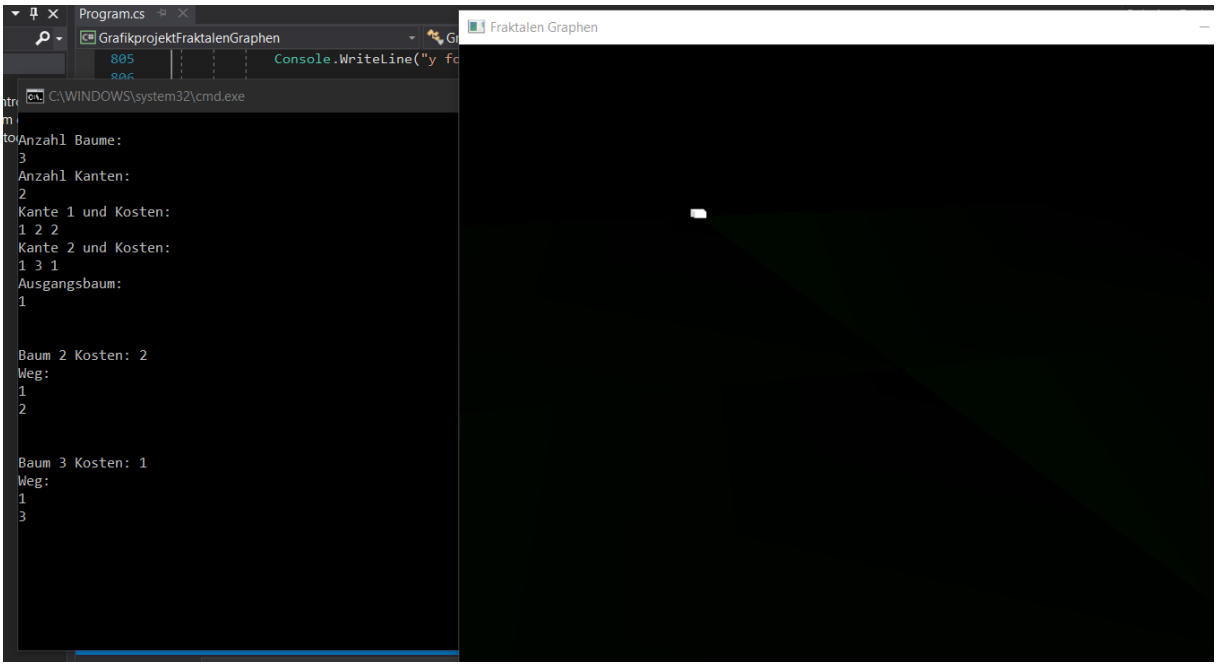
Matei Diana
Fraktalengraphen

FraktalenGraphen – Dokumentation

Mein Grafikprojekt ist eine Fortsetzung des Projektes aus der 10. Klasse in C++, Fraktalengarten. Das Projekt basierte sich auf dem Konzept der Fraktalen, und zwar eine rekursive Funktion, mit Hilfe deren man ein Modell erzeugt, das sich bis zu einem bestimmten Zeitpunkt wiederholt. Weil solche Schemen oft in der Natur vorkommen, kann man sie implementieren, um Pflanzen darzustellen, insbesonders Bäume, deren Äste sich verzweigen. In einer 3D Umwelt konnte man sich den eigenen Mini-Garten bauen. Ein Würfel bewegte sich auf einem Plan (einer Ebene). Die Koordinaten des Würfels wurden gespeichert, und auf diesen Koordinaten konn man verschiedene Elemente des Gartens einfügen, die im Menü links genannt wurden, das auf dem Bildschrim beim Laufen des Projektes erscheinte.

Das Projekt dieses Jahres geht ein Paar Schritte weiter. Es wurde als C# Console Application in Visual Studio 2017 mit Hilfe dem API OpenGL und der Bibliothek Tao.Freeglut gestaltet. Es enthält alle Elemente des vorigen Projektes, aber auch Graphentheorie. Da es nützlich ist, beim Bauen eines eigentlichen Gartens Distanzen im Plan zu berechnen, rechnet auch der Programm die minimale Distanz zwischen einen Ausgangbaum, der ausgewählt wird und alle anderen Bäume. Die Distanz zwischen den Bäumen wird auch von der Tastatur eingelesen.

