

Durchschnitte von Polygonen

Autor
M. Opprecht, 199a

Betreuer
Prof. S.F. Keller

HSR Hochschule für Technik Rapperswil
Oberseestrasse 10
Postfach 1475
CH-8640 Rapperswil
T ++41 (0)55 222 41 11
F ++41 (0)55 222 44 00

Versionskontrolle

<i>Datum</i>	<i>Beschreibung</i>
10.01.02	Draft
03.02.02	Bis Kapitel „Verschnitt von Simplen Polygonen“ fertig
06.02.02	Letzte Kapitel fertiggestellt.

Inhaltsverzeichnis

Einleitung	3
Vorwort	3
Auftrag	3
Dokument	3
Schnittpunkt zweier Segmente	4
Literatur	4
SweepLine Algorithmus	5
Erklärung	5
Beispiel	6
Pseudo-Code	7
Literatur	8
Polygone	9
Nonsimple polygons	9
Simple polygons	9
Convex polygons	9
Verschnitt von konvexen Polygonen	10
O'Rourke, Chien, Olson and Naddor – 1982	10
Toussaint – 1985	12
Verschnitt von Simplen Polygonen	15
Weiler-Atherton	15
Vatti	17
Weitere	18
Literatur	18
Bibliotheken	20
ClipPoly	20
General Polygon Clipper (gpc)	20
Java Topology Suite (JTS)	20
Weiterführende Literatur	21
Glossar	22

Einleitung

Vorwort

An der Hochschule für Technik Rapperswil muss sich im 5 Semester jeder Student der Abteilung Informatik in ein Thema / Problem einarbeiten und Resultate niederschreiben und vortragen.

Auftrag

Das Problem der Durchschnittsberechnung von Polygonen (Vielecken) stellt sich in der Informatik zum Beispiel bei der Geovisualisierung oder bei der graphischen Darstellung dreidimensionaler Objekte auf einem zweidimensionalen Medium. Bei der graphischen Darstellung von Daten aus Geoinformationssystemen treten häufig Verschnittprobleme auf. Verschiedenste Themen, zum Beispiel Topologie, Wohngebiet, Gemeindegrenzen, Abdeckung von Natelantennen, werden miteinander dargestellt und verschnitten um neue Informationen zu gewinnen. Da ein solches Thema aus tausenden von Komponenten bestehen kann, und die Darstellung in akzeptabler Zeit erfolgen muss, sind effiziente Algorithmen gefragt. Einige dieser Algorithmen sollen besprochen werden.

Dokument

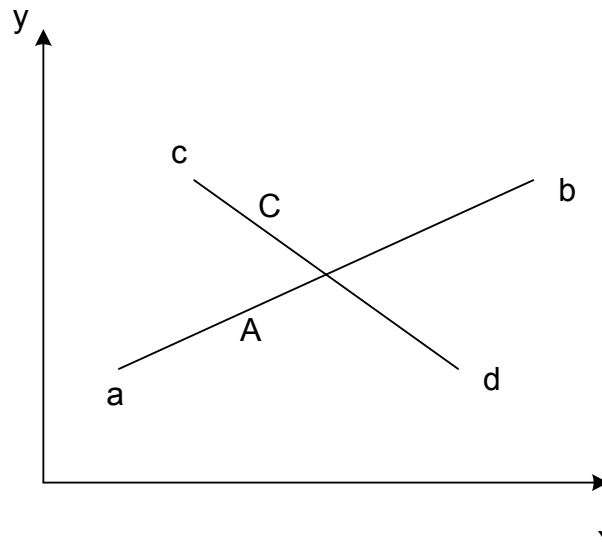
Alle Algorithmen für das Verschneiden von Polygonen reduzieren sich schlussendlich auf das Verschneiden von zwei Kanten. Deshalb beginnt dieses Dokument mit einer Erklärung zum Verschneiden von Segmenten.

Als weiteres wird auf die Polygone im Allgemeinen eingegangen. Die Kategorisierung der Polygone wird diskutiert.

Das Verschneiden von Polygonen bilden den Hauptteil dieses Dokumentes. Zuerst wird auf das Verschneiden von konvexen Polygonen eingegangen, bevor Algorithmen für beliebige Polygone diskutiert werden.

Der Abschluss bildet eine Auflistung von momentan vorhandenen Bibliotheken für das Verschneiden von Polygonen.

Schnittpunkt zweier Segmente



Figur: Zwei Segmente A und C

Der Schnittpunkt zweier Segmente kann einfach berechnet werden. Man betrachte die beiden Segmente zuerst als Linien, welche nach der Formel

$$p(s) = \vec{a} + s * \vec{A}$$

bzw.

$$q(t) = \vec{c} + t * \vec{C}$$

beschrieben werden kann. A, C, a und c sind Vektoren, s und t sind Streckungsfaktoren.

