Aufgabe 9

Ausgehend von der impliziten Darstellung einer Ellipse im Ursprung durch

$$F(x,y) = a^2x^2 + b^2y^2 - a^2b^2 = 0$$

wobei $2a$ der Durchmesser entlang der y-Achse ist, muss der Übergang von der Region 1 in die Region 2 genau an der Stelle geschehen, wo der Anstieg der Tangente kleiner als –1 wird. Dieser Fall tritt ein, wenn der Anstieg entlang der x-Achse kleiner als der Anstieg entlang der y-Achse wird (Achtung, Vorzeichen: die Beträge sind zu vergleichen !).

Nachdem nun hinreichend untersucht worden ist, wie man den Übergang von Region 1 in Region 2 ermittelt, scheinen viel wichtiger jedoch die Entscheidungsvariablen d_1 und d_2 zu sein, die ermitteln, ob der nächste zu zeichnende Punkt E oder SE ist. Die dazugehörige Bestimmung erfordert erneut eine Aufteilung der Berechung in die beiden Regionen.

In der Region 1 ergibt folgendes Bild:

$$M = \left(x+1, y-\frac{1}{2}\right)$$
$$F(M) = a^2(x+1)^2 + b^2\left(y-\frac{1}{2}\right)^2 - a^2b^2$$
$$M_E = \left(x+2, y-\frac{1}{2}\right)$$
$$F(M_E) = a^2(x+2)^2 + b^2\left(y-\frac{1}{2}\right)^2 - a^2b^2$$
$$M_{SE} = \left(x+2, y-\frac{3}{2}\right)$$
$$F(M_{SE}) = a^2(x+2)^2 + b^2\left(y-\frac{3}{2}\right)^2 - a^2b^2$$

Wenn im nächsten Schritt E gewählt werden soll, so ist

$$\begin{aligned} F(M_E) - F(M) &= a^2(x+2)^2 - a^2(x+1)^2 \\ &= a^2(2x+3) \\ &= \delta_E \\ &= d_x \end{aligned}$$

Für SE dagegen:

$$
\begin{aligned}
F(M_{SE}) - F(M) &= a^2(x+2)^2 + b^2\left(y-\frac{3}{2}\right)^2 - a^2(x+1)^2 + b^2\left(y-\frac{1}{2}\right)^2 \\
&= a^2(2x+3) + b^2(-2y+2) \\
&= \delta_{SE} \\
&= d_x + d_y
\end{aligned}
$$

Da die Berechnung der Ellipse in $(0,a)$ startet, lautet der erste zu untersuchende Mittelpunkt $\left(1, a-\frac{1}{2}\right)$:

$$
\begin{aligned}
F\left(1, a-\frac{1}{2}\right) &= a^2 + b^2\left(a-\frac{1}{2}\right)^2 - a^2b^2 \\
&= a^2 - ab^2 + \frac{b^2}{4} \\
&= d_1
\end{aligned}
$$

Man stellt fest, dass zwar d_1 nur einmal berechnet werden muss und daher die Quadrate etc. vertretbar sind, allerdings wäre es bedeutend besser, wenn d_x und d_y sich auf einfache Additionen zurückführen lassen würden. Aus diesem Gründe führe ich dd_x und dd_y ein, die sich aus den jeweiligen Ableitungen ergeben:

