Implementatie van een Tin-based 2D pan-functionaliteit op een geodatabase

Verslag Geo DBMS casestudy

Friso Penninga

Sectie GIS-Technologie
Technische Universiteit Delft
2002

Implementatie van een Tin-based 2D pan-functionaliteit op een geodatabase

Verslag Geo DBMS casestudy

Friso Penninga

Sectie GIS-Technologie
Technische Universiteit Delft
2002

Begeleiders:
drs. Wilko Quak
ir. Edward Verbee

Voorwoord

Dit verslag is geschreven in het kader van het vak geoDBMScasestudy, een keuzevak in de eindstudie van de opleiding Geodesie. Bij dit vak wordt een onderzoek met een duur van vijf weken uitgevoerd naar een onderwerp gerelateerd aan geodatabases, uitgevoerd.

Lezers die geïnteresseerd zijn in de opdracht en de daadwerkelijke implementatie in Java verwijs ik naar paragraaf 2.2 voor de opdracht en Bijlage I waarin alle scripts integraal zijn opgenomen. Hoofdstuk 4 is interessant voor lezers die juist geïnteresseerd zijn in de conceptuele vertaling van het probleem in een aantal algoritmen.

Zelf ben ik dank verschuldigd aan Edward Verbree en Wilko Quak die ondanks een zeer moeizame start mij uiteindelijk toch aan het studeren hebben gekregen.

Delft, juli 2002

Friso Penninga

Inhoudsopgave

Samenvatting

Het doel van deze casestudy was om een Tin-based 2D pan-functionaliteit te implementeren. Deze pan-functie probeert een snelle grafische interface mogelijk te maken door een deel van een tin reeds in te lezen in het geheugen, om zo snel te kunnen reageren op een pan-actie. Het voortdurend bijhouden welk deel van het tin in het geheugen ingelezen moet worden, is onderwerp van deze casestudy. Gekozen is om de data op te slaan in een zelf gedefinieerde datarepresentatie, de double directed winged edge representatie. Deze representatie is een variant op de bekende winged edge representatie en biedt de mogelijkheid edges double directed op te slaan. Met de in dit datamodel beschikbare kennis van topologische relaties is het mogelijk om snel te bepalen welk deel van het tin moet worden ingelezen in het geheugen.

Het zoeken van edges die in het uitbreidingsgebied van het geheugen liggen, gebeurt in drie fasen. Allereerst worden de edges gezocht die snijden met de grenzen van het uitbreidingsgebied. Als tweede fase wordt binnen de hierdoor ontstane ring van snijdende edge een binnenring van edges geconstrueerd. Uiteindelijk worden ook de overige edges in het uitbreidingsgebied gevonden, waarbij de twee in de eerdere fasen geconstrueerde ringen als buitengrenzen dienen.

Deze ideeën zijn uiteindelijk geïmplementeerd in een java-applicatie. Deze applicatie blijkt het mogelijk te maken om snel de edges te vinden die in het uitbreidingsgebied liggen. Binnen de beperkte tijdsduur van dit onderzoek was het helaas niet mogelijk om de performance van deze applicatie te vergelijken met andere technieken, waardoor met de conclusie volstaan wordt dat de applicatie een bevredigende performance heeft.

Naast het implementeren van de beschreven pan-functionaliteit heeft deze casestudy een zeer praktische inslag. Een tweede doel was het leren van Java en het opdoen van ervaring met Oracle (Spatial), JDBC en Java. Met name op programmeergebied is er vooruitgang geboekt, zeker ook doordat alle geschreven code kritisch is doorlopen en besproken met de begeleiding.

1 Inleiding

Het gebruik van viewers op ruimtelijke data neemt tegelijk met de toename van het gebruik van ruimtelijke data zelf, snel toe. Aangezien ook de bestandsomvang van deze ruimtelijke gegevens steeds verder toeneemt, is er behoefte aan methoden om de viewers toch snel te laten werken. Eén van de mogelijkheden die een viewer de gebruiker biedt, is het pannen door data. Hierbij sleept de gebruiker als het ware een venster over de data om steeds andere data te zien. Het in het geheugen ingelezen gebied zal na elke panning-stap aangepast moeten worden. Snelle aanvulling van dit gebied zal het mogelijk maken om een goed presterende viewer te bouwen.

De doelstelling van dit onderzoek is het implementeren van de panning-functie, waarbij een methode ontworpen moet worden om de edges in het uitbreidingsgebied te vinden, gebruik makend van de bekende topologische relaties. Daarnaast is het de bedoeling dat met dit onderzoek meer ervaring wordt opgedaan met het programmeren in Java en het omgaan met Oracle, SQL en JDBC.

De opbouw van dit rapport is als volgt. In hoofdstuk 2 wordt de beoogde panning-functie beschreven en wordt ingegaan op de leerdoelen van de casestudy. Hoofdstuk 3 gaat dieper in op de verschillende methoden voor datarepresentatie, waarbij aan het eind van het hoofdstuk de tijdens dit onderzoek ontworpen datarepresentatie wordt toegelicht. De algoritmen die het samen mogelijk maken om de edges in het uitbreidingsgebied te vinden, zijn onderwerp van discussie in hoofdstuk 4. Hoofdstuk 5 gaat in op de praktische uitvoering en implementatie van deze algoritmen. Het rapport sluit af met een aantal conclusies in hoofdstuk 6.

2 Opdracht: implementatie magic carpet panning

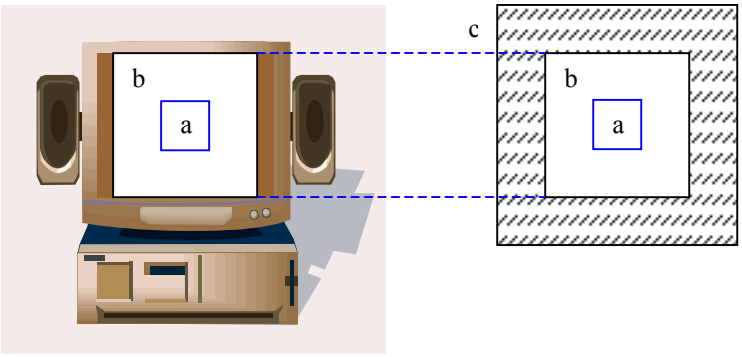
2.1 Inleiding

In dit hoofdstuk wordt de opdracht uitgewerkt. In paragraaf 2.2 wordt de werking van de beoogde pan-functie uitgelicht. De leerdoelen die nagestreefd worden bij deze geoDBMS casestudy zijn uiteengezet in paragraaf 2.3.

2.2 De panning-functie

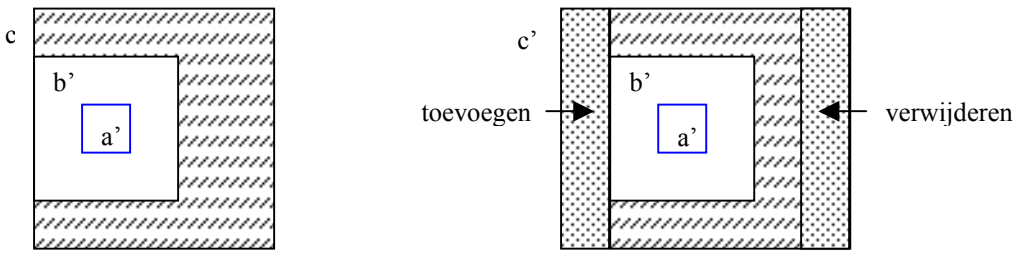
Het doel van deze geoDBMScasestudy is het implementeren van een pan-functie die direct werkt op in een database opgeslagen data, de magic carpet panning. Uitgangspunt van de pan-functionaliteit zijn de drie gebieden A, B en C:

- gebied A: centrumgebied van het display;
- gebied B: in display weergegeven gebied;
- gebied C: in geheugen ingelezen gebied.



Figuur 2.1 De voor de gebruiker zichtbare gebieden A en B en het niet zichtbare gebied C.

De gebruiker kan door de data pannen door de cursor (het centrumgebied A) te verplaatsen, bijvoorbeeld in westelijke richting. De display (gebied B) kan direct worden aangepast zonder eerst de database te benaderen, aangezien de extra data reeds in het geheugen (in gebied C) was ingelezen (zie ook figuur 2.2a). Om te zorgen dat ook de volgende pan-actie snel verwerkt kan worden, zonder eerst de database te benaderen, zal het gebied C aangepast moeten worden (zie figuur 2.2b). Hiervoor wordt een gebiedje westelijk van het oorspronkelijke gebied C opgevraagd uit de database en wordt een gedeelte aan de oostzijde uit het geheugen verwijderd.



Figuur 2.2a) Displaygebied B in westelijke richting verplaatst b)Aanpassing aan gebied C

De prestaties van deze pan-functie hangen af van de snelheid waarmee kan worden bepaald welk deel van de onderliggende data aan het geheugengebied kan worden toegevoegd of uit het gebied kan worden verwijderd. Deze casestudy concentreert zich op de vraag hoe kan

worden bepaald welke data uit de database moet worden opgevraagd om deze uitbreidingsgebieden snel te kunnen vullen. Hierbij zijn wel enkele randvoorwaarden gesteld, zo is de data beschikbaar in tin-structuur en willen we gebruik maken van topologische relaties.

2.3 Leerdoelen

Naast het realiseren van de in de vorige paragraaf beschreven pan-functie, zijn er meer doelstellingen gedefinieerd bij deze casestudy. Deze doelstellingen zijn mede gedefinieerd met het oog op het naderende afstudeeronderzoek. Zo wordt een uitbreiding van de praktische kennis van Oracle en SQL beoogd. Verder zal in het kader van dit onderzoek de objectgeoriënteerde programmeertaal Java worden geleerd. Op inhoudelijk gebied wordt er nader ingegaan op de diverse datastructuren waarin een tin kan worden opgeslagen.

Om meer ervaring met Oracle en SQL op te doen, zal het benodigde tin eerst worden geëxporteerd uit ArcView en vervolgens worden ingelezen in Oracle. Hiervoor zal eerst in Oracle een tabel aangemaakt worden, waar vervolgens met behulp van SQLLoader de data in weggeschreven zal worden. De implementatie van de pan-functie zelf zal gebeuren met behulp van in Java geschreven code, waarbij voor het benaderen van de database JDBC gebruikt zal worden. JDBC is de connectie tussen Java en de database.

3 Methoden voor data-representatie

3.1 Inleiding

Dit hoofdstuk gaat in op de verschillende methoden voor het representeren van data. Deze representatiemethoden maken het mogelijk om bewerkingen op data uit te voeren. De meest eenvoudige methoden ondersteunen weinig bewerkingen en definiëren geen topologische relaties, de meer complexe methoden maken bewerkingen eenvoudiger en sneller, doordat er meer ruimtelijke relaties bekend zijn.

De meest gebruikelijke indeling van de methoden voor data-representatie is de tweedeling tussen field-based en object-based methoden. Field-based methoden komen veel voor in de vorm van rasterrepresentaties, waarbij aan elk veld in het raster een waarde wordt toegekend. De object-based methoden zijn ook wel bekend als vector-methoden. Deze methode is geometrisch nauwkeuriger, aangezien in een raster data geïnterpoleerd wordt om voor elk veld tot een waarde te komen.

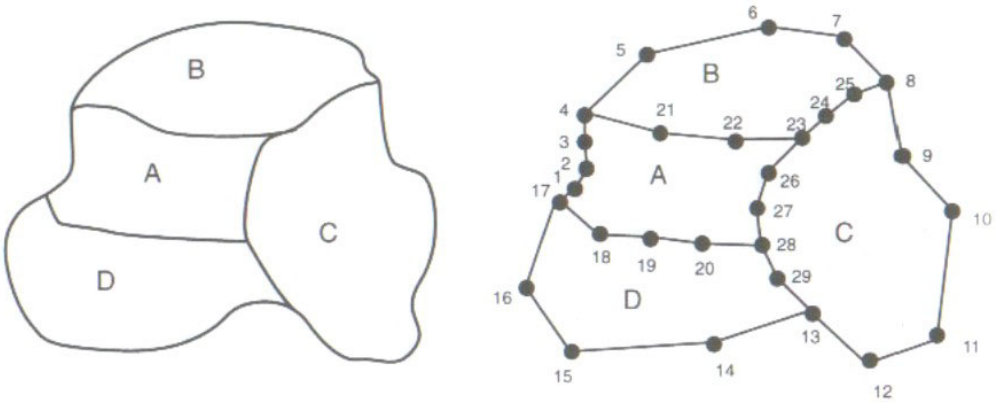
De object-based methoden kunnen weer verder onderverdeeld worden in methoden die betrekking hebben op het geometrische domein en die betrekking hebben op het ruimtelijke objectdomein. De methoden in het geometrisch domein gaan met name in op discretiseringsvraagstukken, bijvoorbeeld het vraagstuk met hoeveel punten een vloeiende lijn beschreven kan worden.

De methoden in het ruimtelijke objectdomein concentreren zich op ruimtelijke relaties tussen objecten, vaak ook aangeduid als topologische relaties. Paragraaf 3.2 gaat in op een aantal verschillende representaties in het ruimtelijke objectdomein. In paragraaf 3.3 wordt vervolgens de gekozen data-representatie toegelicht. Deze representatie wordt uitgebreid beschreven, aangezien het een soort tussenvorm betreft tussen een methode in het geometrische domein en een methode in het ruimtelijke domein.

3.2 Representaties in het ruimtelijke objectdomein

3.2.1 Spaghetti

De datarepresentatie die aangeduid wordt met de naam spaghetti is de minst gestructureerde representatie. In deze representatie is veel geometrie zichtbaar, zoals in figuur 3.2 te zien is.



Figuur 3.2 Spaghetti representatie (detail Worboys, p.192)

Deze figuur laat zien dat de grenzen van de vlakken beschreven worden met behulp van de punten die uit een discretisering van de continue grenzen voortkomen. Voor de vlakken A t/m D worden als volgt de puntnummers opgeslagen die de grenzen opspannen:

A: [1, 2, 3, 4, 21, 22, 23, 26, 27, 28, 20, 19, 18, 17]
B: [4, 5, 6, 7, 8, 25, 24, 23, 22, 21]
C: [8, 9, 10, 11, 12, 13, 29, 28, 27, 26, 23, 24, 25]
D: [17, 18, 19, 20, 28, 29, 13, 14, 15, 16]

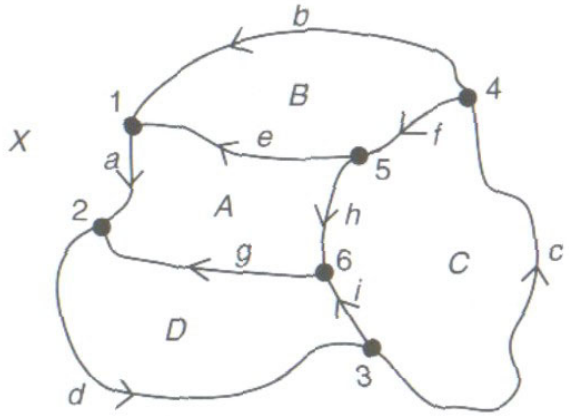
In grijs gemarkeerd is direct een belangrijk nadeel van deze methode weergegeven, namelijk het feit dat gezamenlijke grenzen dubbel worden opgeslagen, gebruik makend van alle punten die in beide grenzen zitten. Een tweede nadeel is dat er geen expliciete ruimtelijke relaties worden weergegeven, zo is bijvoorbeeld de buurrelatie tussen de vlakken A en C alleen af te leiden uit het in beide lijsten voorkomen van de punten [23, 26, 27, 28].