Beim Laufen des Projektes erscheint die Console mit dem Menu und die Einstellungen, die im Programm mit welchen Tasten verändert werden können. Es wird auch user Input verlangt, und zwar wie viele Bäume eingefügt werden, wie viele Verbindungen es zwischen den Bäumen gibt, welche diese sind und den Ausgangsbaum. Es werden die Minimalkosten aufgeschrieben und der Weg. Rechts wird das Grafikfenster initialisiert.



Die Bäume werden mit der Space Taste eingefügt. Wenn man p drückt, erscheinen die Distanzen berechnet mit Dijkstra im Plan.

Matei Diana
Fraktalengraphen

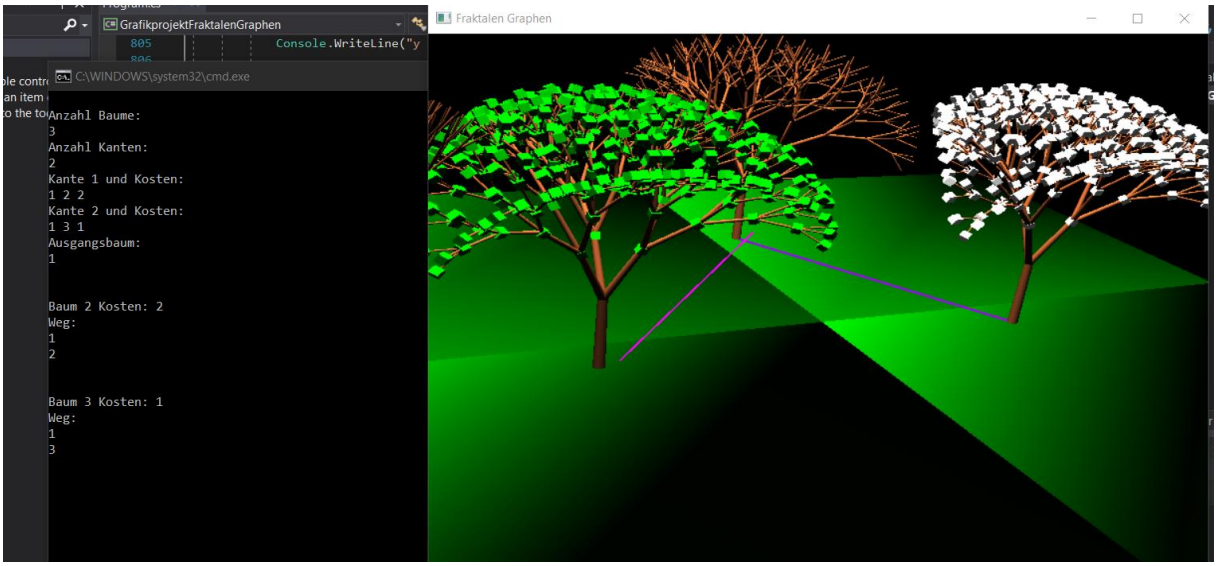


Abbildung 1: Distanzen werden im Plan laut den Eingabedaten eingefügt

Weitere Elemente die eingefügt werden können sind Sträucher mit Taste /, Gras mit . und Teiche mit ,.

