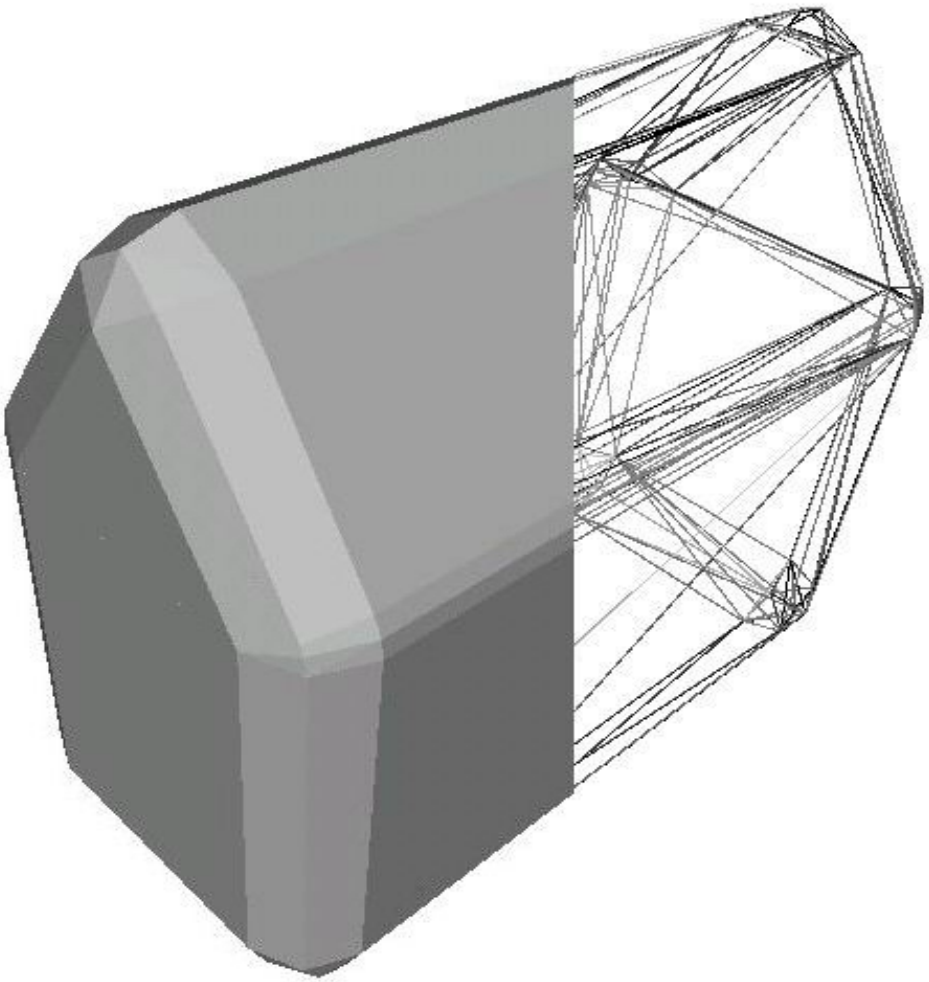
Driedimensionale buffering op basis van Tetraëder Netwerken

opslag en analyse in een 3D-GIS



Jeroen de Vries
TU Delft, faculteit CiTG,
afdeling geodesie, sectie GISt
Delft, 2001

Driedimensionale buffering op basis van Tetraëder Netwerken

Jeroen de Vries

Voorwoord

Dit rapport is een verslag van het onderzoek dat ik in het kader van mijn afstuderen heb gedaan. Het beschrijft hoe een driedimensionale GIS opgezet kan worden, waarbij de elementen opslag, analyse en visualisatie geïntegreerd zijn. De grootste aandacht kregen de opslag en de analyse. 3D-visualisatie is natuurlijk een leuk onderwerp, maar daar is al ruim onderzoek aan besteed, terwijl de andere elementen minstens zo belangrijk zijn.

Ik ben aan veel mensen dank verschuldigd, maar in het bijzonder aan ir. E. Verbree, prof. dr. ir. P. van Oosterom en drs. W. Quak .

Samenvatting

De laatste jaren wordt steeds meer onderzoek gedaan naar de toepassing van driedimensionale GI-systemen, kortweg 3D-GIS. Deze onderzoeken zijn echter vaak meer gericht op de visualisatie van driedimensionale gegevens dan op analyses op in drie dimensies lokaliseerbare (geografische) fenomenen. Bovendien wordt bij onderzoeken naar 3D-GIStechnieken nauwelijks het nut van dergelijke technieken beargumenteerd. Traditionele GIS-analyses, zoals bijvoorbeeld netwerkanalyses, blijken vaak goed toepasbaar in een driedimensionale omgeving, maar er zijn ook situaties denkbaar waarbij driedimensionale analyses wel gewenst zijn. Zo kan het bijvoorbeeld nodig zijn om berekeningen met volumes uit te voeren, of de bepaling van directionele relaties als 'boven', 'onder', 'schuin rechts achter', etc. Dergelijke analyses zullen voornamelijk een meerwaarde hebben bij grootschalige toepassingen, omdat hierbij de hoogtecomponent in verhouding tot de planimetrische componenten steeds belangrijker wordt. Wanneer een (3D-)GIS opgezet wordt, moet gekozen worden hoe de gegevens opgeslagen, bewerkt en gepresenteerd moeten gaan worden.

