1)/nnv;
        xx=r*xn;
        yy=r*yn;
        zz=r*zn;
        glNormal3d(xn,yn,zn);
        glVertex3d(xx,yy,zz);

        v += dv;

        xn = sin(v)*cos(u); // u1 = 0.0; u2 = PI + PI;
        yn = sin(v)*sin(u); // v1 = 0.0; v2 = PI;
        zn = cos(v);        // Schrittw.  dv = (v2-v1)/nnv;
        xx=r*xn;
        yy=r*yn;
        zz=r*zn;
        glNormal3d(xn,yn,zn);
        glVertex3d(xx,yy,zz);

        u += du;

        xn = sin(v)*cos(u); // u1 = 0.0; u2 = PI + PI;
        yn = sin(v)*sin(u); // v1 = 0.0; v2 = PI;
```

```
        zn = cos(v);          // Schrittw. dv = (v2-v1)/nnv;
        xx=r*xn;
        yy=r*yn;
        zz=r*zn;
        glNormal3d(xn,yn,zn);
        glVertex3d(xx,yy,zz);

        v -= dv;

        xn = sin(v)*cos(u); // u1 = 0.0; u2 = PI + PI;
        yn = sin(v)*sin(u); // v1 = 0.0; v2 = PI;
        zn = cos(v);          // Schrittw. dv = (v2-v1)/nnv;
        xx=r*xn;
        yy=r*yn;
        zz=r*zn;
        glNormal3d(xn,yn,zn);
        glVertex3d(xx,yy,zz);
      }
      v += dv;
    }
  glEnd();
  glDisable(GL_NORMALIZE);
}
```

Typische main- und renderScene-Funktion

```
void CALLBACK RenderScene()
{
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glColor3d(1,1,1);
  glMatrixMode(GL_MODELVIEW);

  glCallList(MYSPHERE);

  auxSwapBuffers();
}


void CALLBACK key_up    (void){glRotated(+5.0, 1.0, 0.0, 0.0);}
void CALLBACK key_down  (void){glRotated(-5.0, 1.0, 0.0, 0.0);}
void CALLBACK key_left  (void){glRotated(+5.0, 0.0, 1.0, 0.0);}
void CALLBACK key_right (void){glRotated(-5.0, 0.0, 1.0, 0.0);}
void CALLBACK key_z     (void){glScaled(1/zoom, 1/zoom,1/zoom);}
void CALLBACK key_Z     (void){glScaled(zoom, zoom, zoom);}


int main ()
{
  auxInitDisplayMode(AUX_DOUBLE|AUX_RGBA|AUX_DEPTH);
  auxInitPosition(20,20,300,300);
  auxInitWindow(PRG_TITLE);

  // Initialisieren der Grafikelemente
  if ( !init() )
  {
    char *msg = "Fehler beim Initial. der Grafikelemente";
    MessageBox(NULL, msg, PRG_TITLE, MB_OK | MB_ICONERROR);
    exit(-1);
  }

  auxReshapeFunc(ChangeSize);
  auxKeyFunc( AUX_UP,     key_up);
  auxKeyFunc( AUX_DOWN,   key_down);
  auxKeyFunc( AUX_LEFT,   key_left);
  auxKeyFunc( AUX_RIGHT,  key_right);
  auxKeyFunc( AUX_z,      key_z);
  auxKeyFunc( AUX_Z,      key_Z);


  auxMainLoop(RenderScene);

  return 0;
}
```

Initialisierung (Licht)

```
//------------------------------------------------------------
// init(): Initialisierung des Programms
//------------------------------------------------------------
bool init()
{
  GLfloat ambientLight []={0.3f, 0.3f, 0.3f, 1.0f};
  GLfloat diffuseLight []={0.7f, 0.7f, 0.7f, 1.0f};
  GLfloat specularLight[]={1.0f, 1.0f, 1.0f, 1.0f};
  GLfloat lightPos     []={100.0f, 100.0f, -100.0f,  1.0f};
  GLfloat specref      []={1.0f, 1.0f, 1.0f, 1.0f};

  glCullFace(GL_BACK);       // default GL_BACK, oder GL_FRONT
  glEnable  (GL_CULLFACE);   // Rueckflaechen ausblenden
  glEnable  (GL_DEPTH_TEST); // Z-Buffer

  // Beleuchtung aktivieren
  glEnable    (GL_LIGHTING);
  glEnable    (GL_LIGHT0);
  glEnable    (GL_COLOR_MATERIAL);
  glShadeModel(GL_SMOOTH);

  glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);
  glLightfv(GL_LIGHT0, GL_SPECULAR, specularLight);
  glLightfv(GL_LIGHT0, GL_POSITION, lightPos);

  glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
  glMaterialfv   (GL_FRONT, GL_SPECULAR, specref);
  glMateriali    (GL_FRONT, GL_SHININESS, 100);

  glMatrixMode(GL_MODELVIEW);