Die beiden Gleichungen können nach s und t aufgelöst werden.

$$s = a_0 * (c_1 - d_1) + c_0 * (d_1 - a_1) + d_0 * (a_1 - c_1) / D$$

$$t = \dots$$

wobei

$$D = a_0 * (c_1 - d_1) + b_0 * (d_1 - c_1) + c_0 * (b_1 - a_1) + d_0 * (a_1 - b_1)$$

Ist D Null, so sind die beiden Segmente parallel. Haben die beiden Streckungsfaktoren s und t einen Wert zwischen 0 und 1, so schneiden sich die beiden Segmente.

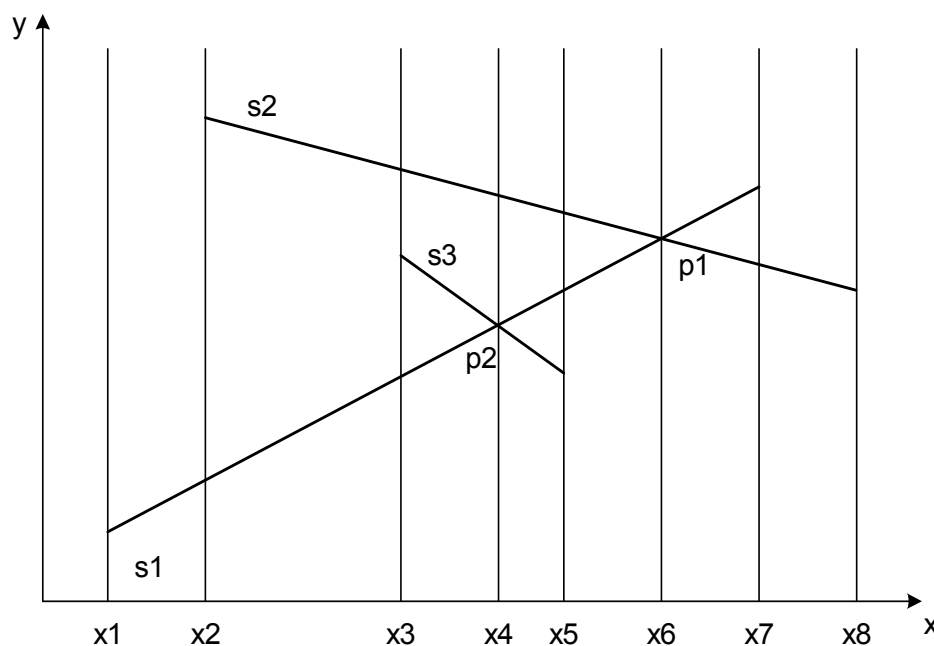
Literatur

Computational Geometry in C; Joseph O'Rourke, ISBN 0-521-44592-2

SweepLine Algorithmus

Der SweepLine Algorithmus findet aus einer Sammlung von Segmenten die Schnittpunkte. Dieser Algorithmus arbeitet in der Zeit $O(N \cdot \log(N))$. Nach der „Brute-Force“ Methode, bei der alle Segmente miteinander auf Schnittpunkte überprüft werden, arbeitet in der Zeit $O(N^2)$.

Erklärung



Der Geschwindigkeitsvorteil kommt daher, dass nur anliegende Segmente auf einen möglichen Schnittpunkt überprüft werden.

Um den Zustand „anliegend“ einführen zu können, muss zuerst eine Ordnung eingeführt werden. In diesem Beispiel ist der Abstand zur x -Achse die gewünschte Ordnung. Beim Punkt x_3 liegt das Segment s_1 an s_3 , aber nicht an s_2 . Beim Punkt x_5 liegt s_1 an s_2 .

Wir führen nun eine Eventliste ein. Ein Event ist ein Start- oder Endpunkt eines Segmentes oder ein Schnittpunkt zweier Segmente. In der Eventliste werden alle Events, sortiert nach der x -Koordinate eingetragen. Selbstverständlich können die Schnittpunkte noch nicht eingetragen werden, da sie noch nicht bekannt sind.

Eventliste beim Start
 $s_1.SP, s_2.SP, s_3.SP, s_3.EP, s_1.EP, s_2.EP$
SP = StartPoint
EP = EndPoint

In einer zweiten Liste, die Statusliste, führen wir den aktuellen Zustand der Segmente untereinander zum Zeitpunkt eines Events.

Die Eventliste wird nun abgearbeitet. Der erste Event wird aus der Liste entnommen und der neue Zustand in der Zustandsliste ermittelt und gespeichert. Dabei können Schnittpunkte nur auftreten, wenn zwei Segmente aneinander zuliegen kommen.

Beispiel

Grundlage ist obige Zeichnung.

SP ist der Startpunkt, EP ist der Endpunkt des Segmentes.

