Polygonverschneidung mit SQL[1]

(mit 7 Bildern und 2 Tabellen)

Von Sarah Tauscher und Karl Neumann, Braunschweig

ZUSAMMENFASSUNG: Einige räumliche Anfragen wie z.B. der Abstand zweier Punkte lassen sich recht direkt in SQL formulieren, und mit Aggregationen sind sogar Summenformeln wie z.B. Umfang und Fläche von Polygonen darstellbar. Weniger bekannt ist jedoch, dass sich auch viel komplexere geometrische Operationen als Anfragen mit Kern-SQL realisieren lassen. Im vorliegenden Text konzentrieren wir uns daher auf die anspruchsvolle GIS-Basisoperation der Polygonverschneidung und entwickeln schrittweise eine einzige (umfangreiche) SQL-Anfrage, die diese Funktion rein relational implementiert.

ABSTRACT: The formulation of some spatial queries like the distance between two points in SQL is straight forward and with aggregations even sum formulas like the perimeter of polygons can be expressed. Less known is the fact that even much more complex geometric operations are realizable as SQL queries. The focus of this paper lies on the challenging GIS base operation polygon overlay and we develop step by step one single (extensive) SQL query which implements this function purely relational.

1 Einleitung

Bereits in den 80er Jahren wurde festgestellt, dass relationale Datenbanken für die Speicherung und die Analyse räumlicher Daten nur bedingt geeignet sind. Die Speicherung der Daten in Relationen ist zwar möglich, doch ist für umfassende Bearbeitungs- und Analysemethoden zusätzliche Anwendungsprogrammierung notwendig, und dieser Ansatz hat sich als ineffizient herausgestellt (vgl. etwa *Schek/Scholl* 1986). Stattdessen setzten sich in diesem Bereich objektrelationale Datenbanken durch, bei denen das Datenbanksystem um räumliche Datentypen und Operationen erweitert wurde (siehe *Rigaux/Scholl/Voisard* 2002).

SQL weist als Anfragesprache für die Bearbeitung und Analyse erhebliche Nachteile auf, da räumliche Konzepte nicht integriert sind, so dass eine Reihe räumlicher Anfragesprachen vorgeschlagen wurden. Eine ausführliche Kritik sowie ein Überblick über verschiedene Ansätze gibt *Egenhofer 1992*. Die dortigen Kritikpunkte beziehen sich im Wesentlichen auf die fehlenden graphischen Ausgabemöglichkeiten und die sehr umständliche Anfrageformulierung. So ist es offensichtlich, dass einige räumliche Anfragen wie z.B. der Abstand zweier Punkte in Kern-SQL durchaus formulierbar sind und mit Aggregationen sogar Summenformeln wie z.B. Umfang und Fläche von Polygonen darstellbar sind. Aber Untersuchungen, welche räumlichen Fragestellungen tatsächlich mit SQL beantwortet werden können und wie effizient deren Auswertung ist, wurden bislang nicht durchgeführt. Der vorliegende Text behandelt daher Möglichkeiten der Umsetzung komplexerer Analysefunktionen in SQL am Beispiel der Flächenverschneidung.

[1] Erscheint in Mitteilungen des Bundesamtes für Kartographie und Geodäsie, Frankfurt M., 2010

Die Flächenverschneidung dient der Verknüpfung verschiedener Informationsschichten und kann somit als eine GIS-Basisoperation betrachtet werden. Ein typisches Beispiel ist die Verknüpfung von Landnutzungsdaten mit geologischen Daten. Der prinzipielle Ablauf der Flächenverschneidung ist folgender (vgl. *Hake/Grünreich/Meng* 2002):

- Identifizieren der Liniensegmente (Polygonseiten) unter Auswertung der Topologie, sofern diese nicht explizit gegeben sind.
- Mit Punkt-in-Fläche-Abfrage prüfen, ob Liniensegmente der einen Fläche innerhalb der anderen Fläche liegen.
- Mit den ermittelten Segmenten die Schnitte berechnen.
- Neue Segmente einschließlich ihrer topologischen Relationen bilden.
- Abschließend den Flächenobjekten die aus den semantischen Beschreibungen der Ausgangsflächen ermittelten Attribute zuordnen.

Es existiert eine Vielzahl verschiedener Algorithmen zur Polygonverschneidung, die zum Großteil aus der algorithmischen Geometrie stammen. Ein Vergleich zahlreicher Ansätze ist z.B. in *Andrews* et al. 1994 zu finden. Hierbei kann grundsätzlich zwischen Algorithmen, die auf topologischen Datenmodellen (z.B. dem Winged-Edge-Modell) arbeiten und denen, die einzelne Polygone betrachten, unterschieden werden.

Im Folgendem wird die Durchführbarkeit der Polygonverschneidung bei Speicherung der Daten in einer relationalen Datenbank und ausschließlicher Verwendung von Kern-SQL zur Berechnung der Schnittpolygone betrachtet. Die Polygone werden hierzu ohne explizite Topologie gespeichert. Zunächst wird dazu im Abschnitt 2 die konkrete Modellierung der Polygone erläutert und das zugehörige Relationenschema entworfen. Anschließend wird in Abschnitt 3 die Schnittberechnung und in Abschnitt 4 das Halbstrahlverfahren vorgestellt. Die Anfrage zur Bestimmung der Schnittpolygone mit diesen Voraussetzungen wird im 5. Abschnitt schrittweise aufgebaut. Anschließend folgt eine kurze Zusammenfassung sowie ein Ausblick auf weitere mögliche Untersuchungen.