Dit afstudeeronderzoek is gericht op het de vraag: '*Hoe kan een 3D-GIS voor grootschalige toepassingen opgezet worden?'*. De nadruk ligt hierbij niet op visualisatie, maar op de opslag van de gegevens en op de analysegereedschappen in drie dimensies. Visualisatie is wel van invloed op met name de opslag van driedimensionale gegevens.

Er bestaan verschillende technieken om objecten geometrisch te modelleren. Veel van deze technieken zijn bedoeld voor CAD-toepassingen en voor visualisatie. Voor toepassing in een 3D-GIS zijn de meest geschikte objectrepresentaties:

- Boundary Representation (B-Rep),
- Cell Decomposition,
- Spatial Occupancy Enumeration (SOE).

De keuze uit bovenstaande opsomming is afhankelijk van het doel van de representatie. Voor een algemene driedimensionale GIS, waarin zowel bewerking als visualisatie van de driedimensionale gegevens wenselijk is, is een bijzondere vorm van Cell Decomposition, namelijk het Tetrahedral Network (TEN) het meest geschikt. Een TEN is een datastructuur waarbij de objecten opgebouwd worden uit niet-overlappende tetraëders en is de driedimensionale tegenhanger van Triangular Irregular Networks (TIN). De TEN heeft de voordelen, dat vrijwel elke vorm te benaderen is, dat een snelle visualisatie mogelijk is, en dat operaties op en met objecten gemodelleerd in een TEN gemakkelijk zijn. De topologische structuur van een TEN is sterker dan bij de B-Rep, SOE en Cell Decomposition met willekeurig gevormde cellen, omdat de cellen eenvormig zijn en de relaties tussen naburige cellen en hun elementen, en tussen de elementen van een individuele cel vastliggen, evenals het onderscheid tussen interieur, grens en exterieur.

Een GIS-datastructuur moet naast geometrie en topologie ook thematische informatie vastleggen. Een driedimensionale GIS stelt bovendien de eis, dat de gegevensstructuur de opslag van een verschillend aantal abstractieniveaus (*Levels-of-Detail, LoD*) mogelijk maakt. Een objectgeoriënteerde structuur voldoet aan die eisen. Een voorbeeld is de opslag van een gebouw ten behoeve van een systeem waarmee dat gebouw beheerd kan worden. Een gebouw heeft bepaalde eigenschappen, en bestaat uit een aantal verdiepingen met eigen eigenschappen. Deze verdiepingen bestaan weer uit ruimtes, bijvoorbeeld kamers, met eigen eigenschappen. Deze ruimtes kunnen vervolgens geometrisch gemodelleerd worden, waardoor een geïntegreerd datamodel ontstaat.

In dit rapport is de buffer als analysegereedschap nader uitgewerkt, omdat vertaling hiervan naar drie dimensies een aparte benadering behoeft. Een buffer is afstandsoperator; met een buffer kan bepaald worden welke objecten zich binnen een bepaalde afstand van een gebufferd object bevinden, of welke juist niet. Een buffer in 2D is daarom een vlak dat gevormd wordt door een band om een object te leggen. Bij de hoekpunten is de buffer, omdat de afstand tussen het object en de buffer overal gelijk moet zijn, in theorie een kromme. Het verwerken en opslaan van krommen is echter complex en, afhankelijk van de gekozen datastructuur, onmogelijk. Een oplossing is om de zijdes van de buffer door te trekken en te laten snijden, welk snijpunt de nieuwe hoekpunten van de buffer vormen. De buffer wordt hierdoor, althans bij convexe objecten, gelijkvormig aan het oorspronkelijke object, maar heeft andere afmetingen. Bij de hoekpunten ontstaat echter een groot afstandsverschil tussen de gevormde en de ideale buffer. Een andere oplossing is het benaderen van de kromme. Deze oplossing is algoritmisch complexer, maar geeft een kleiner afstandsverschil. Bij beide oplossingen ontstaat een verschil tussen de ideale buffer en de gegenereerde buffer.

Het genereren van een driedimensionale buffer verdient grotere aandacht omdat het afstandsverschil tussen een ideale en een gegenereerde buffer, dus het verschil tussen twee volumes, in drie dimensies optreedt. Een buffer in drie dimensies wordt zowel bij de vertices als bij de randen afgerond, en afrondingsfouten ontstaan daardoor niet alleen bij de vertices, maar ook bij de randen. Hierdoor wordt de totale fout, dus het aantal objecten dat ten onrechte wel, of juist niet, op basis van de buffer geselecteerd worden, relevanter. Als verwerking en opslag van gekromde oppervlakken niet wenselijk is, dan is het beste alternatief voor het genereren van een driedimensionale buffer, het genereren van een afgeronde buffer.

Het genereren van een 3D-Buffer op basis van een TEN is zowel voordelig bij de constructie zelf, als bij de verdere verwerking, omdat de buitenwereld van een object in een TEN bekend is. Hierdoor kan de grens van het object bepaald worden op basis waarvan een buffer gemodelleerd wordt. Deze wordt vervolgens weer opgeslagen in een TEN. Het grootste voordeel van de TEN doet zich echter voor bij de verwerking van de buffer. Omdat de buitenwereld van de buffer bekend is, kan bepaald worden welke andere objecten los staan van de buffer. Van de overige objecten wordt de topologische relatie tussen het object en de buffer bepaald. Als het object zich gedeeltelijk in de bufferzone bevindt kan een eventuele clipoperatie beperkt blijven tot alleen de cellen van het object die zich in het bufferlichaam bevinden.