  // S P H E R E  initialisieren.
  glNewList(MYSPHERE, GL_COMPILE);
    glPushMatrix();
      //glCullFace(GL_BACK); default GL_BACK oder GL_FRONT
      glColor3d (0.7, 0.3, 0.0);
      drawSphere( GL_LINE_STRIP, 50, 20, 20 );
      //gluSphere(gluNewQuadric(), 50, 50, 50);
    glPopMatrix();
  glEndList();

  return true;
}
```

Repaint-Funktion

```
//------------------------------------------------------------
// Repaint(): Wird aufgerufen, wenn Teile des Fensters neu
//            gezeichnet werden
//------------------------------------------------------------
void CALLBACK Repaint(int w, int h)
{
  if( h==0 ) h = 1;
  glViewport( 0,0, w, h );
  glMatrixMode( GL_MODELVIEW );
  glLoadIdentity();

  if (w <= h)
  {
    //gluOrtho2D(xMin,xMax,yMin*h/w,yMax*h/w);
    glOrtho(xMin,xMax,yMin*h/w,yMax*h/w, zMin, zMax);
  }
  else
  {
    //gluOrtho2D(xMin*w/h,xMax*w/h,yMin,yMax);
    glOrtho(xMin,xMax,yMin*h/w,yMax*h/w, zMin, zMax);
  }
}
```

BREP RGB Farbwürfel

```
//------------------------------------------------------------
// drawRGBWuerfel(): Zeichnet einen RGB-Würfel
//
// mode              : GL_POLYGON oder GL_LINES
// xPos, yPos, zPos  : Koordinaten der linken hinteren Ecke
// l                 : Kantenlänge
//------------------------------------------------------------
void drawRGBWuerfel( int mode, int dicke,
                     double xPos, double yPos,
                     double zPos, double l)
{
  GLdouble x1, y1, z1;
  GLdouble r1, g1, b1;
  GLdouble r = l/2;

  // Farben der Punkte
  GLdouble cr[] = { 0, 1, 1, 0, 0, 1, 1, 0 };
  GLdouble cg[] = { 0, 0, 1, 1, 0, 0, 1, 1 };
  GLdouble cb[] = { 0, 0, 0, 0, 1, 1, 1, 1 };

  // Position der Kanten
  GLdouble kx[] = { xPos-r, xPos-r, xPos+r, xPos+r, xPos-r,
                    xPos-r, xPos+r, xPos+r };
  GLdouble ky[] = { yPos-r, yPos-r, yPos-r, yPos-r, yPos+r,
                    yPos+r, yPos+r, yPos+r };
  GLdouble kz[] = { zPos-r, zPos+r, zPos+r, zPos-r, zPos-r,
                    zPos+r, zPos+r, zPos-r };

  // Kantentabelle - Anwählen der Kanten für Quad und Farben
  GLint k[][4] =
  {
    { 1, 2, 6, 5 },
    { 0, 4, 7, 3 },
    { 3, 7, 6, 2 },
    { 0, 1, 5, 4 },
    { 4, 5, 6, 7 },
    { 0, 3, 2, 1 },
  };

  // Normalentabelle
  GLdouble n[][3] =
  {
    { xPos+1, yPos  , zPos   },
    { xPos-1, yPos  , zPos   },
    { xPos  , yPos+1, zPos   },
    { xPos  , yPos-1, zPos   },
    { xPos  , yPos  , zPos+1 },
    { xPos  , yPos  , zPos-1 },
  };

  glLineStipple(dicke, 0xFFFF);

  glEnable(GL_LINE_STIPPLE);
  glEnable(GL_NORMALIZE);

  glPushMatrix();
  glRotated(-30.0, 1, 1, 0);

  glBegin(modus);
    for ( int i=0; i<6; ++i ) // 6 Seiten
    {
      glNormal3d(n[i][0], n[i][1], n[i][2]);

      for ( int j=0; j<4; ++j ) // 4 Ecken je Seite
      {
        x1 = kx[k[i][j]];
        y1 = ky[k[i][j]];
        z1 = kz[k[i][j]];

        r1 = cr[k[i][j]];
        g1 = cg[k[i][j]];
```

```
        b1 = cb[k[i][j]];

        glColor3d(r1, g1, b1);
        glVertex3d(x1, y1, z1);
      }
    }
  glEnd();

  glPopMatrix();

  glDisable(GL_LINE_STIPPLE);
  glDisable(GL_NORMALIZE);

}
```