2 Datenmodellierung

Die einfachste Möglichkeit Polygone in Relationen abzuspeichern, besteht in einer Liste von Stützpunkten. Hierzu wird neben dem Identifikator (Id) der Polygone und den X- und Y-Koordinaten der Stützpunkte mindestens ein weiteres Attribut benötigt, durch das die Reihenfolge der Stützpunkte festgelegt wird. Der Einfachheit halber sollten hierfür ganze Zahlen verwendet werden, die bei 1 beginnen und dicht liegen.

Da die konkreten Relationenschemata einen erheblichen Einfluss, nicht nur auf die Performance, sondern auch auf die Durchführbarkeit von Anfragen haben, ist es notwendig, schon während der Modellierung die geplanten Anfragen zu berücksichtigen. Die grundlegenden zur Polygonverschneidung notwendigen Berechnungen sind zum Einem die Bestimmung der Koordinaten der Schnittpunkte und zum Anderen die Ermittlung der Lage der Stützpunkte eines Polygons bezüglich der anderen Polygone. Beide Berechnungen sind

kantenorientiert. Zwar ist der Zugriff auf aufeinanderfolgende Stützpunkte und somit die Bestimmung einer Kante problemlos möglich, jedoch ist hierzu ein Self-Join der Polygon-Relation notwendig. Da die Kanten zudem häufig in der Anfrage zur Polygonverschneidung benötigt werden, ist es ratsam, diese Information redundant abzuspeichern. Die Positionsangabe, die bei der Speicherung von Kanten nicht mehr notwendig ist, um das Polygon zu bestimmen, sollte dennoch beibehalten werden, da hierdurch die Bestimmung der Ordnung der Schnittpunkte untereinander und in Bezug auf die Stützpunkte und somit die Konstruktion des Schnittpolygons vereinfacht wird (siehe Abschnitt 3).

Für die Berechnung der Schnittpunktkoordinaten ist es sinnvoll, darüber hinaus weitere Parameter der Geraden redundant abzuspeichern. Hierzu muss zunächst entschieden werden, welche Geradendarstellung zur Schnittberechnung verwendet werden soll: Die Geradengleichung $y = mx + b$ oder die Hessesche Normalenform $\binom{x}{y}\vec{n} = r$

Für die Geradengleichung wird zunächst die Steigung m und der Achsenabschnitt b benötigt, die sich aus den Stützpunkten berechnen lassen. Problematisch ist allerdings, dass für Parallelen zur Y-Achse weder die Steigung noch der Achsenabschnitt definiert ist. Folglich ist eine Darstelung in dieser Form nicht möglich, stattdessen lautet hier die Gleichung $x = c$. Somit verkompliziert sich nicht nur die Anfrage, sondern auch das Relationenschema müsste entweder um ein Attribut c erweitert werden, das für die Mehrzahl der Tupel nicht benötigt wird, oder b müsste abhängig davon, ob ein Wert für m existiert, entweder als Achsenabschnitt oder Wert für c interpretiet werden.

Im Gegensatz dazu ist die Hessesche Normalform für alle Geraden definiert. Als zusätzliche Attribute werden neben der X- und Y-Komponente des Normalenvektors der Wert $r = \vec{n}\vec{a}$ benötigt, wobei $\vec{a}$ der Startpunkt als Vektordarstellung ist. Somit ergibt sich folgendes Relationenschema für die Speicherung von Polygonen:

Polygon(<u>id</u>, edgeId, startX, startY, endX, endY, nx, ny, r)

Zur Veranschaulichung sind in Tabelle 1 die Einträge der Polygon-Relation für das in Bild 1 gezeigte Polygon eingetragen.

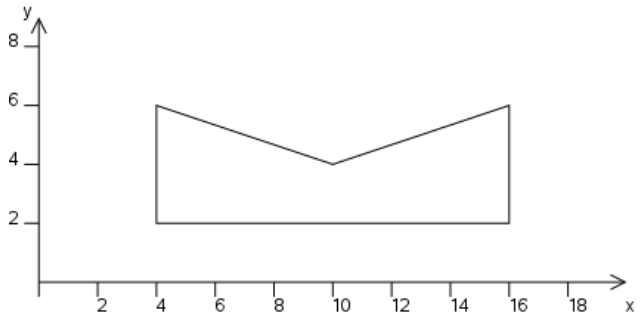

Bild 1 - Beispiel-Polygon.

id	edgeId	startx	starty	endx	endy	nx	ny	r
42	1	4	2	16	2	0	1	2
42	2	16	2	16	6	1	0	16
42	3	16	6	10	4	$1/\sqrt{10}$	$3/\sqrt{10}$	$22/\sqrt{10}$
42	4	10	4	4	6	$-1/\sqrt{10}$	$3/\sqrt{10}$	$2/\sqrt{10}$
42	5	4	6	4	2	1	0	4

Tabelle 1 – relationale Daten eines Polygons.