De theoretische opbouw van een driedimensionale buffer is getest door een bufferalgoritme te programmeren met behulp van CGAL. CGAL staat voor *Computational Geometry Algorithms Library*, en is een voor onderzoek- en onderwijsdoeleinden vrij verkrijgbare programmabibliotheek, waarin driedimensionale datastructuren als de TEN een belangrijke plaats innemen. Het programma laat zien dat de theorie klopt en dat een driedimensionale afgeronde buffer met een simpele code te genereren is voor een object dat gemodelleerd is in een TEN. Hier bewijst het nut van een sterke topologie zich. Voor het genereren van een buffer wordt de afstand tot de grens van een object bepaald, en de grens van een object is een topologische eigenschap. In de code wordt dan ook van de topologie van een TEN gebruik gemaakt om randen op de grens en de bijbehorende grensvlakken van een object te vinden.

De term ‘Geografisch Informatie Systeem’ is een beperkte term die de lading, vooral bijgrootschalige toepassingen, niet volledig dekt. Een 3D-GIS dat gebruik maakt van een objecthiërarchie en de Tetrahedral Network om gegevens op te slaan, en dat driedimensionale analysegereedschappen biedt, zoals de 3D-Buffer, is ook geschikt voor een grotere schaal dan de geografische. In plaats van een gebouw, kan het menselijk lichaam bijvoorbeeld onderwerp van ruimtelijke analyse zijn, waarbij een lichaamsdeel uit weefsel bestaat dat op zijn beurt uit cellen bestaat, die vervolgens in een TEN gemodelleerd worden. Met een 3D-buffer kunnen dan bijvoorbeeld cellen geselecteerd worden die door een tumor beïnvloed worden. De toepassing is anders, maar de techniek hetzelfde, en in plaats van de term ‘grootschalige 3D-GIS is wellicht de term ‘Ruimtelijk Informatiesysteem’ beter.

Abstract

Recently, more and more research is being done into the application of three-dimensional GIS, or 3D-GIS for short. In general, the topic of those researches is just visualization of three dimensional data rather than analysis of three-dimensional phenomena's. Even though research into 3D-GIS is emerging, the reason why and when a three dimensional GIS should be used, is hardly argumented. Conventional GIS tools, like network analysis, can be used in a three dimensional environment rather easily, but some spatial problems require a real three dimensional approach, like the processing of volumetric bodies, for instance, or determining directional relationships like 'above', 'below', etcetera. The importance of such three dimensional tools will grow as the scale of the application grows. In large-scale applications, the relationship between height coordinates and planimetric coordinates will be more relevant. Building a (three dimensional) GIS, one has to decide how to store, retrieve, process and present the spatial data.

The main question in this research is '*How should a 3D-GIS for large-scale applications be set up?*. Emphasis lies on the storage of three-dimensional data and on three-dimensional analysis. Both elements are, however, influenced by visualization.

There are a number of geometric modeling techniques, most of which are developed to allow a fast visualization of objects, or to be used in a CAD environment. From this wide range of techniques, the following three are suitable to model three-dimensional GIS objects:

- Boundary Representation (B-Rep),
- Cell Decomposition,
- Spatial Occupancy Enumeration (SOE),

The right choice depends on the purpose of the representation. A general three-dimensional GIS demands fast object operations, but also a fast visualization of objects, especially if the three-dimensional world is being presented in a real-time Virtual Environment. The Tetrahedral Network (TEN) is a structure that complies with both demands. A Tetrahedral Network is a specific type of Cell Decomposition. The object is, however, represented by subdividing it into just irregular non-intersecting tetrahedrons. A TEN allows a precise estimation of free forms, a fast visualization, and a fast object manipulation. Because the cells are uniform, the connectivity between tetrahedrons and its elements, and between elements of an individual tetrahedron is firmly described. The topological structure of a TEN is stronger than that of the B-Rep, SOE and Cell Decomposition with arbitrary shaped cells.

A GIS data structure must be able to not just store geometry and topology, but thematic information as well. In addition, a three dimensional GIS must be able to represent a number of different levels of detail. An object-oriented structure allows both. A 3D GIS could, for example, be used to monitor a building. A building has some properties, and consists of several floors. A floor on its own has properties and consists of rooms. A room has properties and can be geometrical represented. The combination of thematic, topological and geometric information leads to an integrated data structure.

The buffer zone is described in this report extensively, because the conversion from a two dimensional buffer zone to a three dimensional buffer zone involves some specific problems. A buffer zone is a distance operation; with a buffer zone it's possible to locate features positioned within a given range to an object. A two dimensional buffer is an area, constructed by placing a corridor along an object. The distance is valid for the whole buffer, so at the corners the buffer theoretically consists of curves. The processing and storage of curves is complex and, depending on the data structure, often impossible. One solution to this is to extend the edges of a buffer and to intersect these extended lines. The intersections define new points of the buffer. A buffer constructed this way is, applied to concave objects, a larger version of the buffered object. An error occurs at the corners, because of the difference between the generated and the ideal buffer zone. A different solution is to estimate the curve by inserting extra points. This solution is more complex, but the difference between the estimated buffer and the ideal buffer is smaller. Both solutions generate an error. The construction of a three dimensional buffer zone deserves extra

attention. Because a 3D-buffer is a volume, an error not only occurs at the corners, but along the edges as well. So the error has a larger impact upon the quality of the buffer. When storage or processing of curved surfaces is impossible or not wanted, the best solution is to estimate a rounded buffer zone by inserting extra points and extra edges.