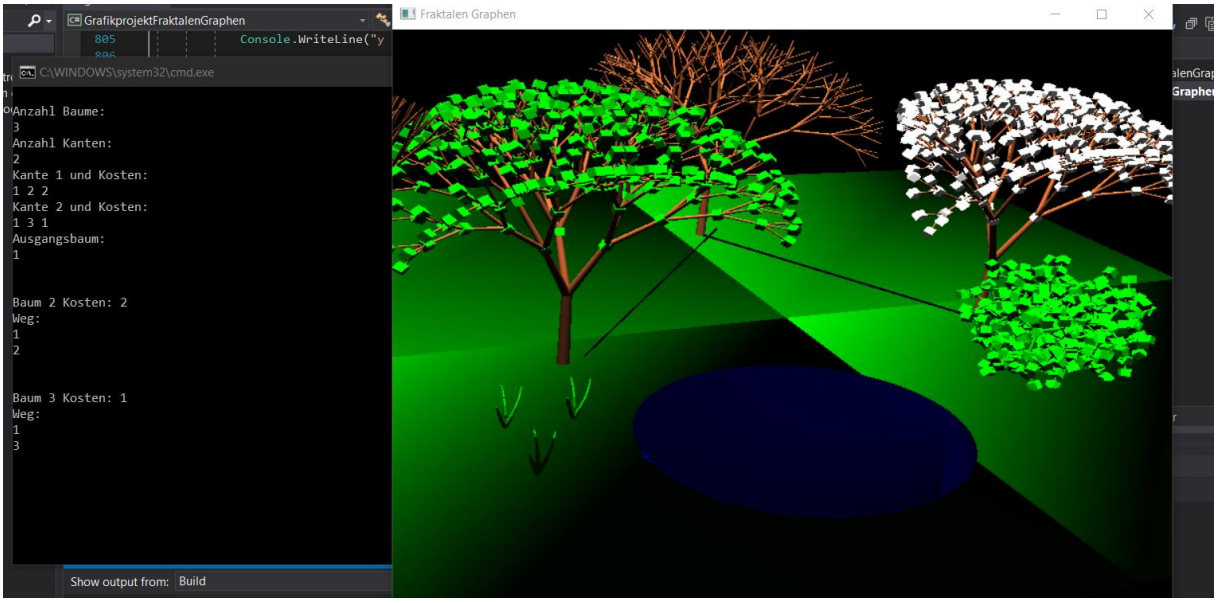


Abbildung 2: Alle Elemente, die man einfügen kann

Matei Diana
Fraktalengraphen

Codeteil

Bäume

Die Bäume bestehen aus Zylinder, die immer dünner werden und sich verzweigen und haben kleine Würfeln als Blüten.

Für das Darstellen von Bäumen wird eine rekursive Funktion benutzt, die als Parameter die Höhe des Stammes (des ersten Zylinders), den Durchmesser der Basis, den Rekursionswinkel, die Parameter für die Farbe des Baumes und eine Variable „ok“ nimmt, die bestimmt, ob der Baum auch Blüten enthalten soll.

Nachdem man die Farbe bestimmt, bildet man einen Zylinder und beginnt das Zeichnen von der Spitze. Gleich danach wird der nächste Zylinder kürzer und dünner, weil man zu den Ästen übergeht. Die Funktion wird noch dreimal aufgerufen in drei verschiedenen Richtungen. Das wird wiederholt, solange das Zylinder größer als 1 ist, damit es nicht eine unendliche Schleife wird.

```
private static void maketree(float height, float Base, float angle,
{
    Gl.glPushMatrix();
    Gl.glPushAttrib(Gl.GL_LIGHTING_BIT);
    Gl.glColor3f(r,g,b);
    MakeCylinder(height, Base);
    Gl.glTranslatef(0.0f, height, 0.0f);

    if (height < 3 && ok != 0)
        leaves(height, lr, lg, lb);

    height -= height * 0.2f;
    Base -= Base * 0.3f;

    if (height>1)
    {
        Gl.glPushMatrix();
        Gl.glRotatef(angle, -1.0f, 0.0f, 0.0f);
        maketree(height, Base, angle, r, g, b, lr, lg, lb, ok);
        Gl.glPopMatrix();

        Gl.glPushMatrix();
        Gl.glRotatef(angle, 0.5f, 0.0f, 0.866f);
        maketree(height, Base, angle, r, g, b, lr, lg, lb, ok);
        Gl.glPopMatrix();

        Gl.glPushMatrix();
        Gl.glRotatef(angle, 0.5f, 0.0f, -0.866f);
        maketree(height, Base, angle, r, g, b, lr, lg, lb, ok);
        Gl.glPopMatrix();
    }
    Gl.glPopMatrix();
```

Abbildung 3: Funktion für den Baum

```
//STRUCT FOR EVERY TREE
struct Tree {
    public int x, y, z;
    public float rtree, gtree, btree;
    public float rleaves, gleaves, bleaves;
    public int leavesok;
    public float angle;
    public int total;
    public int[] wegarray;
    public int wegtotal;


}; static Tree[] T = new Tree[50];
```

Abbildung 4: Struktur für den Baum

Weil man verschiedene Bäume einfügen kann, wird eine Struktur und ein Vektor dem Typ Struktur definiert, dass die Koordinaten des Baumes speichert, die Farbe des Baumes und der Blüten, den Rekursionswinkel, die ok Variable, die bestimmt, ob der Baum Blüten hat, den Vektor für den Dijkstra Weg und eine Variable, die die Länge dieses Wegs bestimmt.