	<i>Eventliste</i>	<i>Zustandsliste</i>	<i>Action</i>
	s1.SP, s2.SP, s3.SP,s3.EP,s1.EP,s2.EP	-	
x1	s1.SP, s2.SP, s3.SP,s3.EP,s1.EP,s2.EP	s1	Es kann noch kein Schnittpunkt auftreten, da erst ein Segment aktuell ist.
x2	s2.SP ,s3.SP,s3.EP,s1.EP,s2.EP	s1, s2	Die beiden Segmente werden auf einen möglichen Schnittpunkt überprüft. Schnittpunkt p1 wird in Eventliste eingetragen.
x3	s3.SP ,s3.EP,p1,s1.EP,s2.EP	s1, s3, s2	Die Segmentpaare s1 – s3 sowie s3 -s2 werden auf Schnittpunkte überprüft. Schnittpunkt p2 wird in Eventliste eingetragen.
x4	p2,s3.EP,p1,s1.EP,s2.EP	s3, s1, s2	Die Segmente s1 und s3 haben die Plätze getauscht. Die neuen Nachbarn s1 und s2 werden auf einen möglichen Schnittpunkt überprüft. Der bereits gefundene Schnittpunkt p2 wird nicht in der Eventliste eingetragen.
x5	s3.EP,p1,s1.EP,s2.EP	s1, s2	Da s1 keinen neuen Nachbar bekommen hat, wird nichts überprüft.
x6	p1, s1.EP, s2.EP	s2, s1	Die Segmente s1 und s2 haben die Plätze getauscht.
x7	s1.EP, s2.EP	s2	Das Segment s2 hat keinen neuen Nachbar bekommen.
x8	s2.EP	-	-

Pseudo-Code

```

Segment_Intersection
{
    sortiere_die_Endpunkte_nach_x_und_y_und_füge_diese_der_Eventliste_hinzu();

    Liste temporäre_Schnittpunkte;

    while(Eventlise != leer)
    {
        p = Eventliste.First();

        if( p ist ein Startpunkt )
        {
            s = p.gibSegment();
            Statusliste.add( s );
            if( s.schneidet_sich_mit( s.gibOberesSegment() ) )
            {
                temporäre_Schnittpunkte.add( Schnittpunkt );
            }
            if( s.schneidet_sich_mit( s.gibUnteresSegment() ) )
            {
                temporäre_Schnittpunkte.add( Schnittpunkt );
            }
        }

        else if( p ist ein Endpunkt )
        {
            s = p.gibSegment();
            sOben = s.gibOberesSegment();
            sUnten = s.gibUnteresSegment();

            if( sOben.schneidet_sich_mit( sUnten ) )
            {
                temporäre_Schnittpunkte.add( Schnittpunkt );
            }

            Statusliste.delete( s );
        }

        else if( p ist ein Schnittpunkt )
        {
            (s1, s2) = p.gibSegmente(); // s1 ist oberhalb von s2

            s3 = s1.gibOberesSegment();
            s4 = s2.gibUnteresSegment();

            if( s3.schneidet_sich_mit( s2 ) )
            {
                temporäre_Schnittpunkte.add( Schnittpunkt );
            }
            if( s4.schneidet_sich_mit( s1 ) )
            {
                temporäre_Schnittpunkte.add( Schnittpunkt );
            }
        }
    }

    while( temporäre_Schnittpunkte != leer )
    {
        p = temporäre_Schnittpunkte.First();

        if( Eventliste.ist_schon_vorhanden( p ) == Falsch )
        {
            Eventliste.add( p ); // Neuer Schnittpunkt gefunden !!
        }
    }
}

```

Literatur

Computational Geometry an Introduction; Franco P. Preparata, Michael IanShamos, ISBN 0-387-96131-3

Algorithms – Second Edition; Robert Sedgewick, ISBN 0-201-06673-4

Erklärung mit Animation:

<http://www.lems.brown.edu/~wq/projects/cs252.html>

Applet für Line-Segment Intersection:

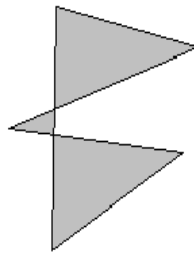
<http://www.cs.brown.edu/courses/cs252/proj/src/Spr98/ya>

Polygone

Ein Polygon wird definiert durch eine endliche Anzahl von Kanten, so dass jeder Endpunkt einer Kante geteilt wird von genau einer anderen Kante.

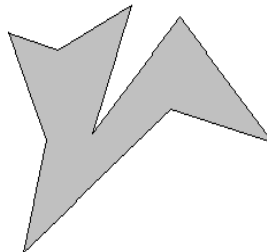
Nonsimple polygons

Beliebige Polygone, deren Seiten sich auch überlappen dürfen.



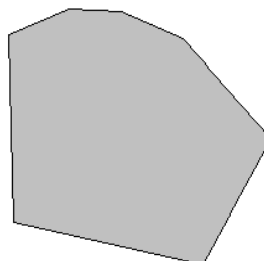
Simple polygons

Beliebige Polygone, deren Seiten sich aber nicht schneiden dürfen.