3.2.2 NAA (node-arc-area) en extended NAA

Zoals de naam aangeeft, is de representatie opgebouwd uit nodes, arcs en areas. Twee nodes spannen een arc op, waarbij de ene node als begin-node wordt gedefinieerd en de andere als eind-node, waardoor een gerichte arc ontstaat. Het voordeel hiervan is dat door een gerichte arc inhoud gegeven wordt aan de begrippen links en rechts. De arcs vormen namelijk de grenzen tussen de areas en voor elke arc wordt weergegeven welke area links en welke area rechts ligt. In figuur 3.3 is te zien dat naast de vlakken A t/m D nu een vijfde vlak is gedefinieerd, vlak X, een vlak dat buiten de eigenlijke data valt. Met dit vlak is aan arcs te zien als ze een buitengrens zijn. De node-arc-area representatie is erg geschikt voor opslag in een relationele database:

ARC (arc-id, begin-node, eind-node, area-links, area-rechts)

De NAA is alleen gericht op vlakken en de relaties tussen vlakken. Om alle ruimtelijke objecten te kunnen beschrijven, zijn in de extendend NAA de begrippen point, polyline en polygon toegevoegd. Fysieke objecten kunnen beschreven worden met deze drie begrippen. Tussen deze drie objecten bestaan vergelijkbare relaties als tussen de node, arc en area. Een polygon is opgebouwd uit meerdere polylines en een polyline bevat weer meerdere points. Points en polylines kunnen echter ook als zelfstandige objecten voorkomen, denk bijvoorbeeld aan een point als representatie voor een verkeerslicht en een polyline als representatie voor een sloot.

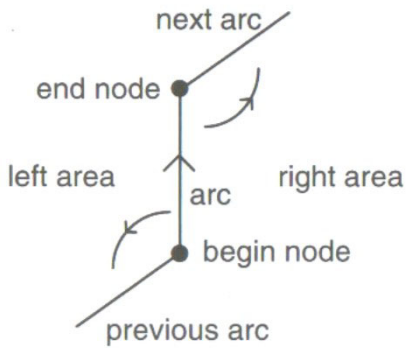


Figuur 3.3 NAA representatie

3.2.3 DCEL (doubly connected edge list)

De DCEL-representatie is volledig gericht op de topologische relaties tussen de entiteiten node, arc (ook wel edge) en area (ook wel face). Ten opzichte van de NAA is de DCEL uitgebreid met de begrippen previous arc en next arc. In figuur 3.4 zijn deze begrippen verduidelijkt: de previous arc is de arc die de begin-node van de huidige node als eind-node heeft. Aangezien dit meerdere arcs kunnen zijn, wordt als previous arc de arc aangeduid die als eerste wordt gevonden tegen de klok in. Voor de next arc is een vergelijkbare definitie: de

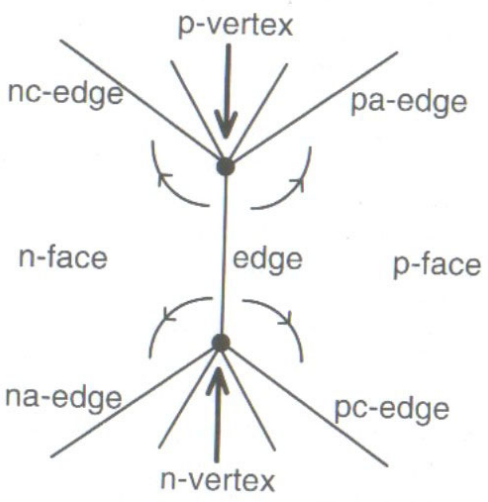
eerste arc die tegen de klok in wordt gevonden die als begin-node de eind-node van de huidige arc heeft. Dankzij deze twee toegevoegde begrippen bevat de DCEL-representatie voldoende gegevens om voor elke node en voor elke area de volgorde van arcs rond de node of area te kunnen bepalen. Het verschil met de NAA-representatie zit in het woord volgorde: met de gegevens in de NAA kunnen wel alle arcs rond een node worden bepaald, maar niet de volgorde waarin ze rond de node gelegen zijn.



Figuur 3.4 De begrippen previous arc en next arc (Worboys, p.197)

3.2.4 Winged edge representatie

De winged edge representatie is een variant op de DCEL-representatie. In figuur 3.5 is te zien welke edges zijn toegevoegd aan de DCEL-representatie om tot de winged edge representatie te komen. Zowel de next edge als de previous edge is nu zowel met de klok mee als tegen de klok in gedefinieerd, waarmee de benaming als volgt is: next clockwise edge (nc-edge), previous edge clockwise (pc-edge), next edge anti-clockwise (na-edge) en previous edge anti-clockwise (pa-edge). Een ander verschil is bovendien dat de edges in de winged edge representatie niet gericht zijn. Er zijn weliswaar een n- en een p-vertex gedefinieerd, maar de keuze welke vertex n en welke vertex p is, is arbitrair. Tegelijk met de keuze voor de n- end e p-vertex liggen ook de n- en de p-face vast.



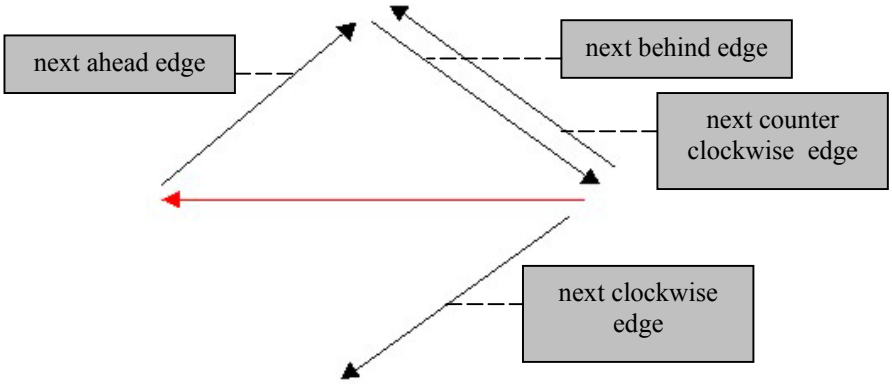
Figuur 3.5 Winged edge representatie (Worboys, p.202)

3.3 Toegepaste data-representatie

De data-representaties die tot nu toe besproken zijn (en dan met name de DCEL en de winged edge representatie) laten duidelijk zien dat het gebruik van topologie zich richt op de representatie in het ruimtelijke domein en niet op het geometrische domein. De in dit onderzoek gebruikte data-representatie is hier echter een variant op. Voor elke edge in een tin worden topologische relaties met de buur-edges en vlakken opgeslagen, maar dit tin wordt gebruikt als geomterische representatie voor een verzameling polygonen. Voor alle edges in het tin kunnen topologische relaties worden opgeslagen, om zo standaard gis-bewerkingen als buffer, intersect en (met name) overlay sneller uit te kunnen voeren. De edges die onderdeel uitmaken van de polygonen hebben een aantal extra attributen, die aangeven dat ze deel uitmaken van een polygoon en welke polygonen links en rechts liggen. Zo is met behulp van de topologische relaties tussen de edges indirect ook de topologie van de polygonen vastgelegd. Dankzij het attribuut dat aangeeft of een edge deel uitmaakt van een polygoon is immers steeds de volgende edge te vinden die ook deel uitmaakt van een polygoon om zo op basis van topologie een polygoon te reconstrueren uit één bekende edge (polygoonsegement).

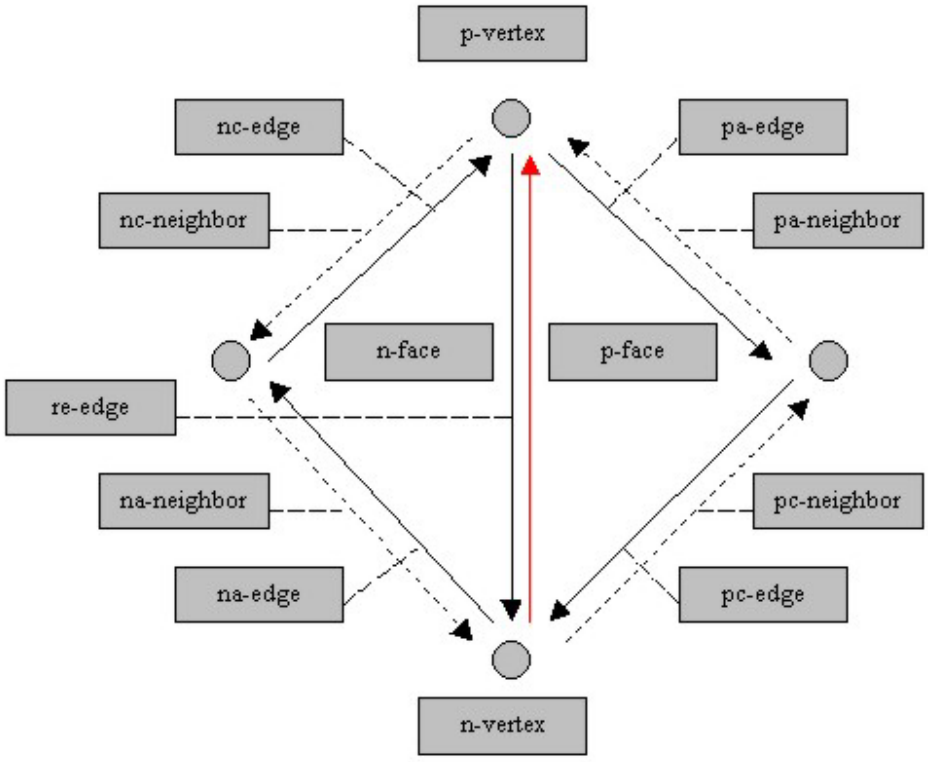
Het gebruik van topologie in een dergelijke data-representatie is dus niet alleen gericht op de relaties tussen de te beschrijven objecten, maar ook op de relaties tussen de edges die gebruikt worden om die objecten te representeren. Als deze representatie vergeleken wordt met de representaties die in het ruimtelijke domein vallen, zit het groter verschil in het feit dat nu ook het achterliggende geometrische model intelligenter wordt gemaakt, om zo bewerkingen te versnellen.

De keuze voor de voor het tin te gebruiken data-representatie is mede ingegeven door het feit dat het tin is gedefinieerd in ArcView. Binnen ArcView is gebruik gemaakt van de klasse TINEdge om het tin te bevragen. Op basis van de direct op te vragen eigenschappen is een data-representatie te construeren zoals afgebeeld in figuur 3.7.



Figuur 3.6 Data-representatie ArcView op basis van direct opvraagbare relaties

Deze data-representatie doet denken aan de DCEL en de winged edge representatie, maar is geen van beiden. Op basis van dit datamodel is een uitgebreider model gedefinieerd, dat de naam double directed winged edge representatie heeft gekregen. Zoals gezegd is het een winged edge representatie, maar dan double directed, oftewel elke edge bestaat zowel van n- naar p-vertex als andersom. Qua benaming resulteert dit voor de nc-, pc-, na- en nc-edge in een nc-, pc-, na- en nc-neighbor. Voor de huidige edge is ook een reverse (re-edge) gedefinieerd. Deze representatie is in figuur 3.7 weergegeven. Dit betekent dat elke edge dus twee keer een vertex, twee keer een face, een reverse edge en acht buur-edges als attributen heeft. Dit kan vervolgens nog uitgebreid worden om de polygonen in het tin te kunen weergeven met behulp van attributen als polygoon-ID en of het polygoon links of rechts van de edge ligt. Dit laatste is in het onderzoek niet geïmplementeerd.



Figuur 3.7 De zelf gedefinieerde double directed winged edge representatie

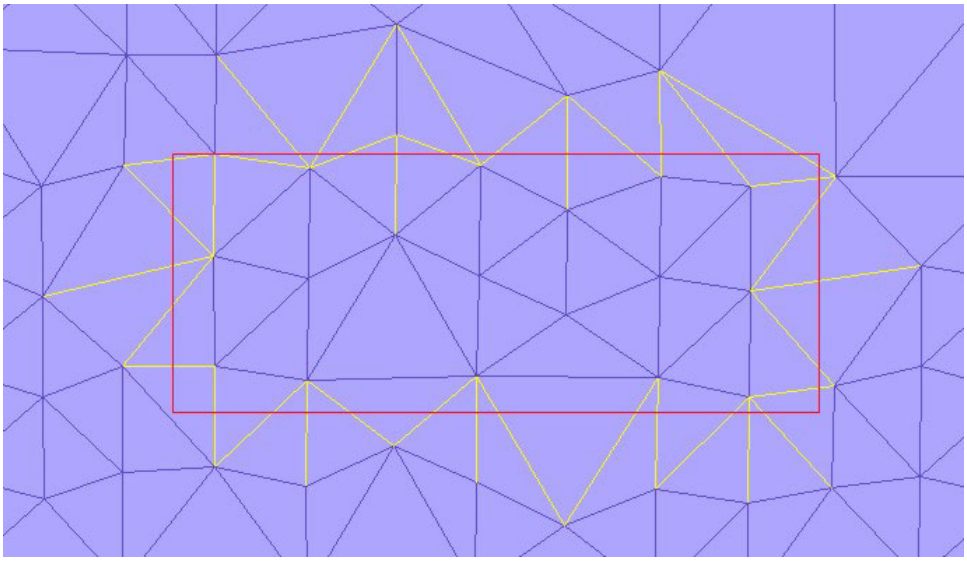
4 Ontwikkelde algoritmen

4.1 Inleiding

In de casestudy is de pan-functionaliteit zoals die in hoofdstuk 2 is geïntroduceerd, gerealiseerd op basis van het in het vorige hoofdstuk beschreven datamodel. Dit hoofdstuk gaat in op de achter de pan-functionaliteit liggende algoritmen. Het zoekproces van alle edges die (gedeeltelijk) in het uitbreidingsgebied liggen, bestaat uit drie fasen. Deze drie fasen worden in drie afzonderlijke paragrafen behandeld. In paragraaf 4.2 wordt beschreven hoe een lijst met edges wordt opgesteld die allen snijden met de grenzen van het uitbreidingsgebied. Vervolgens wordt in paragraaf 4.3 beschreven hoe op basis van deze lijst een soort ring binnen deze ring van snijdende edges wordt geconstrueerd. Paragraaf 4.4 beschrijft tot slot het algoritme waarmee de edges die binnen deze binnenring liggen, worden opgezocht.