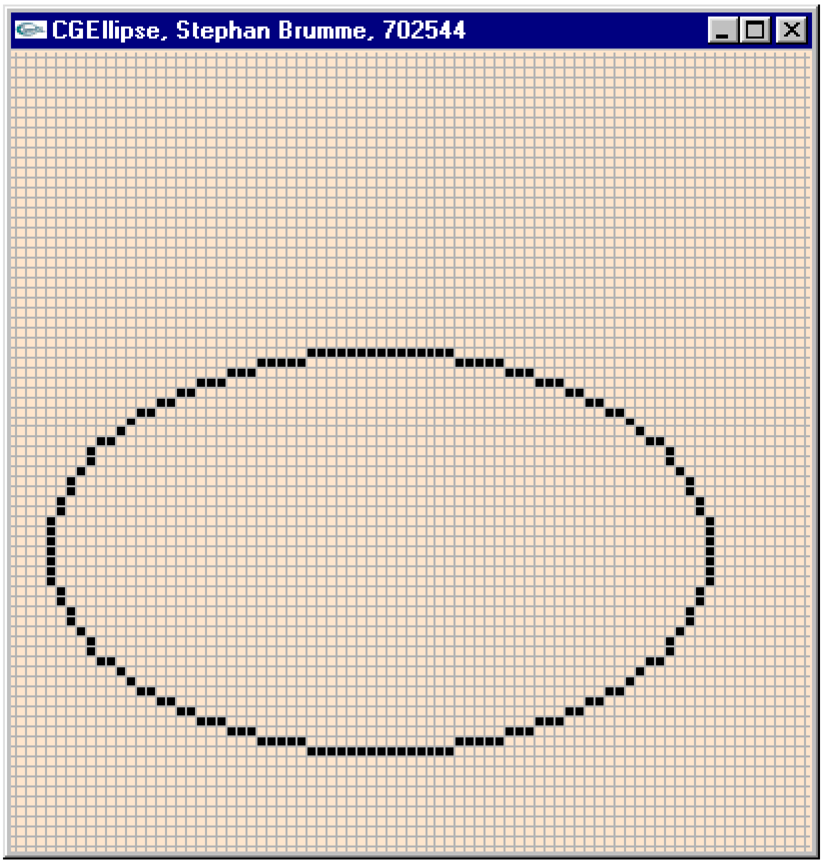
$$
\begin{aligned}
dd_x &= 2a^2 \\
dd_y &= -2b^2
\end{aligned}
$$

mit

$$
\begin{aligned}
d_{x+1} &= d_x + dd_x \\
d_{y+1} &= d_y + dd_y
\end{aligned}
$$

Um mich von der Korrektheit meiner Überlegungen überzeugen zu können, schrieb ich ein kleines OpenGL-Programm, dessen Programmrahmen der Linien-Midpoint-Hausaufgabe entnommen wurde und nur minimale Änderungen ausweist, die sich hauptsächlich in `drawEllipse` und `drawEllipsePoints` konzentrieren.

Mit niedergedrückter Maustaste können interaktiv beliebige Ellipsen dargestellt werden:



Für die Region 1 entstanden dann diese Zeilen:

```
// oben mittig beginnen
int x = 0;
int y = A;

// "Enden" der Ellipsen zeichnen
drawEllipsePoints(midX, midY, 0, A);

// Entscheidungsvariable für E/SE
int d1  = A2-A*B2+B2/4;

// Inkremente 1. und 2.Ordnung für x und y
int dX  = A2*3;
int ddX = A2*2;
int dY  = B2*(-2*A+2);
int ddY = 2*B2;

// solange in Region 1
while (dX < -dY || d1 < 0)
{

    // nach Süden ?
    if (d1 > 0)
    {
        d1 += dY;
        dY += ddY;
        y--;
    }
```

```
        // immer nach Osten
        d1 += dX;
        dX += ddX;
        x++;

        // alle 4 Quadranten zeichnen
        drawEllipsePoints(midX, midY, x, y);
    }
```

Die Region 2 wird ganz ähnlich bearbeitet, hier liegt der wesentliche Unterschied darin, dass M_S und M_{SE} etwas anders berechnet werden, da der Mittelpunkt jetzt in der Horizontalen gesucht wird (siehe M):

$$M = \left(x + \frac{1}{2}, y - 1\right)$$

$$F(M) = a^2\left(x + \frac{1}{2}\right)^2 + b^2(y-1)^2 - a^2b^2$$

$$M_S = \left(x + \frac{1}{2}, y - 2\right)$$

$$F(M_S) = a^2\left(x + \frac{1}{2}\right)^2 + b^2(y-2)^2 - a^2b^2$$

$$M_{SE} = \left(x + \frac{3}{2}, y - 2\right)$$

$$F(M_{SE}) = a^2\left(x + \frac{3}{2}\right)^2 + b^2(y-2)^2 - a^2b^2$$

Somit gilt für S:

$$\begin{aligned} F(M_S) - F(M) &= b^2(y-2)^2 - b^2(y-1)^2 \\ &= b^2(-2y+3) \\ &= \delta_S \\ &= d_y \end{aligned}$$

