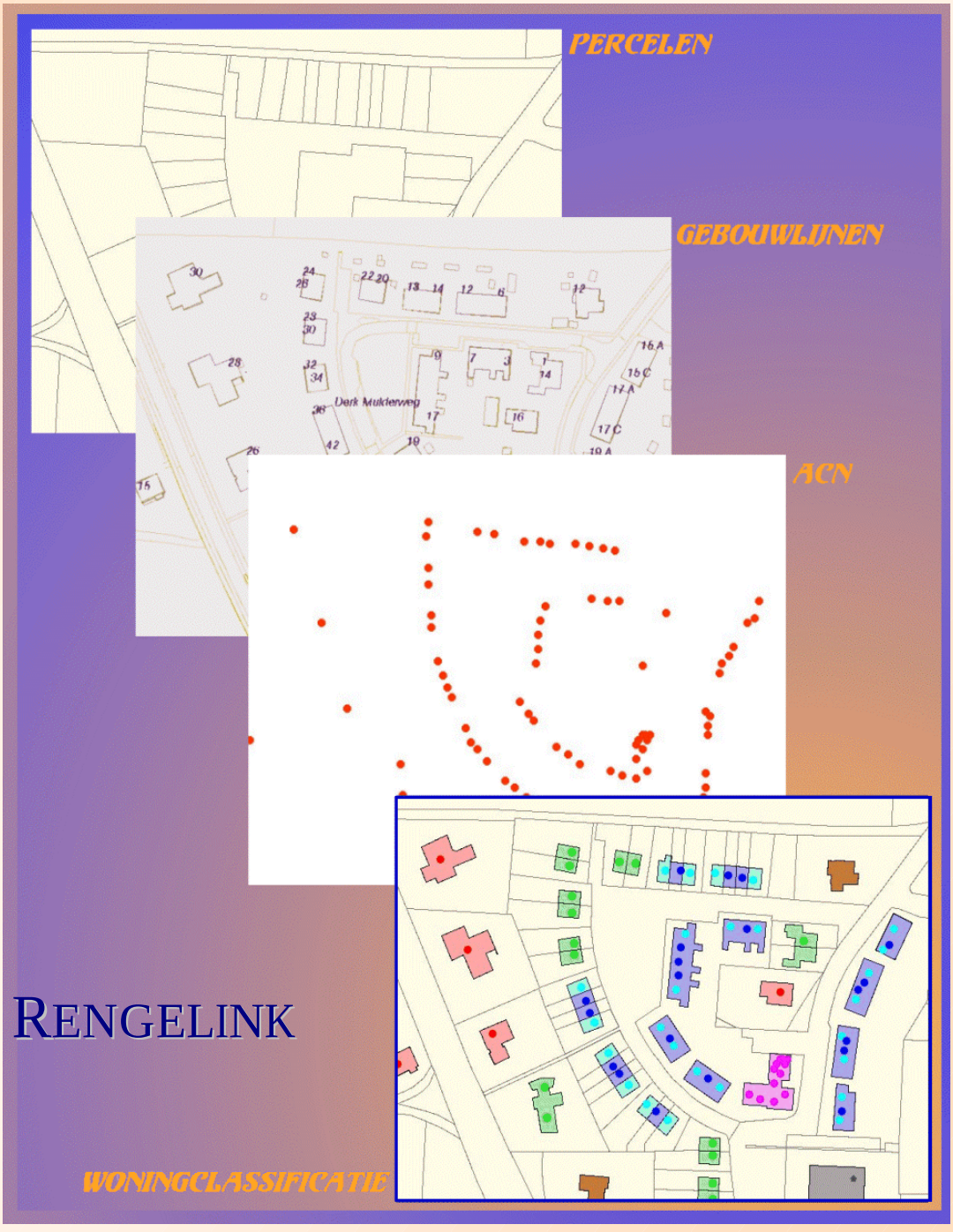
AUTOMATISCH AFLEIDEN EN CLASSIFICEREN VAN WONINGEN UIT KADASTRALE GEGEVENS



M.A. RENGELINK

AUTOMATISCH AFLEIDEN EN CLASSIFICEREN VAN WONINGEN UIT KADASTRALE GEGEVENS

MAUREEN RENGELINK

TECHNISCHE UNIVERSITEIT DELFT
FACULTEIT CiTG
AFDELING GEODESIE
SECTIE GIS-TECHNOLOGIE

Afstudeerhoogleraar: Prof. Dr. Ir Peter van Oosterom
Afstudeerbegeleiders: Ir. Edward Verbree & Drs. Wilko Quak

TU Delft

geodesie

GIS technology

VOORWOORD

Dit afstudeerverslag is het resultaat van mijn afstudeeronderzoek bij de sectie GIS-technologie van Prof. Dr. Ir. Peter van Oosterom. Het afstudeeronderzoek, waarmee ik in het begin van 2000 mee ben begonnen, richt zich op het definiëren en uitvoeren van een model om woningen te classificeren uit kadastrale gegevens.

Verondersteld wordt dat de lezer van dit verslag iets af weet van wat een database is en in het bijzonder een database met geografische componenten.

Hierbij wil ik mijn afstudeerbegeleiders Ir. Erward Verbree en Drs. Wilko Quak bedanken voor de begeleiding tijdens het afstudeeronderzoek. Ook Prof. Dr. Ir. Peter van Oosterom wil ik bedanken voor zijn begeleiding. Hij was ook degene die mij dit onderwerp voorstelde.

Verder wil ik het Kadaster bedanken in de vorm van de volgende personen, Harry Uitermark, Caroline Groot, Bart Maessen en Ted Schut. Zij waren tegelijkertijd met mij bezig aan een project bij het Kadaster dat zo ongeveer hetzelfde doel had als ik voor mijn afstuderen. Zij hebben mij de testdata gegeven en ik heb via hun veel over het onderwerp geleerd.

Delft, december 2000
Maureen Rengelink

INHOUDSOPGAVE

SAMENVATTING

INTRODUCTIE

In dit afstudeeronderzoek is er een methode gedefinieerd die woningen en adressen classificeert. Er is een groeiende behoefte aan een kaart product, waarop voor elke woning een specifiek type is toegekend. Bijvoorbeeld vrijstaande huizen, appartementgebouwen, enz. Een van de doelen van zo een product is het gebruik ervan bij de analyse van trends in huisprijzen in een bepaald gebied. Deze zijn afhankelijk van het woningtype. Met een woning wordt bedoeld een stukje gebouw dat bedoeld is om een gezin in te huisvesten. Op dit moment is er in Nederland nog niet een digitaal product (kaart) waarbij de woningen als een vlakobject gemodelleerd zijn. Er is wel een spaghetti kaart met de grenzen van de gebouwen. Deze grenzen kunnen een hele rij huizen samen zijn, en deze moeten dus nog in individuele stukken verdeeld worden. De gezochte informatie zal worden afgeleid van ruimtelijke en thematische informatie die aanwezig is bij het Nederlandse Kadaster. Dit gebeurd in drie stappen: Eerst wordt er een vlakken laag gemaakt van de grenzen van de gebouwen, zodat elk gebouw een polygoon vormt. De volgende stap is een geometrische versnijding met attribuut overerving (vanaf nu wordt dit kortweg versnijding genoemd) tussen de gebouwenlaag en een percelenlaag.. Hieruit komt een kaartlaag met deelgebouwen. Ook wordt in deze stap voor elk adres bepaald bij welk deelgebouw het hoort. In de derde stap wordt de informatie uit de versnijding samen met andere informatie uit de database gebruikt om automatisch alle deelgebouwen en adressen te classificeren met behulp van beslisregels. De methode, zoals in deze scriptie gedefinieerd, is uitgevoerd op een testset. Deze testset bestaat uit de gegevens over de gemeente Apeldoorn.
VERSCHILLENDE WONINGTYPEN
Gedurende het proces moet er bepaald worden tot welk van de volgende klasse een adres behoort:
* MIDDEN VAN EEN RIJ, dit type krijgt de code "M"
* EIND VAN EEN RIJ, dit type krijgt de code "E"
* VRIJSTAANDE WONING, dit type de code "V"
* TWEE-ONDER-ÉÉN-KAP, dit type krijgt de code "T"
* APPARTEMENT, dit type krijgt de code "A"

Als aanvulling op deze classificatie krijgen alle woningen ook een waardering toegekend. Deze waardering is een indicatie voor het type beslisregel dat is toegepast bij de classificatie van de adressen. Het classificeren van gehuurde woningen zal anders gaan dan de classificatie van koopwoningen. Dit komt omdat het Kadaster in Nederland alleen de eigenaar van kadastrale percelen bijhoud en niet de huur van woningen. Het kan zo zijn dat er op een perceel, dat van een organisatie is, één (deel)gebouw staat wat bestaat uit een rij woningen die verhuurd zijn.

GEBRUIKTE DATASETS

De verschillende datasets die gebruikt zijn, worden allemaal bijgehouden door het Kadaster. De volgende dat kan gebruikt worden voor de afleiding en classificatie van woningen:
* LKI staat voor "Landmeetkundig en Kartografisch Informatiesysteem" en bevat alle ruimtelijke gegevens. Het bevat de locaties en de vorm van alle kadastrale percelen en andere informatie om een kadastrale kaart te maken. De gebouwen worden opgeslagen als lijnen en er zijn tekstlabels voor huisnummers en straatnamen.
* AKR staat voor "Administratieve Kadastrale Registratie" en bevat alle administratieve gegevens. Voor elk perceel in LKI word er veel thematische en wettelijke attributen opgeslagen in AKR. Binnen AKR is expliciet opgeslagen of een perceel een appartementencomplex bevat. De cultuurcode, die voor elk perceel wordt opgeslagen, is een indicatie van het hoofdgebruik van een perceel.

* ACN staat voor "Adres Coördinaten Nederland). Het is een database met geografische coördinaten voor elk adres in Nederland dat erkend wordt door de PTT. Deze informatie is erg belangrijk bij het classificeren van gehuurde woningen.

HET PROCES

Twee belangrijke stappen worden er uitgevoerd : De afleiding van de deelgebouwen en de classificatie van zowel de deelgebouwen als de adressen. Beide zullen nu kort besproken worden.

AFLEIDING VAN DE DEELGEBOUWEN

Nadat de gebouwen en de percelen uit de database zijn gehaald moet er een versnijding tussen beide berekend worden. Hiervoor is gebruik gemaakt van de "Computational Geometry Algorithms Librairy" (CGAL). Om gebruik te kunnen maken van deze C++ bibliotheek worden de gegevens uit de database gehaald en ingelezen in een geschreven programma. Voordat de versnijding werkelijk uitgevoerd kan worden moeten de gebouw worden voorbewerkt. Het resultaat van de versnijding wordt weer ingelezen in de database en gebruikt in de classificatie. De volgende stappen worden uitgevoerd door het geschreven versnijdingsprogramma:
1. Maak een kaartlaag met de gebouwlijnen
2. Sluit alle losse lijnstukken, waar het kan, aan andere lijnstukken. De grens van een gebouw moet uiteindelijk rondgelopen kunnen worden.
3. Alle gesloten gebouwen krijgen een unieke identificatie.
4. De percelen worden ingelezen in een kaartlaag. De percelen laag is al topologisch correct.
5. De versnijding tussen de twee kaartlagen berekenen. Het resultaat is een kaartlaag met deelgebouwen, die ontstaan zijn doordat een gebouw opgedeeld wordt door meerdere percelengrenzen.
6. Voor elk adrescoördinaat wordt bepaald in welk deel van de nieuwe kaartlaag het hoort.

In de database zijn de volgende tabbellen aan gemaakt: deelgebouw, gebouw, perceel, huisnummer en ACN. De resultaten van de versnijding worden ingelezen in deze tabellen. De rest van de attributen wordt gevuld met gegevens uit de originele database en er worden een aantal attributen berekend. Bijvoorbeeld : het aantal deelgebouwen waaruit een gebouw bestaat.

Tabel	Kolomen
Deelgebouwen	Identificatie, perceelidentificatie, gebouwidentificatie, buren, oppervlakte, n_acn, n_huisnummers, indicatie_wonen, woningtype, waardering
Gebouwen	identificatie, oppervlakte, n_deelgebouwen, n_acn, n_huisnummers, n_percelen.
Percelen	identificatie (G_akrobjectnummer), oppervlakte, n_gebouwen, n_deelgebouwen, Cultuurcode
Huisnummers	identificatie, deelgebouwidentificatie, object_tekst
ACN	identificatie, postcode, huisnummer, huisnummertoevoeging, aantal, deelgebouwidentificatie, perceelidentificatie, woningtype, waardering

Overzicht van, voor de classificatie uiteindelijk gebruikte, tabellen en attributen (primaire sleutels, externe sleutels)

CLASSIFICATIE VAN DE DEELGEBOUWEN EN DE ADRESSEN

In het laatste deel van het proces wordt de hierboven besproken tabellen gebruikt in de classificatie van de deelgebouwen en de adressen. Om dit te doen zijn er verschillende regels voor de verschillende woningtype gedefinieerd en omgezet naar Structured Query Language (SQL). Voor de classificatie zijn uiteindelijk geometrische en administratieve attributen gebruikt. Voorbeeld van een beslisregel: Een deelgebouw dat één buur heeft, en onderdeel is van een gebouw dat in tweeën gedeeld is en waar maar twee adressen aanhangen, is een twee-onder-één-kap.
De unieke gebouwidentificatie wordt in dit geval gebruikt om de tabel deelgebouw aan die van gebouw te koppelen. Door deze beslisregels toe te passen op de database met behulp van query's worden de deelgebouwen geclassificeerd.

Twee kolomen, "Woningtype" en " waardering" worden gevuld gedurende dit laatste deel van het proces. Het woningtype zal een van de 5 woningtype krijgen die vermeld zijn onder het kopje verschillende woningen. Het is ook mogelijk dat het woningtype een van de volgeden codes krijgt toegekend: '#', niet bedoeld voor wonen,'*' adres niet aan een deelgebouw te koppelen, '-' niet geclassificeerd. De classificatie wordt in de volgende drie stappen uitgevoerd:
1. Alle deelgebouwen krijgen een classificatie
2. Voortplanting van de deelgebouwclassificaties naar erbij behorende ACN
3. Alle nog niet geclassificeerde ACN krijgen een classificatie met behulp van nog een aantal beslisregels.

KWALITEIT VAN HET RESULTAAT

Omdat het proces automatisch wordt uitgevoerd is er een indicatie van de kwaliteit van het resultaat nodig. Dit gebeurd op twee manieren. Eerst wordt het resultaat van het proces visueel geïnspecteerd. Zodra er systematische fouten optraden, is het algoritme aangepast. Verder is er uit de testset een selectie van ruim 1000 adressen geselecteerd, deze zijn stuk voor stuk bekeken en gecontroleerd. Ongeveer 10% van deze validatieset was fout geclassificeerd. Slechts één vijfde hiervan was het gevolg van fouten in de classificatie. De rest is het gevolg van fouten in de bronbestanden.

CONCLUSIE

Meer dan 96% van alle adressen in de testset hebben uiteindelijk een classificatie gekregen. Het is dus mogelijk om automatisch uit kadastrale gegevens het woningtype af te leiden en te classificeren voor elk adres.

SUMMARY

INTRODUCTION

In this final-thesis a method is defined that classifies houses and addresses on the Cadastral map of the Netherlands. There is a growing need for a map product, on which every house has been assigned a specific type, such as 'apartment building', 'free standing house', and so on. One of the purposes of this product is the analysis of trends in house prices in the country, which depend on the house type. Houses meaning in this paper single units used for one family to live in. Currently, there is no digital map of the Netherlands, on which houses are modelled as objects. However, a spaghetti map of the boundaries of buildings is available. This can be a large building (e.g. a row of houses), which must first be subdivided into individual units. The requested information will be derived from spatial and thematic data at the Dutch Cadastre, in three steps: First we make polygon layer of the boundaries of buildings, so that each building appears as a separate polygon. Second a map overlay of the buildings and parcels is performed to generate an integrated map of parcels and buildings. After the overlay has been performed for each address is determined to which part of the overlayresult it belongs. In the third step, the information from the map overlay was used to perform the automatic classification based on decision-rules. The method defined in this project was performed on a test set, which contains the municipal Apeldoorn.

The goal of this project is to find out whether it is possible to automatically derive and classify the housetype that belongs to an address, and whether the quality of the automatic classification is good enough.

DIFFERENT HOUSE CLASSES

During the process we wish to assign one of the following classes to each address:
* MIDDLE HOUSE IN A ROW, this type will be given the code "M"
* END HOUSE IN A ROW, this type will be given the code "E".
* FREE-STANDING HOUSE, this type will be given the code "V".
* TWO-UNDER-A-ROOF, this type will be given the code "T".
* APARTMENT, this type will be given the code "A".

In addition to this classification, all houses are also assigned a value. This value is an indication of which classification rule is used to classify a address.
The approach for classifying the rented houses will be different from the approach for identifying the privately owned houses. This is because the Cadastre in the Netherlands only registers ownership of cadastral parcels and the rental of living units. For instance, there will be one building (representing a row of houses or living units) on one large parcel when one organisation owns them.

DATASETS USED

The different datasets used for the classification are maintained by the Cadastre. From these databases the following data is available for the classification (and detection) of the houses:
* LKI stands for "Landmeetkundig Kartografische Informatie" and contains all geometric information. It contains the shapes and location of all cadastral parcels and other information to make a cadastral map. The buildings are stored as line elements and text labels are used for house numbers and street names on the Cadastral map.
* AKR stands in Dutch for "Administrative Cadastre Registration", and contains all administrative information. For every parcel in LKI there are many thematic or legal attributes in AKR. It is explicitly marked if a parcel is part of an apartment complex. The Culture codes is an indication of the main use of the parcel.

* The ACN (Adress Coordinates Netherlands) is a database with the geographic coordinates of all the addresses in the Netherlands recognised by the PTT (Royal Dutch Mail). This information is very important to classify rented houses.

THE PROCES

Two main steps are performed: the derivation of house covered part of parcels and the classification of both the CPP's and the addresses. Both will be discussed briefly in the next two parts.

DERIVATION OF HOUSE COVERED PARTS OF PARCELS (CPP)

After extracting all buildings boundaries and parcel from the database, the second step is to perform a map overlay. For this step the Computational Geometry library CGAL is used. In order to use the CGAL C++ library we extract the layer data from the database and read it into the overlay software. Before the overlay is performed some pre-processing should be done on the data. The resulting overlay, together with the administrative data will be copied to a geo relational database. The following steps are performed by the written overlay-program:
1. Make a layer from the building lines
2. Snapping dangling edges. The boundary of a building should be a topologically closed loop. If we have a boundary which is not part of a loop (a dangling edge), one can try to snap such an endpoint to a nearby boundary to form a closed loop.
3. Now all buildings are given a unique number.
4. The parcels already form a topologically correct maplayer, so no preprocessing is needed.
5. Now an overlay between the two layers is performed. This results in a maplayer where buildings are split up when they are standing on different parcels. This is the maplayer with the CPP's.
6. For each address co-ordinate is determined to which CPP it belongs.

In the database the tables CPP (house Covered Part of Parcel), buildings, parcels, house numbers and ACN have been created. There attributes come from the original database, and from the result of the overlay. Additional attributes are calculated. For example: the number of CPP that a building is divided in.

	Kolomen
CPP	Identification, parcelidentification, buildingidentification, neigbours, area, n_acn, n_housenumbers, culturecode_living, type, value
Building	Identification, area, n_cpp, n_acn, n_housenumbers, n_parcel.
Parcels	Identification, area, n_buildings, n_cpp, Culturecode
Housenumbers	Identification, cpp identification, object_text
ACN	Identification, postalares, housenumber, extension, number, cpp identification, parcelidentification, type, value

Table with an overview of the tables and their attributes. These tables are used in the classification of the CPP and the Addresses.

CLASSIFICATION OF THE CPPS AND THE ADRESSES

In the final phase of the process we are going to use the above-described database to classify all the houses. Writing rules for the different house types and expressing these, as SQL queries on the database will do this. For the classification, administrative and geometric aspects of the buildings are used. An example of such a rule might be: A CPP with one neighbour and a building divided into two CPP, is a 'two-under-a-roof'.. The unique building-identifier will be used to perform the join between "CPP" and "Building". By applying these rules to the database, the houses will be classified The culture-code is used to select only the parcels where the main use is living.

The two columns "type" and "value" will be filled during this part of the process. Housetype will contain the assignment to one of the 5 classes given earlier or one of the following classes: '#' building not for living, '*' no building connected to the address and '-' not yet classified. The classification is performed in three steps:
1. All the CPP will get a classification.
2. These CPP-classifications are propagated to ACN.
3. The ACN that have not jet got a classification are classified.

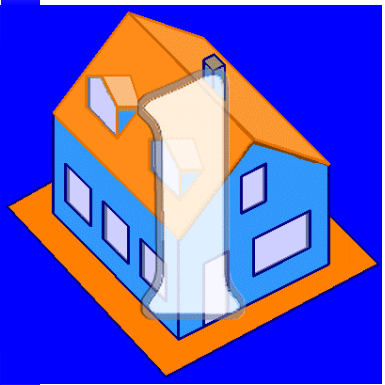
QUALITY OF THE RESULTS

Because the process will be completely automated, an indication of the quality of the results will needed. This is done in two ways. First, the results of the process will be inspected visually. If there appear to be systematic errors, the classification algorithm was adapted. Second a validationset of about 1000 addresses was taken from our testset, and these have been manually tested. About 10% got a wrong classification. But only one-fifth of these were caused by faults in the classification-rules. The rest was caused by errors in the original data.

CONCLUSION

Over 96% of the addresses in our testset were given a classification. So it is possible to automatically derive and classify, from kadastral data, the housetype for each address.

INLEIDING



Nederland wil een vastgoedindex. Een groot deel van alle objecten in de vastgoed zijn gebouwen. Een goede gebouwenclassificatie is een van de stappen in de richting van een goede vastgoedindex. Dit afstudeeronderzoek richt zich op het definiëren van een conceptueel model voor de woningclassificatie in Nederland. Het classificatieproces moet al aanwezige gegevensbestanden gebruiken. De classificatie van woningen houdt in dat voor elk adres bepaald wordt welk type woning erbij hoort. Tot nu toe is nog nergens expliciet opgeslagen welke type woningen er op een perceel staan, uitgezonderd appartementen in het AKR.

1.1 ACHTERGROND

Figuur1-1 Indeling woningen, gebouwen en vastgoed

Het grootste deel van alle vastgoedobjecten in Nederland zijn gebouwen. Het classificeren van gebouwen is een van de stappen in de richting van een goede vastgoedindex. Onder een vastgoedindex wordt in deze afstudeerscriptie verstaan een gestructureerde indeling in klassen van alle vastgoed objecten. Deze indeling kan verwezenlijkt worden door bij elk object op te slaan tot welke vastgoedklasse het object hoort.
“Een gebouw is hetgeen gebouwd is, inz. huis, kantoor, fabriek” [W93]. Een eerste scheiding binnen gebouwen is de scheiding naar woningen en niet-woningen. Een groot deel van de gebouwen in Nederland bestaat uit woningen.
Hoe wordt een woningclassificatie verkregen en waarom zou dat via een automatische methode moeten? In Nederland zijn meer dan 6,3 miljoen woningen (1998) [CBS2000]. Gezien dit grote aantal woningen zou het mooi zijn om met een automatische methode het grootste deel hiervan te berekenen en eventueel vervolgens handmatig worden verbeterd en aangevuld. In principe zou via de automatische classificatie een overzicht van de gebouwen hooguit een aantal keer per jaar berekend hoeven worden. Omdat in principe alle gegevens al ergens opgeslagen zijn is een automatische methode die op een gekozen moment uitgevoerd kan worden een goede oplossing.
Er zijn op de markt reeds enkele bestaande producten verkrijgbaar. Dit zijn echter producten met een zeer lage betrouwbaarheid en niet erg praktisch bruikbaar. Vaak zijn ze gebaseerd op enquêtes van een kleine groep.

1.2 PROBLEEMOMSCHRIJVING

Titel :
*Automatisch **afleiden** en **classificeren** van woningen uit kadastrale gegevens*

Het probleem is nu om voor het Kadaster een proces te beschrijven en te implementeren, zodat het woningendeel van de vastgoedindex uit al aanwezige gegevensbestanden gehaald kan worden. Voor het goed ontwerpen van een woningclassificatie is het van belang wat voor type woning er op een perceel staat. Tot nu toe wordt nog nergens expliciet opgeslagen welk type woningen er op een perceel staan, uitgezonderd appartementen in de Administratieve Kadastrale Registratie (AKR).
Een eigenschap van deze woningbeschrijving is de koppeling / integratie van administratieve en geometrische gegevens. Een voordeel kan zijn dat het woningtype afgeleid kan worden uit bestaande informatie en dat er dus niet opnieuw gegevens hoeven worden ingewonnen. Omdat er ook gebruik gemaakt wordt van geometrische gegevens is het mogelijk om een visuele presentatie (kaarten) te maken van dit product.
Een goede en eenduidige beschrijving tussen de bestaande gegevensbestanden en nieuw af te leiden woningclassificatie is nodig.

Dit nieuwe product kan nuttig zijn voor vastgoedbeleggers en het centraal bureau voor de statistiek (CBS). Een voorbeeld waarbij dit product gebruikt zou kunnen worden is de analyse van trends in de huizenprijzen.
Daarnaast zou dit product gebruikt kunnen worden als gebouwen bestand bij “Virtual Reality” (VR) en 3D visualisatie. In de 3D wereld kunnen standaard modellen voor elk woningtypen gebruikt worden voor de visualisatie. Dit kan tot een overtuigendere virtuele wereld leiden.

1.3 HOOFDVRAAG EN DOELSTELLINGEN

Dit afstudeeronderzoek richt zich op het definiëren van een conceptueel model voor de woningclassificatie in Nederland. De hoofdvraag van deze scriptie luidt als volgt:

Is het mogelijk om een methode te ontwikkelen die automatisch op basis van digitaal aanwezige gegevens woningen afleidt en als een bepaald type classificeert ?

Deelvragen naar aanleiding van het doel en de randvoorwaarden:

- Is het mogelijk om per adres het **woningtype** automatisch af te leiden en te classificeren met behulp van de **kadastrale kaart** en **administratieve gegevens**?
- Kan deze afleiding met behulp van de geometrische basisbibliotheek “Computational Geometry Algorithms Library” (**CGAL**) tot stand komen?
- Wat zijn de redeneerregels waarmee de schatting van een woningtype plaatsvindt?
- Voor die gevallen dat een schatting van het woningtype mogelijk is, hoe goed is die schatting dan?

Dit afstuderen is gebaseerd op een vraag gesteld door Kadata aan de afdelingen Informatie Beleid & Projecten / Beleid Onderzoek Advies (IBP/BOA) en VastgoedInformatie & Geodesie / Geodetisch advies bureau (VIG/GAB) van het Kadaster. Kadata ziet in de Amsterdam Exchanges (AEX) organisatie een kandidaat-klant voor dit product. Tegelijkertijd met mijn afstuderen is er een project bij het Kadaster gaan lopen om deze vraag te beantwoorden.
Omdat dit een afstudeerproject is, is ervoor gekozen om meer nadruk op de wetenschappelijke achtergronden en leeraspecten te leggen. Dit uit zich vooral in het feit dat niet een bestaand GIS product is gebruikt om de geometrische versnijding (engels: overlay) in uit te voeren, maar om CGAL te gebruiken. Vanaf nu zal de geometrische versnijding van kaartlagen, met attribuut overerving kortweg met versnijding worden aangeduid. In CGAL zijn hulpmiddelen aanwezig waardoor het programmeren van een versnijding vergemakkelijkt wordt, maar waarbij het leren over hoe een versnijding in elkaar zit toch naar voren komt.

1.4 RANDVOORWAARDEN

- Er moet geprobeerd worden om alleen gegevens te gebruiken die eigendom zijn van het Kadaster. Bij het Kadaster zijn zowel “Grootschalige Basis Kaart Nederland”[1] gebouwen opgeslagen als gebouwen uit de kadastrale basiskaart, welke door het Kadaster zelf zijn ingewonnen. Het is de bedoeling om alleen deze laatste te gebruiken.
- Het is de bedoeling dat er uiteindelijk een methode komt te liggen die op ieder gekozen moment uitgevoerd kan worden, en dat er niet nieuwe gegevens ingewonnen en opgeslagen zullen worden.
- Het is de bedoeling om zowel koop als huurwoningen te classificeren.
- Er zijn met de opdrachtgever afspraken gemaakt over welke klassen er gebruikt zullen gaan worden. Dit zijn er vijf geworden:
    - Appartement
    - Vrijstaand
    - Twee-onder-één-kap
    - Eindwoning
    - Middenwoning

Dit zijn ongeveer dezelfde uitgangspunten, eisen en randvoorwaarden als dat gelden voor het Kadaster project.

1.5 GEBRUIKTE HARD- & SOFTWARE

Voor de implementatie en uitvoering van de verschillende onderdelen van het classificatieproces is gebruik gemaakt van zowel de centrale computer van de sectie als van een lokale computer.
De centrale computer van de sectie GIS-technologie wordt “Casagrande" genoemd en deze bestaat uit: SUN E3500, 2Gb RAM, 0,6 Tb disk en een Solaris Unix besturingssysteem.
Voordat deze computer draaide is mijn eigen werkstation gebruikt om te implementeren. Mijn werkstation bestaat uit: i586, 128k RAM, 10 Gb disk en SuSE Linux 6.3

De gebruikte database, waarin LKI en AKR gegevens worden opgeslagen, is een INGRES 2.0 database die op de centrale computer van de sectie draait.

Programmeertaal die gebruikt is voor het berekenen van de geometrische versnijding is C++ en CGAL 2.1. Voor het compileren van de C-files is gebruik gemaakt van de “g++ egcs 2.91.66” en bijbehorende makefiles. Makefiles bevatten de afhankelijkheden die

[1] De Grootschalige Basiskaart Nederland (GBKN) is geen product van het kadaster zelf, maar van de Topografische Dienst. Het kadaster mag deze gegevens niet vrij gebruiken.

nodig zijn om programma's uit te voeren. Verder is er ook gebruik gemaakt van: Structured query language (SQL), een teksteditor, en shellscripts, Dit laatste zijn een soort "batch" programma's die in dit onderzoek steeds beginnen met de regel "/bin/sh".

Deze scriptie is in Word 97 geschreven. Verder is gebruik gemaakt van Paint Shop Pro 5.1 om de foto's en de snapshots van de programma's op het scherm bij te werken. Deze twee programma's draaien onder Windows 98.

1.6 STRUCTUURBESCHRIJVING

Deze scriptie is opgebouwd uit twee delen; het eerste deel bevat de achtergronden bij het afstudeeronderzoek en het tweede deel bevat de gehele praktische kant van het onderzoek, de implementatie van een proces dat de woningclassificatie uitvoert. In het volgende hoofdstuk worden kort een aantal begrippen uitgelegd die gebruikt zijn bij dit afstudeerproject. In hoofdstuk drie wordt het Kadaster en de kadastrale gegevens die gebruikt zijn toegelicht. In het vierde hoofdstuk wordt de GIS -operatie geometrische versnijding (engels: overlay) verder toegelicht. In het laatste hoofdstuk vijf, van het eerste deel, wordt het begrip woning en de gekozen woningtypen onder de loep genomen.

Dan begint het tweede deel met een overzicht van het gehele proces en een beschrijving van het totale programma (Hoofdstuk 6). In hoofdstuk zeven wordt de versnijding zoals die in C++ en CGAL geprogrammeerd is besproken. Dit is een belangrijk deel van de afleiding. Hoofdstuk acht bevat informatie over alles wat er in de database met behulp van SQL wordt veranderd. Dit hoofdstuk bevat de classificatieregels en de resultaten van de classificatie. In hoofdstuk negen worden kaarten met de resultaten van delen van het testgebied getoond. In dit hoofdstuk zijn een aantal foto's toegevoegd van gebouwen in de kaarten zodat de lezer inzicht kan krijgen van wat er nu allemaal met de kaart gevisualiseerd wordt.
In hoofdstuk tien wordt ingegaan op de validatie van de resultaten. De fouten in de kaarten kunnen onderverdeeld worden naar fouten die het gevolg zijn van de bronbestanden en fouten van de classificatieregels uit hoofdstuk acht. En daaruit volgen conclusies en aanbevelingen welke beschreven zijn in het elfde en laatste hoofdstuk.

BEGRIPPEN

Voor het lezen van deze scriptie is het nodig om een aantal begrippen te definiëren of te omschrijven. In dit hoofdstuk zullen begrippen als een kaart, de gebruikte C++ bibliotheek CGAL, Structured Query Language (SQL) en daarbij behorende zaken worden besproken.

Dit hoofdstuk bevat vier paragrafen: een geometrie deel en een database deel. In het derde stuk worden deze twee gekoppeld in een GIS. Als laatste worden de gebruikte GIS-software componenten kort beschreven.

Kaart

Computational Geometry Algorithms Library (CGAL)

Structured Query Language (SQL)

GIS

Gebruikte GIS software

2.1 BEGRIPPEN MET EEN GEOMETRISCH COMPONENT

In dit deel komen de basisbegrippen die een geometrisch element hebben aan de orde.

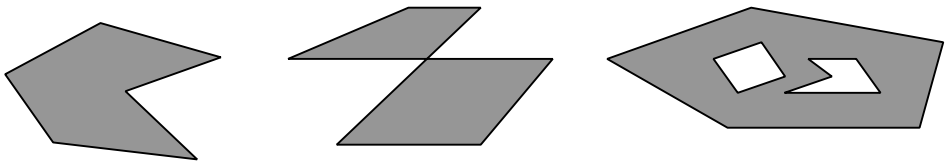
2.1.1 KAART

De kaarten die in dit project gebruikt worden zijn in principe kaarten die bestaan uit verschillende kaartlagen. Kaartlagen kunnen een punt-, lijn- of vlakthema bevatten. Een voorbeeld van een puntthema is een laag met straatmeubilair, of bomen die vastgeprikt zijn op een punt. Een voorbeeld van een lijnthema is een wegenkaart. Een voorbeeld van een vlakthema is bodemgebruik. De belangrijkste kaartlagen voor dit afstudeerproject zijn de kaartlagen met vlakken. Deze vlakken bestaan in principe uit eenvoudige polygonen.

2.1.2 POLYGONEN

Een eerste onderscheid dat we bij polygonen kunnen maken is de eenvoud. Een polygoon is eenvoudig als er niet meer dan twee zijden van één en dezelfde polygoon op een knooppunt uitkomen. Een vlakthema is geldig wanneer de kaartlaag eenvoudige polygonen met of zonder gaten bevat. Deze gaten zijn ook weer in de vorm van eenvoudige polygonen.

Figuur 2-1 Polygonen



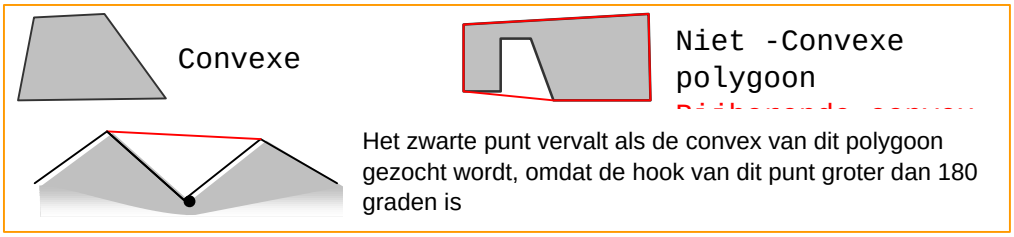
eenvoudig polygoon

niet eenvoudig polygoon (het middelste punt heeft 4 uitgaande lijnen)

eenvoudig polygoon met gaten

Eenvoudige polygonen kunnen weer onderverdeeld worden in convexe en niet convexe polygonen. Een polygoon is convex als een rechte lijn tussen twee punten van dat polygoon altijd volledig binnen het polygoon valt. De Convex is dus ook de omvattende polygoon waarvan de lengte van de zijden opgeteld het kortste is. Een polygoon is convex als ze dezelfde vorm heeft als de bijbehorende convex.

Figuur 2-2 Convexe polygonen



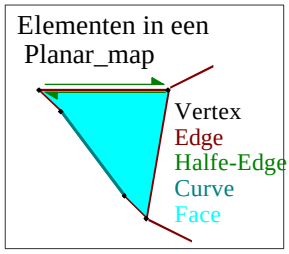
Het zwarte punt vervalt als de convex van dit polygoon gezocht wordt, omdat de hook van dit punt groter dan 180 graden is

2.1.3 COMPUTATIONAL GEOMETRY ALGORITHMS LIBRARY (CGAL)

CGAL staat voor Computational Geometry Algorithms Library.
CGAL is een C++ klasse bibliotheek met geometrische basisobjecten en operaties. Met behulp van deze bibliotheek moet het mogelijk zijn om efficiënt specifieke GIS-functies te programmeren en uit te laten voeren (applicaties bouwen). Van belang is dat deel van de bibliotheek waarin de topologische structuur wordt gedefinieerd en de daarbij behorende versnijdingsfunctie die hiervan gebruik maakt.
CGAL bevat een topologische kaartlaagstructuur (topological_map) en een ruimtelijke kaartlaagstructuur (planar_map). Hier volgt nog een verdere beschrijving van in paragraaf 2.1.5. Eerst wordt de opslagstructuur die voor deze kaartlagen gebruikt wordt besproken.

2.1.4 DOUBLY CONNECTED EDGE LIST (DCEL)

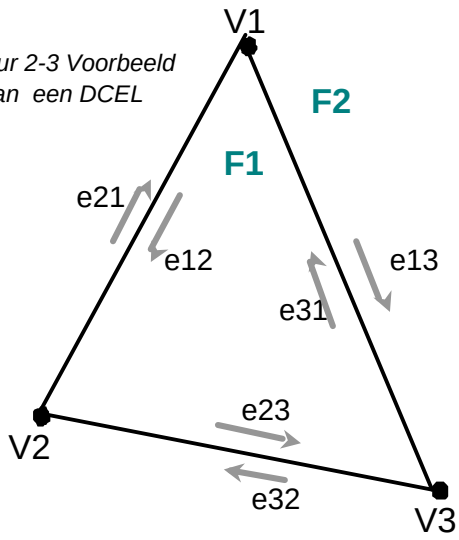
Figuur 2-2 Elementen van een Planar map



De Doubly Connected Edge List is een opslagstructuur om de topologische relaties tussen elementen in een kaartlaag in op te slaan. In principe bestaat een DCEL uit drie collecties met records: 1. punt (vertex), 2. gericht lijnstuk (half-edge) en 3. vlak (face). Vanaf nu zullen de termen face, half-edge en vertex gebruikt worden. Een vertex is een coördinatenpaar. Een half-edge is een gericht lijnstuk. Dit houdt in dat vast staat wat het begin en het eindpunt is. Een lijnstukje bestaat dus uit twee half-edges. Een lijnstuk door (0,0) en (1,1) bestaat dus uit de half-edge (0,0)-(1,1) en uit de half-edge (1,1)-(0,0). Deze twee samen vormen een paar. Een face wordt aan begrensd door een gesloten kring van half-edges.
De records in een DCEL slaan de volgende geometrische en topologische informatie op:

1. Voor een vertex (v) worden de coördinaten opgeslagen in een veld en een verwijzing naar een willekeurige half-edge die de vertex als begin punt heeft.
2. Voor een half-edge (e) wordt er het volgende opgeslagen:
   - een verwijzing naar zijn beginpunt,
   - een verwijzing naar zijn tweeling (de half-edge waarmee het samen een paar vormt). Het is niet nodig om ook het eindpunt op te slaan, omdat dit hetzelfde is als het begin punt van de tweeling.
   - Een verwijzing naar het vlak waarvan het een grens is. De structuur is zo gekozen dat dit vlak aan de linkerkant van de half-edge ligt, waneer je het lijnstuk van begin tot eindpunt zou volgen.
   - Een verwijzing naar de volgende half-edge op de grens van het vlak.
   - Een verwijzing naar de vorige half-edge op de grens van het vlak.
3. Voor een face (f) wordt er een verwijzing naar één van de half-edges die onderdeel is van de buitengrens van het vlak, opgeslagen Tevens wordt voor elk van de eilanden in het vlak een verwijzing naar één van de half-edges op de grens van dit eiland opgeslagen. Dit aantal verwijzingen naar eilanden is een willekeurig aantal en dus niet in een vaste tabelstructuur.

Voor elke vertex en edge staat vast hoeveel informatie er in een record opgeslagen moet worden. Voor een face staat dit niet vast, hoe meer eilanden er in een face zitten hoe meer er opgeslagen moet worden [BKOS97].



Figuur 2-3 Voorbeeld van een DCEL

Vertex	coördinaten	Uitgaande lijn
V1	2,4	E12
V2	0,1	E23
V3	5,0	E31

Half-edge	Origin	Twin	Face	Prev	Next
E12	V1	E21	F1	E31	E23
E21	V2	E12	F2	E32	E13
E13	V1	E31	F2	E21	E32
E31	V3	E13	F1	E23	E12
E23	V2	E32	F1	E12	E31
E32	V3	E23	F2	E13	E21

Face	buitenkant	Binnenkant
F1	-	E12
F2	E21	-

Deze manier van opslaan, waarbij de zijden van een vlak in tweeën en met verwijzingen wordt opgeslagen, wordt ook wel “Winged-Edge” structuur genoemd. De informatie in een DCEL is genoeg om eenvoudige operaties uit te voeren, zoals het berekenen van de omtrek van een vlak. Het is bijvoorbeeld mogelijk om over de grens van een willekeurig vlak te lopen door gebruik te maken van de verwijzingen naar de volgende half_edge, en te beginnen bij de half-edge waarnaar verwezen wordt door de face.

2.1.5 TOPOLOGICAL_MAP & PLANAR_MAP

Het verschil tussen een Topological_map en een Planar_map in CGAL is dat in een Topological_map alleen de topologie zit. Een Planar_map is een uitbreiding op de Topological_map met waarden (coördinaten) voor de punten. Als we zeggen dat we een vlak hebben dat begrensd wordt door de drie zijden e1, e2 en e3, dan hebben we al een (simpele) Topological_map. Als we de knooppunten van de zijden vastpinnen op coördinaten dan hebben we een Planar_map. Zo kunnen we ook hun namen verklaren. Een Topological_map voor de topologie van een kaart. Dit gebeurd door de tabellen aan te maken en de geheugenplekken te reserveren en de verwijzingen naar geheugenplaatsen op te slaan. Een Planar_map voor een ruimtelijke kaart. Zowel de Topological_map als de Planar_map maken gebruik van een DCEL zoals die binnen de Topological_map gedefinieerd is, maar pas binnen een Planar_map worden er echt waarden in de geheugenplekken, gereserveerd voor de x en y coördinaten van de punten, weggeschreven.

2.2 DATABASE BEGRIPPEN

2.2.1 STRUCTURED QUERY LANGUAGE(SQL)

Alhoewel SQL een algemeen geaccepteerde bevragingstaal is zijn er toch altijd wat de syntaxis betreft per database kleine verschillen, hoewel de algemene strekking hetzelfde is. De database waarmee gewerkt is binnen dit afstuderen is de Ingres 2.0 database. Ingres maakt net als vele andere databases gebruik van een logfile waarin alle opdrachten uitgevoerd worden en niet gelijk in de database zelf. Gaat alles goed dan kan er met behulp van het commando "commit" alle commando's in de logfile ook in de werkelijke database worden uitgevoerd. Door "autocommit" te gebruiken wordt een geslaagd commando in de logfile telkens automatisch gelijk ook uitgevoerd in de werkelijke database. De volgende SQL commando's zijn belangrijk geweest tijdens het classificatie proces:

"\g" Opdrachten kunnen over meerdere regels uitgespreid worden, pas als er een keer de backslash plus een g en enter worden doorgegeven wordt de opdracht uitgevoerd, in veel databases staat dit gelijk met de ";". Dit wordt gezien als de separator tussen meerdere SQL statements.

Help Door gebruik te maken van het help commando kan een overzicht van de database of een overzicht van een tabel, view of index worden weergegeven.

- *Help*
  Alles wat in de database zit en waar de gebruiker toegang tot heeft.
- *Help* tabelnaam
  Een overzicht van alle kolomen van de tabel
- *Help Table* tabelnaam
  Uitgebreidere informatie waarin ook het aantal rijen terug te vinden is.

Create Er zijn twee mogelijkheden om een tabel te maken:

1. Een tabel volledig opnieuw aanmaken, waarbij voor alle kolomen het type en de voorwaarden gedefinieerd moeten worden.
2. Met behulp van een selectie opdracht, waarbij de namen, types en voorwaarden van de tabel waaruit wordt geselecteerd worden overgenomen.

Select Hier volgt een deel van de syntax van het commando "Select":

Box 2-1 SQL Select syntax

```
syntax:
        Select kolomnamen
        from tabelnaam
        where voorwaarden
        group by kolomnaam
```

Voorbeeld:
Select
classificatie, waardering, **count**(distinct id)
from deelgebouwen
where classificatie <>'-'
group by classificatie, waardering

Vergelijkingen van waarden:

- Waarden kunnen vergeleken worden met behulp van =, <>, >, >=, < en <=, maar ook met ***between, not between*** of ***in***. Deze laatste wordt gevolgd door een lijst met meerdere waarden.
- De waarden uit tekst kolomen kunnen met behulp van + aan elkaar samen gevoegd worden tot 1 waarde. (In andere databases wordt hiervoor wel || gebruikt.)

Het select statement kan ook gebruikt worden om een samenvatting te maken van de gegevens die in de tabel zitten. Hiervoor wordt veel gebruik gemaakt van de term **count**. In dit voorbeeld is “count (distinct id)” terug te vinden. Dit commando telt het aantal regels die aan de voorwaarden voldoet. Vanwege de term “distinct” telt “count” alleen die identificaties die van elkaar te onderscheiden zijn. Op deze manier kunnen dubbele regels eruit gehaald worden.

Update

Het aanpassen van waarden in een tabel gebeurt met het commando “Update”

Box 2-2 SQL Update syntax

Syntax:

```
Update tabelnaam
from andere tabelnaam
set kolomnaam=waarde
where voorwaarden
```

Voorbeeld:

```
Update deelgebouwen
from gebouwen
set classificatie='V', waardering,=1
where deelgebouwen.gebouw_id=gebouwen.id
      and gebouwen.n_dlgebouwen=1
      and deelgebouwen.n_acn=1
```

2.2.2 NAAMGEVING BINNEN DE DATABASE

Twee kolommen met dezelfde naam in verschillende tabellen kunnen van elkaar onderscheiden worden door de naam van de tabel en een punt ervoor te zetten. Voorbeeld: “gebouw.id”. Alleen als er geen onderscheid mogelijk is moet de naam van de tabel voor de kolomnaam komen te staan, maar het mag altijd. Voor de leesbaarheid van de SQL-statements heb ik altijd de naam van de tabel erbij gezet.
Als er gebruik gemaakt wordt van een tabel die niet het eigen eigendom is, dan moet er ook weer de naam van de eigenaar en een punt voor de tabelnaam komen te staan. Voorbeeld: ”maureen.deelgebouwen”. Het is alleen mogelijk om tabellen te gebruiken van andere als zij daar toestemming voor hebben gegeven door middel van het statement: “grant select”.
Wanneer de naam van een tabel uit meerdere woorden bestaat is er gebruik gemaakt van de underscore (_).
Voorbeeld: “deelgebouw.gebouw_id=gebouw.id”.

2.3 GEOGRAFISCH INFORMATIE SYSTEEM (GIS)

Een geografisch informatiesysteem of kortweg GIS wordt ook we “Spatial Information System” genoemd. Als we de term splitsen in Geografisch en Informatiesysteem, dan wordt informatiesysteem vaak gezien als een database en geografisch houdt in dat er een ruimtelijke component in zit. Geografisch wordt nog wel eens vervangen door spatial (=ruimtelijk). Deze omschrijving is nogal ruim, want een gewoon adressen bestand van een willekeurige vereniging zou volgens deze beschrijving al een GIS zijn.

Wat de geodeet naast de opslag van betekenisvolle objecten ook bij een GIS vindt horen is de computer en software om de data te kunnen bevragen. Een GIS bestaat dus uit: 1 data met een geografische component en 2 een bevragingssysteem. Een belangrijk onderdeel van een GIS is de interactiviteit wat zich vaak uit in het gebruiken van de kaart als interface. Hieronder vallen onder andere het editen, invoeren, analyseren, opslaan, selecteren, visualiseren, distribueren, ... Een van de doelen van een GIS is het ondersteunen bij het nemen van beslissingen. Het begrip *beslissingsondersteunend* hoort thuis in een definitie van een GIS.
GIS is anders dan andere informatie systemen omdat de opgeslagen data verwijzen naar objecten met een specifieke locatie in de ruimte. De data heeft een ruimtelijk adres. Binnen een GIS wordt de data meestal onderscheiden in geometrische data en attribuut data. De geometrische data bevatten verwijzingen naar de locatie en dimensie van objecten, terwijl de attribuutdata de niet geometrische karakteristieken bevat. De objecten binnen een GIS worden geometrisch weergegeven met punten, lijnen, vlakken en volumes.
Binnen een GIS is het mogelijk te exploreren, analyseren en presenteren. Het exploreren houdt in dat de dataset bekeken wordt, verkend, de gebruiker gaat op zoek naar structuren en delen van de dataset die van interesse zijn voor hem. De analyse wordt gebruikt om nieuwe informatie uit de bestaande data set te halen door relaties tussen meerdere sets te maken, bijvoorbeeld geometrische versnijdingen. Het presenteren houdt in dat met behulp van de juiste grafische regels de informatie wordt overgebracht naar een publiek.

Wat betekent GIS voor dit afstudeeronderzoek?

Zoals uit het volgende hoofdstuk over het Kadaster en de kadastrale gegevens zal blijken bevatten de gebruikte datasets ruimtelijke componenten. Eigenlijk bestaat het afstudeeronderzoek uit een GIS analyse van de datasets. Bij de zoektocht naar het beantwoorden van de hoofdvraag is de grootste activiteit het analyseren van de aanwezige data. Er worden geen nieuwe gegevens toegevoegd maar relaties gelegd tussen al bestaande data.

2.3.1 DE GEBRUIKTE GIS PROGRAMMA'S

ArcView is een software pakket bedoeld voor het visualiseren, bekijken, bevragen en analyseren van ruimtelijke data. ArcView is platform onafhankelijk, wat inhoudt dat de software onder verschillende besturingssystemen draait. ArcView is ontwikkeld door ESRI. ArcView werkt met zogenaamde projecten, hierin worden alle verwijzingen naar gegevens vastgelegd. Verwijzingen zijn bijvoorbeeld tabellen, lay-outs, legenda. Een project bevat dus zelf geen tabbellen met de ruimtelijke en attribuut data. Binnen het afstudeerproject is dit programma eigenlijk alleen gebruikt om de gevonden resultaten te presenteren.

Arc/Info Omdat het niet gelukt is om het geschreven overlay programma goed werkend te krijgen, door fouten in CGAL zelf, heb ik gebruik gemaakt van het resultaat dat Wilko Quak verkregen heeft in het kader van het kadaster project. Dit resultaat is gemaakt met behulp van Arc/Info, vandaar dat deze toch in het lijstje genoemd is, terwijl ik er zelf niet mee gewerkt heb. De problemen met CGAL staan beschreven in hoofdstuk 7. Arc/Info is een krachtig reken programma om GIS operaties in uit te voeren.

ArcExplorer Gedurende de tijd dat ArcView niet beschikbaar was op de casagrande (centrale computer van de sectie), zijn de resultaten steeds met dit programma bekeken. ArcExplorer is een viewer waarmee Arc/Info bestanden gevisualiseerd kunnen worden. De mogelijkheden binnen dit programma met name het maken en opslaan van een goede legenda zijn beperkt.

Om de gegevens uit de database in deze programma's te visualiseren moeten ze vanuit de database geëxporteerd worden naar bestanden die ingelezen kunnen worden. De geometrisch attributen van de gebruikte datasets zijn aan het begin uitgelezen met behulp van een zelf geschreven programma zodat ze gelijk vanuit de database naar een formaat worden omgezet dat door al deze programma's kunnen worden gelezen. Deze programma's bevatten een verzameling van unix-commando's, SQL statements en Arc/Info commando's die met het in en uitlezen van gegevens te maken hebben. Meer over hoe deze "executables" werken is terug te vinden in hoofdstuk 6. Nadat de geometrie uitgelezen is vind er alleen nog maar uitwisseling van attributen weer van en naar de database.

KADASTER &
KADASTRALE GEGEVENS

In dit hoofdstuk zal een overzicht gegeven worden van alle gegevens die voor dit afleidings en classificatie proces zijn gebruikt. Een van de eisen was dat alleen gegevens gebruikt zouden worden, voor het afleidings en classificatie proces, die eigendom zijn van het Kadaster. Vandaar dat er eerst een kleine introductie volgt wat het Kadaster is en daarna zullen achtereenvolgens AKR, LKI en ACN besproken worden.

Kadaster

Landmeetkundig en Kartografisch Informatiesysteem (LKI)

Administratieve Kadastrale Registratie (AKR)

Adres Coördinaten Nederland (ACN)

3.1 KADASTER

Het Kadaster heeft als doel ‘de rechtszekerheid te bevorderen bij het maatschappelijk verkeer inzake vastgoed’. Tot dat ‘vastgoed’ behoren niet alleen onroerende zaken, maar ook schepen en luchtvaartuigen. Rechtszekerheid wil in dit geval zeggen, dat duidelijk is aan wie een bepaald vastgoedobject toebehoort en wat er de kenmerken van zijn. Om dit doel te bereiken, verzamelt het Kadaster gegevens over vastgoed in Nederland, houdt deze nauwkeurig bij in openbare registers en op kadastrale kaarten en geeft informatie aan bedrijven, particulieren en andere belanghebbenden. [i1]
Op het moment wordt de grote schaal topografische en kadastrale gegevens beheerd in het LKI-systeem, welke de gegevens opslaat in een Ingres database. LKI staat voor landmeetkundig kartografisch informatiesysteem. Rechten van subjecten op objecten en andere administratieve gegevens worden bijgehouden in het AKR-systeem. AKR staat voor automatische kadastrale registratie [OMQ2000]. LKI zal ook wel naar verwezen worden met de term geometrische database en AKR zal ook wel naar verwezen worden met administratieve database.