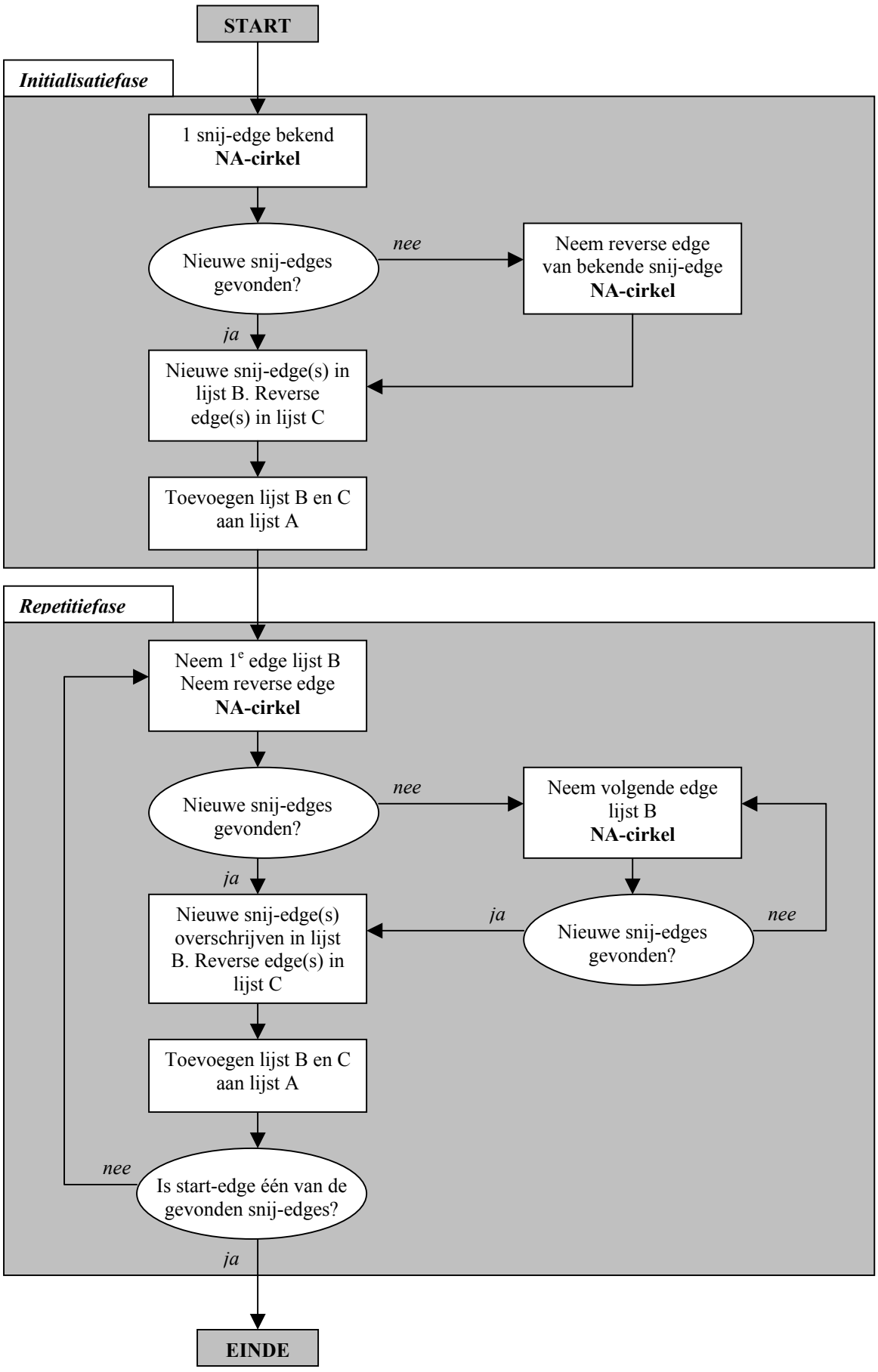
4.2 Het opsporen van edges, snijdend met de uitbreidingsgrenzen

Als uitgangspunt voor het detecteren van de snijdende edges is aangenomen dat één snijdende edge bekend is. Op basis van deze edge dienen vervolgens alle andere snijdende edges gevonden te worden. In figuur 4.1 is te zien dat de ring van snijdende edges een soort zig-zag-patroon vertoont, aangevuld met een aantal “losse” uiteinden.

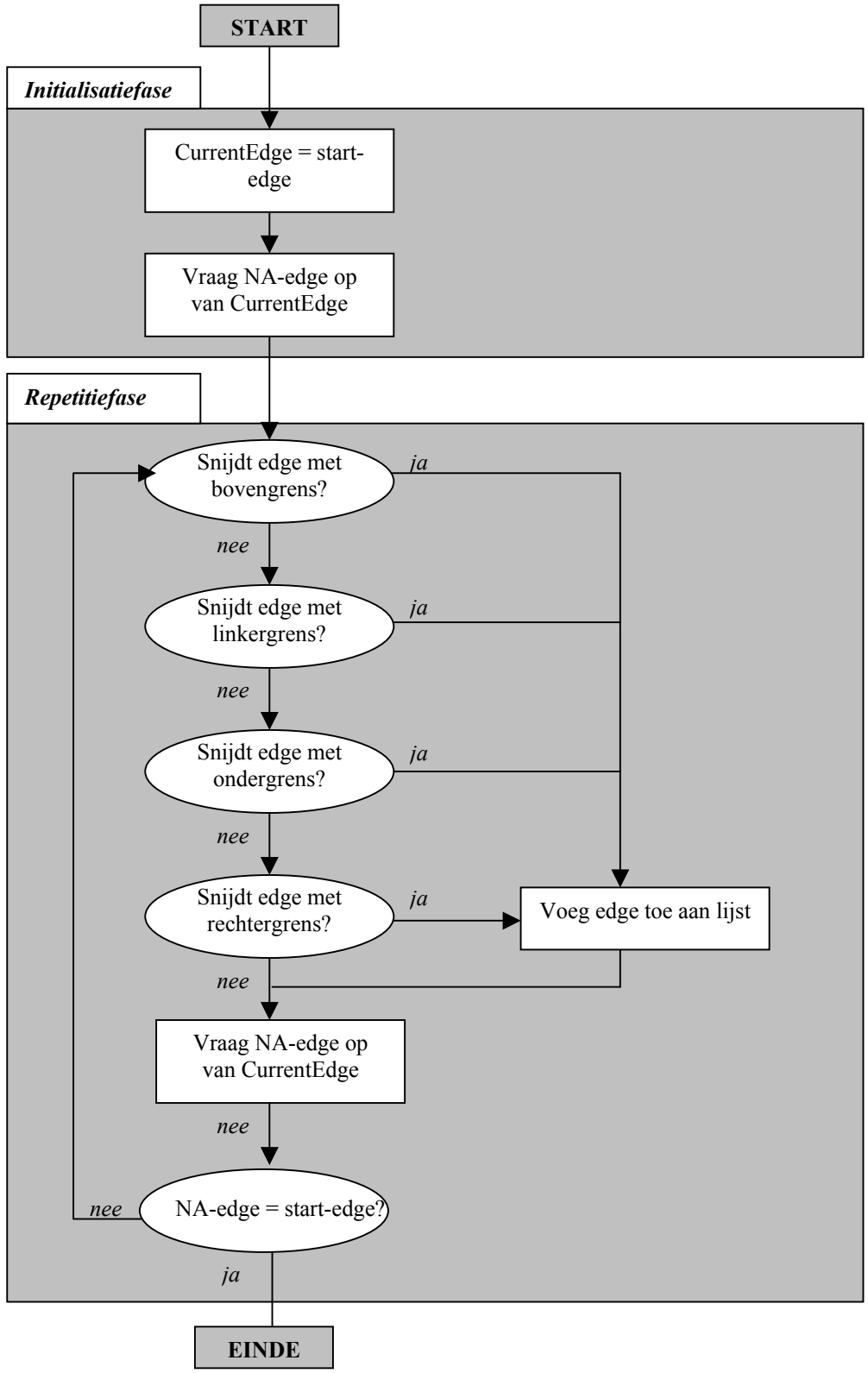


Figuur 4.1 Uitbreidingsgebied (rood) met alle snijdende edges (geel)

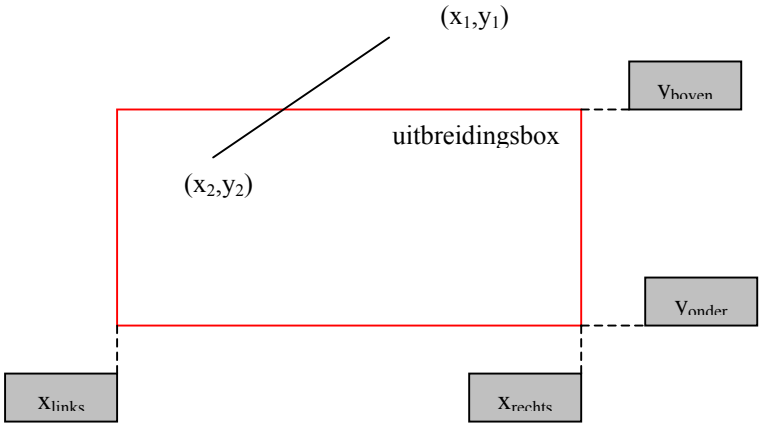
Als men zich nu realiseert dat de edges in het datamodel gericht zijn, is het mogelijk een algoritme te ontwerpen dat op basis van een gevonden snij-edge steeds op zoek gaat naar de volgende snij-edge. Het eindpunt van een snijdende edge is immers het startpunt van de volgende snijdende edge, tenzij het om een los uiteinde gaat. In figuur 4.2 staat het ontworpen algoritme weergegeven. Het start met een initialisatiefase, waarin één snij-edge bekend is. Aan de hand van deze snij-edge wordt gezocht naar de volgende snij-edge(s), door het uitvoeren van het NA-cirkel algoritme (figuur 4.3). In dit algoritme worden alle edges opgevraagd die de eind-node van de huidige snijdende edge als start-node hebben. Dit wordt gedaan door steeds de Na-edge (next edge anti-clockwise) op te vragen, vandaar de naam NA-cirkel. Voor deze edges wordt vervolgens getest of ze snijden met één van de vier grenzen van het uitbreidingsgebied. De testvoorwaarden zijn uitgewerkt in figuur 4.4. Dit alles



Figuur 4.2 Algoritme voor het vinden van alle snijdende edges. Details zijn uitgewerkt in figuren 4.3 en 4.4



Figuur 4.3 NA-cirkel algoritme, onderdeel van het algoritme voor het zoeken van snijdende edges



Implementatie

Snijdt edge met bovengrens?

if ($y_1 \leq y_{boven} \leq y_2$) or ($y_2 \leq y_{boven} \leq y_1$) and

if ($x_{links} \leq ((y_{boven} - y_1) / (y_2 - y_1)) * (x_2 - x_1) + x_1 \leq x_{rechts}$)

snijpunt!

Snijdt edge met linkergrens?

if ($x_1 \leq x_{links} \leq x_2$) or ($x_2 \leq x_{links} \leq x_1$) and

if ($y_{onder} \leq ((x_{links} - x_1) / (x_2 - x_1)) * (y_2 - y_1) + y_1 \leq y_{boven}$)

snijpunt!

Snijdt edge met ondergrens?

if ($y_1 \leq y_{onder} \leq y_2$) or ($y_2 \leq y_{onder} \leq y_1$) and

if ($x_{links} \leq ((y_{onder} - y_1) / (y_2 - y_1)) * (x_2 - x_1) + x_1 \leq x_{rechts}$)

snijpunt!

Snijdt edge met rechtergrens?

if ($x_1 \leq x_{rechts} \leq x_2$) or ($x_2 \leq x_{rechts} \leq x_1$) and

if ($y_{onder} \leq ((x_{rechts} - x_1) / (x_2 - x_1)) * (y_2 - y_1) + y_1 \leq y_{boven}$)

snijpunt!

Figuur 4.4 Uitwerking van voorwaarden om te bepalen of een edge snijdend is

resulteert in een lijst met snij-edges die teruggegeven wordt aan het hoofd-algoritme. Het kan echter zijn dat er geen snijpunten zijn gevonden; dit is het geval als de snijdende edge een los uiteinde is van de kring van snijdende edges. Als dit het geval is, wordt de reverse edge genomen (dezelfde edge in tegenovergestelde richting) en wordt nogmaals het NA-cirkel-algoritme uitgevoerd. Van de gevonden snij-edge(s) wordt vervolgens de bijbehorende reverse-edge opgevraagd. De lijst met gevonden snij-edges (lijst B genaamd in figuur 4.2) en de lijst met bijbehorende reverse-edges (lijst C genaamd in figuur 4.2) worden toegevoegd aan lijst A, de lijst waarop uiteindelijk alle snijdende edges komen te staan.

Na deze initialisatie komt het algoritme in een repeterende fase, die doorgaat totdat de kring van snijdende edges gesloten is. Deze fase is grotendeels identiek aan de initialisatiefase. Uit de lijst met gevonden snij-edges (lijst B) wordt een edge gepakt, waarvoor gezocht wordt naar de volgende snij-edge(s). Als deze niet gevonden worden (wat betekent dat de edge een los uiteinde is van de kring van snijdende edges), wordt de volgende edge uit lijst B gepakt, net zolang tot er nieuwe snij-edges gevonden worden. Van de gevonden edges worden wederom de reverse edges opgevraagd, samen worden deze edges toegevoegd aan de grote lijst met snijdende edges.

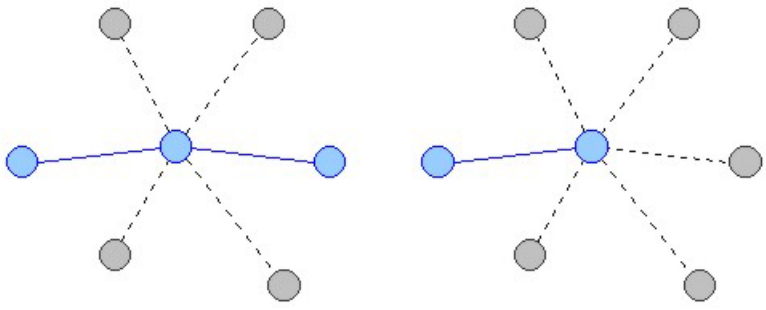
Uiteindelijk resulteert het zoekproces in een lijst met alle snijdende edges, zoals die in figuur 4.1 zijn weergegeven. Bedenk dat achter elke weergegeven edge eigenlijk twee edge schuilen, die in tegengestelde richting zijn gedefinieerd.

4.3 Creëren ring edges binnen ring snij-edges

Nu de edges die gedeeltelijk in het uitbreidingsgebied liggen bekend zijn, kan een begin worden gemaakt met het vinden van alle edges die geheel in het uitbreidingsgebied liggen. Het idee achter het algoritme (figuur 4.6) is verduidelijkt in figuur 4.5. Voor elke gevonden snij-edge worden alle edges met dezelfde start-node opgevraagd. In deze cirkel liggen twee of één snij-edges (in blauw).