The construction of a 3D-buffer based on a TEN pays of both at the construction itself as well at the further processing of the buffer, because the exterior of an object, modeled in a TEN, is known. This means that the boundary of an object can be found based upon which the buffer is modeled. This buffer is stored again in a new TEN. Because the exterior of the buffer is known, all objects can be found which are not disjoint from the buffer, i.e. all objects which are completely or partly within the buffer. When needed, a clipping algorithm only has to be executed for object cells, which are partly within the buffer.

The theoretical construction of a three dimensional buffer zone is tested by programming the buffering algorithm wit use of CGAL. CGAL stands for *Computational Geometry Algorithms Library* and is a freely available program library. Three dimensional data structures form an important part of CGAL. The program shows that the theory is valid and that it's possible to generate a three-dimensional buffer zone based on a TEN, with a simple code. The code makes use of the topology of the TEN to find the boundary edges and the boundary facets.

The indication 'Geographic Information Systems' is very limiting, especially when a GIS is used in large-scale applications. Application of a 3D-GIS that stores its data in an object hierarchical structure in combination with a TEN, and offers three-dimensional GIS tools, like a 3D-bufferzone, is not confined to just geographical environments. Such 3D-GIS could, for instance, just as easy be used as a medical tool, wherein the subject is not a building, but a human body, which consists of organs, which consist of tissue, etc. The influence of a tumor on other cells or other tissues could be determined using a 3D-buffer. The environment is different, but the techniques stay the same, and instead of GIS, the name 'Spatial Information System' should be preferable.

Inhoud

VOORWOORD ..... **I**

SAMENVATTING ..... **III**

ABSTRACT ..... **V**

1. INLEIDING **1**

2. DEFINITIES **3**

2.1 EEN DEFINITIE VAN 3D-GIS 3
2.2 GIS-ANALYSES: EEN INVENTARISATIE 4
directionele relaties *5*
transformaties *5*
objecteigenschappen *5*
metriek *5*
topologie *6*
attribuutoperaties *6*
laagoperaties *7*
zichtbaarheidanalyse *7*
visualisatie *7*
2.3 GEOMETRISCHE MODELLEN 8
vectorgebaseerde representaties *8*
getesseleerde representaties *9*
analytische representaties *10*
hybride representaties *12*
objectrepresentaties en 3D-GIS *12*

3. DE OPSLAG IN EEN 3D-GIS TOEGEPAST OP EEN CASE **15**

3.1 BESCHRIJVING VAN DE CASE 15
3.2 EISEN AAN DE OPSLAG IN EEN 3D-GIS 15
3.3 UITWERKING VAN DE CASE IN 3D 16
thematische structuur: objecthiërarchie *16*
geometrische structuur: het Tetrahedral Network *17*
geïntegreerd datamodel *19*

4. TETRAHEDRAL NETWORKS IN CGAL **21**

4.1 CGAL: COMPUTATIONAL GEOMETRY ALGORITHMS LIBRARY 21
4.2 TEN DATASTRUCTUUR IN CGAL 21
4.3 VISUALISATIE VAN DE DATASTRUCTUUR 23

5. EEN DRIEDIMENSIONAAL ANALYSEGEREEDSCHAP: BUFFERING IN 3D **25**

5.1 BUFFERS IN TWEE DIMENSIES 25
soorten buffers *26*
verbeterde buffers: vier varianten *27*
5.2 BUFFERS IN DRIE DIMENSIES 32
de voordelen van een buffer als Tetrahedral Network *32*
selectie van tetraëders op basis van topologie *33*
de opbouw van een afgeronde 3D buffer *35*
5.3 HET PROGRAMMA '3DBUFFER' 36
stap 1: vindt de boundary edges en de boundary facets *38*
stap 2: bepaal de verplaatsingsvectoren *39*
stap 3: verplaatsen van de edges *40*
stap 4: voeg tussenpunten toe *40*
resultaat *40*

5.4 DRIEDIMENSIONALE BUFFERING VOOR CONCAVE OBJECTEN 42
buffering van concave objecten in drie dimensies *44*

6. CONCLUSIES EN AANBEVELINGEN 45

6.1 CONCLUSIES 45
6.2 AANBEVELINGEN 46

LITERATUUR 49

GEBRUIKTE SOFTWARE 51

BIJLAGE 1: BRONCODE 3DBUFFER 53

BIJLAGE 2: HET IN- EN UITVOERBESTAND VAN EEN BUFFER VOOR EEN KUBUS 61

1. Inleiding

De laatste jaren wordt steeds meer onderzoek gedaan naar de toepassing van driedimensionale GI-systemen, kortweg 3D-GIS. Deze onderzoeken zijn echter vaak meer gericht op de visualisatie van driedimensionale gegevens dan op analyses op in drie dimensies lokaliseerbare (geografische) fenomenen. Daarnaast wordt bij onderzoeken naar 3D-GIStechnieken nauwelijks het nut van dergelijke technieken beargumenteerd. Traditionele GIS-analyses, zoals bijvoorbeeld netwerkanalyses, blijken vaak goed toepasbaar in een driedimensionale omgeving. Er zijn echter situaties denkbaar waarbij driedimensionale analyses wel gewenst zijn. Zo kan het bijvoorbeeld nodig zijn om berekeningen met volumes uit te voeren, of de bepaling van topologische relaties als 'boven', 'onder', 'schuin rechts achter', etc. Dergelijke analyses zullen met name bij grootschalige toepassingen van belang zijn, omdat hierbij de hoogtecomponent in verhouding tot de planimetrische componenten steeds belangrijker wordt. Wanneer een (3D-)GIS opgezet wordt, moet gekozen worden, hoe de gegevens opgeslagen worden en hoe analysegereedschappen geïmplementeerd worden.