Convex polygons

Polygone, deren Seiten sich nicht überschneiden und der innere Winkel zwischen zwei Seiten ist immer kleiner als 180 Grad.



Verschnitt von konvexen Polygonen

Unter Verschnitt versteht man das Überlagern zweier Polygone (A & B) und das Berechnen der vier möglichen Schnittmengen beziehungsweise Flächen:

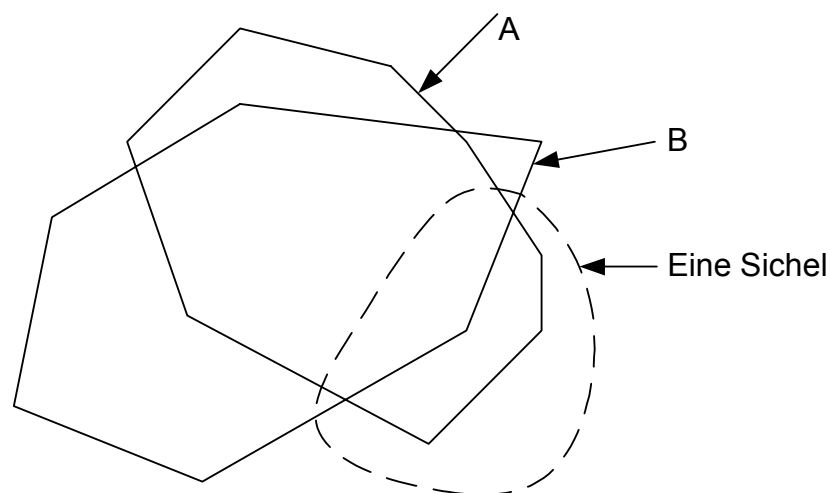
1. A und B
2. A oder B
3. A ohne B
4. B ohne A

Es gibt verschiedene Algorithmen, welche das Problem des Verschnittes von zwei Polygonen lösen. Der Zeitaufwand für das Berechnen der Flächen kann bei beliebigen Polygonen quadratisch sein $O(N * M)$. Der Zeitaufwand bei konvexen ist dagegen bei geeigneten Algorithmen linear $O(N + M)$. Die Bezeichner N beziehungsweise M stehen für die Anzahl Ecken der Polygone.

O'Rourke, Chien, Olson and Naddor – 1982

Im folgenden wird ein Algorithmus von O'Rourke erläutert. Dieser Algorithmus arbeitet in linearer Zeit.

Die Umhüllende des Verschnittes (A und B) besteht abwechslungsweise aus einem Fragment des Polygons A und des Polygons B. Jedes dieser Fragmente des einen Polygons wird nun von einem Fragment des anderen Fragmentes umhüllt. Ein Paar solcher Fragmente wird „Sichel“ genannt. Die beiden Endpunkte einer Sichel sind Schnittpunkte der beiden Polygone.



Man denkt sich nun auf jedem Polygon eine aktuelle Seite. Wenn sich die beiden Seiten schneiden, hat man einen Schnittpunkt und somit einen Punkt des Verschnittes und einen Endpunkt einer Sichel gefunden. Man schreitet nun auf den beiden Polygonen fort und merkt sich alle Punkte des einen Polygons, bis wieder ein Schnittpunkt auftritt. Nach einem

Schnittpunkt schreitet man gleich fort, merkt sich diesmal aber die Punkte des anderen Polygons.

Der Algorithmus des „Fortschreitens“ genügt folgenden Regeln. Die aktuellen Seiten werden hier als Vektoren betrachtet und mit den Grossbuchstaben A und B gekennzeichnet. Der Endpunkt (der beim Pfeil) des jeweiligen Vektors ist mit einem Kleinbuchstaben gekennzeichnet. Die Funktion H() bezeichnet die Halbfläche, begrenzt durch die angegebene Polygoneite, die den Verschnitt beinhaltet.

<i>Kreuzprodukt $A \times B$</i>	$a \in H(B)$	$b \in H(A)$	<i>Fortschreiten auf...</i>
> 0	WAHR	WAHR	A
> 0	WAHR	FALSCH	A oder B
> 0	FALSCH	WAHR	A
> 0	FALSCH	FALSCH	B
< 0	WAHR	WAHR	B
< 0	WAHR	FALSCH	B
< 0	FALSCH	WAHR	A oder B
< 0	FALSCH	FALSCH	A