Und SE:

$$\begin{aligned} F(M_{SE}) - F(M) &= a^2\left(x + \frac{3}{2}\right)^2 + b^2(y-2)^2 - a^2\left(x + \frac{1}{2}\right)^2 + b^2(y-1)^2 \\ &= a^2(2x+2) + b^2(-2y+3) \\ &= \delta_{SE} \\ &= d_x + d_y \end{aligned}$$

Erneut bringt uns die 2.Ableitung eine Zurückführung der Berechnungen auf Additionen:

$$dd_x = 2a^2$$
$$dd_y = -2b^2$$

Interessant ist, dass diese Formeln *exakt* denen aus Region 1 entsprechen und daher nicht neu gebildet werden müssen.

Natürlich muss auch hier wieder eine Initialisierung des Mittelpunktes stattfinden, da wir den bisherigen nicht weiterverwenden können:

$$F\left(x+\frac{1}{2}, y-1\right) = a^2\left(x+\frac{1}{2}\right)^2 + b^2(y-1)^2 - a^2b^2$$
$$= a^2x^2 + a^2x + \frac{1}{4} + b^2(y-1)^2 - a^2b^2$$

Die Endebedingung besteht im Erreichen der x-Achse.
Umgesetzt sieht die Region 2 wie folgt aus:

```
// Entscheidungsvariable für SE/S
int d2 = A2*x*x+A2*x + B2*(y-1)*(y-1) - A2*B2;

// Inkremente 1.Ordnung für x und y
dX     = A2*(2*x+2);
dY     = B2*(-2*y+3);
// Inkremente 2.Ordnung können unverändert aus Region 1 übernommen werden

// bis x-Achse errreicht
while (y > 0)
{
    // nach Osten ?
    if (d2 < 0)
    {
        d2 += dX;
        dX += ddX;
        x++;
    }

    // immer nach Süden
    d2 += dY;
    dY += ddY;
    y--;

    // alle 4 Quadranten zeichnen
    drawEllipsePoints(midX, midY, x, y);
}
```

Aufgabe 10

a) Es ist definitionsgemäß

$$v \times w = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \times \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix}$$

Für das Skalarprodukt $v \bullet (v \times w)$ gilt dann:

$$\begin{aligned} v \bullet (v \times w) &= \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \bullet \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix} \\ &= v_1 (v_2 w_3 - v_3 w_2) + v_2 (v_3 w_1 - v_1 w_3) + v_3 (v_1 w_2 - v_2 w_1) \\ &= v_1 v_2 w_3 - v_1 v_3 w_2 + v_2 v_3 w_1 - v_1 v_2 w_3 + v_1 v_3 w_2 - v_2 v_3 w_1 \\ &= 0 \end{aligned}$$

Analog für $w \bullet (v \times w)$:

$$\begin{aligned} w \bullet (v \times w) &= \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \bullet \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix} \\ &= w_1 (v_2 w_3 - v_3 w_2) + w_2 (v_3 w_1 - v_1 w_3) + w_3 (v_1 w_2 - v_2 w_1) \\ &= w_1 v_2 w_3 - w_1 v_3 w_2 + w_2 v_3 w_1 - w_1 v_2 w_3 + w_1 v_3 w_2 - w_2 v_3 w_1 \\ &= 0 \end{aligned}$$

Da zwei Vektoren genau dann orthogonal zueinander sind, wenn das Skalarprodukt 0 ist, haben obige Gleichungen gezeigt, dass $v \perp v \times w$ und $w \perp v \times w$ gilt.