3 Berechnung der Schnittpunkte

Für die Bestimmung des Schnittpunktes zweier Strecken wird zunächst der Schnittpunkt der Geraden berechnet und anschließend überprüft, ob dieser auf beiden Strecken liegt. Zur Berechnung des Schnittpunktes der Geraden werden die Geradengleichungen gleichgesetzt. Durch Auflösen nach x bzw. y ergibt sich

$$x_s = \frac{r_1 n_{y2} - r_2 n_{y1}}{n_{x1} n_{y2} - n_{y1} n_{x2}} \qquad y_s = \frac{r_2 n_{x1} - r_1 n_{x2}}{n_{x1} n_{y2} - n_{y1} n_{x2}}$$

Für den Fall, dass der Term $n_{x1} n_{y2} - n_{y1} n_{x2}$ Null wird, sind beide Gleichungen nicht definiert. Da es sich bei dem Normalenvektor nie um den Nullvektor handelt, kann dieser Fall nur eintreten, wenn n_1 und n_2 linear abhängig und somit die Strecken parallel sind. Parallele Kanten können von der Berechnung der Schnittpunkte aber ausgeschlossen werden, da sie entweder keinen Schnittpunkt aufweisen oder sich überlappen. Wie in Bild 2 zu sehen ist, bestehen immer auch Schnittpunkte zwischen jeweils einer Nachfolge- und einer Vorgängerkante mit einer der beiden Parallelen. In diesem Beispiel ist v_{g1} der Schnittpunkt von e_{g4} und e_{b1} und v_{b2} ein Schnittpunkt von e_{g1} und e_{b2}. Dies liegt daran, dass ein Stützpunkt immer zu zwei Kanten gehört. Aus diesem Grund werden auch Stützpunkte, die auf einer Kante liegen, die keine Überlappung aufweist, doppelt als Schnittpunkt erfasst.

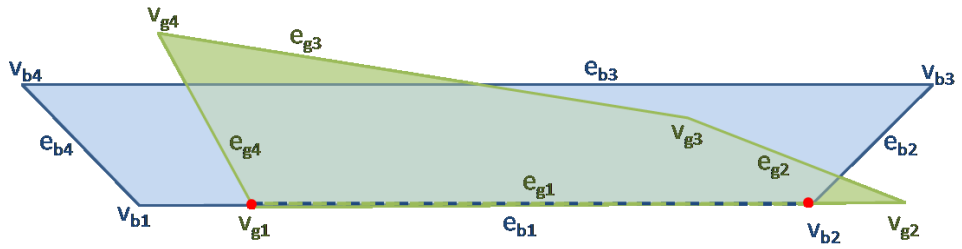

Bild 2 - Berechnung der Schnittpunkte bei überlappenden Kanten.

Um die Reihenfolge der Schnittpunkte im resultierenden Polygon bestimmen zu können, ist es notwendig, die Position des Schnittpunktes bzgl. beider Polygone zu ermitteln. Hierzu bietet es sich an, folgende Formel zu verwenden:

$$pos_i = kantenNr_i + \frac{|\vec{x} - \vec{a_i}|}{|\vec{e_i} - \vec{a_i}|}$$

Hierbei bezieht sich der Index i auf das Polygon, $\vec{x}$ bezeichnet den Schnittpunkt, $\vec{a}$ den Startpunkt und $\vec{e}$ den Endpunkt der Kante. Auf diese Weise ist auch eine eindeutige Zuordnung möglich, wenn auf einer Kante mehrere Schnittpunkte liegen. In Tabelle 2 sind die Werte für die Positionen der Schnittpunkte aus Bild 3 (links) dargestellt. Die Reihenfolge der Schnittpunkte kann sich in Bezug auf die beiden Polygone auch unterscheiden, wie in Bild 3 (rechts) zu sehen ist. Die Indizes der Schnittpunkte sind nach der Umlaufrichtung des grünen Rechtecks vergeben. In Bezug auf das blaue Polygon lautet die Reihenfolge jedoch: x_5, x_4, x_3, x_6, x_7, x_2, x_1, x_8

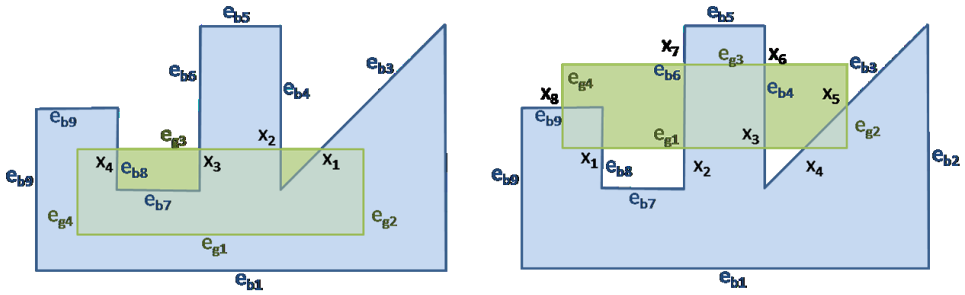

Bild 3- Bestimmung der Position von Schnittpunkten.

Schnittpunkt	Position bzgl. grünem Polygon	Position bzgl. blauem Polygon
X1	3,25	3,75
X2	3,375	4,25
X3	3,625	6,75
X4	3,875	8,5

Tabelle 2 – Position von Schnittpunkten.