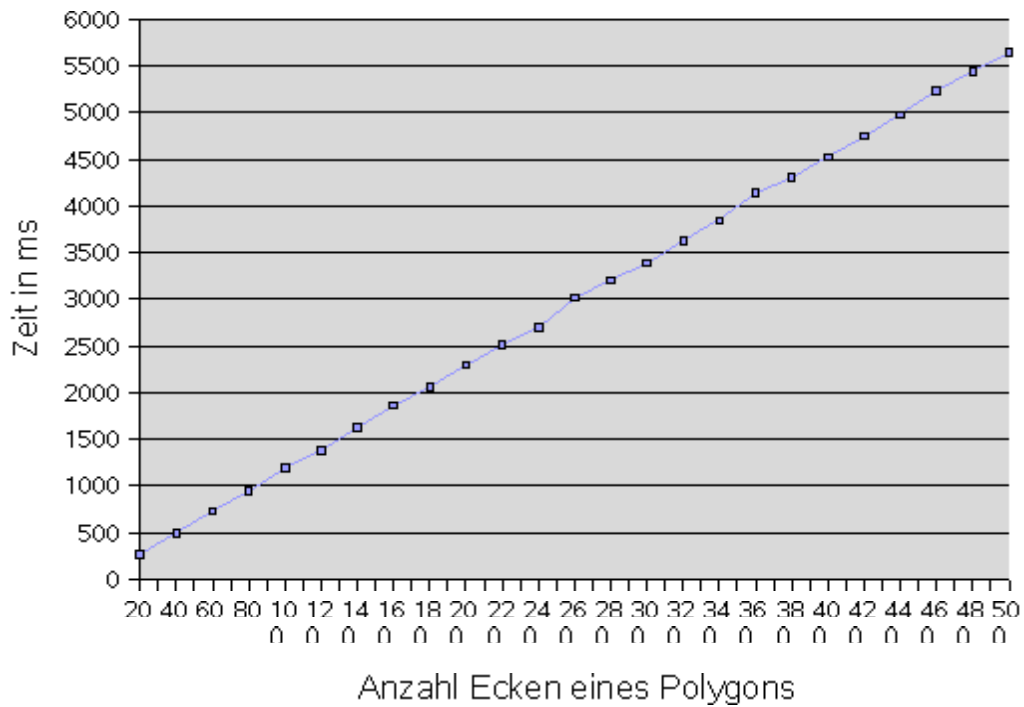
Geschwindigkeitstest

Diesen Algorithmus habe ich vollständig in Java implementiert.

Um die Geschwindigkeit messen zu können, musste ich einen Generator schreiben, welcher mir konvexe Polygone generiert.

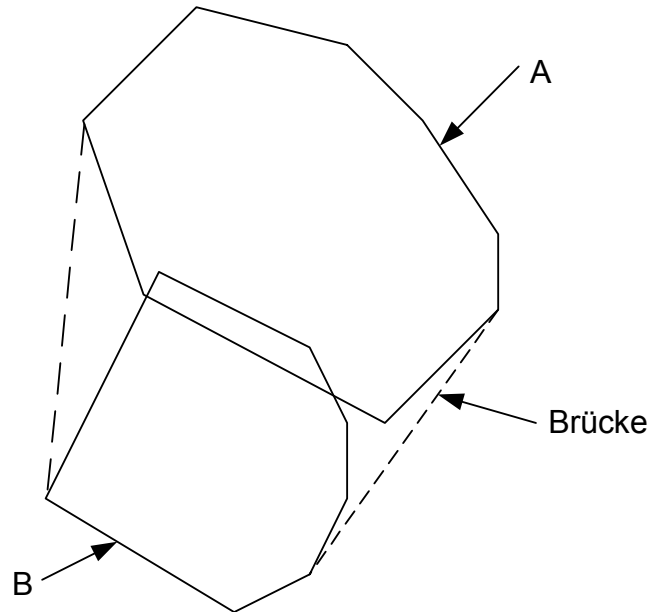
<i>Anzahl Ecken</i>	<i>Zeit in ms</i>	<i>Anzahl Ecken</i>	<i>Zeit in ms</i>
20	270	280	3205
40	491	300	3385
60	731	320	3625
80	951	340	3845
100	1192	360	4146
120	1382	380	4307
140	1622	400	4526

<i>Anzahl Ecken</i>	<i>Zeit in ms</i>	<i>Anzahl Ecken</i>	<i>Zeit in ms</i>
160	1863	420	4747
180	2063	440	4977
200	2293	460	5237
220	2514	480	5438
240	2694	500	5648
260	3014		



Toussaint – 1985

Einen völlig anderen Ansatz wählte Toussaint bei seinem Algorithmus, welcher ebenfalls mit linearem Zeitaufwand arbeitet.



Als erster Schritt wird die „Convex Hull“, also die Umhüllende um die beiden Polygone gelegt. Wie man eine solche Umhüllende berechnet, wird hier nicht erläutert. Es sei auf weiterführende Literatur verwiesen.