b) Die Herleitung beruht im wesentlichen auf der Verwendung der Definition des Vektorproduktes und der Addition von Vektoren:

$$
\begin{aligned}
v \times (w+u) &= \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \times \begin{pmatrix} w_1 + u_1 \\ w_2 + u_2 \\ w_3 + u_3 \end{pmatrix} \\
&= \begin{pmatrix} v_2(w_3 + u_3) - v_3(w_2 + u_2) \\ v_3(w_1 + u_1) - v_1(w_3 + u_3) \\ v_1(w_2 + u_2) - v_2(w_1 + u_1) \end{pmatrix} \\
&= \begin{pmatrix} v_2 w_3 - v_3 w_2 + v_2 u_3 - v_3 u_2 \\ v_3 w_1 - v_1 w_3 + v_3 u_1 - v_1 u_3 \\ v_1 w_2 - v_2 w_1 + v_1 u_2 - v_2 u_1 \end{pmatrix} \\
&= \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix} + \begin{pmatrix} v_2 u_3 - v_3 u_2 \\ v_3 u_1 - v_1 u_3 \\ v_1 u_2 - v_2 u_1 \end{pmatrix} \\
&= v \times w + v \times u
\end{aligned}
$$

c)

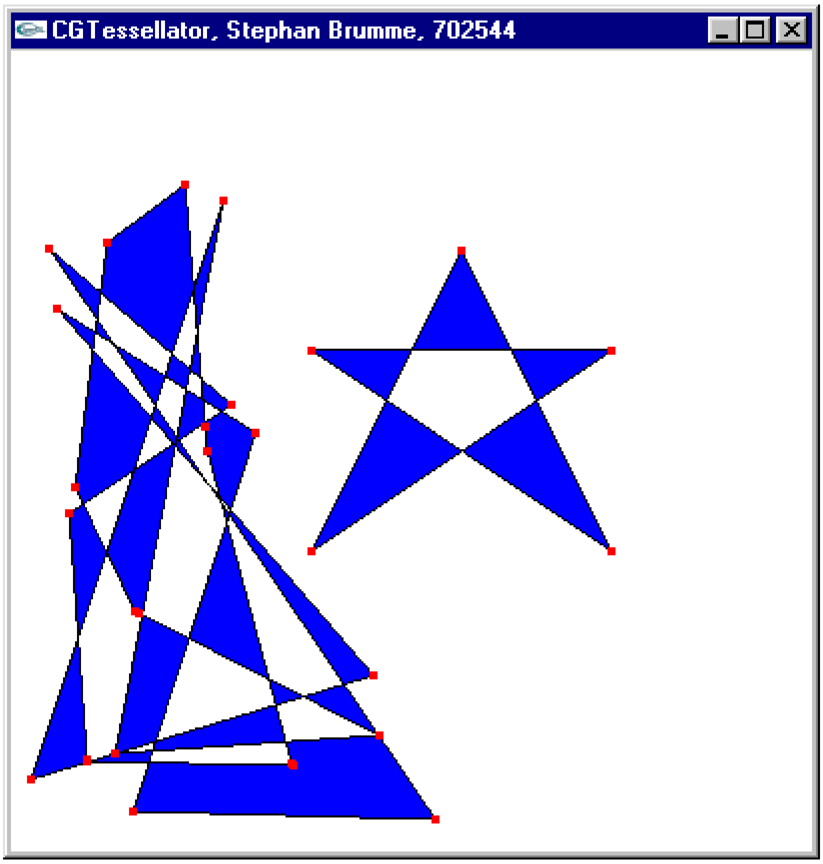
$$
\begin{aligned}
(v \times w) \bullet (v \times w) &= \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix} \bullet \begin{pmatrix} v_2 w_3 - v_3 w_2 \\ v_3 w_1 - v_1 w_3 \\ v_1 w_2 - v_2 w_1 \end{pmatrix} \\
&= (v_2 w_3 - v_3 w_2)^2 + (v_3 w_1 - v_1 w_3)^2 + (v_1 w_2 - v_2 w_1)^2 \\
&= |v \times w|^2
\end{aligned}
$$

Da $|v \times w|$ dem Flächeninhalt des von v und w aufgespannten Parallelogramms entspricht, ist $|v \times w|^2 = (v \times w) \bullet (v \times w)$ des Quadrat eben dieser Figur.

Nur wenn v und w linear abhängig sind, ist $|v \times w|^2 = 0$.