Sträucher

Die Funktion für das Darstellen der Sträucher ist ähnlich mit der der Bäumen. Sie nimmt als Parameter die Länge des ersten Astes, den Durchmesser der Basis und den Rekursionswinkel. Es wird das erste Zylinder gezeichnet und es wird von der Spitze begonnen. Man zeichnet die Blätter als Würfel. Der Zylinder wird nachher wieder kürzer und dünner. Wenn die Höhe größer als 0.5 ist, werden weiter 3 Äste gezeichnet, in 3 verschiedenen Richtungen. Auch hier gibt es eine Struktur für die Sträucher, die die Koordinaten speichert.

```
private static void makebush(float height, float Base, float angle)
{
    Gl.glPushMatrix();

    Gl.glPushAttrib(Gl.GL_LIGHTING_BIT);
    Gl.glColor3f(0, 0.3f, 0);

    MakeCylinder(height, Base);
    Gl.glTranslatef(0.0f, height, 0.0f);

    Gl.glPushMatrix();
    Gl.glColor3f(0, 0.3f, 0);                          //color
    Glut.glutSolidCube(0.35);                            //1 leaf
    Gl.glPopMatrix();

    height -= height * 0.2f;                                    //m
    Base -= Base * 0.3f;                                        //m

    if (height > 0.5)                                            /
    {

        Gl.glPushMatrix();
        Gl.glRotatef(angle, -1.0f, 0.0f, 0.0f);
        makebush(height, Base, angle);          //recursive call
        Gl.glPopMatrix();

        Gl.glPushMatrix();
        Gl.glRotatef(angle, 0.5f, 0.0f, 0.866f);
        makebush(height, Base, angle);          //recursive call
        Gl.glPopMatrix();

        Gl.glPushMatrix();
        Gl.glRotatef(angle, 0.5f, 0.0f, -0.866f);
        makebush(height, Base, angle);           //recursive call
```

Abbildung 5: Funktion für Sträucher

Gras

Die Funktion für das Gras ist die dritte rekursive Funktion in diesem Projekt. Sie nimmt als Parameter die Höhe des Zylinders, den Durchmesser der Basis und den Rekursionswinkel. Nachdem die Farbe auf grün gestellt wird, zeichnet man den Zylinder und beginnt von der Höhe. Der Zylinder wird nachher dünner und kürzer. Falls die Höhe größer als 0.2 ist, wird die Funktion noch einmal aufgerufen, um das Gras weiter krumm zu zeichnen.

```
private static void makegrass(float height, float Base, float angle)
{
    Gl.glPushMatrix();
    Gl.glColor3f(0, 0.3f, 0);
    MakeCylinder(height, Base);                                  //ma
    Gl.glTranslatef(0.0f, height, 0.0f);
    height -= height * 0.2f;                                     //ma
    Base -= Base * 0.3f;
    if (height > 0.2)
    {
        Gl.glPushMatrix();
        Gl.glRotatef(angle, -1.0f, 0.0f, 0.0f);
        makegrass(height, Base, angle);
        Gl.glPopMatrix();
    }
    Gl.glPopMatrix();
}
```

Abbildung 6: Funktion für das Gras

Teiche

Die Teiche erscheinen als kleine Kreise auf dem Bildschirm. Diese werden nicht rekursiv erzeugt und die Funktion dafür nimmt keine Parameter. Sie haben die vorbestimmte Farbe blau. Sie haben als Basis auch einen Zylinder, der aber nicht zu hoch ist. Auf dem Kontur des Zylinders wird darauf ein Kreis gezeichnet und zwar aus mehreren Linien sehr eng eineinander, was auf 360 Grad wiederholt wird.

Wie bei den anderen Elementen, werden die Koordinaten der Teiche in einem Vektor von Typ Struktur gespeichert.