Wenn sich die beiden Polygone überlappen, so gibt es zwangsweise „Brücken“ auf dieser Umhüllende, welche die beiden Polygone verbindet. Diese Brücken sind Segmente, mit den Endpunkten auf je einem Polygon. Dabei ist sichergestellt, dass die Brücke nie zwei Eckpunkte auf dem selben Polygon verbindet, da sonst das Polygon nicht konvex wäre.

Sind alle Brücken gefunden, so können die Schnittpunkte leicht berechnet werden. Dazu wird folgender Algorithmus gebraucht, welcher ebenfalls mit linearem Zeitaufwand arbeitet und hier als Pseudocode angegeben ist.

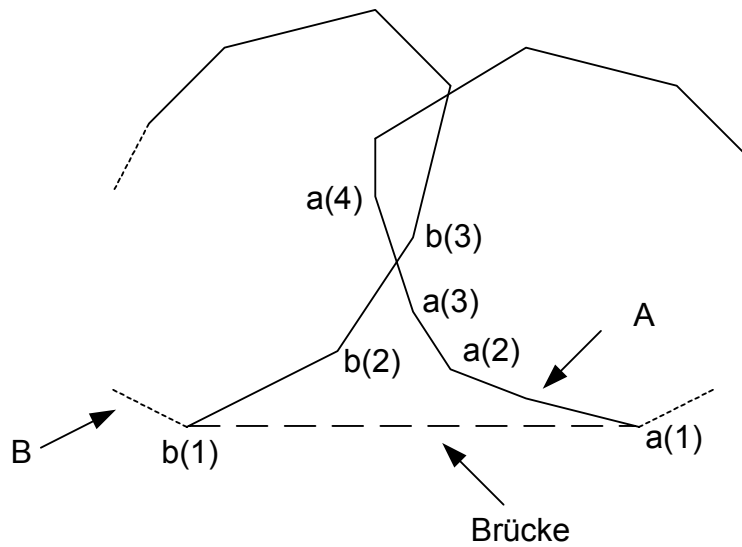
```

int i = 1;
int j = 1;
boolean finished;

do {
    finished = true;

    while( leftTurn( a(i), a(i + 1), b(j + 1) )) {
        j++;
        finished = false;
    }
    while( rightTurn( b(j), b(j + 1), a(i + 1) )) {
        i++;
        finished = false;
    }
} while( ! finished )

```



Nach Abschluss des Algorithmus' schneiden sich die beiden aktuellen Segmente $a(i) - a(i + 1)$ beziehungsweise $b(j) - b(j + 1)$.

Literatur

Computational Geometry an Introduction; Franco P. Preparata, Michael IanShamos, ISBN 0-387-96131-3

Computational Geometry in C; Joseph O'Rourke, ISBN 0-521-44592-2

Erläuterung von Toussaint mit Applet: <http://cgm.cs.mcgill.ca/~godfried/teaching/cg-projects/97/Plante/CompGeomProject-EPlante/algorithm.html>

Verschnitt von Simplen Polygonen

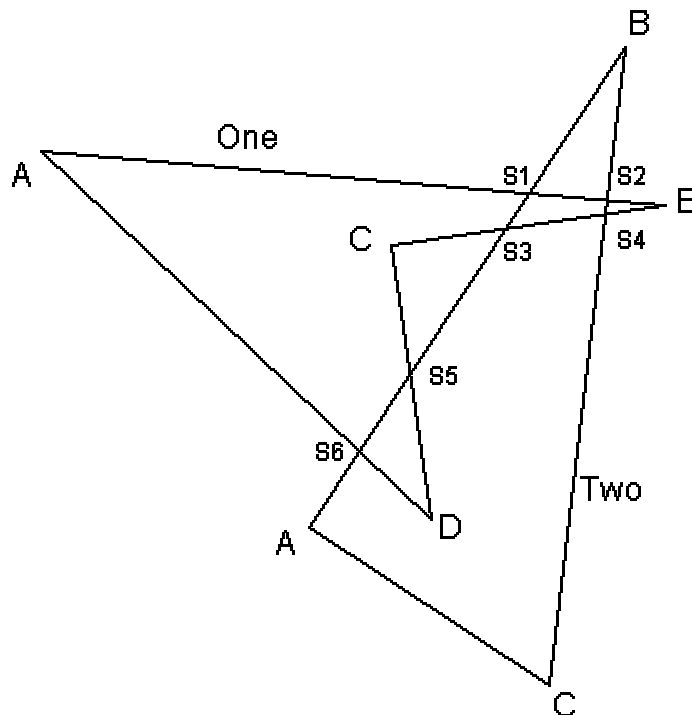
Weiler-Atherton

Der Weiler-Atherton Algorithmus beherrscht Simple Polygone, Enklaven ('Löcher') und alle vier mögliche Schnittmengen (A und B, A oder B, A ohne B, B ohne A).

Der Weiler-Atherton Algorithmus wurde 1980 entwickelt. Es war der erste Algorithmus für den Verschnitt von beliebigen Polygone.