Aufgabe 11

Die Implementation ist relativ einfach gehalten, mit der Maus legt man neue Punkte fest, über Tastaturbefehle kann deren Darstellungsform geändert werden.



Taste	Aktion
k	neue Kontur beginnen
s	einen Stern zeichnen lassen
c	Bildschirm und Konturen löschen
1	Windungsregel ODD (Standard)
2	Windungsregel NONZERO
3	Windungsregel POSITIVE
4	Windungsregel NEGATIVE
5	Windungsregel ABS_GEQ_TWO
q	Programm beenden

Der Quellcode findet sich im Anhang wieder, ich habe die von mir eingefügten Zeilen weitgehend kommentiert.

Anhang

Die hier aufgeführten Quelltexte beziehen sich nur auf die Dateien, die im gegebenen Programmrahmen geändert werden mussten. Aus diesem Grunde findet man hier nicht CGApplication.cpp etc.

Quelltext zu Aufgabe 9

source file "cgellipse.h"

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 4

// Stephan Brumme, 702544
// last changes: May 19, 2001

#ifndef CG_BRESENHAM_H
#define CG_BRESENHAM_H

#include "cgapplication.h"
#include "cgraster.h"

class CGEllipse : public CGApplication {
public:
    CGEllipse(int width, int height);

    virtual void onInit();
    virtual void onDraw();
    virtual void onSize(unsigned int newWidth,unsigned  int newHeight);

    virtual void onButton(MouseButton button, MouseButtonEvent event, int x, int y);
    virtual void onMove(MouseButton button, int x, int y);

private:
    // zeichenen der Ellipse in das Raster raster_
    void drawEllipse(int x1, int y1, int x2, int y2);
    void drawEllipsePoints(int midx, int midy, int x, int y);

    // interne Raster-Klasse
    CGRaster raster_;

    // Fenstergroesse speichern
    int winWidth_;
    int winHeight_;

    // linke untere und rechte obere Ecke des Rechteckes, das die Ellipse beschreibt
    int xBegin_;
    int yBegin_;
    int xEnd_;
    int yEnd_;
    bool First_;
};

#endif // CG_BRESENHAM_H
```

source file "cgellipse.cpp"

```
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 4

// Stephan Brumme, 702544
// last changes: May 19, 2001
```

```
#include "cgellipse.h"


CGEllipse::CGEllipse(int width, int height) : raster_(width,height) {
    // Objektvariablen initialisieren
    First_ = true;
}

void CGEllipse::onInit() {
    // Hintergrundfarbe weiss
    glClearColor(1, 0.9, 0.8, 1);

    // ohne perspektivische Verzerrung, da nur 2D
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, raster_.width()-1, 0, raster_.height()-1);
}

void CGEllipse::onDraw() {
    // Colorbuffer loeschen
    glClear(GL_COLOR_BUFFER_BIT);

    // Zeichenen der Raster-Klasse
    raster_.draw();

    // Backbuffer anzeigen
    swapBuffers();
}

void CGEllipse::onSize(unsigned int newWidth, unsigned int newHeight) {
    // neue Fenstergröße speichern
    winWidth_ = newWidth;
    winHeight_ = newHeight;
    glViewport(0, 0, newWidth - 1, newHeight - 1);
}

void CGEllipse::onButton(MouseButton button, MouseButtonEvent event, int x, int y) {
    // Sichern der x und y Werte
    // Anfang und Ende unterscheiden durch MouseButtonEvent:
    // MouseButtonDown = Start
    // MouseButtonUp = End

    // Mausposition in Rasterkoordinaten umrechnen
    x = (int)(0.5 + x / ((float)winWidth_ /(raster_.width()-1)));
    y = (int)(0.5 + y / ((float)winHeight_/(raster_.height()-1)));

    // Linienanfang
    if (event == MouseButtonDown)
    {
        xBegin_ = x;
        yBegin_ = y;
    }

    // Linienende
    if (event == MouseButtonUp)
    {
        xEnd_ = x;
        yEnd_ = y;
        // Endpunkt steht fest, Linie darf gezeichnet werden
        First_ = false;
    }