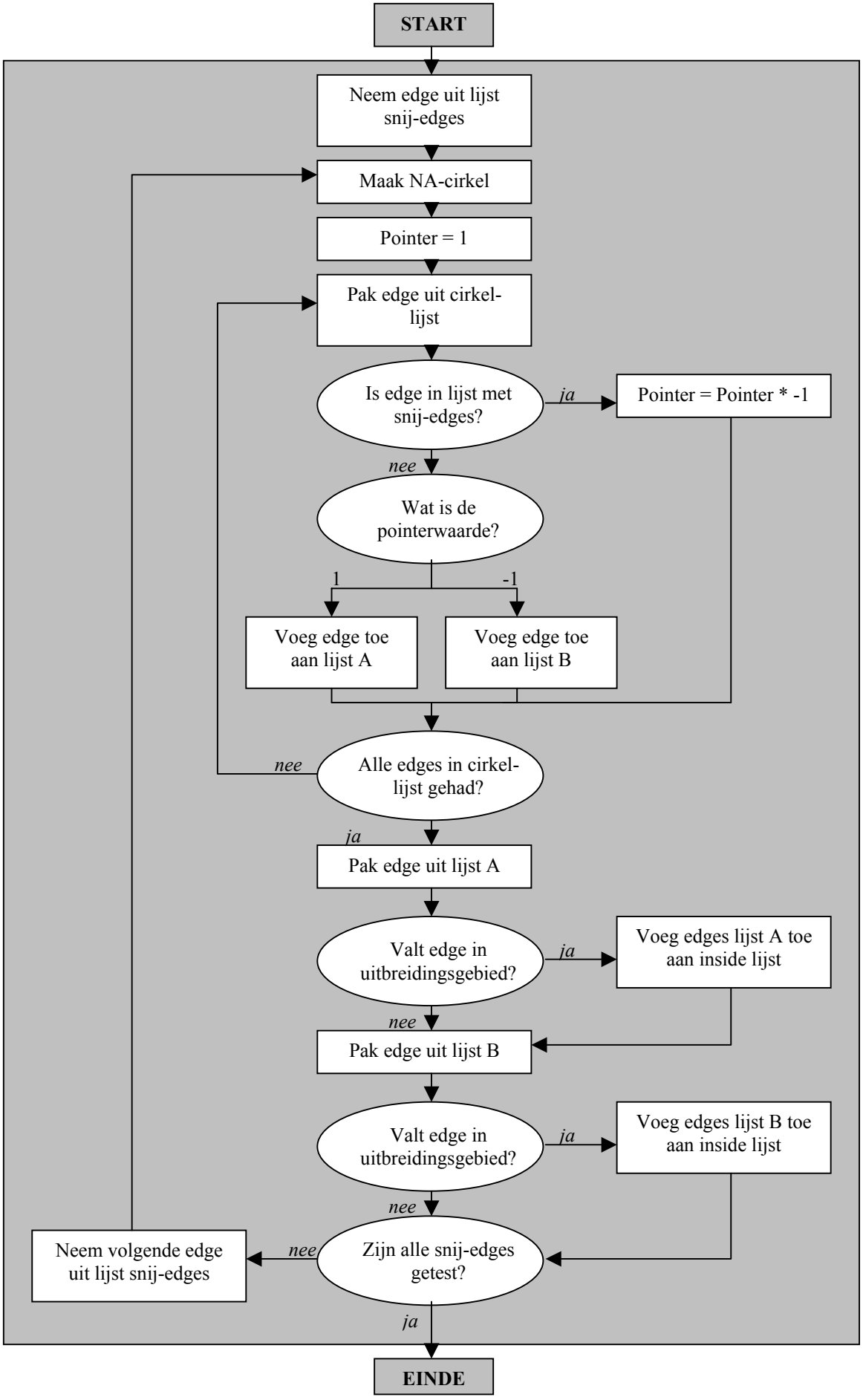
In het eerste geval scheiden deze snij-edges twee delen van de cirkel. Eén van beide delen ligt in het gebied, wat getest wordt door van één edge van beide delen de eind-node op te vragen en met de bijbehorende coördinaten te testen of de edge in of buiten het uitbreidingsgebied ligt. De edges die in het gebied liggen, worden toegevoegd aan de lijst met alle edges in het uitbreidingsgebied.

In het tweede geval wordt de cirkel niet opgedeeld, maar liggen de edges of allemaal in of allemaal buiten het uitbreidingsgebied. Ook hier wordt met behulp van de coördinaten van de eind-node van één edge gekeken of de edges toegevoegd moeten worden aan de lijst met alle edges in het uitbreidingsgebied.



Figuur 4.5 Links: of de edges boven of de edges onder vallen in het uitbreidingsgebied
Rechts: alle edges vallen of in of buiten het uitbreidingsgebied

In figuur 4.6 is het algoritme in schematische vorm uitgewerkt. De lijst met edges die dezelfde start-node hebben, wordt opgebouwd door steeds de NA-edge op te vragen, totdat de cirkel compleet is. Vervolgens worden de edges in de cirkel verdeelt over lijst A en lijst B. In het voorbeeld uit figuur 4.5 zou in het linkerplaatje lijst A bestaan uit de twee bovenste edges en



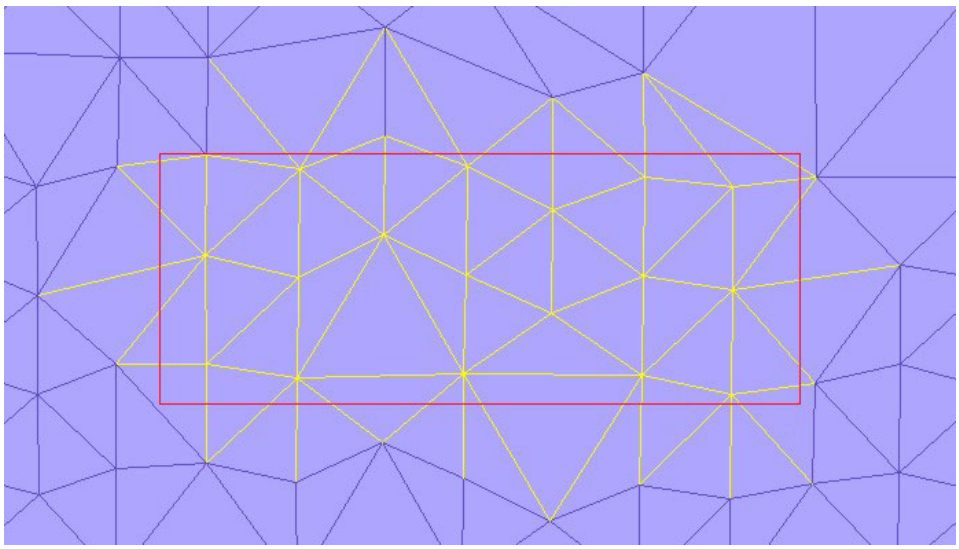
Figuur 4.6 Algoritme voor het creëren van een ring edges binnen de ring snij-edges

lijst B uit de twee onderste edges. De snijdende edges worden dus in geen van beide lijsten opgeslagen. In het rechterplaatje komen alle edges (behalve de snijdende edge) in lijst A terecht. Vervolgens wordt getest welke van de twee lijsten de edges bevat die in het uitbreidingsgebied liggen. Die edges worden toegevoegd aan de inside-list, een tijdelijke lijst waarop de edges van de binnenring staan. Deze edges worden vervolgens (dit is weggelaten in figuur 4.6) uitgelezen en samen met de bijbehorende reverse edges toegevoegd aan de grote lijst met alle edges die (gedeeltelijk) in het uitbreidingsgebied vallen.

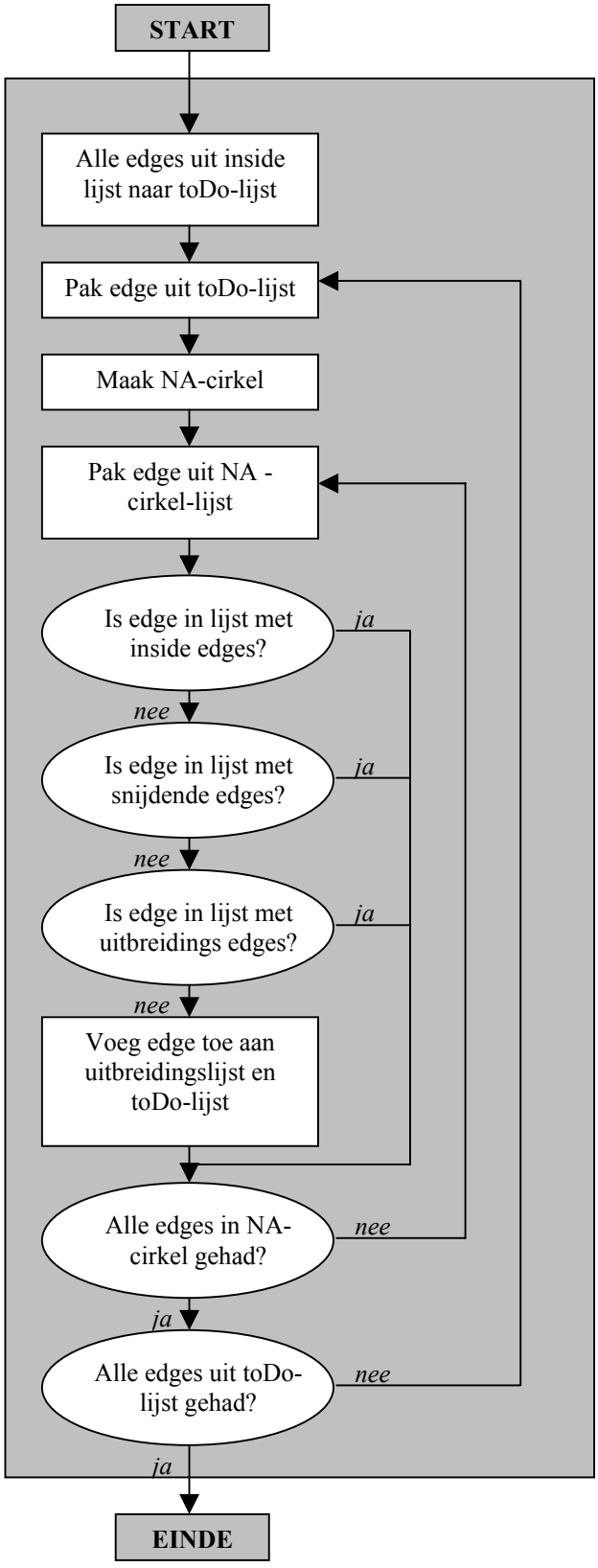
4.4 Opvragen overige edges in uitbreidingsgebied

Als de algoritmen uit de vorige twee paragrafen zijn toegepast, kan de rest van de edges in het uitbreidingsgebied worden opgevraagd. Om te beginnen worden de edges uit de inside-lijst uit het vorige algoritme weggeschreven naar een toDo-lijst. Uit deze lijst wordt een edge gepakt, waarvoor weer alle edges met dezelfde start-node worden opgevraagd met behulp van een NA-cirkel. Voor elk van deze edges wordt gekeken of de edge voorkomt op de lijst met snijdende edges, de lijst met edges uit de binnenring en de lijst met alle edges die (gedeeltelijk) in het uitbreidingsgebied liggen. Komt de edge op geen van deze drie lijsten voor, dan wordt die toegevoegd aan de lijst met edges in het uitbreidingsgebied. Dit algoritme staat uitgewerkt in figuur 4.8. Voor elke edge die wordt toegevoegd, wordt ook de reverse edge opgevraagd en toegevoegd. Dit laatste is weggelaten uit het schematische overzicht in figuur 4.8.

Het toepassen van de algoritmen die in deze en de twee voorgaande paragrafen beschreven staan, leidt tot een lijst waarop alle edges voorkomen die (gedeeltelijk) voorkomen in het uitbreidingsgebied. In figuur 4.7 staan deze edges voor een testgebied weergegeven. Merk op dat de edges binnen dit uitbreidingsgebied (hetzelfde gebied als in figuur 4.1) reeds bekend waren na het uitvoeren van de algoritmen uit de voorgaande twee paragrafen. Als het uitbreidingsgebied groter was geweest, waren er ook door het in deze paragraaf behandelde algoritme nog nieuwe edges toegevoegd. Merk ook op dat het vinden van de overige edges in het uitbreidingsgebied puur op basis van topologie gebeurt, zonder dat er gebruik gemaakt word van coördinaten van de nodes.



Figuur 4.7 De edges (geel) die (gedeeltelijk) in het uitbreidingsgebied liggen



Figuur 4.8 Algoritme voor het vinden van de overige edges in het uitbreidingsgebied

5 Implementatie pan-algoritmen

5.1 Inleiding

Om de in het vorige hoofdstuk beschreven algoritmen ook daadwerkelijk te implementeren is onder andere een behoorlijke hoeveelheid code geschreven. In paragraaf 5.2 komt aan de orde hoe vanuit ArcView een tin is bevraagd en weggeschreven naar een databasefile. Dit tin is vervolgens in Oracle geladen met behulp van SQL*Loader, waar paragraaf 5.3 kort op in gaat. Vervolgens worden in paragraaf 5.4 een aantal achtergronden van JDBC behandeld, de connectie met de database waardoor de database met in Java geschreven code kon worden benaderd. Deze Java-applicatie, die de eigenlijke pan-functionaliteit implementeert, komt tot slot in paragraaf 5.5 aan bod.

5.2 Exporteren tin vanuit ArcView

Zoals in paragraaf 3.3 reeds is opgemerkt, zijn in ArcView in de TINEdge-mode maar een beperkt aantal omliggende edges direct opvraagbaar. Op basis van een aantal kenmerken, waaronder het feit dat in ArcView een tin double directed gedefinieerd is, is gekozen om een double directed winged edge representatie te construeren. Deze representatie leidde ertoe dat één tabel gevuld moest worden met data van het tin.

Het in Avenue geschreven script (zie Bijlage II) verwacht van de gebruiker dat het tin als Active theme in de View is geladen. Verder verwacht het script van de gebruiker een reeds aangemaakte tabel (in DBASE IV formaat) om de resultaten in weg te schrijven. In het stuk code hieronder worden de benodigde velden in de tabel gezocht (vergelijk ook met figuur 3.7).

```
theIDField = theVTab.FindField("Edge_id")
theREField = theVTab.FindField("R_edge")
theNVField = theVTab.FindField("N_vertex")
thePVField = theVTab.FindField("P_vertex")
theNAeField = theVTab.FindField("Na_edge")
theNAnField = theVTab.FindField("Na_neighbo")
theNCeField = theVTab.FindField("Nc_edge")
theNCnField = theVTab.FindField("Nc_neighbo")
thePAeField = theVTab.FindField("Pa_edge")
thePAnField = theVTab.FindField("Pa_neighbo")
thePCeField = theVTab.FindField("Pc_edge")
thePCnField = theVTab.FindField("Pc_neighbo")
theNFField = theVTab.FindField("N_face")
thePFField = theVTab.FindField("P_face")
```

In deze opsomming herkennen we twee edge-id's, één van de desbetreffende edge en de ander van de bijbehorende reverse edge. Vervolgens twee velden voor de n- en de p-vertex, die in deze velden zullen worden weggeschreven als coördinatenpaar, aangezien de nodes in de TINEdge mode van ArcView geen unieke identificatie hebben. De notatie in dit veld zal X@Y@Z zijn, al werkt de pan-functionaliteit slechts in twee dimensies. In de volgende acht velden zijn de twee previous en de twee next edges te herkennen, samen met hun reverse edges. De laatste twee velden geven de twee faces weer. Opgemerkt moet worden dat edges in de TINEdge mode geen uniek identificatienummer hebben. In deze mode heeft elke driehoek (face) een uniek nummer en zijn de drie bijbehorende edges steeds 0, 1 en 2 genummerd.

```
theEdgeString = theStringList.Get(i)
theDigitsLength = theEdgeString.Count - 9
theDigits = theEdgeString.Right(theDigitsLength)
theTwoIndexes = theDigits.AsTokens (", ")
theIDString =
        theTwoIndexes.Get(0).AsString+theTwoIndexes.Get(1).AsString
theID = theIDString.AsNumber
```

Het construeren van unieke edge-id's gebeurt in het bovenstaande stuk code. De lijst bevat een aantal strings waarin de edges als volgt zijn opgebouwd: "TINEdge: 140, 2". In de code wordt eerst het deel "TINEdge: " verwijderd, het overgebleven deel wordt gesplitst, de komma en de spatie verwijderd en vervolgens samengevoegd tot het unieke id 1402. Deze handelswijze heeft als gevolg dat de lijst met id's geen continue reeks is, maar juist van de vorm 10, 11, 12, 20, 21, 22, 30, 31, 32,…Er is bewust voor gekozen om de edge-id's niet om te nummeren tot een serie 1, 2, 3, 4, … , omdat het nu eenvoudig mogelijk blijft om uit de zelf gedefinieerde id terug te redeneren naar de ArcView gecombineerde id bestaande uit TriangleIndex en de interne edge-index. De exacte wijze van uitlezen en wegschrijven van het tin is te vinden in Bijlage II. Uiteindelijk resulteert het runnen van het script in een tabel, waarvan in figuur 5.1 een deel is weergegeven. Merk op dat de eerste drie edge-id's (0, 1 en 2) slaan op de edge's 0, 1 en 2 van driehoek 0 en dus eigenlijk 00, 01, en 02 zijn.