Somit lautet die komplette Anfrage zur Schnittpunktberechnung, die als View gespeichert wird:

```
01 CREATE VIEW Schnitt AS
02 SELECT p1.id, p2.id, pos1, pos2, xs, ys
03 FROM Polygon p1, Polygon p2
04 WHERE(xs BETWEEN p1.startX AND p1.endX
05          OR xs BETWEEN p1.endX AND p1.startX)
06 AND(xs BETWEEN p2.startX AND p2.END
07          OR xs BETWEEN p2.endX AND p2.startX)
08 AND(ys BETWEEN p1.startY AND p1.endY
09          OR ys BETWEEN p1.endY AND p2.startY)
10 AND(ys BETWEEN p2.startY AND p2.endY
11          OR ys BETWEEN p2.endY AND p2.startY)
12 AND p1.id < p2.id  AND p1.m <> p2.m
```

Da es sich bei den berechneten Schnittpunkten um die Schnittpunkte der Geraden handelt, muss noch überprüft werden ob sie tatsächlich auf den Polygonkanten liegen. Weil man keine direkte Information über die Lage der Start- und Endpunkte zueinander hat, müssen hierzu alle vier möglichen Intervalle überprüft werden (Zeilen 04 bis 11). Hierbei sind pos1, pos2, xs und ys jeweils durch die in den oben angegebenen Gleichungen definierten Terme zu ersetzen, was aber aus Gründen der Übersichtlichkeit unterlassen wurde.

4 Das Halbstrahlverfahren

Die zweite grundlegende Operation, die benötigt wird, ist die Bestimmung der Punktlage, um die Stützpunkte eines Polygons zu bestimmen, die innerhalb des anderen Polygons liegen. Hierfür bietet sich das Halbstrahlverfahren an, da es nicht nur für beliebige Polygone anwendbar ist, sondern auch recht direkt in SQL ausgedrückt werden kann.

Der prinzipielle Ablauf ist dabei wie folgt: Von einem Punkt aus wird ein waagerechter Strahl nach rechts gezeichnet und die Anzahl der Schnittpunkte dieses Strahls mit den Polygonkanten wird gezählt. Ist die Anzahl der Schnittpunkte ungerade, so liegt der Punkt innerhalb des Polygons, andernfalls liegt er außerhalb.

Die folgende Formel gibt an, ob der Punkt mit den Koordinaten qx und qy rechts von der Kante liegt, die durch startx, starty, endx und endy gegeben ist. Bei xs handelt es sich um die X-Koordinate des Schnittpunktes zwischen dem Strahl und der Kante.

$$\Big(\big((starty > qy) \wedge (endy < qy)\big) \vee \big((starty < qy) \wedge (endy > qy)\big) \wedge (xs > qx)\Big)$$
$$\vee \big(((starty = qy) \wedge (startx > qy) \wedge (endy < qy\,)\,)$$
$$\vee ((endy = qy) \wedge (endx > qx) \wedge (starty < qy)))$$

Die Begründung, dass dieses Verfahren in allen Fällen das gewünschte Ergebnis liefert, ist in *Schmitt/Deussen/Kreeb* 1996 zu finden. Hier wird lediglich durch Bild 4 exemplarisch gezeigt, dass die ermittelte Anzahl der Schnittpunkte auch in den Sonderfällen, in denen der Strahl durch Stützpunkte oder auf Kanten verläuft, zum korrekten Ergebnis führt. So verläuft der Strahl von p_1 durch den gemeinsamen Stützpunkt der Kanten e_2 und e_3 und da die anderen beiden Stützpunkte unterhalb des Strahles liegen, werden beide Kanten als geschnitten

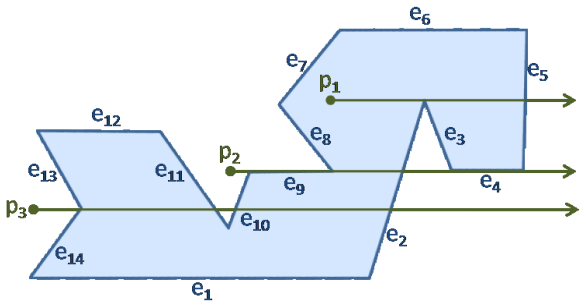

Bild 4 - Beispiel zum Halbstrahlverfahren.

gewertet und nehmen somit keinen Einfluss auf das Ergebnis. Dahingegen wird der gemeinsame Stützpunkt der Kanten e_{13} und e_{14}, durch den der von p_3 ausgehende Strahl verläuft, nur einmal gezählt, da die anderen Stützpunkte dieser Kanten auf unterschiedlichen Seiten des Strahles liegen. Folglich wird der Übergang vom Äußeren zum Inneren des Polygons, der an dieser Stelle stattfindet, registriert. Für Strahlen, die mit waagerechten Kanten überlappen, wie z.B. der Strahl von p_2 mit den Kanten e_4 und e_9, werden die Kanten nicht berücksichtigt. Dies ist in denselben Beobachtungen begründet, auf Grund derer Schnittpunkte zwischen Parallelen nicht betrachtet werden müssen und die in Abschnitt 2 erläutert sind. Punkte, die auf den Kanten des Polygons liegen, werden bei diesem Verfahren als außenliegende Punkte angesehen.

Somit ergibt sich folgende SQL-Anfrage zur Ermittlung aller Stützpunkte, die innerhalb eines anderen Polygons liegen. In den Zeilen 05 bis 11 ist die obige Formel kodiert und xs bezeichnet hier die X-Koordinate des Schnittpunktes zwischen der Polygonkante, die durch (out.startX, out.startY) sowie (out.endX, out.endY) gegeben ist und dem Strahl von (in.startX, in.startY).

```
01 SELECT in.startX, in.startY
02 FROM   Polygon in
03 WHERE (SELECT COUNT(*)
04       FROM Polygon out
05       WHERE (xs > in.startX
06       AND ((out.startY > in.startX AND out.endy < in.startY)
07        OR(out.endY > in.startY AND out.startY > in.startY)))
08        OR (((out.startY = in.startY AND out.startX > in.startY
09        AND out.endY < in.startY))
10        OR ((out.endY = in.startY AND out.endX > in.startY
11        AND out.startY < in.startY)))) % 2 = 1
```