Dit afstudeeronderzoek is gericht op het de vraag: '*Hoe kan een 3D-GIS voor grootschalige toepassingen opgezet worden?'*. De nadruk ligt hierbij niet op visualisatie, maar op de opslag van de gegevens en op de analysegereedschappen, alhoewel beide componenten beïnvloed worden door, en invloed hebben op, de visualisatiemogelijkheden. Aan de hand van literatuuronderzoek worden bestaande analysevormen en beschikbare modelleringtechnieken geïnventariseerd. Aan de hand van een case en op basis van de inventaris, wordt onderzocht hoe een GIS-toepassing in drie dimensies gemodelleerd kan worden. Ten slotte wordt aan de hand van een praktijktest een analysegereedschap, namelijk een driedimensionale bufferzone, nader uitgewerkt.

Dit rapport is als volgt ingedeeld. In hoofdstuk 2 wordt de term '3D-GIS' gedefinieerd. Daarnaast worden in dat hoofdstuk GIS-analyses en geometrische modellen geïnventariseerd. De case wordt beschreven in hoofdstuk 3, waarin aangegeven is hoe een algemeen driedimensionale GIS gemodelleerd kan worden. Op basis van deze hoofdstukken wordt vervolgens de driedimensionale buffering als analyse vorm uitgewerkt voor objecten die gemodelleerd zijn met behulp van zogenoemde Tetrahedral Networks. Hoofdstuk 4 beschrijft hoe dergelijke netwerken in CGAL geïmplementeerd zijn. CGAL is een programmabibliotheek dat bestemd is voor gebruikt in CAD, GIS, VR, etc. en wordt gebruikt om een programma te schrijven dat een driedimensionale buffer genereert. In hoofdstuk 5 wordt de driedimensionale buffering op basis van tetraëder netwerken daadwerkelijk uitgewerkt door te beschrijven hoe een dergelijke buffer in theorie gemaakt kan worden, en hoe dat in de praktijk, op basis van CGAL, kan. In hoofdstuk 6 volgen conclusies en aanbevelingen.

2. Definities

Aangezien het in dit rapport beschreven onderzoek de opbouw van een 3D-GIS beschrijft, en een 3D-GISanalyse nader uit werkt in een praktijktest, is het van belang om vast te stellen wat in dit rapport de term 3D-GIS gedefinieerd wordt, vooral omdat daar in de literatuur geen eenheid over bestaat. Dit zelfde geldt ook voor GIS-analyses. Daarnaast bestaan er verschillende manieren om objecten in drie dimensies te representeren, en de keuze daaruit heeft invloed op de te ontwikkelen algoritme voor de 3D-analyse, maar ook op de visualisatiemogelijkheden.

Dit hoofdstuk geeft antwoord op bovenstaande achtergrondproblemen. In paragraaf 2.1 wordt de term 3D-GIS gedefinieerd. Paragraaf 2.2 inventariseert de bestaande GIS-analyses en geeft aan hoe deze in drie dimensies toegepast kunnen worden, waarna paragraaf 2.3 tenslotte een overzicht geeft van 3D-modelleringstechnieken.

2.1 een definitie van 3D-GIS

In de literatuur is in toenemende mate informatie te vinden over GI-systemen waarbij een driedimensionale component een rol speelt. Deze rol is echter vaak verschillend, evenals het doel van dergelijke GI-systemen, hetgeen zich ook uit in een veelheid aan termen om een 'driedimensionale GIS' aan te geven. Zo bestaan er 3DGI-systemen, GeoVE's, VRGI-systemen, 2,5DGI-systemen, etc.

Het valt buiten de doelstellingen van dit rapport om alle vormen van 3D-GIS te bespreken, maar grofweg kan de term 3D-GIS op de volgende manier ingedeeld worden:

1. GIS met VR-interface
2. GIS met terreinmodel
3. 'echt' 3D-GIS

Bovenstaande indeling is niet normatief bedoeld, maar beschrijft wat men in de literatuur met de term 3D-GIS bedoelt.

Meestal wordt een 3D-GIS beschouwd als een GIS gekoppeld aan Virtual Reality (VR) of, algemener, een 'Virtual Environment' (VE) om GIS-gegevens aan te spreken [loo98, mac99, etc.]. Dergelijke systemen laten zich kenmerken door driedimensionale visualisaties, waarbij de virtual environment dient als interface voor 2D-GISfunctionaliteit.

Vaak komt men de term 3D-GIS tegen om een GIS te beschrijven die gebaseerd is op een terreinmodel. De aanduiding '3D' is in deze context ongelukkig gekozen, omdat het vaak gaat om een systeem waarbij de gegevens op twee dimensies geïndexeerd wordt, dus op de x- en y-coördinaat, en waarbij de hoogteformatie als (pseudo-)attribuut aan deze planaire coördinaten wordt gekoppeld [o.a. rap91]. Hierdoor is het niet mogelijk om meer dan één hoogtecomponent per coördinatenpaar op te slaan, hetgeen voor een terreinmodel in de praktijk voldoende is. De term '2,5D' is in dit geval een betere benaming.

In dit rapport wordt met 3D-GIS een 'echte' 3D-GIS aangeduid. Worboys [wor95] geeft een omschrijving voor een dergelijke 3D-GIS: 'A three-dimensional Spatial Information system should be able to model, represent, manage, manipulate, analyse and support decisions based upon information associated with three-dimensional phenomena'. Dit houdt in, dat de gegevens in een 'echte' 3D-GIS niet alleen in drie dimensies gevisualiseerd kunnen worden, maar dat de gegevens ook in een 3D-database worden opgeslagen, en dat 3D-analyses en 3D-bewerkingen mogelijk moeten zijn. Een 'echte' 3D-GIS is daarom een geïntegreerde 3D-GIS.