3.2 ADMINISTRATIEVE KADASTRALE REGISTRATIE (AKR)

Het kadastraal uittreksel is een overzicht met de basisgegevens uit de kadastrale registratie op een bepaalde datum. Dit overzicht vermeldt o.a.:

- Naam, adres, geslacht en geboortedatum van de kadastrale eigenaar
- Kadastrale aanduiding;
- Oppervlakte
- Zakelijke rechten;

grootte	integer(4)	70
bladnr	integer(4)	2
ruitletter	char(1)	E
x	integer(4)	199735
y	integer(4)	461270
beb_code	char(1)	1
soort_cult	char(2)	11
ind_meer_cult	char(1)	N
koopsom	integer(4)	185000
koopjaar	integer(4)	1990
object_id	integer(4)	140436868
classif	integer(4)	201
tmin	char(19)	23-08-1999 14:09:57
tmax	char(19)	01-01-1990 00:00:00
x_akr_objectnummer	char(17)	BBG01C 03016G0000

Figuur 3-1 Voorbeeld van een aantal gegevens in AKR

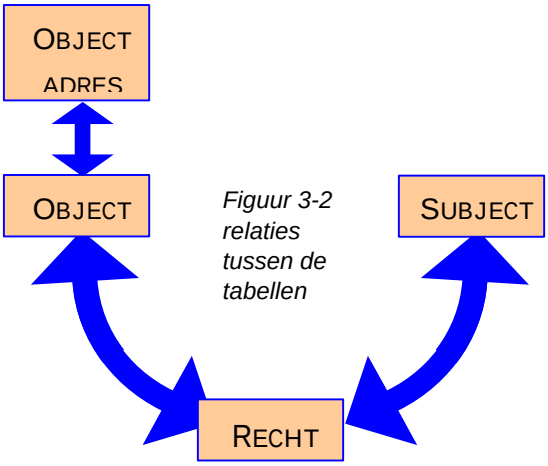
Een kadastraal uittreksel kan een vastgoedobject, maar ook een schip of vliegtuig betreffen. Het uittreksel is bedoeld om aan te geven wie bij het Kadaster als eigenaren van een bepaald object staan geregistreerd op een bepaalde datum. Daarnaast wordt aangegeven welke objecten ten name van een eigenaar staan geregistreerd bij het Kadaster op een bepaalde datum. [i2]

3.2.1 MASSALE OUTPUT (MO)

Massale output is een van de producten van het Kadaster. Dit product is gericht op gemeenten. Het bestaat uit een verzameling gegevens uit AKR samen voor een gegeven gebied en een gegeven tijdstip. Het MO-overzicht bestaat uit een heleboel verschillende tabellen. In onderstaande lijst wordt een overzicht gegeven van deze tabellen. De tabellen die voor dit afstudeerproject interessant zijn, hebben een accent gekregen.

Groepsrelatie Huwelijksrelatie Objectbelemmering **Object** **Objectadres** Objectmutatie Ontstaan uit Onzelfstandig deel Overgegaan in	**Recht** Register 9 Register 9 tekst Rente **Subject** Subjectmutaties subjectrelatie

Lijst 3-1 Overzicht van de tabellen die bij een massale output horen.



Figuur 3-2 relaties tussen de tabellen

De tabellen uit het MO die uiteindelijk gebruikt zullen worden zijn Object en Objectadres. Verder zal er voor de controle van de gevonden classificaties gebruik worden gemaakt van de eigenaren van een object. Via de tabel recht kan de eigenaar uit de subject-tabel worden geselecteerd. Dit gebeurd om te kijken of de eigenaar een natuurlijk persoon is of een niet-natuurlijk persoon. Voorbeelden van niet natuurlijke personen zijn BV en NV.
In figuur 3-2 is de relatie weergegeven tussen de voor dit project belangrijkste tabellen. In de tabel recht wordt een object aan één of meer subjecten gekoppeld.
De volgende lijst bevat een overzicht van een aantal attributen uit deze vier tabellen. Alleen de attributen die voor het afstudeerproject interessant zijn worden hier weergegeven, er zijn er nog veel meer.

ObjectAdres	Object	Recht	Subject
Kadastrale gemeentecode Sectie Perceelnummer Objectindexletter Objectindexnummer	Kadastrale gemeentecode Sectie Perceelnummer Objectindexletter Objectindexnummer	Kadastrale gemeentecode Sectie Perceelnummer Objectindexletter Objectindexnummer	Subjectnummer
Woonplaats	Grootte	Gerechtigde	Natuurlijke persooncode
Straatnaam	Bebouwingscode		Soort niet natuurlijke persoon
Huisnummer	Soort cultuur onbebouwd		Woonplaats
Huisletter	Indicatie meer culturen		Straatnaam
Huisnummertoevoeging	Koopsom		Huisnummer
Soort cultuur bebouwing	Koopjaar		Huisletter
Indicatie meer culturen			Huisnummertoevoeging
Postcode			Postcode

Lijst 3-2 Een overzicht van de attributen per tabel, De gekleurde attributen geven de externe sleutels aan.

```
CREATE TABLE MO_recht_selectie
AS SELECT
      x_akr_objectnummer, gerechtigde
FROM MO_recht
```

Uit de AKR tabel over recht kan de relatie tussen object en subject, ofwel eigendom en eigenaar worden gehaald, hiervoor worden de kolomen voor de identificatie, en de gerechtigde geselecteerd.

```
CREATE TABLE MO_subject
AS SELECT subject_id, nat_pers_code,
     soort_niet_nat,
     postcode+huisnummer
     +huisletter+huisnummertoevoeging
     as s_adres
FROM MO  subject
```

De AKR tabel over subjecten (bijvoorbeeld eigenaren). Hieruit worden de subjectidentificatienummers geselecteerd. Deze zijn gelijk met de waarden uit de kolom gerechtigde van de vorige tabel (MO_recht), verder worden geselecteerd de natuurlijke persoonscode, het soort niet natuurlijke persoon (vaak BV en NV), en het adres door de waarden uit de kolommen postcode, huisnummer, huisletter en huisnummertoevoeging samen te voegen.

3.2.2 CULTUURCODE

De cultuurcode is terug te vinden in twee tabellen: Object en Objectadres. Over het algemeen geldt dat als een object onbebouwd is de code in de objecttabel staat en dat wanneer het object(perceel) wel bebouwd is dat dan de code in de tabel objectadres staat. In de objecttabel heet het attribuut waarin de cultuurcode zit “soort_cult”, in de objectadres tabel heet het attribuut “soort_cult_beb”. We selecteren alleen de codes 11 tot en met 15 plus de 00 uit de objectadrestabel. Dit laatste omdat voor deze objecten nog niet bepaald is wat het hoofdgebruik is, en deze dus ook woningen kunnen bevatten.

Betekenis van de codes 11 tot en met 15:
11 staat voor eengezinswoningen,
12 staat voor meergezinswoningen,
13 staat voor bijzondere woonvormen,
14 staat voor boerderijen,
15 staat voor recreatie (2e woning).

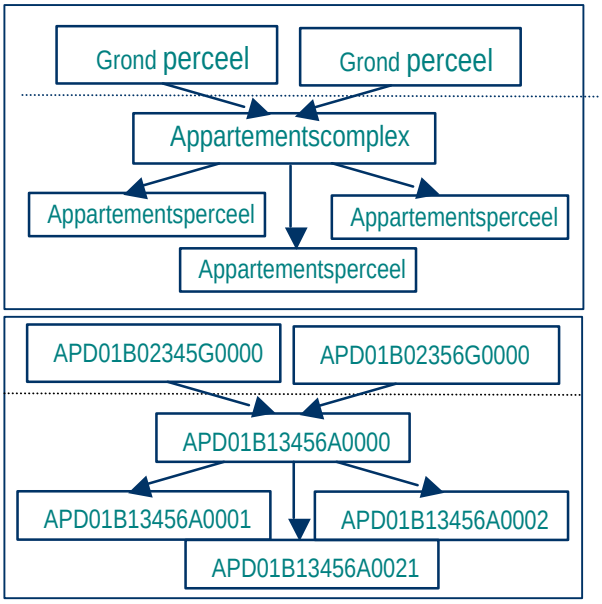
Tabel 3-1 Overzicht Cultuurcodes in de set Apeldoorn

Overzicht	MO_Object	MO_Objectadres
Code 00	44132	20482
Codes 11-15	9	62025
Codes die als deel wonen in zich hebben (vb winkel & wonen)	1626	466
Totaal objecten	69444	90540
Totaal gebruikt in classificatie (codes 00,11-15) totaal en in %	--	82507
		91.2 %

Er is gekozen, door de opdrachtgever, om alleen de cultuurcodes waarbij de hoofdmoot wonen is te gebruiken om te classificeren. Juist de codes die niet alleen wonen in zich hebben zouden het opstellen van de classificatieregels bemoeilijken. Tevens is het in deze gevallen niet mogelijk om de woning van het niet-woning deel te onderscheiden. Het doel van de classificatie is een schatting van het type van alle woningen, dit woord schatting houdt in dat we niet voor de hele 100% hoeven te gaan.

3.2.3 ADMINISTRATIEF MODEL

Binnen het administratief model worden de relaties tussen deelpercelen, appartementpercelen en gehele percelen beschreven.



Figuur 3-3 Structuur tussen percelen en appartementsrechten, eerst in tekst daarna in bijbehorende objectcodes.

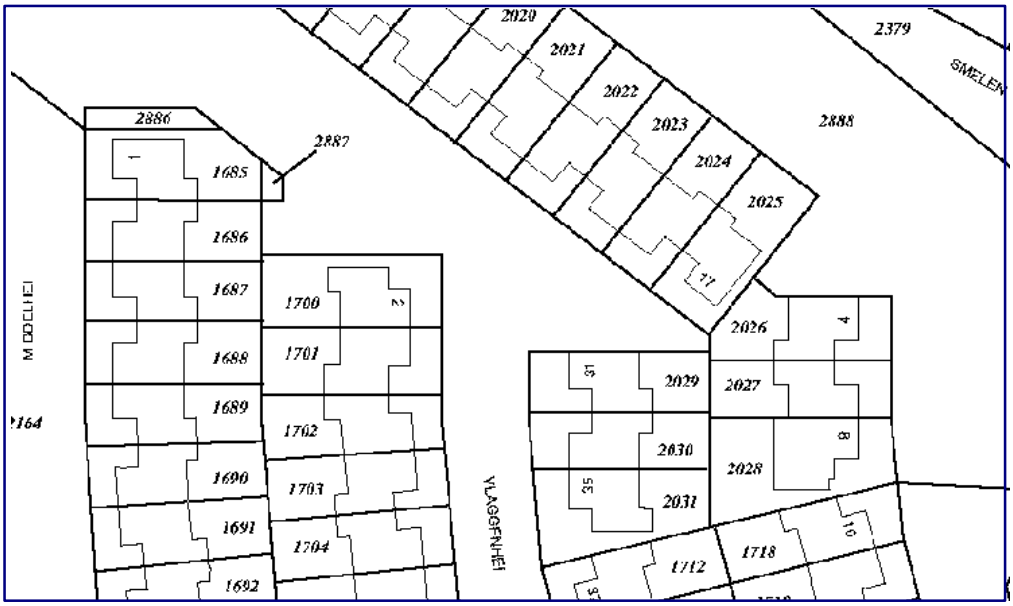
De gehele percelen zijn de percelen zoals die in de kaart getekend worden. De identificatie van een geheel perceel bestaat uit een gemeente code (wdb02), een sectie(B) en een perceelnummer(02762), een indexletter (G) en een indexnummer(0000). In het geval van de gehele percelen is de indexletter altijd “G” en het indexnummer “0000” Maar er zijn naast percelen nog twee soorten administratieve objecten: appartementsrechten en deelpercelen. Deelpercelen, ontstaan als er een deel van een perceel verkocht is, maar deze nog niet landmeetkundig is opgemeten. Pas zodra deze delen opnieuw zijn ingemeten en verwerkt worden de deelpercelen pas echte percelen en krijgen ze een eigen geheelperceel objectnummer.

Een deelperceel verschilt alleen in de laatste 5 tekens van het erbij behorende gehele perceel. Een deelperceel heeft de indexletter “D” en een indexnummer van “0001” of hoger.
Appartementen bestaan alleen als administratieve objecten. Op één (of meerdere) percelen zit dan een appartementscomplex met een totaal andere objectidentificatie. De laatste 5 tekens van het objectnummer van een appartementencomplex zijn “A0000”. Het perceelnummer verschilt van het onderliggende perceel. Dit appartementscomplex is dan weer opgedeeld in Appartementen die hetzelfde begin van de objectidentificatie hebben, maar waarbij de laatste 4 tekens groter dan “0000” zijn.

3.3 LANDMEETKUNDIG EN KARTOGRAFISCH INFORMATIESYSTEEM (LKI)

In LKI zijn alle gegevens die nodig zijn om een kadastrale kaart te tekenen opgeslagen.

De Kadastrale Kaart

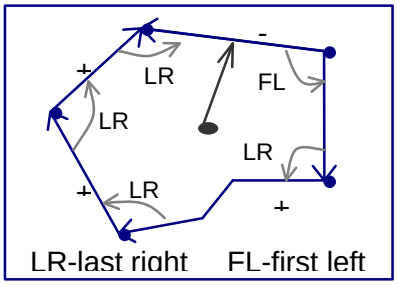


Figuur 3-4 Voorbeeld van een stukje kadastrale kaart

De kadastrale kaart geeft op schaal een overzicht van de ligging van de kadastrale percelen en grenzen. Op de kaart staan kadastrale grenzen, perceelnummers en gebouwen. De kaart is onderverdeeld naar kadastrale gemeente, sectieletter en bladnummer. De gebruiker kan zich op de kaart oriënteren aan de hand van het perceelnummer, de kadastrale grenzen, topografie en namen van straten en waterwegen. De kadastrale kaart vormt de geografische koppeling in een digitaal vastgoedbestand. De kadastrale kaart is een visuele presentatie van gegevens uit de geometrische database [i3].

Het geometrische model

Sinds 1997 worden alle veranderingen door de tijd expliciet in het database management systeem (DBMS) opgeslagen. Dit gebeurt door voor alle metrische attributen (punt, polylijn of box) naast de meetgegevens ook de attributen tmin en tmax toe te voegen. Objecten zijn geldig tussen tmin en tmax in. Het geometrische model van de kadastrale percelen (figuur 3-5) is gebaseerd op de winged_edge topologie [OMQ2000]



Figuur 3-5 Topologie structuur

De LKI gegevens zijn in een aantal tabellen in de database opgeslagen. De volgende lijst geeft een overzichtje van deze tabellen. Deze tabellen zijn te herkenen doordat ze allemaal met “xfio_” beginnen. Degene die voor dit afstudeeronderzoek interessant zijn hebben een accent gekregen. In een van de attributen van de lijnen tabel wordt weergegeven waar de gegevens hun herkomt hebben. Staat hier een “G” dan is het lijnstuk afkomstig uit de Grootschalige Basis Kaart Nederland (GBKN). Staat er een “B” dan is het lijnstuk afkomstig uit de kadastrale Basiskaart. Het komt voor dat beide codes aan het lijnstuk hangen.

Mapdata
Element
Map
Author
Boundary
Gcpunt
Symboolpunt
Polygoon
Topoline
Line
Parcel
Parcelover
Symboolpunt
Text

Lijst 3-3 tabellen met LKI-gegevens

3.4 ADRES COÖRDINATEN NEDERLAND (ACN)

ACN is een database met geografische coördinaten van alle adressen in Nederland, die bekend zijn bij de PTT. De geografische positie van de adrescoördinaat binnen in een perceel is afhankelijk van de werkelijke situatie. In principe geldt het volgende:

- Als er maar één adres gekoppeld is aan het perceel, dan kan de adrescoördinaat overal in het perceel liggen, niet specifiek in een gebouw dat op het perceel staat.
- Als er meer dan één hoofdgebouw op het perceel staat, dan zullen de adrescoördinaten in het gebouw liggen waarbij het adres hoort.
- Als er een gebouw is met meer dan één adres (bijvoorbeeld appartementen) dan wordt het adres binnen in het gebouw geplaatst. Soms hebben alle adressen in een gebouw dezelfde coördinaat, soms hebben verschillende adressen verschillende coördinaten. [K98]

ACN-ID	2031164	ACN-ID	2031163
X	196211000	X	196250000
Y	477657000	Y	477620000
WOONPLAATS	WENUM WIESEL	WOONPLAATS	WENUM WIESEL
STRAATNAAM	Jonas	STRAATNAAM	Jonas
POSTCODE	7345EB	POSTCODE	7345EB
HUISNR	23	HUISNR	19
TOEV		TOEV	

Figuur 3-6 Voorbeeld van een tweetal records in ACN

GEOMETRISCHE VERSNIJDING

Binnen het onderzoek moest er een geometrische versnijding met attribuutovererving (engels: overlay) worden uitgevoerd tussen twee kaartlagen, bestaande uit vlakken. Meer specifiek moest een vereniging (engels: union) worden gemaakt tussen de gebouwen en percelen. Dit vormt een groot deel van het onderzoek. Daarom is er een heel hoofdstuk gewijd aan de achtergronden van één van de belangrijkste GIS operaties die er zijn: de geometrische versnijding. De versnijding is een rekenintensieve operatie, en er bestaan een aantal verschillende algoritmen voor.

Definitie Versnijding

Algoritmen

Rekencomplexiteit

Standaard overlay in CGAL

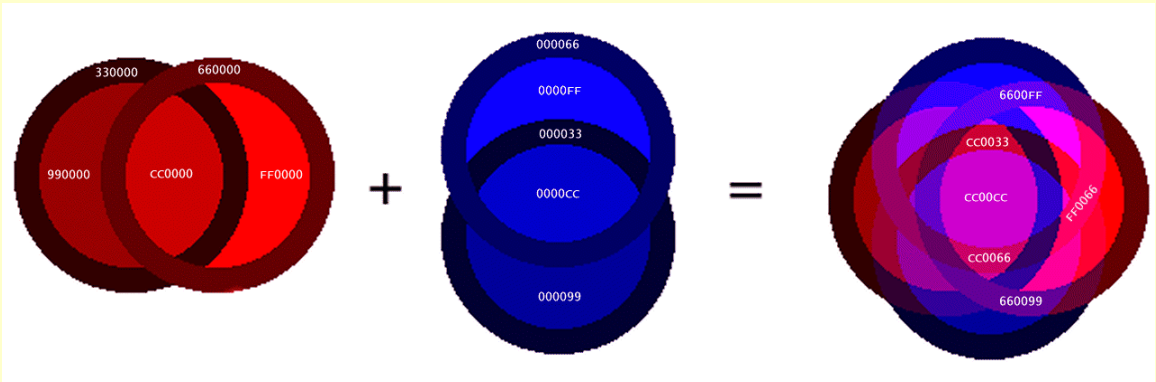
Attribuutovererving

4.1 INLEIDING

Het principe van overlay is het combineren van twee aparte kaarlagen met elkaar tot één nieuwe kaartlaag. De kenmerken van de nieuwe kaart zijn afkomstig uit de gecombineerde kaarten.
Stel we hebben kaartlaag A en we hebben kaartlaag B. Dan kunnen deze kaartlagen in theorie op 16 verschillende manieren gecombineerd worden naar een nieuwe kaartlaag (zie bijlage A). De belangrijkste (en meest gebruikte) combinaties zijn intersectie (A AND B) en vereniging (A OR B). Intersectie is het gebied waar de kaarten A en B elkaar overlappen. Een voorbeeld waarbij gebruik wordt gemaakt van intersection binnen GIS operaties is het beantwoorden van een vraag als; “welke woongebieden hebben last van geluidsoverlast?”. Alleen de gebieden die aan alle twee de kenmerken voldoen (woningen **en** binnen geluidszone) moeten dan geselecteerd worden.
Met behulp van de boolean expressies is alleen nog maar bepaald, welke vlakken terug zullen worden gevonden in de nieuwe kaart, maar de kaart moet ook betekenis krijgen, dit gebeurt door de kenmerken van de originele kaarten door te geven aan de nieuwe kaart. Een voorbeeld hiervan is het volgende figuur (figuur 4-1). De kaarten die in de overlay gaan zijn rood en blauw, de resulterende kaart bestaat uit veel meer kleuren: rood, blauw en paars. Alleen die gebieden die in beide originele kaartlagen gekleurd waren krijgen verschillende tinten paars.

Paarse kaart: alle mogelijk voorkomende kleuren (RGB-code in hexadecimaal) opgeteld uit de originele kaarten.

Blauwe kaart→ / Rode kaart	-	000033	000066	000099	0000CC	0000FF
-	-	000033	000066	000099	0000CC	0000FF
330000	330000	330033	330066	330099	3300CC	3300FF
660000	660000	660033	660066	660099	6600CC	6600FF
990000	990000	990033	990066	990099	9900CC	9900FF
CC0000	CC0000	CC0033	CC0066	CC0099	CC00CC	CC00FF
FF0000	FF0000	FF0033	FF0066	FF0099	FF00CC	FF00FF



Figuur 4-1 Overlay van een rode met een blauwe kaart.
Kleuren zijn weergegeven in hexadecimale RGB code.

Kaarten zijn ingedeeld in lagen. Het soort geografische data kan sterk verschillen: een wegen kaart kan voornamelijk bestaan uit een collectie van lijn elementen, terwijl een plaatsnamenkaart zal bestaan uit punten met labels en een kaart met provincies zal een vlakken kaart zijn, toch maken ze samen een overzicht van een heel land. Als deze kaarten over elkaar gelegd worden is het mogelijk om een thematische overlay visueel te bekijken. Als er nu specifieke combinaties benadrukt moeten gaan worden dan moeten de kaartlagen niet alleen visueel getoond worden maar zullen ze echt moeten worden versneden met elkaar zodat de interesse gebieden berekend en geselecteerd kunnen worden. De gevonden gebieden kunnen visuele kenmerken krijgen zodat ze worden benadrukt. [BMcD98]

4.2 DEFINITIE VAN EEN VERSNIJDING

De geometrische versnijding gebeurt in principe in 3 stappen:

1 Het vinden van snijpunten tussen elementen van verschillende lagen;
2 Nieuwe knooppunten en lijnstukken definiëren;
3 Opnieuw opbouwen van de structuur (identificaties toekennen en curves en faces opnieuw opbouwen).

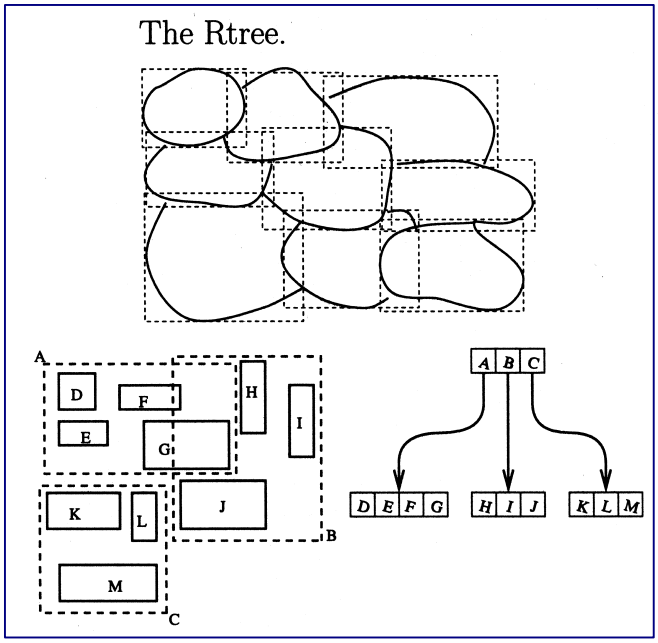
Een definitie van een versnijding tussen 2 kaarten is:
We hebben kaart K_1 en het restgebied r_1 daaromheen, en we hebben kaart K_2 en het restgebied r_2 daaromheen.
$(K_1 \cup r_1) \times (K_2 \cup r_2) \rightarrow (K_3 \cup r)$ waarbij $r = r_1 \cap r_2$
Het gebied of beter gezegd het bereik wat we willen is het bereik van K_1 en K_2 samen. Dus alles wat er uit deze combinatie komt, behalve het gebied r. De kenmerken of attributen van elementen uit de nieuwe kaart zijn een functie van de kenmerken uit de originele kaarten: $A_3 = f(A_1, A_2)$.
Een definitie van een versnijding van meer dan 2 kaarten gaat analoog aan de hierboven gegeven definitie.
Voor het bepalen van de snijpunten moet er een tolerantie gedefinieerd worden. Een van de redenen hiervoor is het voorkomen van slivers. Een tolerantie geeft aan waneer twee punten als hetzelfde punt moeten worden gezien. Voor deze tolerantie wordt vaak ε (epsilon) gedefinieerd. Als de afstand tussen twee punten kleiner is als ε dan worden deze samen gevoegd in één punt.

De thematische attributen uit de twee kaartlagen kunnen:

- Allemaal overgenomen worden in de nieuwe kaart.
- Gedeeltelijk overgenomen worden in de nieuwe kaart
- Gebruikt worden in functies, die de nieuwe thematische attributen bepalen.

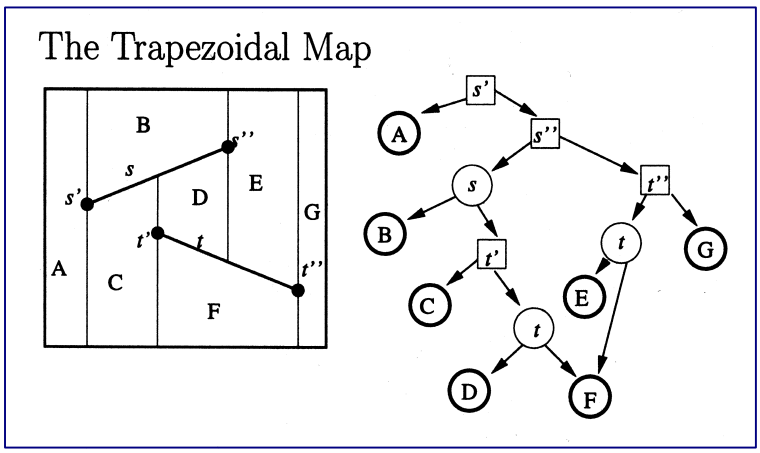
4.3 OPSLAG STRUCTUREN

Logische operaties op lijnen en polygonen die resulteren in elementen die opgedeeld of verwijderd worden uit de database zorgen voor zowel “computational cost” als ook opslagruimte omdat de hoeveelheden afhangen van de operaties die uitgevoerd moeten worden. Een voorbeeld, als twee polygonen elkaar versnijden tijdens een vereniging (engels: union) zodat er 3 polygonen ontstaan dan moeten er 3 polygonen en hun attributen, afkomstig uit de originele twee polygonen, opnieuw in de database worden opgeslagen. In de praktijk blijkt dat het aantal polygonen dat verwijderd en toegevoegd moet worden is erg groot en kan van tevoren niet altijd geschat worden. Om te zorgen dat dit soort overhead binnen de perken blijft worden dit soort dure operaties vaak op sub-sets van de totale dataset uitgevoerd. Het aanpassen van ruimtelijke data is meer dan alleen een record weggooien en een andere toevoegen; alle topologische connecties moeten ook aangepast worden [BMcD98]. Een hulpmiddel bij het aanpassen van de data is een goede opslagstructuur. Een voorbeeld hiervan is de Rtree. In figuur 4-2 is een grafische weergave van deze opslagstructuur weergegeven. De gegevens worden ingedeeld volgens boundingboxes. Een bounding box om een groep objecten (vlak of lijn) wordt gevormd door het laagste en hoogste x- en y coördinaat. Eerst grote boundingboxes(A, B en C) en binnen deze boundingboxes, die elkaar enigszins kunnen overlappen word een verdere opdeling gemaakt in kleinere boundingboxes (D t/m M). Deze opdeling zou steeds verder door kunnen gaan totdat er in een boundingbox nog maar één lijnelement zit.



Figuur 4-2 grafische weergave van een Rtree. [Bron Q95]

4.3.1 TRAPEZOIDAL MAP



Figuur 4-3 grafische weergave van een Trapezoidal map. [Bron Q95]

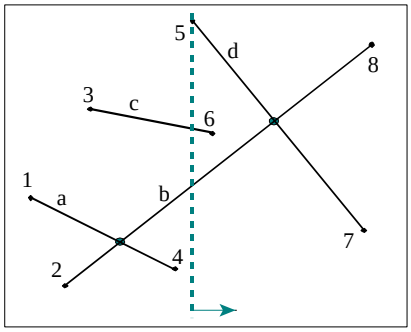
De trapezoidal map dankt zijn naam aan de vormen die ontstaan wanneer de scheidingslijnen tussen de verschillende gebieden en de originele lijnstukken uit de kaart worden getekend. In figuur 4-3 is een grafische weergave van een trapezoidal map te zien. Het hele gebied wordt in trapeziumvormen verdeeld. Deze opslagstructuur is geprogrammeerd in CGAL. Dit is zelfs de default opslag structuur. Deze structuur is (ongemerkt) gebruikt bij het programmeren van de versnijding.

4.4 VERSNIJDINGSALGORITMEN

Er wordt nu verder ingegaan op stap één van de versnijding: het vinden van de intersectiepunten. Het berekenen van de intersectiepunten kan op verschillende manieren. Het meest simpele algoritme is het “Brute force “ algoritme. Deze test elk lijnelement van de ene kaart met elk lijnelement uit de andere kaart. Aannemend dat elke kaart (laag) n lijnelementen bevat, duurt dit proces $O(n^2)$ [O94]. Het is mogelijk om dit efficiënter te doen en de duur korter te maken dan $O(n^2)$, Dit kan door rekening te houden met de verdeling van de aanwezige lijnen. Hiervoor zijn verschillende algoritmen ontworpen. Voorbeelden van deze algoritmen zijn Plane-sweep, Uniform Grid en de R-tree methode. Deze worden in de volgende deelparagrafen behandeld

4.4.1 PLANE-SWEEP ALGORITHME

Het “plane sweep” algoritme sorteert alle knooppunten op hun x-coördinaten. De beide dimensies (x en y) worden dus niet gelijk behandeld. Het “Plane Sweep” algoritme (Bentley & Ottman zijn één van de eerste die deze methode beschreven hebben in 1979 [094]) maakt al gedeeltelijk gebruik van het feit dat lijnen in een bepaald deel van het interessegebied niet kunnen snijden met lijnen in een heel ander deel. Dit gebeurt door de eindpunten van alle lijnstukken in een prioriteiten rij te zetten, gebaseerd op hun x-coördinaten als de sweeplijn vertikaal gedefinieerd wordt. Het is ook mogelijk om de sweeplijn horizontaal te definiëren, dan worden de lijnstukken op hun y-coördinaat gesorteerd.



Figuur 4-4 vertikale sweeplijn

Naast de prioriteitenlijst (Q) wordt er ook een constructieboom (R) bijgehouden met actieve lijnen. Actieve lijnen zijn lijnen die nog steeds door de sweeplijn gesneden worden. Lijnen worden op snijdingen gecontroleerd met hun buren [094]. Een van de volgende drie mogelijkheden gebeuren als er een punt verwijderd wordt uit de prioriteitenlijst.

Tabel 4-1 Eenvoudig voorbeeld van een vertikale sweeplijn

Q	R	Optie	Wat gebeurt er
1 2 3 4 5 6 7 8	Leeg	1	R+a
2 3 4 5 6 7 8	a	1	R+b, Q+i1
3 i1 4 5 6 7 8	b, a	1	R+c
i1 4 5 6 7 8	b, a, c	3	R a,b verwisselen
4 5 6 7 8	a, b, c	2	R-a
5 6 7 8	*b, c*	*1*	*R+d*
6 7 8	b, c, d	2	R-c, Q+i2
i2 7 8	b, d	3	R b,d verwisselen
7 8	d, b	2	R-d
8	b	2	R-b
Leeg	Leeg	Eind	

1 Punt p is een linker eindpunt van lijn s: lijn s wordt toegevoegd aan R (in volgorde van de y coördinaten) en als de lijn snijdt met de lijnen uit R die direct voor of na s komen, dan wordt het intersectie punt toegevoegd aan Q.
2 Punt p is een rechter eindpunt van lijn s: lijn s wordt uit R verwijderd. De boven buur en onder buur van het verwijderde lijnelement worden nu nog vergeleken op intersectie punten, en als deze bestaat wordt dit punt ook weer in Q toegevoegd.
3 Punt p is een intersectiepunt van 2 lijnelementen (s en t): verwissel de volgorde van s en t in R en kijk of ze snijden met hun nieuwe buren, zo ja voeg deze intersectie punten dan weer toe aan Q.

Rekencomplexiteit van de plane sweep methode:

Wanneer n het aantal elementen uit een kaart is en k het aantal gevonden snijpunten tussen 2 kaarten zijn, dan worden er 2n+k punten verwijderd uit Q. Elke bijhouding van zowel Q als R kost O(log n), dus het algoritme draait in O((n+k) log n). De meeste algoritmen gebruiken O(n+k) aan ruimte [094].
Om de plane sweep methode te verbeteren zijn er een tweetal mogelijkheden [KBS91b]: De eerste, “sweep-line partition”, selecteert als invoer alleen de polygonen die dicht bij de sweeplijn gelokaliseerd zijn. Hierdoor zal de wachtrij Q alleen maar gevuld zijn met een klein deel van de hele kaartlaag en dus aan het begin niet helemaal gevuld worden met alle polygonen.
De tweede methode deelt de totale kaart op in horizontale (of verticale, afhankelijk van hoe de sweeplijn gedefinieerd is) stroken, die als een stuk in het plane-sweep algoritme worden gestopt. Deze methode heet de “Strip plane-sweep”. Er moet bij deze laatste methode wel extra aandacht worden besteed aan het aan elkaar sluiten van de verschillende strips. Hier wordt verder niet op ingegaan op hoe dat in zijn werk gaat. Zie hiervoor literatuur van Kriegel [KBS91b].

4.4.2 UNIFORM GRID

Een andere manier van opdelen gebeurt binnen het uniform grid algoritme. Dit algoritme deelt het interesse gebied op in uniforme delen. Binnen deze delen worden alle elementen met “brute force” met elkaar vergeleken op versnijdingen. Brute force betekent dat alle lijnen van de ene kaart vergeleken worden met alle lijnen uit de andere kaart. Bij de uniform grid gebeurt de intersectie alleen nog maar tussen lijnstukken gelokaliseerd in dezelfde gridcellen. Een voordeel van deze methode is dat de x en y richting een gelijke rol krijgen.
Deze methode werkt goed bij uniform verdeelde data, maar minder bij niet uniforme verdeling. Dit komt omdat de verschillende gridcellen de ene keer erg leeg zijn en de andere keer vol met lijnen. Alle lijnen binnen een volle cel worden allemaal met elkaar vergeleken. Binnen deze cellen neemt de rekencomplexiteit extra toe. Verder kunnen lijnen tot meerdere cellen behoren. Deze methode is minder geschikt voor geografische data, omdat over het algemeen deze data niet uniform verdeeld is.

4.4.3 R-TREE VERSNIJDINGSALGORITME

Dit algoritme is nauw verbonden met de opslag structuur R-tree. In paragraaf 4.3 is al kort ingegaan op wat nu eigenlijk een R-tree is ingegaan. Het principe van deze overlay methode is dat alles wat eenmaal opgeslagen is in de R-tree ook topologisch correct is. Dit houdt in dat er geen snijpunten zijn, zonder dat er een knooppunt is. Elke drie of meer lijnen die eindigen in een zelfde knooppunt hebben allemaal per definitie exact dezelfde coördinaten in dit knooppunt. Dit is nodig zodat de topologie van de nieuwe kaart gereconstrueerd kan worden.
Vanwege deze reden bevat het algoritme zowel “InsertRtree()” als “CheckedInsert()”. Bij de eerst wordt er geen controle uitgevoerd op intersectie bij het toevoegen van lijnen. Dit kan in principe niet voorkomen binnen een bestaande kaartlaag. Als een van de twee kaarten met een bestaande Rtree is opgeslagen, kan deze direct met de insertRtree worden ingeladen in de nieuwe kaart. De andere kaartlaag moet daarna met de CheckedInsert worden ingelezen.
De overlay met behulp van Rtree bestaat uit de volgende commando’s:

- *CreateRtree():* Maakt een nieuwe (lege)Rtree aan;
- *InsertRtree(lijn):* Voor alle elementen uit kaart 1 (mits deze al topologisch correct is, dwz. geen intersecties, behalve op knooppunten.):
- *CheckedInsert(lijn):* Voor alle elementen uit kaart 2. Alle lijnen worden in eerste instantie getest op overlappende bounding boxen met al eerder ingevoegde lijnstukken. Als de bounding boxen elkaar raken of overlappen wordt pas echt gecontroleerd op versnijdingen. De gehele Rtree wordt langsgelopen. En als de grote bounding box (A,B en C in figuur 4-2) al niet overlapt met de nieuwe lijn, hoeven de verdere opdelingen in deze box niet meer gecontroleerd te worden.

Alle lijnelementen zitten dan in de nieuwe Rtree. Daarna moet alleen nog de juiste topologie bepaald worden, zodat er vlakken ontstaan en de juiste attributen bij deze vlakken bepaald worden. Dit gebeurt met:

- *FormAreaLoops();*
- *ComputeAttributes();*

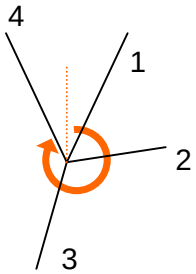
Hierna is er een kaart met de juiste attributen bij de juiste vlakken ontstaan. Vervolgens is het mogelijk om de Rtree weg te gooien. Alhoewel het ook gebruikt kan worden voor het snel terug vinden van bepaalde vlakken, en dergelijke. Als je de nieuw gevonden kaart wilt gaan bevragen is het mogelijk om de Rtree nog even te behouden. De belangrijkste functie is de CheckedInsert, met daarin gevat de Intersect functie.
Deze CheckedInsert ziet er als volgt uit:

```
CheckedInsert(new_line)          //recursieve functie
        Intersectie= FALSE;
        Overlap_line = RetrieveFirstRtree(box_of(new_line, epsilon));
        While ((overlap_edge != NULL_id) and (not intersected)) do
                If Intersect (overlap_line, new_line, over1, over2, new1, new2) then
                        (Intersectie=TRUE);
                        RemoveRtree(overlap_line);
                        InsertRtree(over1);
                        InsertRtree(over2);
                        CheckedInsert(new1);
                        CheckedInsert(new2);
                Else
                        Overlap_line = RetrieveNextRtree();
                End_if;
        End_while
If (not Intersectie) then InsertRtree(new_line);
```

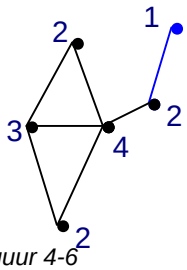
Tekstbox 4-1 Rtree CheckedInsert [O94]

Een nieuwe lijn wordt getest op overlap met behulp van zijn boundingbox plus een afstand epsilon met eerder toegevoegde lijnen in de Rtree. Binnen de intersect functie moet er bepaald worden of de lijn ook werkelijk een versnijding heeft met een lijn uit de Rtree. Een overzicht van mogelijke relaties tussen twee lijnen is terug te vinden in figuur 7-3 en 7-4 uit hoofdstuk 7.
Als er werkelijk een versnijding tussen twee lijnen is, worden er (maximaal) vier lijnstukjes teruggegeven door de functie intersect: over1, over2, new1 en new2. De versneden lijn wordt uit de Rtree gegooid en de nieuwe lijnstukken (over1 en over2 worden weer met behulp van insert ingevoegd in de Rtree). De andere twee stukken worden met opnieuw checkedInsert toegevoegd, want ze kunnen nog meer lijnen snijden. De intersect functie hoeft niet altijd 4 lijnstukken terug te geven. Het kan ook zo zijn dat een lijn niet snijdt, maar raakt aan een andere lijn.

Topologie reconstructie en voortplanting van attributen



Figuur 4-5 volgorde van lijnen rondom een punt



Figuur 4-6 Danglingedge, bij alle knooppunten zijn het aantal uitgaande lijnen aangegeven

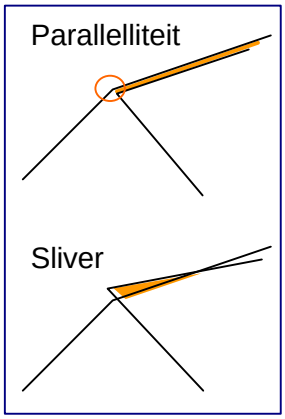
Dit stuk draait om de opdrachten *FormAreaLoops* & *ComputeAttributes*. Het is verstandig om deze twee stappen tegelijk uit te voeren, omdat ook bij het berekenen van de attributen alle vlakken langsgelopen moeten worden. Als alle lijnelementen zijn toegevoegd, kunnen de Arealoops worden gemaakt. Dit kan met behulp van alle knooppunten die een lijst bij zich hebben, waarop is bijgehouden wat de inkomende en uitgaande lijnen zijn. Deze lijsten zijn gesorteerd op hoek. Een van de dingen die daarbij ook moeten gebeuren zijn alle dangling edges te bekijken en weg te gooien of aanpassen. Dangling edges worden bepaald door knooppunten die maar één uitgaande lijn hebben. Dat is de definitie die in dit afstudeeronderzoek gebruikt wordt. Deze losse lijnen willen we niet als we het hebben over kaartlagen, opgebouwd uit vlakken. De blauwe lijn in figuur 4-6 wordt door sommigen ook wel lone-edge genoemd. Afhankelijk van de definitie zou het lijnstuk dat verbonden is met deze blauwe lijn ook aangemerkt kunnen worden als een dangling edge.
Door het vormen van de AreaLoops kunnen we de vlakken van de nieuwe kaart bepalen en door de links-, rechtsinformatie van de lijnen kunnen we de juiste originele vlakidentificaties en daarbij de attributen terugvinden. Speciale aandacht moet besteed worden aan eilanden. Zolang de nieuwe vlakken zowel lijstukken uit de ene als de andere kaart bevat zal het geen probleem zijn om de originele identificatie terug te vinden en daarbij ook de gewenste attributen. Maar wanneer we te maken hebben met vlakken die slechts grenzen bevatten uit één kaart zal er ook gekeken moeten worden naar het vlak of die vlakken aan de andere kant van de grenzen. Zeker in het geval van eilanden zijn er zowel problemen bij het Area-loop als ComputeAttributes. Want het eiland moet gekoppeld worden aan zijn ouder. Dit kan het beste door een ruimtelijk zoeken in de Rtree met een Point-in-Polygon test.
De opdracht AreaLoops wordt pas gestopt als alle lijnen twee keer zijn gebruikt; een keer links en een keer rechts. Dit is hetzelfde aantal als alle half-edges wanneer er van de DCEL gebruik wordt gemaakt.
Het terugvinden van de juiste attributen uit de originele kaart kan alleen als de links-, rechtsinformatie goed wordt doorgegeven en bijgehouden gedurende het gehele proces. Een alternatief is om achteraf een punt binnen elk polygoon te berekenen en met een point-in-polygon algoritmen het originele vlak in beide kaarten terug te zoeken om zo de attributen door te kunnen geven. Dit terugvinden van de attributen kan efficiënt gebeuren als er voor de invoerkaarten een ruimtelijke zoekstructuur bestaat, zoals de Rtree. Het vinden van een punt binnen een polygoon is nog niet zo eenvoudig. Zolang de polygoon een convexe vorm heeft “is het geen punt” (=eenvoudig) en kan bijvoorbeeld het zwaartepunt berekend worden, maar is het polygoon niet convex, dan moet er een andere methode gezocht worden. Dit wordt in paragraaf 4.7 verder besproken.

De rekencomplexiteit van het Rtree Algoritme:

De Map-overlay met n invoerelementen en k intersecties, gebruikt op zijn meest k Rtree element verwijderingen en n+3k element toevoegingen. Elke verwijdering en toevoeging kost O(log n) om de Rtree weer juist te krijgen (balanced). Daarom is er in het ergste geval een rekencomplexiteit van O((n+k)(log(n+k))). De benodigde ruimte is O(n+k)

4.5 FRAGMENTATIE PROBLEMEN IN DE VERSNIJDING

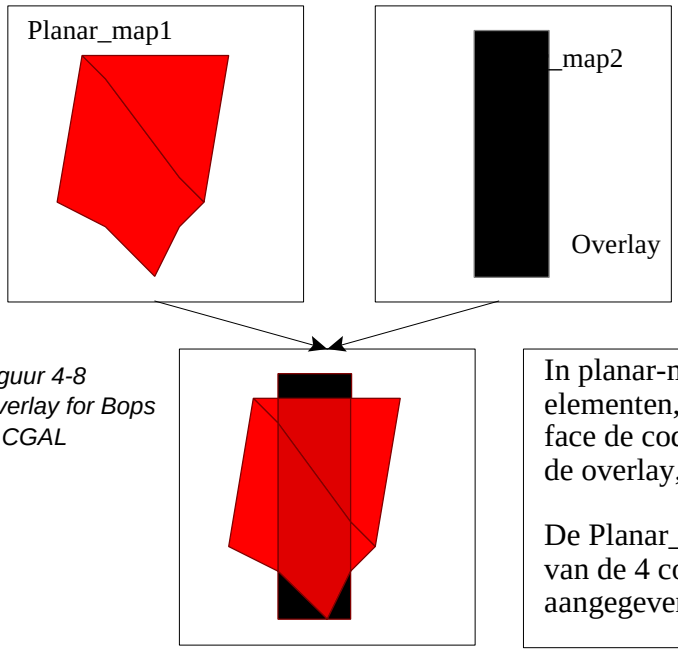
Een topologisch correcte kaartlaag houdt minimaal in dat er op alle eindpunten en kruisingen van lijnen knooppunten zitten. Bij de versnijding kunnen situaties ontstaan waarbij het resultaat volgens deze definitie correct is, maar die niet wenselijk zijn. Voorbeelden hiervan zijn “parallelliteit” en slivers. Deze laatste worden ook wel “spurios” polygonen genoemd. In figuur 4-7 is van beide een grafisch voorbeeld gemaakt.



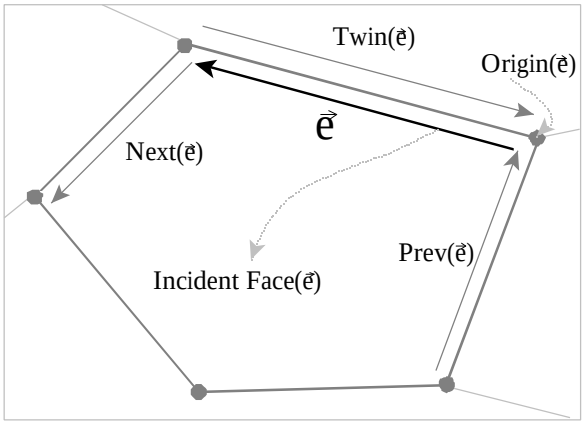
Figuur 4-7 Fragmentatie problemen

In sommige gevallen kan de versnijding zorgen voor “spurious” polygoon. Dit is vaak te danken aan kleine verschillen bij het digitaliseren van de grenzen die op dezelfde plak zouden moeten liggen. Dit kan onder andere komen door vervormingen van de analoge kaarten waar vanaf gedigitaliseerd is, of omdat de verschillende kaartlagen ook vanaf verschillende analoge kaarten zijn gedigitaliseerd. Maar ook fouten tijdens het landmeten kunnen de oorzaken zijn. [BMcD98]
Er zijn verschillende oplossingen voor dit probleem:
Ten eerste kan er 1 kaartlaag aangewezen worden als zijnde degene die niet mag veranderen, en de anderen worden vervormd om op deze kaartlaag aan te sluiten. Een andere mogelijkheid is het verwijderen van deze ongewilde polygonen als hun oppervlakte kleiner is dan een bepaalde waarde. Hierbij moet wel bepaald worden bij welke aangrenzende polygoon het spurious polygoon wordt toegevoegd. De derde methode is om een “smoothing” window over deze polygonen te laten gaan en er een gemiddelde grens te laten berekenen.
Belangrijk bij het wegwerken van deze problemen is het bepalen van de tolerantiemaat (epsilon). In figuur 4-7 is bijvoorbeeld een cirkel om een knooppunt getekend met een straal gelijk aan de tolerantie. Omdat het knooppunt van de andere lijn binnen deze cirkel valt zal dit punt als een punt moeten worden beschouwd.

4.6 STANDAARD OVERLAY IN CGAL



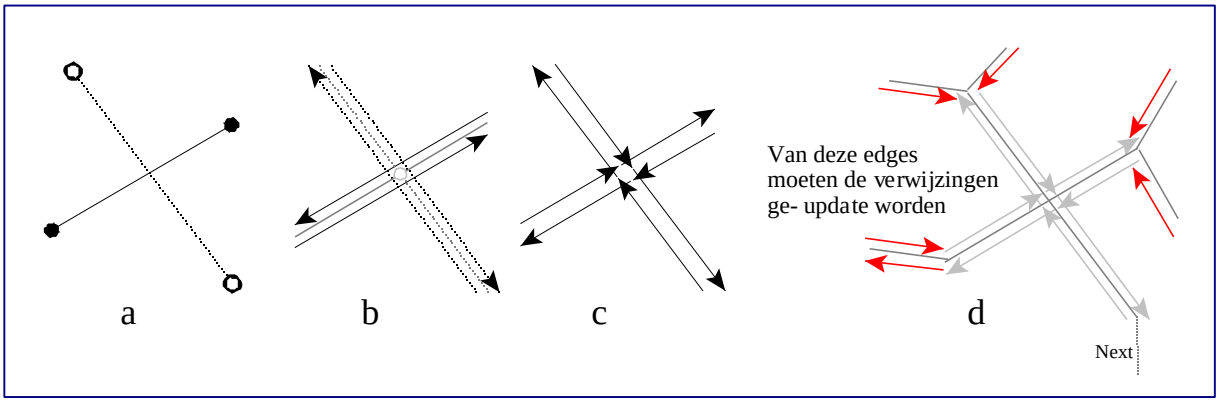
Figuur 4-8 Overlay for Bops in CGAL



Figuur 4-8 Opslagstructuur CGAL

In hoofdstuk twee is al de opdeling in "vertices, edges and faces" besproken. De kaartlagen zijn opgeslagen in een DCEL, waarbij de volgorde van de records bepaald wordt via de opslagstructuur van een trapezoidal map. De DCEL (doubly connected edge list) houdt in dat er voor elk vlak en punt één record en voor elk lijnstuk twee records worden bijgehouden. Alle geometrische en topologische informatie wordt hierin opgeslagen, en het is mogelijk om extra informatie/ attributen op te slaan. Een van de voorbeelden uit CGAL, waar extra informatie is toegevoegd, is het kaarttype dat gebruikt wordt bij de in CGAL aanwezige versnijding. Hier wordt een attribuut “kleur” toegevoegd. In deze kaartlaag hebben alle punten, lijnen en vlakken een extra attribuut gekregen in de vorm van een getal tussen 0 en 3, waarbij 0 gereserveerd is voor het onbegrensde vlak (Alles wat buiten de buitenste grenzen valt).
De ene kaartlaag wordt rood genoemd, wat inhoud dat het attribuut van alle objecten een waarde “1” krijgt. De andere kaartlaag wordt zwart genoemd. Dit houdt in dat het attribuut van alle objecten een waarde “2” krijgt. Tijdens het versnijden wordt voor elk punt lijn en vlak opnieuw

een code bepaald. Aan de hand van deze code is te bepalen uit welke kaartlaag een bepaald punt, lijn of vlak komt.
Er moet nog al wat gebeuren binnen de DCEL als er twee lijnen versneden worden. Over het algemeen moet er één lijnstuk verwijderd worden en drie of vier nieuwe lijnstukken met de juiste verwijzingen worden toegevoegd. Daarnaast moeten alle elementen die naar het oude verwijderde lijnstuk verwijzen ge-update worden. Elk lijnstuk heeft als attributen verwijzingen naar het voorgaande en het volgende lijnstuk. Verder bevatten ze verwijzingen naar het beginpunt van de lijn en naar het bijbehorende face. Een eenvoudig voorbeeld van een DCEL en de aanpassingen in deze DCEL, na de versnijding, is te vinden in bijlage B1.



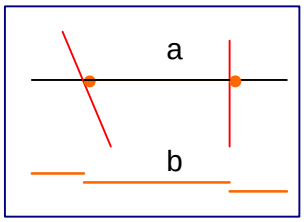
Figuur 4-9 versnijding van twee lijnen (a), opdeling in half_edges (b), nieuw ontstane halfedges (c) en de halfedges waarvan de verwijzingen moeten worden aangepast.

De in CGAL aanwezige versnijding staat in bestand “Pm_overlay_for_bobs.h”. Het belangrijkste stuk uit dit bestand staat hieronder weergegeven:

Tekstbox 4-2 de belangrijkste aanroepen uit de versnijding

```
Pm_overlay_for_bobs (const Planar_map& pm1,
                     const Planar_map& pm2,
                     Planar_map_for_bobs& pm_out)
    {
        Segment_container l_out;
        Collect_edges (pm1, pm2, l_out);
        Insert_edges (l_out, pm_out);
        Colorize (pm_out);
    }
```

“Segment_container “ is een lijst met lijnstukken. Dit wordt een container genoemd. Deze wordt in de volgende stap gevuld met alle lijnstukken die uiteindelijk in de kaartlaag moeten komen.
Collect_edges (pm1, pm2, lout) is een apart stuk programma. Binnen dit stuk programma vindt de berekening van de versnijding tussen alle lijnstukken plaats. Alle segmenten worden op de juiste plaats (volgorde



figuur 4-10 Gevonden versnijdingspunten (a) en uiteindelijk gevormde segmenten (b)

wordt bepaald door de trapezoidal mapstructuur) in de container gestopt. Hierbij krijgt elk segment de juiste links-, rechtsinformatie als attribuut mee krijgt. Om dit te bepalen worden nog twee belangrijke delen van het programma aangeroepen, "Intersect_segment_planar_map (segment, kleur, planarmap, punt_container)" en "Collect_portions(punt a en b, linker kleur, rechterkleur, punt_container, segment_container)".
Eerst worden voor alle lijnen (segmenten) in de ene kaartlaag de intersectiepunten bepaald met de tweede kaartlaag. Deze punten krijgen een kleurcode 3 (rood en zwart). Daarna wordt in Collect Portions de originele lijn opgedeeld in delen en deze worden in de container gestort. Deze twee stappen, het berekenen van snijpunten en opdelen van het originele lijnstuk, gebeurt ook voor alle lijnen in de tweede kaart. Alle versnijdingen worden dus twee keer berekend. De uitvoer van Collect_edges is een gevulde lijst (Segment_container) met alle lijnstukjes die in de uiteindelijke kaart terecht moeten komen.

Insert_edges (l_out, pm_out) is een zelfstandig stuk programma. Binnen deze functie worden alle lijnstukjes uit de container in de kaart gevoegd. Bij de lijnstukjes zit ook de kleurcode informatie. Er wordt voor gezorgd dat deze ook goed in de kaart komt. Door het toevoegen van alle stukken uit de container wordt ook de structuur van de kaart weer opgebouwd. Door de volgorde van de elementen in de containerlijst kan het vullen van de kaartlaag snel gebeuren. In deze stap worden de topologische verwijzingen naar andere elementen in de DCEL opgeslagen.