EDGE_ID	N_VERTEX	P_VERTEX	NA_EDGE	NA_NEIGHBO	NC_EDGE	NC_NEIGHBO	PA_EDGE	PA_NEIGHBO	PC_EDGE	PC
0	181944@42821	181944@42821	542	741	540	20	1	3580	2	
1	181944@42821	181947@42821	3581	520	3582	3601	2	3770	0	
2	181947@42821	181944@42821	3771	3872	3772	682	0	541	1	
10	181941@42822	181944@42822	30	1051	31	112	11	3620	12	
11	181944@42822	181944@42822	3621	3550	3622	3530	12	980	10	
12	181944@42822	181941@42822	981	91	982	961	10	32	11	
20	181944@42821	181941@42821	541	0	542	741	21	100	22	
21	181941@42821	181941@42821	101	392	102	130	22	90	20	
22	181941@42821	181944@42821	91	981	92	521	20	540	21	
30	181941@42822	181941@42822	1052	1411	1050	1022	31	112	32	

*Figuur 5.1 Resultaat van het uitlezen van het tin en het wegschrijven naar een *.dbf-file*

5.3 Importeren tin in Oracle

Het was oorspronkelijk de bedoeling het tin in Oracle in te lezen vanuit de in ArcView aangemaakte *.dbf-file. Dit bleek echter lastiger dan verwacht en van meerdere kanten is door ervaren Oracle-gebruikers aangeraden om de *.dbf-file om te zetten in een ASCII-bestand. Dit resulteerde in het bestand tin.data, waarvan in figuur 5.2 een fragment te zien is.

```
0,541,181944@428213@8.5,181944@428216@8.72,542,741,540,20,1,3580,2,3770,54,0
1,3580,181944@428216@8.72,181947@428213@8.5,3581,520,3582,3601,2,3770,0,541,358,0
2,3770,181947@428213@8.5,181944@428213@8.5,3771,3872,3772,682,0,541,1,3580,377,0
10,32,181941@428222@8.67,181944@428223@8.92,30,1051,31,112,11,3620,12,980,3,1
11,3620,181944@428223@8.92,181944@428220@8.61,3621,3550,3622,3530,12,980,10,32,362,1
12,980,181944@428220@8.61,181941@428222@8.67,981,91,982,961,10,32,11,3620,98,1
20,540,181944@428216@8.72,181941@428215@8.56,541,0,542,741,21,100,22,90,54,2
21,100,181941@428215@8.56,181941@42821 9@8.85,101,392,102,130,22,90,20,540,10,2
```

Figuur 5.2 De tin-data in ASCII-formaat

Om deze data vervolgens met behulp van SQL*Loader te kunnen inlezen, dienen er twee bestanden aangemaakt te worden, namelijk een model-file en een control-file. In de model-file wordt de tabel kolom voor kolom gedefinieerd. In Bijlage III zijn de complete files te vinden, maar het definiëren van de tabel gebeurt als volgt:

```
create table tin(ID integer, Re integer,nvertexX number,…
```

De tabel krijgt hier een naam ("tin") en vervolgens worden de kolommen gedefinieerd door het create table-statement kolomnaam en datatype als attribuut mee te geven. Nu met deze model.sql file een tabel is aangemaakt, kan die ingelezen worden. De wijze waarop dit gebeurt, is te vinden in de control-file.

```
load data
infile 'tin.data' "str"
append
```

```
into table tin
fields terminated by ','
trailing nullcols (
        ID integer external(10),
        Re integer external(10),
        nvertexX float external terminated by '@',
```

Aan dit fragment van de controlfile is te zien dat eerst wordt aangegeven in welke file de data te vinden is (tin.data), dan in welke tabel die ingelezen moet worden (tabel tin) en vervolgens hoe de string geïnterpreteerd dient te worden. In dit geval worden de velden gescheiden bij komma's, tenzij anders aangegeven, zoals in de laatste regel waar wordt aangegeven dat de coördinaten gescheiden worden door een @.

Om beide files effect te laten hebben, worden ze uitgevoerd onder UNIX door ze in het programmaatje doit.sh aan te roepen:

```
#!/bin/sh
set -v
sqlplus <user id>/<password> @model.sql
sqlldr  <user id>/<password> CONTROL=tin.ctl
```

Uiteindelijk resulteert dit in het vullen van de tabel en worden een aantal andere files aangemaakt, waaronder het bestand tin.log. In figuur 5.3 is dit bestand te zien, dat informatie geeft over het verloop van het vullen van de tabel. Zo is onder andere te zien dat er 2508 records zijn gelezen en dat dit bij geen enkel record fout is gegaan.

```
SQL*Loader: Release 9.0.1.0.0 - Production on Thu Jun 6 14:46:54 2002

(c) Copyright 2001 Oracle Corporation.  All rights reserved.

Control File:   tin.ctl
Data File:      tin.data
  File processing option string: "str"
  Bad File:     tin.bad
  Discard File:  none specified

 (Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:    none specified
Path used:      Conventional

Table TIN, loaded from every logical record.
Insert option in effect for this table: APPEND
TRAILING NULLCOLS option in effect

   Column Name                  Position   Len  Term Encl
------------------------------ ---------- ----- ---- ----
ID                                 FIRST    10   ,
RE                                  NEXT    10   ,
NVERTEXX                            NEXT     *   @
NVERTEXY                            NEXT     *   @
NVERTEXZ                            NEXT     *   ,
PVERTEXX                            NEXT     *   @
PVERTEXY                            NEXT     *   @
PVERTEXZ                            NEXT     *   ,
NAE                                 NEXT    10   ,
NAN                                 NEXT    10   ,
NCE                                 NEXT    10   ,
NCN                                 NEXT    10   ,
PAE                                 NEXT    10   ,
PAN                                 NEXT    10   ,
PCE                                 NEXT    10   ,
PCN                                 NEXT    10   ,
NF                                  NEXT    10   ,
PF                                  NEXT    10   ,
```

Figuur 5.3 Het bestand tin.log (vervolg op volgende pagina)

```
Table TIN:
  2508 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.


Space allocated for bind array:                 108288 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:          2508
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Thu Jun 06 14:46:54 2002
Run ended on Thu Jun 06 14:46:56 2002

Elapsed time was:     00:00:01.37
CPU time was:         00:00:00.20
```

Figuur 5.3 (vervolg)

5.4 Database connectie JDBC

Om vanuit Javaprogramma's gebruik te kunnen maken van in databases opgeslagen data, zal eerst een verbinding met de database tot stand moeten worden gebracht. Hiervoor bestaan verschillende protocollen, zoals de common gateway interface (CGI), active server pages (ASP), PHP en JDBC. Het belangrijkste voordeel van JDBC is dat dankzij het gebruik van Java de connectie platformonafhankelijk kan werken, waar bijvoorbeeld ASP gebonden is aan Microsoftproducten.

Aangezien er veel verschillende databases bestaan, is voor het functioneren van de database connectie van belang dat er een driver geladen is die de merkspecifieke kenmerken van de database bevat. JDBC bevat zelf een DriverManager, die zoekt naar de juiste driver. Met de onderstaande regel script wordt een driver geregistreerd die communicatie met Oracle databases mogelijk maakt:

```
DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
```

Vervolgens kan de DriverManager opdracht gegeven worden om een connectie tot stand te brengen (zie code). Deze specifieke verbinding krijgt de naam "conn" en is van het objecttype Connection. Om deze connectie conn te maken, wordt de URL van de database gegeven, optioneel aangevuld met een user-id en een password:

```
Connection conn = DriverManager.getConnection
 ("jdbc:oracle:thin:@www. gdmc.nl:1521:geobase","user_id", "password");
```

Nu de connectie een feit is, kan een statement worden opgesteld, dat naar de database gestuurd kan worden. Dit Statement met de naam "stmt" is opgesteld in SQL en kan betrekking hebben op een query, bijvoorbeeld met behulp van het select-commando, of op een update, bijvoorbeeld met commando's als create en drop. Het gecreëerde statement moet vervolgens ook worden uitgevoerd, in dit geval met executeQuery. Tussen aanhalingstekens het SQL-statement, dat ook direct in een sql-omgeving na de prompt had kunnen worden ingetypt. Het gebruikte select-commando heeft de structuur select … from … , optioneel aangevuld met where… . In dit geval wordt de next edge anti-clockwise opgevraagd van een bepaalde edge (bij de edge met id edgeID) uit tabel tin. Het resultaat van de query komt in een object van het type ResultSet, hier rset genaamd.

```
Statement stmt = conn.createStatement ();
ResultSet rset = stmt.executeQuery ("select NAe from tin where
                                                    ID="+edgeID);
```

De resultaten kunnen uit de resultset gelezen worden. In dit geval heeft de resultset de volgende vorm:

```
Nae
1402
```

In de resultset staat de cursor automatisch aan het begin van de eerste regel, waar hier de kolomnaam vermeld staat. Om de cursor naar de regel te krijgen waar de edge-id staat waar we in geïnteresseerd zijn, gebruiken we het commando rset.next(). Vervolgens kan de integer waarde worden uitgelezen en opgeslagen in de variabele naEdge:

```
rset.next();
int naEdge = rset.getInt("NAe");
```

Vervolgens worden het statement, de resultset en de connectie gesloten.

5.5 Implementatie m.b.v. Java

5.5.1 Inleiding

Om de gewenste pan-functionaliteit te implementeren zijn uiteindelijk vijf programma's in Java geschreven. Het programma PanningMain, dat in paragraaf 5.5.2 aan bod komt, bevat de main method en implementeert alle algoritmen. Onderdeel van deze algoritmen zijn een aantal bewerkingen op edges, bijvoorbeeld de bewerking om de next edge anti-clockwise te krijgen. Deze bewerkingen worden mogelijk gemaakt in het programma Edge, onderwerp van paragraaf 5.5.3. Verschillende keren is het voor deze bewerkingen nodig om bevragingen uit te voeren op het tin in de database. Dit wordt geïmplementeerd in het programma Database, dat in paragraaf 5.5.4 wordt behandeld. In paragraaf 5.5.5 komt vervolgens het programma AddBox aan de orde, waarin alle bewerkingen ten aanzien van het uitbreidingsgebied zijn geïmplementeerd. In paragraaf 5.5.6 komt het programma Queue aan bod, nodig om data in queue (een variant op stacks) op te kunnen slaan. De complete code is terug te vinden in Bijlage I. De slotparagraaf 5.5.7 gaat kort in op de resultaten van de implementatie.

5.5.2 PanningMain.java

PanningMain implementeert de in hoofdstuk 4 beschreven algoritmen. Bij het uitvoeren van de taken wordt regelmatig gebruik gemaakt van objecten van de types Edge en AddBox. Naar deze objecten worden messages gestuurd en van deze objecten kunnen ook messages terug worden ontvangen. In onderstaande code wordt een object aangemaakt met de naam theEdge van het objecttype Edge. Vervolgens wordt een message naar dit object gestuurd, namelijk getReverseEdge met het edge ID van theEdge als attribuut. Het object theEdge stuurt vervolgens een integer terug, die opgeslagen wordt in variabele reEdge.

```
Edge theEdge = new Edge(edgeID);
int reEdge = theEdge.getReverseEdge(edgeID);
```

Voor het aanmaken van nieuwe objecten is het vereist dat de objectklasse is gedefinieerd. Veel klassen zijn binnen Java reeds beschikbaar, zoals klassen die de kenmerken van een boolean vastleggen of die van een List. Ook is het mogelijk om zelf nieuwe objectklassen te definiëren, zoals in deze case is gedaan met onder andere de klasse Edge. In PanningMain worden deze objecten gebruikt, in de andere programma-onderdelen (die in de komende paragrafen volgen) worden de objecten en de mogelijke bewerkingen op die objecten gedefinieerd.

PanningMain is in vier delen te onderscheiden, waarvan de eerste drie in hoofdstuk 4 aan bod zijn gekomen. Het programma begint met het vinden van alle edges die snijden met de grenzen van het uitbreidingsgebied. In het tweede deel wordt binnen die ring van snijdende

edges een ring van edges geconstrueerd. In het derde worden ten slotte alle andere edges die in het uitbreidingsgebied vallen, opgezocht. Tot slot worden in het vierde deel de gevonden edges weggeschreven naar een nieuwe tabel in Oracle Spatial. Samen met de algoritmen zoals ze in hoofdstuk 4 te vinden zijn, is het mogelijk de code in Bijlage I.1 te begrijpen. De code begint met het proces om alle snij-edges te vinden. De gevonden volgende snij-edges worden steeds opgeslagen in theIntersectList, die gevraagd wordt aan een Edge-object. De snijdende edges uit theIntersectList worden weggeschreven naar de addList, tezamen met de bijbehorende reverse edges. Voor de edges in de addList wordt gekeken of ze reeds voorkomen in de verzamellijsten van snij-edges allList (alle gevonden snij-edges) en allReverseList (alle bij de edges in allList behorende reverse edges) en desgewenst toegevoegd. Als alle snijdende edges gevonden zijn, worden beide lijsten weggeschreven naar de allEdgesList. Voor een goed begrip van het gebruik van lijsten is het belangrijk om de aandacht te vestigen op het bestaan van de zogenaamde Wrapper klassen. Het idee hierachter is dat in een lijst alleen objecten kunnen worden opgeslagen en geen primitieven zoals integers, floats, characters, enzovoorts. Om dergelijke getallen toch te kunnen opslaan, zijn er Wrapper klassen gedefinieerd, waarmee het mogelijk is een Integer object te creëren dat een bepaalde integer waarde bevat. In onderstaande code wordt uit de lijst allList item i opgevraagd en vervolgens wordt de integerwaarde verkregen via het intValue commando. Omgekeerd kan het commando new Integer (int) gebruikt worden. In de eerste versie van de code werden zowel een variable met de Integer aangemaakt als met de integer-waarde. In de definitieve versie is dit samengepakt, om zo het aantal variabelen en daarmee het geheugengebruik te beperken.

```
int edge = ((Integer) allList.get(i)).intValue();
```

Het tweede deel van PanningMain construeert de ring van edges binnen de snij-edges. Hiervoor worden de edges in allList één voor één afgelopen; de gevonden edges in de binnenring worden opgeslagen in de insideList. Als vervolgens de binnenring compleet is, worden de insideList en de bijbehorende reverse edges ook weer naar de allEdgesList weggeschreven. De insideList is ook het uitgangspunt voor het derde deel van het script, waarin de overige edges in het uitbreidingsgebied worden opgeslagen. Alle edges uit de insideList worden weggeschreven naar de toDoQueue, een speciaal gedefinieerd objecttype dat recursieve bewerkingen mogelijk maakt. In het vierde deel wordt de lijst met edge-ID's weggegeven aan Database om een nieuwe tabel aan te maken met de gevonden edges.