2.2 GIS-analyses: een inventarisatie

Voordat bepaald kan worden welke GIS-analyses beter of slechts in drie dimensies uitgevoerd kunnen worden, is het belangrijk om de bestaande GIS-analyses te inventariseren. In de literatuur bestaat geen consensus over welke analyses uitgevoerd (moeten) kunnen worden in een GIS. Hierdoor bestaan er verschillende manieren om de GIS-analyses te categoriseren. Ook de term 'GIS-analyse' zelf is niet scherp gedefinieerd. Vergelijkbare termen zijn 'GIS-gereedschap', 'analysegereedschap', 'GIS-functionaliteit'. In dit rapport worden deze termen gebruikt, waarbij die bewerkingen bedoeld worden, die in een GIS ingezet kunnen worden om ruimtelijke problemen op te lossen. Databasemanagement wordt in dit rapport bijvoorbeeld niet tot GIS-functionaliteit gerekend.

Een bekende manier om GIS-functionaliteit in te delen is bijvoorbeeld door te bepalen welke vragen er mee opgelost kunnen worden:

- locatie (wat is daar?)
- toestand (waar is het?)
- trend (wat is er veranderd?)
- route (wat is de beste weg naar…?)
- patroon (wat is het patroon?)
- model (wat… als…?)

Bovenstaande opsomming geeft een heldere indeling van functionaliteit die men van een GIS kan verwachten, maar is voor toepassing in dit onderzoek te algemeen.

Door verschillende bronnen te vergelijken kan een overzicht worden gegeven van functionaliteiten die een GIS over het algemeen moet bevatten. Deze lijst is door verschillende definities niet volledig, een voorbeeld van verschillende definities voor dezelfde analysegereedschap is de buffer; deze kan beschouwd worden als een afstandsoperatie, maar ook als overlayoperatie. Door de verschillende definities is in de volgende inventarisatie enige subjectiviteit onvermijdelijk. Over het algemeen zijn de volgende functionaliteiten te onderscheiden [bat98, bur98, loo98, orf98, ran97, vri99]:

- directionele relaties
- transformaties
- objecteigenschappen
- metriek
- topologie
- attribuutoperaties
- thematische functies
- laagoperaties
- zichtbaarheidfuncties
- visualisatie

Tussen de verschillende analyses in bovenstaande opsomming zijn geen duidelijke grenzen te trekken. Zo kan het bijvoorbeeld wenselijk zijn om attribuutoperaties op basis van directionele eigenschappen uit te voeren. Daarnaast moet opgemerkt worden, dat de analyses die met een GIS uitgevoerd kunnen worden, afhankelijk zijn van het doel van een GIS. In de volgende subparagrafen zal elke analysevorm nader beschreven worden.

directionele relaties

Directionele relaties geven de ligging van een object ten opzichte van andere objecten aan. Door relaties als ‘boven’, ‘onder’, etc. toe te laten, is deze analyse in drie dimensies toe te passen. De volgende directionele relaties kunnen worden onderscheiden:

- boven
- onder
- achter
- voor
- links
- rechts
- ten noorden van
- ten zuiden van
- ten oosten van
- ten westen van
- etc.

De uitkomst van een analyse op basis van directionele relaties, kan een verzameling objecten zijn, of een booleaanse waarde. De opdracht ‘selecteer alle objecten ten noorden van dit object’ zal een verzameling objecten geven, terwijl de vraag ‘ligt dit object voor een ander object?’ met ja of nee beantwoord kan worden.

transformaties

Met transformatie wordt bedoeld, dat objecten in een GIS geschaald, geroteerd en verplaatst kunnen worden. Hoewel de transformatie vaak als analyse wordt genoemd, is het in feite een functionaliteit van een GIS dat de gebruiker kan helpen een bepaald probleem te analyseren en op te lossen. Hierbij dient het getransformeerde object als invoer voor verdere ‘echte’ analyses.

Transformaties in 2D worden bepaald door ten hoogste 5 transformatie parameters: twee schaalfactoren, twee translatieparameters en een rotatiehoek. De vertaling van deze ‘analyse’ houdt een uitbreiding in naar 9 transformatieparameters: drie schaalfactoren, drie translatieparameters en drie rotatiehoeken.

objecteigenschappen

Met objecteigenschappen worden hier de eigenschappen van een object bedoeld die op basis van de geometrie van het object te beschrijven zijn, zoals oppervlakte, middelpunt, etc. Deze analysevorm kan uitgebreid worden voor drie dimensies met eigenschappen als inhoud, massamiddelpunt van een volume, etc.

metriek

Met metriek worden metingen van geometrische relaties tussen objecten bedoeld, zoals de afstand tussen twee punten, de maximum- en minimumafstand tussen twee (niet-punt-)objecten, de hoek tussen objecten vanuit een bepaald punt, etc. In tegenstelling tot directionele relaties, is de uitkomst hier een getal of een verzameling getallen.