Colorize (pm_out) is een zelfstandig deel van het totale programma. Alle lijnstukken zitten nu goed in de kaart, maar de vlakken zijn nog niet voorzien van een kleur. Deze moeten allemaal langsgelopen worden en ook de juiste kleur krijgen. Dit gebeurt door naar alle grenzen van een vlak te kijken en daar de kleur (code 0, 1, 2, 3) vandaan te kopiëren.

Rekencomplexiteit van de versnijdingsmethode zoals die in CGAL zit:

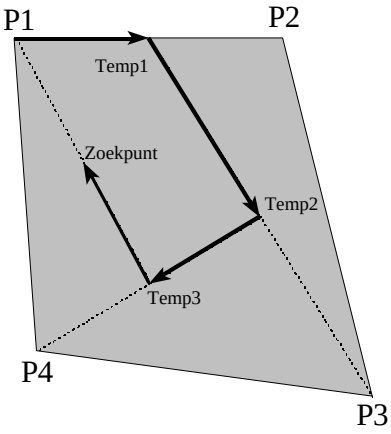
Als er in kaart 1 n segmenten en in kaart 2 ook n segmenten zijn, dan is het aantal vlakken maximaal 2/3 n (vlakken bestaan uit minimaal drie lijnstukken en één lijnstuk kan maximaal tot twee vlakken horen). Het aantal versnijdingen wordt aangeduid met k. Deze methode van het berekenen van de versnijdingen is een soort "brute-force" algoritme. Collect edges gebruikt 2n keer het berekenen met Intersect_segment_planar map. Elke keer dat deze functie aangeroepen wordt worden de n segmenten van de andere kaart doorlopen. In het slechtste geval dus 2 n^2. Daarnaast wordt 2 n keer collect_portions doorgelopen om de segment container te vullen met zowel niet versneden als de versneden segmenten.

Daarna worden weer alle 2 (n+k) segmenten in de kaart gelezen (Insert_edges). Daarna wordt in Colorize van alle faces en vertex de kleur bepaald. Om de kleur van een face te bepalen worden alle grenzen doorlopen. Dus worden weer alle segmenten doorlopen, dit kost ook weer 2(n+k). Alle punten worden apart doorlopen, dit zijn er minder dan n(anders zou je maar 1 grote cirkel hebben.), In totaal: $2n^2$ + 2 n + 2 (n+k) + ½ n + 2 (n+k) in totaal, waarbij een ½ n de schatting is voor het aantal punten (vertex). Dit betekent een rekencomplexiteit in het slechtste geval van $O(n^2)$. Met de mogelijkheden die CGAL biedt zou dit naar mijn mening toch sneller moeten kunnen.

4.7 PUNT IN EEN POLYGOON?

In de versnijding, zoals die beschreven is in de paragraaf hierboven, is besproken hoe voor de versnijdingskaartlaag bepaald kan worden uit welke invoerkaart een bepaald lijnstuk komt. Over het algemeen wil een gebruiker toch meer weten. Meestal moet voor elk vlak een identificatie of een bepaald attribuut uit de originele invoerkaarten worden gehaald, zodat de versnijdingskaart een thematische kaart kan worden waarbij de attributen bepaald worden uit de originele kaart. Hoe kunnen we de vlakidentificatie / attributen terugvinden?
Bij deze vraag denken we gelijk aan het point-in-polygon algoritme. Bij het terugvinden van de attributen en de identificatie van de originele kaarten zijn er eigenlijk twee stappen: De eerste is het vinden van een punt in een gegeven polygoon. Tweede stap is het vinden van de polygoon waarin een gegeven punt zich bevindt. Dit laatste zoeken gebeurt twee keer, in beide invoerkaartlagen.

4.7.1 EEN PUNT VINDEN IN EEN GEGEVEN POLYGOON



Figuur 4-11 bepalen van een punt in het polygoon.

De eerste stap, het vinden van een punt in een polygoon kan op verschillende manieren. Voor convexe figuren is het bepalen van een punt in het polygoon het gemakkelijkst. Er kan bijvoorbeeld gebruik gemaakt worden van het berekenen van het zwaartepunt (zelfs het zwaartepunt van drie punten van het polygoon is een punt in het polygoon zelf). Omdat het binnen voor dit afstudeeronderzoek geprogrammeerde versnijding toch nodig was om de polygoon rond te lopen is, is de volgende methode gebruikt om een punt binnen het polygoon te vinden. Bij elke stap is er een tijdelijk punt bepaald wat het gemiddelde is tussen het tijdelijke en het volgende punt. Zie figuur 4-11 voor het verloop van het tijdelijke punt.

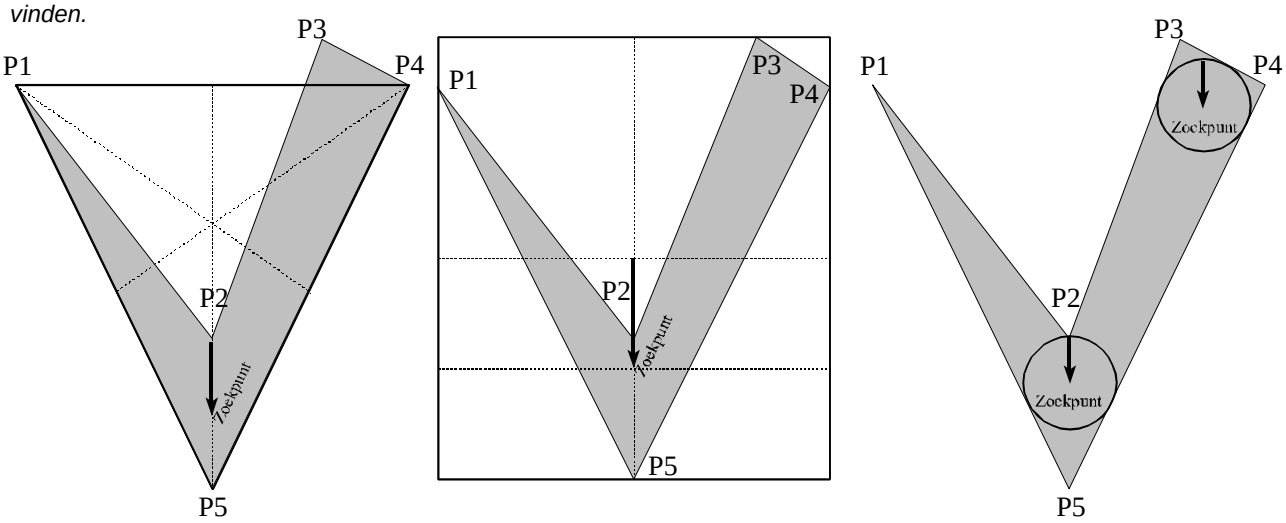
Bij niet convexe polygoon is het moeilijker om een punt binnen de polygoon te vinden. Er zijn veel meer methoden, maar er zullen er drie worden besproken: 1. boundingbox-middellijn, 2. convextriangulatie, 3.centrum van de grootste omvatte cirkel.

1. De bounding box en de middenlijn. Nadat de bounding box (bbox) is bepaald uit de maximum en minimum coördinaten kan het middelpunt van de bbox worden berekend. Daarna kan of de horizontale of de verticale lijn vanuit dit middelpunt worden gekozen. Kijken naar het aantal kruisingen van deze lijn met het polygoon geeft aan of dit punt wel of niet in de polygoon valt. Kruist hij éénmaal dan ligt het punt in de polygoon. Zolang er niet maar één kruising gevonden wordt, de lijn tussen het huidige middenpunt en de maximale of minimale coördinaat steeds weer in tweeen delen en het aantal kruisingen met de polygoon tellen.

2. Deze methode begint met het vinden van drie punten die een deel van de convex en hiervan een driehoek vormen. De andere punten langs gaan en kijken of deze binnen de driehoek vallen. Is dat niet het geval dan is het zwaartepunt van de driehoek een punt in de polygoon. Is dat wel het geval dan is het middenpunt van de lijn tussen dit punt en de dichtstbijzijnde van de drie hoekpunten een punt in de polygoon.

3. De grootst omvatte cirkel: Het centrum van de grootst omvatte cirkel, ligt op het skelet van de polygoon. Het skelet kan gevonden worden door eerst een lijn-Voronoi diagram te bouwen, dit kost O(n log n), waarbij n het aantal punten van de polygoon is [CGAL2000].

Figuur 4-12 Voorbeelden van de 3 methoden om een punt in een polygoon te vinden.

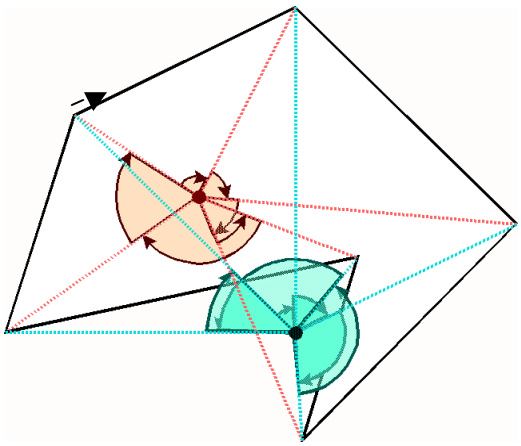


4.7.2 EEN POLYGOON VINDEN BIJ EEN GEGEVEN PUNT

De tweede stap in het vinden van een de originele attributen is een polygoon terug vinden bij een gegeven punt Hiervoor wordt een Point-in-polygon algoritme gebruikt. Dit is eigenlijk een algoritme waarbij nagegaan wordt of een gegeven punt in een gegeven polygoon hoort. Dit kan ook weer op een aantal verschillende manieren. Er doen twee verhalen voor over dit onderwerp de ronde, het verhaal van de slak en het verhaal van de slang.

De slak: (fictief verhaal) Neem een intelligente slak en geef hem een schrijfbord mee en geef hem de opdracht om richting de noordpool te lopen, laat hem elke keer dat hij over een grens van de polygoon heen gaat deze tellen. Telt de slak een oneven aantal grens overgangen dan bevond het punt zich in de polygoon, maar zijn het aantal grensovergangen even (of nul) dan bevond het punt zich buiten de polygoon. Je kan de slak natuurlijk ook al laten stoppen net voorbij de maximale y van het polygoon. Bij het vinden van de juiste polygoon uit de kaart kan dit principe ook wel gebruikt worden, maar dan moet de slak gewoon stoppen bij de eerste grensovergang en de identificatie van de lijn teruggeven. Vanuit de topologie kan dan bepaald worden vanuit welk vlak de slak kwam lopen.

De slang: Je staat in een kamer samen met een slang die zichzelf in de staart bijt. Als je de slang van bek tot staart volgt dan zijn er twee mogelijkheden: Of je draait één maal om je as en wordt door de slang ingesloten, of je kijkt alleen maar heen en weer, dan sta je buiten de slang. Deze methode om te bepalen of een gegeven punt in een gegeven vlak ligt kan ook op de volgende manier worden beschreven. Vanuit elke twee opvolgende punten en het te onderzoeken punt wordt een driehoek gevormd en het azimut (de hoek in p) berekend. Als alle punten van de polygoon zijn doorlopen kan dan de som van al die hoeken opgeteld worden. Is deze som (een oneven aantal keer) 360 graden dan bevindt het punt zich in de polygoon. Zo niet dan bevindt het punt zich buiten de polygoon.



Figuur 4-13 Azimut methode. Het oranje punt gaat helemaal rond en bevindt zich in de polygoon. Het groene punt gaat heen en weer en bevindt zich dus buiten het polygoon.

Tot zo ver alle achtergronden van de versnijding Een aantal elementen van dit hoofdstuk zijn gebruikt om een eigen versnijdingprogramma te schrijven. Dit is terug te vinden in Hoofdstuk 7.

WONINGEN

In dit hoofdstuk zal eerst naar het begrip woning worden gekeken. Alle mogelijk denkbare types worden gereduceerd tot vijf. Er wordt een overzicht van de kenmerken van deze vijf woningtypen gemaakt.
Elke woning zal uiteindelijk geclassificeerd moeten worden naar een van de vijf klassen. Elke woning moet tevens een positie op de kaart krijgen. Een identificatienummer vormt de sleutel tot elke woning en deze wordt net als de bijbehorende positie (x- en y coördinaat) en bijbehorend adres uit het ACN gehaald.

Woningtypen

Overzicht van woningtypes

DUWOZ

Kenmerken van de vijf verschillende woningtypen

5.1 INLEIDING

Een echte definitie van woning type, welke bedoeld is voor de vastgoedindex bestaat nog niet. De definities van verschillende type woningen zijn gedurende het afstudeeronderzoek vast komen te staan. Deze definities zijn belangrijk voor het opstellen van de kennisregels die gebruikt moeten gaan worden om de woningen uit de data te extraheren. Alhoewel in het begin de definities nog vrij open waren, is de definitievorming enigszins een inductief proces geweest. Tijdens het vormen van de beslis- / classificatieregels zijn alle kenmerken van de vijf woningtypen, zoals ze in dit afstudeerproject gebruikt zijn, pas vast komen te staan.

5.2 CLASSIFICATIEINDELING WONINGTYPEN

Door de opdrachtgever is vastgesteld dat er maar vijf woning typen gedefinieerd zullen worden, en deze zijn:

1. **Appartement** **(A)**
2. **Vrijstaand** **(V)**
3. **Twee-onder-één-kap** **(T)**
4. **Middenwoning** **(M)**
5. **Eindwoning** **(E)**

Deze indeling lijkt veel op de indeling die door het centraal bureau voor de statistiek (CBS) gebruikt wordt. De grootste CBS-klasse is in de indeling hierboven in twee groepen verdeeld.

Tabel 5-1: Woning aantallen in Nederland

Woningen van particuliere huishoudens, medio 1998 [CBS2000].	
Woningen naar bouwwijze	**aantallen ×1000**
Eengezinswoningen	4157,7
waar van:	
Vrijstaand	921,5
Twee-onder-een-kap	731,8
Hoek- of tussenwoning	2504,4
Flatwoning e.d.	1822,5

Het is zeker discutabel of deze indeling voldoende toereikend is, maar dit was het startpunt. In deze indeling zal boven en beneden woningen (2-lagen woningen) onder de appartementen komen te vallen. Onder de appartementen zitten natuurlijk ook de flats van 13 verdiepingen. In tabel 5-1 is te zien dat dit een van de grootste klassen is.

Ander discussiepunt is een type woning dat onder middenwoning dan wel hoekhuis terecht zal komen: de geschakelde woningen. Geschakelde woningen zijn woningen die tot een rij worden gevormd doordat de bijbehorende garages(bijgebouwen) de woningen aan elkaar schakelen. Binnen LKI is het niet mogelijk om de geschakelde garages van de woningen te scheiden. Ze vormen samen het hoofdgebouw zoals het opgeslagen is in LKI.

5.3 OVERZICHT VAN IN NEDERLAND VOORKOMENDE WONINGTYPES

Het is de bedoeling geweest om zo veel mogelijk van de bestaande woningtypen en beschrijvingen van woningen te vinden en deze in te delen in een van deze 5 groepen. Een eerste brainstorm naar woningtypen geeft het volgende rijtje: rijtjeshuis, hoekhuis, villa, vrijstaande woning, appartement, galerijflat, portiekflat, portiekwoning, 2-onder-1-kap, studio, maisonnette, …
Om een overzicht van alle mogelijke woningtypen te krijgen is er gekeken naar de categorieën die horen bij de wet Waardering Onroerende Zaken (WOZ) en op internet bij de makelaars.

Makelaars maken onder andere gebruik van [i4] [i5]:

- *Woning*: woonhuis, appartement of recreatiewoning.
- *Typen*: tussenwoning, hoekwoning, vrijstaand, verspringende tussenwoning (schakelwoning) en helft van dubbel.
- *Woning soort*: bijzonder object, eenvoudige woning, herenhuis, bovenwoning, appartement, eengezinswoning en monumentaal pand.

In “waarderingsinstructie 1999” van de waarderingskamer in het kader van de wet WOZ worden de volgende categorieën genoemd:

- Flats met lift, appartementen, maisonnettes en duplexwoningen
- Flat zonder lift, Boven-, beneden-, portiek- en etagewoningen
- Eengezins-, rij-, hoek-, tussen- en drive-in woningen
- 2 onder 1 kap-, geschakelde woningen en herenhuizen
- vrijstaande-, individuele woningen en woonboerderijen
- Recreatie-, bejaarden-, studentenwoningen en overige woningen.

Van de tot nu toe gevonden typen woningen heb ik gekeken of ze in mijn woordenboek staan en zo ja wat dan de definitie is.[W93]

Flats geheel op één verdieping liggende wooneenheid in flatgebouw.
Appartementen woning, deel uitmakend van een groter gebouw.
Maisonnettes wooneenheid in groot woongebouw met slaapverdieping boven de woonverdieping.
Duplexwoningen woning voor twee kleine gezinnen die later bij minder woningnood, kan worden verbouwd tot één groter huis.

Bovenwoning bovenhuis: verdiepingswoning, afgescheiden van het andere gedeelte van het huis en met afzonderlijke ingang.
Benedenwoning benedenhuis: benedenste gedeelte van een huis.
Portiekwoning huis met een portiek, inz. Huis dat met een aantal andere gebruik maakt van dezelfde portiek.
Etagewoningen woongelegenheid met en op verdiepingen, maar geen hoogbouw.
Eengezinswoning huis, berekend op bewoning door één gezin, dus zonder onder- of bovenburen (en dikwijls met een tuin).
Rijtjeshuis een eenvoudig woonhuis in een rij van dezelfde huizen.
Hoekhuis huis op de hoek van een straat.
Tussenpand gelegen tussen twee (of meer) andere percelen, (tegenstelling vrijstaand huis).
Drive-in woningen woning met ingebouwde garage.
Twee-onder-één-kapwoning - niet gevonden.
Geschakelde woningen - niet gevonden.
Herenhuizen 1 deftig woonhuis 2 gezinswoning voor één familie (geen flat).
Vrijstaande woning – alleenstaand.
Woonboerderijen boerderij verbouwd tot, ingericht als woonhuis.
Bejaardenwoning - woning, gebouwd met het oog op de bijzondere behoeften van bejaarden.
Studentenhuis - door studenten bewoond huis

Intuïtief zijn deze gevonden woningtypen ingedeeld in de vijf hoofdtypen. Van de vijf hoofdtypen worden in paragraaf 5.5 de kenmerken waarop in dit afstudeeronderzoek geselecteerd wordt opgesteld. Kenmerken zijn onder te verdelen naar onder andere: aanduiding, bestemming, omschrijving, gegevens, gebruikers en oppervlakte.

Woningtypen	Gevonden categorieën, intuïtief ingedeeld.
Middenwoning of hoekhuis	Duplex-, tussen-, verspringende woning, ééngezins-, rij-, hoek-, tussenwoning, drive-in woningen.
2-onder-1 kap	Geschakelde woningen, helft van dubbel
Vrijstaand	Woonboerderijen, individuele woningen, villa's
Appartement	bejaarden-, aanleunwoning, portiek-, flat (met en zonder lift), boven-, benedenwoning, maisonnettes, etagewoningen, studio, studentenwoningen,
Meerdere typen mogelijk	Recreatiewoning, bijzonder object, herenhuis, monumentaal pand, grachtenpand

5.4 VERGELIJKING MET WOZ-CLASSIFICATIES

In het kader van de wet Waardering Onroerende Zaken hebben alle gemeenten in Nederland onder andere alle woningen in hun gemeente moeten waarderen. Om dit te doen hebben de meeste gemeenten gebruik gemaakt van een uniforme soort-object-code, maar de classificatie gebeurt bij elke gemeente op hun eigen manier. Alle (250) Nederlandse gemeenten hebben dit zelf in beheer. De soort-object-

code voor woningen heeft vier posities [verslag over duwoz codes van het kadasterproject]:

- Het eerste cijfer geeft de hoofdsoort aan
- Het tweede cijfer geeft het soort gebruik aan
- Het derde cijfer geeft de vorm van het object aan
- Het laatste (vierde) geeft een nadere aanduiding aan

Alleen als het eerste cijfer een 1 is. Hebben we met woningen te maken. Om het woningtype te bepalen is het derde cijfer van belang. Hier volgen de indelingen van het derde teken, horende bij een 1 als eerste teken (en een tweede teken ongelijk aan 7 of 8).

1. Vrijstaand
2. Twee-onder-één kap
3. Rij (repeterende dakconstructie)
4. Hoek (laatste van een rij)
5. Tussenwoning (niet repeterende dakconstructie)
6. Eindwoning (laatste van een serie tussenwoningen)
7. Geschakelde woning
8. Etage
9. Woonwagen / woonboot

Tabel 5-5 Relatie tussen DUWOZ codes en de classificaties waar deze waarschijnlijk in terecht zullen komen

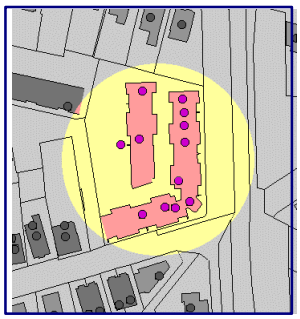
DUWOZ-code	Afstudeerclassificatie	
1*1*	V	Vrijstaand
1*2*	T	Twee-onder-één kap
1*3*	M	Middenwoning
1*4*	E	Eindwoning
1*5*	M	Middenwoning
1*6*	E	Eindwoning
1*7*	M	Middenwoning
1*8*	A	Appartement

Voordelen van de methode zoals in dit afstudeeronderzoek wordt voorgesteld, ten opzichte van het verzamelen van de WOZ-gegevens van alle Nederlandse gemeenten:

- Alle gegevens, die gebruikt worden, komen bij één instelling (het kadaster) vandaan.
- Een geografische locatie wordt aan het resultaat gekoppeld, zodat het mogelijk is ruimtelijke analyses uit te voeren.
- Er zijn mogelijkheden tot het visualiseren met behulp van kaarten.
- Het is een uniforme aanpak in de classificatie van woningen.

5.5 KENMERKEN VAN DE VERSCHILLENDE WONING TYPEN

Elk type woning heeft zo zijn eigen kenmerken in geometrie en op administratief gebied. De eisen die aan de deelgebouwen worden gesteld om te bepalen tot welk woningtype het behoort, zullen nu worden besproken. Alleen de kenmerken, die uiteindelijk gebruikt zijn in de classificatie, komen aan bod. In de beschrijvingen zal gesproken worden van deelgebouwen en hoofdgebouwen. Hoofdgebouwen zijn de gebouwen die gevormd worden door de lijnen met code hoofdgebouw uit het LKI. Deelgebouwen zijn de gebouwen die uit de versnijding van deze hoofdgebouwen met de percelen komen.

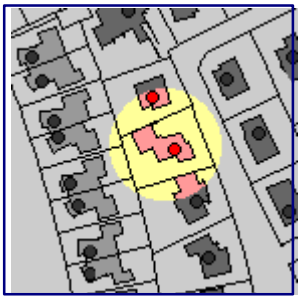


Appartementen, ideale case:

Administratieve kenmerken:

- De cultuurcode moet wonen als hoofdgebruik aangeven.
- Het geheel perceel (G), waartoe het gebouw en de adressen behoren, kan gekoppeld worden aan een administratief appartementsperceel (A). De relaties tussen deze administratieve objecten staat in paragraaf 3.2.2.

Vrijstaande woning, ideale case:



Administratieve kenmerken:

- De cultuurcode moet wonen als hoofdgebruik aangeven.
- Het aantal adressen op de bijbehorende adrescoördinaat is één.

Geometrische kenmerken:

- Geen buren.
- Het hoofdgebouw is opgedeeld in één deelgebouw.
- Onder het hoofdgebouw bevindt zich maar één perceel.
- Op het perceel staat maar één hoofdgebouw.
- Één adrescoördinaat.
- In het deelgebouw / hoofdgebouw staat één huisnummer.

Twee-onder-een-kap woning, ideale case:

Administratieve kenmerken:

- De cultuurcode moet wonen als hoofdgebruik aangeven.
- Het aantal adressen op de bijbehorende adrescoördinaat is één.

Geometrische kenmerken:

- Het aantal buren is één.
- Het hoofdgebouw is opgedeeld in twee deelgebouwen.
- Onder het hoofdgebouw bevinden zich twee percelen.
- Op het perceel staat maar één hoofdgebouw.
- Één adrescoördinaat in het deelgebouw.
- Twee adrescoördinaten in het hoofdgebouw.
- In het deelgebouw staat een huisnummer.

- In het hoofdgebouw staan twee huisnummers.

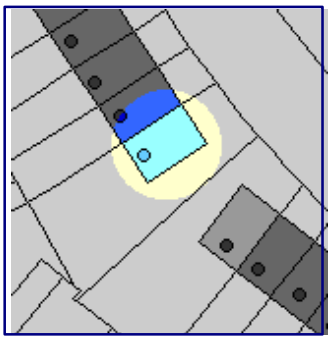
Rijtjeswoning, ideale case:

Administratieve kenmerken:

- De cultuurcode moet wonen als hoofdgebruik aangeven.
- Het aantal adressen op de bijbehorende adrescoördinaat is één.

Geometrische kenmerken:

- Het hoofdgebouw is opgedeeld in meer dan twee deelgebouwen.
- Het aantal buren is gelijk aan één of meer.
- Op het perceel staat maar één hoofdgebouw.
- Er bevindt zich maar één adrescoördinaat in het deelgebouw .
- In het gebouw staan twee huisnummers..



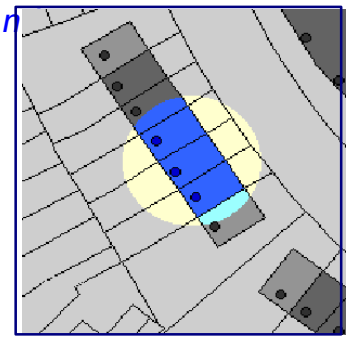
Onderschijding tussen ein en middenwoning:

Eindwoning:

- Het aantal buren is één.
- In het deelgebouw staat één huisnummer.

Middenwoning:

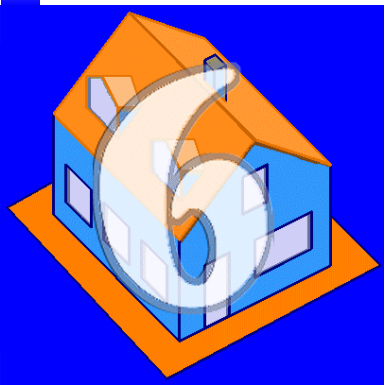
- Meer dan één buur.



Tot zo ver de ideale cases. Dit zijn de vijf woningtypen die we denken te kunnen onderscheiden en die door de opdrachtgever zijn vastgesteld. Niet alle woningen zullen aan al de bovengenoemde eisen voldoen. Er zijn meerdere redenen waarom woningen niet aan alle eisen voldoen. Een aantal redenen liggen in de bron bestanden. Hier zal wanneer de classificatieregels opgesteld worden, in hoofdstuk 8, verder op in gegaan worden. Een andere reden waarom niet alle woningen aan deze eisen voldoen, zit hem in het verschil tussen koop- en huurwoningen. De kenmerken zoals ze hier zijn opgesteld gelden in principe voor koopwoningen. Meestal is het zo dat er meerdere huurwoningen op een perceel staan. Daarom zijn de adrescoördinaten zo belangrijk. Ze worden onder andere gebruikt in de classificatie om een vrijstaand huis te onderscheiden van een rijtje huurhuizen. In een vrijstaand huis ligt één adres en in een rijtjes huis liggen er drie of meer. De adressen in het ACN-bestand (paragraaf 3.4) worden gebruikt als sleutel tot alle woningen.

OVERZICHT

VAN DE AFLEIDING & CLASSIFICATIE METHODE



Het gehele proces dat doorlopen is, wordt in een overzicht weergegeven. Met behulp van figuren, schema's en teksten zal de afleiding en classificatie van gebouwen uit kadastrale gegevens worden doorlopen.

De gehele automatische afleidings- en classificatie methode wordt per kadastrale gemeente uitgevoerd door een uitvoerend bestand. Dit bestand, dat bestaat uit een aantal commando's en aanroepen naar andere uitvoerende bestanden, geeft het gehele proces goed weer. Hoe de uiteindelijke implementatie en inhoud van deze andere uitvoerende bestanden is, is gedetailleerd in de volgende twee hoofdstukken terug te vinden.

6.1 INLEIDING

Het uiteindelijke programma bestaat uit een bestand waarin na elkaar een aantal verschillende commando's worden gegeven en andere uitvoerende bestanden worden aangeroepen. Vanaf nu zal dit bestand het hoofdprogramma genoemd worden. De aangeroepen uitvoerende bestanden zullen deelprogramma's genoemd worden. De belangrijkste deelprogramma's bevatten de geometrische versnijding en de classificatie. De achtergronden bij deze twee deelprogramma's worden beschreven in hoofdstuk 7, de geometrische versnijding, en in hoofdstuk 8 de classificatie. Dit hoofdstuk dient alleen om een overzicht te geven van het hoofdprogramma en het gehele proces.

6.2 HET HOOFDPROGRAMMA

Het hoofdprogramma heeft de naam "Do_it_all.sh" gekregen. Dit programma is een samenvoeging van een aantal andere programma's. De regels die in dit programma staan hebben in deze paragraaf een lichtblauwe kleur gekregen. Hieronder volgt een overzicht van het hoofdprogramma:

```
#!/bin/sh
set -v
```

Voorbereiding

make_acn.sh
: Aanmaken van de invoerbestanden die gebruikt zullen worden in een point-in-polygoon zoekalgoritme. Dit bevat de volgende elementen: een identificatienummer, x- en y-coördinaat en het aantal dat op deze coördinaat ligt.

make_akr_attributes.sh
: Een selectie wordt gemaakt van de gegevens die gebruikt zullen worden in de classificatie.

make_gebouwen.sh
: Aanmaken van het invoerbestand waaruit later de kaartlaag gebouwen kan worden gereconstrueerd. Deze gegevens zijn een selectie van de lijnen met een code voor hoofdgebouwen uit de LKI-tabel "Xfio_line"

make_parcels.sh
: Aanmaken van het invoerbestand waaruit later de kaartlaag gebouwen kan worden gereconstrueerd. Deze gegevens komen uit de LKI-tabel "Xfio_boundary"

make_text.sh
: Aanmaken van de invoerbestanden die gebruikt zullen worden in een point-in-polygoon zoekalgoritme. Alleen de punten die locaties zijn van huisnummers worden uit het lki-text bestand geselecteerd.

Versnijding

do_overlay.sh
: In het geval van het afstudeerproject zou dit inhouden de versnijding in CGAL, deze zal in meer detail besproken worden in hoofdstuk 7. Omdat dit programma wegens bugs niet werkend is wordt hier een versnijding in Arc/info uitgevoerd, zoals het geprogrammeerd is voor het Kadaster project.

Nabewerking

post_text.sh
: Point-in-polygoon zoekalgoritme gebruiken om te bepalen in welke deelgebouwen huisnummers vallen.

post_acn.sh
: Point-in-polygoon zoekalgoritme gebruiken om te bepalen in welke deelgebouwen en in welke percelen adrescoördinaten vallen.

dump_arc.sh
: Het berekende resultaat van de versnijding met Arc/Info wordt opgeslagen in een gedefinieerd resultaat.

Inladen

load_ingres.sh
: Ingelezen in de database van de attributen die berekend zijn in Arc/Info.

Attribuut berekening

update_acn.sh
: Uitrekenen van het aantal ACN dat in een deelgebouw, hoofdgebouw en perceel ligt.

update_huisnummers.sh
: Uitrekenen van het aantal huisnummers dat in een hoofdgebouw ligt, door alle huisnummers op te tellen die in de deelgebouwen liggen.

update_deelgebouwen.sh
: Uitrekenen van een aantal attributen.

update_gebouwen.sh
: Uitrekenen van het aantal deelgebouwen waaruit een gebouw bestaat.

update_percelen.sh
: Uitrekenen van een aantal attributen, zoals het totaal aantal ACN op een perceel.

Classificatie

all_classify.sh
: deze bestaat uit:

- Classify_deelgebouwen.sh
  De classificatie van de deelgebouwen zoals wordt besproken in hoofdstuk 8.
- Propagate_to_acn.sh
  Het doorkopiëren van de gevonden deelgebouwclassificaties naar de bijbehorende adressen.
- Classify_ACN.sh
  De classificatie van de adressen zoals wordt besproken in hoofdstuk 8.

Een groot gedeelte van de deelprogramma'a zijn terug te vinden in de bijlagen. De classificatieprogramma's zijn terug te vinden in Bijlage F. De update bestanden zijn terug te vinden in bijlage E.

6.3 OPDELING IN STAPPEN

Het gehele programma kan opgedeeld worden in een voorbereiding, een versnijding, een nabewerking, een attribuutberekening en de uiteindelijke classificatie. De voorbereiding gebeurt met behulp van met name SQL. Dit wordt gebruikt om de benodigde gegevens uit de database te halen en over te zetten naar een vorm die gebruikt kan worden in de versnijding. Daarna wordt de versnijding uitgerekend.

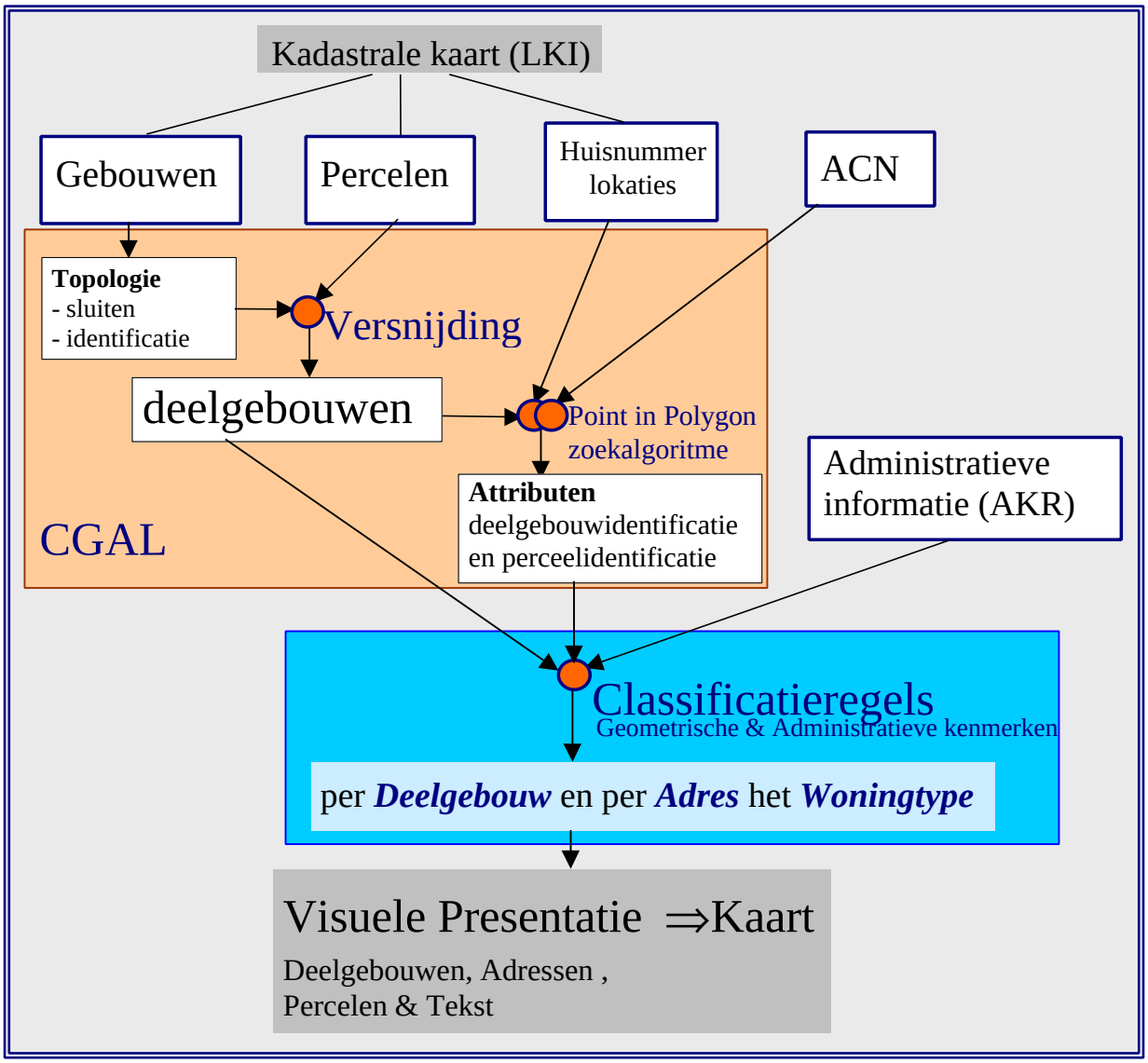
Uiteindelijk had dit met een geschreven programma moeten gebeuren, waarbij CGAL gebruikt zou worden, maar dat is niet gelukt. Daarvoor in de plaats is het programma gekomen dat voor het kadasterproject is gemaakt. Dit heeft ook gevolgen gehad voor de voorbereidingsfase. De benodigde gegevens zijn niet meer alleen naar een tekstbestand geschreven, maar ingelezen naar Arc/Info. In de versnijding wordt als extra de oppervlakte berekend van de deelgebouwen en worden het aantal buren geteld dat een deelgebouw heeft. Na de versnijding bestaat er een resultaten kaartlaag met voor elk vlak een unieke identificatie, de bijbehorende gebouwidentificatie (kaartlaag 1), de perceelidentificatie (kaartlaag 2), een oppervlakte en het aantal buren. De buren zijn de buurvlakken dat een gebouwidentificatie bevat.
In de nabewerkingfase wordt voor de adressen via een point-in-polygon zoekalgoritme bepaald in welk perceel en in welk deelgebouw dit adres valt. Voor de huisnummers wordt er bepaald in welk deelgebouw ze vallen. Dit laatste is alleen geprogrammeerd in het Kadaster proces. Ik was er nog niet aan toe gekomen om dit te programmeren voordat de bugs in de bibliotheek het mij verhinderde om het programma af te maken. Dit zou er in het CGAL-programma hetzelfde uitzien als de zoekoperatie voor adressen. De problemen met CGAL komen aan het einde van hoofdstuk 7 aan de orde.

Nadat de resultaten van de versnijding zijn ingeladen in de database, kunnen er nog een aantal attributen met behulp van SQL worden berekend. Dit stuk bevat het samenvatten van de adrescoördinaten. Voor alle deelgebouwen en gebouwen wordt het aantal adressen dat erin valt geteld. Ook worden nog een aantal andere attributen berekend. Dit staat beschreven in paragraaf 8.3.2. Hier na kan de classificatie worden berekend. Dit is beschreven in paragraaf 8.4.

De resultaten van de classificatie moesten natuurlijk bekeken worden. Daarom zijn de resultaten weer uitgeschreven naar Arc/Info. De shape-files zijn ingelezen in ArcView en samen met de percelen en teksten gevisualiseerd in een kaart. Delen van deze kaart zijn terug te vinden in hoofdstuk 9 en 10. Deze stap is ook weergegeven in schema 6-1. Dit is niet een doel van het afstudeerproject, maar het is wel nodig om een kwaliteitscontrole uit te voeren en om conclusies te kunnen trekken.

Het volgende schema geeft een overzicht van het gehele proces:

Figuur 6-1 Schema van het gehele proces dat is uitgevoerd in dit afstudeerproject.



GEPROGRAMMEERDE VERSNIJDING

De versnijding is geprogrammeerd in C++. Hierbij is gebruik gemaakt van de mogelijkheden die de Geometrische Algoritmen bibliotheek “CGAL” biedt. Het grootste doel van de versnijding is om deelgebouwen te bepalen. Het is niet gelukt om het programma uit te voeren met de werkelijke dataset. Soms lukte het wel om kleine eenvoudige voorbeelden door te rekenen. De redenen waardoor het programma nog niet werkt zijn:

- Er zitten nog te veel bugs in het programma.
- De gevonden bugs zitten niet in het zelf geschreven deel, maar in CGAL zelf.

Geschreven programma

Snijdende lijnen

Ondervonden problemen

7.1 INLEIDING

De aanwezige versnijding in CGAL voldeed niet om te gebruiken binnen dit afstudeerproject. Daarom is er geprobeerd deze aan te passen. Uiteindelijk is er een heel nieuw geometrische versnijdingsprogramma geschreven. In het nieuwe programma zijn ook de volgende deelstappen terug te vinden:

- Het berekenen van de snijpunten tussen de twee kaartlagen
- Het vormen van de nieuwe lijnstukken.
- Het berekenen van de vlak attributen door langs de grenzen te lopen.

Verder zijn een aantal veranderingen doorgevoerd. Deze aanpassingen van de geometrische versnijding, die volledig gericht zijn op het doel van het afstudeeronderzoek, worden in dit hoofdstuk besproken. Het geheel nieuw geschreven programma is terug te vinden in bijlage C1.

7.2 AANPASSINGEN TEN OPZICHTE VAN DE AL AANWEZIGE VERSNIJDING.

Als eerste is bijvoorbeeld een DCEL aangemaakt waarbij de lijnen tabel twee extra attributen mee krijgt: linkerobject identificatie en rechterobject identificatie. Verder krijgt ook de vlakken tabel een aantal attributen erbij.

DCEL met dubbele informatie

Deze bibliotheek bevat de definitie van een DCEL(zie paragraaf 2.1.4) De volle functionaliteit van een standaard DCEL is overgenomen door overerving en is uitgebreid. De grenzen en de vlakken kunnen extra informatie bevatten. Deze mogelijkheid is gebruikt om de links rechts informatie in de lijnen te stoppen. Verder wordt voor de vlakken één, twee of drie identificaties vastgelegd: een unieke identificatie en twee identificaties voor overerving uit de originele kaarten. Deze worden pas gevuld als de versnijding heeft plaats gevonden. Daarnaast bevatten de vlakken nog een vierde attribuut dat gebruikt wordt om te tellen hoe vaak er een ACN in dat vlak valt.

Verder bevat het programma in plaats van het berekenen van de intersectie punten een checked_insert. Omdat de lijnen van de gebouwen niet topologisch correct zijn, kunnen ze niet direct ingelezen worden in de kaartlaag maar moet dit een gecontroleerde vorm van invoegen worden. Voor het echte versnijden van de twee kaartlagen wordt de eerste kaartlaag gekopieerd en worden alle lijnen met behulp van een aangepaste versie, van de gecontroleerde vorm van invoegen, in deze kopie toegevoegd. Hierdoor hoeft niet elke versnijding twee keer worden berekend. Hoe dit komt wordt verderop in dit hoofdstuk besproken.

Als laatste aanpassing aan de versnijding wordt er voor de punten niet berekend welke attributen erbij horen. We zijn uiteindelijk toch alleen maar geïnteresseerd in de vlakken.

Extra toevoegingen aan het programma.

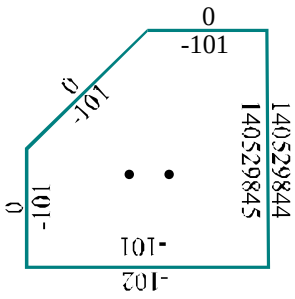
Naast een volledige geometrische versnijding met attribuutovererving zijn er nog een aantal stukken programma toegevoegd die nodig waren voor de afleiding en classificatiemethode:

- Een stukje topologieconstructie is toegevoegd.
- Er worden unieke identificaties voor alle gebouwen bepaald.
- Het inlezen en wegschrijven vanuit een tekstbestand is toegevoegd
- Bij het wegschrijven van de vlakken met een deelgebouwidentificatie worden het aantal buren en de oppervlakte berekend en weggeschreven.
- Ook wordt er een point-in-polygoon algoritme gebruikt om voor alle ACN te bepalen in welk versnijdingsvlak ze liggen, via dit vlak worden de attributen deelgebouwidentificatie en perceelidentificatie aan de adressen gekoppeld. Er zijn een aantal zoek algoritmen al in CGAL geprogrammeerd en één hiervan wordt gebruikt.

7.3 HET UITEINDELIJKE VERSNIJDINGSPROGRAMMA

Hierin zit alle functionaliteit om de uiteindelijke versnijding te berekenen tussen twee planar_map's met de aangepaste DCEL. Met lijnstukken wordt vanaf nu bedoeld een paar halfedges. Alle lijnstukken met hun links / rechtsinformatie van de eerste planar_map wordt gekopieerd naar de uiteindelijke planar_map. Daarna worden de lijnstukken van de tweede planar_map gecontroleerd op versnijdingen met alle lijnen uit de uiteindelijke planar_map. Als er een versnijding plaatsvindt worden er twee dingen uitgevoerd:

- De versneden lijn die al in de uiteindelijke kaartlaag aanwezig is wordt aangepast (gesplitst).
- En het snijpunt tussen beide lijnen komt in een puntenlijst.



Figuur 7-1 resultaat na de versnijding en voor het berekenen van de attributen.

Zodra de lijn uit de tweede planar_map gecontroleerd is op versnijdingen met alle lijnen uit de uiteindelijke planar_map worden de punten uit de puntenlijst gebruikt om het toe te voegen lijnstuk om te zetten in nieuwe lijnstukjes, welke tijdelijk bewaard worden in een lijnstukken lijst. De lijnstukken in deze laatste lijst worden, nadat alle lijnstukken uit de tweede kaartlaag zijn gecontroleerd, zonder problemen in de uiteindelijke kaartlaag toegevoegd. Continue wordt voor elk lijnstuk de links-, rechtsinformatie bijgehouden. Een vlak van deze planar_map ziet er dan uit als weergegeven in figuur 7-1

Daarna wordt aan de hand van de links-, rechtsinformatie van de grenzen van een vlak de perceelidentificatie en gebouwidentificatie bepaald. Als er

geen perceelidentificatie uit de grenzen gehaald kan worden, wordt er een punt van een van de grenslijnen gebruikt als zoekpunt in een Point-in-Polygon algoritme om alsnog de identificatie te vinden van het perceel waar het deelgebouw dan volledig in valt. De reden dat de links-, rechtsinformatie gebruikt wordt, en niet altijd een punt-in-polygoon-test, is omdat het bij point-in-polygoon zoekalgoritme moeilijk is om een punt binnen het polygoon te vinden. Ook de tweede stap kan rekenintensief zijn. Op deze manier wordt het vinden van een punt dat gebruikt zal moeten worden in het point-in-polygoon zoekalgoritme ook weer vergemakkelijkt, omdat gewoon een punt op de zijde van een deelgebouw gebruikt kan worden als zoekpunt.
Voor het goed slagen van de Versnijding op deze manier, zit als aanname dat *de grenzen van een gebouw nooit allemaal met de grenzen van percelen samen vallen*, zodat een polygoon in de eindkaart zijn grenzen altijd gedeeltelijk of helemaal uit de oorspronkelijke gebouwenkaart haalt en er dus altijd een grens is met links-, rechtsinformatie van de deelgebouwen. Mocht blijken dat er toch gebouwen zijn die geheel op de perceelsgrenzen vallen dan is het wel mogelijk om de gevormde Versnijding aan te passen, om wanneer er gemeenschappelijke lijnen zijn, deze lijnen te voorzien van dubbele links-, rechtsinformatie. Ik heb ervoor gekozen dit nog niet te doen omdat ik hier, in het begin, geen rekening mee heb gehouden. Ik verwacht niet dat het vaak voor zal komen. De meeste woningen hebben wel iets van een stuk stoep voor de woning, een buitenplaatsje, tuin of groenvoorziening, in geval van een flat, dat ook tot het perceel behoort.
Het resultaat van deze Versnijding is nu een planar_map die alle informatie van de eerste kaartlaag bevat, de versnijdingspunten van de eerste kaartlaag met de tweede kaartlaag, nieuwe lijnstukken die afkomstig zijn uit de tweede kaartlaag en attribuut informatie uit beide kaartlagen. In de Boolean tabel is dit (A and B). Hiervoor wordt de term “intersection” ook wel gebruikt.

7.4 DE STAPPEN IN HET VERSNIJDINGSPROGRAMMA.

De volgende stappen worden uitgevoerd:

- Inlezen van gebouwgegevens in een Planar_map (zie hoofdstuk 2.1.5). Er wordt gelijk op topologie gecontroleerd, hierbij gaat het om dubbele of versnijdende lijnen. Dit gebeurt door de lijnen niet direct in de Planar_map toe te voegen maar door gebruik te maken van een stukje programma “Checked_Insert” waarin gekeken wordt of het nieuw in te voegen lijnstukje niet de al aanwezige lijnstukken snijd. Dit is net zoiets als bij het Rtree-algoritme.
- Aanpassen van de topologie van de Planar_map “gebouw” door vlakken te sluiten, vlakken te nummeren en links-, rechtsinformatie aan de lijnen te hangen. De nummering van deze vlakken is negatief,

zodat direct duidelijk is of we met een gebouwidentificatienummer of met een perceel identificatienummer te maken hebben.

- Inlezen van perceelgegevens in een Planar_map. Deze gegevens zijn in principe al topologisch correct en de links / rechts informatie wordt gelijk opgeslagen in de Planar_map “Perceel”.
- Het berekenen van de Versnijding met behulp van een aangepaste Checked_Insert. Het grootste verschil met de gewone Checked_insert is dat de gevonden lijnstukken van de tweede kaartlaag (percelen) eerst in een lijst komen te staan, want ze hoeven niet met andere lijnen uit de tweede kaartlaag gecontroleerd worden. Pas als alle lijnstukken uit de tweede kaartlaag gecontroleerd zijn worden ze zonder problemen toegevoegd aan de uiteindelijke kaartlaag. Hieruit komt de Planar_map “Deelgebouw” waarvan de lijnen verschillende links, rechts informatie bevatten.
- Inlezen van de ACN per regel en het zoeken van het bijbehorende perceel en deelgebouwvlak. Dit zoeken gebeurt met een in CGAL aanwezig point-in-polygon zoekalgoritme. De coördinaten van het ACN-punt, samen met de identificaties van het gevonden perceelvlak en deelgebouw vlak worden weggeschreven in een nieuw tekstbestand (“adres.txt”). Verder kan voor elk perceel en deelgebouwvlak bijgehouden worden hoe vaak hij gevonden wordt. Op deze manier wordt er geteld hoeveel ACN er in deze vlakken ligt. Dit is nog niet uitgevoerd.
- Het wegschrijven van “perceel” en “Deelgebouw” naar vier tekstbestanden. Omdat we deze bestanden uiteindelijk willen inlezen in ArcView in een shape-file wordt de vorm van de attributen gescheiden.. Deze worden later weer samen gevoegd, samen met de rest van de Administratieve gegevens. Met behulp van een “AVENUE” script wordt de tekstfile met de geometrie ingelezen in ArcView en omgezet naar een shape-file (bijlage C2). Het samenvoegen gebeurt ook met een “AVENUE” script dat een vereniging (join) uitvoert tussen twee tabellen (bijlage C3). Gevormd worden de bestanden “perceel.txt” en “deelgebouw.txt” welke een ID en de vorm van de polygonen bevat. Daarnaast worden de bestanden “perceelattributen.txt” en “deelgebouwattributen.txt” gevormd. Deze laatste twee worden weer ingelezen in de database om mee te doen aan de classificatie met behulp van SQL-statements.

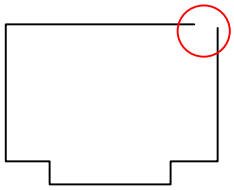
Bij het inlezen van de lijnen die uiteindelijk de gebouwen moeten vormen moet er gelet worden op versnijdingen met andere lijnen uit hetzelfde bestand wil het lukken om de lijnen in een planar_map te krijgen. Hoe dit in zijn werk gaat wordt in de volgende paragraaf beschreven.

7.5 TOPOLOGIECONSTRUCTIE

Bij de topologieconstructie worden alle lijnstukken met behulp van de informatie uit de DCEL langsgelopen en wordt er steeds gecontroleerd of

het volgende lijnstuk dezelfde is als het tweelinglijnstuk. Wanneer dat het geval is hebben we te maken met een zogenoemde “dangling edge” ofwel los uiteinde. Elke keer dat er zo een los uiteinde gevonden wordt, wordt er gekeken of de lengte hiervan niet te klein is. Wanneer dit het geval is hebben we te maken met een overschot. Daarna wordt er eventueel gekeken of het vorig gevonden losse eindstuk van een reeks, dichtbij genoeg is om deze alsnog te verbinden. Met een reeks wordt bedoeld alle lijnen die met elkaar via knooppunten verbonden zijn en samen een losstaand deel (eiland) in het unbounded face vormen. Over het algemeen is dit natuurlijk steeds één gebouw. Deze vorm van het omgaan met losse uiteinde is gebaseerd op de volgende aannames:

- *De meest voor de hand liggende bij elkaar horende uiteinden behoren tot dezelfde reeks.*
- *De meeste gaten ontstaan door het net niet sluiten van de lijnstukken, waardoor een gebouw eruit ziet zoals op figuur 7-2.*

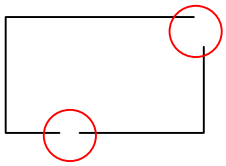


Figuur 7-2 Meest voorkomende topologie fout die gecorrigeerd moet worden.

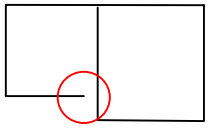
Het op deze manier sluiten van polygonen die gebouwen vormen gaat goed zolang er zich niet meer dan één gat in het (nu nog fictieve) polygoon bevindt en er niet nog meer losse uiteinden zijn.

De volgende situaties worden op deze manier niet opgevangen. Deze zouden nog geprogrammeerd moeten worden:

- Meerdere gaten in de lijnen rond een gebouw (figuur 7-3a). Dit zou kunnen door alle dangling edges die geen overschot zijn (andere eindpunt meer dan 2 uitgaande lijnen) in een lijst te zetten en met elkaar te vergelijken
- “Undershoots”, dit zijn lijnen die te kort zijn en die dus niet bij een andere dangling edge horen figuur 7-3b. Vaak moet het punt waar deze op aan moet sluiten nog berekend worden.



Figuur 7-3a Hier gaat het nu nog fout



Figuur 7-3b Undershoot

Voor het toekennen van de identificatienummers wordt de aanname gedaan dat *alle gevormde vlakken gebouwen zijn*. In principe ziet de kaartlaag van de gebouwen eruit als een groot leeg vlak met allemaal eilandjes. Er wordt op dit moment geen rekening gehouden met patio’s, maar mocht dit nodig zijn dan is dat te programmeren door voor alle vlakken te kijken of het eilanden bevat.