    // Raster löschen
    raster_.clear();
    // Linie im Raster zeichnen
    if (!First_)
        drawEllipse(xBegin_, yBegin_, xEnd_, yEnd_);

    // Raster auf dem Bildschirm darstellen
    onDraw();
}

void CGEllipse::onMove(MouseButton button, int x, int y) {
    // Endpunkt ermitteln
    onButton(button, MouseButtonUp, x, y);
```

```
}

void CGEllipse::drawEllipsePoints(int midx, int midy, int x, int y)
{
    // zeichne in allen 4 Quadranten, beachte Verschiebung aus Ursprung heraus
    raster_.setPixel(midx+x, midy+y);
    raster_.setPixel(midx+x, midy-y);
    raster_.setPixel(midx-x, midy+y);
    raster_.setPixel(midx-x, midy-y);
}

void CGEllipse::drawEllipse(int x1, int y1, int x2, int y2) {
    // Ellipsen Algorithmus
    // Zeichnen mit der Raster-Klasse

    // ändere ggf. Eckpunkte, sodass untere linke Ecke per x1/y1
    // und obere rechte per x2/y2 definiert wird
    if (x2 < x1)
    {
        int temp;

        temp = x2;
        x2 = x1;
        x1 = temp;
    }
    if (y2 < y1)
    {
        int temp;

        temp = y2;
        y2 = y1;
        y1 = temp;
    }

    // Ellipsenparameter
    int A = (y2-y1) / 2;
    int B = (x2-x1) / 2;
    int A2= A*A;
    int B2= B*B;

    // verweigere zu kleine Ellipsen
    if (A == 0 || B == 0)
        return;

    // Ellipsenmittelpunkt
    int midX = x1+B;
    int midY = y1+A;

    // oben mittig beginnen
    int x = 0;
    int y = A;

    // "Enden" der Ellipsen zeichnen
    drawEllipsePoints(midX, midY, 0, A);

    // Entscheidungsvariable für E/SE
    int d1  = A2-A*B2+B2/4;

    // Inkremente 1. und 2.Ordnung für x und y
    int dX  = A2*3;
    int ddX = A2*2;
    int dY  = B2*(-2*A+2);
    int ddY = 2*B2;

    // solange in Region 1
    while (dX < -dY || d1 < 0)
    {

        // nach Süden ?
        if (d1 > 0)
        {
            d1 += dY;
            dY += ddY;
            y--;
        }
```

```
        // immer nach Osten
        d1 += dX;
        dX += ddX;
        x++;

        // alle 4 Quadranten zeichnen
        drawEllipsePoints(midX, midY, x, y);
    }


    // Entscheidungsvariable für SE/S
    int d2 = A2*x*x+A2*x + B2*(y-1)*(y-1) - A2*B2;

    // Inkremente 1.Ordnung für x und y
    dX     = A2*(2*x+2);
    dY     = B2*(-2*y+3);
    // Inkremente 2.Ordnung können unverändert aus Region 1 übernommen werden

    // bis x-Achse errreicht
    while (y > 0)
    {
        // nach Osten ?
        if (d2 < 0)
        {
            d2 += dX;
            dX += ddX;
            x++;
        }

        // immer nach Süden
        d2 += dY;
        dY += ddY;
        y--;

        // alle 4 Quadranten zeichnen
        drawEllipsePoints(midX, midY, x, y);
    }
}

int main(int argc, char* argv[]) {
    CGEllipse ellipse(81, 81);
    ellipse.start(argc, argv, "CGEllipse, Stephan Brumme, 702544");
    return(0);
}
```

Quelltext Aufgabe 11

source file "cgtessellator.h"

```
//
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 4
//

// Stephan Brumme, 702544
// last changes: May 20, 2001


#ifndef CG_TESSELATOR_H
#define CG_TESSELATOR_H

#include "cgapplication.h"

class CGTessellator : public CGApplication {
public:
    CGTessellator();
    ~CGTessellator();
```