topologie

Onder topologie wordt verstaan: een beschrijving van de ruimte die onafhankelijk is van transformaties op deze ruimte. Burrough en McDonnell beschrijven topologie als ‘een term dat verwijst naar de continuïteit van ruimte en ruimtelijke eigenschappen, zoals connectiviteit, die niet beïnvloed worden door een continue verstoring’ [bur98]. Topologie beschrijft, met andere woorden, de vorm van de ruimte; de grens van een gesloten lichaam verdeelt de ruimte bijvoorbeeld in een ruimte binnenin het lichaam, een ruimte buiten het lichaam en de grens zelf. Van Oosterom [oos94] geeft aan, dat er slechts vijf aparte topologische relaties beschreven hoeven te worden, om alle mogelijke topologische relaties tussen elke mogelijke combinatie van twee objecten te beschrijven: *touch, in, cross, overlap* en *disjoint:*
met:
object: λ
grens van het object: $\partial\lambda$
interieur van het object: $\lambda^0 = \lambda - \partial\lambda$
kunnen deze relaties opgeschreven worden als:

- touch:
  $\langle \lambda_1, touch, \lambda_2 \rangle \Leftrightarrow \left(\lambda_1^0 \cap \lambda_2^0 = \varnothing\right) \wedge \left(\lambda_1 \cap \lambda_2 \neq \varnothing\right)$
- in:
  $\langle \lambda_1, in, \lambda_2 \rangle \Leftrightarrow \left(\lambda_1 \cap \lambda_2 = \lambda_1\right) \wedge \left(\lambda_1^0 \cap \lambda_2^0 \neq \varnothing\right)$
- cross:
  $\langle \lambda_1, cross, \lambda_2 \rangle \Leftrightarrow \left(\dim(\lambda_1^0 \cap \lambda_2^0) < \max(\dim(\lambda_1^0), \dim(\lambda_2^0))\right) \wedge \left(\lambda_1 \cap \lambda_2 \neq \lambda_1\right) \wedge \left(\lambda_1 \cap \lambda_2 \neq \lambda_2\right)$
- overlap:
  $\langle \lambda_1, overlap, \lambda_2 \rangle \Leftrightarrow \left(\dim(\lambda_1^0) = \dim(\lambda_2^0) = \dim(\lambda_1^0 \cap \lambda_2^0)\right) \wedge \left(\lambda_1 \cap \lambda_2 \neq \lambda_1\right) \wedge \left(\lambda_1 \cap \lambda_{21} \neq \lambda_2\right)$
- disjoint:
  $\langle \lambda_1, disjoint, \lambda_2 \rangle \Leftrightarrow \lambda_1 \cap \lambda_2 = \varnothing$

Bovenstaande relaties gelden ook in drie dimensies. Hoewel gediscussieerd kan worden, of nabijheid werkelijk een topologische relatie is, wordt deze wel als zodanig beschouwd.

attribuutoperaties

Met attribuutoperaties worden operaties op de database bedoeld, zoals bijvoorbeeld selectie van records op basis van een query. Een veelgebruikte querytaal, maar zeker niet de enige, is natuurlijk SQL. Ook overlayoperaties, zoals in de volgende paragraaf apart genoemd, worden vaak gerekend tot attribuutoperaties. De attribuutoperaties zijn in principe dimensieonafhankelijk, maar de opslag van gegevens in drie dimensies kan bijzondere eisen stellen aan de databaserepresentatie van deze gegevens, en daarmee op de soort en de manier van attribuutoperaties die op de dataset uitgevoerd kunnen worden. Zo kan het bijvoorbeeld beter zijn, om de gegevens op te slaan in een objectgeoriënteerde database, waardoor SQL niet toepasbaar is.

laagoperaties

Met laagoperaties (eng.: *layer based operations*) worden analyses bedoeld die gebaseerd zijn op verschillende lagen. Er zijn twee soorten laagoperaties te onderscheiden: bufferzones en '*boolean overlay*'[wor95].

Boolean overlay is de combinatie van één of meerdere lagen tot een enkele laag, dat gebaseerd is op een booleaanse operatie op de invoerlagen. De boolean overlay is niet aan een bepaalde dimensie gebonden, als de term 'laag' in meerdere dimensies beschouwd kan worden.

Bufferzones zijn gebieden die locaties bevatten, die zich binnen een gegeven reikwijdte van een gegeven *set features* bevinden. Buffers zijn meestal cirkelvormig of rechthoekig rond punten, of corridors van een constante breedte om lijnen en gebieden [wor95, bur98]. Ook in driedimensionale GI-systemen worden dergelijke tweedimensionale bufferzones meestal voldoende geacht. Door de buffer zone te definiëren als een volume met een bepaalde vorm en grootte, wordt deze analysevorm echter echt driedimensionaal. Alle objecten die zich geheel of gedeeltelijk binnen een bufferzone bevinden, worden geselecteerd door een boolean overlay uit te voeren met als invoer de bufferzone en de lagen die de gewenste objecten bevatten.

zichtbaarheidanalyse

Met zichtbaarheidanalyse wordt bepaald, welke objecten vanuit een bepaald punt zichtbaar zijn (eng.: *line-of-sight*), welke objecten de zicht belemmeren (eng.: *obscuration*), welke oppervlaktes zichtbaar zijn (eng.: *areas-seen*). Over het algemeen kan een zichtbaarheidanalyse met tweedimensionale gegevens uitgevoerd worden, maar de zichtbaarheidanalyse geldt dan slechts voor een horizontaal vlak. De toepassing in drie dimensies biedt extra mogelijkheden, zo kan een zichtlijn in andere dan horizontale richtingen berekend worden, en kunnen verwante analyses uitgevoerd worden zoals, bijvoorbeeld, lichtinval of schaduwering.

visualisatie

Visualisatie wordt niet altijd als een GIS-analyse erkend, omdat visualisatie op zich geen analyse is. Toch past visualisatie in dit overzicht thuis, omdat het een belangrijk hulpmiddel voor de GIS-gebruiker kan zijn om problemen te lokaliseren, te analyseren en op te lossen. De gebruiker voert in feite de analyse uit, waarbij de visualisatie als invoer dient.