5 Polygonverschneidung per SQL-Select

Typischerweise werden die Schnittpolygone bestimmt, indem ausgehend von einem Schnittpunkt solange jeweils die Kante des Polygons verfolgt wird, die innerhalb des anderen Polygons liegt, bis der Anfangspunkt erneut erreicht wird. Welche Kante zu verfolgen ist, lässt sich z.B. durch Betrachtung der Winkel zwischen den Kanten feststellen (vgl. *Xie/Ye/Wu* 2008). Eine Umsetzung dieser sequentiellen Strategie in SQL ist nicht möglich, da in SQL keine Schleifenkonstrukte zur Verfügung stehen. Dennoch sind die Schnittpolygone berechenbar, da es im Grunde ausreichend ist, nicht die Folge sondern die Menge der Kanten zu bestimmen. Hierfür werden in der Select-Liste folgende Werte benötigt: die Id des Schnittpolygons, X- und Y-Koordinate des Startpunktes, X- und Y-Koordinate des Endpunktes sowie die Kantennummer.

Um eine neue eindeutige Id zu bestimmen, existieren mehrere Möglichkeiten, die abhängig von dem gewünschten Format sowie dem Format der Ids der Input-Polygone sind. Ist z.B die Länge aller Ids gleich, so wäre die einfachste Möglichkeit, die Konkatenation der beiden Ids zu verwenden, sofern die Id der Schnittpolygone die doppelte Länge aufweisen darf. Wenn

aus dem Schnitt zweier Polygone jedoch mehrere Polygone resultieren, ist es nicht ohne Weiteres möglich, diesen Polygonen unterschiedliche Ids zuzuweisen. Im Folgenden wird vereinfachend die Id der Schnittpolygone als „neueId“ bezeichnet, ohne ihre genaue Berechnung festzulegen.

Als Anfangs- und Endpunkte der gesuchten Kanten kommen jeweils die Stützpunkte der beiden Input-Polygone sowie die Schnittpunkte zwischen diesen in Frage, wobei es die folgenden vier sinnvollen Kombinationen gibt:

a) Kante zwischen zwei Stützpunkten
b) Kante zwischen zwei Schnittpunkten
c) Kante zwischen Schnitt- und Stützpunkt
d) Kante zwischen Stütz- und Schnittpunkt

Des Weiteren ist zu berücksichtigen, zu welchem der beiden Polygone ein Stützpunkt ursprünglich bzw. bei d) zu welchem Input-Polygon die Kante, auf der die Schnittpunkte liegen, gehört. Da es sich somit bei den Start- und Endpunkten um Attribute aus unterschiedlichen Relationen bzw. um unterschiedliche Attribute einer Relation handeln kann, ist es notwendig, die Anfrage in acht Teilanfragen zu zerlegen und diese zu vereinigen.

Im Folgenden werden zunächst einige verkürzende Schreibweisen und Erläuterungen, die für alle Teilanfragen relevant sind, beschrieben. Anschließend werden die vier Typen (a) - d)) erläutert und abschließend kurz auf die Berechnung der Kantennummern für die Schnittpolygone eingegangen.

Von zwei Polygonen, die sich schneiden, wird das Polygon mit der kleineren Id im Folgenden als P1 und das andere als P2 bezeichnet, auch bezeichnet pos1 immer die Position bzgl. P1 und pos2 die Position bzgl. P2. Um die anderen vier Teilanfragen zu erstellen, die hier nicht explizit erwähnt werden, ist es lediglich notwendig, die Rolle der beiden Polygone innerhalb der Anfragen zu vertauschen. Die Abfrage zur Überprüfung der Punktlage, die in Abschnitt 4 erläutert wurde, wird durch die Angabe von welchem Punkt die Lage bzgl. welchen Polygons überprüft werden soll ersetzt, also z.B. „[Startpunkt der Kante von P1 in P2]“. Um später die Reihenfolge der Kanten der Schnittpolygone bestimmen zu können, ist es zunächst notwendig, den Kanten relativ zu den beiden Input-Polygonen eine Position zuzuordnen (pos1, pos2). Für Schnittpunkte sind diese schon im View „Schnitte“ berechnet worden. Für die Stützpunkte wird als Position die Kantennummer der Kante, dessen Startpunkt sie sind, verwendet und bzgl. des anderen Polygons wird die Position des Vorgänger-Schnittpunktes verwendet.

a) Kante zwischen zwei Stützpunkten
Eine Kante eines Input-Polygons ist auch eine Kante des Schnittpolygons, wenn sie komplett innerhalb des anderen Polygons liegt, wie z.B. in Bild 5 die Kanten e_{g5} und e_{g1}.

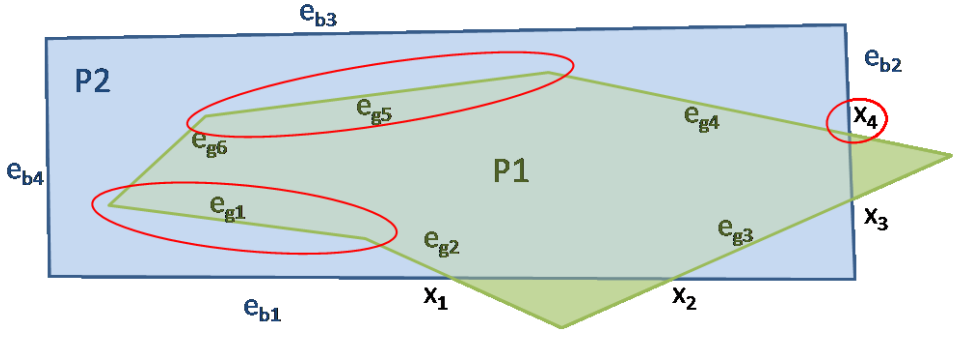

Bild 5 - Kanten zwischen zwei Stützpunkten.