```
    virtual void defineStar();

    virtual void onInit();
    virtual void onDraw();
    virtual void onSize(unsigned int newWidth, unsigned int newHeight);
    virtual void onKey(unsigned char key);
    virtual void onButton(MouseButton button, MouseButtonEvent event, int x, int y);

private:
    // zeichnet das Polygon mit den GLU-Routinen
    void drawPolygon();

    // GLUtesselator Objekt
    GLUtesselator* tobj_;

    // Hilfsklasse fuer Punkte
    class Point {
    public:
        Point(double x = 0, double y = 0) {
            dta_[0] = x;
            dta_[1] = y;
            dta_[2] = 0;
        }
        GLdouble dta_[3];
    };

    // interne Arrays zum speichern der Punkte und
    // der Konturen
    Point pts_[1000];       // alle Punkte
    int pos_;               // Array-Position in pts_
    int contourSize_[50];   // Anzahl der Punkte innerhalb einer Kontur
    int contours_;          // Anzahl der Konturen

};

#endif // CG_TESSELATOR_H
```

source file "cgtessellator.cpp"

```
//
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 4
//

// Stephan Brumme, 702544
// last changes: May 20, 2001

#include "cgtessellator.h"

// Function Makro
#if defined(__GNUC__) && !defined(__STRICT_ANSI__)
#define FUNC GLvoid(*)(...)
#elif defined(_MSC_VER)
#define FUNC void(__stdcall*)()
#else
#define FUNC GLvoid(*)()
#endif

// GLU tesselator globale Hilfsfunktionen

// Fehler abfangen
void CALLBACK error(GLenum err) {
    cerr << gluErrorString(err) << endl;
}

// Schnitt von Kanten, nimmt keine Änderungen vor
void CALLBACK combineCallback(GLdouble coords[3],
                              GLdouble *vertex_data[4],
                              GLfloat weight[4], GLdouble **dataOut )
```

```
{
    GLdouble* vertex = new GLdouble[3];
    vertex[0] = coords[0];
    vertex[1] = coords[1];
    vertex[2] = coords[2];
    *dataOut = vertex;
}

// stellt Ecke dar
void CALLBACK vertexCallback(GLvoid *vertex)
{
    glVertex3dv((GLdouble*)vertex);
}

// beginnt ein Primitv
void CALLBACK beginCallback(GLenum which)
{
    glBegin(which);
}

// beendet ein Primitiv
void CALLBACK endCallback()
{
    glEnd();
}


// CGTessellator
CGTessellator::CGTessellator() {
    // Anlegen des TessObj
    tobj_ = gluNewTess();
    gluTessProperty(tobj_, GLU_TESS_WINDING_RULE, GLU_TESS_WINDING_ODD);

    // Datenstruktur initialisieren
    pos_ = 0;
    contours_ = 1;
    contourSize_[0] = 0;
}

CGTessellator::~CGTessellator() {
    // Zerstoeren des TessObj
    gluDeleteTess(tobj_);
}

// Setze Punkte fuer einen Stern
void CGTessellator::defineStar() {
    // 5 Punkte insgesamt
    pos_ = 5;
    // 1 Kontur
    contours_ = 1;
    // diese Kontur enthält alle bisher definierten Punkte
    contourSize_[0] = 5;
    pts_[0] = Point(150.0, 150.0);
    pts_[1] = Point(225.0, 300.0);
    pts_[2] = Point(300.0, 150.0);
    pts_[3] = Point(150.0, 250.0);
    pts_[4] = Point(300.0, 250.0);

    // Stern zeichnen
    glutPostRedisplay();
}

void CGTessellator::onInit() {
    glClearColor(1, 1, 1, 1);

    // festlegen der Vertex, Begin, End und Error Callback
    // Beispiel: Combine Callback, Achtung: FUNC-Cast!
    gluTessCallback(tobj_, GLU_TESS_COMBINE, (FUNC) &combineCallback);
    gluTessCallback(tobj_, GLU_TESS_VERTEX, (FUNC) &vertexCallback);


    gluTessCallback(tobj_, GLU_TESS_BEGIN, (FUNC) &beginCallback);
    gluTessCallback(tobj_, GLU_TESS_END, (FUNC) &endCallback);

    gluTessCallback(tobj_, GLU_TESS_ERROR, (FUNC) &error);
```

```
}

void CGTessellator::drawPolygon() {
    gluTessBeginPolygon(tobj_, NULL);