7.6 VERSNIJDINGEN VAN LIJNEN

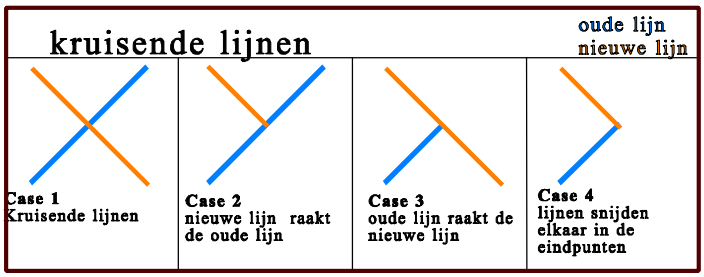
Zowel bij het inlezen van de gebouwlijnen als bij de uitvoer van de Versnijding komen er intersecties tussen lijnen voor. Alle lijnen worden steeds met elkaar vergeleken. Er kunnen dan 14 mogelijkheden optreden. De meest eenvoudige van deze twee is natuurlijk dat ze elkaar totaal niet snijden. Ze kunnen dan beide in de planar_map worden ingelezen. De andere mogelijkheden, en wat er dan moet gebeuren met de wederzijdse

lijnstukken, zal nu worden besproken in twee delen. Het eerste deel zijn kruisende lijnen en het tweede deel zijn de parallelle lijnen. Bij het berekenen van versnijdingen tussen segmenten (lijnstukken) binnen CGAL wordt er een object teruggegeven dat of een punt of een segment (lijnstuk) bevat. De figuren 7-4 en 7-5 zijn gebaseerd op figuur 4 uit [O94], maar dan aangepast op de objecten (point en segment) die binnen CGAL worden gedefinieerd. Alleen parallelle lijnen leveren een segment als resultaat.

7.6.1 VERSNIJDING IS EEN PUNT

Ook een gezamenlijk eindpunt wordt binnen CGAL gezien als een versnijdingspunt. Hierom is het nodig om de volgende cases te onderscheiden en dus met behulp van voorwaarden programmeren Wanneer we te maken hebben met kruisende lijnen is de versnijding een punt. Er zijn vier verschillende mogelijkheden.

Figuur 7-4 versnijdingsresultaat is een punt



Case 1

Wanneer de twee lijnen elkaar werkelijk snijden moeten beide lijnen in tweeën worden gesplitst. De oude lijn moet weggegooid worden en vervangen worden door zijn twee delen, en de nieuwe lijn moet in twee delen in de planar_map worden toegevoegd. Deze case onderscheidt zich van de volgende drie doordat het gevonden punt geen eindpunt van zowel de oude als de nieuwe lijn is.

Case 2

Wanneer de nieuwe lijn een lijn uit de kaart alleen maar raakt ergens in het binnenste van deze lijn, hebben we te maken met een van de drie mogelijkheden van rakende lijnen. De oude lijn moet weer vervangen worden door twee nieuwe delen en de nieuwe lijn kan in zijn geheel worden toegevoegd in de kaart. Deze case onderscheidt zich doordat het gevonden punt een eindpunt is van de nieuwe lijn.

Case 3

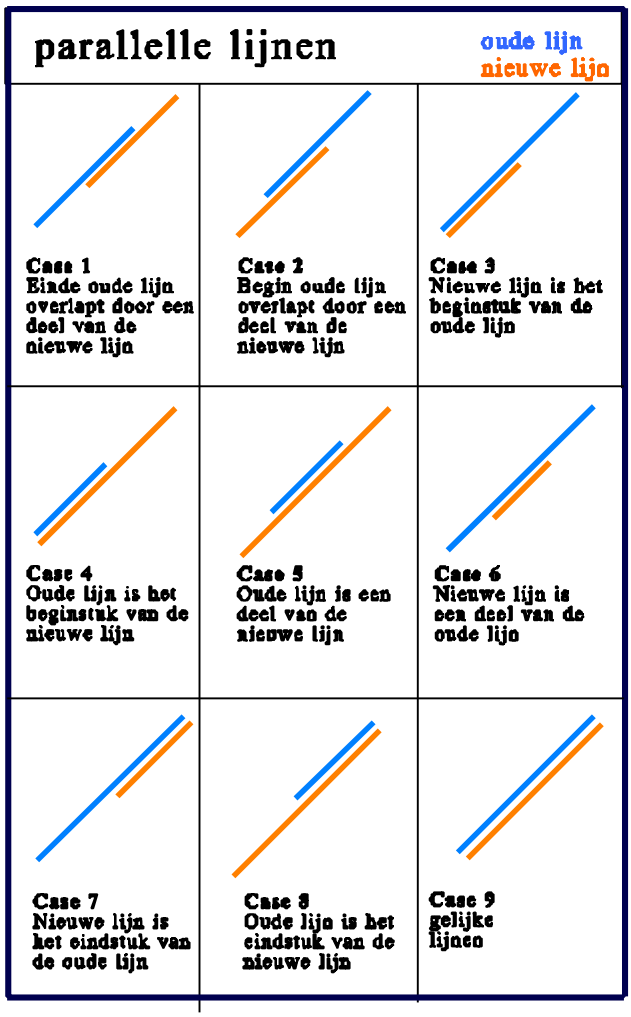
Een bestaande lijn raakt de nieuw toe te voegen lijn. In dit geval moet de nieuwe lijn in twee stukken worden toegevoegd aan de kaartlaag. Deze case onderscheidt zich doordat het gevonden punt een eindpunt is van de lijn uit de kaartlaag.

Case 4

Dit is in de gebruikte kaartlagen voor dit afstudeerproject, waarschijnlijk de meest voorkomende vorm van versnijding, twee uiteinden komen samen in één punt. De nieuwe lijn kan gewoon toegevoegd worden. Deze case onderscheidt zich doordat het eindpunt van de nieuwe lijn samenvalt met het eindpunt van de oude lijn.

7.6.2 VERSNIJDING IS EEN LIJNSTUK

Het is ook heel goed mogelijk dat de te versnijden lijnen elkaar gedeeltelijk overlappen. Dit komt omdat de lijnen dan parallel lopen. CGAL geeft dan voor het versnijdingsresultaat een segment terug. In deze situatie zijn er negen mogelijkheden, deze zijn van elkaar te onderscheiden door rekening te houden met de begin- en eindpunten van de originele lijnen en het geretourneerde segment. Het segment is dat deel dat beide lijnen bevat.



Figuur 7.5 Versnijding van Parallelle lijnen kunnen één van de volgende segmenten retourneren.

Er moet een keuze gemaakt worden vanaf welke lijn het gezamenlijk deel komt dat opnieuw ingevoegd gaat worden. Als je alleen met lijnen zonder attributen te maken hebt, is dat niet belangrijk en kies je voor degene die het makkelijkst geprogrammeerd kan worden, in mijn geval is dat het nieuwe lijnstukje wanneer we bezig zijn met het inlezen van de

gebouwen. Maar zijn het lijnen met attributen, dan is de keuze van de lijn en de richting van de lijn belangrijk. Binnen het berekenen van de Versnijding bevatten de lijnstukken links / rechts informatie. Nu is het dus belangrijk dat het gezamenlijk stuk of de informatie van allebei of van de meest belangrijke krijgt. Binnen dit project is er in eerste instantie voor gekozen om de informatie van de percelen mee te nemen. Dit vanwege de aanname dat niet alle grenzen van een gebouw op de perceelgrenzen vallen en er dus altijd wel een ander lijnstukje om het polygoon zit die de juiste identificatie voor het vlak kan leveren. Het is mogelijk om te programmeren dat allebei worden meegenomen, dat zou een hoop ellende kunnen voorkomen. Het beginpunt is gedefinieerd als het meest linkse punt (of laagste in het geval van verticale lijnen) en het eindpunt is het meest rechtse punt (of hoogste in het geval van verticale lijnen).

Case 1

Het begin van het segment valt samen met het begin van de nieuwe lijn, en het einde van het segment valt samen met het einde van een oude lijn. Zowel de oude als de nieuwe lijn zullen aangepast moeten worden. De oude lijn moet vervangen worden door het stuk van begin tot waar het gezamenlijk deel begint. Het gezamenlijk deel moet als een lijnstuk worden toegevoegd, en van het nieuwe stuk moet alleen dat deel, dat begint bij het einde van het gezamenlijk deel tot het einde van het lijnstuk, worden toegevoegd.

Case 2

Het begin van het segment valt samen met het begin van de oude lijn, en het einde van het segment valt samen met het einde van de nieuwe lijn. Zowel de oude als de nieuwe lijn zullen aangepast moeten worden. De oude lijn moet vervangen worden door het stuk van het einde van het segment tot het einde van het oude lijnstuk. Het gezamenlijk deel moet als een lijnstuk worden toegevoegd, en van het nieuwe stuk moet alleen van het begin tot het begin van het segment worden toegevoegd.

Case 3, 4, 7 en 8

Een van de twee lijnstukken gaat helemaal op in de ander en ze hebben of een gezamenlijk begin (3, 4) of een gezamenlijk eindpunt (7,8).

Case 5 en 6

Een van de twee lijnstukken gaat helemaal op in de ander en ze hebben niet een zelfde begin of eindpunt. Een van de lijnen is hetzelfde als het geretourneerde segment. In 5 is dat de oude lijn in 6 is dat de nieuwe lijn.

Case 9

De lijnstukken zijn helemaal hetzelfde, en tevens ook hetzelfde als het geretourneerde segment. In dit geval kan het nieuwe lijnstuk gewoon vergeten worden.

In Box 7-1 is het schema weergegeven dat gebruikt is om de verschillende situaties voor de segmenten terug te vinden en de juiste opdelingen van de oorspronkelijke lijnstukken te kunnen uitvoeren. Het is mogelijk om een ander schema te maken om de verschillende situaties terug te vinden. Hierin zal de volgorde van de voorwaarden omgewisseld zijn. Maar deze zal er ongeveer hetzelfde uitzien.

Tekst Box 7-1 Algoritme / voorwaarde volgorde die gebruikt is om de verschillende segment situaties te kunnen behandelen.

Het begin van het segment valt samen met het begin van het oude lijnstuk → Cases 2,3,4,5,8,9

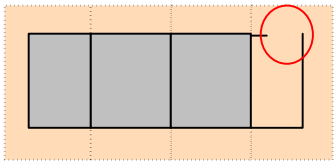
- Het begin van het segment valt ook nog samen met het begin van het nieuwe lijn segment → Cases 3,4,9
  - Het eind van het segment valt samen met het einde van het oude lijnstuk → Cases 4,9
    - Het eind van het segment valt samen met het einde van het nieuwelijnstuk → ***Case 9*** →weggooien oude lijn en invoegen nieuwe lijn
    - Anders → ***Case 4*** →weggooien oude lijn en invoegen nieuwe lijn in twee delen.
  - Het eind van het segment valt samen met het einde van het nieuwelijnstuk → ***Case 3*** →weggooien oude lijn , vervangen door het eindstuk van de oudelijn en invoegen nieuwe lijn.
- Het begin van het segment valt *niet* samen met het begin van het nieuwe lijn segment → Cases 2,8,5
  - Het eind van het segment valt samen met het eind van het nieuwe lijnstuk → Cases 2,8
    - Het eind van het segment valt samen met het eind van het oude lijnstuk → ***Case 2*** →weggooien oude lijn, vervangen door het eindstuk van de oude lijn en invoegen nieuwe lijn in twee delen.
    - Anders → ***Case 8*** →weggooien oude lijn en invoegen nieuwe lijn in twee delen.
  - Anders → ***Case 5*** →weggooien oude lijn en invoegen nieuwe lijn in drie delen.

Anders 1,6,7

- Het eind van het segment valt samen met het eind van de oude lijn.→ Cases 1,7
  - Het eind van het segment valt samen met het eind van de nieuwe lijn → ***Case 7***→weggooien oude lijn, vervangen door het begin van de oude lijn en invoegen nieuwe lijn
  - Anders → ***Case 1***→weggooien oude lijn, vervangen door het begin van de oude lijn en invoegen nieuwe lijn in twee delen.
- Anders → ***Case 6***→weggooien oude lijn, vervangen door het begin en het einde van de oude lijn plus het invoegen nieuwe lijn.

Voordelen van deze Versnijding is dat alle lijnen in de kaartlaag ook allemaal meedoen, ook wanneer we geen gesloten vlakken hebben zoals in figuur 7-6. Deze lijnen gaan wel de Versnijding in. Lijnen zoals deze hebben tijdens de identificatie een code –2 als links rechts informatie gekregen. Dit betekent dat het lijnen zijn die zowel aan de ene kant als aan de andere kant het “unbounded face” bevatten.
Aan het einde van de topologieconstructie hebben vlakken als identificatie voor de gebouwen of een 0 (unbounded- face) of een –2 (losse lijnen die niet gesloten zijn omdat het gat te groot is.) of –101 en lager. Deze codes worden doorgekopieerd naar de deelgebouwen. Deelgebouwen met een code –2 of –101 en lager worden weggeschreven naar het eindbestand.

Opmerkingen bij figuur 7-6

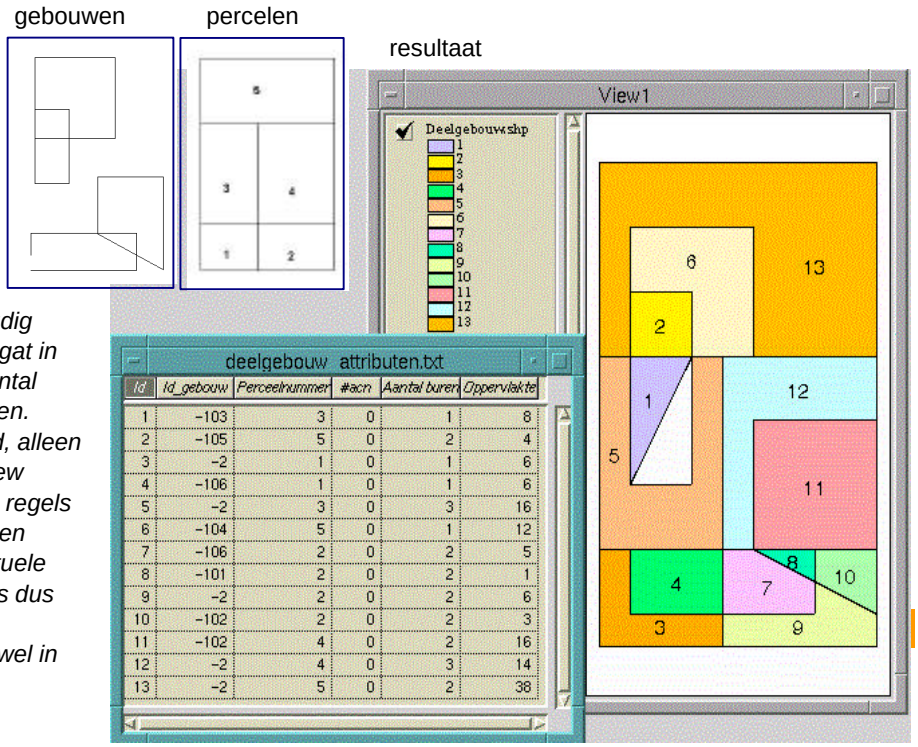


Figuur 7-6 niet gesloten lijn met uiteindelijk gevormde vlakken.

Alle vlakken in dit voorbeeld hebben de code -2. Binnen het laatste oranje vlak is het laatste deel van de losse lijn overgebleven. De losse uiteinden kunnen samen met het bijbehorende perceelvlak naar een apart bestandje weggeschreven, zodat het eventueel later mocht men dit willen met de hand kan worden bijgewerkt. Het onderscheidt tussen wat een deelgebouw is en wat alleen een stuk perceel is kan niet worden gemaakt. Het is misschien een optie om voor het perceel het aantal keren dat versnijdingscase 1 van de kruisende lijnen voorkomt te tellen. Dit zou kunnen door elke keer dat een perceelgrens versneden wordt een teller 1 op te hogen. Hebben we te maken met vier versnijdingen dan staat er op het perceel een middenwoning. Zijn het er maar twee of minder dan kan het of een eindwoning of een twee-onder-één-kap zijn. Deze optie is verder niet meer uitgewerkt.

7.7 PROBLEMEN MET CGAL

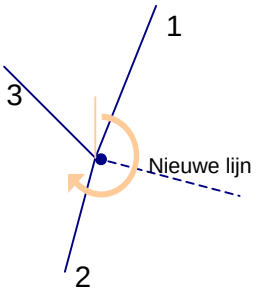
Het geschreven programma is uitgevoerd met een aantal zeer eenvoudige proefsetjes, waarbij de coördinaten hele getallen waren tussen 0 en 20. Een van die resultaten is terug te vinden in figuur 7-7. Maar de bestanden die gebruikt moeten gaan worden zijn natuurlijk veel



Figuur 7-7 Een eenvoudig voorbeeld, waarin een gat in de lijnen zat en een aantal versnijdingen voorkomen. Deze is juist uitgevoerd, alleen bij het inlezen in ArcView werden de eerste twee regels als tekst regels ingelezen (kolomnamen en eventuele commentaarregel) en is dus het allereerste punt weggevallen. Deze zit wel in het resultaat van de versnijding.

ingewikkelder.
De coördinaten van een punt in LKI ziet er als volgt uit: (147000000, 530000000). Dit lijken te grote waarden te zijn om mee te werken in CGAL als integers. Omdat we voor de berekeneing van de intersectiepunten hetzelfde aantal decimalen willen stelt dit ook weer eisen aan de afronding wanneer we gebruikmaken van reals. De ideale keuze waarin de coördinaten gegoten moeten worden is nog niet gekozen. CGAL heeft standaard de volgende twee vormen waarin coördinaten kunnen worden weergegeven: Cartesian en homogeneous. (1,3333 0,6666) in cartesian coördinaten is hetzelfde als (4 2 3) in homogeneous coördinaten. Homogeneous coördinaten bevatten drie getallen, waarbij de laatste een gemeenschappelijke deler is. Door afronding kan een punt dat twee keer berekend wordt, en eigenlijk hetzelfde moet zijn, toch kleine verschillen vertonen. Het is mogelijk om de types integer en real uit te breiden met andere typen getallen. Dit kan bijvoorbeeld door LEDA toe te voegen aan CGAL en het type LEDA-rational te gebruiken. Dit type zou preciezer moeten werken (meer decimalen). Maar dit type had tot gevolg dat het gebruiken van lijsten weer niet mogelijk was, dat was nog niet geprogrammeerd.

Nadat het programma gecompileerd is kan het uitgevoerd worden. Vooral bij het uitvoeren van het programma kwamen er een aantal problemen naar voren. Er zijn drie problemen naar voren gekomen, waarvan het maar gelukt is om er één op te lossen.



Figuur 7-8 de nieuw toe te voegen lijn hoort tussen 1 en 2 in

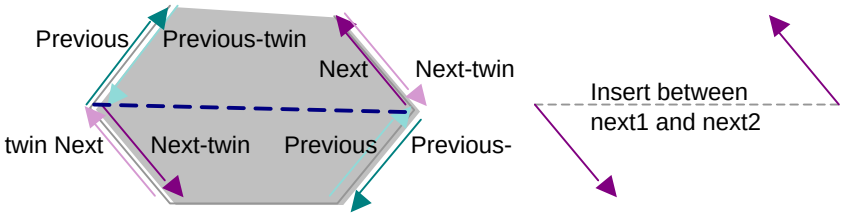
Foutmelding “precondition violation”

CGAL werkt veel met voorwaarden. Elke stukje programma waarin iets berekend wordt gebruikt pre condities. De pre conditie waar schijnbaar niet aan voldaan werd is: “face-handle first = face-handle second” Er is veel tijd gaan zitten in het achterhalen waar in dit geval de fout zat. Als er een nieuw lijnstuk in de kaartlaag (Planar_map) toegevoegd wordt moet er berekend worden waar deze in de DCEL en de Trapezoidal structuur terecht komt. Om dat te doen wordt er, voor zowel het begin als het eindpunt, gekeken tussen welke twee uitgaande lijnen van dat bepaalde punt het nieuw in te voegen lijnstuk zich bevind. (zie figuur 7-8). Dit gebeurt door alle uitgaande lijnen van een punt rond te lopen met het volgende stuk programma.

```
Previous=1, next=2
While NOT nieuwe lijn tussen previous en next
      Previous=next
      Next = volgende uitgaande lijn (3)
```

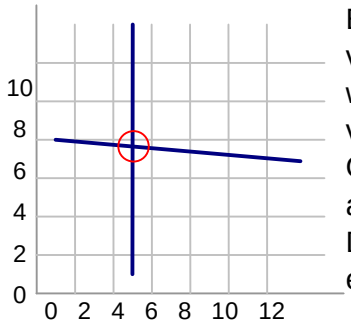
Het resultaat van deze loop, de gevonden volgende lijn, wordt gebruikt om de lijn werkelijk toe te voegen in een bestaand vlak. Beide lijnstukken die als next gevonden zijn worden gebruikt om de nieuwe lijn toe te voegen. Deze moeten wel altijd naar hetzelfde vlak wijzen (figuur 7-9). Dat is de

pre-conditie die soms fout gaat. Op een of andere manier is het gebruik



Figuur 7-9 Invoegen van een nieuwe lijn in een vlak.

van next niet altijd de goede. Daarom heb ik een extra controle ingebouwd, zodat wanneer de preconditie niet geldt gekeken word of in plaats van next niet zijn tweeling gebruikt kan worden.



Figuur 7-10 versnijding tussen twee lijnen.

Een andere oorzaak van de bovengenoemde foutmelding was het missen van intersectiepunten. Het volgende programma (tekstbox 7-2) hebben we opgestuurd naar de CGAL beheerders. Het is een programma dat de versnijding van twee gegeven lijnen berekend (figuur 7-10).
Op de vraag of de twee punten elkaar sneden, gaf het programma het antwoord “false”, terwijl dit toch duidelijk het geval was.
De CGAL-beheerders waren het met ons eens dat hier een probleem zat en dat deze in het stukje bibliotheek “segment_to_segment _intersection.h” zat.

Tekstbox 7-2 Eenvoudig programma dat de versnijding van twee lijnen controleert

```
#include <iostream>
#include <fstream.h>
#include <stdlib.h>

#include <CGAL/basic.h>
#include <CGAL/Cartesian.h>
#include <CGAL/Pm_segment_exact_traits.h>
#include <CGAL/Pm_default_dcel.h>
#include <CGAL/Segment_2_Segment_2_intersection.h>

typedef double Basetype;
typedef CGAL::Cartesian<Basetype> Rep_class;
typedef CGAL::Pm_segment_exact_traits<Rep_class>
Pmtraits;

typedef Pmtraits::Point Point;
typedef Pmtraits::X_curve Curve;

int main()
{
   Curve first(Point(5,1),Point(5,14));
   Curve second(Point(1,8),Point(13.41,7.04));

   cout << "first: " << first << "\n";
   cout << "second: " << second << "\n";

    if (do_intersect(first,second))
      cout << "true\n";
   else
      cout << "false\n";
}
```

Zij hebben ons toen de nieuwe versie van deze bibliotheek opgestuurd. Met dit nieuw stukje bibliotheek werkte het programma zoals het in textbox 7-2 geschreven had naar behoren (nu kregen we antwoord “true”), maar bleef het intersectiepunt nog steeds leeg (niet uitgerekend).

Foutmelding: “Segmentation fault”

Dit is een fout die met de planar_map te maken heeft en die bij het opbouwen van de kaartlaag op het beeldscherm komt. Meerdere mensen die CGAL en de planarmap gebruiken schijnen dit probleem te hebben opgemerkt. Volgens de CGAL beheerders ligt dit probleem aan de mogelijkheden en beperkingen van het gekozen cijfertype. Voorgesteld is om een work-around te gebruiken en “CGAL_NO_LEDA_HANDLE” te definiëren. Dit is nog niet geprobeerd.

Een laatste opmerking over CGAL:
Een klein vraagteken moet geplaatst worden bij de robuustheid van CGAL! Ook met de allernieuwste versie van CGAL (2.2) werkt het programma nog steeds niet zoals we willen.

Doordat er zoveel problemen ontstonden en er veel tijd in ging zitten om de gevonden fouten in CGAL zelf te lokaliseren en eventueel op te lossen, ben ik gestopt om een volledig werkend programma te maken. Voor het vervolg van het afstuderen, de classificatie, is er gebruik gemaakt van het resultaat van de versnijdingsberekening die is uitgevoerd voor het Kadasterproject. Deze versnijding is berekend in Arc/Info.

CLASSIFICATIE PROCES

Voordat er geclassificeerd kan worden moeten eerst de juiste tabellen met de juiste attributen worden verkregen. De tabellen deelgebouwen, gebouwen, huisnummers en ACN moeten gevuld worden met de resultaten van de versnijding. Voor deze tabellen en de tabel percelen moet daarna nog een aantal attributen worden berekend. Daarna kan pas de classificatie met behulp van beslisregels worden uitgevoerd. De classificatie gebeurt in twee delen. Eerst worden de deelgebouwen geclassificeerd en daarna alle bekende adressen. Al deze stappen zullen voornamelijk met behulp van programma's geschreven in Structured Query Language (SQL), worden uitgevoerd.

In en uitvoer van databasegegevens

Aanpassen van tabellen

Uitvoeren van de classificatie met beslisregels

Resultaten

8.1 INLEIDING

In dit hoofdstuk worden de volgende stappen besproken:

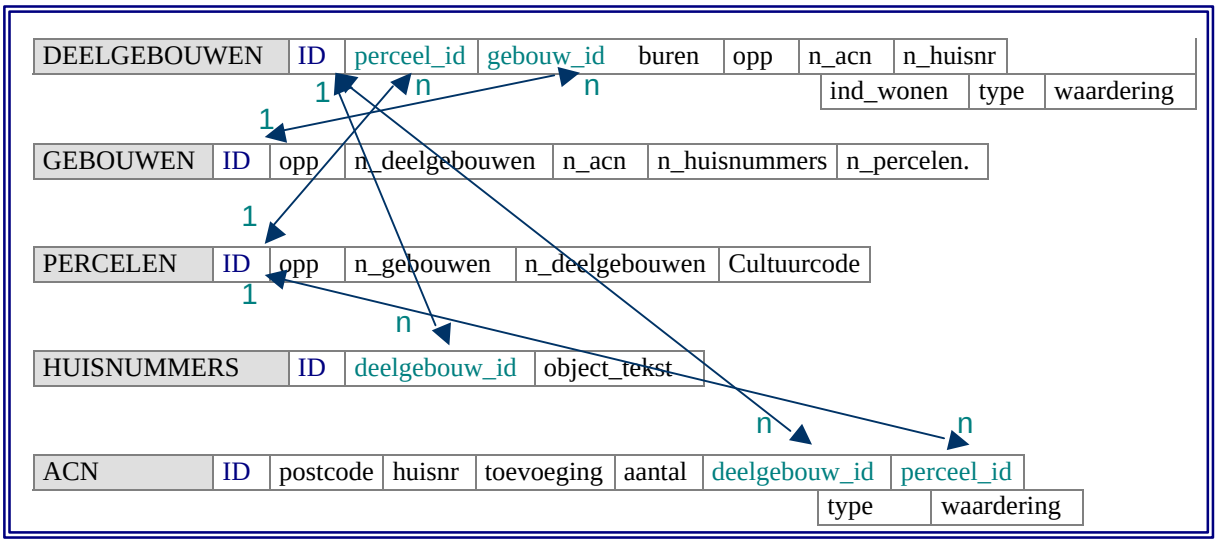
- Het inlezen en wegschrijven van bestanden van en naar de database
- Het vervaardigen van tabellen met alle gegevens die gebruikt worden voor de classificatie met beslisregels. Voor het vervaardigen van de tabellen zullen er selecties op en verenigingen van tabellen worden gemaakt.
- Het uitvoeren van beslisregels om de laatste twee kolommen van de tabel deelgebouwen te vullen. De laatste twee kolommen zijn het type en een waardering.
- Daarna zullen ook de attributen type en waardering in de tabel ACN worden gevuld, zodat er een classificatie per adres kan worden verkregen.

Hier volgt een overzicht van de tabellen die in de database aangemaakt zullen worden, zoals ze er aan het eind uit zullen zien

Tabel	Kolomen
Deelgebouwen	Identificatie, perceelidentificatie, gebouwidentificatie, buren, oppervlakte, n_acn, n_huisnummers, indicatie_wonen, woningtype, waardering
Gebouwen	identificatie, oppervlakte, n_deelgebouwen, n_acn, n_huisnummers, n_percelen.
Percelen	identificatie (G_akrobjectnummer), oppervlakte, n_gebouwen, n_deelgebouwen, Cultuurcode
Huisnummers	identificatie, deelgebouwidentificatie, object_tekst
ACN	identificatie, postcode, huisnummer, huisnummertoevoeging, aantal, deelgebouwidentificatie, perceelidentificatie, woningtype, waardering

Tabel 8-1, overzicht van, voor de classificatie uiteindelijk gebruikte, tabellen en attributen (primaire sleutels, externe sleutels)

Figuur 8-1 Relaties tussen de verschillende tabellen, alle zijn 1:n.



8.2 AANMAKEN VAN DE TABELLEN NODIG VOOR DE CLASSIFICATIE

Het aanmaken van de tabel met de deelgebouwen en het bepalen van bijbehorende attributen gebeurt in een aantal stappen. Als eerste hebben we al de tabel met deelgebouwen en berekende attributen uit de versnijding in tekstformaat. Deze moet in de database worden ingelezen en er moeten nog een heleboel administratieve attributen berekend worden. Voor de classificatie zullen de tabellen deelgebouwen, ACN, perceel, gebouw en huisnummers gebruikt worden. Ook voor deze tabellen zullen er gegevens in de database moeten worden ingelezen.

8.2.1 IN EN UITVOER TUSSEN TEKSTBESTANDEN EN DE DATABASE

Gedurende het gehele afleidings- en classificatieproces wordt er twee keer uit de database naar een tekstbestand en een keer vanuit een tekstbestand naar de database geschreven. Eerst moeten de juiste gegevens die voor de versnijdingsoperatie worden gebruikt uitgeschreven worden. Daarna moeten de resultaten van de versnijding weer ingelezen worden en uiteindelijk moeten de resultaten van de classificatie naar een bestand geschreven worden om uiteindelijk gevisualiseerd te kunnen worden. Dit laatste is nodig omdat ervoor gekozen is om ArcView te gebruiken om kaarten te maken van de resultaten en dit programma bekend is bij de afstudeerster. Het is in theorie mogelijk om de resultaten ook vanuit de database te bekijken, maar dan moeten ook de geografische gegevens die berekend zijn in de versnijding ingelezen worden in de database in een record die ook ruimtelijke gegevens kan opslaan zodat ze geschikt is om te visualiseren. Binnen dit afstudeerproject is echter geen tijd gereserveerd om uit te vinden hoe dat in zijn werk gaat.

De uitvoer naar een tekstbestand dat gebruikt zal worden in de versnijding gebeurt met een uitvoerend bestand. Deze bevatten een aantal systeem commando's en SQL commando's. In principe gebeuren er de volgende zaken in zo'n programma:

- het weggooien van het oude bestand,
- het weggooien van de oude, tijdelijke tabel (drop table),
- het creëren van de tabel met behulp van select statement,
- de gegevens in de tabel kopieren naar een tekst (.txt) bestand,
- de opmaak van het tekstbestand aanpassen mocht dat nodig zijn.

Een voorbeeld van zo'n "shell script" (een soort batch programma) is terug te vinden in tekstbox 8-1. De reden van het aanpassen van de tekstfile is dat de gehele vorm van een polylijn dan wel polygoon, als één attribuut wordt weggeschreven en er dus als volgt uitziet:

((x,y),(x,y),(x,y),(x,y),(x,y),(x,y),(x,y),(x,y)).Alle haakjes en komma's zullen vervangen moeten worden door tabs.
Verder zal er aan het einde van alle regels '-1' gezet moeten worden zodat het einde van de regels in het versnijdingsprogramma (zie hoofdstuk 7) kunnen worden herkend. C++ kan wel einde van regels wegschrijven, maar herkent ze bij standaard manier van inlezen niet. Ze worden net als spaties en tabs behandeld.

```
#!/bin/sh
# Exporteer delen van de tabel deelgebouwen naar een tekst file
set -v
# Lees de globale variabelen in, hierin worden $DB en $RESULTDIR gedefinieerd
globals.sh
# Gooi oude resultaatfiles weg.
rm -f $RESULTDIR/deelgebouwen_classificatie.copy.gz
# Kopieer de resultaten vanuit Ingres naar een file.
sql $DB << EOF
copy deelgebouwen(
   id= c0tab,
   classif_text= c0tab,
   waardering= c0tab,
   classif_oud= c0nl
) into '$TMPDIR/deelgebouwen_classificatie.copy'
\g
EOF
# Schrijf de classificaties naar arc.
cat $TMPDIR/deelgebouwen_classificatie.copy |\
   sed 's/ //g' |\
   sed 's/           /,/g' > $TMPDIR/overlay_classificatie.txt
# Gooi het tussenresultaat weg
rm -f $TMPDIR/deelgebouwen_classificatie.copy
```

Programma's,
SQL,
UNIX commando's

Tekstbox 8-1 voorbeeld van het exporteren naar een tekstbestand. De groene tekst geeft het formaat weer waarop elke regel weggeschreven moet worden

De gegevens die uitgelezen worden zijn:

- Uit LKI-tabel Xfio_parcel de attributen linker object, rechterobject en de vorm (engels: shape) van de polylijn. Dit zal gebruikt worden om de percelen laag op te bouwen.
- Uit LKI-tabel Xfio_line het attribuut vorm van de polylijn, waarbij alleen de hoofdgebouwen geselecteerd zullen worden. Dit zal gebruikt worden om de gebouwen laag op te bouwen.
- Uit LKI-tabel Xfio_text de attributen: identificatie, x- en y-coördinaat van het objectpunt en een code. Alleen de objecten met code 2, dit staat voor huisnummers, worden gebruikt om te koppelen aan de deelgebouwen. Dit zal uiteindelijk de tabel huisnummers opleveren.
- Uit de ACN-tabel de attributen identificatienummer, x- en y-coördinaat van de locatie.

Voor de visualisatie worden ook een aantal tabellen uitgelezen: De deelgebouwen met hun classificaties en ACN met hun classificaties. De percelen en LKI_tekst zijn al aan het begin van het proces omgezet naar bestanden die in ArcView ingelezen kunnen worden. Deze laatste is bedoeld om er een overzichtelijke kaart van te maken

door straatnamen en huisnummers uit deze kaartlaag in beeld te zetten.

Het inlezen van de resultaten van de versnijding.

Het inlezen van de tekstbestanden naar de database kan op een bijna vergelijkende manier. Een voorbeeld van zo'n invoer is te vinden in de volgende tekstbox.

```
 $GEOHOME/ingres/copyrel $DB deelgebouwen \
    -n id -n area -q g_akr_objectnummer -n gebouwen_id -n n_buren -n n_acn \
    -n n_huisnummers -q cc_11tot15 -n classificatie
of
copy deelgebouwen(
   id= c0tab,
   classif_text= c0tab,
   waardering= c0tab,
   classif_oud= c0nl
) from '$TMPDIR/deelgebouwen_classificatie.copy'\g
```

Texstbox 8-2 Voorbeeld van commando's om gegevens uit een tekstbestand naar de database te schrijven.

Alleen de attributen zullen ingelezen worden. De geometrie is weggeschreven in een bestand dat ingelezen kan worden in ArcView (zie hoofdstuk 2.4). De volgende tabellen worden gevuld met gegevens elk uit een eigen bestand: deelgebouwen, gebouwen, ACN en huisnummers.

8.2.2 DE SELECTIE VAN DE JUISTE GEGEVENS IN DE DATABASE

We willen natuurlijk alleen de op dat moment geldige gegevens. Om dat te doen zal bij elke selectie steeds geselecteerd worden op "tmax gelijk aan nul". Dit omdat een selectie op deze voorwaarde alle op dat moment geldige elementen selecteert.

```
CREATE TABLE boundary_selection
AS SELECT l_obj, r_obj, shape
FROM xfio_boudary
WHERE tmax =0 and overlaps (rect, bbox)=1
```

Uit de LKI tabel voor de grenzen worden de kolommen voor de identificatievan het linker en rechter vlak en de vorm geselecteerd. Als voorwaarde weer tmax = 0 en eventueel extra de voorwaarde dat alle gegevens binnen een klein gebied liggen zodat ik een kleine testset heb om voor het versnijdingsprogramma te gebruiken.

```
CREATE TABLE line_selection
AS SELECT object_id, shape
FROM xfio_line
WHERE tmax =0 and overlaps (rect, bbox)=1
        And classif in (11,13)
```

Uit de LKI tabel voor de lijnen worden de kolomen voor de identificatie en de vorm geselecteerd. Dezelfde voorwaarden als voor de grenzen gelden. Maar nu wordt ook een selectie op de classificatie codes gemaakt, zodat alleen de lijnen overblijven

die bij de hoofdgebouwen horen. De belangrijkste van deze code is waarde 11, wat voor hoofdgebouw staat [LKI99].

Een andere tabel die uiteindelijk niet gebruikt is in het classificatieproces, maar die toch besproken moet worden is de tabel met objectadressen. Dit voorbeeld illustreert de mogelijkheid om kolomen samen te voegen.

```
CREATE TABLE MO_object_adres_selection
AS SELECT x_akr_objectnummer,
    postcode+huisnummer +huisletter
    +huisnummertoevoeging AS o_adres,
    soort_cult_beb
FROM MO_object
WHERE tmax >0 , soort_cullt in (11, ...)
```

Uit de AKR tabel voor de percelenadressen worden de kolomen voor de AKR-identificatie en soort cultuurcode bebouwing, samen met postcode, huisnummer, huisletter, huisnummertoevoeging als adres geselecteerd. Omdat al deze attributen hetzelfde type, karakters, hebben, kunnen ze door er een plus tussen te zetten samengevoegd worden naar één waarde. De postcode is gedefinieerd als 6 karakters, het huisnummer als 5 karakters en de huisletter als één karakter en de huisnummertoevoeging als 4 karakters, het resulterende type van adres zal dan dus een lengte krijgen van 16 karakters (6+5+1+4)
Als voorbeeld, de punten in het gevormde adres geven in dit geval lege posities weer.

Postcode:	3118XX
Huisnummer	16
Huisletter	leeg
Huisnummertoevoeging	leeg
Het gevormde adres	3118XX...16.....

8.3 HET AANPASSEN VAN DE TABELLEN MET BEHULP VAN SQL

Eerst volgt een overzicht van de tabellen en de stappen die doorlopen worden om ze aan te passen. Daarna volgt een verdere beschrijving van de aanpassingen. Een voorbeeld van de gegevens in de tabellen, één gevulde regel in de deelgebouwen tabel met de bijbehorende regels in de andere tabellen, is terug te vinden in figuur 8.4.

8.3.1 EEN OVERZICHT VAN DE TABELLEN EN HUN ATTRIBUTEN.

Na de versnijding en nadat de attributen weer zijn ingelezen in de database zien de tabellen die nodig zijn voor de classificatie er als volgt uit:

Tabel	Kolomen
Deelgebouwen	identificatie, perceelidentificatie, gebouwidentificatie, buren, oppervlakte

Gebouwen	identificatie, oppervlakte
Percelen	identificatie (G_AKR_objectnummer), oppervlakte
Huisnummers	identificatie, deelgebouwidentificatie, object_tekst
ACN	identificatie, aantal, deelgebouwidentificatie, perceelidentificatie

Tabel 8-2 Startpunt van de tabellen na de versnijding. De bij elkaar horende sleutels hebben dezelfde kleur.

Omdat niet alle ACN in een deelgebouw valt, is het ook nodig om voor alle ACN te bepalen in welk perceel de ACN liggen. Hierdoor kunnen de ACN die nog niet bij een deelgebouw horen in een later stadium, wanneer er maar één deelgebouw op een perceel ligt alsnog aan dit deelgebouw gekoppeld worden. Dit is voor de huisnummers niet nodig omdat deze altijd in een gebouw liggen. Ook geldt voor de huisnummertabel dat het geen onderdeel vormt van het te behalen resultaat, deze tabel is alleen een hulpmiddel en zou aan het einde van de classificaties weggegooid kunnen worden.

De volgende stap is om deze tabellen met nog een aantal attributen aan te vullen. Dit gebeurt met behulp van Structured Query Language (SQL). Deze attributen hebben een naam die beginnen met “n_”, waarbij n voor aantal staat. Na het berekenen van een aantal attributen met aantallen, zien de aangepaste tabellen er als volgt uit.

Tabel	Kolomen
Deelgebouwen	ID, perceelidentificatie, gebouwidentificatie, buren, oppervlakte, *n_acn, n_huisnummers.*
Gebouwen	Identificatie, oppervlakte, *n_deelgebouwen, n_acn, n_huisnummers, n_percelen.*
Percelen	Identificatie (G_AKR_objectnummer), oppervlakte, *n_acn, n_deelgebouwen*

Tabel 8-3 tabellen na berekening van aantallen.

De tabel percelen moet bijgevuld worden uit de tabellen MO_Object en MO_Objectadres met het attribuut cultuurcode. Omdat dit in principe meer dan 1 cultuurcode kan zijn wordt de laagste waarde die <>00 uit alle gevonden waarde overgenomen om het attribuut cultuurcode in de tabel percelen te vullen. Daarna wordt voor elk deelgebouw bepaald of het bijbehorende perceel een cultuurcode heeft dat wonen indiceert.
Na het toevoegen van deze attributen zien de aangepaste tabellen er als volgt uit:

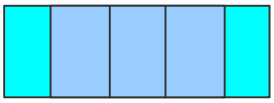
Tabel	Kolomen
Deelgebouwen	ID, perceelidentificatie, gebouwidentificatie, buren, oppervlakte, n_acn, n_huisnummers,

	indicatie_wonen.
Percelen	Identificatie (G_AKR_objectnummer), oppervlakte, n_acn, n_deelgebouwen*, Cultuurcode*

Tabel 8-4 tabellen na aanvulling van een aantal attributen uit de bestaande database.

8.3.2 BEREKENEN VAN NOG LEGE ATTRIBUTEN.

Alle bestanden met SQL-statements die voor deze stap in het gehele proces worden gebruikt zijn terug te vinden in bijlage E.
In deze paragraaf wordt beschreven hoe de aantallen berekend worden die in de verschillende tabellen met “n_” zijn aangeduid. Dit kan gezien worden als het bepalen van de werkelijke waarde n in elke 1:n relatie.



Figuur 8-2 Voorbeeld van één gebouw met vijf deelgebouwen

Een voorbeeld in tekstvorm ziet er als volgt uit:

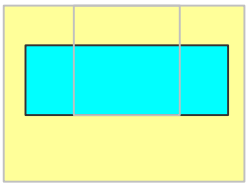
Het aantal deelgebouwen dat bij een gebouw hoort kan geteld worden door het tellen van het aantal keren dat de unieke gebouwidentificatie voorkomt in de kolom gebouw_id van de deelgebouwen tabel. Hiervoor wordt een tabel met tussenresultaat aangemaakt.

In SQL:

```
CREATE TABLE
tussenresultaat_telling_deelgebouwen
SELECT count(deelgebouw_id) as aantal,
gebouw_id
FROM deelgebouwen
GROUP BY gebouw_id
\p\g
```

Hierna kunnen vanuit dit tussenresultaat de gebouwentabel via de gebouwidentificatie worden aangepast. Dit is een 1 op 1 relatie.

```
UPDATE gebouw
FROM tussenresultaat
SET n_deelgebouwen=tussenresultaat.aantal
WHERE tussenresultaat.gebouw_id =
gebouw.gebouw_id
\p\g
```



Figuur 8-3 Twee percelen, en drie deelgebouwen. Het grijze perceel wordt aan 3 kanten omringddoor hetzelfde perceel

Het berekenen van de attributen *n_acn, n_huisnummers* uit de tabel *deelgebouwen, n_acn, n_huisnummers, _percelen.* Uit de tabel *gebouwen* en *n_acn, n_deelgebouwen* uit de tabel *percelen* gaat op een zelfde manier. De SQL statements die hiervoor gebruikt zijn, zijn terug te vinden in bijlage F. De verwachting is dat n_deelgebouwen en n_percelen in de tabel gebouwen meestal hetzelfde is. De reden waarom ook het aantal percelen geteld wordt is om speciale situaties als weergegeven in figuur 8.4 te kunnen onderscheiden van normale situaties.

Deelgebouw							
ID	Gebouw_id	Perceel_id	N_buren	N_ACN	N_huisnr	ind_wonen	Oppervlakte
12032	1007682	APD01Z405149G0000	0	3	3	T	$2.982 \cdot 10^8$

Gebouw					
ID	N_deelgebouwen	N_percelen	N_ACN	N_huisnr	Oppervlakte
1007682	1	1	3	3	$2.982 \cdot 10^8$

Perceel				
AKR_id	Cultuur_code	N_gebouwen	N_ACN	Oppervlakte
APD01Z405149G0000	11	1	3	$2.982 \cdot 10^8$

ACN						
ID	Aantal	Postcode	huisnr	Toevoeging	Deelgebouw_id	Perceel_id
2111247	1	7321 CW	77	A	12032	APD01Z405149G0000
2111248	1	7321 CW	77	B	12032	APD01Z405149G0000
2111249	1	7321 CW	79		12032	APD01Z405149G0000

Figuur 8-5 een voorbeeld met gevulde, bij elkaar behorende, records (tabel huisnummers is weggelaten).

Vervolgens is alles klaar om de classificatie te gaan uitvoeren. De tabel huisnummers wordt nu niet verder meer gebruikt. Hierboven volgt een een overzicht van de tabellen en de relaties tussen de tabellen die gebruikt zullen worden voor de classificatie. Voor elke tabel is één of meer records ingevuld behorende bij één deelgebouw.

8.3.3 HET AANPASSEN VAN DE TABEL ACN

Alle coördinaten die op een perceel met precies één deelgebouw liggen worden doorgekopieerd naar dit deelgebouw. Dit gaat als volgt in zijn werk. Eerst wordt een tijdelijke tabel aangemaakt met alle percelen met 1 deelgebouw. Vanuit deze tabel wordt 1 op 1 het attribuut deelgebouwidentificatie van de ACN-tabel gevuld met de deelgebouwidentificatie die bij het perceel hoort dat vermeld is in het attribuut perceelidentificatie. Deze update van de adressentabel gebeurt pas halverwege de classificatie en alleen de lege cellen worden gevuld. Dit gebeurt pas halverwege omdat we wel onderscheid willen maken in waardering tussen gebouwen met ACN die juist gelokaliseerd zijn en gebouwen met adressen die niet of een minder goede locatie hebben. In SQL ziet deze aanpassing er als volgt uit:

```
CREATE TABLE deelgebouw_op_perceel
SELECT deelgebouw_id, perceel_id FROM percelen
WHERE n_deelgebouwen=1
\p\g
UPDATE CAN FROM deelgebouw_op_perceel
SET deelgebouwenidentificatie = deelgebouw_id
WHERE ACN.perceel_id = deelgebouw_op_perceel.perceel_id
    AND ACN.deelgebouwenidentificatie=' '
\p\g
```

Hierna moeten ook de n_acn van de deelgebouwen weer opnieuw worden berekend.

8.4 HET UITVOEREN VAN DE CLASSIFICATIE VAN DEELGEBOUWEN

Voor het uitvoeren van de classificatie zijn een aantal beslisregels opgesteld in tekstvorm aan de hand van de kenmerken van de verschillende klassen zoals beschreven staat in paragraaf 5.3. Deze zijn daarna omgezet naar SQL-statements (ascii-tekst, bijlage F). De beslisregels zijn bedoeld om aan de hand van de nu aanwezige attributen een onderscheiding naar woningtype te maken met behulp van de verschillen in de records. De beslisregels zullen uiteindelijk allemaal in SQL-vorm gezet worden en gezamenlijk in één bestand bewaard worden.
Voordat het uitvoeren van de beslisregels kan gebeuren, zullen er eerst twee extra kolomen aan de tabel deelgebouwen en ACN moeten worden toegevoegd. Dit zijn de attributen woningtype en waardering. Voordat de classificatie begint krijgen alle deelgebouwen een default waarde. De default waarde is voor woningtype '-' en voor waardering '0'.

Het woningtype is één van de karakters:

- # Cultuurcode niet wonen
- – Nog niet geclassificeerd
- A Appartement
- E Eindwoning
- M Middenwoning
- T Twee-onder-één-kap woning
- V Vrijstaande woning
- S Schuur, dit zijn alle vrijstaande gebouwen zonder een ACN.

De waardering is een getal en geeft aan onder welke groep classificatieregels de classificatie is uitgevoerd. Een waardering van 0 hoort alleen bij de woningtypen # en -. Een waardering van 1 houdt in dat de classificatie de ideale situaties zijn, zoals ze beschreven zijn in hoofdstuk 5. Een waardering 2 geeft aan dat het adres niet in het deelgebouw valt maar erbuiten. Dit is een versoepeling van de eisen op ACN.

De beslisregels zijn opgesteld aan de hand van karakteristieken van de vijf verschillende klassen. Eerst zal voor alle klassen het ideale geval worden beschreven in de volgende paragraaf, Daarna volgt een overzicht in tabel vorm van alle classificaties. In paragraaf 8.4.4 worden de versoepelingen van de karakteristieken beschreven. In deze paragraaf zal verder ingegaan worden op de waardering. Daarnaast zijn er nog de beslisregels voor de adressen. Deze worden besproken in paragraaf 8.5

8.4.1 DE BESLISREGELS

Op dit moment zijn er zo'n twintig beslisregels opgesteld. Deze beslisregels kunnen opgedeeld worden in twee groepen. De groep met 5 beslisregels voor de ideale situaties en de rest, welke aanpassingen zijn van deze vijf hoofdregels. De aanpassingen bestaan uit versoepelingen van de kenmerken die bij de woningtype te bedenken zijn.
Voordat we een overzicht geven van de classificaties moet de volgende view genoemd worden: AKR_a (G_akr_objectnummer, X_akr_objectnummer). Dit is een view op de database en bevat de relaties tussen gehele percelen en appartementen. Deze is al in de database aanwezig en is niet apart voor dit project aangemaakt. Met behulp van deze view wordt de eerste classificatie uitgevoerd. Dit is de classificatie van appartementen en deze classificatieregel krijgt een waardering van 1.

Hieronder is een tabel met een overzicht van de classificaties van de ideale cases weergegeven. Deze krijgen allemaal een waardering 1. Deze tabel bestaat uit een samenvoeging van de vijf tabellen die gebruikt worden voor de classificatie (eerste regel). Op de tweede regel zijn de attributen per tabel terug te vinden die gebruikt worden in de classificatie. En wanneer er gebruik gemaakt wordt van de koppelingen (externe sleutels) tussen de verschillende tabellen zijn de bij behorende vlakken hetzelfde gekleurd. De getallen in de tabel geeft aan waar het bijbehorende attribuut aan moet voldoen. Dezelfde regels die hier zijn weergegeven worden in de volgende paragraaf in SQL weergegeven.

Deelgebouwen										Gebouwen						Percelen						ACN				AKR_A	
classificatie	waardering	Deelgebouw_id	perceel_id,	gebouw_id	buren	oppervlakte	n_acn	n_hnr	indicatie wonen	Gebouw_id	Oppervlakte	n_deelgebouwen	n_ACN	n_Percelen	n_hnr	Perceel_id	Oppervlakte	Cultuurcode	n_ACN	n_Gebouwen	n_deelgebouwen	ACN_id	Aantal	Deelgebouw_id	Perceel_id	G_akr_objectnummer	X_akr_objectnummer
A	1																										
V	1				0		1	1	Y			1		1	1					1			1				
T	1				1		1	1	Y			2	2	2	2					1			1				
E	1				1		1	1	Y			>2		>2	2					1			1				
M	1				>1		1		Y			>2		>2	2					1			1				

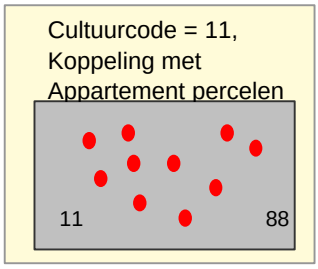
Tabel 8-5 classificatieregels voor de ideale cases.

A Appartement
V Vrijstaand
T twee-onder-één-kap
E Eindwoning
M Middenwoning

Voorbeeld: Een deelgebouw met geen buren, één huisnummer en adrescoördinaat, een cultuurcode wonen, dat hoort bij een gebouw dat uit één deelgebouw bestaat, op één perceel ligt en één huisnummer bevat, waarvan op het bijbehorende perceel maar één gebouw staat en de bijbehorende adrescoördinaat maar één adres bevat is een vrijstaand huis

8.4.2 SQL VORMEN VAN DE BESLISREGELS

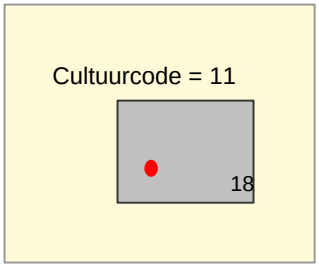
De karakteristieken van de in de vorige paragraaf genoemde woningtypen zijn gebruikt om SQL beslisregels te vormen. Hier volgt een overzicht van de ideale gevallen de koppelingen tussen de tabellen zijn steeds in groenblauw weergegeven.