Hierzu genügt es zu überprüfen, ob ein Punkt der Kante innerhalb des anderen Polygons liegt und ob sie geschnitten wird. Somit ergibt sich folgende SQL-Anfrage, um alle Kanten des Polygons P1 zu bestimmen, die innerhalb von P2 liegen:

```
01 SELECT neueId, p.edgeId AS pos1, s1.pos2,
02       p.startX, p.startY, p.endX, p.endY
03 FROM Schnitt s1, Polygon p
04 WHERE s1.id1 = p.id
05 AND NOT EXISTS (SELECT * FROM Schnitte s2
06                   WHERE s2.pos1 BETWEEN p.edgeId AND p.edgeId+1
07                   AND s2.id1 = s1.id1 AND s2.id2 = s1.id2)
08 AND s1.pos1 = (SELECT min(pos)
09             FROM ((SELECT max(pos1) AS pos FROM Schnitt s4
10                   WHERE s4.id1 = s1.id1 AND s4.id2 = s1.id2
11                   AND s4.pos1 < edgeId) UNION
12             SELECT max(pos1) AS pos FROM Schnitt s4
13                   WHERE s4.id1 = s1.id1 AND s4.id2 = s1.id2))
14 AND[Startpunkt der Kante von P1 in P2]
```

In den Zeilen 03 bis 05 wird kontrolliert, dass die Kante nicht geschnitten wird. Hierzu muss lediglich die Position der Schnittpunkte betrachtet werden, da deren Vorkommastelle immer die Kantennummer der Kante angibt, auf der der Punkt liegt.

Die Zeilen 08 bis 13 dienen ausschließlich der Bestimmung der Position bzgl. des zweiten Polygons. Das Union innerhalb dieser Unterabfrage ist notwendig, da der größte kleinere Schnittpunkt nicht unbedingt eine kleinere Position als die Kantennummer aufweist. So ist sowohl für die Kanten e_{g5} als auch für e_{g1} der Vorgänger-Schnittpunkt x_4, dessen Position bzgl. P1 ca. 4,25 beträgt. Allgemein gilt, dass der zweite Teil der Unterabfrage immer dann relevant ist, wenn zwischen diesem Schnittpunkt und dem Endpunkt der betrachteten Kante die Kante liegt, deren Kantennummer 1 ist.

b) Kante zwischen zwei Schnittpunkten

Da es sich bei den Kanten des Schnittpolygons entweder um Kanten der Input-Polygone oder um Abschnitte auf diesen handelt, können Kanten nur durch zwei Schnittpunkte definiert sein, sofern diese auf der gleichen Kante eines Input-Polygons liegen, wie beispielsweise in Bild 6 die beiden Schnittpunkte x_2 und x_3, die beide auf der Kante e_{g3} liegen.

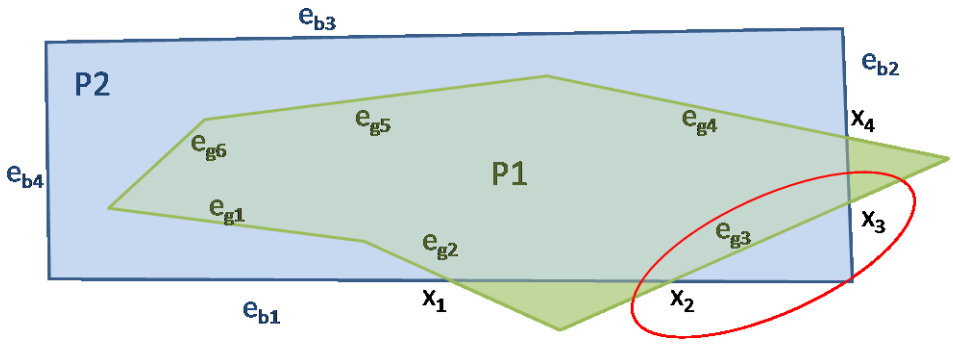

Bild 6 - Kante zwischen zwei Schnittpunkten.

Des Weiteren muss dieser durch die Schnittpunkte definierte Abschnitt innerhalb des anderen Polygons liegen und die beiden Schnittpunkte müssen direkt aufeinander folgen. Diese Kanten werden durch folgende SQL-Anfrage bestimmt:

```
01 SELECT neueId, s1.pos1, s1.pos2, s1.x AS startX,
02              s1.y AS startY, s2.x AS endX, s2.y AS endY
03 FROM Schnitt s1, Schnitt s2
04 WHERE s1.pos1 < s2.pos2
05 AND s1.id1 = s2.id2 AND s1.id2 = s2.id2
06 AND NOT EXISTS (SELECT * FROM Schnitt s3
07                   WHERE s3.pos1 > s1.pos1 AND s3.pos1 < s2.pos1
08                   AND s1.id1 = s3.id1 AND s1.id2 = s3.id2)
09 [Mittelpunkt der Strecke zwischen den Schnittpunkten in P2]
10 AND floor(s1.pos1) = floor(s2.pos1)
```