    // alle Punkte durchlaufen
    int nVertexPtr = 0;

    // alle Polygone
    for (int nContour = 0; nContour < contours_; nContour++)
    {
        gluTessBeginContour(tobj_);

        // alle Eckpunkte eines Polygons
        for (int nVertex=0; nVertex<contourSize_[nContour]; nVertex++)
        {
            gluTessVertex(tobj_, pts_[nVertexPtr].dta_, pts_[nVertexPtr].dta_);
            nVertexPtr++;
        }

        gluTessEndContour(tobj_);
    }
    gluTessEndPolygon(tobj_);
    // Tessellation des Polygons mit GLU-Routinen
}

void CGTessellator::onDraw() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Zeichnen des Polygons (blau)
    glColor3f(0,0,1);
    drawPolygon();

    // Zeichnen des Polygon-Randes (schwarz), benutzt erneut Tessellator
    glColor3f(0,0,0);
    gluTessProperty(tobj_, GLU_TESS_BOUNDARY_ONLY, GL_TRUE);
    drawPolygon();
    gluTessProperty(tobj_, GLU_TESS_BOUNDARY_ONLY, GL_FALSE);

    // Zeichnen der Eckpunkte (rot)
    glPointSize(4);
    glColor3d(1,0,0);

    glBegin(GL_POINTS);
    for (int i = 0; i < pos_; i++)
        glVertex2dv(pts_[i].dta_);
    glEnd();

    // tauschen wie im Swingerclub
    swapBuffers();
}

void CGTessellator::onSize(unsigned int newWidth, unsigned int newHeight) {
    glClear(GL_COLOR_BUFFER_BIT);
    glViewport(0, 0, newWidth - 1, newHeight - 1);

    glMatrixMode(GL_PROJECTION_MATRIX);
    glLoadIdentity();
    gluOrtho2D(0, newWidth-1, 0, newHeight-1);
    glMatrixMode(GL_MODELVIEW_MATRIX);

}

void CGTessellator::onKey(unsigned char key) {
    // Key Belegung
    switch (key) {
        // beenden
    case 'q': exit(0); break;

        // Stern zeichnen
    case 's': defineStar(); break;

        // neue Kontur beginnen
    case 'k': contourSize_[contours_++] = 0; break;
```

```
        // alle Punkte/Bildschirm löschen
    case 'c': contours_ = 1;
              contourSize_[0] = 0;
              pos_ = 0;
              break;

        // Tessellationsalgorithmen
    case '1': gluTessProperty(tobj_, GLU_TESS_WINDING_RULE, GLU_TESS_WINDING_ODD); break;
    case '2': gluTessProperty(tobj_, GLU_TESS_WINDING_RULE, GLU_TESS_WINDING_NONZERO); break;
    case '3': gluTessProperty(tobj_, GLU_TESS_WINDING_RULE, GLU_TESS_WINDING_POSITIVE); break;
    case '4': gluTessProperty(tobj_, GLU_TESS_WINDING_RULE, GLU_TESS_WINDING_NEGATIVE); break;
    case '5': gluTessProperty(tobj_, GLU_TESS_WINDING_RULE, GLU_TESS_WINDING_ABS_GEQ_TWO);
break;
    }

    // und neu zeichnen
    glutPostRedisplay();
}

void CGTessellator::onButton(MouseButton button, MouseButtonEvent event,
                            int x, int y) {
    // Punkt speichern
    pts_[pos_] = Point(x, y);
    contourSize_[contours_-1]++;
    pos_++;

    // und neu zeichnen
    glutPostRedisplay();
}

int main(int argc, char* argv[]) {
    CGTessellator tess;
    tess.start(argc, argv, "CGTessellator, Stephan Brumme, 702544",
        CGTessellator::ColorBuffer| CGTessellator::DoubleBuffer |
CGTessellator::StencilBuffer,
        200, 400, 400);
    return(0);
}
```