Figuur 8-4 A Appartementssituatie

Appartementen:

```
UPDATE Deelgebouwen
FROM akr_object_a
SET Classificatie = A, Waardering=1
WHERE
Deelgebouwen.perceelnr=akr_object_a.g_akr_objectnummer
      AND deelgebouw.classificatie=-1
```

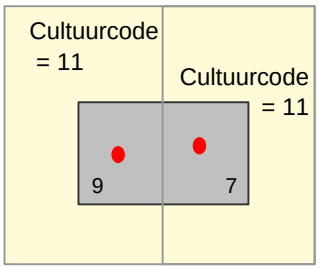


Figuur 8-4 B Situatie Vrijstaande woning

Percelen zijn lichtgeel gekleurd, deelgebouwen zijn nog grijsgekleurd en adrescoordinaten zijn rood

Vrijstaande woning

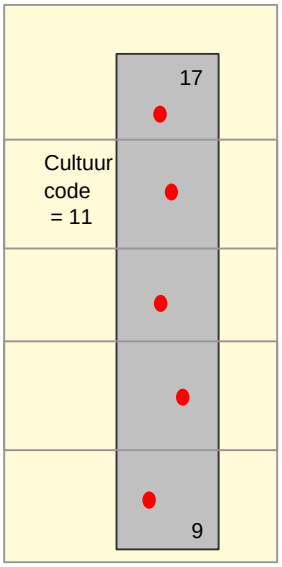
```
UPDATE Deelgebouwen
FROM Gebouwen, Percelen, ACN
SET Classificatie = V, Keuze=1
WHERE deelgebouwen.gebouw_id=gebouw.id
AND deelgebouwen.perceel_id=perceel.id
     AND ACN.in_deelgebouw=deelgebouw.id
     AND deelgebouw.classificatie=-1
     AND deelgebouw.cc11tot15 = 'T'
     AND deelgebouw.buren = 0
     AND gebouw.n_deelgebouwen = 1
     AND gebouw.n_percelen = 1
     AND perceel.n_hoofdgebouw = 1
     AND deelgebouw.n_ACN = 1
     AND acn.aantal = 1
     AND deelgebouw.n_huisnummers = 1
```



Figuur 8-4 C situatie twee-onder-één-kap

Twee onder een kap

```
UPDATE Deelgebouwen
FROM Gebouwen, Percelen, ACN
SET Classificatie = T, Keuze=1
WHERE deelgebouwen.gebouw_id=gebouw.id
      AND deelgebouwen.perceel_id=perceel.id
      AND ACN.in_deelgebouw=deelgebouw.id
      AND deelgebouw.classificatie=-1
      AND deelgebouw.cc11tot15 = 'T'
      AND deelgebouw.buren = 1
      AND gebouw.n_deelgebouwen = 2
      AND gebouw.n_percelen = 2
      AND perceel.n_hoofdgebouw = 1
      AND deelgebouw.n_ACN = 1
      AND acn.aantal = 1
      AND deelgebouw.n_huisnummers = 1
```



Figuur 8-4 D situatie rijtjeswoningen

Middenwoning

```
UPDATE Deelgebouwen
FROM Gebouwen, Percelen, ACN
SET Classificatie = M, Keuze=1
WHERE deelgebouwen.gebouw_id=gebouw.id
      AND deelgebouwen.perceel_id=perceel.id
      AND ACN.in_deelgebouw=deelgebouw.id
      AND deelgebouw.classificatie=-1
      AND deelgebouw.cc11tot15 = 'T'
      AND deelgebouw.buren > 1
      AND gebouw.n_deelgebouwen > 2
      AND gebouw.n_percelen > 2
      AND perceel.n_hoofdgebouw = 1
      AND deelgebouw.n_ACN = 1
      AND acn.aantal = 1
```

Hoek woning

```
UPDATE Deelgebouwen
FROM Gebouwen, Percelen, ACN
SET Classificatie = E, Keuze=1
WHERE deelgebouwen.gebouw_id=gebouw.id
      AND deelgebouwen.perceel_id=perceel.id
      AND ACN.in_deelgebouw=deelgebouw.id
      AND deelgebouw.classificatie=-1
      AND deelgebouw.cc11tot15 = 'T'
      AND deelgebouw.buren = 1
      AND gebouw.n_deelgebouwen > 2
      AND gebouw.n_percelen > 2
      AND perceel.n_hoofdgebouw = 1
      AND deelgebouw.n_ACN = 1
      AND acn.aantal = 1
      AND n_huisnummers = 1
```

8.4.3 OVERZICHT VAN DE REGELS IN TABEL VORM

Op de volgende pagina volgt een overzicht van alle beslisregels voor het classificeren van deelgebouwen. De attributen die de tabellen koppelen hebben dezelfde kleur gekregen. Als bij een bepaalde beslisregels het vlakje bij een attribuut gekleurd is dan geeft dat aan dat er gebruik gemaakt wordt van een koppeling met een van de andere tabellen via dit attribuut.
De volgorde waarop de beslisregels worden toegepast komt overeen met de volgorde zoals ze hier in de tabel worden toegepast. De classificatie en de waardering samen zijn een unieke aanduiding voor de classificatieregel die is toegepast. In de laatste kolom zijn de aantallen weergegeven van het aantal deelgebouwen uit de testset Apeldoorn dat met behulp van deze beslisregel is geclassificeerd.

Tabel 8-6 Een overzicht van de gedefinieerde beslisregels.

Deelgebouwen										Gebouwen					Percelen					ACN			AKR_a		
classificatie	waardering	Deelgebouw_id	Perceel_id,	Gebouw_id	Buren	Oppervlakte	n_ACN	n_huisnummersr	CultuurCode_11t/mn15	Gebouw_id	n_deelgebouwen	n_ACN	n_Percelen	n_huisnummers	Perceel_id	Cultuurcode	n_ACN	n_Gebouwen	n_deelgebouwen	ACN_id	Aantal	Deelgebouw_id	G_AKR_objectnummer	X_AKR_objectnummer	Aantallen in de testset Apeldoorn
A	1								T															T	297
#	0															>15									2454
V	1				0		1	1	T		1		1	1											9010
T	1				1		1	1	T		2	<3	2	2											3959
E	1				1		1	1	T		>2		>2	>0											3037
M	1				>1		1	<2	T		>2		>2	>0											6167
S	0				0		0										<								715
V	2				0		0	1	T								1	1							1212
T	2				1		0	1	T		2		2				1	1							914
E	2				1		0	1	T		>2		>2				1	1							1536
M	2				>1		0		T		>2		>2				1	1							3831
ACN-UPDATE zie paragraaf 8.3.3																									
A	3								T												>7				229
V	3				0		1		T		1		1												243
T	3				1		1		T		2	<3	2												296
E	3				1		<2		T		>2		>2												93
M	3				>1		<2		T		>2		>2												248
V	8				0	<120	1	1								00										224
T	8				1	<120	1	1			2	2	2			00										123
E	8				1	40-100	1	1			>2		>2	>0		00										156
M	8				>1	40-100	1	<2			>2		>2			00										328
V	4				0		1																			138
T	4				0		2																			995
M	4				>1		1																			13
V	5				0		1	1																		0
T	5										2	2	2													58
T	6				0		2	2																		0
T	7				0			2									2	1								0
A	5					f<30	>2																			202
M	5					f>50	>2																			1283
M	6					f>40	>2																			399
M	7					f>30	>2																			124
T	9				1			2									<3									31
M	9				>1		>1																			116
Classificatie	Waardering	Deelgebouw_id	Perceel_id,	Gebouw_id	Buren	Oppervlakte	n_ACN	n_hnr	CC_11t/mn15	Gebouw_id	n_deelgebouw	n_ACN	n_Percelen	n_hnr	Perceel_id	Cultuurcode	n_ACN	n_Gebouwen	n_deelgebouw	ACN_id	Aantal	Deelgebouw_id	G_akr_objectnr	X_akr_objectnr	Aantallen
Deelgebouwen										Gebouwen					Percelen					ACN			AKR_a		

Tekstbox 8-3: Opmerkingen bij tabel 8-6

- T staat voor True (waar)
- f = Area / n_CAN
  De oppervlakte gedeeld door het aantal bijbehorende adressen
- De gekleurde vakken geven de sleutels tussen twee verschillende tabellen aan.
  - Groen voor de koppeling tussen deelgebouwen en percelen,
  - Turkoois voor de koppeling tussen deelgebouwen en gebouwen,
  - Paarse voor de koppeling tussen deelgebouwen en ACN.

Als een van de eerste dingen die gedaan worden is het toekennen van classificatiekarakter '#' en waardering '0' aan alle cultuurcodes die niet wonen als hoofdgebruik hebben.
Vervolgens een uitleg over Regel S0, deze selecteert alle vrijstaande gebouwen waarvan er geen adres in het gebouw zelf valt en waarvan het aantal adressen op het perceel kleiner is dan het aantal deelgebouwen. De adressen die wel op het perceel liggen worden aan een van de andere deelgebouwen op het perceel gekoppeld. Deze regel is overgenomen van het kadasterproject.

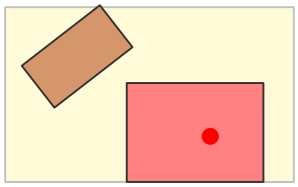
Tijdens het woningindex project, zijn er bij het kadaster ook beslisregels opgesteld voor de classificatie. De regels van dit afstudeerproject lijken veel op deze regels. Sommige zijn zelfs helemaal hetzelfde, maar er zijn zeker ook verschillen. De Kadasterregels zijn in een zelfde tabel als hierboven weergegeven in bijlage D. Zij hebben de waardering en het woningtype niet van elkaar gescheiden, elk deelgebouw / adres krijgt een getal tussen –3 en 99 toegekend. Er is gebruik gemaakt van deze kadasterregels voor het opstellen van de beslisregels zoals ze gelden in het afstudeerproject. De consistentie van de beslisregels, zoals ze bij het kadasterproject zijn gedefinieerd, kon beter. De verbeteringen zijn in deze tabel terug te vinden. Ook is geprobeerd om de regels in te delen naar gelijke waarderingen. Dit had tot gevolg dat de volgorde van de classificatieregels een klein beetje veranderd is. Alle bedachte classificatieregels zijn in tabel 8-6 verwerkt. Sommige leveren niks op (0 in de laatste kolom). Dit komt omdat het aanpassen van de tabellen met betrekking tot de adrescoördinaten sommige regels uitsluiten. Een andere reden is dat sommige regels te veel op elkaar lijken.

Tabel 8-7 Volgorde waarin de classificaties uitgevoerd worden.

Volgorde	- 0	A 1	# 0	V 1	T 1	E 1	M 1	S 0	V 2	T 2	E 2	M 2	ACN-update	
Vervolg	A 3	V 3	T 3	E 3	M 3	V 8	T 8	E 8	M 8	V 4	T 4	M 4	V 5	T 5
Vervolg	T 6	T 7	A 5	M 5	M 6	M 7	T 9	M 9						

Letter=woningtype
Getal=waardering

8.4.4 BESCHRIJVING CLASSIFICATIEREGELS NIET-IDEALE CASES.



Figuur 8-5 voorbeeld van een vrijstaand huis met een schuur.

Nadat de ideale cases van de deelgebouwen zijn geclassificeerd worden de deelgebouwen, die niet aan een adres gekoppeld kunnen worden, geclassificeerd als schuren. Dit zijn gebouwen waarin geen adres valt en waarvoor geldt dat het aantal gebouwen groter is dan het aantal adressen.

De aanpassingen op de eisen ten opzichte van de ideale cases kunnen onderverdeeld worden in:

1. ACN vanuit het deelgebouw naar het perceel
2. Het loslaten van de kenmerken wat betreft huisnummers
3. Selectie op cultuurcode '00'
4. Gebruik maken van oppervlakte verhoudingen
5. Huurhuissituaties, waarbij het aantal en de plaatsing van adrescoördinaten belangrijk worden.

Er zijn een aantal redenen waarom al deze verschillende classificaties zijn gedefinieerd. Ten eerste omdat gebleken is dat de adrescoördinaten toch redelijk vaak buiten de gebouwen vallen. Een andere reden is omdat ook huurwoningen en deelgebouwen met nog niet bekende cultuurcode geclassificeerd dienen te worden.
In de volgende tabel wordt een korte omschrijving gegeven van de waarderingen. De volgorde van deze tabel is dezelfde als de volgorde waarin geclassificeerd is.

Tabel 8-8 Overzicht van de waarderingen.

Waardering	Omschrijving
1	Ideale cases
2	Adressen buiten het huis en eisen wat betreft huisnummer vervallen
	ACN, deelgebouwen, gebouwen tabellen aanpassen
3	Laatste reeks met cultuurcode dat wonen indiceert
8	Cultuurcode 00 en oppervlakte voorwaarde
4	Alleen eisen aan deelgebouwen
V, T 5-7	Aantallen van ACN belangrijk (huursituaties)
A, M 5-7	Oppervlakteverhoudingen
9	Overig

Ten eerste moeten de classificaties V2, T2, E2 en M2 genoemd worden. Dit zijn de classificaties waarbij de adressen niet meer in het deelgebouw hoeven te liggen, maar precies één adres op het perceel hebben. Als het deelgebouw verder, behalve het hebben van een huisnummer in het deelgebouw, aan de eisen voldoet dan krijgt het een waardering "2".

Nadat deze classificaties zijn uitgevoerd worden de adressen, waarvoor het mogelijk is, gekoppeld aan een deelgebouw. Hoe dit gebeurt is beschreven in paragraaf 8.3.3. Hierna worden de rest van de classificaties uitgevoerd.

De eerstvolgende groep van vijf classificaties is de laatste groep waarbij de eis dat een cultuurcode gelijk aan wonen (11-15) nog geldt. Deze groep krijgt een waardering van 3. Daarna wordt de cultuurcode “00” gebruikt in de volgende groep classificaties, dit is de laatste keer dat op de cultuurcode geselecteerd wordt.
Alle classificatieregels worden nu per hoofdklasse besproken.

Appartementen:

A3 Voor deze classificatie is het hele appartementen-koppelingsverhaal uit AKR losgelaten. Als er bij een deelgebouw 1 adrescoördinaat ligt, met voor deze coördinaat een aantal adressen dat groter is dan zeven, dan valt dit deelgebouw onder deze classificatie. De intentie van de adrescoördinaten is dat ze in principe zoveel mogelijk bij de ingang van een gebouw liggen. Dus als er een heleboel bij elkaar op een en dezelfde plek liggen dan zou dat kunnen duiden op een gezamenlijke ingang voor het gehele gebouw.

A5 Ook voor deze classificatie is het hele appartementenverhaal uit AKR losgelaten. Deze beslisregel komt bijna aan het einde en maakt gebruik van de verhouding oppervlakte van het deelgebouw staat tot aantal adrescoördinaten (ACN). Wanneer de oppervlakte per adres coördinaat kleiner wordt dan $30*10^6$ mm^2 (30 vierkante meter) is het niet erg aannemelijk dat alle adressen zich op de begane grond zullen bevinden. Dit is een veel te kleine waarde als vloeroppervlak voor een woning, dus hebben we dan waarschijnlijk te maken met meerdere woonlagen, en dus appartementen.

De vrijstaande woningen:

V3 Losgelaten eis ten opzichte van de hoofdclassificatie is de eis dat er een huisnummer in het gebouw moet liggen. Dit komt voor als er heel veel van dezelfde losse huizen in een rij staan, dan willen de middelste wel eens geen huisnummer hebben. Huisnummers zijn vooral van belang bij het begin en einde van een straat. Verder is de eis dat het aantal deelgebouwen maar 1 mag zijn losgelaten. Maar die eis zit eigenlijk al verborgen in het feit dat er geen buren mogen zijn

V4 Bij waardering 4 wordt alleen gekeken naar de eisen op het deelgebouw zelf. Er wordt niet gekeken naar het gebouw of het perceel. Geen buren en precies één adres, houdt in dat het een vrijstaand huis is.

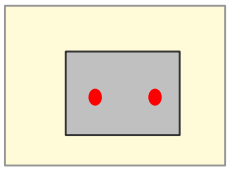
V5 Deze is hetzelfde als V4, alleen toegevoegd de eis dat er een huisnummer in het gebouw moet staan. Daarom levert deze classificatie niets op. V4 en V5 zouden omgewisseld moeten worden.

V8 De verschillen met de ideale case zijn in dit geval geen eisen aan het gebouw, want omdat er geen buren zijn, zijn die hetzelfde als het deelgebouw. Verder is er op cultuurcode 00 geselecteerd. Omdat dit ook niet-woningen kunnen zijn is er een oppervlakte eis toegevoegd

zodat fabrieken of iets dergelijks niet als woning geclassificeerd worden. Dit houdt in dat het gebouw niet te groot mag zijn.

De twee-onder-één-kap woning:

T3 Eisen aan huisnummers zijn losgelaten. Deze klasse bevat dus ook de deelgebouwen die gekoppeld zijn aan een adres tijdens de ACN-aanpassing.



Figuur 8-6 B een twee-onder-één-kap op één perceel

T4 Dit zijn als het ware de huursituaties van de twee-onder-één-kap woningen. Wanneer het een vrijstaand gebouw betreft dat niet is opgedeeld, maar waar precies twee adressen in het gebouw vallen dan wordt het een twee-onder-één-kap woning. Zie figuur 8-6 B.

T5 In dit geval wordt er niet meer naar het deelgebouw zelf gekeken, maar wordt er naar het gehele gebouw gekeken. Als er precies twee deelgebouwen, twee ACN en twee percelen zijn, dan is het een twee-onder-één-kap.

T6 Deze classificatie is bijna hetzelfde als T4. Vandaar dat ook hier deze classificatie niets oplevert. Hier is één extra classificatieeis: namelijk dat er ook twee huisnummers in het deelgebouw moeten liggen. Deze twee classificaties zouden weer omgedraaid moeten worden.

T7 Dit is een classificatie waarbij er geen adressen in het vrijstaande deelgebouw vallen, maar waarbij er precies twee adressen in het perceel vallen. Deze classificatie levert niets op, omdat deze situatie al opgevangen wordt door het aanpassen van de ACN-tabel halverwege de classificatie.

T8 Er is op cultuurcode 00 geselecteerd. Omdat dit ook niet-woningen kunnen zijn is er een oppervlakte eis toegevoegd zodat fabrieken of iets dergelijks niet als woning geclassificeerd worden. Dit houdt in dat het gebouw niet te mag groot zijn. Verder moeten er ook precies 2 adressen bij het gehele gebouw horen.

T9 Eén buur, twee huisnummers en minder dan 3 adressen op het perceel.

Middenwoning:

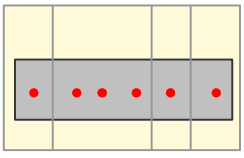
M3 Eisen aan huisnummers zijn losgelaten. Deze klasse bevat tevens de deelgebouwen die gekoppeld zijn aan een adres tijdens de ACN-aanpassing.

M4 Alleen eisen aan het deelgebouw zelf, meer dan één buur en precies één adres in het gebouw.

M5, M6, M7 Deze zijn ongeveer hetzelfde. Alleen is de oppervlakte van het deelgebouw per adres steeds kleiner. Als een gebouw meer dan twee adressen heeft is het een rijtjeswoning. Het gehele deelgebouw krijgt dan de classificatie middenwoning.

M8 Er is op cultuurcode 00 geselecteerd. Omdat dit ook niet-woningen kunnen zijn is er een oppervlakte eis toegevoegd zodat fabrieken of iets dergelijks niet als woning geclassificeerd worden.

Figuur 8-6 C Voorbeeld van een rijtje met koop en huurhuizen



M9 Het aantal buren is groter dan 1 en het aantal ACN is groter dan één. Het komt wel eens voor dat er in een rij een aantal koop en een

aantal huurhuizen samen zitten. Het koophuis staat dan op 1 perceel, maar de aan elkaar grenzende huurhuizen staan dan vaak samen ook op één perceel.

Eindwoning:

E3 Eisen aan huisnummers zijn los gelaten. Deze klasse bevat ook de deelgebouwen die gekoppeld zijn aan een adres tijdens de ACN-aanpassing.

E8 Classificatie van eindwoningen met een cultuurcode 00. Omdat dit ook niet-woningen kunnen zijn is er een oppervlakte eis toegevoegd. De oppervlakte van het deelgebouw moet tussen de 40 en 100 m^2 zijn.

Voor eindwoningen zijn er niet meer classificaties. Een van de redenen hiervoor is dat wanneer we met een rij huizen te maken hebben die niet goed in delen is opgedeeld we dit deelgebouw als middenwoning hebben geclassificeerd. Het einde van de rij is pas bij het classificeren van de adressen te bepalen.

8.5 DE CLASSIFICATIE VAN DE ADRESSEN

Als laatste stap om het doel te bereiken moeten de adressen geclassificeerd worden. Voor de waarderingen geldt dat alle toegevoegde beslisregels die zijn toegepast op adressen een waardering groter of gelijk aan 10 krijgen.
Nadat de deelgebouwen een classificatie gekregen hebben, is de volgende stap om alle adressen een classificatie te geven. Eerst wordt weer een scheiding gemaakt tussen woonadressen en niet-woonadressen, door middel van de cultuurcode. Daarna worden de woonadressen geclassificeerd in drie stappen:

1. Kopieren van de classificaties uit de bijbehorende deelgebouwen.
2. Classificeren van nog niet geclassificeerde adressen met beslisregels
3. Het bepalen van de eindwoningen van rijtjes door het maximum en het minimum huisnummer te berekenen.

De eerste stap gaat met behulp van het volgende SQL commando:

```
update acn
from deelgebouwen
set classif_text = deelgebouwen.classif_text,
waardering = deelgebouwen.waardering
where acn.in_deelgebouw = deelgebouwen.id
\p\g
```

Voor de adressen die niet aan een deelgebouw gekoppeld zijn, wordt nog geprobeerd een classificatie te vinden via het perceel. Deze vorm van classificeren krijgt classificatiewaardering 10. Het adres krijgt dan het maximum (over classif_text) van alle classificaties van deelgebouwen die op het perceel liggen. Dit houd in dat de volgende

volgorde geldt (V, T, S, M, E, A). Twee adressen hebben op deze manier de classificatie middenwoning gekregen.

8.5.1 BESLISREGELS VOOR ADRESSEN

In de tweede stap worden de beslisregels gebruikt, deze zullen in deze deelparagraaf worden besproken.

Classificatie ‘A’, waardering 11

Het adres valt op een perceel met een appartementencomplex, maar het gebouw is niet in de gebouwen kaartlaag terechtgekomen, doordat het of niet in het bestand stond, of niet door de topologieconstructie is gekomen. Dit laatste houdt in dat er een gat in de lijnen zat dat het gebouw zou moeten vormen, en dat dit gat groter is dan 1 meter.
In SQL ziet dit er als volgt uit:

A11
```
update acn
from akr_object_a
set classif_text = 'A',
waardering = 11
where classif_text in ('-', '#')
   and akr_object_a.g_akr_objectnummer =
   acn.g_akr_objectnummer
\p\g
```

Classificatie ‘M’, waardering 15

Elk adres dat in een deelgebouw valt dat nog niet geclassificeerd is, maar welk deelgebouw tot een rij behoort, wordt een middenwoning. Om dat te doen wordt er eerst een hulptabel aangemaakt waarin alle gebouwen zitten, die een deelgebouw bevatten dat als een middenwoning geclassificeerd is. Dit gebeurt met het volgende SQL commando:

```
create table rijtjesgebouwen
as select gebouwen.id,count(acn.id)
   from acn,gebouwen,deelgebouwen
   where acn.classif_text = 'M'
      and deelgebouwen.gebouwen_id = gebouwen.id
      and acn.in_deelgebouw = deelgebouwen.id
   group by gebouwen.id
\p\g
```

Daarna kan met behulp van deze tabel de volgende classificatieregel worden uitgevoerd:

M15
```
update acn
from rijtjesgebouwen,gebouwen,deelgebouwen
set classif_text = 'M',
waardering = 15
where rijtjesgebouwen.id = gebouwen.id
   and deelgebouwen.gebouwen_id = gebouwen.id
   and deelgebouwen.id = acn.in_deelgebouw
```

```
   and acn.classif_text = '-'
\p\g
```

8.5.2 EINDWONINGEN ONDERSCHEIDEN VAN MIDDENWONINGEN

In de laatste stap van de classificatie van adressen worden de middenwoningen van de eindwoningen onderscheiden. Dit gebeurt in twee delen. Eerst worden de adressen met het laagste huisnummer van de rij geclassificeerd en daarna op dezelfde manier de adressen met het hoogste huisnummer van de rij. Om dit te doen moet er weer een hulptabel worden aangemaakt. Voor elk rijtje moet er bepaald worden wat het hoogste, dan wel laagste, huisnummer is. De tabel wordt als volgt gemaakt:

```
create table rijtjes_nummer
as select gebouwen.id , min(adrtxt) as laagnummer,
max(adrtxt) as hoognummer
   from acn,gebouwen,deelgebouwen
   where deelgebouwen.gebouwen_id = gebouwen.id
      and acn.in_deelgebouw = deelgebouwen.id
   group by gebouwen.id
\p\g
```

Daarna kunnen de adressen die het einde van de rij zijn met behulp van de volgende twee classificatieregels worden geclassificeerd:

E16

```
update acn
from rijtjes_min,gebouwen,deelgebouwen
set classif_text = 'E',
waardering = 16
where rijtjes_nummer.laagnummer = acn.adrtxt
   and rijtjes_nummer.id = gebouwen.id
   and deelgebouwen.gebouwen_id = gebouwen.id
   and acn.classif_text = 'M'
   and deelgebouwen.id = acn.in_deelgebouw
\p\g
```

E17

```
update acn
from rijtjes_max,gebouwen,deelgebouwen
set classif_text = 'E',
waardering =17
where
       rijtjes_max.hoognummer = acn.adrtxt
   and rijtjes_max.id = gebouwen.id
   and deelgebouwen.gebouwen_id = gebouwen.id
   and acn.classif_text = 'M'
   and deelgebouwen.id = acn.in_deelgebouw
\p\g
```

Alle classificaties die in dit afstudeerproject zijn gedefinieerd, zijn nu beschreven. Deze classificaties zijn uitgevoerd op een testset. Deze testset bestaat uit alle gegevens van de kadastrale gemeente Apeldoorn.

8.6 RESULTATEN VAN DE CLASSIFICATIE

In deze paragraaf zijn de resultaten van de classificatie van de testset in de vorm van tabellen terug te vinden. De classificaties zijn in te delen naar de classificatie van deelgebouwen en die van de adressen. Er wordt een vergelijking gemaakt ,in aantallen, met de gevonden kadasterclassificaties.

Tabel 8-9 Voor de gegevens set Apeldoorn per woningtype het aantal deelgebouwen dat geclassificeerd is.

Afstuderen		Kadaster		
Code	Aantal deelgebouwen	Aantal deelgebouwen	Code	
#	2454	2454	-3	(niet wonen)
-	971	957	-1	(geen classif)
A	728	666	10-19	(appartement)
V	10827	10634	20-29	(vrijstaand)
T	6376	6598	30-39	(2-onder-1-kap)
E	4822	4848	40-49	(eindwoning)
M	12509	12530	50-59	(middenwoning)
S	715	715	98	(schuren)

Per classificatie en waardering (samen de sleutel tot de beslisregel die is uitgevoerd) het aantal adressen in de testset. De beslisregels die op de adressen zijn uitgevoerd, zijn in vet en donkerblauw weergegeven.

Classificatie	waardering	Aantal	Classificatie	waardering	Aantal	Classificatie	waardering	Aantal
#	0	3983	M	1	6060	T	1	4037
*	0	1979	M	2	3769	T	2	914
-	0	316	M	3	71	T	3	296
A	1	4691	M	4	4	T	4	1990
A	3	7184	M	5	4648	T	5	63
A	5	2469	M	6	1692	T	8	123
A	**11**	**503**	M	7	669	T	9	50
E	1	3059	M	8	325	V	1	9304
E	2	1536	M	9	215	V	2	1212
E	3	93	**M**	**10**	**2**	V	3	243
E	8	156	**M**	**15**	**88**	V	4	138
E	**16**	**1936**				V	8	224
E	**17**	**1907**						

Tabel 8-10 Voor de gegevensset Apeldoorn, per classificatie en waardering, het aantal adressen dat geclassificeerd is.

Alle classificaties opgeteld voor elke hoofdklasse geeft de volgende tabel, waarin gelijk een vergelijking met het kadaster wordt gegeven in aantallen.

Tabel 8-11 Voor de gegevensset Apeldoorn per woningtype het aantal adressen dat geclassificeerd is.

Afstuderen		Kadaster		
Code	Aantal ACN	Aantal ACN	Code	betekenis
#	3983	3809	-3	(niet wonen)
-	316	625	-1	(geen classificatie)
*	1979	1255	-2	(geen gebouw)
A	14847	14443	10-19	(appartement)
V	11121	11831	20-29	(vrijstaand)
T	7473	7608	30-39	(2-onder-1-kap)
E	8687	8664	40-49	(eindwoning)
M	17543	17714	50-59	(middenwoning)

Met deze gegevens kan het gemiddelde aantal woningen in een rij worden berekend. Als we ervan uitgaan dat elke rij twee eindwoningen heeft dan zijn er 4343.5 (8687 ÷ 2) rijtjes woningen in de kadastrale gemeente Apeldoorn. Dit betekent dat er in Apeldoorn gemiddeld 6 (17543 ÷ 4343.5 +2) woningen in een rij zitten.

VISUALISATIE

Aan het einde van de classificatie wordt er een tabel met attributen vanuit de database weggeschreven naar een bestand wat ingelezen kan worden in ArcView. De reden hiervoor is dat dan de resultaten in een kaartje gevisualiseerd kunnen worden. Naast de woningen worden ook kaartlagen met adrescoördinaten, percelen en tekst ingelezen in ArcView. Als laatste wordt een legenda gemaakt. Het resultaat van dit visualisatieproces is een kaart. Dit laatste deel hoort niet bij de automatische methode en dient alleen om de resultaten visueel te inspecteren en te presenteren. De gepresenteerde kaarten bevatten de classificaties per huis en/of de classificaties per adres. De gevonden hoofdklassen zijn bepaald met de classificatieregels, die zijn beschreven in hoofdstuk 8, en bevat de (hoofd)klassen: appartement, middenwoning, hoekwoning, twee-onder-een-kap en vrijstaand.

ArcView

Legenda

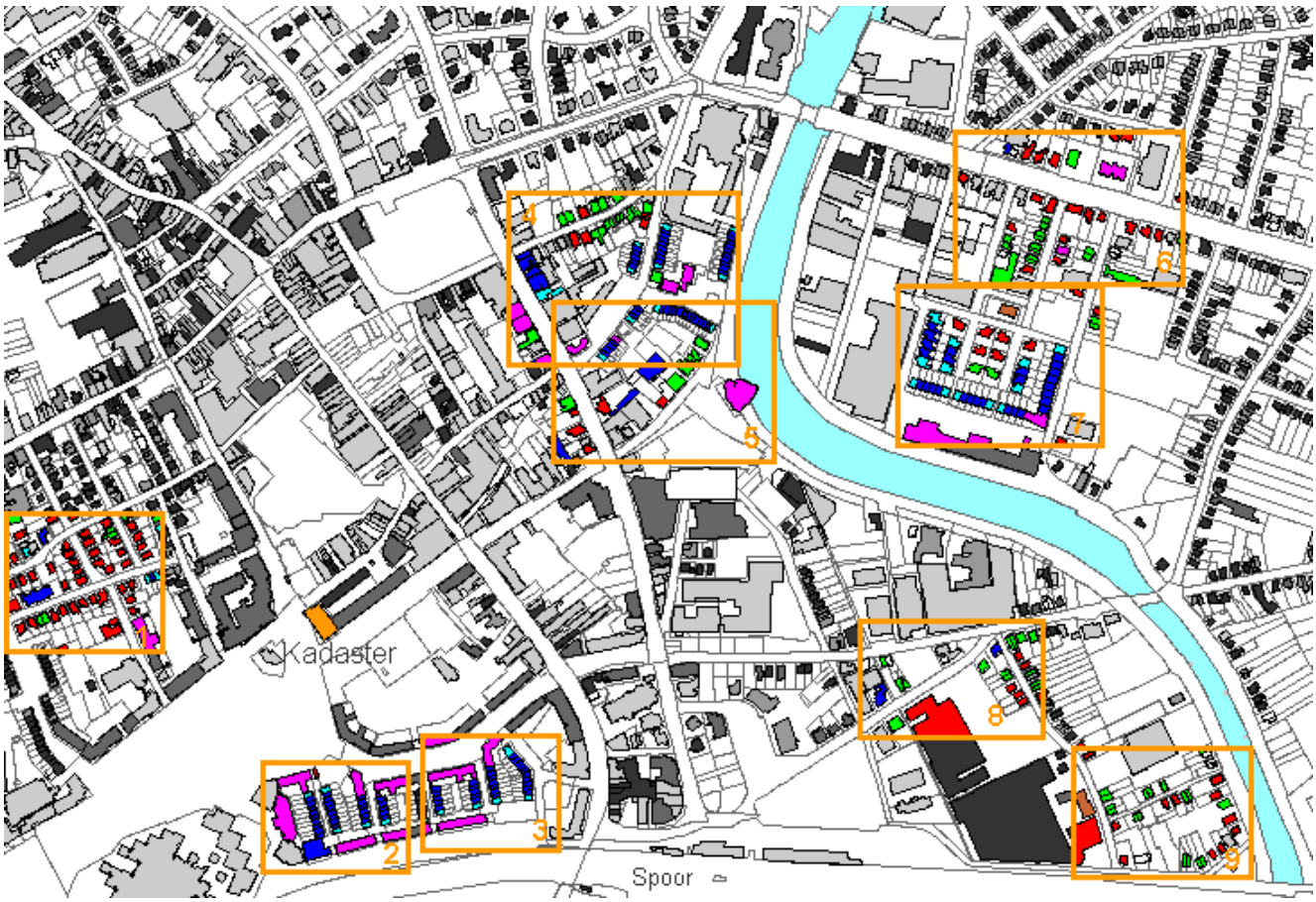
Voorbeeldkaarten

Foto's

9.1 INLEIDING

De resultaten van de classificatie van de dataset Apeldoorn zijn ingelezen in een ArcView project. Met behulp van een visuele inspectie op de kaart is er een eerste indruk gekregen van de resultaten. Het zag er veelbelovend uit, alhoewel er hier en daar toch wel kleine fouten te vinden zijn.

In de komende pagina's zullen er een aantal kaartjes getoond worden die gemaakt zijn met de dataset van Apeldoorn. Bij de eerste negen kaartjes ben ik zelf in het gebied geweest en heb ik (digitale) foto's gemaakt. Op vier van deze kaarten zijn vijf locaties uit de validatieset (zie hoofdstuk 10) terug te vinden. Deze locaties waren op de lijst voor een bezoek terecht gekomen. De bezoeklijst bestaat uit gebouwen uit de valideringsset waarvan, op basis van de informatie op de kaart en eigenaargegevens, niet met zekerheid gezegd kan worden of het juist geclassificeerd is. Dit kan bijvoorbeeld komen, doordat er een adres te veel in een perceel ligt. Of dat het aantal adressen in een gebouw toch net teveel leek te zijn voor rijtjeshuizen en het dan waarschijnlijk twee woonlagen op elkaar zijn. Alleen een visuele inspectie op de kaart bracht niet direct uitsluitsel of het adres goed of fout geclassificeerd was.



Overzichtskaart 9 - 1 Centrum van Apeldoorn, ten noorden van de spoorlijn

9.2 LEGENDA BIJ DE KAARTEN

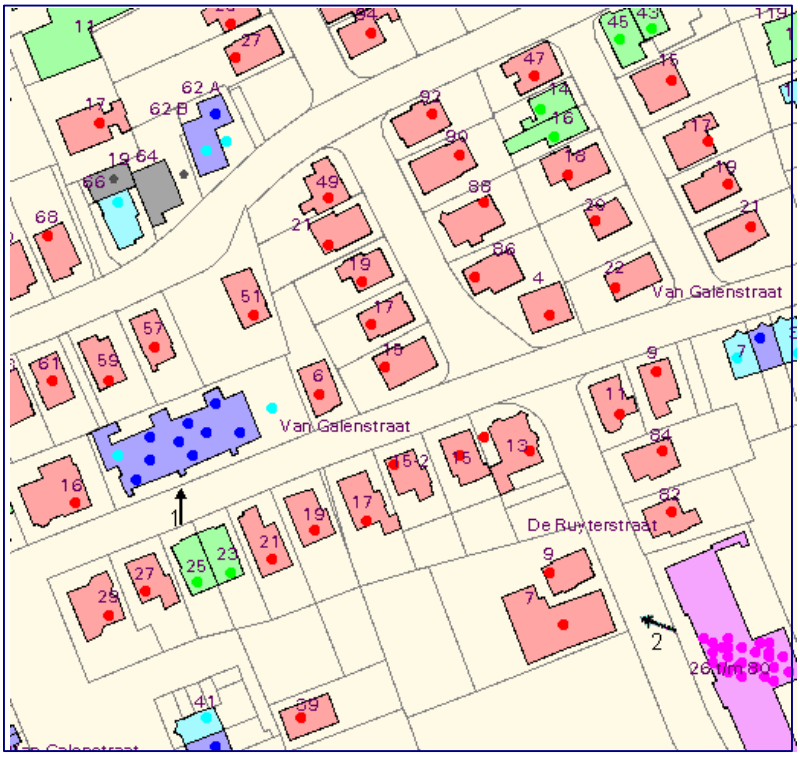
Legenda bij de nu volgende kaarten:

De ACN en de deelgebouwen hebben bijna dezelfde klassen en eenzelfde kleur, alleen zijn de adressen feller gekleurd. ACN zijn punten in de kaart, terwijl de deelgebouwen (het resultaat van de overlay) vlakken zijn. De adressen hebben nog een extra klasse ten opzichte van de deelgebouwen en dat is de klasse waarbij er wel een adres op een perceel valt, maar er niet een deelgebouw aan het adres te koppelen valt. Dit kan komen doordat het bijbehorende gebouw niet de topologieconstructie doorgekomen is of doordat deze nog niet in het invoerbestand zit. Dit laatste is een gevolg van het verschil in actualiteit tussen de invoer bestanden. Er is wel geprobeerd om dezelfde actualiteitsdatum te gebruiken, maar bij ACN wordt dit niet per adres bijgehouden.

9.3 KAARTEN MET FOTO'S

Nu volgen een aantal pagina's vol met voorbeeldkaarten en bijbehorende foto's zodat de lezer een beeld kan vormen bij de kaarten.

Opmerkingen bij deze kaart en foto's:
In deze kaart zijn alle type woningen terug te vinden, maar voornamelijk vrijstaande woningen (rood). In deze kaart zijn ook twee adressen terug te vinden die in de validatieset zaten en foutief zijn geclassificeerd. Het blauwe gebouw waarvan de eerste foto 1A en 1B zijn genomen zijn boven- en benedenwoningen en vallen in de classificatie dus onder appartementen en niet zoals hier weergegeven uit middenwoningen en hoekhuizen. Uiteindelijk zijn hier dus elf adressen fout geclassificeerd. Het tweede fout geclassificeerd adres is een kerkje en deze zou dus niet een cultuurcode wonen moeten hebben. Dit is een fout in AKR.



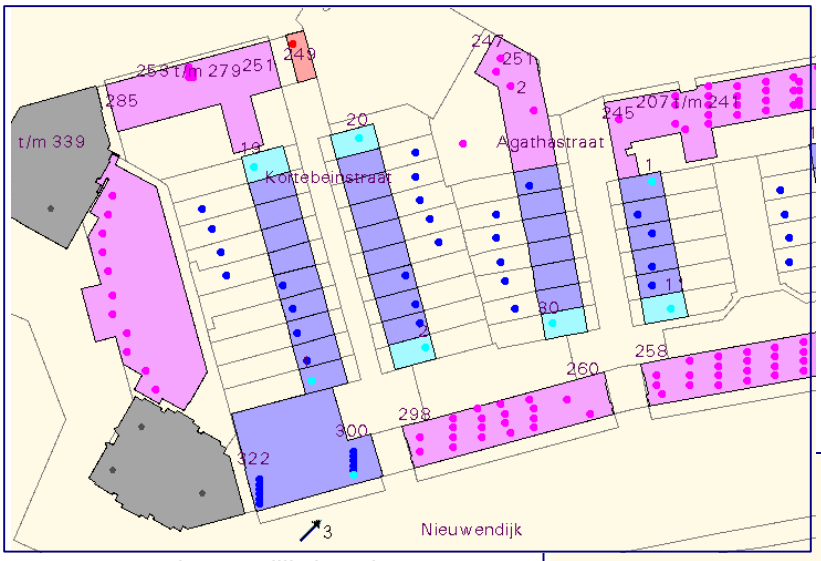
Kaart 9 - 1 Van Galenstraat

Foto 1 A

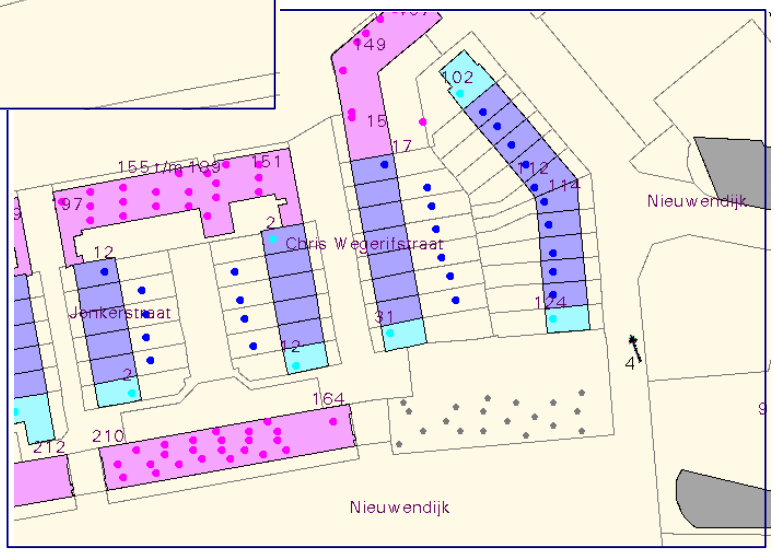
Foto 1 B

Foto 2 Kerk

Opmerkingen bij kaartje twee en drie en bijbehorende foto's:
Op deze kaartjes zijn voornamelijk rijtjeshuizen en appartementen te zien. In kaart 2 is de verwisseling van appartement met rijtjeshuis te zien (foto3). Deze verwisseling is een fout die veroorzaakt wordt door de classificatie regels. De oppervlakte verhouding deelgebouw/acn is groter dan 30 m^2. Bij een na-controle van deze situatie bleken ook fouten in LKI hier invloed op gehad te hebben.
In kaartje 3 is te zien dat er een appartementsgebouw mist. Deze fout is dus het gevolg van een fout in de LKI.



Kaart 9 - 2 Nieuwendijk (west)



Kaart 9 - 3 Nieuwendijk (Oost)

Foto 3 Nieuwendijk appartementen

Foto 4 Nieuwendijk rijtjeswoningen

Opmerkingen bij kaartje 4 en bijbehorende foto's:

Op deze kaart zijn weer alle classificaties vertegenwoordigd. Deze keer een voorbeeld van een appartement (foto 5) en een voorbeeld van een rijtjeswoning (foto 6). Opvallend in deze kaart is het groene gebouw met precies 2 huisnummers en twee adressen in hetzelfde perceel als het appartementen complex dat ook in de foto wordt weergegeven.
Op foto 6 kunnen we nog net twee van dezelfde balkonnetjes als in foto 5 terug vinden. Dit groene gebouw is dus ook gewoon onderdeel van het appartementen complex ernaast. Omdat het maar twee adrescoördinaten bevat classificeert het algoritme deze adressen als een twee-onder-één-kap. Deze classificatie fout had voorkomen kunnen worden als de ACN juist gepositioneerd zouden zijn.

Kaart 9 - 4 Wiepkingstraat

Foto 5

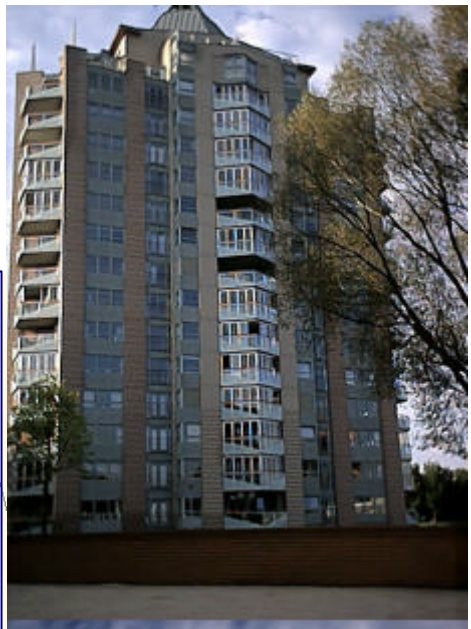
Foto 6

Foto 7 Kanaalzicht

Opmerkingen bij kaartje vijf en bijbehorende foto's :

Op deze kaart weer twee voorbeelden met foto's: een torenflat (appartementen) en een twee-onder-één-kap woning. Deze hoge torenflat is in de wijde omgeving goed zichtbaar. Ook vanaf het centraalstation en vanuit het kadastergebouw is dit appartementencomplex te zien. Opvallend in de twee onder een kap woningen zijn, dat er een aantal LKI huisnummers zijn waar geen adrescoördinaat bij hoort.

Foto 8 C Achterkant torenflat



Kaart 9 - 5

Foto 8 B Voorkant torenflat

Foto 8 A Ingang torenflat

Foto 9

Opmerkingen bij kaartje zes en bijbehorende foto's:

Op dit kaartje zijn voornamelijk vrijstaande woningen en twee-onder-één-kap woningen te vinden. Foto 9 laat een villa zien, die binnen de classificatie dus een vrijstaande woning zou moeten worden. Maar dit gebouw, dat op één perceel staat, heeft twee adressen en is daarom een twee onder een kap geworden. Op foto 10 is duidelijk de helft van een twee onder een kap terug te vinden. Deze woning is nog niet geclassificeerd omdat ook hier weer een adres te veel in het gebouw staat. Dit is dus een gevolg van een fout in het bronbestand ACN. Bij het veldbezoek was er geen brievenbus met 5A te vinden. Deze laatste woning was ook weer een van de adressen uit de validatieset die foutief of nog niet geclassificeerd was.

Kaart 9 - 6

Opvallend verder in deze kaart zijn de twee grote groene gebouwen. Dit zijn weer gebouwen met twee adressen, in een groot gebouw. Tijdens het veld bezoek heb ik kunnen zien dat het huis nummer 7 inderdaad een woning is, maar dat aan deze woning een bedrijfje vast zit. Hier valt de grens tussen woning en bedrijf dus niet op de perceelgrens.

Foto 10

Foto 11

Opmerkingen bij kaartje zeven en bijbehorende foto's :

Nogmaals een kaartje waarop alle types voorkomen, met daarbij foto's van appartementen en rijtjeshuizen. Foto 13 toont een voorbeeld waarbij een rijtjeshuis aan een appartementencomplex grenst.

Kaart 9-7

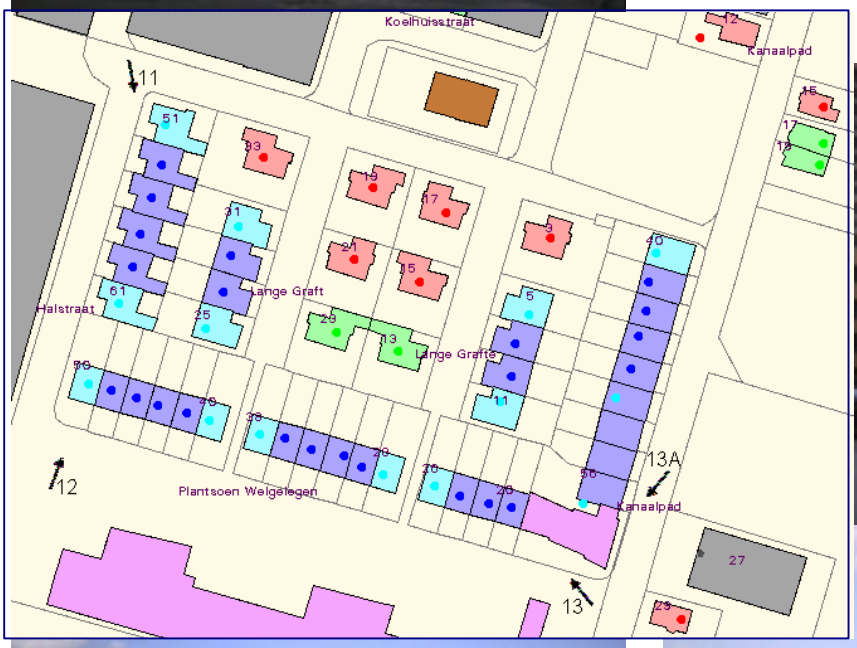


Foto 13A

Foto 12

Foto 13

Foto 14

Het appartementencomplex, onder in de kaart, genomen vanaf de andere kant van het kanaal. Dit is hetzelfde gebouw als zichtbaar is op foto 11

Opmerkingen bij kaartje acht en bijbehorende foto's:

Foto 15

Foto 16

Kaart 9 - 8

Foto 16 is weer een foto van een woning uit de validatie waarvan het vermoeden bestond dat het fout geclassificeerd was. Hier bleek echter dat het gebouw niet als woning dienst deed en dus inderdaad als cultuurcode niet wonen heeft. De andere foto's zijn voorbeelden van een vrijstaande woning en twee-onder-één-kap.

Foto 17

Foto 18

Kaart 9 - 9

Opmerkingen bij kaartje negen en bijbehorende foto:

Een kaartje met voornamelijk juist geclassificeerde vrijstaande en twee onder één kap woningen.

Opvallend in dit kaartje zijn de lichtgrijze, nog niet geclassificeerde, twee-onder-één-kap woningen.

Foto 19

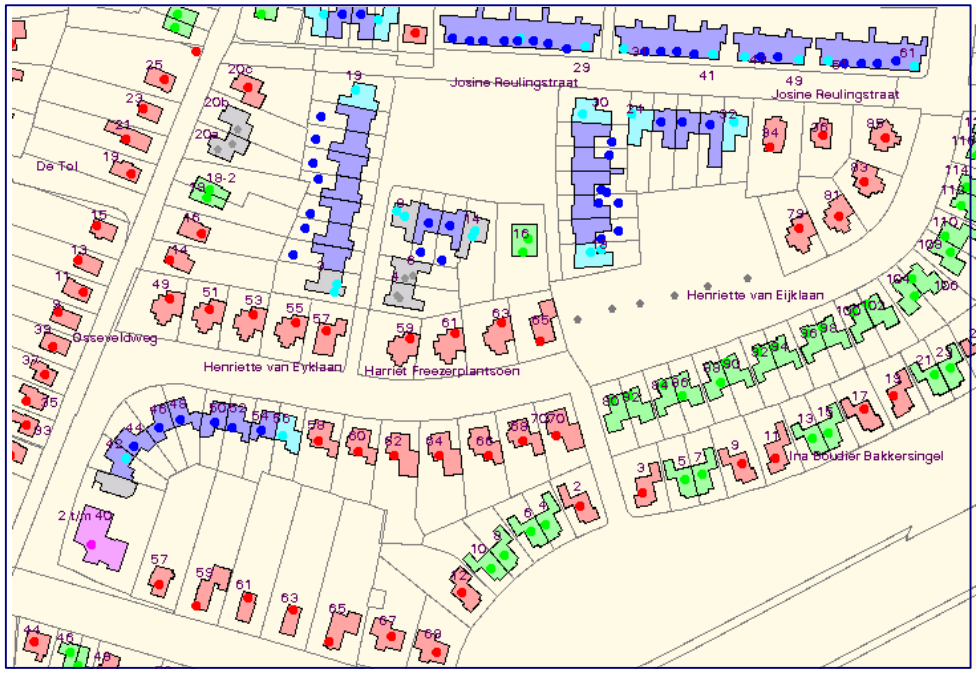
Weer valt op dat er fouten in het ACN bestand zitten. Er zijn in deze kaart eigenlijk drie soorten ACN fouten te ontdekken:

- ACN buiten het gebouw,
- ACN in het verkeerde deel van het gebouw,
- of een ACN teveel in het gebouw.

Tot zo ver de voorbeeld kaarten waarbij een aantal foto's zijn genomen zodat de lezer een beeld krijgt bij de kaarten.

9.4 VOORBEELDKAARTEN

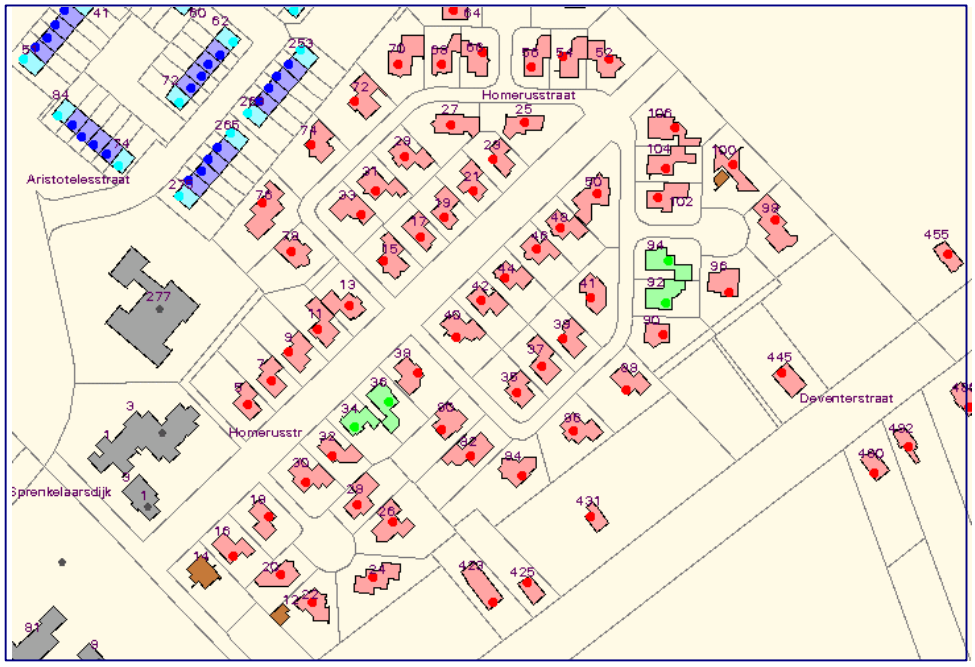
Nu volgen een aantal voorbeeld kaartjes uit de dataset Apeldoorn. Kaarten met voorbeelden van juist geclassificeerde rijtjeshuizen, vrijstaande woningen en twee-onder-één-kap woningen.

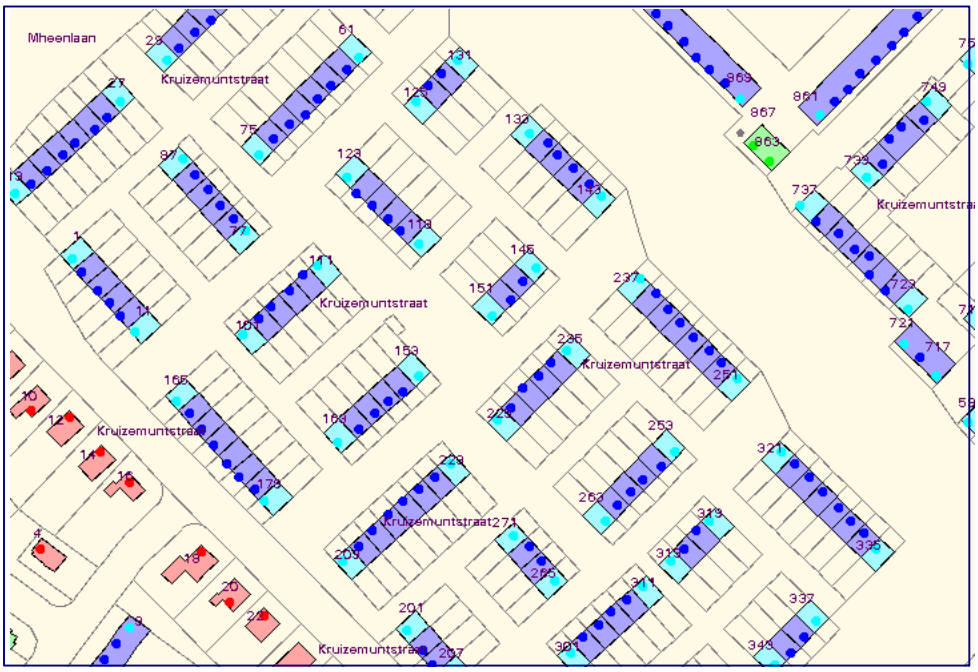


Kaart 9 - 10

In kaarten 10 en 11 zijn de vrijstaande woningen (rood) en de twee-onder-één-kap (groen) goed vertegenwoordigd. In kaart 10 is bovenin een voorbeeld van verhuurde rijtjeshuizen (blauw) terug te vinden.