Neue Teiche werden mit der Taste x eingefügt.

```
private static unsafe void makelake()
{
    float color = 1.0f;
    double x = 0.0f;
    double y = 0.0f;
    float angle = 0.0f;
    float angle_stepsize = 0.1f;
    Glu.GLUquadric qobj1;
    qobj1 = Glu.gluNewQuadric();

    Gl.glPushMatrix();                                  //dr
    Gl.glColor3f(0, 0, 1);
    Gl.glRotatef(-90, 1.0f, 0.0f, 0.0f);
    Glu.gluCylinder(qobj1, 5.0f, 5.0f, 0.1f, 20, 20);
    Gl.glPopMatrix();

    Gl.glRotatef(-90, 1.0f, 0.0f, 0.0f);            //draw
    Gl.glBegin(Gl.GL_POLYGON);
    angle = 0.0f;
    while (angle < 2 * 3.1415)
    {
        x = 5 * Math.Cos(angle);
        y = 5 * Math.Sin(angle);
        Gl.glColor3f(0, 0, color);
        color -= 0.01f;
        Gl.glVertex3f((float)x, (float)y, 0.1f);
        angle = angle + angle_stepsize;
    }
    Gl.glVertex3f(5, 0.0f, 0.1f);
    Gl.glEnd();
}
```

Abbildung 7: Funktion für Teiche

Dijkstra

Bei dem Algorithmus von Dijstra werden die Anzahl der Bäume (der Knoten), die Anzahl der Verbindungen (der Kanten), welche diese sind, welche Kosten sie haben und der Ausgangsbaum (Ausgangsknoten) von der Tastatur eingelesen. Es wird eine Matrix mit den Kosten laut den Kanten aufgebaut. Laut der Logik des Algorithmus sucht man in der Matrix Umwege von dem Ausgangsbaum zu allen Anderen, falls sie kürzer als die direkte Distanz sind. Das wird im D Vektor gespeichert.

```
//Suchen von Umwegen
for (i = 1; i <= n; i++)
{
    mini = 2000;
    for (j = 1; j <= n; j++)
        if (S[j] == 0)
            if (D[j] < mini)
            {
                mini = D[j];
                poz = j;
            }
    S[poz] = 1;
    for (j = 1; j <= n; j++)
        if (S[j] == 0 && A[poz, j] != 2000)
            if (D[j] > D[poz] + A[poz, j])
            {
                D[j] = D[poz] + A[poz, j];
                TD[j] = poz;
            }
```

Abbildung 8: Dijkstra

Diese Distanzen werden im Plan durch die Funktion drawDijkstra eingefügt. Jeder Baum hat einen Vektor mit dem Weg und die Länge des Weges. Die Linien werden in Bezug auf die Koordinaten der Bäume, die im Vektor für den Weg erscheinen gezeichnet.

```
private static void drawDijkstra()
{
    //Farbe jedes Weges
    float rd=0, gb=0, bd=0.5f;

    //Koordinaten der Linien (damit sie nicht ubereinstimmen)
    float mx=-0.5f, my=-0.5f, mz=-0.5f;

    for (int i = 1;  i<=n; i++)
    {
        if (lineok != 0 && i!=ausgangsk)
        {
            for (int w=1; w<T[i].wegtotal-1; w++)
            {
                Gl.glEnable(Gl.GL_LINE_SMOOTH);
                Gl.glBegin(Gl.GL_LINES);
                Gl.glColor3f(rd, gb, bd);
                Gl.glLineWidth(30);

                //Es werden Linien gezeichnet zwischen zwei Baumen im Weg
                Gl.glVertex3f(T[T[i].wegarray[w]].x+mx, T[T[i].wegarray[w]].y+my, T[T[i].wegarray[w]].z+mz);
                Gl.glVertex3f(T[T[i].wegarray[w+1]].x+mx, T[T[i].wegarray[w+1]].y+my, T[T[i].wegarray[w+1]].z+mz
                Gl.glEnd();
                Gl.glDisable(Gl.GL_LINE_SMOOTH);
            }


        }
         rd += 0.1f;
         mx += 0.5f;
         my += 0.5f;
         mz += 0.5f;
    }
```

Abbildung 9: Dijkstra im Plan

Bibliographie

Schulbuch

Grafikprojekt der 10. Klasse

https://gist.github.com/ashwin/5803190

https://www.codeproject.com/Articles/23778/OpenGL-3D-Navigation2-With-Tao-and-C-Tao-OpenGL-Ta

https://www.youtube.com/watch?v=LcHCygwIgLo&t=344s

https://www.youtube.com/watch?v=xNGmQ6lO6NY