Vreemd genoeg kent Java niet standaard de klassen Queue en Stack. Het mooie van deze twee collecties is dat ze het mogelijk maken om bewerkingen recursief uit te voeren, zonder dat vooraf bekend is hoe vaak de bewerkingen nodig zijn. Een queue en een stack zijn een soort opbergplekken waar steeds objecten aan kunnen worden toegevoegd en van af kunnen worden gepakt. Bewerkingen kunnen dan bijvoorbeeld herhaald worden zolang er nog objecten in de queue of stack zitten. Het verschil tussen een queue en een stack zit in de volgorde waarin objecten de collectie weer verlaten. Voor een queue geldt het principe first in, first out, voor een stack last in, first out. Voor het script PanningMain is dit niet echt een cruciaal verschil, de keuze voor een queue is triviaal.

5.5.3 Edge.java

In het script Edge wordt eerst vastgelegd hoe een nieuw object van het type Edge gedefinieerd wordt. Zo is bij het commando om een nieuw object aan te maken het attribuut edgeID vereist.

```
public Edge(int ID) {
        edgeID = ID;
        ArrayList theCoords = Database.getCoords(ID);
        x1 = ((Double) theCoords.get(0)).doubleValue();
        y1 = ((Double) theCoords.get(1)).doubleValue();
        x2 = ((Double) theCoords.get(2)).doubleValue();
        y2 = ((Double) theCoords.get(3)).doubleValue();
```

Aan de hand van deze ID worden de coördinaten van de n- en de p-vertex opgevraagd en als attributen aan dit object toegevoegd. Vervolgens komen de bewerkingen aan bod die mogelijk

zijn op een object van het type Edge, namelijk getIntersectList, getReverseEdge, getNAelist en getCoords. De eerste bewerking, getIntersectList, is de meest uitgebreide. Dit stuk code implementeert namelijk het algoritme voor het vinden van edges die snijden met de grenzen van het uitbreidingsgebied (AddBox), zoals dat in paragraaf (en figuur) 4.2 is behandeld.

De overige drie bewerkingen zijn een stuk eenvoudiger te doorzien en doen niet veel meer dan het teruggeven van attributen aan het script PanningMain. Deze attributen worden op hun beurt over het algemeen weer gevraagd aan objecten van de typen database en addbox. Ook deze objecttypen zijn zelf gedefinieerd, namelijk in de scripts Database en AddBox.

5.5.4 Database.java

In dit script wordt eerst in een stuk static code eenmalig de DriverManager gezocht en de connectie tot stand gebracht. Dit is een aanpassing ten opzichte van eerdere versies van de code, wat de performance van de applicatie drastisch heeft verbeterd.

```
static {
        try {
                DriverManager.registerDriver (new
                oracle.jdbc.driver.OracleDriver());
                conn = DriverManager.getConnection
                ("jdbc:oracle:thin:@www.gdmc.nl:1521:geobase","<user>",
                "<password>");
                stmt = conn.createStatement(
                ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
```

Vervolgens worden de databasebewerkingen getNAedge, getReverseEdge en getCoords geïmplementeerd. Deze bewerkingen lijken wellicht hetzelfde als de bewerkingen op de klasse Edge en tot op zekere hoogte zijn de bewerkingen op de klasse Edge ook niets meer dat bewerkingen die worden doorgegeven aan de klasse Database. Hiervoor is gekozen omdat dat vanuit objecten geredeneerd netter is. Een edge heeft een next edge anti-clockwise, een edge heeft een reverse edge en een edge heeft coördinaten. Dat al die zaken in een database zitten opgeslagen, doet daar niets aan af. Daarom worden vanuit het main script die zaken aan de edge-objecten gevraagd en worden achter de schermen de waarden uit de database opgevraagd. Hiermee is gepoogd om de leesbaarheid van de main method te vergroten.

Aan het eind van Database wordt de method buildResultTable geprogrammeerd. Deze code is een bewerking van een stuk bestaande code en begint allereerst met het aanmaken van een nieuwe tabel in Oracle Spatial. Ten opzichte van de bestaande tabel is het enige verschil het veld edgegeometry, waarin de edge als ruimtelijk object opgeslagen zal worden. De resultset die bij deze tabel hoort, is updatable. Deze resultset wordt voor elk item geupdated, waarna een complete rij in de resultset in de tabel wordt ingevoegd. Dit wordt gedaan voor elke edge die (gedeeltelijk) in het uitbreidingsgebied ligt.

5.5.5 AddBox.java

Het uitbreidingsgebied is een object van het type AddBox. In het script wordt uitgewerkt hoe een uitbreidingsgebied wordt aangemaakt met de coördinaten y-boven, y-onder, x-links en x-rechts. In het script zijn verder de acties geïmplementeerd om deze coördinaten in te stellen en op te vragen. In een verdere uitbreiding van de panning-functie zullen de coördinaten van de nieuwe uitbreidingsbox afkomstig zijn van de gebruikersinterface in plaats van hard-coded uit de main method.

5.5.6 Queue.java

In paragraaf 5.5.2 over het script PanningMain is al uitgelegd welk nut een queue heeft en waarom de klasse Queue nog apart gedefinieerd moest worden. In het script is vastgelegd hoe nieuwe waarden worden toegevoegd (achteraan), hoe ze opgevraagd en direct verwijderd worden, hoe het aantal items in de collectie kan worden bevraagd en hoe kan worden vastgesteld of een bepaald item reeds in de collectie aanwezig is.

5.5.7 Resultaten implementatie

Uiteindelijk heeft al dit programmeerwerk geresulteerd in een werkende applicatie die op basis van de vier coördinaten en een eerste snij-edge alle edges weet te vinden die geheel of gedeeltelijk in het uitbreidingsgebied liggen. De performance van de applicatie viel in eerste instantie echter tegen, met het vinden van 170 edges in een kleine test was de applicatie bijna twee minuten bezig. In het concept-rapport werd gesteld dat dit waarschijnlijk lag aan het feit dat elke keer dat de database benaderd werd, een nieuwe connectie tot stand werd gebracht. Door het tot stand brengen van de connectie onder te brengen in een stuk static code, is voorkomen dat deze code telkens opnieuw doorlopen wordt. De performance is hierdoor flink verbeterd, het vinden van dezelfde 170 edges kost nu een kleine 12 seconden.

Ten opzichte van de eerste versie van de code zijn een flink aantal wijzigingen doorgevoerd om de code netter en beter overzichtelijk te maken. Bovendien is het aantal benodigde variabelen (en daarmee het geheugenbeslag) flink afgenomen. Qua performance zorgden deze aanpassingen niet tot een merkbaar snellere applicatie. Later in het onderzoek is nog gepoogd om de applicatie te testen met een grotere dataset en een groter uitbreidingsgebied. Dit is niet gelukt doordat er zich een aantal problemen op de PC voordeden, die gerelateerd waren aan de grootte van het uitvoerbestand dat ArcView aanmaakt. Deze problemen zorgden voor veel vertraging waardoor de test niet meer op tijd kon worden uitgevoerd.

6 Conclusie

De uitgevoerde casestudy heeft, zoals reeds in paragraaf 2.3 is vermeld, met name praktische leerdoelen. Hierdoor is er ook geen expliciete onderzoeksvraag gedefinieerd, aangezien een uitgebreider onderzoek, gecombineerd met het aanleren van onder andere Java, niet mogelijk is binnen de beschikbare periode van vijf weken. De hier gepresenteerde conclusies hebben dan ook meer het karakter van samenvattende conclusies omtrent de uitgevoerde werkzaamheden dan dat ze echte onderzoeksconclusies bevatten.

- Het is gelukt een werkend algoritme te bouwen, dat op basis van vier coördinaten en één bekende snij-edge alle edges vindt die (gedeeltelijk) binnen het uitbreidingsgebied liggen.
- Het is niet gelukt om die functionaliteit te presenteren in een grafische user-interface, waarin de gebruiker daadwerkelijk kan pannen. Binnen de beschikbare tijd bleek het niet mogelijk een werkende interface te bouwen. De bibliotheek voor grafische toepassingen binnen Java is redelijk complex, aangezien het aantal objecttypen waarmee je te maken krijgt, snel toeneemt.
- De performance van de gebouwde applicatie is sinds de aanpassing waardoor de database-connectie nog maar een keer wordt gecreeerd, bevredigend. Het vinden van de edges kost voor het testgebied een kleine 12 seconden. Het aanmaken van de nieuwe resultatentabel maakt de applicatie echter trager; de complete bewerking duurt nu zo'n 22 seconden voor het testgebied.
- Ik heb een aardige basiskennis van Java verzameld, waarmee het in ieder geval mogelijk is om in het vervolg niet in onderzoeken stuk te lopen op een gebrek aan praktische vaardigheden. Sinds de concept-versie is veel aandacht besteed aan het verbeteren van de Java-code, onder andere om het geheugenbeslag te verkleinen. Deze wijzigingen zijn zowel alleen als onder begeleiding doorgevoerd.
- Met JDBC, SQL en Oracle Spatial is enige ervaring opgedaan, waarmee het in ieder geval mogelijk is databases te benaderen vanuit Java programma's. Alhoewel het gebruik van SQL beperkt is gebleven tot het select-commando, ben ik nu wel bekend met de online documentatie waarmee het creëren van andere SQL-statements geen onoverkomelijke problemen zou moeten opleveren.
- Met name op het gebied van datarepresentaties is mijn kennis uitgebreid, reden om dit onderwerp in hoofdstuk 3 uitgebreid te behandelen.

Literatuurlijst

Boeken:

- Barnes, D.J., *Object-Oriented Programming with Java, an Introduction*. New Jersey: Prentice Hall, 2000.
- Decker, R. en Hirschfield, S., *programming.java, An Introduction to Programming Using Java*. Pacific Grove: Brooks/Cole, 2000.
- Speegle, G.D., *JDBC: Practical Guide for Java Programmers*. San Francisco: Morgan Kaufmann, 2002.
- Worboys, M., *GIS - A computing perspective*. London: Taylor and Francis, 1995.

Software:

- *ArcView*, versie 3.2, Redlands, Environmental Systems Research Institute.
- *Java 2 Platform, Standard Edition,* versie 1.4.1, Santa Clara, Sun Microsystems.

Internet:

- *Java 2 SDK, Standard Edition Documention*:
  http://www.gdmc.nl/qual/intern/java/j2sdg1.4.0/
  geraadpleegd tussen 25 mei 2002 en 27 juni 2002
- *Oracle Concepts*: http://www.gdmc.nl/oradoc
  geraadpleegd tussen 25 mei 2002 en 5 juni 2002
- *Oracle SQL*Plus Quick Reference*: http://www.gdmc.nl/oradoc
  geraadpleegd tussen 25 mei 2002 en 5 juni 2002
- *Oracle SQL Reference*: http://www.gdmc.nl/oradoc
  geraadpleegd tussen 25 mei 2002 en 5 juni 2002