Kaart 9 - 11

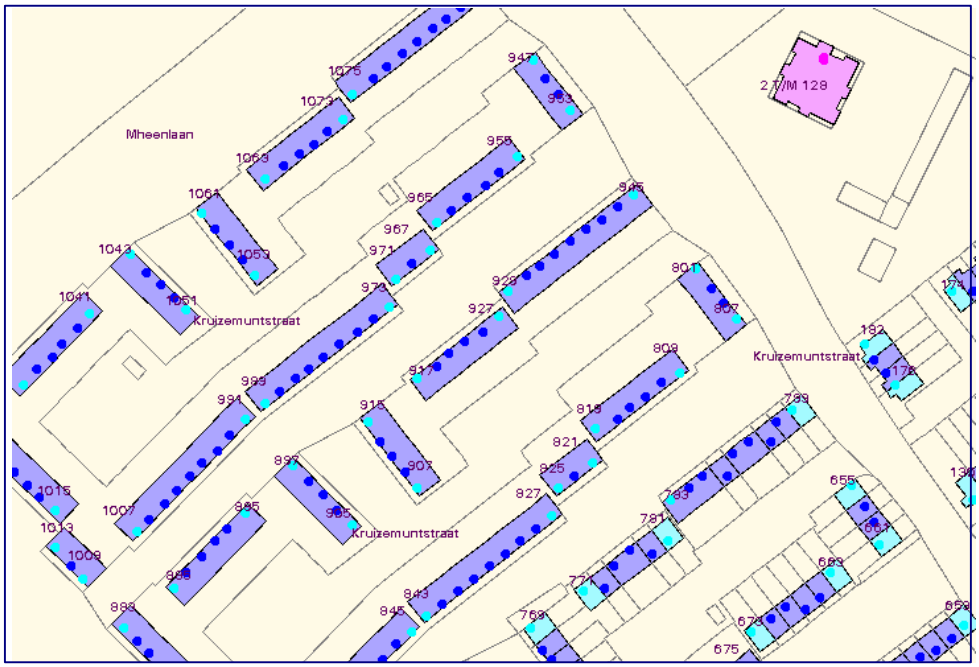




Kaart 9 - 12 Voorbeeld rijtjes woningen koophuizen

Kaart 12 en kaart 13 tonen allebei rijtjeshuizen, maar alleen in kaart 12 zijn de hoekwoningen als deelgebouwen terug te vinden. Dit zijn dan ook koopwoningen. Kaart 13 toont juist voornamelijk huurhuizen. Hier is te zien dat de adressen van de verhuurde hoekwoningen wel terug gevonden zijn door het hoogste / laagste huisnummer van een rij, de classificatie hoekwoning te geven.

Kaart 9 - 13 Voorbeeld rijtjeswoningen huurhuizen.



De laatste twee voorbeeldkaarten van dit hoofdstuk: 14 en 15.

Kaart 9 - 14 nog een voorbeeld van rijtjeswoningen

In kaart 14 valt onder andere het gebouw rechtsbovenaan op. Aan dit gebouw valt te zien dat het niet altijd goed gaat om hoekwoningen van huurhuizen terug te vinden. De methode van het hoogste en laagste huisnummer als eindwoningen van een rij te classificeren gaat vooral fout als het gebouw in twee verschillende straten gelegen is. Ook komt het voor dat de ACN niet juist geplaatst is. Alleen veldbezoek kan aantonen of dit gebouw juist of fout is geclassificeerd.

In kaart 15 valt de classificatie van rijtjeswoningen aan de Oostkreek op. Eén deel is koop, het andere deel is huur. De ACN behorende bij de huisnummers 1 t/m 9 en 2 t/m 10 liggen hier per vijf adressen op één locatie. Deze adressen hadden als appartementen geclassificeerd moeten worden. Hier zit een fout in de classificatieregels. In het volgende hoofdstuk (10) wordt ingegaan op alle soorten fouten.

Kaart 9 - 15 In deze kaart zijn voorbeelden van vrijstaande, twee-onder-één-kap en rijtjeswoningen te zien

KWALITEITS-CONTROLE

Binnen het project dat bij het kadaster loopt vormt de controle van het resultaat een belangrijke rol. Er zijn twee testsets gedraaid. Eén van heel de gemeente Apeldoorn, en één van stad Utrecht. De testset van Apeldoorn is de testset die ook voor dit afstuderen gebruikt is. Als kwaliteitscontrole is een valideringsset gemaakt die bestaat uit een genormeerde selectie van ongeveer 1000 woningen. Deze zijn eerst op de kaart en eventueel, als er twijfel bestaat over de juistheid van de classificatie, in het veld gecontroleerd.
De lijst met 1000 woningen die gecontroleerd zijn heb ik naast de resultaten van de afstudeerclassificatie gelegd en deze vergeleken. In dit hoofdstuk worden de resultaten hiervan kort besproken.

Selectie van controle woningen

Soorten fouten

Mogelijke oorzaken van gevonden fouten

10.1 INLEIDING

Gedurende het einde van week 39 van het jaar 2000 ben ik bezig geweest met de kwaliteitscontrole van de resultaten van het classificatieproces. Er zijn binnen het kadasterproject duizend woningen geselecteerd waarvan het grootste deel op de kaart gecontroleerd is. Daarnaast zijn er 79 woningen in het terrein gecontroleerd. Het veldwerk is gebeurd in week 40 van 2000. Ik heb zelf 5 adressen in de buurt van het hoofdkantoor van het Kadaster bezocht. Bij de controle is alleen gekeken naar de hoofdklassen. Deze controlelijst is ook gebruikt bij de controle van de classificaties die tijdens het afstudeeronderzoek zijn uitgevoerd. Alle fout gevonden classificaties en verschillen tussen de kadasterclassificatie en afstudeerclassificaties zijn nogmaals bekeken.

10.2 CONTROLE VAN DE WONINGEN MET BEHULP VAN EEN VALIDERINGSSET

Uit alle Kadasterclassificaties zijn in totaal rond de duizend adressen geselecteerd, waarbij elke klasse vertegenwoordigd is. Van deze ruim duizend adressen zijn de gevonden classificaties stuk voor stuk gecontroleerd. De aantallen per klasse zijn ongeveer de wortel uit het totaal keer dat een klasse voorkomt, dus uit een klasse van 9000 worden er ongeveer 95 geselecteerd, terwijl uit een klasse van 12 er 4 worden geselecteerd. Er zijn totaal 1187 records gecontroleerd. Deze records zijn door de deelnemers van het kadasterproject gecontroleerd. Ik heb er zelf ook ruim 200 gecontroleerd. Voor vijf adressen ben ik in het terrein langs geweest. De controle bestond onder andere uit het visueel controleren van de situatie met behulp van een kaart op het beeldscherm. Verder werd in veel gevallen de eigenaar bij het perceel gezocht en zo gecontroleerd of het adres een woonadres is. Dit was niet altijd nodig omdat het soms direct zichtbaar was of iets goed of fout geclassificeerd was.

Veldwerk

Zowel op vrijdag 29 september 2000 als op donderdag 5 oktober 2000 heb ik een aantal foto’s van het testgebied genomen. Op donderdag 5 oktober ben ik zelfs langs vijf van de adressen gegaan die in de valideringsset terecht waren gekomen. Deze foto’s, en de situaties waar ze bij horen, zijn terug te vinden in hoofdstuk 9.

De set met de ruim duizend adressen die zijn gecontroleerd wordt de valideringsset genoemd. De set met alle niet-goede antwoorden en alle verschillen in hoofdklassen tussen de valideringsset en de afstudeerclassificatie wordt de vergelijkingsset genoemd. Nu volgt een overzicht van de aantallen per klasse die gecontroleerd zijn.

Tabel 10-1 Aantallen in de verschillende sets

Classificatie	Totaal aantal	In de valideringsset. Aantal	% goed	Goed	Niet Goed	Geen bezoek	Wel bezoek	Vergelijkingsset Aantal	Goed	Fout	nog onduidelijk	-
#	3970	70		50	20	61	9	22	2	14	6	
*	316	89		39	50	85	4	84	34	49	1	
-	2482	18		6	12	12	6	18	6	8	3	1
A	14357	208	94.23	196	12	188	20	18	6	8	4	
E	11121	212	90.09	191	21	205	7	26	5	17	4	
M	7473	258	94.19	243	15	244	14	15	0	10	5	
T	8775	161	**73.91**	119	**42**	147	14	43	1	**34**	7	1
V	17455	171	91.81	157	14	166	5	14	0	11	2	1

De gevonden percentages in tabel 10-1 zijn berekend door uit te gaan van de resultaten van de valideringsset van het kadaster en daar de verschillen van te nemen met de classificaties die in dit afstudeerproject zijn bepaald. De foute classificaties zijn per hoofdklasse opgeteld. Omdat het niet een willekeurige steekproef is maar de kleinere klassen waar we al minder zeker van de classificatie zijn een relatief grote vertegenwoordiging hebben. Als we alle classificaties zouden controleren zullen de percentages hoger liggen.

Met een zelfde classificatie wordt bedoeld dat wanneer een adres de afstudeerclassificatie “A” heeft dat dan de kadasterclassificatie tussen 9 en 20 valt. “V” houdt in een Kadasterclassificatie tussen 19 en 20.
“T” houdt in een Kadasterclassificatie tussen 29 en 40.
“M” houdt in een Kadasterclassificatie tussen 39 en 50.
En “E” houdt in een Kadasterclassificatie tussen 49 en 60.

Verschillen met het kadaster	aantal	verbeterd
Nog onbepaald	7	
FOUT	54	15
GOED	54	- 6
- (leeg)	1	
Hetzelfde als het kadaster (zitten niet in de vergelijkingsset)	**aantal**	
-	2	
Nog onbepaald	25	
FOUT	97	

Tabel 10-2 Opmerkingen over de vergelijkingsset, dit zijn de aantallen in het laatste stuk van tabel 10-1.

Een overzicht van de verschillen tussen de deelgebouwen classificaties van het kadaster en dit afstuderen. Alleen de verschillen die een andere hoofdklasse hebben, zijn hier weergegeven.

Tabel 10-2 De opvallende verschillen tussen de gevonden classificaties van het kadaster en het afstuderen.

Classificatie	Waardering	Aantal	Kadaster classificatie
-	0	**149**	30
-	0	**79**	31
-	0	1	32
-	0	**36**	39
-	0	11	41
-	0	11	42
-	0	11	44
-	0	**33**	52
A	3	3	50
A	5	10	-1
A	5	2	22
A	5	1	30
A	5	1	31
A	5	12	35
A	5	2	44
A	5	**27**	50
A	5	2	55
A	5	2	57

Classificatie	Waardering	Aantal	Kadaster classificatie
E	2	**34**	39
E	3	5	34
E	3	2	39
M	4	2	-1
M	4	2	39
M	5	4	-1
M	5	13	35
M	6	4	35
M	7	1	31
M	7	5	35
M	8	2	39
M	9	13	-1
T	3	**32**	42
T	4	59	-1
T	4	8	22
T	4	3	24
T	9	23	-1
V	3	84	-1
V	4	122	-1

Alle aantallen groter dan 25 die bij het kadaster een andere classificatie hebben zijn geaccentueerd. De getallen waar hier nog een “–“ staat is het gevolg van strengere eisen binnen de classificatieregels ten opzichte van de classificaties van het kadaster. De verschillen binnen classificatie A5 (50), E2 (39) en T3 (42) zullen later besproken worden.

De goede classificaties van het kadaster die in de vergelijkingset terecht waren gekomen, waren onder andere het gevolg van classificatie regel A 11 waar een tikfout in zat, dit zijn al 21 classificaties. De andere waren de 6 classificatie ook door een tikfout, classificatie E15, wat M15 had moeten zijn. Verder heeft het kadaster alle adressen die alleen op een perceel lagen waar geen gebouw aan te koppelen valt als vrijstaande huizen te classificeren. Dit gaat soms goed, maar ook rijtjeswoningen waar het gebouw mist kregen deze classificatie, in het afstudeerproject is er daarom voor gekozen om deze classificaties niet uit te voeren en gewoon als ‘*’ te laten staan. Dit zijn nog eens 21 van de 54 goede kadasterclassificaties die in de vergelijkingsset terecht zijn gekomen. Dus er zijn uiteindelijk maar 6 door het kadaster goed geclassificeerde woningen binnen dit

afstudeerproject fout geclassificeerd. Vijf hiervan zijn fouten in classificatie A5.

Tabel 10-3 De vergelijkingset en de uitkomst van de controles per classificatieregel

Opsplitsing	Totaal in Apeldoorn	Aantal in valideringsset	aantal in de vergelijkingsset	goed	fout	Nog onbekend	-
A1	4691	68	1	0	0	1	
A3	7184	85	6	0	5	1	
A5	2469	53	10	6	2	2	
A11	13	2	1	0	1		
E1	3059	55	1	0	1	0	
E2	1536	46	6	0	6	0	
E3	93	4	1	0	1	0	
E8	156	12	0	0	0		
E15	242	12	10	5	4	1	
E16	1854	43	4	0	.	1-3	
E17	1835	40	4	0	.	1-3	
M1	6060	54	1	0	1		
M2	3769	47	0				
M3	71	2	0				
M4	4	0					
M5	4648	69	2	0	1	1	
M6	1692	38	4	0	3	1	
M7	669	28	8	0	5	3	
M8	325	18	0				
M9	215	2	0				
M10	2	0					
T1	4037	58	2	0	1	1	
T2	914	14	2	0	2		
T3	296	13	7	0	7		
T4	1990	55	23	2	**17**	3	1
T5	63	4	1	0	0	1	
T8	123	11	2	0	1	1	
T9	50	6	6	0	5	1	
V1	9304	96	4	0	3	1	
V2	1212	35	1	0	1		
V3	243	20	5	0	3	1	1
V4	138	5	4	0	4		
V8	224	15	0				

Opvallend is het aantal fouten in A5, E15 en T4. E15 hoort niet te bestaan. Dit is het gevolg van een tikfout. De E had een M moeten zijn. Doordat al deze adressen al een E hadden hebben ze niet meer meegedaan in de bepaling van het hoogste en laagste nummer. Het is dus toeval dat er eindwoningen goed geclassificeerd zijn. Dit komt omdat vooral die einden van een rij waar twee adressen in vallen en nog niet geclassificeerd waren, in de classificatie M15 terecht zijn gekomen. De verbetering aan A11 en M15 zijn uitgevoed en meegenomen in de resultaten die genoemd zijn in hoofdstuk 8.

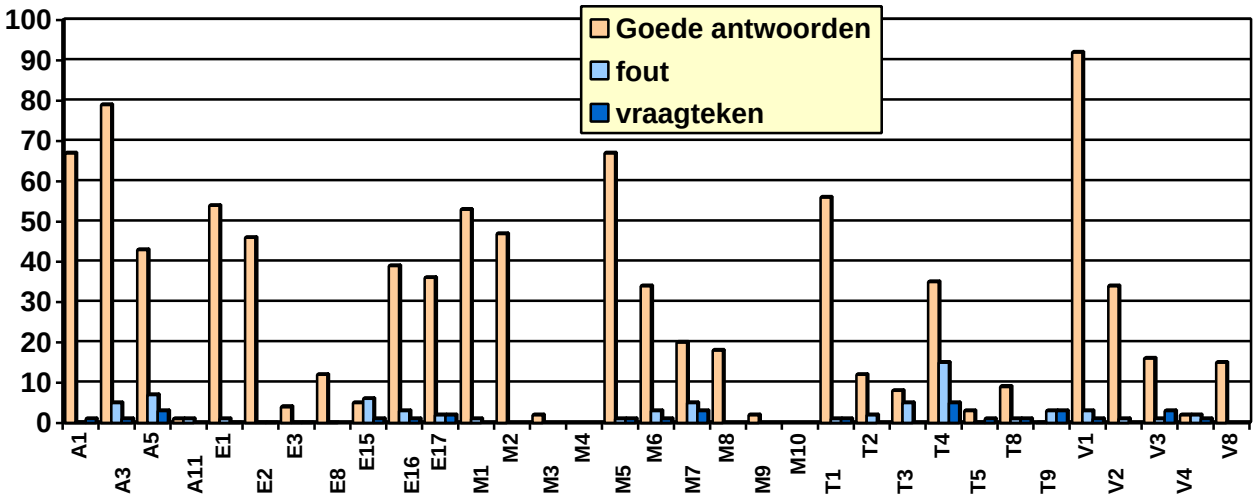


Diagram bij tabel 10-2. Van alle classificaties in de valideringsset een onderverdeling naar goed, fout en nog onbepaald.

Nu wordt ook zichtbaar waar verhoudingsgewijs de grootste fouten liggen. De grootste fouten zitten dus in E15 (meer fout dan goed) en T4. De fouten in E15 waren te verklaren en het classificatieprogramma is daarop aangepast.

De fouten in classificatie T4 zijn niet direct te verklaren. Daarom halen we de classificatie weer even terug:

T4 Dit zijn als het ware de huursituaties van de twee-onder-één-kap woningen. Wanneer het een vrijstaand gebouw betreft dat niet is opgedeeld, maar waar precies twee adressen in het gebouw vallen dan wordt het een twee-onder-één-kap woning.

Bij het controleren, op de kaart, van deze fouten blijkt dat vaak een soort dubbele adressen bij een gebouw horen. Zoals huisnummer 2 en 2a of 10 en 10-I. Dit zouden gewoon één en hetzelfde adres kunnen zijn. Het beste zou zijn om in deze gevallen ook de huisnummers van de twee adressen te contoleren. Er zou dan in principe moeten gelden dat of het nummer verschillend is, of dat beide een toevoeging hebben die verschillend is.

De verschillen binnen classificatie E2 (39) en T3 (42). Alle classificaties die in E2 terecht zijn gekomen zijn goed, de kadaster classificatieregel was te veel versoepeld ten opzichte van de ideale cases. De verschillen tussen in classificatie regel T3 waren meestal het gevolg van fouten in LKI of ACN en beide classificaties waren in de testset meestal fout. Dus deze classificatieregels zijn erg gevoelig voor fouten in de bronbestanden.

10.3 SOORTEN FOUTEN

Tot nu toe zijn de fouten per classificatieregel bekeken en gecontroleerd. Wat bij die controles opviel was dat de gevonden fouten verschillende oorzaken hadden. Deze oorzaken zijn in de volgende vier hoofdklassen ingedeeld:

1 Fouten in LKI kunnen inhouden dat er geen (gesloten) gebouw is of dat lijnen die het gebouw moeten vormen niet de goede codes hebben, waardoor het gebouw niet sluit. Ook is het nog mogelijk dat een gebouw maar gedeeltelijk is ingemeten, en dus überhaupt niet sluit. Ook blijkt er op sommige plekken een verschuiving tussen het lijnen bestand en het percelen bestand te zijn, waardoor gebouwen versneden worden door een perceelgrens, terwijl dit niet zou moeten.

2 Fouten in AKR, zitten vooral in fouten in de cultuurcode.

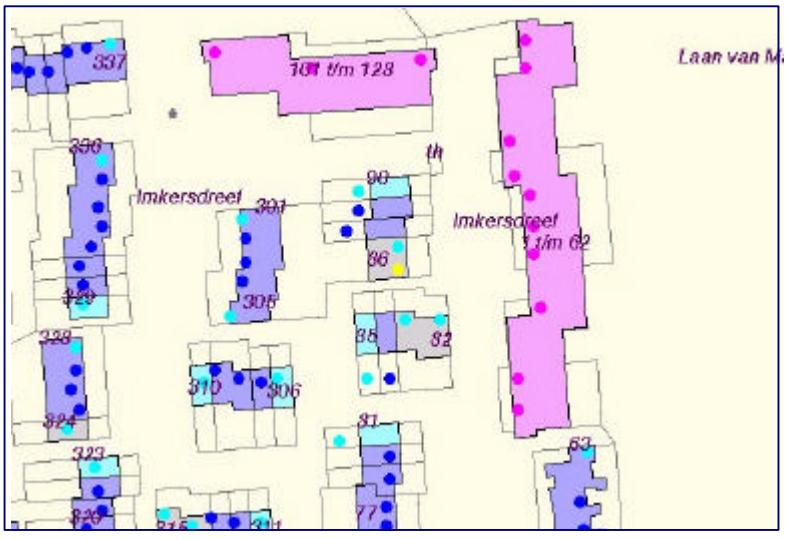
3 Fouten in ACN, verkeerd geplaatste ACN, twee keer hetzelfde adres, maar verschillende schrijfwijze. Een voorbeeld in de valideringsset hiervan waren de adressen met alleen een verschillend huisnummer plus toevoeging: 12, 12+2, 12+II. Waarschijnlijk zal voor al deze adressen gelden dat het ook werkelijk hetzelfde adres is en dat dit dus een vrijstaande woning zou moeten zijn.

4 Fouten in de classificatieregels. Dit zijn fouten veroorzaakt door een verkeerde volgorde of het loslaten van te veel kenmerken / eisen. Ook kunnen bepaalde uitgangspunten het gevolg zijn van fouten. Zoals het uitgangspunt dat een rij huizen in principe op rij worden genummerd. Dit blijk niet altijd het geval te zijn.

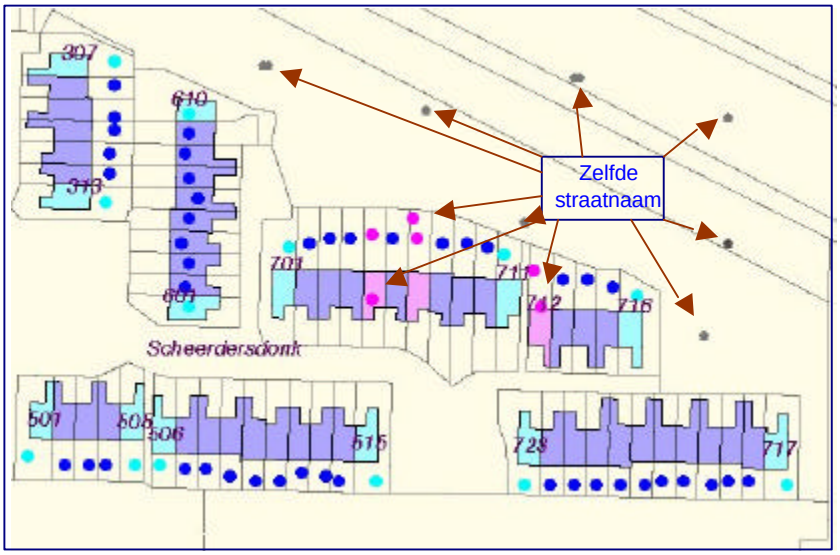
10.3.1 KAARTEN MET VOORBEELDEN VAN FOUTE CLASSIFICATIES

Nu volgen een aantal voorbeeld kaarten waarin fouten zitten. Deze zullen kort besproken worden.

In Kaart10 -1 is meerdere keren dezelfde fout terug te vinden. De gele stip is een adres dat in de valideringsset hoort. Maar in drie van de huizenblokken zijn er drie hoekhuizen gedefinieerd. Allen schijnen tot classificatieregel E15 te horen, wat dus eigenlijk M 15 had moeten zijn en deze zijn dus niet echt fout. (fouttype 4)



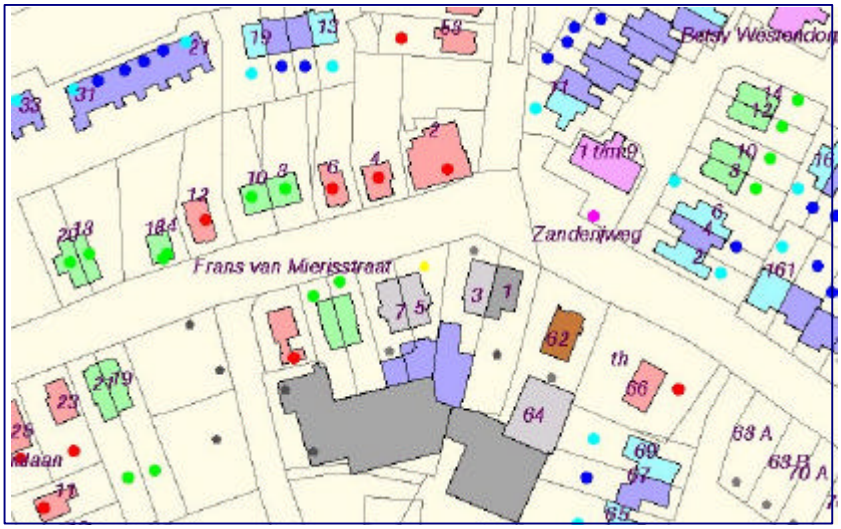
Kaart 10-1
Fout in de classificatieregels.



Kaart 10-2 Foute plaatsing ACN

In kaart 10 -2 zijn de gevolgen van foute ACN plaatsingen terug te vinden, Alle grijze en een aantal paarse stippen hebben als straatnaam Spinnersdonk. Deze hele buurt is dus op een heel verkeerde plek in de kaart terechtgekomen, waar ze fouten veroorzaken in de classificatie van andere adressen. Voor alle drie de paarse vlakken geldt dat er één paars punt is met één adres voor de Scheerdersdonk en de andere bevat er een aantal voor de Spinnersdonk. (fouttype 3)

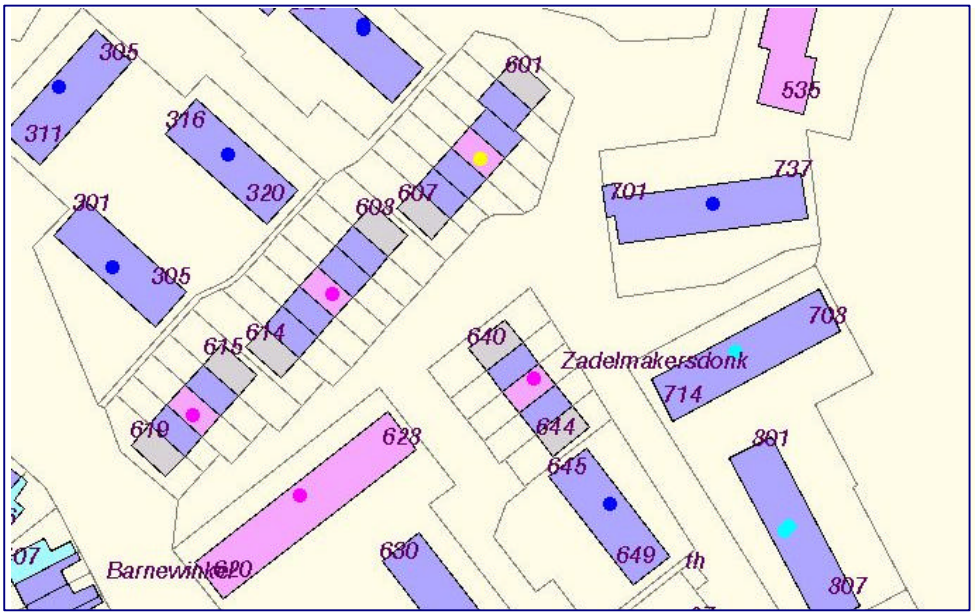
In kaart 10 -3 is te zien dat onder andere doordat de adressen weer buiten de gebouwen vallen er geen classificatie mogelijk is. Het perceel dat bij adres “Frans van Mierisstraat 3" hoort loopt onder het bedrijvenpand door. Hierdoor is er een deelgebouw aangemaakt dat op dit perceel valt. Het grote grijze gebouw zorgt er hier voor dat 5 woningen niet geclassificeerd kunnen worden. Dit is onder andere een fout van type 3, maar ook is het mogelijk dat een fout van type 1 meespeelt.



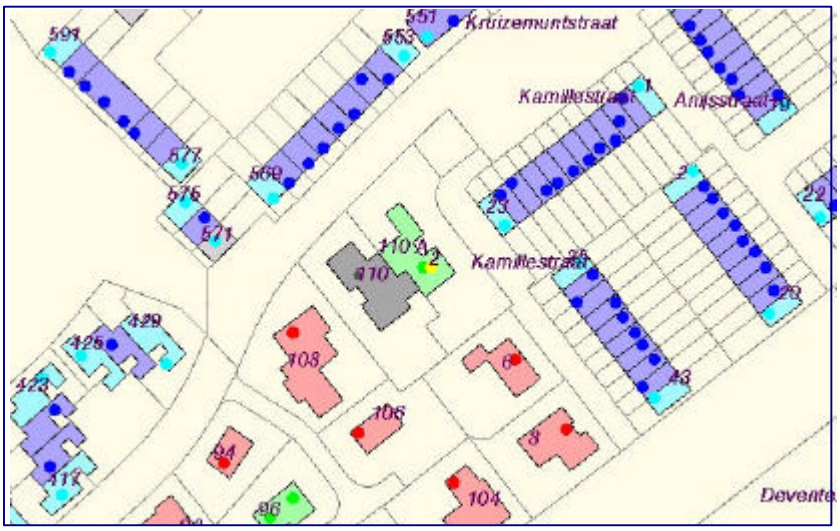
Kaart 10-3 Problemen met deelgebouwen als de ACN uit het gebouw valt.

Kaart 10-4 A: Fout geclassificeerde rijtjes.

In deze twee kaarten is zichtbaar dat soms de adressen van een rij woningen allemaal op dezelfde locatie terechtkomen. Dit gaat goed wanneer de rij woningen huurhuizen zijn en op één perceel liggen. Maar wanneer de rij door percelen opgedeeld worden dan worden de adressen aan een verkeerd deelgebouw toegekend. Dit is zowel een fout van type 3 als ook een fout van type 4, want het is mogelijk om deze situatie op te vangen. De deelgebouwen die hier paars zijn gekleurd zijn geclassificeerd met behulp van classificatieregel A5 (type=a, waardering=5), waarbij het deelgebouw door het aantal adressen gedeeld wordt en er gekeken word of deze kleiner is dan 30 m^2



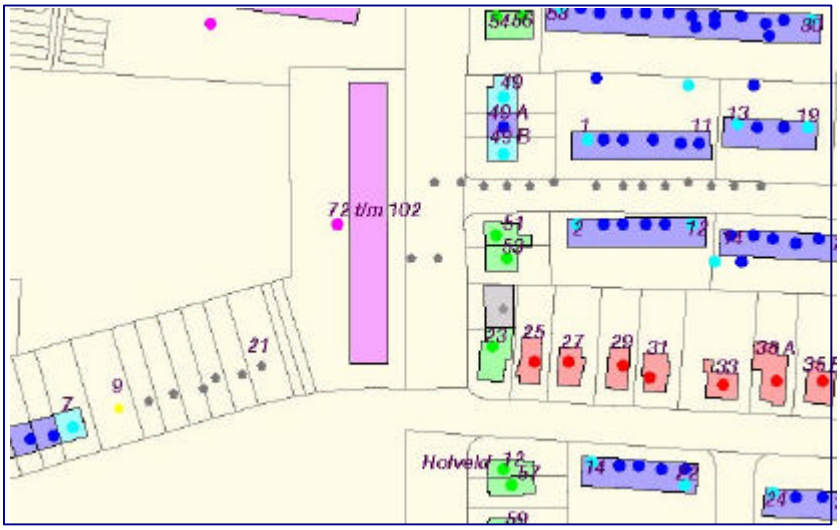
Kaart 10-4 B: Fout geclassificeerde rijtjes.



Kaart 10-5 drie adressen in een in tweeën gedeeld gebouw en tevens 3 echt verschillende huisnummers.

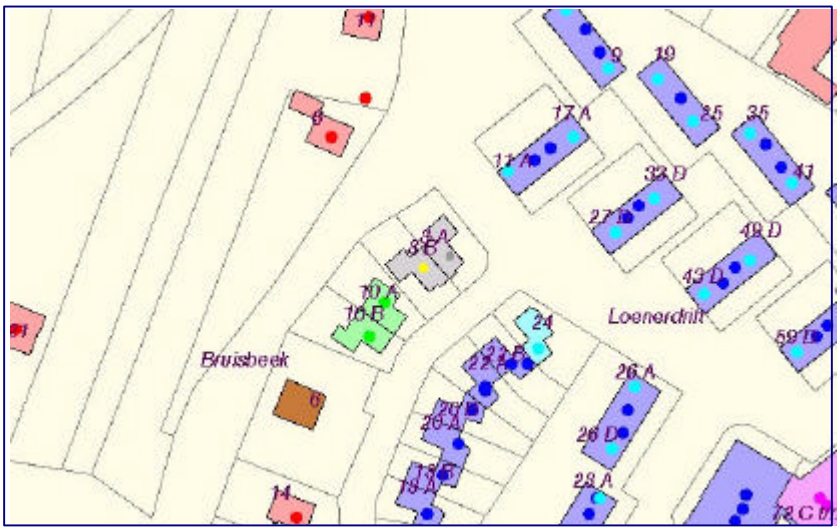
Kaart 10-6 Drie adressen in een in tweeën verdeeld gebouw

Op de kaarten 10-5 en 10-6 op deze pagina is te zien dat een gebouw, dat in twee deelgebouwen is verdeeld, drie adressen bevat. In het eerste is te zien dat er ook werkelijk 3 huisnummers bij het gebouw horen. Een deel van het gebouw is geclassificeerd als niet zijnde een woning. Juist in dit deel valt maar één adres. Waarschijnlijk had het andere deel van het gebouw een middenwoning moet zijn. Ook het gebouw in kaart 10-6 had dus eigenlijk geclassificeerd moeten worden als een middenwoning. Dit is een fout van type 4. Het is mogelijk in deze kaart dat het huis met huisnummer 10 hetzelfde is als 10-1. Dan zou het wel een twee-onder-één-kap moeten zijn.



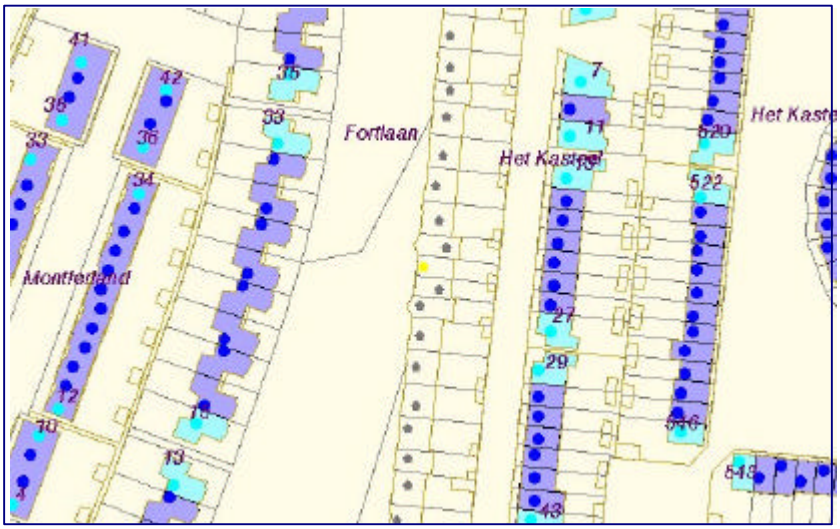
Kaart 10-7 Een rij waar het gebouw mist en een rij fout geplaatste ACN.

In kaart 10-7 is een duidelijk geval te zien van een rij waar het gebouw mist. Dit is een fout in LKI, type 1. Ook is weer een voorbeeld te zien van een hele reeks fout geplaatste ACN, fouttype 3.



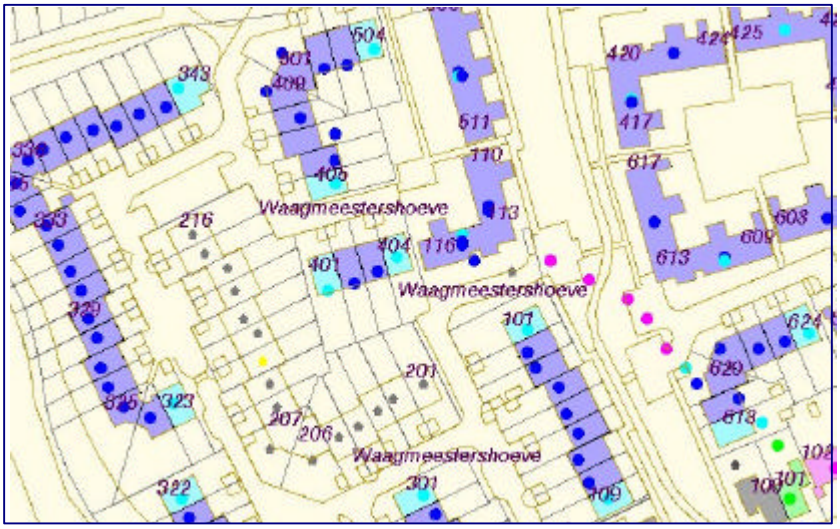
Kaart 10-8 Gevolgen van 1 fout geplaatste adres

In kaart 10-8 is weer een fout in de adressen te zien. Het adres met huisnummer 6 heeft dezelfde coördinaat gekregen als nummer 8A. Hierdoor is zowel de twee-onder-één-kap niet geclassificeerd en is het deelgebouw waar nummer 6 eigenlijk bij hoort fout geclassificeerd. Fouttype 3



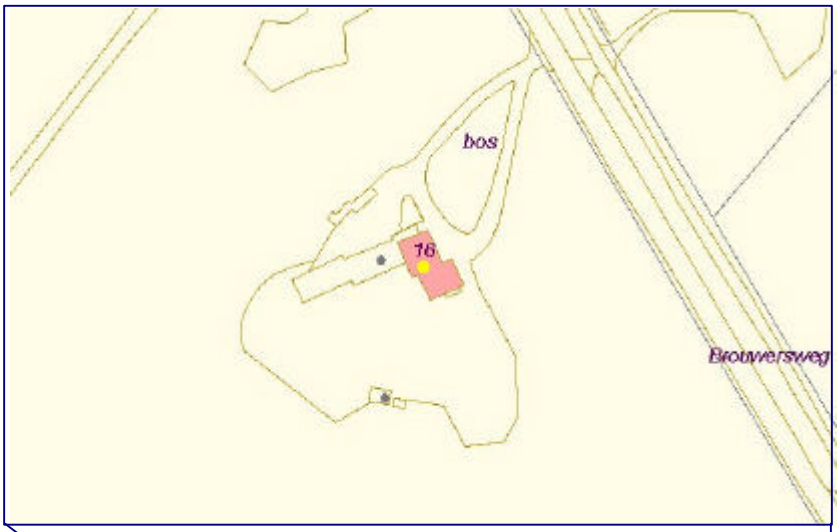
Kaart 10-9 Een rij woningen waarvan het erbij behorende gebouw niet gesloten is.

In kaart 10-9 is een fout te zien die het gevolg is van fouten in LKI. Aan deze kaart zijn de LKI-lijnen toegevoegd. Het is dus niet gelukt om dit gebouw te sluiten.



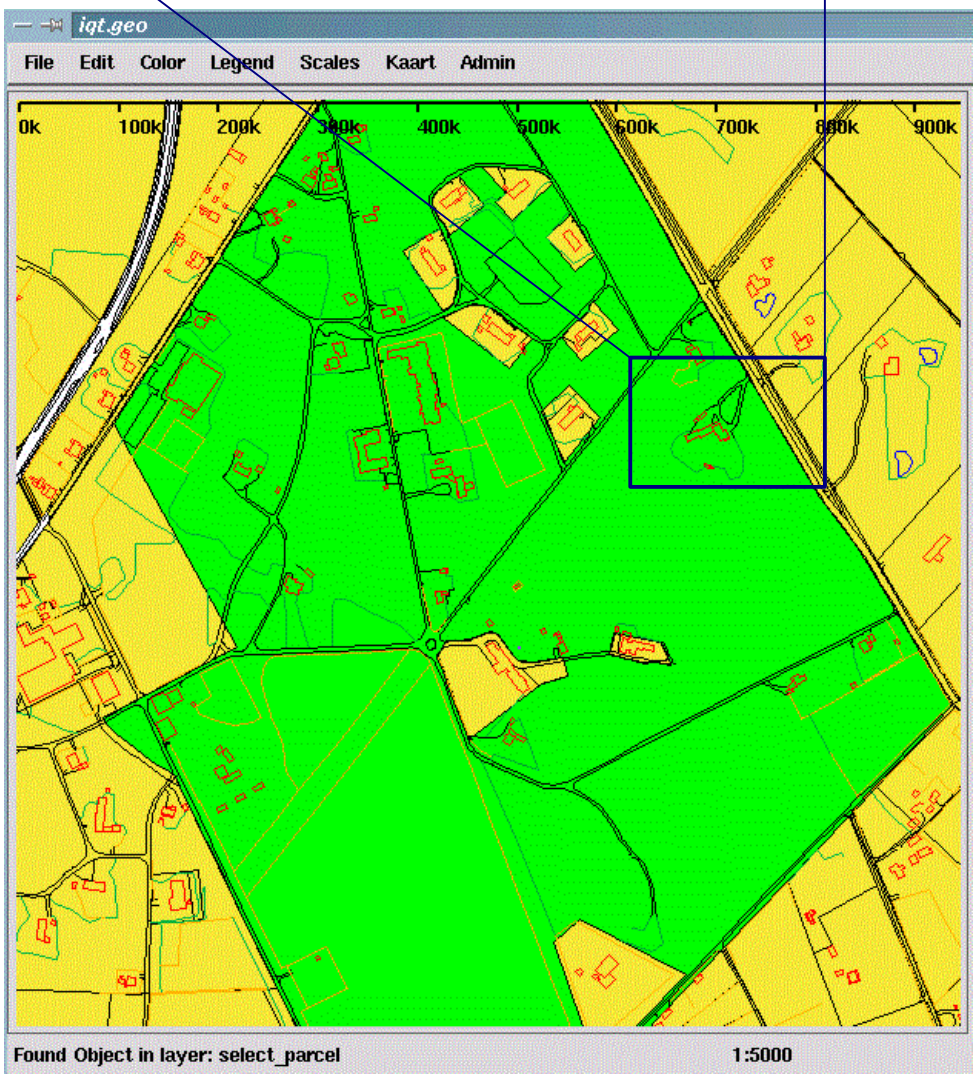
Kaart 10—11 Twee voorbeelden: gebouw niet gesloten en foute plaatsing ACN

In kaart 10-10 is nog een voorbeeld te zien van een rij huizen waar het niet gelukt is het gebouw te sluiten. Ook is er een voorbeeld te zien van verkeerd geplaatste ACN (roze punten).

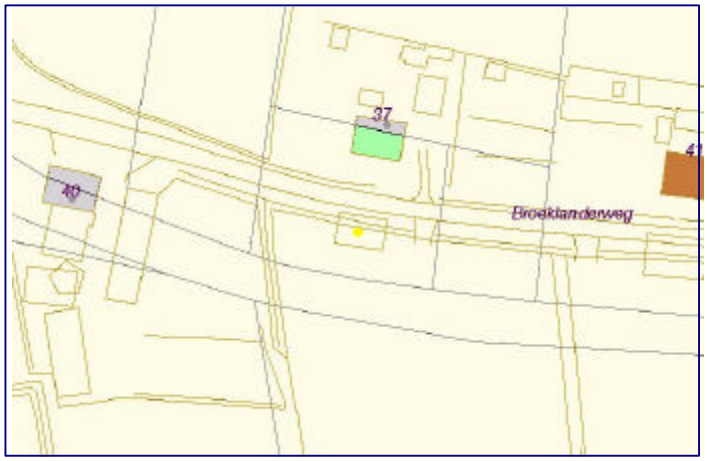


Kaart 10-11 Voorbeeld van een fout in de classificatiecode.

In kaart 10-11 zijn een gebouw en adressen te zien waarvan de cultuurcode in AKR waarschijnlijk fout is. In het snapshot is te zien hoe gigantisch het bijhorende perceel is. Het perceel is eigendom van een stichting.
Fout van type 2.



Snapshot 10-1 Snapshot van de Kadaster-query-tool, waarin het bijbehorende perceel geselecteerd (groen) is.



Kaart 10-12 Zeer opmerkelijk. Toen de LKI-lijnen toegevoegd werden aan de kaart leek het wel of er hier een verschuiving had plaatsgevonden. Deze fout is een paar keer geconstateerd.

Wel heel erg vreemd is kaart 10-12. Ook hier zijn weer de LKI lijnen toegevoegd. En het lijkt wel of de laag met de percelen verschoven is ten opzichte van de lijnen die uit de LKI-lijnen tabel komen. Dit is een fout van type 1.

10.3.2 CONCLUSIES BIJ DE VOORBEELDEN

Tot zo ver de kaarten met voorbeelden. Een aantal dingen is opgevallen. Uit al deze voorbeelden mag geconcludeerd worden dat:

1. Fouten in LKI over het algemeen inhouden dat er één of meer adressen niet geclassificeerd kunnen worden. Deze adressen krijgen een classificatie “geen gebouw” en zijn achteraf dus wel terug te vinden.
2. Fouten in het AKR houden over het algemeen in dat één of meer adressen uiteindelijk een foute classificatie krijgen. Dit is mogelijk op te vangen door te kijken naar de natuurlijke persoon code van de eigenaar. Zeker in het geval van vrijstaande en twee-onder-één-kap woningen, omdat deze over het algemeen niet vaak verhuurd worden. Juist huursituatie zorgen ervoor dat er niet altijd naar de eigenaar gekeken kan worden want dit zijn vaak woningstichtingen die geen natuurlijk persoon zijn.
3. Fouten in ACN hebben soms tot gevolg dat niet alleen het fout geplaatste adres verkeerd wordt geclassificeerd, maar ook de adressen van het perceel waar deze in terecht is gekomen.
4. Fouten in de classificatie bestaan grotendeels omdat bepaalde gevallen niet zijn opgevangen. Dit houdt in dat ze er verbeteringen mogelijk zijn.

Het is moeilijk om voor elk type fout aan te geven om hoeveel procent het gaat. Omdat sommige fouten op één adres doorwerken, terwijl andere op een aantal adressen doorwerken. Ook komt het voor dat een fout geclassificeerd adres het gevolg is (kan zijn) van meer dan 1 fouttype. Over het algemeen kan gesteld worden dat de meeste fouten het gevolg zijn van fouten in de bronbestanden.

.Fouttype	Aantal in testset.
1 LKI	22
2 AKR	24
3 CAN	55
4 Classificatieregels	23

Tabel 10-4 De fouten naar type ingedeeld. De foute classificaties als gevolg van de tikfouten zitten hier niet meer in

10.4 MOGELIJKE VERBETERINGEN

Er zijn een aantal verschillende mogelijkheden om het aantal fout geclassificeerde of nog niet geclassificeerde deelgebouwen / adressen te verbeteren. Puntsgewijs worden er een aantal hiervan genoemd:

- Samenvoegen buurpercelen met een zelfde eigenaar.
- Meenemen natuurlijke persoon code van de eigenaren, met name voor vrijstaande en twee-onder-één-kap woningen.
- Een na-controle uitvoeren door voor alle gebouwen, met name de twee-onder-één-kap woningen de adressen te controleren op een zelfde postcode.
- Door bij de percelen aan te geven of de grenzen versneden worden door gebouwgrenzen, kunnen bijvoorbeeld vrijstaande huizen waarbij het gebouw mist geclassificeerd worden. Dit berekenen van het wel of niet versnijden kan op de manier die beschreven is in hoofdstuk 7 (geometrische versnijding). Misschien is het mogelijk om zelfs het aantal keren te tellen dat een perceelgrens versneden wordt, dit zou een maat kunnen zijn om het aantal buren te bepalen.

10.4.1 AANPASSINGEN AAN DE CLASSIFICATIEREGELS:

Als laatste van dit hoofdstuk nog een aantal voorstellen om de classificatieregels aan te passen. Dit ter verbetering van de gevonden classificatiefouten.

- E 15 moet M 15 zijn, Dit was een tikfout en deze verbetering is toegepast in de berekening van de aantallen in hoofdstuk 8.
- Toevoegen van een Classificatie voor appartementen: is het aantal adressen op een perceel zonder gebouw groter dan 20 dan worden het appartementen, zeker als alle adressen op 1 positie liggen.
- Toevoegen van nog een classificatie voor middenwoningen: Het aantal buren groter dan nul en meer dan één adres in het deelgebouw, en meer dan drie adressen in het gehele gebouw. (deelgebouw.n_buren>0, deelgebouw.n_acn>1, gebouw.n_acn>3)
- Volgordes omwisselen. V4,V5 verwisselen en T4,T6 verwisselen.
- Voor de waarderingen geldt dat 8 beter vervangen kan worden door 4 en rest 1 opschuiven, overeenkomstig de volgorde die tijdens de classificatie gedraaid wordt. De reden waarom dat nog niet zo is, is omdat deze in een eerder stadium pas zo laat gedraaid werden.
- Voordat A5 (oppervlakte door het aantal ACN delen) uitgevoerd wordt, eerst de situatie in de kaart 10-4 oplossen. Dit kan door de classificatie niet naar deelgebouwen, maar naar gebouw kijken als n_deelgebouwen >3 en n_acn = n_deelgebouwen dan middenwoning.

CONCLUSIES & AANBEVELINGEN

Het doel, een methode om woningen af te leiden en te classificeren vanuit Kadastrale gegevens, is verwezenlijkt. In de hoofdstukken 6 tot en met 8 wordt deze methode in detail beschreven..

De hoofdconclusie is dat er nu een methode ligt om (ongeveer 96.5 % van) de adressen in te delen in een van de volgende klassen: appartement, vrijstaand, twee-onder-één-kap, eindwoning, middenwoning en geen woning .

De overgebleven 3.5% van de testset is opgedeeld in adressen die niet aan een gebouw gekoppeld kunnen worden en adressen die nog niet geclassificeerd zijn. Op basis van bestaande kadastrale gegevens worden de classificaties bepaald.

11.1 CONCLUSIES

Voordat de conclusies kunnen worden gegeven zal eerst nog even de hoofdvraag en het doel van dit afstuderen even terug gehaald worden. Deze was als volgt:

> *Is het mogelijk om een methode te ontwikkelen die automatisch op basis van digitaal aanwezige gegevens woningen afleidt en als een bepaald type classificeert?*

In de volgende paragraaf zullen één voor één de deelvragen en daarna de hoofdvraag worden beantwoord.

11.1.1 BEANTWOORDING DEELVRAGEN

Deelvraag 1: Is het mogelijk om per adres het woningtype automatisch af te leiden en te classificeren met behulp van de kadastrale kaart en administratieve gegevens?
Antwoord: Het is mogelijk om met behulp van een automatische methode voor de meeste adressen het woningtype af te leiden uit kadastrale gegevens. In de testset is 95,75 % van de adressen geclassificeerd. Van de overgebleven 4,25 % is ruim 88% niet te koppelen aan een gebouw. De kenmerken van de gebouwen worden juist gebruikt om de classificaties uit te voeren.

Het is gelukt om een groot deel van alle adressen in de testset “Apeldoorn” te classificeren met behulp van een geschreven programma. Dit programma bevat weer aanroepen naar andere programma’s. De gehele classificatie bestaat uit programmaregels die unix, Arc/Info en SQL commando’s bevatten.

Voor de classificatie is gebruik gemaakt van LKI gegevens, AKR gegevens en ACN gegevens. Attributen uit de tabellen LKI-line, LKI-parcel en LKI-tekst zijn gebruikt. Uit deze laatste zijn de plaatsingen van huisnummers gehaald. Uit de AKR dataset zijn de object- en objectadrestabel voornamelijk gebruikt. Alle adressen uit ACN hebben uiteindelijk een woningclassificatie gekregen. De adrescoördinaten hebben een belangrijke rol gekregen in het classificatieproces.

Deelvraag 2: Kan deze afleiding met behulp van de geometrische basisbibliotheek “Computational Geometry Algorithms Library” (**CGAL**) tot stand komen?
Antwoord: Op dit moment is het nog niet mogelijk om de CGAL-bibliotheek te gebruiken om het geometrische deel van de afleiding mee te berekenen. Dit komt vooral door de fouten die nog in CGAL zitten.

Het is niet gelukt om in de uiteindelijke classificatie gebruik te maken van CGAL voor de versnijding van de gebouwen met de percelen. CGAL bied wel veel mogelijkheden, maar er zitten toch een aantal onoverkomelijke “bugs” in de bibliotheken die gebruikt moeten worden. De belangrijkste bug die gevonden is in de bibliotheek, is het in sommige gevallen niet berekenen van versnijdingspunten, terwijl deze er toch echt wel waren.
Dit heeft tot gevolg gehad dat in de uiteindelijke classificatie gebruik is gemaakt van de berekening van de overlay met behulp van Arc/Info zoals deze gedefinieerd is voor het Kadasterproject.

Deelvraag 3: Wat zijn de redeneerregels waarmee de schatting van een woningtype plaatsvindt?
Antwoord:. De classificatieregels die zijn opgesteld maken gebruik van de geometrische en attribuutkenmerken. De vijf woningtypen hebben verschillende kenmerken waarmee ze van elkaar te onderscheiden zijn. Bijvoorbeeld wanneer het aantal buren groter dan nul is zullen we niet meer te maken hebben met een vrijstaande woning. Er zijn 36 classificatieregels opgesteld voor het bereken van het woningtype aan de hand van deze kenmerken. De meeste (32) worden gebruikt om de deelgebouwen een classificatie te geven. De overige vier worden gebruikt om adressen te classificeren. Het gaat te ver om al deze classificaties hier te noemen, een overzicht en beschrijving is te vinden in paragraaf 8.4.3.

Tabel	Kolomen
Deelgebouwen	ID, perceelidentificatie, gebouwidentificatie, buren, oppervlakte, n_acn, n_huisnummers, indicatie_wonen, woningtype, waardering
Gebouwen	Identificatie, oppervlakte, n_deelgebouwen, n_acn, n_huisnummers, n_percelen.
Percelen	identificatie (G_AKR_objectnummer), oppervlakte, n_gebouwen, n_deelgebouwen, Cultuurcode
Huisnummers	identificatie, deelgebouwidentificatie, object_tekst
CAN	identificatie, postcode, huisnummer, huisnummertoevoeging, aantal, deelgebouwidentificatie, perceelidentificatie, woningtype, waardering

Tabel 10-1 (zelfde als tabel 8-1) Een overzicht van de gebruikte tabellen met gebruikte attributen.

Aan de tabellen deelgebouwen en ACN worden de attributen woningtype en waardering toegevoegd. Het woningtype krijgt in de classificatie één van de volgende codes toegekend:

- # Cultuurcode niet wonen
- – Nog niet geclassificeerd.
- A Appartement
- E Eindwoning
- M Middenwoning
- T Twee-onder-één-kap woning
- V Vrijstaande woning
- S Schuur, alle vrijstaande gebouwen zonder een ACN.