Dieser Algorithmus arbeitet wie alle Algorithmen für beliebige Polygone mit einem quadratischen Zeitaufwand $O(N * M)$.

Erklärung



Erster Schritt: Vorbereitung

Wir führen zwei Listen ein. Je eine für die beiden Polygone (One & Two). In den Listen sind die Eckpunkte der Polygone dem Uhrzeigersinn entsprechend eingeordnet.

Zweiter Schritt: Berechnen der Schnittpunkte

Nun wird jede Kante des Polygons 'One' mit den Kanten des Polygons 'Two' auf einen Schnittpunkt untersucht. Wird ein Schnittpunkt gefunden, so wird in beiden Listen ein neuer Schnittpunkt eingefügt, der zwischen den Endpunkten der jeweiligen Kante liegt. Dieser Schnittpunkt muss uns ermöglichen, von einer Liste in die andere wechseln zu können.

Man muss beachten, dass auf einer Kante mehrere Schnittpunkte auftreten können. Daher müssen die gefundenen Schnittpunkte innerhalb einer Kante ebenfalls sortiert eingefügt werden.

Dritter Schritt: Kategorisierung

Wir durchlaufen nun alle Punkte des Polygons 'One'. Dabei werden allen im zweiten Schritt gefundenen Schnittpunkten ein Status zugewiesen.

<i>Status</i>	<i>Beschreibung</i>
Entering	Das aktuelle Polygon tritt in das andere Polygon ein.
Exiting	Das aktuelle Polygon tritt aus dem anderen Polygon aus.

Wir speichern die kategorisierten Schnittpunkte in zwei Listen - eine für 'Entering'-Schnittpunkte und eine für 'Exiting'-Schnittpunkte.

Die Listen sollten nun so aussehen.

<i>Liste für Polygon 'One'</i>	<i>Liste für Polygon 'Two'</i>	<i>'Entering'-Liste</i>
A	A	S1
S1 (Entering)	S6	S4
S2 (Exiting)	S5	S5
B	S3	
S4 (Entering)	S1	
S3 (Exiting)	B	
C	S2	
S5 (Entering)	S4	
D	C	
S6 (Exiting)		

Vierter Schritt: Auswertung

Man nehme einen Schnittpunkt aus der 'Entering'-Liste. Wechsle zu diesem Schnittpunkt auf dem Polygon 'One'. Wandere nun weiter bis ein weiterer Schnittpunkt gefunden wird. Wechsle nun durch diesen Schnittpunkt zum anderen Polygon und wandere weiter bis zum nächsten Schnittpunkt. Das gesuchte Polygon ist gefunden, wenn der Schnittpunkt wieder dem Ausgangsschnittpunkt entspricht.

Ein erstes Polygon des Verschnitts ist nun berechnet. Anhand der 'Entering'-Liste können die restlichen Polygone berechnet werden.

Der vierte Schritt in Pseudo-Code:

```

/* Gegeben:
ListeOne: Liste mit Eckpunkten und Schnittpunkten des Polygons 'One'
ListeTwo: Liste mit Eckpunkten und Schnittpunkten des Polygons 'Two'
EnteringListe: Liste mit allen Entering-Schnittpunkten des Polygons 'One'
*/

void start(void) {

while( ! EnteringListe.empty() )
{
StartPoint = EnteringListe.first();
EnteringListe.delete( CurrentPoint );
CurrentList = ListeOne;
CurrentList.setCurrentPoint( StartPoint );
Print( „Polygon start“ );

do
{
NextPoint = CurrentList .next();
if( NextPoint is_a EnteringPoint )
{
EnteringListe.delete( NextPoint );
ChangeList();
CurrentList.setCurrentPoint( NextPoint );
}
Print( NextPoint );
} while( NextPoint != StartPoint );

Print( „Polygon end“ );
}
Print( „finished“ );
}

```

Vatti

Der Vatti-Algorithmus ist ein weiterer Algorithmus für Simple Polygone. Er wurde 1992 von B. R. Vatti veröffentlicht.

Leider habe ich keine Erläuterung zu diesem Algorithmus gefunden.

Die freie C Bibliothek gpc ist eine Implementierung von Vatti.