Bijlage I: Javacode

I.1 PanningMain.java

```
import java.util.*;
import java.sql.*;
import java.math.*;
import java.io.*;
import java.awt.*;

class PanningMain {
    public static void main (String [] args) throws Exception {
        AddBox box = new AddBox(428258.00, 428249.46 , 181967.94, 181988.50);
//
// Phase 1: Find all edges intersecting the box edges
//

        // Phase 1.1: Initialization phase
        ArrayList allEdgesList = new ArrayList();
        ArrayList allList = new ArrayList();
        ArrayList allReverseList = new ArrayList();

        // initialize start-edge, store this edge and re-edge in lists
        int edgeID = 4400;
        allList.add(new Integer(edgeID));
        int startEdge = edgeID;
        Edge theEdge = new Edge(edgeID);
        int reEdge = theEdge.getReverseEdge(edgeID);
        allReverseList.add(new Integer(reEdge));

        // get intersections from startedge
        ArrayList theIntersectList = theEdge.getIntersectList(box);

        // test if intersections have been found
        boolean empty = theIntersectList.isEmpty();

        // take reverse edge and find intersections if no intersections have
                                                                been found yet
        if (empty == true) {
            edgeID = reEdge;
            startEdge = edgeID;
            theEdge = new Edge(edgeID);
            theIntersectList = theEdge.getIntersectList(box);
        }

        // add intersecting edges to addList and their reverses to
                                                            allReverseList, if
        // the edges do not appea already in the lists
        int listSize = theIntersectList.size();
        for (int i=0; i < listSize; i++) {
            int itemi = ((Integer) theIntersectList.get(i)).intValue();
            Edge theTempEdge = new Edge(itemi);

            boolean alreadyIn = allList.contains(new Integer(itemi));
            boolean alreadyIn2 = allReverseList.contains(new Integer(itemi));
            if ((alreadyIn == false) && (alreadyIn2 == false)) {
                allList.add(new Integer(itemi));
            }
            alreadyIn = allList.contains(new Integer
                                    (theTempEdge.getReverseEdge(itemi)));
            alreadyIn2 = allReverseList.contains(new Integer
                                    (theTempEdge.getReverseEdge(itemi)));
            if ((alreadyIn == false) && (alreadyIn2 == false)) {
                allReverseList.add(new Integer
                                    (theTempEdge.getReverseEdge(itemi)));
            }

        }
```

```
// Phase 1.2: Repetition phase

// repete until the next intersecting edge is the startedge
boolean loopComplete = false;
do {
    // take first edge from intersectlist, get reverse edge, search
                                                    intersections
    ArrayList orderList = new ArrayList(theIntersectList);
    int edgePointer = 0;
    int firstEdge = ((Integer) orderList.get(edgePointer)).intValue();
    Edge theFirstEdge = new Edge(firstEdge);
    int firstReEdge = theFirstEdge.getReverseEdge(firstEdge);
    Edge theFirstReEdge = new Edge (firstReEdge);
    theIntersectList = theFirstReEdge.getIntersectList(box);

    // test if intersections have been found
    empty = theIntersectList.isEmpty();

    // if no intersections have been found, take next edge and search
                                                    intersections
    // as long as no intersections have been found
    while (empty == true) {
        edgePointer += 1;
        int nextEdge = ((Integer)
                            orderList.get(edgePointer)).intValue();
        Edge theNextEdge = new Edge(nextEdge);
        int nextReEdge = theNextEdge.getReverseEdge(nextEdge);
        Edge theNextReEdge = new Edge (nextReEdge);
        theIntersectList = theNextReEdge.getIntersectList(box);
        empty = theIntersectList.isEmpty();
    }

    // store intersecting edges in addlist and their reverses in
                                                    allReverselist
    listSize = theIntersectList.size();

    for (int i=0; i < listSize; i++) {
        int itemi = ((Integer) theIntersectList.get(i)).intValue();
        Edge theTempEdge = new Edge(itemi);
        int tempReverse = theTempEdge.getReverseEdge(itemi);
        boolean alreadyIn = allList.contains(new Integer (itemi));
        boolean alreadyIn2 = allReverseList.contains(new Integer
                                                    (itemi));
        if ((alreadyIn == false) && (alreadyIn2 == false)) {
            allList.add(new Integer (itemi));
        }
        alreadyIn = allList.contains(new Integer(tempReverse));
        alreadyIn2 = allReverseList.contains(new Integer(tempReverse));
        if ((alreadyIn == false) && (alreadyIn2 == false)) {
            allReverseList.add(new Integer(tempReverse));
        }
        if (itemi == startEdge) {
            loopComplete = true;
        }
    }
} while (loopComplete == false);

// adding allList to allEdgesList
int allSize = allList.size();
for (int i=0; i < allSize; i++) {
    Integer edgei = (Integer) allList.get(i);
    allEdgesList.add(edgei);
}

// adding allReverseList to allEdgesList; print results
int reverseSize = allReverseList.size();
for (int i=0; i < reverseSize; i++) {
    Integer edgei = (Integer) allReverseList.get(i);
    allEdgesList.add(edgei);
}
```

```
//
// Phase 2: Finding inside edges of AddBox, using intersecting edges as
// boundaries
//

        allSize = allList.size();
        ArrayList insideList = new ArrayList();


        // for each intersecting edge the na-circlelist will be searched
        for (int i=0; i < allSize; i++) {
            int intersectEdge = ((Integer) allList.get(i)).intValue();
            Edge theIntersectEdge = new Edge(intersectEdge);
            ArrayList theNAeList = theIntersectEdge.getNAeList(intersectEdge);

            // sorting the nae-list in a possible inside and outside part
            int naeSize = theNAeList.size();
            ArrayList listA = new ArrayList();
            ArrayList listB = new ArrayList();
            int listPointer = 1;
            for (int j=0; j < naeSize; j++) {
                Integer testeri = (Integer) theNAeList.get(j);
                boolean isIntersecting = allEdgesList.contains(testeri);

                if (isIntersecting == true) {
                    listPointer = listPointer * -1;
                }
                else {
                    if (listPointer == 1) {
                        listA.add(testeri);
                    }
                    if (listPointer == -1) {
                        listB.add(testeri);
                    }
                }
            }

            // testing if listA contains inside edges, if so: add to insideList
            boolean onlyOneList = listB.isEmpty();
            boolean listAempty = listA.isEmpty();

            if (listAempty == false) {
                int testEdge = ((Integer) listA.get(0)).intValue();
                Edge theTestEdge= new Edge(testEdge);

                double xpvertex = theTestEdge.x2;
                double ypvertex = theTestEdge.y2;

                if ((box.getLeft() <= xpvertex) && (xpvertex <=
                                                    box.getRight())) {
                    if ((box.getBottom() <= ypvertex) && (ypvertex <=
                                                    box.getTop())) {
                        int aSize = listA.size();
                        for (int k=0; k <  aSize; k++) {
                            Integer itemi = (Integer) listA.get(k);
                            boolean alreadyInside = insideList.contains(itemi);
                            if (alreadyInside == false) {
                                insideList.add(itemi);
                            }

                        }
                    }
                }
            }

            // if listB isnot empty,testing if listB contains inside edges, if
                                                    so: add to insideList
            onlyOneList = listB.isEmpty();
            if (onlyOneList == false) {
                int testEdge2 = ((Integer) listB.get(0)).intValue();
                Edge theTestEdge2= new Edge(testEdge2);

                double xpvertex2 = theTestEdge2.x2;
```

```
                double ypvertex2 = theTestEdge2.y2;

                if ((box.getLeft() <= xpvertex2) && (xpvertex2 <=
                                                    box.getRight()))  {
                    if ((box.getBottom() <= ypvertex2) && (ypvertex2 <=
                                                    box.getTop())) {
                        int bSize = listB.size();
                        for (int l=0; l < bSize; l++) {
                            Integer itemi = (Integer) listB.get(l);
                            boolean alreadyInside = insideList.contains(itemi);
                            if (alreadyInside == false) {
                                insideList.add(itemi);
                            }
                        }
                    }
                }
            }
        }


        // writing reverses of insidelist to toDoQueue, writing insidelist +
                                                    reverses to allEdgesList
        Queue toDoQueue = new Queue();
        int allEdgesSize = allEdgesList.size();
        int insideSize = insideList.size();
        int minCapacity = insideSize * 2 + allEdgesSize;
        allEdgesList.ensureCapacity(minCapacity);

        for (int i=0; i < insideSize; i++) {
            int edge = ((Integer) insideList.get(i)).intValue();
            Edge thaEdge = new Edge(edge);
            int thaReverse = thaEdge.getReverseEdge(edge);
            toDoQueue.addID(thaReverse);
            allEdgesList.add(new Integer(edge));
            allEdgesList.add(new Integer(thaReverse));
        }

//
// Phase 3: Finding all inside edges, using intersecting and inside ring as
// boundaries
//

        // for each edge in queue all edges with the same n-vertex wil be
       // searched if these edges do not appear in the list with all inside
      // edges (which include the edges found in phase 1 and 2), these edges
       // will be added to the list with inside edges and tot the queue


        int edgesToDo = toDoQueue.size();
        while (edgesToDo > 0) {
            int currentID = toDoQueue.getID();
            Edge currentEdge = new Edge(currentID);
            ArrayList currentNAeList = currentEdge.getNAeList(currentID);
            int currentSize = currentNAeList.size();
            for (int i=0; i < currentSize; i++) {
                int currentNAe = ((Integer) currentNAeList.get(i)).intValue();
                Edge curEdge = new Edge(currentNAe);
                int curReEdge = curEdge.getReverseEdge(currentNAe);
                boolean containsEdge = allEdgesList.contains(new
                                                    Integer(currentNAe));
                boolean containsReEdge = allEdgesList.contains(new
                                                    Integer(curReEdge));
                if (containsEdge == false) {
                    toDoQueue.addID(currentNAe);
                    allEdgesSize = allEdgesList.size();
                    minCapacity = allEdgesSize + 1;
                    allEdgesList.ensureCapacity(minCapacity);
                    allEdgesList.add(new Integer (currentNAe));
                }
                if (containsReEdge == false) {
                    toDoQueue.addID(curReEdge);
                    allEdgesSize = allEdgesList.size();
                    minCapacity = allEdgesSize + 1;
```

```
                    allEdgesList.ensureCapacity(minCapacity);
                    allEdgesList.add(new Integer(curReEdge));
                }
            }
            edgesToDo = toDoQueue.size();
        }

//
// Phase 4: exporting results to new Oracle Spatial table
//

        Database.buildResultTable(allEdgesList);

    }
}
```

I.2 Edge.java

```
import java.util.*;
import java.sql.*;
import java.math.*;
import java.io.*;
import java.awt.*;

class Edge {

    private int edgeID;

    public int getID() {
        return edgeID;
    }

    public double x1;
    public double x2;
    public double y1;
    public double y2;

    public String toString() {
        return("Edge(" + edgeID + "," + x1 + "," + x2 + "," + y1 + "," + y2 +
                                                                    ")");
    }

    public Edge(int ID) {
        // define an edge with ID and 2d-coords as attributes
        edgeID = ID;
        ArrayList theCoords = Database.getCoords(ID);
       x1 = ((Double) theCoords.get(0)).doubleValue();
       y1 = ((Double) theCoords.get(1)).doubleValue();
       x2 = ((Double) theCoords.get(2)).doubleValue();
       y2 = ((Double) theCoords.get(3)).doubleValue();
    }

    public ArrayList getIntersectList (AddBox thebox) {

        // initialize Na-circle: create intersectList, startEdge, get naEdge
                                                            from database
        ArrayList intersectList = new ArrayList();
        int startEdgeID = getID();
        int naEdgeID = Database.getNAedge(getID());

        // find all edges with the same n-vertex as startEdge, test for each
                                                            edge if this
        // edge is intersecting. if so, add edge to intersectlist

        do {
            Edge theNAedge = new Edge(naEdgeID);
             if (thebox.intersects(theNAedge)) {
                 intersectList.add(new Integer (theNAedge.getID()));
             }
            naEdgeID = Database.getNAedge(theNAedge.getID());
```

```
        } while (naEdgeID != startEdgeID);

        return intersectList;
    }

    public int getReverseEdge (int ID) {

        // input: ID (integer) output: ID (int) of reverse edge
        try {
            return Database.getReverseEdge(ID);
        }
        catch(SQLException e) {
            System.out.println(e.getMessage());
           return 0;
        }
    }

    public ArrayList getNAeList (int ID) {

        // input: ID (int) output: list with ID's  (int) of edges with same n-
                                                                    vertex
        ArrayList naeList = new ArrayList();
        int startEdge = ID;
         int naEdge = Database.getNAedge(ID);

         do {
             naeList.add(new Integer(naEdge));
             ID = naEdge;
             naEdge = Database.getNAedge(ID);
         } while (naEdge != startEdge);

         return naeList;
    }

}
```

I.3 Database.java

```
import oracle.sql.STRUCT;
import oracle.jdbc.driver.*;
import oracle.sdoapi.OraSpatialManager;
import oracle.sdoapi.adapter.*;
import oracle.sdoapi.geom.*;
import oracle.sdoapi.util.*;
import java.sql.*;
import java.math.*;
import java.io.*;
import java.awt.*;
import java.util.*;

class Database {

    public Database() {
    }


    // open connection to database
    public static Connection conn = null;
    public static Statement stmt = null;

    static {
        try {
            DriverManager.registerDriver (new oracle.jdbc.driver.
                                                        OracleDriver());
            conn = DriverManager.getConnection
                ("jdbc:oracle:thin:@www.gdmc.nl:1521:geobase","<user>",
                                                        "<password>");

           stmt = conn.createStatement(
               ResultSet.TYPE_SCROLL_SENSITIVE,
```

```
            ResultSet.CONCUR_UPDATABLE);
    }
    catch(SQLException e){
        System.err.println("SQLE:" + e);
    }
}

public static int getNAedge(int edgeID) {

    try {
        ResultSet rset = stmt.executeQuery ("select NAe from tin where
                                                              ID="+edgeID);
        rset.next();
        int naEdge = rset.getInt("NAe");
        rset.close();

        return naEdge;
    }
    catch(SQLException e) {
        System.out.println(e.getMessage());
        return 0;
    }
}

public static int getReverseEdge(int edgeID) throws SQLException {

    ResultSet rset = stmt.executeQuery ("select Re from tin where
                                                              ID="+edgeID);
    rset.next();
    int reEdge = rset.getInt("Re");
    rset.close();

    return reEdge;
}

public static ArrayList getCoords(int naEdge) {
    try {
        ResultSet rset = stmt.executeQuery ("select nvertexX, nvertexY,
                          pvertexX, pvertexY from tin where ID="+naEdge);
        rset.next();
        double x1 = rset.getDouble("nvertexX");
        double y1 = rset.getDouble("nvertexY");
        double x2 = rset.getDouble("pvertexX");
        double y2 = rset.getDouble("pvertexY");

        // close resultset, statement, connection
        rset.close();
        ArrayList naCoords = new ArrayList();
        naCoords.add(new Double(x1));
        naCoords.add(new Double(y1));
        naCoords.add(new Double(x2));
        naCoords.add(new Double(y2));

        // return coordinates of edge
        return naCoords;
    }
    catch(SQLException e) {
        System.err.println("Database.java:80: " + e);
        return null;
    }

}

public static void buildResultTable (ArrayList edgeList) throws Exception {
    System.err.println("Enter buildResultTable");

    ArrayList allEdgesList = edgeList;
    String tablename = "frisotable";

    // Get GeometryAdapter, that converts our geometry object to oracle
                                                                  STRUCT.
    GeometryAdapter sdoAdapter =
        OraSpatialManager.getGeometryAdapter("SDO", "8.1.6",
```

```
                                        STRUCT.class, null, null, conn);

// With these array we create two strings:
//    createtable -- the create table statement that we send to oracle.
//     attributelist -- a list with the attributes that we need to
//     update the table. This second table is only needed because
//     of an oracle bug.

String createtable = "create table " + tablename +" (" +
    "edgegeometry mdsys.sdo_geometry," +
    "ID integer," +
    "Re integer," +
    "nvertexX number," +
    "nvertexY number," +
    "nvertexZ number," +
    "pvertexX number," +
    "pvertexY number," +
    "pvertexZ number," +
    "NAe integer," +
    "NAn integer," +
    "NCe integer," +
    "NCn integer," +
    "PAe integer," +
    "PAn integer," +
    "PCe integer," +
    "PCn integer," +
    "NF integer," +
    "PF integer" +
    ")";
String attributelist = "edgegeometry,ID,Re,nvertexX,nvertexY,nvertexZ,"
 + "pvertexX, pvertexY,pvertexZ,NAe,NAn,NCe,NCn,PAe,PAn,PCe,PCn,NF,PF";


// Create the table in which we want to put the shapefile.
// In case the table exists we drop the old table first.

 System.err.println("dropping old table");
try {
    stmt.executeUpdate("drop table " + tablename);
}
catch(SQLException e) {

   // If the drop statements gave an error because the table did not
                                                          exists
   // we can ignore the exception. Otherwise we rethrow it.

   if (e.toString().indexOf("ORA-00942") == -1) {
       System.err.println("Catching exception in Database.java:154" +
                                                       e);
       throw e;
   }
}

 System.err.println("creating new table '" + tablename + "'");
 System.err.println("create statement :  '" + createtable + "'");
stmt.executeUpdate(createtable);

// Now we need an Updatable ResultSet that points to this table.
       Normally
// we can get such a set by executing:
//    "select * from tablename"
// but because of an Oracle bug we currently execute:
//    "select att1,att2,att3 from tablename"

 System.err.println("acces table");
    Statement stmt2 = conn.createStatement(
        ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_UPDATABLE);
ResultSet resultset = stmt2.executeQuery("select " + attributelist + "
                                    from " + tablename);

// Get a geometryfactory.
GeometryFactory factory = OraSpatialManager.getGeometryFactory();
```

```
// This was the preparation, now we are ready to insert the edges.
// below we insert 10  edge as an example.
int numberAllEdges = allEdgesList.size();
for (int i=0; i < numberAllEdges; i++) {
     int id = ((Integer) allEdgesList.get(i)).intValue();

     ResultSet rset = stmt.executeQuery ("select * from tin where
                                                    ID="+id);
     rset.next();
     int re = rset.getInt("Re");
     double nx = rset.getDouble("nvertexX");
     double ny = rset.getDouble("nvertexY");
     double nz = rset.getDouble("nvertexZ");
     double px = rset.getDouble("pvertexX");
     double py = rset.getDouble("pvertexY");
     double pz = rset.getDouble("pvertexZ");
     int nae = rset.getInt("NAe");
     int nan = rset.getInt("NAn");
     int nce = rset.getInt("NCe");
     int ncn = rset.getInt("NCn");
     int pae = rset.getInt("PAe");
     int pan = rset.getInt("PAn");
     int pce = rset.getInt("PCe");
     int pcn = rset.getInt("PCn");
     int nf = rset.getInt("NF");
     int pf = rset.getInt("PF");

     // close resultset
     rset.close();

    // Start editing a new record in the table.
    resultset.moveToInsertRow();

    // First we insert the geometrty of the shapefile. This takes some
    // steps:
    //   - We get the SHPGeometry from the shapefile.
    //   - This we covert to an sdoapi geometry with ShapeToGeometry.
    //   - Using the sdoAdapter we covert this to an oracle STRUCT.
    //   - this is then inserted into the InsertRow.

    double[] linecoords = { nx , ny, px, py};
    Geometry g = factory.createLineString(linecoords);
    STRUCT dbObject =
              (STRUCT)sdoAdapter.exportGeometry(STRUCT.class,g);
    resultset.updateObject("edgegeometry" , dbObject);

    resultset.updateInt("ID",id);
    resultset.updateInt("Re",re);
    resultset.updateDouble("nvertexX",nx);
    resultset.updateDouble("nvertexY",ny);
    resultset.updateDouble("nvertexZ",nz);
    resultset.updateDouble("pvertexX",px);
    resultset.updateDouble("pvertexY",py);
    resultset.updateDouble("pvertexZ",pz);
    resultset.updateInt("NAe",nae);
    resultset.updateInt("NAn",nan);
    resultset.updateInt("NCe",nce);
    resultset.updateInt("NCn",ncn);
    resultset.updateInt("PAe",pae);
    resultset.updateInt("PAn",pan);
    resultset.updateInt("PCe",pce);
    resultset.updateInt("PCn",pcn);
    resultset.updateInt("NF",nf);
    resultset.updateInt("PF",pf);

    // Now we insert the complete record we created in the inserteRow
                                                    into
    // the database.
    resultset.insertRow();
    System.err.println("record with id " + id + " inserted");
}
```

```
        //
        // That's All Folks.
        //
        System.err.println("done");
    }
}
```