Tijdens de classificatie worden de attributen woningtype en waardering gevuld. Eerst worden alle deelgebouwen geclassificeerd met een dertigtal classificatieregels. Daarna krijgen alle adressen die bij een bepaald deelgebouw horen dezelfde classificatie. Als laatste stap worden de adressen verder bewerkt met nog een aantal classificatieregels. Dit zijn drie classificatieregels om voor gevonden rijtjes de juiste midden en hoekwoningen te vinden.

Deelvraag 4: Voor die gevallen dat een schatting van het woningtype mogelijk is, hoe goed is die schatting dan?
Antwoord: Het eerste deel van het antwoord: Naast een woningtype wordt er ook een waardering toegekend aan een deelgebouw en adres. Deze waardering samen met de classificatie zelf geeft aan welke classificatieregel er is toegepast. De waardering is een getal. De classificaties die een waardering van 1 krijgen zijn de ideale gevallen. Hoe hoger de waardering hoe meer van de kenmerken er vervallen of versoepeld zijn.
Het tweede deel van het antwoord: De testset is gevalideerd (hoofdstuk 10), hieruit volgt dat de meeste foute classificaties het gevolg zijn van fouten in de bronbestanden.

Veel van de fout geclassificeerde of niet geclassificeerde woningen zijn het gevolg van de fouten in de bronbestanden. Door de fouten en kwaliteit van de bronbestanden was het nodig om meer dan één beslisregel per woningtype te definiëren afhankelijk van de zwakheden van de bronbestanden. Met name de adrescoördinaten lagen vaak buiten het gebouw.

Deze beslisregels en combinaties kunnen er bij helpen om fouten in de bronbestanden te detecteren en waarschijnlijk met wat aanpassingen gebruikt worden om deze te verbeteren, denk bijvoorbeeld aan de ACN coördinaten die niet in het gebouw liggen, deze kunnen we nu vinden.

11.1.2 BEANTWOORDING HOOFDVRAAG

Hoofdvraag: Is het mogelijk om een methode te ontwikkelen die automatisch op basis van digitaal aanwezige gegevens woningen afleidt en als een bepaald type classificeert?
Antwoord: Ja, het is mogelijk. Met de methode zoals deze beschreven is in dit afstudeerverslag worden woningen geclassificeerd naar een bepaald type. Deze methode maakt gebruik van aanwezige digitale gegevens om nieuwe gegevens te berekenen en deze te gebruiken om met behulp van classificatieregels de woningen te classificeren.

De uiteindelijke methode bestaat uit de volgende stappen:

- Het selecteren van objecten met attributen die de percelen en gebouwen moeten vormen.
- Het vormen van kaartlagen die gebruikt worden voor de versnijding.
- Het berekenen van deelgebouwen door een versnijding tussen percelen en gebouwen.
- met behulp van een point-in-polygoon zoekalgoritme bepalen in welk deelgebouw en perceel de adrescoördinaten en liggen.
- met behulp van een point-in-polygoon zoekalgoritme bepalen in welk deelgebouw de huisnummercoördinaten liggen.
- De attributen die gebruikt worden in de classificatie berekenen. Sommige attributen worden berekend uit geometrische relaties, zoals het aantal buren en de oppervlakte. Maar ook met behulp van SQL worden er een aantal attributen berekend, zoals het aantal deelgebouwen waaruit een gebouw bestaat.
- Het classificeren van de deelgebouwen met behulp van classificatieregels.
- Het classificeren van adressen met behulp van de resultaten uit de vorige stap, aangevuld met nog een aantal classificatieregels.

Ruim 96.5% van de adressen in de testset hebben een woningtype gekregen. Van de overgebleven 3,5 % kan op dit moment geen woningtype worden bepaald. 3% zijn adressen die niet aan een gebouw gekoppeld kunnen worden. En slechts 0.5% kan met behulp van de classificatieregels die zijn opgesteld niet een woningtype toegekend krijgen.
Niet alle gevonden classificaties zijn juist. Van de validatieset (ruim 1000 adressen) was ongeveer 10% fout geclassificeerd. De meeste fouten werden veroorzaakt door fouten in de bronbestanden. Slechts 1/5 van de fouten wordt veroorzaakt door fouten in de classificatieregels. Bijvoorbeeld appartementen en rijtjeswoningen die verwisseld worden. Dit komt vooral voor bij woningen die waarschijnlijk twee woonlagen vormen.
Een ander voorbeeld zijn eindwoningen en middenwoningen die verwisseld worden. Dit komt vooral voor wanneer er in een rij twee straten samen komen of wanneer het gebied op een ongewone manier is genummerd. Voor de classificatieregels is ervan uitgegaan dat een rij opvolgend genummerd is en dat een rij in zijn geheel tot een bepaalde straat hoort.

11.2 AANBEVELINGEN

Als de problemen / bugs uit CGAL gehaald zijn kan de overlay als nog met behulp van deze bibliotheek gebeuren. Dit heeft een aantal voordelen:

- De kaartlagen in CGAL hoeven niet gesloten gebouwen te bevatten, ook niet te sluiten gebouwlijnen kunnen gebruikt worden om met de percelen te versnijden
- Voor de opdelingen van percelen met niet gesloten gebouwen kunnen het aantal opdelingen worden geteld en het aantal buren dat ook versneden is door een niet gesloten gebouw worden geteld.
- Deze kenmerken kunnen ook gebruikt worden om nog niet geclassificeerde adressen te classificeren.

Om het aantal fout geclassificeerde adressen te verminderen zouden er een aantal aanpassingen aan de classificatieregels moeten gebeuren. De belangrijkste is het aanpassen van de volgorde waarop de classificaties gebeuren. Ook kunnen er nog classificatieregels toegevoegd worden. Ter verbetering van de classificatie zouden de volgende aanpassingen aan de classificatieregels kunnen worden toegepast:

- Nog een classificatie voor appartementen: Het aantal adressen op een perceel zonder gebouw is groter dan 20 dan worden het appartementen.
- Nog een classificatie voor middenwoningen: Het aantal buren groter dan nul en meer dan één adres in het deelgebouw, en meer dan drie adressen in het gehele gebouw.
- Voordat de oppervlakte door het aantal ACN wordt gedeeld, eerst de situatie waarbij alle adressen van een opgedeelde rij in 1 perceel liggen oplossen. Dit kan door in de classificatie niet naar het deelgebouw, maar naar het gehele gebouw te kijken.
- De oppervlaktewaarden, zoals 30 m^2 kunnen misschien aangepast worden, deze waarden zijn een intuïtieve benadering van een fuzzy grens. Onderzocht kan worden of hier betere waarden voor gebruikt kunnen worden.
- Er zouden huisnummercontoles kunnen worden uitgevoerd voor alle twee-onder-één-kap, en rijtjeswoningen, zodat fout geplaatste adressen kunnen worden gevonden en hun invloed op de andere adressen in het gebouw terug gedraaid kan worden.

11.2.1 VOOR VERDER ONDERZOEK:

Er zou onderzocht kunnen worden hoe de fouten van de bron bestanden doorwerken in de uiteindelijke classificaties. Een gebouw dat niet gesloten is (LKI-fout) kan ervoor zorgen dat er één adres fout geclassificeerd wordt (vrijstaand huis), maar ook dat een hele rij woningen niet geclassificeerd worden. Een fout geplaatste adrescoördinaat kan zelf fout geclassificeerd worden, maar ook een ander adres. Dit komt bijvoorbeeld voor waneer het fout geplaatste adres in een vrijstaande woning terecht komt. Beide worden dan een twee-onder-één- kap.

Er zou onderzocht kunnen worden of deze methode kan helpen in het opsporen en verbeteren van fouten in de bronbestanden. De adressen horen bijvoorbeeld in de bijbehorende percelen en gebouwen te liggen, dit is nog vaak niet het geval.

Verder zouden alle gegevens, die in principe vlakken vormen, in LKI een betere opslagstructuur moeten krijgen. Dit zijn bijvoorbeeld de gebouwen. Een eerste stap daarnaar toe is het sluitend maken van alle gebouwen. Daarna zouden de gebouwen nog een identificatienummer kunnen krijgen, zodat ze op eenzelfde manier opgeslagen kunnen worden als de percelen.

BRONNEN

Literatuur

[CBS2000]
Centraal Bureau voor de Statistiek, Statistisch Jaarboek 2000, blz 180

[BKOS97]
M. de Berg, M. van Kreveld, M. Overmars & O. Schwarzkopf. Computational Geometry: Algorithms and Applications. Springer Verlag, Berlin, 1997.

[BMcD98]
P.A. Burrough & R.A. McDonnell – Principles of Geographical Information Systems – Oxford University Press – 1998

[KBS91a]
H.P. Kriegel, T Brinkhoff & R. Schneider The Combination of Spatial Acces Methods and Computational Geometry in Geographic Database Systems.1991a, overgenomen uit [O94]

[KBS91b]
H.P. Kriegel, T Brinkhoff & R. Schneider, An Efficient Map Overlay Algorithm Based on Spatial Access Methods and Computational Geometry in Geographic Database Management Systems, pp 194-211 Berlin. Springer-Verlag. 1991b

[K98]
Productbeschrijving Adrescoördinaten Nederland, Kadata BV, Apeldoorn 1998

[LKI99]
Classificatie tabel LKI 3.26, Kadaster/ITS/APS/CKA september 1999

[094]
P. van Oosterom, An R-tree based map- overlay algorithm, TNO Physics and Electronics Laboratory, The Hague, The Netherlands, 1994.

[OMQ2000]
P. van Oosterom, B Maessen and W. Quak, Spatial, thematic and temporal views, DH2000, 9th International Symposium on Spatial Data Handling, Beijing, China 2000.

[Q95]
Wilko Quak, Towards Integration of complex spatial analysis and databases. sheets 1995

[S97]
B. Stein, A Point about Polygons, Linux Journal March 97, http://home.earthlink.net/~bobstein/inpoly/

[U34]
Technical Issues in GIS, Unit 34 - Polygon Overlay Operation, NCGIA Core Curriculum, http://geography.otago.ac.nz/Mirrors/NCGIA_GIS_CURRICULUM/u34.html

[W93]
Wolters' Woordenboek Nederlands Koenen, 29e druk, Wolters-Noordhoff bv, Nederland 1992/1993.

[W99]
Waarderingsinstructie 1999, de waarderingskamer.

Internet:

[i1] http://www.kadaster.nl/ profiel/index.html
[i2] http://www.kadaster.nl/producten/AKR
[i3] http://www.kadaster.nl/producten/kadastrale kaart
[i4] http://www.nvm.nl
[i5] http://www.vissermakelaardij.nl/pagina12206-koop.htm)

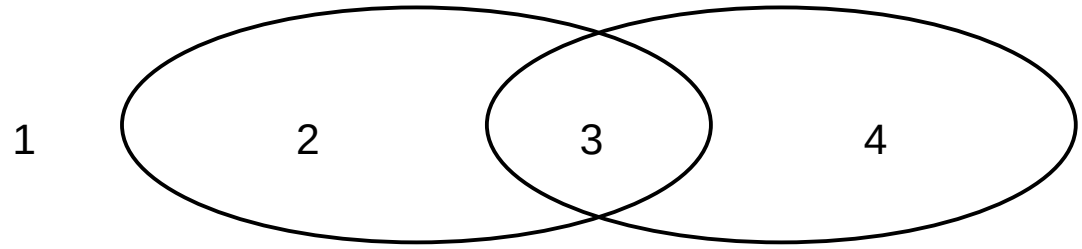
Algemene bronnen van informatie:

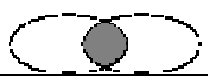
Handleiding MO van het Kadaster
Documentatie CGAL 2.1 http://www.cs.uu.nl/CGAL/

BIJLAGEN

Programma’s met SQL commando’s

Bijlage A: Boolean operaties op twee polygonen

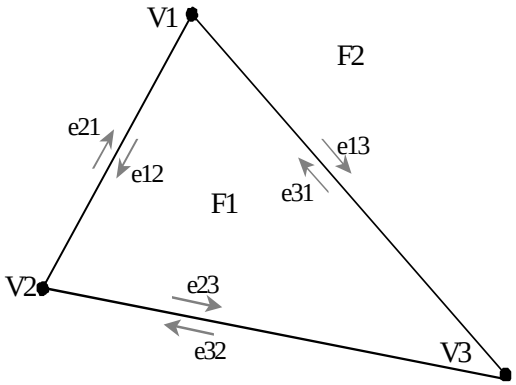


BOOLEAN OPERATIONS	1	2	3	4	Grafische weergave
NOT NULL	1	1	1	1	
A OR (NOT B)	1	1	1	0	
(NOT A) OR (NOT B) = NOT (A AND B)	1	1	0	1	
NOT B	1	1	0	0	
(NOT A) OR B	1	0	1	1	
(A AND B) OR ((NOT A) AND (NOT B))	1	0	1	0	
NOT A	1	0	0	1	
(NOT A) AND (NOT B) = NOT (A OR B)	1	0	0	0	
A OR B	0	1	1	1	
A	0	1	1	0	
(A AND (NOT B)) OR ((NOT A) AND B)	0	1	0	1	
A AND (NOT B)	0	1	0	0	
B	0	0	1	1	
A AND B	0	0	1	0	
(NOT A) AND B	0	0	0	1	
NULL	0	0	0	0	

Bijlage B1 DCEL en wat er met de DCEL gebeurt bij een versnijding

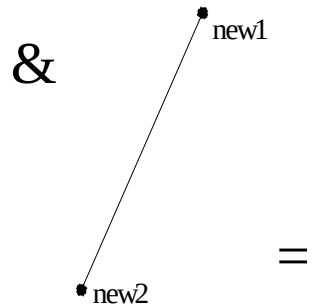
DCEL ***voor*** de versnijding:

knooppunt	coördinaten	Uitgaande lijn
V1	2,4	E12
V2	0,1	E23
V3	5,0	E31



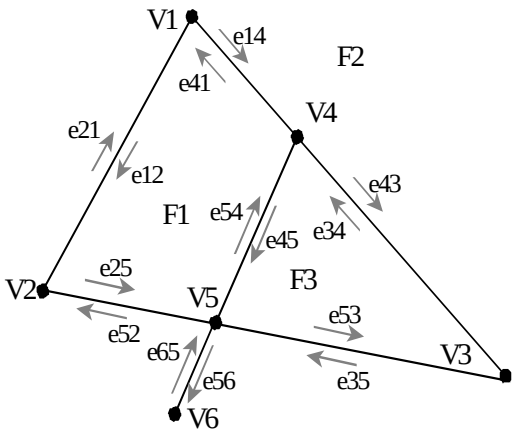
lijn	origin	Twin	face	Prev	next
E12	V1	E21	F1	E31	E23
E21	V2	E12	F2	E32	E13
E13	V1	E31	F2	E21	E32
E31	V3	E13	F1	E23	E12
E23	V2	E32	F1	E12	E31
E32	V3	E23	F2	E13	E21

face	Out	In
F1	Nil (eiland)	E12
F2	E21	Nil



DCEL ***na*** de versnijding:

knooppunt	coördinaten	Uitgaande lijn
V1	2,4	E14
V2	0,1	E21
V3	5,0	E34
V4	4,3	E45
V5	3,0.5	E54
V6	1,-1	E65



lijn	origin	Twin	face	prev	next
E12	V1	E21	F1	E41	E25
E21	V2	E12	F2	E52	E14
E14	V1	E41	F2	E21	E43
E41	V4	E14	F1	E54	E12
E45	V2	E32	F3	E34	E31
E54	V3	E23	F1	E25	E41
E43	V4	E34	F2	E14	E35
E34	V3	E43	F3	E53	E45
E25	V2	E52	F1	E12	E54
E52	V5	E25	F2	E65	E21
E56	V5	E65	F2	E35	E65
E65	V6	E56	F2	E56	E52
E53	V5	E35	F3	E45	E34
E35	V3	E53	F2	E43	E56

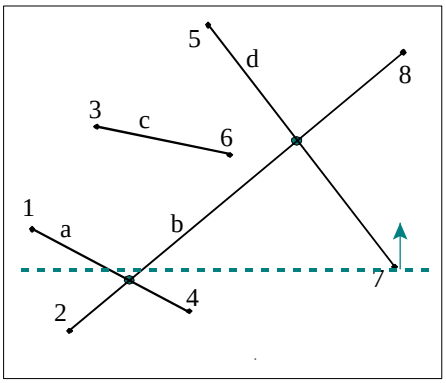
face	Out	In
F1	Nil (eiland)	E12
F2	E21	Nil
F3	Nil	E45

Er zijn in totaal zes nieuwe lijnen toegevoegd (dit zijn 12 nieuwe halfedges), en twee verwijderd, er zijn drie nieuwe punten toegevoegd. F1 de verwijzing naar een edge op de rand is gecontroleerd en geldt nog steeds.

Bijlage B2: Voorbeeld van een sweeplijn.

Nu nog eens tabel 4-1 met bijbehorende plaatje, maar dan een horizontale sweeplijn

Q	R	Mogelijkheid	Wat gebeurt er
2 4 7 1 6 3 8 5	Leeg	1	R + b
4 7 1 6 3 8 5	b	1	R + a, Q + i1
i1 7 1 6 3 8 5	b a	3	In R a, b verwisselen
7 1 6 3 8 5	*a b*	*1*	*R + d, Q + i2*
1 6 i2 3 8 5	a b d	2	R – a
6 i2 3 8 5	b d	1	R + c
i2 3 8 5	c b d	3	In R b, d verwisselen
3 8 5	c d b	2	R – c
8 5	d b	2	R – b
5	d	2	R – d
leeg	Leeg	Einde	



Bijlage C1 Het versnijdingsprogramma

```
// ============================================================
// uses       : CGAL-2.1
// release_date  : 2000, January 11
//
// file          : overlaytest.C
// date          : 10 maart 2000
// revision_date : 23 aug 2000
// author(s)     : Maureen Rengelink
//
// coordinator   :
//
// email         :
// ============================================================

#include <iostream>
#include <fstream.h>
#include <stdlib.h>

#include <CGAL/basic.h>
#include <CGAL/Cartesian.h>
//#include <CGAL/Homogeneous.h>
#include <CGAL/Pm_segment_exact_traits.h>
#include <CGAL/Pm_default_dcel.h>
#include <CGAL/Planar_map_2.h>

#ifndef CGAL_SEGMENT_2_SEGMENT_2_INTERSECTION
#include <CGAL/Segment_2_Segment_2_intersection.h>
#endif

#include <CGAL/Pm_naive_point_location.h>
//#include <CGAL/Pm_walk_along_line_point_location.h>

typedef double                                  Basetype;
typedef CGAL::Cartesian<Basetype>               Rep_class;
typedef CGAL::Pm_segment_exact_traits<Rep_class>     Pmtraits;

typedef Pmtraits::Point               Point;
typedef Pmtraits::X_curve             Curve;

#ifndef PM_MO_defaultdcel
#include "Pm_with_double_info_dcel.h"
#endif

typedef CGAL::Pm_mo_default_dcel<Pmtraits> mDcel;

typedef CGAL::Planar_map_2<mDcel, Pmtraits > Planar_map_ot;

typedef Planar_map_ot::Face_handle            face_handle;
typedef Planar_map_ot::Halfedge_handle        halfedge_handle;
typedef Planar_map_ot::Vertex_handle          vertex_handle;
typedef mDcel::Info_face                             Info_face;
typedef mDcel::Info_edge                             Info_edge;
typedef Planar_map_ot::Face_iterator          face_i;
typedef Planar_map_ot:: Locate_type           loct;

#include <CGAL/Point_2.h>
#include <CGAL/Polygon_2.h>
#include <list>
#include <set>

typedef CGAL::Polygon_traits_2<Rep_class> ptrai;
typedef ptrai::Point_2 Pntp;
typedef std::list<Pntp> poly_cont;
typedef CGAL::Polygon_2<ptrai,poly_cont> Polygon;

bool SMALLER=true;
bool LARGER=false;

template<class T>
class Segment_for_pm : public T
{
```

```
  int     _lf_id;
  int     _rf_id;
public:
  int get_lf_id()       const { return _lf_id; }
  int get_rf_id()       const { return _rf_id; }
  Segment_for_pm(T tt,
                       int clf, int crf) : T(tt),
                                   _lf_id(clf),
                                   _rf_id(crf) { }
  ~Segment_for_pm() { }
};

typedef std::list<Segment_for_pm<Curve> > Segment_container;

struct compare
{
   bool operator() (const Point pnt1, const Point pnt2) const
   {
         if(pnt1.x() < pnt2.x()) return true;
         if((pnt1.x() == pnt2.x()) && (pnt1.y() < pnt2.y())) return true;
         else return false;
   }
};
struct comp_id
{
   bool operator() (const int id1, const int id2) const
   {
         if(id1 < id2) return true;
         else return false;
   }
};

void checked_insert_overlay(Planar_map_ot& pm,
                               Point p1, Point p2,
                               int lf, int rf,
                               Segment_container& Seg_cont)
{
   typedef std::set<Point,compare> P_container;
   P_container  pc;
   typedef std::list<halfedge_handle> H_container;
   H_container hc1;
   typedef std::list<Segment_for_pm<Curve> > C_container;
   C_container hc2;
   //hc1 bevat wegtegooien elementen, hc2 bevat de vervangende elementen;

   halfedge_handle hei;

   Point tussen, tussen2, wisselp;
   Curve segm, cv, cv2;
   CGAL::Object obj;
   Point ptA, ptB, tus,tus2;
   double x1,y1,x2,y2;
    x1=p1.x();
    y1=p1.y();
    x2=p2.x();
    y2=p2.y();
   ptA=Point(x1,y1);
   ptB=Point(x2,y2);

   if ((p1.x() <= p2.x()) && (p1.y() < p2.y()))
   {
         cv=Curve(ptA,ptB);
   }
   else
   {
         cv=Curve(ptB,ptA);
         int wissellr=lf;
         lf = rf;
         rf=wissellr;
         wisselp=p1;
         p1=p2;
         p2=wisselp;
   }
```

```
      int hei_l, hei_r;
for (hei=pm.halfedges_begin(); hei!=pm.halfedges_end(); hei++)
{
  hei++; //twins komen na elkaar in de lijst, niet allbei doen
  x1=hei->source()->point().x();
  y1=hei->source()->point().y();
  x2=hei->target()->point().x();
  y2=hei->target()->point().y();
  cv2=Curve(Point(x1,y1), Point(x2,y2));
  if(do_intersect(cv,cv2))
  {
      hei_l=hei->info().info1;
      hei_r=hei->twin()->info().info1;
      obj=intersection(cv,cv2);
      if (assign(tus, obj))
      {         Point tussenround(
                 (double)((int)(((tus.x())*100))*0.01),
                 (double)((int)(((tus.y())*100))*0.01));
                tussen=tussenround;
              if ( (tussen == (*hei).curve().source() )
              || (tussen == (*hei).curve().target() ) )
              {//2,4
              if (! ((tussen ==  p1 ) || (tussen ==  p2 ) ))
              {//2
                     pc.insert(tussenround);
              }
              }
              else
              {//1,3
              Curve c1((*hei).source()->point(),tussen);
              Curve c2(tussen,(*hei).target()->point());
                     hc1.push_back(hei);
                     hc2.push_back(Segment_for_pm<Curve> (c1, hei_l, hei_r));
                     hc2.push_back(Segment_for_pm<Curve> (c2, hei_l, hei_r));
                     if (!( (tussen ==  p1 ) || (tussen ==  p2 ) ))
              {//1
                            pc.insert(tussenround);
                     }
              }
      }
      if (assign(segm, obj))
      {
              hc1.push_back(hei);
              //cout << "gevonden i is segment" <<"\n";
              //vergelijken richting hei en segment.
              if ( (hei->source()->point().x()< hei->target()->point().x())
                     || ( (hei->source()->point().x() ==
                           hei->target()->point().x())
                     && (hei->source()->point().y() <
                           hei->target()->point().y()) )
                 )
                     tus = segm.source();
                     tus2 = segm.target();
              }
              else
              {
                     tus2 = segm.source();
                     tus = segm.target();
              }

              Point tussenround(
                 (double)((int)((tus.x()*100))*0.01),
                 (double)((int)((tus.y()*100))*0.01));
               tussen = tussenround;
              Point tussenround2(
                 (double)((int)((tus2.x()*100))*0.01),
                 (double)((int)((tus2.y()*100))*0.01));
               tussen2 = tussenround2;

              if  (tussen ==  (*hei).source()->point() )
              {// casses 2,3,4,6,8,9
                     if (tussen==p1)
                     { //cases 3,4,9
                            if (tussen2==(*hei).target()->point())
```

```
                                {//4,9
                                        if(!(tussen2==p2))
                                        {//4
                                                pc.insert(tussenround2);
                                        }//else 9
                                }
                                else
                                {//3
                                        Curve k3(tussen2,(*hei).target()->point());
                                        hc2.push_back((Segment_for_pm<Curve>
                                                        (k3, hei_l, hei_r)));
                                }
                        }
                        else
                        {//2,5,8
                                if (tussen2==p2)
                                {//2,8
                                        if (tussen2==(*hei).target()->point())
                                        {//8
                                                pc.insert(tussenround);
                                        }
                                        else
                                        {//2
                                                pc.insert(tussenround);
                                                Curve k2(tussen2,(*hei).target()-
        >point());
                                                hc2.push_back((Segment_for_pm<Curve>
                                                                (k2, hei_l, hei_r)));
                                        }
                                }
                                else
                                {//5
                                        pc.insert(tussenround);
                                        pc.insert(tussenround2);
                                }
                        }
                }
                else
                {//1,6,7
                        if (tussen2==(*hei).target()->point())
                        {//1,7
                                if ((tussen2==p2))
                                {//7
                                        Curve k7((*hei).source()->point(),tussen);
                                        hc2.push_back((Segment_for_pm<Curve>
                                                        (k7, hei_l, hei_r)));
                                }
                                else
                                {//1
                                        Curve k1((*hei).source()->point(),tussen);
                                        hc2.push_back((Segment_for_pm<Curve>
                                                        (k1, hei_l, hei_r)));
                                        pc.insert(tussenround2);
                                }
                        }

                        else
                        {//6
                                Curve k61((*hei).source()->point(),tussen);
                                Curve k62(tussen2,(*hei).target()->point());
                                hc2.push_back((Segment_for_pm<Curve>
                                                (k61, hei_l, hei_r)));
                                hc2.push_back((Segment_for_pm<Curve>
                                                (k62, hei_l, hei_r)));
                        }
                }

        }
  }
}
if (pm.is_valid())
{
      H_container::iterator hc_i1;
      for (hc_i1=hc1.begin(); hc_i1!=hc1.end(); hc_i1++)
```

```
	{
		cout<<"verwijderen edge\n";
		pm.remove_edge(*hc_i1);
	}
}
//cout<< "check1\n";

if (pm.is_valid())
{
	Curve cv_vervanging;
	Point p1,p2;
	int lv,rv;
	halfedge_handle new_edge, hh, hh2;
	vertex_handle v1,v2;
	loct lt,lt2;
	C_container::iterator hc_i2;
	for (hc_i2=hc2.begin(); hc_i2!=hc2.end(); hc_i2++)
	{	//cout<<"vervanging\n";

		p1=(*hc_i2).source();
		p2=(*hc_i2).target();
		cout << p1 << "\t"<<p2 <<"\n";
		if (p1!=p2)
		{
			cv_vervanging=Curve(p1,p2);
			lv=(*hc_i2).get_lf_id();
			rv=(*hc_i2).get_rf_id();

			//new_edge = (Planar_map_ot::Halfedge_iterator)
				//pm.insert(cv_vervanging);
			hh=pm.locate(cv_vervanging.source(),lt);
			cout << lt << " !\n";
			if (lt==1)
			{	v1= hh->target();
				hh2=pm.locate(cv_vervanging.target(),lt2);
				if (lt2==1)
				{	v2= hh2->target();
					new_edge = pm.insert_at_vertices
						(cv_vervanging, v1, v2);
				}
				else
				{	new_edge = pm.insert_from_vertex
						(cv_vervanging, v1, true);}
			}
			else
			{	hh2=pm.locate(cv_vervanging.target(),lt2);
				if (lt2==1)
				{	v2= hh2->target();
					new_edge = pm.insert_from_vertex
						(cv_vervanging, v2, false);
				}
				else
				{	if (lt2==4)
					{
						new_edge = pm.insert
							(cv_vervanging);
					}
					if (lt2==3)
					new_edge = pm.insert_in_face_interior
						(cv_vervanging, hh2->face());
					if (lt2==2)
					new_edge = pm.insert_in_face_interior
						(cv_vervanging, hh->face());
				}
			}
			cout<<"lt=" << lt <<" \n" ;
			cout<<"lt2=" << lt2 <<" \n" ;
			if (lt==2 && lt2==2)
			{
				cout << "er gaat iets fout met de new_edge\n"
				<< "beide punten vallen op een lijn\n"
				<< p1 << "\t"<< p2 <<"\n";
			}
			else
```

```
                        {
                        cout<<new_edge->curve()<<"\n";
                        // er gaat iets fout bij de new-edge lt==3 ,
                        // deze is leeg ??

                        Info_edge ie;
                        if (new_edge->source()->point()==cv_vervanging.source())
                        {
                               ie.info1 = lv;
                               (*new_edge).set_info(ie);
                               ie.info1 = rv;
                               (*new_edge).twin()->set_info(ie);
                        }
                        else
                        {
                               ie.info1 = rv;
                               (*new_edge).set_info(ie);
                               ie.info1 = lv;
                               (*new_edge).twin()->set_info(ie);
                        }
                        }//einde else niet lege new-edge
                  }

            }
         }
      if (pm.is_valid())
      {
            Point A,B,wissel;
            A=p1;
            P_container::iterator pc_i;
            //cout<<"voor insert nieuwe lijn\n";
            for (pc_i=pc.begin(); pc_i!=pc.end(); pc_i++)
            {
                   B=*pc_i;
                   //cout<<A<<" "<<B<<"\n";
                   if(A != B)
                   {
                          cout << "in container "<<A<<" - "<<B<<"\n
                          Seg_cont.push_back(Segment_for_pm<Curve>
                                               (Curve(A,B),lf,rf));
                          A=B;
                   }
            }
           B=p2;
            //cout << "laatste insert  van deze lijn  " << A << B <<"\n";
            if(( A.x() != B.x() ) || ( A.y() != B.y() ))
            {
                   Seg_cont.push_back(Segment_for_pm<Curve>(Curve(A,B),lf,rf));
            }
      }
      else
      {
            cout<<"pm, niet meer valid\n";
      }
}

void Insert_edges(const Segment_container& all_segs,
                        Planar_map_ot& pm)
{
      Segment_container::const_iterator s_i;
      Planar_map_ot::Halfedge_iterator         new_edge;
      halfedge_handle hh, hh2;
      vertex_handle v1,v2;
      loct lt,lt2;

      for (s_i=all_segs.begin(); s_i!=all_segs.end(); s_i++){

         if ( (*s_i).source() != (*s_i).target() )
         {
                   Info_edge ie;
                   //new_edge= (Planar_map_ot::Halfedge_iterator)
                   //      pm.insert(Curve((*s_i).source(), (*s_i).target()));
                   Curve cv=Curve((*s_i).source(), (*s_i).target());
                   cerr << "inserting: " << cv <<"\t"<<(*s_i).get_lf_id()
```

```
                        << " left - right " <<(*s_i).get_rf_id() << "\n";
                        hh=pm.locate(cv.source(),lt);
                        cout << lt<< " @ /n";
                        if (lt==1)
                        {       v1= hh->target();
                                hh2=pm.locate(cv.target(),lt2);
                                if (lt2==1)
                                {       v2= hh2->target();
                                        pm.insert_at_vertices(cv, v1, v2);
                                }
                                else
                                pm.insert_from_vertex(cv, v1, true);
                        }
                        else
                        {       hh2=pm.locate(cv.target(),lt2);
                                if (lt2==1)
                                {       vertex_handle v2= hh2->target();
                                        pm.insert_from_vertex(cv, v2, false);
                                }
                                else
                                {
                                        if (lt2==4)
                                                pm.insert(cv);
                                        if (lt2==3 && lt ==3)
                                                pm.insert_in_face_interior
                                                        (cv, hh->face());
                                }
                        }
                if (new_edge->source()->point()==(*s_i).source())
                {

                        ie.info1 = (*s_i).get_lf_id();//info_l
                        (*new_edge).set_info(ie);
                        ie.info1 = (*s_i).get_rf_id();//info_l
                        (*new_edge).twin()->set_info(ie);
                }
                else
                {
                        ie.info1 = (*s_i).get_rf_id();//info_l
                        (*new_edge).set_info(ie);
                        ie.info1 = (*s_i).get_lf_id();//info_l
                        (*new_edge).twin()->set_info(ie);
                }
           }//einde check
        }
        return;
}

void Colorize(Planar_map_ot& pm1, Planar_map_ot& pm)
{

        face_i    fi;

        Info_face info_f;

        info_f.info1 = 0;
        info_f.info2 = 0;


        pm.unbounded_face()->set_info(info_f);

        int waarde1=0;
        int waarde2=0;

        loct lt;
        Point zoekpunt;
        halfedge_handle he, he_next;

        for(fi=pm.faces_begin(); fi!=pm.faces_end(); ++fi)
        {  if (!(*fi).is_unbounded())
          {
            he = (*fi).halfedge_on_outer_ccb();
            he_next = he;
            waarde1=0;
```

```
            waarde2=0;
            do{
                 if ((he_next->info().info1) > 0)
                {
                  waarde2=he_next->info().info1;
                  //cout <<"perceelgrens gevonden: "<<he_next->info().info1;
                }
                if ((he_next->info().info1) < 0)
                {
                  waarde1=he_next->info().info1;
                  //cout<<"gebouw: "<<he_next->info().info1;
                }
                he_next = (*he_next).next_halfedge();
            } while (he!=he_next);
            if (waarde2 == 0)
            {
               zoekpunt = midpoint(he->source()->point(),he->target()->point());
                    //midden van een lijn van maken
               cout <<"zoekpunt\t"<< zoekpunt <<"\n";
               he= pm1.locate(zoekpunt, lt);
               waarde2 = he -> info().info1;
            }
            info_f.info2 = waarde2;
            info_f.info1 = waarde1;
            (*fi).set_info(info_f);
            cout <<"gebouw("<<waarde1<<") perceel("<<waarde2<<")\n";
            cout <<"gebouw("<<(*fi).info().info1<<") perceel("
                  <<(*fi).info().info2<<")\n";
          }// einde if
        }// einde for
        return;
           // nog omgaan met eilandjes..
}// einde colorize

halfedge_handle checked_insert(Planar_map_ot& pm, Point p1, Point p2)
{
   typedef std::set<Point,compare> P_container;
   P_container  pc;
   typedef std::list<halfedge_handle> H_container;
   H_container hc1;
   typedef std::list<Curve> C_container;
   C_container hc2;
    //hc1 bevat wegtegooien elementen, hc2 bevat de vervangende elementen;

   halfedge_handle hei;

   Point tussen, tussen2, wisselp;
   Curve segm, cv, cv2;
   CGAL::Object obj;
   Point ptA, ptB, tus, tus2;
   double x1, x2, y1, y2;

   ptA=Point(p1.x(),p1.y());
   ptB=Point(p2.x(),p2.y());

   if ((p2.x() < p1.x()) || ((p2.x() == p1.x()) && (p2.y() < p1.y())) )
   {
         cv=Curve(ptB,ptA);
         wisselp=p1;
         p1=p2;
         p2=wisselp;

   }
   else
   {
         cv=Curve(ptA,ptB);
   }
   for (hei=pm.halfedges_begin(); hei!=pm.halfedges_end(); hei++)
   {
     halfedge_handle check1,check2;
     check1=hei;
     check2=hei++; //twins komen na elkaar in de lijst, niet allebei
     if (
         (       (check1->source()->point() != check2->source()->point())
```

```
    && (check1->target()->point() != check2->target()->point()))
    ||
    ( (check1->source()->point() != check2->target()->point())
    &&(check1->target()->point() != check2->source()->point()) )
    )
{
x1=hei->source()->point().x();
y1=hei->source()->point().y();
x2=hei->target()->point().x();
y2=hei->target()->point().y();
cv2=Curve(Point(x1,y1), Point(x2,y2));
if(do_intersect(cv,cv2))
    cout <<"intersectie\n";
{
    obj=intersection(cv,cv2);
    if (assign(tus, obj))
    {
           Point  tussenround(
              (double)((int)((tus.x()*100))*0.01),
              (double)((int)((tus.y()*100))*0.01));
            tussen = tussenround;
           if ( (tussen ==  (*hei).curve().source() )
           || (tussen ==  (*hei).curve().target() ) )
           {//2,4
           //cout << " gevonden punt is een eindpunt" << "\n";
           if(! ( (tussen ==  p1) || (tussen ==  p2) ))
           {//2
                  pc.insert(tussenround);
           }
           }
           else
           {//1,2
           Curve c1((*hei).source()->point(),tussen);
           Curve c2(tussen,(*hei).target()->point());
                  hc1.push_back(hei);
                  hc2.push_back(c1);
                  hc2.push_back(c2);
                  if (!( (tussen ==  p1 ) || (tussen ==  p2 ) ))
           {//1
                         pc.insert(tussenround);
                  }
           }
    }
    if (assign(segm, obj))
    {
           //cout << "gevonden i is segment" <<"\n";
           //vergelijken richting hei en segment.
           if ( (hei->source()->point().x()< hei->target()->point().x())
                  || ( (hei->source()->point().x() ==
                        hei->target()->point().x())
                  && (hei->source()->point().y() <
                        hei->target()->point().y()) )
              )
           {//zelfde orientatie als segm, anders andersom
                  tus = segm.source();
                  tus2 = segm.target();
           }
           else
           {
                  tus2 = segm.source();
                  tus = segm.target();
           }
           Point tussenround(
              (double)((int)((tus.x()*100))*0.01),
              (double)((int)((tus.y()*100))*0.01));
            tussen = tussenround;
           Point tussenround2(
              (double)((int)((tus2.x()*100))*0.01),
              (double)((int)((tus2.y()*100))*0.01));
            tussen2 = tussenround2;

           hc1.push_back(hei);
           if  (tussen ==  (*hei).source()->point() )
           {// casses 2,3,4,5,8,9
```

```
                        if (tussen==p1)
                        { //cases 3,4,9
                                if (tussen2==(*hei).target()->point())
                                {//4,9
                                        if(!(tussen2==p2))
                                        {//4
                                                pc.insert(tussenround2);
                                        }
                                        //else 9
                                }
                                else
                                {//3
                                        Curve k3(tussen2,(*hei).target()->point());
                                        hc2.push_back(k3);
                                }
                        }
                        else
                        {//2,8,5
                                if (tussen2==p2)
                                {//8,2
                                        if (tussen2==(*hei).target()->point())
                                        {//2
                                                Curve k2(tussen2,
                                                        (*hei).target()->point());
                                                hc2.push_back(k2);
                                                pc.insert(tussenround);
                                        }
                                        else
                                        {//8
                                                pc.insert(tussenround);

                                        }
                                }
                                else
                                {//5
                                        pc.insert(tussenround);
                                        pc.insert(tussenround2
                                }
                        }
                }
                else
                {//1,6,7
                        if (tussen2==(*hei).target()->point())
                        {//1,7
                                if (tussen2==p2)
                                {//7
                                        Curve k7((*hei).source()->point(),
                                                        tussen);
                                        hc2.push_back(k7);
                                }
                                else
                                {//1
                                        Curve k1((*hei).source()->point(),
                                                        tussen);
                                        hc2.push_back(k1);
                                        pc.insert(tussenround2);
                                }
                        }
                        else
                        {//6
                                Curve k61((*hei).source()->point(),tussen);
                                Curve k62(tussen2,(*hei).target()->point());
                                hc2.push_back(k61);
                                hc2.push_back(k62);
                        }
                }
        }
  }
  }
}
if (pm.is_valid())
{
      H_container::iterator hc_i1;
      for (hc_i1=hc1.begin(); hc_i1!=hc1.end(); hc_i1++)
```

```
	{
		cout<<"verwijderen edge "
		<< (*hc_i1)->source()->point().x()<<" "
		<< (*hc_i1)->source()->point().y()<<"\t"
		<< (*hc_i1)->target()->point().x()<<" "
		<< (*hc_i1)->target()->point().y()<<"\n";
		pm.remove_edge(*hc_i1);
		cout << pm.is_valid()<<"\n";
	}
}
if (pm.is_valid())
{
	C_container::iterator hc_i2;
	halfedge_handle hh, hh2;
	vertex_handle v1,v2;
	loct lt, lt2;
	for (hc_i2=hc2.begin(); hc_i2!=hc2.end(); hc_i2++)
	{	cout<<"vervanging "
		<< hc_i2->source()<< " - "
		<< hc_i2->target()<<"\n";
		if ((hc_i2->source())!= (hc_i2->target()))
		{
			//pm.insert(*hc_i2);
			hh=pm.locate(hc_i2->source(),lt);
			cout << lt <<" \n";
			if (lt==1)
			{	v1= hh->target();
				hh2=pm.locate(hc_i2->target(),lt2);
				if (lt2==1)
				{	v2= hh2->target();
					pm.insert_at_vertices(*hc_i2, v1, v2);
				}
				else
				pm.insert_from_vertex(*hc_i2, v1, true);
			}
			else
			{	hh2=pm.locate(hc_i2->target(),lt2);
				if (lt2==1)
				{	v2= hh2->target();
					pm.insert_from_vertex(*hc_i2, v2, false);
				}
				else
				{	if (lt2==4)
						pm.insert(*hc_i2);
					if (lt2==3 && lt ==3)
						pm.insert_in_face_interior
							(*hc_i2, hh->face());
				}
			}
			cout<<"lt2=" << lt <<" %\n" ;
		}
		cout << pm.is_valid()<<"\n";
	}
	//cout<<"uit container\n";
}

if (pm.is_valid())
{
	Point A,B,wissel;
	A=p1;
	P_container::iterator pc_i;
	halfedge_handle hh, hh2;
	vertex_handle v1, v2;
	loct lt,lt2;
	//cout<<"voor insert nieuwe lijn\n";
	for (pc_i=pc.begin(); pc_i!=pc.end(); pc_i++)
	{
		B=*pc_i;
		//cout<<A<<" "<<B<<"\n";
		if(A != B)
		{
			cout << "insert "<<A<<" - "<<B<<"\n";
			//pm.insert(Curve(A,B));
			hh=pm.locate(A,lt);
```

```
                        cout << lt << " #\n";
                        if (lt==1)
                        {       v1= hh->target();
                                hh2=pm.locate(B,lt2);
                                if (lt2==1)
                                {       v2= hh2->target();
                                        pm.insert_at_vertices
                                                (Curve(A,B), v1, v2);
                                }
                                else
                                pm.insert_from_vertex(Curve(A,B), v1, true);
                        }
                        else
                        {       hh2=pm.locate(B,lt2);
                                if (lt2==1)
                                {       v2= hh2->target();
                                        pm.insert_from_vertex
                                                (Curve(A,B), v2, false);
                                }
                                else
                                {
                                        if (lt2==4)
                                                pm.insert(Curve(A,B));
                                        if (lt2==3 && lt ==3)
                                                pm.insert_in_face_interior
                                                        (Curve(A,B), hh->face());
                                }
                        }
                        //cout << "gesplitst insert II ";
                        cout<<"lt2=" << lt <<" %\n" ;
                        A=B;
                }
        }
       B=p2;
        if(( A.x() != B.x() ) || ( A.y() != B.y() ))
        {
            //return pm.insert(Curve(A,B));
            hh=pm.locate(A,lt);
                        if (lt==1)
                        {       vertex_handle v1= hh->target();
                                hh2=pm.locate(B,lt2);
                                if (lt2==1)
                                {       vertex_handle v2= hh2->target();
                                        pm.insert_at_vertices
                                                (Curve(A,B), v1, v2);
                                }
                                else
                                pm.insert_from_vertex(Curve(A,B), v1, true);
                        }
                        else
                        {       hh2=pm.locate(B,lt2);
                                if (lt2==1)
                                {       vertex_handle v2= hh2->target();
                                        pm.insert_from_vertex
                                                (Curve(A,B), v2, false);
                                }
                                else
                                {
                                        if  (lt2==4)
                                                pm.insert(Curve(A,B));
                                        if (lt2==3 && lt ==3)
                                                pm.insert_in_face_interior
                                                        (Curve(A,B), hh->face());
                                }
                        }
        }
    }
    else
    {
          cout<<"pm, niet meer valid\n";
    }

}
```

```
void write(Planar_map_ot& pm3)
{
  ofstream overlayresultaat ("./uitvoer/deelgebouw.txt", ios::out);
  if (!overlayresultaat)
  {
        cerr << "cannot open output file\n";
        exit (1);}
  ofstream attribuutresultaat ("./uitvoer/deelgebouw_attributen.txt", ios::out);
  if (!attribuutresultaat)
  {
        cerr << "cannot open output file\n";
        exit (1);}
  ofstream ll ("./uitvoer/ll.txt", ios::out);
  if (!ll)
  {
        cerr << "cannot open output file\n";
        exit (1);}

  if (pm3.is_valid())
  {
    Planar_map_ot::Face_iterator fit3;
    Planar_map_ot::Halfedge_handle eh, eh_next;

    Point pt, pt_next, pt_in;
    int teller=0;
    attribuutresultaat<<"ID\t"<<"ID_gebouw\t"
         <<"Perceelnummer\t"<<"#ACN\t"<<"Aantal buren\t"<<"Oppervlakte\n";

    typedef std::set<int,comp_id> buren;
    for (fit3= pm3.faces_begin();
         fit3!=pm3.faces_end();
         ++fit3)
    {
       Info_face fi;
       if (!(*fit3).is_unbounded())
       {
          Polygon p;
          buren brn;
         teller=teller+1; //ID
         double xin, yin;

         fi =(*fit3).info();
         if (!(fi.info1==1) && !(fi.info1==-1))
         {
                 attribuutresultaat<<teller<<"\t"
                              << fi.info1<<"\t" << fi.info2<<"\t"<<"0\t";
                 eh = (*fit3).halfedge_on_outer_ccb();
                 eh_next = eh;
                 pt = (*eh).source()->point();
                 xin=(pt.x()), yin=(pt.y());
                 pt_in = Point(0,0);
                do
                {
                   overlayresultaat <<teller
                              <<",\t"<< xin <<",\t"<< yin << "\n";
                   p.push_back(Pntp(xin,yin));
                   face_handle buurface = eh_next->twin()->face();
                   int buur=(buurface->info().info1);
                   int buur2=(buurface->info().info2);
                   if (buur<-1)
                   {
                       brn.insert(buur2);
                   }
                   eh_next = (*eh_next).next_halfedge();
                   pt_next =(*eh_next).source()->point();
                   xin=(pt_next.x()), yin=(pt_next.y());
                 } while (eh!=eh_next);
                //overlayresultaat << "\n";
                attribuutresultaat << brn.size() <<"\t"<<p.area()<<"\n";
          }
          if (fi.info1==-1)
          {
                 eh = (*fit3).halfedge_on_outer_ccb();
                 eh_next = eh;
```

```
                pt = (*eh).source()->point();
                xin=(pt.x()), yin=(pt.y());
               do
               {
                  ll <<teller <<",\t"<< xin <<",\t"<< yin << "\n";
                  eh_next = (*eh_next).next_halfedge();
                  pt_next =(*eh_next).source()->point();
                  xin=(pt_next.x()), yin=(pt_next.y());

                } while (eh!=eh_next);
        }//endif

      }//if unbounded

  }//einde for
  }
  else {
    std::cout << "The overlay is not a valid planar map!\n";
  }
  ll.close();
  overlayresultaat.close();
  attribuutresultaat.close();
}

void write_no_info(Planar_map_ot& pm1, Planar_map_ot& pm2)
{
  ofstream building ("./uitvoer/buildings.txt", ios::out);
  if (!building)
  {
         cerr << "cannot open output file\n";
         exit (1);}
  ofstream parcel ("./uitvoer/parcels.txt", ios::out);
  if (!parcel)
  {
         cerr << "cannot open output file\n";
         exit (1);}

  Planar_map_ot::Halfedge_handle eh, eh_next;
  Point pt, pt_next, pt_in;

  if (pm1.is_valid())
  {
    Planar_map_ot::Face_iterator fit3;

    for (fit3= pm1.faces_begin();
         fit3!=pm1.faces_end();
         ++fit3)
    {
       if (!(*fit3).is_unbounded())
       {
                eh = (*fit3).halfedge_on_outer_ccb();
                eh_next = eh;
                pt = (*eh).source()->point();
                double xin, yin;
                xin=(pt.x()), yin=(pt.y());

               do
               {
                  building <<eh_next->info().info1 <<",\t"<< xin <<",\t"<< yin <<
        "\n";
                  eh_next = (*eh_next).next_halfedge();
                  pt_next =(*eh_next).source()->point();
                  xin=(pt_next.x()), yin=(pt_next.y());
                } while (eh!=eh_next);
         }//if
      }//for
  }//ifisvalid
  else {
    std::cout << "gebouwen is not a valid planar map!\n";
  }
  building.close();

  if (pm2.is_valid())
  {
```

```
    Planar_map_ot::Face_iterator fit2;

    for (fit2= pm2.faces_begin();
         fit2!=pm2.faces_end();
         ++fit2)
    {
       if (!(*fit2).is_unbounded())
       {
                 eh = (*fit2).halfedge_on_outer_ccb();
                 eh_next = eh;
                 pt = (*eh).source()->point();
                 double xin, yin;
                 xin=(pt.x()), yin=(pt.y());

                do
                {
                   parcel <<eh_next->info().info1 <<",\t"
                               << xin <<",\t"<< yin << "\n";
                   eh_next = (*eh_next).next_halfedge();
                   pt_next =(*eh_next).source()->point();
                   xin=(pt_next.x()), yin=(pt_next.y());
                 } while (eh!=eh_next);
          }//if
      }//for
  }//ifisvalid
  else {
    std::cout << "percelen is not a valid planar map!\n";
  }
  parcel.close();
}

int main()
{
  CGAL::Pm_naive_point_location<Planar_map_ot> pl_strategy;
//  CGAL::Pm_walk_along_line_point_location<Planar_map_ot> pl_strategy;
//pl_strategy(bool preprocess=true);

  Planar_map_ot pm1(&pl_strategy);
  Planar_map_ot pm2(&pl_strategy);

  cerr.precision(15);
  cout.precision(15);

  Info_face info_f;
  info_f.info1=0;
  info_f.info2=0;
  face_handle uf1=pm1.unbounded_face();
  uf1->set_info(info_f);
  face_handle uf2=pm2.unbounded_face();
  uf2->set_info(info_f);

  ifstream gebouw ("lander.txt", ios::in);
  ifstream perceel ("banders.txt", ios::in);

  if (!gebouw)
  {
         cerr << "cannot open input file\n";
         exit (1);}
  if (!perceel)
  {
         cerr << "cannot open input file\n";
         exit (1);}

        int ID;
         double Xcoord;
         double Ycoord;
         double Xcoordnext;
         double Ycoordnext, test;
         halfedge_handle A;
         Point pntA, pntB, pntC;
         //int tel=0;
         int first=0;
  while (!(test == -2))
  {
```

```
        gebouw >> ID >> Xcoord >> Ycoord ;
        pntA = Point((Xcoord)*1, (Ycoord)*1);

        gebouw >> test;
        //cout<<"\n"<< tel <<"\n";
        //tel=tel+1;
        //int tel2=10;
        do
        {
          //cout<< tel2 <<"\t";
          //tel2=tel2+1;
          Xcoordnext = test;
          gebouw >> Ycoordnext;
          pntB = Point((Xcoordnext)*1,(Ycoordnext)*1);
          first=first+1;
          //cout << "voor elke keer ci" << "\n" <<pntA <<" "<<pntB <<"\n";
          if(pntA !=pntB)
          {
                if(first==1)
                {
                        pm1.insert(Curve(pntA,pntB));
                }
                else
                {
                        checked_insert(pm1, pntA, pntB);
                }
          }
          else
          {
                cout <<"fout in invoer, zelfde punt twee keer "<<pntA<<"\n";
          }
          gebouw >> test;
          //cout << "test = "<< test<<"\n" ;
          pntA=pntB;
        }while ((!(test == -1)) && (!(test == -2)));
  }
  cout << "klaar met inlezen gebouw" <<"\n";
  gebouw.close();


//void Toekennen_id (Planar_map_ot pm1, int n)
//{
   face_i fit;
   Info_face if1;
   Info_edge ie1;
   halfedge_handle eh, eh_next;

  // cout << " fit ";

  if (pm1.is_valid())
  {
        //halfedge_handle eind;
        Point eind1,eind2;

        typedef Planar_map_ot::Halfedge_iterator ph_edge_i;
        typedef Planar_map_ot::Holes_iterator ph_i;
        typedef Planar_map_ot::Ccb_halfedge_circulator ch_c;

        ph_i hi=pm1.unbounded_face()->holes_begin();
        ph_edge_i he1;
        int hls=0;

        typedef std::list<Curve> C_cont;
        C_cont misser;
        C_cont::iterator mi;

        do
        {       ch_c cc =*hi;
                he1 = (ph_edge_i)(*cc).twin();
                halfedge_handle he_tw =(*he1).twin();//buitenste

                halfedge_handle he1_next=(*he_tw).next_halfedge();
                halfedge_handle he2=(*he1_next).twin();
```

```
        he1_next=(*he1_next).next_halfedge();
        halfedge_handle he_tw_tw=he1_next;
        Info_edge ie_unb;
        ie_unb.info1=-2;
        ie_unb.info2=-2;

        if ((he2->face()->is_unbounded())
                && (he2->twin()->face()->is_unbounded()))
        {
                he2->set_info(ie_unb);
                do
                {
                        cout << "he_next: "<< he1_next->curve()
                                <<" twinprevios: "<<he2->curve()<<"\n";
                        if (he2==he1_next)
                        {
                                cout <<"zelfde: "<<he2->curve()<<"\n";
                                if(hls==1)
                                {
                                        //eind=he1_next;
                                        eind2=(*he1_next).source()->point();
                                        hls=hls+1;
                                }
                                else
                                {
                                        eind1=(*he1_next).source()->point();
                                 if( (squared_distance(eind1, eind2)) <= 1.0)
                                        {
                                        misser.push_back(Curve(eind1,eind2));
                                        cout<<"lijntoevoeging\n";
                                        }
                                        else
                                        {
                                                ie1.info2=-1;
                                                //eind->set_info(ie1);
                                                he1_next->set_info(ie1);
                                        }
                                        hls=0;
                                }

                        }
                        he2=(*he1_next).twin();
                        he1_next=(*he1_next).next_halfedge();
                }while (he1_next!=he_tw_tw);
        }
        hi++;

}while(!(hi==pm1.unbounded_face()->holes_end()));
halfedge_handle hh,hh2;
vertex_handle v1,v2;
loct lt, lt2;
for (mi=misser.begin(); mi!=misser.end(); mi++)
{
        //pm1.insert(*mi);
        hh=pm1.locate(mi->source(),lt);
        cout << lt <<" %\n";
                if (lt==1)
                {       v1= hh->target();
                        hh2=pm1.locate(mi->target(),lt2);
                        if (lt2==1)
                        {       v2= hh2->target();
                                pm1.insert_at_vertices(*mi, v1, v2);
                        }
                        else
                        pm1.insert_from_vertex(*mi, v1, true);
                }
                else
                {       hh2=pm1.locate(mi->target(),lt2);
                        if (lt2==1)
                        {       v2= hh2->target();
                                pm1.insert_from_vertex(*mi, v2, false);
                        }
                        else
                        {
```

```
                                        if (lt==4)
                                        {
                                               pm1.insert(*mi);

                                        }
                                }
                        }
                        cout<<"lt2=" << lt <<" %\n" ;
                }
                cout << "toekennen ID's \n";
                int m = -100;
                if1.info1=-1;
                if1.info2=-1;
                if1.info3=-1;
                ie1.info1=-1;
                ie1.info2=-1;

                for (fit=pm1.faces_begin(); fit!=pm1.faces_end(); ++fit)
                {
                if (!(*fit).is_unbounded())
                {       m = m - 1;
                        cout << m << " \n";
                        if1.info1 = m;
                        (*fit).set_info(if1);
                         eh=(*fit).halfedge_on_outer_ccb();
                         ie1.info1 = m;
                         eh_next = eh;
                         eh_next->set_info(ie1);
                         do{
                                eh_next = (*eh_next).next_halfedge();
                                eh_next->set_info(ie1);
                                if (eh_next->twin()->face()->is_unbounded())
                                {
                                        ie1.info1 = 0;
                                        eh_next->twin()->set_info(ie1);
                                        ie1.info1 = m;
                                }
                                cout << "x";
                        } while (eh!=eh_next);
                 }
                 else
                 {cout<<"unb face\n";}
                }
        }
        else
        {
                cout << "planar map gebouw is niet valid" <<"\n";
                exit(1);}
//}

        Planar_map_ot::Halfedge_iterator hei;
        for (hei=pm1.halfedges_begin(); hei!=pm1.halfedges_end(); hei++)
        {
        cout << hei->source()->point()<<" "<< hei->target()->point()
                << " ID: " <<(*hei).info().info1<<"\n";}

        int   lobj, robj;
        Info_face if2;
        Info_edge ie2;

        if2.info2=0;
        ie2.info2=0;
        Curve cv;
        test =0.0;
        cout << "inlezen perceel\n";
        first=0;
        while (!(test==-2))
        {
                perceel >> lobj >> robj;
                perceel >> Xcoord >> Ycoord;
               perceel >> test;
               Xcoordnext =  Xcoord;
               Ycoordnext = Ycoord;
                do
```

```
          {
            Xcoord = Xcoordnext;
            Ycoord = Ycoordnext;
            Xcoordnext = test;
            //cout <<test<<"\n";
            perceel>> Ycoordnext;
            pntA = Point((Xcoord*1),(Ycoord)*1);
           pntB = Point((Xcoordnext*1),(Ycoordnext)*1);
           first=first+1;
           if ((pntA.x()<pntB.x())
                 || ( ( pntA.x()==pntB.x() ) && ( pntA.y()<pntB.y() ) ) )
           {
                 cv=Curve(pntA,pntB);
           }
           else
           {
                 cout << "omwisselen\n";
                 cv=Curve(pntB,pntA);
                 int idwissel=lobj;
                 lobj=robj;
                 robj=idwissel;

           }
           //A=pm2.insert(cv);
           if (first==1)
           {
                 A=pm2.insert(cv);
           }
           else
           {
                 A=checked_insert(pm2, cv.source(), cv.target());
            }
            //ie2.info1 = lobj;
            //A->set_info(ie2);
            //ie2.info1 = robj;
            //A->twin()->set_info(ie2);

            perceel>>test;
          }while ((!(test == -1.0)) && (!(test == -2.0)));
   }

   if (!(pm2.is_valid()))
   {
         cout << "planar map perceel is niet valid\n";
   }
   cout <<"klaar met inlezen perceel\n";
   perceel.close();

  // Toekennen_id (pm2, 100000);
{
         cout << "toekennen ID's \n";
         int m = 1000;

         for (fit=pm2.faces_begin(); fit!=pm2.faces_end(); ++fit)
         {
         if (!(*fit).is_unbounded())
         {      m = m + 1;
                cout << m << " \n";
                if1.info1 = m;
                (*fit).set_info(if1);
                 eh=(*fit).halfedge_on_outer_ccb();
                 ie1.info1 = m;
                 eh_next = eh;
                 eh_next->set_info(ie1);
                 do{
                        eh_next = (*eh_next).next_halfedge();
                        eh_next->set_info(ie1);
                        if (eh_next->twin()->face()->is_unbounded())
                        {
                               ie1.info1 = 0;
                               eh_next->twin()->set_info(ie1);
                               ie1.info1 = m;
                        }
                        cout << "x";
```

```
                } while (eh!=eh_next);
          }
          else
          {cout<<"unb face\n";}
         }//einde for

}

  write_no_info(pm1,pm2);

  halfedge_handle e_i;

  Segment_container Seg_cont;

  for (e_i=pm2.halfedges_begin(); e_i!=pm2.halfedges_end(); e_i++)
  {
        e_i++;
        Point p1=(*e_i).source()->point();
        Point p2=(*e_i).target()->point();
        int lf=(*e_i).info().info1;
        int rf=(*e_i).twin()->info().info1;
        cout << "left("<< lf <<")  right("<<rf<<")\n";
        checked_insert_overlay(pm1, p1, p2, lf, rf, Seg_cont);
  }
  cout << "\n intersectiepunten berekenen klaar,
               \n nu nog inlezen in de kaart\n\n";
  Insert_edges(Seg_cont,pm1);
  cout <<"klaar met overlay, beginnen met terughalen identificatie\n";
  Colorize(pm2,pm1);

  // inlezen ACN per regel en weerwegschrijven plus gebouw en perceel_id
   Planar_map_ot::Halfedge_iterator hei2;
   for (hei2=pm1.halfedges_begin(); hei2!=pm1.halfedges_end(); hei2++)
   {
   cout << hei2->source()->point()<<" "<< hei2->target()->point()
   << " ID: " <<(*hei2).info().info1<<"\n";}

  write(pm1);
  cout   <<"aantal punten: "<< pm1.number_of_vertices()
         <<"\n aantal lijnen: " <<pm1.number_of_halfedges()
         <<"\n aantal vlakken: " <<pm1.number_of_faces()<<"\n";
  return 0;
}
```