In den Zeilen 06 bis 08 erfolgt die Überprüfung, dass kein weiterer Schnittpunkt zwischen den Schnittpunkten liegt. Um festzustellen, ob zwei Schnittpunkte auf einer Kante liegen, genügt es wieder, die Position der Schnittpunkte zu betrachten: Ist die Vorkommastelle identisch, liegen sie auf derselben Kante (Zeile 10)

c) Kante zwischen Schnitt- und Stützpunkt

In diesem Fall müssen folgende Bedingungen erfüllt sein: Der Stützpunkt liegt innerhalb des anderen Polygons, der Schnittpunkt liegt auf der Kante, deren Endpunkt der Stützpunkt ist, und es liegen keine weiteren Schnittpunkte zwischen den beiden Punkten. Ein Beispiel für diesen Fall ist in Bild 7 markiert. Der Endpunkt der Kante e_{g4} liegt innerhalb von P2 und der Schnittpunkt x_4 liegt als einziger auf dieser Kante.

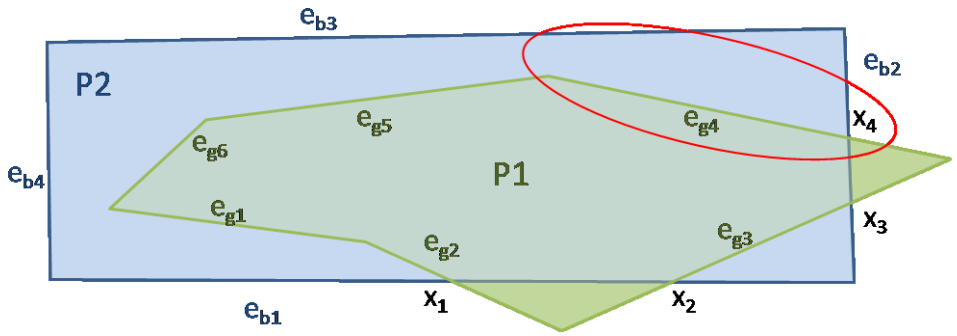

Bild 7 - Kante zwischen Schnitt- und Stützpunkt.

Dies wird durch folgende SQL-Anfrage sichergestellt:

```
01 SELECT neueId, s1.pos1, s1.pos2, s1.x AS startX,
02       s1.y AS startY, p.startX AS endX, P.startY AS endY
03 FROM Schnitt s1, Polygon P
04 WHERE s1.id1 = p.id
05 AND p.edgeId< (SELECT min(pos1 FROM Schnitt s2
```

```
06                  WHERE s2.pos1 > s1.pos1
07                  AND s1.id1 = s2.id1 AND s1.id2 = s2.id2)
08 AND (p.edgeId = ceil(s1.pos1)
08 OR(p.edgeId = 1
09 AND s1.pos1 = (SELECT max(pos1) FROM Schnitt s2
10                  WHERE s1.id1 = s2.id2 AND s2.id2 = s1.id2)
11 AND s1.pos1 > (SELECT max(edgeId) FROM Polygon p2
12                  WHERE p2.id = p.id)))
13 [Startpunkt der Kante von P1 liegt in P2]
```

In der Unterabfrage in den Zeilen 05 bis 07 wird der nächstgrößere Schnittpunkt berechnet, denn falls dieser größer als die Kantennummer des Stützpunktes ist, gibt es keinen Schnittpunkt, der zwischen den beiden Punkten liegt. Die Überprüfung, dass die beiden Punkte auf einer Kante liegen, erfolgt in den Zeilen 08 bis 13. Hierzu ist eine Fallunterscheidung nötig, da die einfache Überprüfung anhand der Position des Schnittpunktes nur gilt, wenn der Stützpunkt nicht die Position 1 hat. Falls doch, muss der Schnittpunkt auf der Kante mit der größten Id liegen.

d) Kante zwischen Stütz- und Schnittpunkt
Diese Teilanfrage ist analog zum Fall c), nur dass die Reihenfolge von Schnitt- und Stützpunkt vertauscht ist, so dass in Zeile 08 die Position des Schnittpunktes abgerundet wird. Aus diesem Grund ist auch die folgende Sonderbehandlung nicht notwendig, da die Vorkommastelle immer die Kante angibt, auf der der Schnittpunkt liegt.

```
01 SELECT neueId, p.edgeId AS pos1, s1.pos2,
02       p.startX, p.startY, s1.x AS endX, s1.y AS endY
03 FROM Schnitt s1, Polygon p
04 WHERE s1.id1 = p.id
05 AND NOT EXISTS (SELECT * FROM Schnitt s2
06                   WHERE s2.pos1 < s1.pos1 AND p.edgeId < s2.pos1
07                   AND s1.id1 = s2.id1 AND s1.id2 = s2.id2)
08 AND p.edgeId = floor(s1.pos1)
09 [Startpunkt der Kante von P1 liegt in P2]
```

Setzt man nun diese acht Teilanfragen zu einer Anfrage zusammen ergibt sich folgende Struktur:

```
01 [Kante von P1 in P2]
15 UNION
16 [Kante von P2 in P1]
30 UNION
31 [Kante zwischen zwei Schnittpunkten auf einer Kante von P1]
41 UNION
42 [Kante zwischen zwei Schnittpunkten auf einer Kante von P2]
52 UNION
53 [Kante zwischen einem Schnittpunkt und einem Stützpunkt von
   P1]
```

```
66 UNION
67 [Kante zwischen einem Schnittpunkt und einem Stützpunkt von
   P2]
80 UNION
81 [Kante zwischen einem Stützpunkt von P1 und einem Schnitt-
   punkt]
91 UNION
92 [Kante zwischen einem Stützpunkt von P2 und einem Schnitt-
   punkt]
```

Wird nun jeweils noch die verkürzte Schreibweise für die Überprüfung der Punktlage durch die Anfrage aus Abschnitt 4 ersetzt und die Sicht „Schnitte“ durch ihre Definition, besteht die komplette Anfrage aus ca. 450 Zeilen.