I.4 AddBox.java

```
import java.util.*;
import java.sql.*;
import java.math.*;
import java.io.*;
import java.awt.*;

class AddBox {
    public AddBox(double yt, double yb, double xl, double xr) {
        setTop(yt);
        setBottom(yb);
        setLeft(xl);
        setRight(xr);
    }

    public void setTop(double top) {
        ytop = top;
    }

    public void setBottom(double bottom) {
        ybottom = bottom;
    }

    public void setLeft(double left) {
        xleft = left;
    }

    public void setRight(double right) {
        xright = right;
    }

    public double getTop() {
        return ytop;
    }

    public double getBottom() {
        return ybottom;
    }

    public double getLeft() {
        return xleft;
    }

    public double getRight() {
        return xright;
    }

    private double ytop;
    private double ybottom;
    private double xleft;
    private double xright;

    public String toString() {
       return("Box(" + xleft + "," + ytop + "," + xright + "," + ybottom +
                                      ")");
    }

    /**
      * Return true if Edge e intersects with the boundary line of this box
      */
```

```
    public boolean intersects(Edge e) {
        if ((ytop <= e.y1) ^ (ytop <= e.y2)) {
            double xcross = ((ytop - e.y1)/(e.y2 - e.y1)) * (e.x2 - e.x1) +
                                                            e.x1;

            if ((xleft <= xcross) && (xcross <= xright)) {
                return true;
            }
        }
        if ((xleft <= e.x1) ^ (xleft <= e.x2)) {
            double ycross = ((xleft - e.x1)/(e.x2 - e.x1)) * (e.y2 - e.y1) +
                                                            e.y1;

            if ((ybottom <= ycross) && (ycross <= ytop)) {
                return true;
            }
        }
        if ((ybottom <= e.y1) ^ (ybottom <= e.y2)) {
            double xcross = ((ybottom - e.y1)/(e.y2 - e.y1)) * (e.x2 - e.x1) +
                                                            e.x1;

            if ((xleft <= xcross) && (xcross <= xright)) {
                return true;
            }
        }
        if ((xright <= e.x1) ^ (xright <= e.x2)) {
           double ycross = ((xright - e.x1)/(e.x2 - e.x1)) * (e.y2 - e.y1) +
                                                            e.y1;

           if ((ybottom <= ycross) && (ycross <= ytop)) {
                return true;
            }
        }

        // The segment does not intersect with any of the boundary lines.
        // we can return false.
        return false ;
    }
}
```

I.5 Queue.java

```
import java.util.*;
import java.sql.*;
import java.math.*;
import java.io.*;
import java.awt.*;

class Queue {
    public void addID (int ID) {
        getQueue().add(new Integer(ID));
    }

    public int getID () {
        int ID = ((Integer) getQueue().removeFirst()).intValue();
        return ID;
    }

    public int size() {
        return getQueue().size();
    }

    public boolean contains (int ID) {
        boolean contain = getQueue().contains(new Integer(ID));
        return contain;
    }

    private LinkedList getQueue() {
        return queue;
```

```
    }

    private LinkedList queue = new LinkedList();
    private int ID;
    private Integer IDi;

}
```

Bijlage II: ArcViewscript

```
' ================================================================================

' TINEdge --> (existing) .dbf-file
' using double directed winged-edge representation

' ================================================================================

' Description:
' ---
' For each edge the following elements are being stored:
'  edge-id, r-edge, n-vertex, p-vertex, n-face, p-face,
'  na-edge, nc-edge, pa-edge, pc-edge.
' As TIN's are double directed in ArcView, the following
' items are added to the standard winged-edge representation:
'  na-neighbor, nc-neighbor, pa-neighbor, pc-neighbor.
' These neighbors are the same edges, but in opposite direction

' Requires:
' ---
' Database with fields, for instance build in MS Access
' with the above mentioned fields

' By Friso Penninga, May 2002
' ================================================================================

MsgBox.Banner("titlebanner.gif".AsFileName, 1, "TIN 2 Double Directed Winged
Edge Representation")


' ===
' Reading TIN from View
' ===

theView = av.GetActiveDoc
theActiveTheme = theView.GetActiveThemes

if (theActiveTheme.Count = 0) then
   MsgBox.Error ("No active theme in View","Fatal error")
   return NIL
end

if (theActiveTheme.Count > 1) then
   MsgBox.Error ("Too many active themes in View","Fatal error")
   return NIL
end

theSTheme = theActiveTheme.Get(0)
theTIN = theSTheme.GetSurface
theNumTriangles = theTIN.GetNumTriangles


' ===
' Retrieving output databasefile
' ===

theDBFName = MsgBox.Input("Please give the name of the output .dbf file.","User
input: Insert filename",".dbf")
theExtensionCheck = theDBFName.Right (4)
if (theExtensionCheck <> ".dbf") then
   theFileName = MsgBox.Input("Please re-enter the name of the output .dbf
file. NOTICE: the filename should end with .dbf","Error: Re-insert
filename",".dbf")
end

theVTab = av.GetProject.FindDoc( theDBFName ).GetVTab
theVTab.SetEditable(TRUE)


' Checking if database is empty
```

```
theNumRecords = theVTab.GetNumRecords
if (theNumRecords > 0) then
   theEmptyIndicator = MsgBox.YesNo("The database you selected isn't empty. Do
you want to empty this database first (if yes: database will be emptied; if no:
TIN data will be added).","User input: Database isn't empty", TRUE)
   if (theEmptyIndicator = TRUE) then
      for each i in theVTab
         theVTab.RemoveRecord (i)
      end
   end
end


' Getting field information

theIDField = theVTab.FindField("Edge_id")
theREField = theVTab.FindField("R_edge")
theNVField = theVTab.FindField("N_vertex")
thePVField = theVTab.FindField("P_vertex")
theNAeField = theVTab.FindField("Na_edge")
theNAnField = theVTab.FindField("Na_neighbo")
theNCeField = theVTab.FindField("Nc_edge")
theNCnField = theVTab.FindField("Nc_neighbo")
thePAeField = theVTab.FindField("Pa_edge")
thePAnField = theVTab.FindField("Pa_neighbo")
thePCeField = theVTab.FindField("Pc_edge")
thePCnField = theVTab.FindField("Pc_neighbo")
theNFField = theVTab.FindField("N_face")
thePFField = theVTab.FindField("P_face")


' ===
' Edgewise filling of the table
' ===

for each theTriangleIndex in 0..(theNumTriangles-1)

   for each theEdgeIndex in 0..2

      theStringList = {}
      theTINEdge = TINEdge.Make(theTin,theTriangleIndex,theEdgeIndex)
      theEdgeIDString = theTINEdge.AsString
      theReverseEdge = theTINEdge.GetNeighboringEdge
      theREdgeString = theReverseEdge.AsString
      theNeighboringTriangle = theReverseEdge.GetTriangle

' The N Vertex and P Vertex defined as coordinates

      theLineZ = theTINEdge.AsLineZ
      the1Coord = theLineZ.ReturnStart
      the2Coord = theLineZ.ReturnEnd
      theNVertexString = the1Coord.AsString
      theNDigitsLength = theNVertexString.Count - 8
      theNDigits = theNVertexString.Right(theNDigitsLength)
      theNCoordinate = theNDigits.AsTokens (", ")
      theNVertex =
theNCoordinate.Get(0).AsString+"@"+theNCoordinate.Get(1).AsString+"@"+theNCoord
inate.Get(2).AsString
      thePVertexString = the2Coord.AsString
      thePDigitsLength = thePVertexString.Count - 8
      thePDigits = thePVertexString.Right(thePDigitsLength)
      thePCoordinate = thePDigits.AsTokens (", ")
      thePVertex =
thePCoordinate.Get(0).AsString+"@"+thePCoordinate.Get(1).AsString+"@"+thePCoord
inate.Get(2).AsString

' NA-Edge, NA-Neighbor

      theNextCCWEdge = theTINEdge.GetNextCCW
      theNAEdgeString = theNextCCWEdge.AsString
      theNANeighborEdge = theNExtCCWEdge.GetNeighboringEdge
      theNANeighborString = theNANeighborEdge.AsString
```

```
' NC-Edge, NC-Neighbor

      theNAEdgeIndex = theTINEdge.GetNextCCW.GetEdge
      theNeighborEdgeIndex = theTINEdge.GetNeighboringEdge.GetEdge
      theNCEdgeIndex = 3 - theNeighborEdgeIndex - theNAEdgeIndex
      theNCTINEdge = TINEdge.Make(theTin, theNeighboringTriangle,
theNCEdgeIndex)
      theNCEdgeString = theNCTINEdge.AsString
      theNCNeighborString = theNCTINEdge.GetNeighboringEdge.AsString

' PA-Edge, PA-Neighbor

      theNextAheadEdge = theTINEdge.GetNextAhead
      thePAEdgeString = theNextAheadEdge.AsString
      thePANeighborString = theNextAheadEdge.GetNeighboringEdge.AsString

' PC-Edge, PC-Neighbor

      thePCEdgeString = theTINEdge.GetNextBehind.AsString
      thePCNeighborString = theTINEdge.GetNextCW.AsString

' N-face and P-face

      theNFace = theNeighboringTriangle
      thePFace = theTriangleIndex

' Building edgeID's

      theStringList = {theEdgeIDString, theREdgeString, theNAEdgeString,
theNANeighborString, theNCEdgeString, theNCNeighborString, thePAEdgeString,
thePANeighborString, thePCEdgeString, thePCNeighborString}
      theNumberList = {}

      for each i in 0..9
         theEdgeString = theStringList.Get(i)
         theDigitsLength = theEdgeString.Count - 9
         theDigits = theEdgeString.Right(theDigitsLength)
         theTwoIndexes = theDigits.AsTokens (", ")
         if (theTwoIndexes.Get (0).AsString = "-1") then
            theIDString = "-1"
            theID = theIDString.AsNumber
            theNumberList.Add(theID)
         else
            theIDString =
theTwoIndexes.Get(0).AsString+theTwoIndexes.Get(1).AsString
            theID = theIDString.AsNumber
            theNumberList.Add(theID)
         end
      end

' Write to table

       theNewRec = theVTab.AddRecord
       theVTab.SetValue (theIDField, theNewRec, theNumberList.Get(0))
       theVTab.SetValue (theREField, theNewRec, theNumberList.Get(1))
       theVTab.SetValue (theNVField, theNewRec, theNVertex)
       theVTab.SetValue (thePVField, theNewRec, thePVertex)
       theVTab.SetValue (theNAeField, theNewRec, theNumberList.Get(2))
       theVTab.SetValue (theNAnField, theNewRec, theNumberList.Get(3))
       theVTab.SetValue (theNCeField, theNewRec, theNumberList.Get(4))
       theVTab.SetValue (theNCnField, theNewRec, theNumberList.Get(5))
       theVTab.SetValue (thePAeField, theNewRec, theNumberList.Get(6))
       theVTab.SetValue (thePAnField, theNewRec, theNumberList.Get(7))
       theVTab.SetValue (thePCeField, theNewRec, theNumberList.Get(8))
       theVTab.SetValue (thePCnField, theNewRec, theNumberList.Get(9))
       theVTab.SetValue (theNFField, theNewRec, theNFace)
       theVTab.SetValue (thePFField, theNewRec, thePface)

   end

' Showing progress of calculations

    progress = (theTriangleIndex / (theNumTriangles-1)) * 100
```

```
    proceed = av.SetStatus(Progress)
    if (proceed.Not) then
      av.ClearStatus
      av.ShowMsg("Stopped")
      exit
    end

end

theVTab.SetEditable(FALSE)
MsgBox.Info("Computations completed. Results are stored in "+theDBFName+"
.","Finished")
```

Bijlage III: inlezen data in Oracle m.b.v. SQL*Loader

III.1 model.sql

```
set echo on;

drop table tin;
create table tin(
    ID integer,
    Re integer,
    nvertexX number,
    nvertexY number,
    nvertexZ number,
    pvertexX number,
    pvertexY number,
    pvertexZ number,
    NAe integer,
    NAn integer,
    NCe integer,
    NCn integer,
    PAe integer,
    PAn integer,
    PCe integer,
    PCn integer,
    NF integer,
    PF integer
);
exit;
```

III.2 tin.ctl

```
load data
infile 'tin.data' "str"
append
into table tin
fields terminated by ','
trailing nullcols (
         ID integer external(10),
         Re integer external(10),
         nvertexX float external terminated by '@',
         nvertexY float external terminated by '@',
         nvertexZ float external,
         pvertexX float external terminated by '@',
         pvertexY float external terminated by '@',
         pvertexZ float external,
         NAe integer external(10),
         NAn integer external(10),
         NCe integer external(10),
         NCn integer external(10),
         PAe integer external(10),
         PAn integer external(10),
         PCe integer external(10),
         PCn integer external(10),
         NF integer external(10),
         PF integer external(10)
         )
```

III.3 doit.sh

```
#!/bin/sh

set -v

sqlplus <user-name> / <password> @model.sql
sqlldr  <user-name> / <password> CONTROL=tin.ctl
```