C2

"gen2shp"-avenue script

komt van de volgende site vandaan: http://gis.esri.com/arcscripts/
gevonden onder de naam ASCII-to-Polygonshape

```
'Converts ASCII-File to polygon shapefile
'script is a simple alteration of the example script gps2shp that comes with ArcView
'it creates polygons from points specified by an id
'file should look like this
' 1001, x1, y1
' 1001, x2, y2
' 1001, x3, y3
' 1002, x1, y1  .. and so on
'-----------------------------------------------------------------
'gps2shp modified by Michael Herter
'-----------------------------------------------------------------
gpsName = FileDialog.Show("*.*","ASCII-File","Select File to Convert")
if (gpsName = nil) then
  exit
end

gpsFile  = LineFile.Make(gpsName, #FILE_PERM_READ)
totalRecs = gpsFile.GetSize

' Specify the output shapefile...
defaultName    = FileName.Make("$HOME").MakeTmp("shape","shp")
shpName  = FileDialog.Put( defaultName,"*.shp","Output Shape File" )
if (shpName = nil) then
  exit
end

shpName.SetExtension("shp")
shpFTab = FTab.MakeNew(shpName, Polygon)

fields = List.Make
idfield = Field.Make ("ID", #FIELD_BYTE, 6, 0)
fields.Add (idfield)

shpFTab.AddFields(fields)
shpField = shpFTab.FindField("Shape")

gpsRec = 0

av.ShowStopButton
av.ShowMsg("Converting"++gpsName.GetBaseName+"...")

pointList = List.Make

'buf = gpsFile.ReadElt

buf = gpsFile.ReadElt
gpsTokens = buf.AsTokens(",")
idold = gpstokens.get(0).asnumber
thePoint = gpsTokens.Get(1).trim.AsNumber@gpsTokens.Get(2).trim.AsNumber
pointList.Add( thePoint )

while (gpsfile.isatend.not)

  buf = gpsFile.ReadElt
  gpsTokens = buf.AsTokens(",")
  idnr = gpstokens.get(0).asnumber

  if (idold <> idnr) then

      rec = shpFTab.AddRecord
      shpFTab.SetValue( idField, rec, idold)
      pl = Polygon.Make( {pointList} )
      shpFTab.SetValue( shpField, rec, pl )
      pointlist.empty
      idold = idnr
  end

  if (buf = Nil) then
    break
```

```
  end

  thePoint = gpsTokens.Get(1).trim.AsNumber@gpsTokens.Get(2).trim.AsNumber
  pointList.Add( thePoint )

  gpsRec = gpsRec + 1
  progress = (gpsRec / totalRecs) * 100
  proceed = av.SetStatus( progress )

  if (proceed.Not) then
    av.ClearStatus
    av.ShowMsg( "Stopped" )
    exit
  end
end

rec = shpFTab.AddRecord
shpFTab.SetValue( idField, rec, idold)
pl = Polygon.Make( {pointList} )
shpFTab.SetValue( shpField, rec, pl )

av.ClearStatus
av.ClearMsg

shpFTab.Flush

MsgBox.Info( gpsRec.AsString++"records converted." ,"Conversion Completed" )
```

C3 **"join_shape_attribuut" avenuescript**

```
theTable = av.GetProject.FindDoc("deelgebouw_attributen.txt").GetVtab

theView=av.GetActiveDoc
theTheme=theView.GetThemes.Get(0)
theTable2=theTheme.GetFtab


theTable2.Join(theTable2.FindField("Id"),theTable, theTable.FindField("Id"))
```

C4 weer inlezen in de database.”**read_overlayattributen.sh"**

```
#!/bin/sh
#
set -v

DB=${1:-apeldoorn}
TMPDIR=`pwd`

#rm -f overlay.dat

sql -f8N15.5 $DB << EOF
set nologging\p\g

\date
drop table tmp1\p\g
create table overlay_attributen(
        dg_id   integer4,
        gebouw_id integer4,
        perceel_id      integer4,
        acn_in_geb      char(2),
         buren          char(2),
        oppervlak       integer4)
        \p\g

copy overlay_attributen(
         dg_id=c0tab,
         gebouw_id=c0tab,
         perceel_id=c0tab,
         acn_in_geb=c0tab,
         buren=c0tab,
         oppervlak=c0nl)
         from '$TMPDIR/overlayresultaat_attributen.dat'\p\g
EOF
```

Bijlage D kadasterclassificaties

De classificaties zoals ze bij het kadaster op het moment van de bevriezingsdatum golden.

Deelgebouwen									Gebouwen					Percelen					ACN			AKR_a		
Kadaster classificatie	Deelgebouw_id	perceel_id,	gebouw_id	buren	area	n_acn	n_hnr	Cc11t/mn15	Gebouw_id	n_deelgebouwen	n_ACN	n_Parcels	n_hnr	Perceel_id	Cultuur_code	n_ACN	n_Gebouwen	n_deelgebouwen	ACN_id	Number	Deelgebouw_id	G_akr_objectnummer	A_akr_objectnummer	Aantallen in de testset Apeldoorn
-1															**<16**									**957**
10																							**T**	**297**
-3															**>15**									**2454**
20				**0**		**1**	**1**	**T**		**1**														**9010**
30				**1**			**1**	**T**		**2**	**<3**	**2**												**5179**
40				**1**		**1**	**1**	**T**		**>2**		**>2**								**<2**				**3037**
50				**>1**			**<2**	**T**		**>2**		**>2**	**>0**											**10315**
98				**0**		**0**										**<**	**>**							**715**
23				**0**		**0**	**0**									**1**	**1**							**24**
33				**1**		**0**	**1**			**2**						**1**	**1**							**4**
42				**1**		**0**	**0**	**T**								**1**	**1**							**70**
59				**>1**		**0**		**T**		**>2**		**>2**				**1**	**1**							**11**
GEDEELTELJKE ACN-UPDATE																								
15								**T**												**>7**				**226**
21				**0**		**0**	**1**									**1**								**1220**
22				**0**		**1**	**0**																	**156**
24				**0**	**<120**	**1**	**1**								**00**									**224**
31				**1**		**1**		**T**		**2**		**2**												**216**
32				**1**	**<120**		**1**			**2**	**2**	**2**			**00**									**124**
34				**1**			**2**				**<=2**									**1**				**22**
35				**0**		**2**														**1**				**913**
36				**1**	**%>50**		**2**				**<=2**									**1**				**0**
38				**0**			**2**			**1**	**2**													**27**
39						**2**	**2**																	**95**
41				**1**		**<2**	**1**	**F**		**>2**		**>2**			**00**									**167**
43				**1**		**1**	**1**	**T**		**>2**		**>2**				**<2**								**0**
44				**1**		**<2**		**T**		**>2**		**>2**								**<2**				**1574**
51				**>1**		**1**																		**407**
52				**>1**		**<2**	**<2**			**>2**		**>2**												**41**
53				**>1**	**40-50**		**<2**	**F**		**>2**		**>2**	**>0**		**00**									**0**
16					**%<30**	**>2**																		**143**
55					**%>50**	**>2**																		**1250**
56					**%>40**	**>2**																		**397**
57					**%>30**	**>2**																		**109**
58					**50-100**	**>2**		**F**							**00**									**0**
K_classificatie	Deelgebouw_id	perceel_id,	gebouw_id	buren	area	n_acn	n_hnr	Cc11t/mn15	Gebouw_id	n_deelgebouw	n_ACN	n_Parcels	n_hnr	Perceel_id	Cultuur_code	n_ACN	n_Gebouwen	n_deelgebouw	ACN_id	Number	Deelgebouw_id	G_akr_objectnr	A_akr_objectnr	Aantallen
Deelgebouwen									Gebouwen					Percelen					ACN			AKR_a		

Bijlage E Het programma dat de gehele methode uitvoerd

E1 Hoofd programma

```
#!/bin/sh

set -v

make_acn.sh
make_akr_attributes.sh
make_gebouwen.sh
make_parcels.sh
make_text.sh

do_overlay.sh

post_text.sh
post_acn.sh

dump_arc.sh
load_ingres.sh

update_acn.sh
update_huisnummers.sh
update_deelgebouwen.sh
update_gebouwen.sh
update_percelen.sh

all_classify.sh
```

E2.1 Voor-bereiding ACN

```
#!/bin/sh
#
# Author: Wilko Quak (quak@geo.tudelft.nl)
#
# Create an arcinfo maplayer from an ACN adrescoordinaten dump.
# uses ARCDIR environment variable.
#
set -v
#
# Lees de globale variabelen in.
#
. globals.sh

rm -f $RESULTDIR/acn.e00 $RESULTDIR/acn.e00.gz

bunzip2 -c $ACNFILE | \
   acn_to_arc.perl > $TMPDIR/acn_arc.gen

(cd $ARCDIR ; arc ) << EOF
precision double highest
kill acn all
create acn
generate acn
input $TMPDIR/acn_arc.gen
points
quit
build acn point
index acn POINT
quit
EOF

bunzip2 -c $ACNFILE | \
  acn_to_info.perl > $TMPDIR/acn_info.txt

(cd $ARCDIR ; arc ) << EOF
tables
select acn_attributes
erase acn_attributes
y
```

```
define acn_attributes
acn-id 4 5 B
woonplaats 40 40 C
straatnaam 40 40 C
postcode 40 40 C
Huisnr 5 5 C
Toev 5 5 C
Adrtxt 10 10 C
count 4 4 B

select acn_attributes
add from $TMPDIR/acn_info.txt
quit
quit
EOF

rm -f $RESULTDIR/acn.e*

(cd $ARCDIR ; arc ) << EOF
indexitem acn_attributes acn-id
indexitem acn.pat acn-id
joinitem acn.pat acn_attributes acn.pat acn-id
tables
select arc.pat
alter acn-id,acn-id,12,,,
quit
export cover acn $RESULTDIR/acn
quit
EOF

gzip $RESULTDIR/acn.e00

rm -f $TMPDIR/acn_arc.gen
rm -f $TMPDIR/acn_info.txt
```

E2.2 Voor-bereiding gebouwen

```
#!/bin/sh
#
# uses ARCDIR environment variable.
#
set -v
#
# Lees de globale variabelen in.
#
. globals.sh

gunzip -c $LKIDIR/xfio_line.dat.gz | \
   process_xfio_line.perl > $TMPDIR/xfio_line.gen

(cd $ARCDIR ; arc ) << EOF
/*
/* Load the spatial information of all lines into a
/* layer xfio_line.
/*
precision double highest
kill xfio_line all
create xfio_line
generate xfio_line
input $TMPDIR/xfio_line.gen
lines
quit
build xfio_line line
index xfio_line line
quit
EOF

gunzip -c $LKIDIR/xfio_line.dat.gz | \
  xfio_line_to_info.perl > $TMPDIR/xfio_line.txt

(cd $ARCDIR ; arc ) << EOF
/*
/* Load the attribute information of all lines into a table xfio_line_att.
```

```
/*
tables
select xfio_line_att
erase xfio_line_att
y
define xfio_line_att
xfio_line-id 4 5 B
ogroup 4 4 I
classif 4 4 I
tmax 4 4 B

select xfio_line_att
add from $TMPDIR/xfio_line.txt
quit
/*
/* Join the attributes in xfio_line_att to the lines in xfio_line.aat.
/*
joinitem xfio_line.aat xfio_line_att xfio_line.aat xfio_line-id
quit
EOF

(cd $ARCDIR ; arc ) << EOF
kill huisgrenzen all
reselect xfio_line huisgrenzen LINE
res classif in {11,21} and ogroup = 1 and tmax = 0

n
n
index huisgrenzen line
quit
EOF

(cd $ARCDIR ; arc ) << EOF
kill gebouwen_snap all
copy huisgrenzen gebouwen_snap
arcedit
editcoverage gebouwen_snap
editfeature arc
select all
unsplit none
save
quit
/*
/* Snap
/*
index gebouwen_snap line
arcedit
edit gebouwen_snap
editfeature arc
select dangle
snapcoverage gebouwen_snap
snapping CLOSEST 300
snapfeatures arc arc
snap
save
quit
/*
/* Matchnode
/*
kill gebouwen_matc all
index gebouwen_snap line
matchnode gebouwen_snap gebouwen_matc 300 gebouwen_snap extend
/*
/* Clean gebouwen
/*
kill gebouwen all
copy gebouwen_matc gebouwen
clean gebouwen gebouwen 300 50 poly
/*
/* Set ishuis attribute.
/*
tables
select gebouwen.pat
alter gebouwen-id,gebouwen-id,12,,,
```

```
additem gebouwen.pat ishuis 1 1 I 0
quit
arcedit
ec gebouwen
ef poly
select all
calc ishuis = 1
quit
y
y
kill gebouwen2 all
dissolve gebouwen gebouwen2 ISHUIS
kill gebouwen
copy gebouwen2 gebouwen
kill gebouwen2
createlabels gebouwen 1000000
build gebouwen
/*
/* Unsplit
/*
arcedit
editcoverage gebouwen
editfeature arc
select all
unsplit none
save
quit
dropitem gebouwen.pat gebouwen.pat ishuis
/* kill gebouwen_matc all
/* kill gebouwen_snap all
clean gebouwen_matc gebouwen_matc 300 50 poly
clean gebouwen_snap gebouwen_snap 300 50 poly
build gebouwen poly
quit
EOF

rm -f $TMPDIR/xfio_line.txt
rm -f $TMPDIR/xfio_line.gen
```

E2.3 Voor-bereiding percelen

```
#!/bin/sh
#
# uses ARCDIR environment variable.
#
set -v

. globals.sh

gunzip -c $LKIDIR/xfio_parcel.dat.gz | \
   process_xfio_parcel.perl > $TMPDIR/xfio_parcel.gen

gunzip -c $LKIDIR/xfio_boundary.dat.gz | \
   process_xfio_boundary.perl > $TMPDIR/xfio_boundary.gen
#
# Percelen zijn opgebouwd uit boundaries en middelpunten.
# De grenzen komen uit xfio_boundary. De middelpunten uit
# xfio_parcel. Deze twee files worden ingeladen, in de
# layer parcels gezet, en er wordt een kaartlaag van gemaakt.
#
(cd $ARCDIR ; arc ) << EOF
precision double highest
kill parcels
create parcels
generate parcels
input $TMPDIR/xfio_parcel.gen
points
input $TMPDIR/xfio_boundary.gen
lines
quit
build parcels poly
index parcels poly
quit
```

```
EOF
#
# Laad de administratieve attributen uit lki
# - Dit is het g_akr_objectnummer. Eigenlijk hetzelfde als x_akr_objectnummer
#   alleen in dit geval zijn alleen gehele percelen beschreven.
#
gunzip -c $LKIDIR/xfio_parcel.dat.gz | \
  xfio_parcel_to_info.perl > $TMPDIR/parcels_attributes.txt

(cd $ARCDIR ; arc ) << EOF
tables
select parcels_attributes
erase parcels_attributes
y
define parcels_attributes
parcels-id 4 5 B
g_akr_objectnummer 17 17 C

select parcels_attributes
add from $TMPDIR/parcels_attributes.txt
quit
quit
EOF
#
# Join de niet geografische attributen aan de geografische.
#
(cd $ARCDIR ; arc ) << EOF
indexitem parcels_attributes parcels-id
indexitem parcels.pat parcels-id
joinitem parcels.pat parcels_attributes parcels.pat parcels-id
tables
select parcels.pat
alter parcels-id,parcels-id,12,,,
quit
quit
EOF

rm -f $TMPDIR/xfio_parcel.gen
rm -f $TMPDIR/xfio_boundary.gen
rm -f $TMPDIR/parcels_attributes.txt
```

E2.3 Voorbereiding huisnummers

```
#!/bin/sh
#
set -v

. globals.sh

gunzip -c $LKIDIR/xfio_text.dat.gz | \
   process_xfio_text.perl > $TMPDIR/xfio_text.gen

(cd $ARCDIR ; arc ) << EOF
/*
/* Load the spatial information of all lines into a
/* layer xfio_line.
/*
precision double highest
kill xfio_text all
create xfio_text
generate xfio_text
input $TMPDIR/xfio_text.gen
points
quit
build xfio_text point
index xfio_text point
quit
EOF

gunzip -c $LKIDIR/xfio_text.dat.gz | \
  xfio_text_to_info.perl > $TMPDIR/xfio_text.txt

(cd $ARCDIR ; arc ) << EOF
```

```
/*
/* Load the attribute information of all lines into a
/* table xfio_line_att.
/*
tables
select xfio_text_att
erase xfio_text_att
y
define xfio_text_att
xfio_text-id 4 5 B
ogroup 4 4 I
classif 4 4 I
otext 40 40 C

select xfio_text_att
add from $TMPDIR/xfio_text.txt
quit
/*
/* Join the attributes in xfio_text_att to the texts in xfio_text.aat.
/*
joinitem xfio_text.pat xfio_text_att xfio_text.pat xfio_text-id
quit
EOF
```

E3 Bijbehorend perceel zoeken

```
#!/bin/sh
#
# Author: Wilko Quak (quak@geo.tudelft.nl)
#
set -v
#
# Lees de globale variabelen in.
#
. globals.sh

(cd $ARCDIR ; arc ) << EOF
/*
/* For every acn coordinate find out in which parcel it is.
/* This results in a new attribute g_akr_objectnummer in
/* acn.pat
/*
kill acn2 all
intersect acn PARCELS acn2 point
dropitem acn2.pat acn2.pat acn#,acn-id,parcels#,parcels-id,area,perimeter
kill acn all
copy acn2 acn
kill acn2
/*
/* For every acn coordinate find out in which deelgebouw it is.
/* This results in a new attribute g_akr_objectnummer in
/* acn.pat
/*
kill acninhuis all
intersect acn overlay acninhuis point
quit
EOF

(cd $ARCDIR ; arc ) << EOF
dropitem acninhuis.pat acninhuis.pat
area,perimeter,acn#,OVERLAY#,NBUREN,G_AKR_OBJECTNUMM,PARCELS-ID,acn-id
dropitem acn.pat acn.pat area,perimeter

tables
select acn.pat
alter acn-id,acn-id,12,,,
select acninhuis.pat
alter acninhuis-id,acninhuis-id,12,,,
quit
quit
EOF
EOF
```

E4 Inladen

```
#!/bin/sh
# Author: Wilko Quak (quak@geo.tudelft.nl)
#
# Loads all files dumped by arc to ingres.
#
# - uses GEOHOME environment variable to find the copyrel
#   application.
#
set -v
#
# Lees de globale variabelen in.
#
. globals.sh
#
# Maak tabellen aan die gebruikt worden voor de gebouwenanalyse.
#
sql $DB << EOF
drop table acn_in_deelgebouw \p\g
create table acn_in_deelgebouw(
   acn_id             integer4,
   deelgebouwen_id    integer4
) \p\g
grant select on acn_in_deelgebouw to public;\p\g


drop table pap_in_deelgebouw \p\g
create table pap_in_deelgebouw(
   pap_id             integer4,
   deelgebouwen_id    integer4
) \p\g
grant select on pap_in_deelgebouw to public;\p\g

drop table hnr_in_deelgebouw \p\g
create table hnr_in_deelgebouw(
   hnr_id             integer4,
   deelgebouwen_id    integer4
) \p\g
grant select on hnr_in_deelgebouw to public;\p\g

drop table gebouwen \p\g
create table gebouwen(
   id                 integer4,
   area               float8,
   n_deelgebouwen     integer4,
   n_acn             integer4,
   n_percelen         integer4,
   n_huisnummers      integer4
) \p\g
grant select on gebouwen to public;\p\g

drop table deelgebouwen \p\g
create table deelgebouwen(
   id                 integer4,
   area               float8,
   g_akr_objectnummer char(17),
   gebouwen_id        integer4,
   n_buren            integer4,
   n_acn              integer4,
   n_huisnummers      integer4,
   cc_11tot15         char(1),
   classificatie      integer4
) \p\g
grant select on deelgebouwen to public;\p\g

drop table huisnummers\p\g
create table huisnummers(
   id                 integer4,
   ogroup             integer4,
   classif            integer4,
   otext              char(40),
   g_akr_objectnummer char(17),
```

```
   in_deelgebouw      integer4
) \p\g
grant select on huisnummers to public;\p\g

drop table acn\p\g
create table acn(
   id                 integer4,
   plaats             char(40),
   straatnaam         char(40),
   postcode           char(40),
   huisnummer         integer4,
   huistoev           char(5),
   adrtxt             char(10),
   aantal             integer4,
   g_akr_objectnummer char(17),
   in_deelgebouw      integer4,
   classificatie      integer4
) \p\g
grant select on acn to public;\p\g

drop table pap\p\g
create table pap(
   id                 integer4,
   plaats             char(40),
   straatnaam         char(40),
   postcode           char(40),
   huisnummer         char(5),
   huisletter         char(5),
   huistoev           char(5),
   aantal             integer4,
   g_akr_objectnummer char(17),
   in_deelgebouw      integer4
) \p\g
grant select on pap to public;\p\g
EOF
#
# Vul de tabellen met de dumps uit arc/info.
#
cat $TMPDIR/acn_in_deelgebouw.txt |\
 sed 's/,/     /g' |\
 $GEOHOME/ingres/copyrel $DB acn_in_deelgebouw \
    -n acn_id -n deelgebouwen_id

cat $TMPDIR/pap_in_deelgebouw.txt |\
 sed 's/,/     /g' |\
 $GEOHOME/ingres/copyrel $DB pap_in_deelgebouw \
    -n pap_id -n deelgebouwen_id

cat $TMPDIR/hnr_in_deelgebouw.txt |\
 sed 's/,/     /g' |\
 $GEOHOME/ingres/copyrel $DB hnr_in_deelgebouw \
    -n hnr_id -n deelgebouwen_id

cat $TMPDIR/gebouwen.txt |\
 sed 's/,/     /g' |\
 sed 's/$/     0       0       0       0/g' |\
 $GEOHOME/ingres/copyrel $DB gebouwen \
    -n id -n area -n n_deelgebouwen -n n_acn -n n_percelen -n n_huisnummers

cat $TMPDIR/deelgebouwen.txt |\
 sed 's/,/     /g' |\
 sed "s/\'//g" |\
 sed "s/$/     0       0       F       -1/g" |\
 $GEOHOME/ingres/copyrel $DB deelgebouwen \
    -n id -n area -q g_akr_objectnummer -n gebouwen_id -n n_buren -n n_acn \
    -n n_huisnummers -q cc_11tot15 -n classificatie

cat $TMPDIR/huisnummers.txt |\
 sed 's/,/     /g' |\
 sed "s/\'//g" |\
 sed "s/$/     -1/g" |\
 $GEOHOME/ingres/copyrel $DB huisnummers \
    -n id -n ogroup -n classif -q otext -q g_akr_objectnummer -n in_deelgebouw
```

```
cat $TMPDIR/pap.txt |\
 sed 's/,/     /g' |\
 sed "s/\'//g" |\
 sed "s/$/     -1/g" |\
 $GEOHOME/ingres/copyrel $DB pap \
    -n id -q plaats -q straatnaam -q postcode -q huisnummer -q huisletter \
    -q huistoev -n aantal -q g_akr_objectnummer -n in_deelgebouw

cat $TMPDIR/acn.txt |\
 sed 's/,/     /g' |\
 sed "s/\'//g" |\
 sed "s/$/     -1     -1/g" |\
 $GEOHOME/ingres/copyrel $DB acn \
    -n id -q plaats -q straatnaam -q postcode -n huisnummer \
    -q huistoev -n aantal -q g_akr_objectnummer -q adrtxt -n in_deelgebouw \
    -n classificatie
#
# Bouw indices op de verschillende tabellen.
#
sql $DB << EOF
   modify acn_in_deelgebouw to isam unique on acn_id with fillfactor = 100;\p\g
   modify pap_in_deelgebouw to isam unique on pap_id with fillfactor = 100;\p\g
   modify hnr_in_deelgebouw to isam unique on hnr_id with fillfactor = 100;\p\g
   modify gebouwen to isam unique on id with fillfactor = 100;\p\g
   modify huisnummers to isam unique on id with fillfactor = 100;\p\g
   modify deelgebouwen to isam unique on id with fillfactor = 100;\p\g
   modify pap to isam unique on id with fillfactor = 100;\p\g
   modify acn to isam on id with fillfactor = 100;\p\g
EOF
```

E5.1 Update ACN

```
#!/bin/sh

set -v
#
# Lees de globale variabelen in.
#
. globals.sh

sql $DB << EOF

set autocommit on \p\g
--
-- Zet het attribuut in_deelgebouw op -1 voor alle coordinaten.
--
update acn
set in_deelgebouw = -1
\p\g
--
-- Zet het attribuut in_deelgebouw voor alle acn coordinaten
-- die in een deelgebouw liggen.
--
update acn
from acn_in_deelgebouw
set in_deelgebouw = deelgebouwen_id
where acn.id = acn_in_deelgebouw.acn_id
\p\g

EOF
```

E5.2 Update deelgebouwen

```
#!/bin/sh
#
# Lees de globale variabelen in.
#
. globals.sh

sql $DB << EOF
```

```
set autocommit on \p\g
--
-- set n_huisnummers
--
drop table deelcount \p\g

create table deelcount as
select count(distinct id) as aantal, in_deelgebouw
from huisnummers
group by in_deelgebouw
\p\g

update deelgebouwen
from deelcount
set n_huisnummers = deelcount.aantal
where deelcount.in_deelgebouw = deelgebouwen.id
\p\g

drop table deelcount
\p\g
--
-- set n_acn
--
update deelgebouwen
set n_acn = 0;
\p\g

drop table deelcount \p\g

create table deelcount as
select count(distinct id) as aantal, in_deelgebouw
from acn
group by in_deelgebouw
\p\g

update deelgebouwen
from deelcount
set n_acn = deelcount.aantal
where deelcount.in_deelgebouw = deelgebouwen.id
\p\g

drop table deelcount
\p\g
--
-- Zet cc_11tot15 als dit deelgebouw op een perceel staat
-- met cultuurcode tussen 11 en 15.
--
update deelgebouwen
set cc_11tot15 = 'F';
\p\g

update deelgebouwen
from akr_objectadres
set cc_11tot15 = 'T'
where (akr_objectadres.g_akr_objectnummer = deelgebouwen.g_akr_objectnummer) and
   (akr_objectadres.soort_cult_beb in ('11','12','13','14','15'));
\p\g

EOF
```

E5.3 Update gebouwen

```
#!/bin/sh
#
# Lees de globale variabelen in.
#
. globals.sh

sql $DB << EOF
--
-- set n_percelen
--
```

```
create table perceelcount as
select count(distinct g_akr_objectnummer) as aantal, gebouwen_id
from deelgebouwen
group by gebouwen_id
\p\g

update gebouwen
from perceelcount
set n_percelen = perceelcount.aantal
where perceelcount.gebouwen_id = gebouwen.id
\p\g

drop table perceelcount
\p\g
--
-- set n_huisnummers
--
drop table deelcount \p\g

create table deelcount as
select sum(n_huisnummers) as aantal, gebouwen_id
from deelgebouwen
group by gebouwen_id
\p\g

update gebouwen
from deelcount
set n_huisnummers = deelcount.aantal
where deelcount.gebouwen_id = gebouwen.id
\p\g

drop table deelcount \p\g
--
-- set n_deelgebouwen
--
drop table deelcount \p\g

create table deelcount as
select count(distinct id) as aantal, gebouwen_id
from deelgebouwen
group by gebouwen_id
\p\g

update gebouwen
from deelcount
set n_deelgebouwen = deelcount.aantal
where deelcount.gebouwen_id = gebouwen.id
\p\g

drop table deelcount \p\g
--
-- set n_acn
--
drop table deelcount \p\g

create table deelcount as
select sum(n_acn) as aantal, gebouwen_id
from deelgebouwen
group by gebouwen_id
\p\g

update gebouwen
from deelcount
set n_acn = deelcount.aantal
where deelcount.gebouwen_id = gebouwen.id
\p\g

drop table deelcount \p\g

EOF
EOF
```

E6
Uitvoer

```
#!/bin/sh
#
# Exporteer de tabel acn gedeeltelijk naar de
# file $RESULTDIR/acn_classificatie.copy.gz
#
# Lees de globale variabelen in.
#
. globals.sh
#
# Gooi oude resultaatfiles weg.
#
rm -f $RESULTDIR/acn_classificatie.copy.gz
rm -f $TMPDIR/acn_classificatie.copy
#
# Kopieer de resultaten vanuit Ingres naar een file.
#
sql $DB << EOF
copy acn(
   id= c0tab,
   classif_text= c0tab,
   waardering= c0tab,
   classif_oud= c0nl
) into '$TMPDIR/acn_classificatie.copy'
\g
EOF
#
# Schrijf de classificaties naar arc.
#
cat $TMPDIR/acn_classificatie.copy |\
   sed 's/ //g' |\
   sed 's/     /,/g' > $TMPDIR/acn_classificatie.txt

(cd $ARCDIR/arc ; arc ) << EOF
tables
select acn_classificatie
erase acn_classificatie
y
define acn_classificatie
acn-id 4 5 B
classif_text 1 1 C
waardering 4 4 I
classif_oud 4 4 I

select acn_classificatie
add from $TMPDIR/acn_classificatie.txt
quit
quit
EOF
#
# Join de niet geografische attributen aan de geografische.
#
(cd $ARCDIR/arc ; arc ) << EOF
dropitem acn.pat acn.pat classificatie
indexitem acn_classificatie acn-id
indexitem acn.pat acn-id
joinitem acn.pat acn_classificatie acn.pat acn-id
tables
select acn.pat
alter acn-id,acn-id,12,,,
quit
quit
EOF

rm -f $TMPDIR/acn_classificatie.copy
```

Bijlage F de classificatieregels.

Vanaf hier zijn de deelprogramma's anders dan die in het kadasterproject.

overzicht

```
#!/bin/sh

do_classificatie.sh

propagate_to_acn.sh

do_classificatie_acn.sh

export_acn_classificatie.sh
export_deelgebouwen_classificatie.sh
```

F1 Deelgebouw-classificatie

```
#!/bin/sh
#
# Classificeer alle deelgebouwen.
#
# Lees de globale variabelen in.
#
. globals.sh
#
# Doe de classif_text.
#
sql $DB << EOF

set autocommit on \p\g
--
-- Initieel krijgen alle gebouwen classif_text '-'.
-- en een waardering die 0 is
--
update deelgebouwen
set classif_text = '-',
waardering = 0
\p\g
--
-- Geef alle gebouwen die op een perceel met appartementen staan
-- classif_text 10. (Appartement).
--
update deelgebouwen
from akr_object_a
set classif_text = 'A',
waardering = 1
where
       classif_text = '-'
   and akr_object_a.g_akr_objectnummer = deelgebouwen.g_akr_objectnummer
\p\g
--
-- Geef alle deelgebouwen met een perceel met cultuurcode '--'
-- ,dit zijn deelgebouwen met cultcode anders dan 00, 11-15,
-- classif_text '#' (Niet Wonen).
--
update deelgebouwen
from percelen
set classif_text = '#',
waardering=0
where
       deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and deelgebouwen.classif_text = '-'
   and percelen.cultuurcode = '--'
\p\g
--
-- Eerst doen we de ideale cases.
--
-- Vrijstaand
```

```
-- aanpassing: nparcels en nhuisnummer zijn 1 van gebouwen,
-- dit is alleen voor de voledigeis, maakt eigenlijk iets uit.
--
update deelgebouwen
from gebouwen
set classif_text = 'V',
waardering = 1
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_buren = 0
   and deelgebouwen.n_acn = 1
   and deelgebouwen.n_huisnummers = 1
   and gebouwen.n_deelgebouwen = 1
   and gebouwen.n_percelen = 1
   and gebouwen.n_huisnummers = 1
\p\g

-- toegevoegd de eis dat er een acn in het deelgebouw ligt.
-- en er twee huisnummers in het gebouw liggen.

update deelgebouwen
from gebouwen
set classif_text = 'T',
waardering = 1
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and cc_11tot15 = 'T'
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_buren = 1
   and deelgebouwen.n_acn = 1
   and deelgebouwen.n_huisnummers = 1
   and gebouwen.n_deelgebouwen = 2
   and gebouwen.n_percelen = 2
   and gebouwen.n_acn < 3
   and gebouwen.n_huisnummers = 2
   \p\g

-- toegevoegd  n_huisnummers in gebouw > 0  en acn aantal weg, zit al in n_acn

update deelgebouwen
from gebouwen
set classif_text = 'E',
waardering = 1
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_buren = 1
   and deelgebouwen.n_acn = 1
   and deelgebouwen.n_huisnummers = 1
   and gebouwen.n_deelgebouwen > 2
   and gebouwen.n_percelen > 2
   and gebouwen.n_huisnummers >0
\p\g

-- toegevoegd n_acn =1

update deelgebouwen
from gebouwen
set classif_text = 'M',
waardering = 1
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_acn = 1
   and deelgebouwen.n_buren > 1
   and deelgebouwen.n_huisnummers < 2
   and gebouwen.n_deelgebouwen > 2
   and gebouwen.n_percelen > 2
   and gebouwen.n_huisnummers > 0
\p\g
```

```
--
-- Classificeer schuren.
--
update deelgebouwen
from percelen
set classif_text = 'S',
waardering = 0
where
   deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_acn = 0
   and percelen.n_acn < percelen.n_gebouwen
   and deelgebouwen.n_buren = 0
\p\g
--
-- Nu draaien we de classif_texts waarbij de acn buiten het
-- gebouw en in het perceel ligt. Met 1 gebouw op het perceel.
--
-- n_huisnummers is veranderd van 0 naar 1,
-- en cultcode nog steeds true voor alle waarderingen 2
--
update deelgebouwen
from percelen
set classif_text = 'V',
waardering = 2
where
       deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.n_huisnummers = 1
   and deelgebouwen.n_acn = 0
   and deelgebouwen.n_buren = 0
   and percelen.n_acn = 1
   and percelen.n_gebouwen = 1
\p\g

-- symbolisch ook n_percelen = 2
update deelgebouwen
from percelen,gebouwen
set classif_text = 'T',
waardering = 2
where
       deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.n_acn = 0
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_huisnummers = 1
   and deelgebouwen.n_buren = 1
   and percelen.n_gebouwen = 1
   and percelen.n_acn = 1
   and gebouwen.n_deelgebouwen = 2
   and gebouwen.n_percelen = 2
\p\g

-- n_huisnummers van 0 naaar 1 veranderd

update deelgebouwen
from percelen, gebouwen
set classif_text = 'E',
waardering = 2
where
       deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_buren = 1
   and deelgebouwen.n_huisnummers = 1
   and deelgebouwen.n_acn = 0
   and gebouwen.n_deelgebouwen > 2
   and gebouwen.n_percelen > 2
   and percelen.n_gebouwen = 1
   and percelen.n_acn = 1
\p\g
```

```
update deelgebouwen
from gebouwen,percelen
set classif_text = 'M',
waardering = 2
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_acn = 0
   and deelgebouwen.n_buren > 1
   and gebouwen.n_deelgebouwen > 2
   and gebouwen.n_percelen > 2
   and percelen.n_acn = 1
   and percelen.n_gebouwen = 1

\p\g
--
-- Acn coordinaten die niet in een gebouw liggen, maar wel in een
-- perceel met precies 1 gebouw daarop kunnen we aan het betreffende
-- gebouw toekennen.
--
create table percelen_met_1_gebouw
as
   select g_akr_objectnummer
   from deelgebouwen
   group by g_akr_objectnummer
   having count(id) = 1
\p\g

update acn
from deelgebouwen,percelen_met_1_gebouw
set in_deelgebouw = deelgebouwen.id
where
       (acn.g_akr_objectnummer = deelgebouwen.g_akr_objectnummer)
   and (percelen_met_1_gebouw.g_akr_objectnummer = deelgebouwen.g_akr_objectnummer)
   and (acn.in_deelgebouw = '-')
\p\g

drop table percelen_met_1_gebouw \p\g

-- nog een stukje over n_acn toegevoegd.

create table deelcount as
select count(distinct id) as aantal, in_deelgebouw
from acn
group by in_deelgebouw
\p\g

update deelgebouwen
from deelcount
set n_acn = deelcount.aantal
where deelcount.in_deelgebouw = deelgebouwen.id
and n_acn < deelcount.aantal
\p\g

drop table deelcount
\p\g
--
-- Draai nu de rest van de classif_texts.
--
-- eigenlijk de idealecases nog een keer, nu nog niet gebeurd,
-- want ACN update van vorige ddorloop zit er nog in ,
-- dus dat zou toch 00 opleveren.
--
update deelgebouwen
from acn
set classif_text = 'A',
waardering = 3
where
       acn.in_deelgebouw = deelgebouwen.id
   and (deelgebouwen.classif_text = '-')
   and (deelgebouwen.cc_11tot15 = 'T')
```

```
   and (acn.aantal > 7)
\p\g

update deelgebouwen
from gebouwen
set classif_text = 'V',
waardering = 3
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.n_acn = 1
   and deelgebouwen.n_buren = 0
   and gebouwen.n_deelgebouwen = 1
   and gebouwen.n_percelen = 1
\p\g

-- n_acn in gebouwen <3 toegevoegd.

update deelgebouwen
from gebouwen
set classif_text = 'T',
waardering = 3
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.n_buren = 1
   and deelgebouwen.n_acn = 1
   and gebouwen.n_deelgebouwen = 2
   and gebouwen.n_percelen = 2
   and gebouwen.n_acn < 3
\p\g

--  acn_aantal < 2 weg, zit al in deelgebouw.n_acn.
update deelgebouwen
from gebouwen,acn
set classif_text = 'E',
waardering = 3
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and acn.in_deelgebouw = deelgebouwen.id
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.n_buren = 1
   and deelgebouwen.n_acn < 2
   and gebouwen.n_deelgebouwen > 2
   and gebouwen.n_percelen > 2
\p\g

update deelgebouwen
from gebouwen
set classif_text = 'M',
waardering = 3
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.cc_11tot15 = 'T'
   and deelgebouwen.n_buren > 1
   and deelgebouwen.n_acn < 2
   and gebouwen.n_deelgebouwen > 2
   and gebouwen.n_percelen > 2
\p\g

update deelgebouwen
from percelen
set classif_text = 'V',
waardering = 8
where
       deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and deelgebouwen.classif_text = '-'
   and percelen.cultuurcode = '00'
   and deelgebouwen.n_huisnummers = 1
   and deelgebouwen.n_acn = 1
```

```
   and deelgebouwen.n_buren = 0
   and deelgebouwen.area < 120000000
\p\g

-- deelgeb.n_acn=1 toegevoegd
update deelgebouwen
from percelen,gebouwen
set classif_text = 'T',
waardering = 8
where
       deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.classif_text = '-'
   and percelen.cultuurcode = '00'
   and deelgebouwen.n_huisnummers = 1
   and deelgebouwen.n_buren = 1
   and deelgebouwen.n_acn = 1
   and deelgebouwen.area < 120000000
   and gebouwen.n_acn = 2
   and gebouwen.n_deelgebouwen = 2
   and gebouwen.n_percelen = 2
\p\g

-- cc wegehaald, zit a in cultcode = 00
-- geb.huisnummers toegevoegd, en opervlakte eis toegevoegd.

update deelgebouwen
from gebouwen,percelen
set classif_text = 'E',
waardering = 8
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and percelen.cultuurcode = '00'
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_buren = 1
   and deelgebouwen.n_huisnummers = 1
   and deelgebouwen.n_acn = 1
   and deelgebouwen.area between 40000000 and 100000000
   and gebouwen.n_deelgebouwen > 2
   and gebouwen.n_percelen > 2
   and gebouwen.n_huisnummers > 0
\p\g

update deelgebouwen
from gebouwen,percelen
set classif_text = 'M',
waardering = 8
where
       (deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer)
   and gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.classif_text = '-'
   and percelen.cultuurcode = '00'
   and deelgebouwen.n_buren > 1
   and deelgebouwen.n_acn = 1
   and deelgebouwen.n_huisnummers < 2
   and deelgebouwen.area between 40000000 and 100000000
   and gebouwen.n_deelgebouwen > 2
   and gebouwen.n_percelen > 2
\p\g

update deelgebouwen
set classif_text = 'V',
waardering = 4
where
      deelgebouwen.classif_text = '-'
   and deelgebouwen.n_acn = 1
   and deelgebouwen.n_buren = 0
\p\g

update deelgebouwen
set classif_text = 'T',
waardering = 4
where
```

```
   deelgebouwen.n_acn = 2
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_buren = 0
\p\g

update deelgebouwen
set classif_text = 'M',
waardering = 4
where
       classif_text = '-'
   and n_buren > 1
   and n_acn = 1
\p\g

-- heel erg aangepast vanwege copy acn naar deelgebouwen.

update deelgebouwen
set classif_text = 'V',
waardering = 5
where
   classif_text = '-'
   and n_huisnummers = 1
   and n_acn = 1
   and n_buren = 0
\p\g

update deelgebouwen
from gebouwen
set classif_text = 'T',
waardering = 5
where
       gebouwen.id = deelgebouwen.gebouwen_id
   and deelgebouwen.classif_text = '-'
   and gebouwen.n_deelgebouwen = 2
   and gebouwen.n_huisnummers = 2
   and gebouwen.n_acn = 2
\p\g

update deelgebouwen
set classif_text = 'T',
waardering = 6
where
       classif_text = '-'
   and n_huisnummers = 2
   and n_acn = 2
   and n_buren = 0
\p\g

update deelgebouwen
from percelen
set classif_text = 'T',
waardering = 7
where
   deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and deelgebouwen.classif_text = '-'
   and deelgebouwen.n_huisnummers = 2
   and deelgebouwen.n_buren = 0
   and percelen.n_gebouwen = 1
   and percelen.n_acn = 2
\p\g

update deelgebouwen
set classif_text = 'A',
waardering = 5
where
   classif_text = '-'
   and n_acn > 2
   and (area/n_acn) < 30000000
\p\g

-- in de volgende zitten eindhuizen en middenwoningen samen.
update deelgebouwen
set classif_text = 'M',
waardering = 5
```

```
where
       classif_text = '-'
   and n_acn > 2
   and ( area / n_acn ) > 50000000
\p\g

update deelgebouwen
set classif_text = 'M',
waardering = 6
where
       classif_text = '-'
   and n_acn > 2
   and ( area / n_acn ) > 40000000
\p\g

update deelgebouwen
set classif_text = 'M',
waardering = 7
where
       classif_text = '-'
   and n_acn > 2
   and ( area / n_acn ) > 30000000
\p\g

update deelgebouwen
from percelen
set classif_text = 'T',
waardering = 9
where
       deelgebouwen.classif_text = '-'
   and deelgebouwen.n_huisnummers = 2
   and deelgebouwen.n_buren = 1
   and deelgebouwen.g_akr_objectnummer = percelen.g_akr_objectnummer
   and percelen.n_acn < 3
\p\g

update deelgebouwen
set classif_text = 'M',
waardering = 9
where
       classif_text = '-'
   and n_acn > 1
   and n_buren  > 1
\p\g
--
-- Maak een overzichtje van de classif_texts.
--
select classif_text, waardering, count(distinct id)
from deelgebouwen
group by classif_text, waardering
\p\g
```

F2 Voortplanting

```
EOF
#!/bin/sh
#
# Author: Wilko Quak (quak@geo.tudelft.nl)
# Edited by Maureen Rengelink
#
# Propageer de classificaties die in de tabel
# deelgebouwen hangen door naar de tabel acn.
#
# Classificaties:
#:'_' Bijbehorend deelgebouw niet geclassificeerd
# '*' : Geen bijbehorend deelgebouw gevonden
# A: Appartement
# V: Vrijstaand.
# T : Twee onder een kap.
# E : Eindwoning.
# M : Middenwoning.
```

```
# S : Geen woning.
# en bijbehorende waardering.
#
# Lees de globale variabelen in.
#
. globals.sh

sql $DB << EOF

set autocommit on \p\g
--
-- Reset all.
--
update acn
set classif_text = '*', waardering = 0
\p\g
--
-- Propageer classif_text van ieder deelgebouw
-- naar bijbehorende coordinaat.
--
update acn
from deelgebouwen
set classif_text = deelgebouwen.classif_text,
waardering=deelgebouwen.waardering
where acn.in_deelgebouw = deelgebouwen.id
\p\g
--
-- Propageer acn die in een perceel staan met maar 1 deelgebouw.
--
drop table classif_text_per_perceel \p\g
create table classif_text_per_perceel
as
   select max(classif_text) as classif_text,percelen.g_akr_objectnummer
   from deelgebouwen,gebouwen,percelen
   where
          deelgebouwen.gebouwen_id = gebouwen.id
      and percelen.g_akr_objectnummer = deelgebouwen.g_akr_objectnummer
      and percelen.n_acn = 1
   group by percelen.g_akr_objectnummer
   having count(distinct gebouwen_id) = 1
\p\g

update acn
from classif_text_per_perceel
set classif_text = classif_text_per_perceel.classif_text,
waardering = 10
where
   classif_text_per_perceel.g_akr_objectnummer = acn.g_akr_objectnummer
   and in_deelgebouw = -1
\p\g

drop table classif_text_per_perceel \p\g
--
-- Maak een overzichtje van de classif_texts.
--
select classif_text,count(distinct id)
from acn
group by classif_text
\p\g

EOF
```

F3 ACN classificatie

```
#!/bin/sh
#
# Classificeer alle deelgebouwen.
#
# Lees de globale variabelen in.
#
. globals.sh
```

```
sql $DB << EOF

set autocommit on \p\g
--
-- Zet classif_text voor acn op een perceel zonder woonbestemming op #.
--
update acn
from percelen
set classif_text = '#',
waardering = 0
where
       acn.g_akr_objectnummer = percelen.g_akr_objectnummer
   and percelen.cultuurcode = '--'
\p\g
--
-- Alle acn die op een appertementsperceel liggen worden appartement.
--A 11
update acn
from akr_object_a
set classif_text = 'A',
waardering = 11
where
       classif_text in ('-', '*')
   and akr_object_a.g_akr_objectnummer = acn.g_akr_objectnummer
\p\g
--
-- Zoek alle gebouwen waar een rijtjes huis inzit.
-- Ongeclassificeerde huizen in deze gebouwen zijn ook
-- rijtjeshuizen.
--
drop table rijtjesgebouwen
\p\g
create table rijtjesgebouwen
as
   select gebouwen.id,count(acn.id)
   from acn,gebouwen,deelgebouwen
   where
          acn.classif_text = 'M'
      and deelgebouwen.gebouwen_id = gebouwen.id
      and acn.in_deelgebouw = deelgebouwen.id
   group by gebouwen.id
\p\g

update acn
from rijtjesgebouwen,gebouwen,deelgebouwen
set classif_text = 'E',
waardering = 15
where
       rijtjesgebouwen.id = gebouwen.id
   and deelgebouwen.gebouwen_id = gebouwen.id
   and deelgebouwen.id = acn.in_deelgebouw
   and acn.classif_text = '-'
\p\g
drop table rijtjesgebouwen \g
--
-- Maak een tabel met voor ieder gebouw het acn met maximale huisnummer.
-- Als deze acn als rijtjeshuis is geclassificeerd dan maken we daar
-- een hoekhuis van.
--
drop table rijtjes_nummer \g
create table rijtjes_nummer
as
   select gebouwen.id , min(adrtxt) as laagnummer, max(adrtxt) as hoognummer
   from acn,gebouwen,deelgebouwen
   where
          deelgebouwen.gebouwen_id = gebouwen.id
      and acn.in_deelgebouw = deelgebouwen.id
   group by gebouwen.id
\p\g

update acn
from rijtjes_min,gebouwen,deelgebouwen
set classif_text = 'E',
```

Tikfout: stond eerst '#', wat de niet woningen

```
waardering = 16
where
       rijtjes_nummer.laagnummer = acn.adrtxt
   and rijtjes_nummer.id = gebouwen.id
   and deelgebouwen.gebouwen_id = gebouwen.id
   and acn.classif_text = 'M'
   and deelgebouwen.id = acn.in_deelgebouw
\p\g

update acn
from rijtjes_nummer,gebouwen,deelgebouwen
set classif_text = 'E',
waardering =17
where
       rijtjes_nummer.hoognummer = acn.adrtxt
   and rijtjes_nummer.id = gebouwen.id
   and deelgebouwen.gebouwen_id = gebouwen.id
   and acn.classif_text = 'M'
   and deelgebouwen.id = acn.in_deelgebouw
\p\g
drop table rijtjes_nummer \g
--
-- Maak een overzichtje van de classif_texts.
--
select classif_text,count(distinct id)
from acn
group by classif_text
\p\g
```