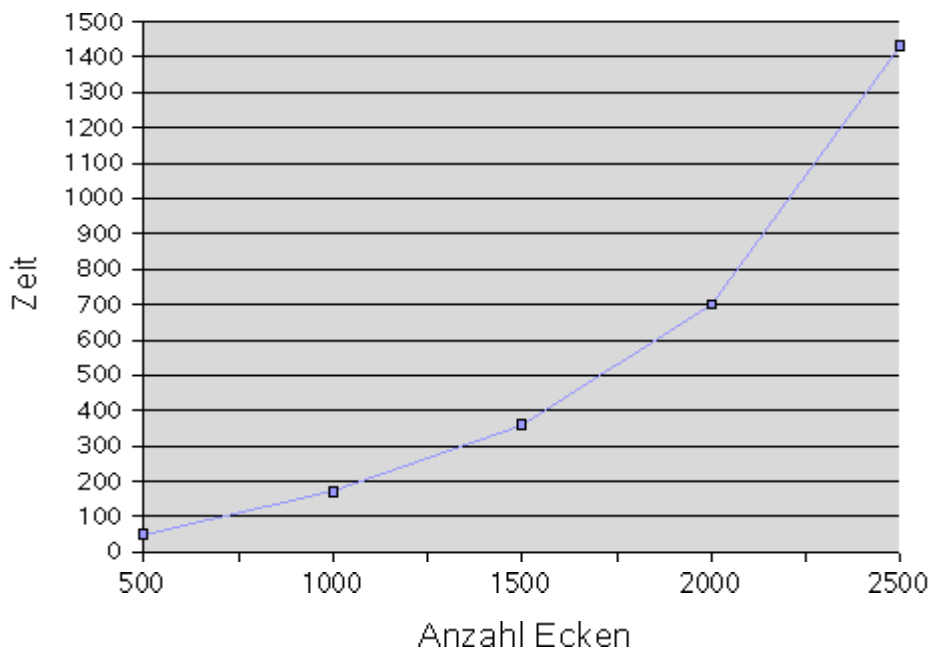
Geschwindigkeitstest

<i>Anzahl Ecken</i>	<i>Zeit in ms</i>
---------------------	-------------------



<i>Anzahl Ecken</i>	<i>Zeit in ms</i>
500	50
1000	171
1500	360
2000	701
2500	1432

Vatti



Weitere

Auf dem Internet habe ich einen weiteren Algorithmus gefunden, der dem Weiler-Atherton Algorithmus' aber sehr ähnlich sieht.

Eine genaue Erklärung findet sich unter: <http://davis.wpi.edu/~matt/courses/clipping>

Literatur

Weiler-Atherton,
<http://cs1.bradley.edu/public/jcm/weileratherton.html>

Polygon Clipping,
http://bart.ihpca.psu.edu/practical_training/main/node9.html

Clipping Algorithms

<http://www.cs.uvm.edu/~snapp/teaching/CS274/lectures/clipping.pdf>

Klamer Schutte Algorithmus – Erweiterung von Weiler-Atherton

<http://www.ph.tn.tudelft.nl/~klamer/appendices/appendices.html>

Bibliotheken

ClipPoly

ClipPoly ist eine C++ Bibliothek die den Weiler-Atherton Algorithmus implementiert. Leider unterstützt diese Bibliothek keine Enklaven.

Die Bibliothek ist frei erhältlich unter:

http://www.ph.tn.tudelft.nl/People/klamer/clippoly_entry.html

General Polygon Clipper (gpc)

Wie bereits oben besprochen implementiert gpc den Vatti Algorithmus. Gpc unterstützt Enklaven und alle vier Arten von Schnittmengen (A und B, A oder B, A ohne B, B ohne A).

Die Bibliothek ist frei erhältlich unter:

<http://www.cs.man.ac.uk/aig/staff/alan/software//index.html>

Java Topology Suite (JTS)

JTS 100% in Java geschrieben und implementiert alle fundamentalen 2D Algorithmen. JTS ist die leistungsstärkste und komfortabelste Bibliothek der hier genannten.

JTS unterstützt alle mögliche Arten von geometrischen Objekten.

So zum Beispiel:

- (Multi-) Point
- (Multi-) Line
- (Multi-) Polygon

Die Bibliothek ist frei erhältlich unter:

<http://www.vividsolutions.com/jts/jtshome.htm>

Weiterführende Literatur

Applet von O'Rourke zu seinem Buch „Computational Geometry in C (Second Edition)“
<http://cs.smith.edu/~orourke/books/CompGeom/CompGeom.html>

Verschiedenste Tools für „Computational Geometry“
http://www.mathtools.net/MATLAB/Computational_geometry/index.html
http://www.mathtools.net/Java/Computational_geometry/index.html

Generation of Random Polygonal Objects
<http://www.cosy.sbg.ac.at/~held/projects/rpg/rpg.html>

Convex Hull Applet
<http://www.cse.unsw.edu.au/~lambert/java/3d/hull.html>

Glossar

<i>English</i>	<i>Deutsch</i>	<i>Erklärung</i>
Clipping	Ausschnitt	
Convex	Konvex	
Edge	Kante, Rand	
Hole	Enklave, Loch	Ein Bereich, der vollständig in einem Polygon integriert ist, aber nicht zum Polygon dazugehört.
Intersection	Durchschnitt	
Polygon	Polygon	Siehe Kapitel „Polygone“
Segment	Segment	Eine Linie mit Anfangs- und Endpunkt
Vertex	Scheitelpunkt	Zum Beispiel Eckpunkte bei konvexen Polygonen