Bislang wurden zu den Kanten der Schnittpolygone nur die Positionen bzgl. der beiden Input-Polygone ermittelt. Da hierdurch die Reihenfolge der Kante durch Sortierung zunächst nach pos1 und bei gleicher pos1 nach pos2 bestimmt werden kann, ist es auch möglich, ganzzahlige Postionen zu erhalten, die dicht liegen. Allerdings ist es hierzu notwendig, zu jeder Kante die Anzahl der Kanten desselben Schnittpolygons zu bestimmen, die eine kleinere Position haben. Dies kann durch folgende SQL-Anfrage geschehen, wobei „Result“ jeweils durch die komplette Anfrage zur Bestimmung der Schnittpolygone zu ersetzen ist, wodurch sich eine Gesamtanfrage mit ca. 900 Zeilen ergibt.

```
01 SELECT r1.id, count(*) AS edgeId,
02       r1.startX, r1.startY, r1.endX, r1.endY
03 FROM   Result r1, Result r2
04 WHERE r1.id = r2.id
05 AND  (r1.pos1 > r2.pos1
06       OR (r1.pos1 = r2.pos1 AND r1.pos2 >= r2.pos2))
07 GROUP BY    r1.id, r1.pos1, r1.pos2,
08             r1.startX, r1,startY, r1.endX, r1.endY
```

6 Zusammenfassung und Ausblick

Im vorliegenden Text haben wir skizziert, wie die anspruchsvolle GIS-Basisoperation der Polygonverschneidung als SQL-Anfrage rein relational implementiert werden kann. Dazu haben wir zunächst das zur Speicherung der Polygone verwendete Relationenschema hergeleitet und sind dann auf die SQL-Anfrage eingegangen, die zur Schnittpunktberechnung der Polygonsegmente benötigt wird. Eine weitere SQL-Anfrage realisiert das Halbstrahlverfahren zur Bestimmung der Lage von Stütz- und Schnittpunkten relativ zu betrachteten Polygonen. Die eigentliche (umfangreiche) Anfrage zur Polygonverschneidung wurde schließlich aus diesen Bausteinen und aus acht Fallunterscheidungen systematisch zusammengesetzt. Die Fallunterscheidungen behandeln dabei die verschiedenen Lagen der Stützpunkte der Ergebnispolygone in ihren Rollen als Stütz- und Schnittpunkten der Ausgangspolygone.

Die resultierende Gesamtanfrage umfasst ca. 900 Zeilen SQL-Code und wurde inzwischen an einigen kleineren Beispiel-Polygonen getestet. Ein umfassender Laufzeittest mit Polygonen realistischen Umfangs steht noch aus. Dazu planen wir Testdaten vorzugsweise aus dem Bereich Bodennutzung/Bodenarten zum Einem in dem hier vorgestellten Relationenschema abzuspeichern und zum Anderen mit Hilfe einer räumlichen Erweiterung eines relationalen Datenbanksystems. Bei den Zeitmessungen erwarten wir zwar eine deutlich schnellere Antwortzeit bei der Verwendung der räumlichen Datentypen, nehmen aber an, dass auch bei der Verwendung der rein relationalen Lösung noch akzeptable Antwortzeiten beobachtet werden können. Dazu müssen allerdings alle in Betracht kommenden Attribute und Attributkombinationen unserer vorgeschlagenen Polygonrelation mit passenden Indexen, also zusätzlichen Zugriffsstrukturen, versehen werden. Auch der Optimierer des zu verwendenden relationalen Datenbanksystems ist für die Laufzeitexperimente von zentraler Bedeutung, da in unserer Anfrage zur Polygonverschneidung z.B. sehr viele identische relationale Teilausdrücke auftreten, die natürlich vom Datenbanksystem nur einmal ausgewertet werden sollten.

Literatur

Andrews, D. S.; Snoeyink, J.;Boritz J.; Chan, T.; Denham, G.; Harrison, J.; Zhu, C.: Further Comparison of Algorithms for Geometric Intersection Problems, In Proc. 6th International Symposium on Spatial Data Handling, Volume 2, 1994, pp. 709 - 724.

Egenhofer, M. J.: Why not SQL!, International Journal of Geographical Information Science, Volume 6, Issue 2, 1992, pp. 71 – 85.

Hake, G.; Grünreich, D.; Meng, L.: Kartographie, 8. Aufl., de Gruyter, 2002.

Rigaux, P.; Scholl, M.; Voisard, A.: Spatial Databases with Application to GIS, Morgan Kaufman, 2002.

Schek, H.; Scholl, M. H.: The relational model with relation-valued attributes, Information Systems, Volume 11, Issue 2, 1986, pp. 137-147.

Schmitt, A.; Deussen, O.; Kreeb, M.: Einführung in graphisch-geometrische Algorithmen, B.G. Teubner, Stuttgart, 1996.

Xie, Z.; Ye, Z.; Wu, L.: A Design for Polygon Overlay Algorithm in the Simple Feature Model, Seventh International Conference on Grid and Cooperative Computing, 2008, pp.680-685.