Een bekende visualisatievorm is natuurlijk de kaart, maar tegenwoordig zijn andere visualisatievormen mogelijk zoals animatie en simulatie. Ook het weergeven van variabelen door middel van geluid (sonificatie) of tast (bijvoorbeeld *force feedback*) wordt over het algemeen als visualisatie beschouwd. Bij visualisatie van driedimensionale gegevens wordt al gauw gedacht aan virtual reality, waarbij de gebruiker door een al dan niet bestaande omgeving kan navigeren en hiermee interactie kan hebben. Maar er bestaan ook minder interactieve 3D-visualisaties, zoals bijvoorbeeld een terreinmodel waarbij van tevoren en *fly-through* is berekend.

2.3 geometrische modellen

Voordat een driedimensionale GIS opgezet kan worden, is het van belang om vast te stellen op welke manier objecten geometrisch gerepresenteerd worden. Er zijn verschillende representaties beschikbaar waarmee objecten in drie dimensies gemodelleerd kunnen worden en de keuze daaruit is van invloed op een aantal factoren. Sommige technieken zijn bijvoorbeeld bedoeld om een snelle visualisatie mogelijk te maken, andere technieken zijn bedoeld om efficiënt mee te kunnen rekenen, terwijl andere technieken weer meer geschikt zijn om objecten in een efficiënte gegevensstructuur op te slaan. Bij de keuze voor een bepaalde geometrische representatie voor toepassing in een driedimensionale GIS moet een afweging worden gemaakt tussen deze factoren.

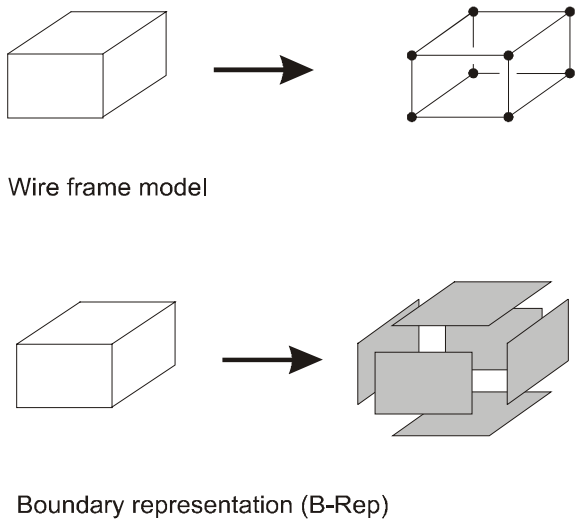
De meeste beschrijvingen van geometrische objectmodellen gaan uit van een onderverdeling in draadmodellen, oppervlaktemodellen en lichaamsmodellen. Dat is een handzame, maar onnauwkeurige indeling, omdat het slechts aangeeft welke primitieve de hoogste dimensie heeft in een model (resp. lijn, vlak, volume). Breuning [bre96]geeft een indeling die nauwkeuriger is:

- vectorgebaseerde representaties,
- getesseleerde representaties,
- analytische representaties,
- hybride representaties.

Deze paragraaf beschrijft in deze volgorde de representaties die bij deze categorieën horen nader, waarbij voor een groot deel gebruik wordt gemaakt van de beschrijving van Breuning [bre96] en van Raper & Kelk [rap91]. Ten slotte wordt aangegeven welke objectrepresentaties het meest geschikt zijn voor toepassing in een 3D-GIS. Elke representatie bestaat uit een aantal primitieven. Dit kunnen 0-dimensionale, 1-dimensionale, 2-dimensionale en 3-dimensionale primitieven zijn, bijvoorbeeld respectievelijk een punt, een lijn, een vlak, een volume. In de literatuur (o.a. [wor95]) worden ook de termen *0-extent, 1-extent, 2-extent* en *3-extent* gebruikt. Omwille van de duidelijkheid wordt waar nodig de engelse benaming, ook van de verschillende representaties, gehandhaafd.

vectorgebaseerde representaties

Bij vectorgebaseerde representaties (afbeelding 2.1) ligt de geometrische informatie van objecten vast door de posities van punten. Andere geometrische vormen, zoals lijnen, polygonen, lichamen, etc., worden hiervan afgeleid. Een bekende vectorgebaseerde representatie is het draadmodel, of *wireframe model*. Een andere representatievorm is de *Boundary Representation (B-Rep)*. Vaak wordt apart de *Vector Boundary Representation (VBR)* onderscheiden, maar deze is in een voorbeeld van een B-Rep. De volgende subparagrafen beschrijven daarom slechts de wireframe en de Boundary Representation.



Afbeelding 2.1: Vectorgebaseerde representaties. Vectorgebaseerde representaties zijn onder te verdelen in Wireframe models (bovenste gedeelte van de afbeelding) en Boundary Representations (onderste gedeelte van de afbeelding) In een Wireframe model worden alleen vertices en randen van een object opgeslagen, terwijl in de Boundary Representation (of B-Rep) een object gerepresenteerd wordt door zijn grenselementen.

Wireframe model

Bij het wireframe model, of draadmodel, wordt de geometrie van een object gerepresenteerd door lijnsegmenten of door curven. De opslag van een dergelijk model vindt plaats in twee tabellen: een tabel met (de coördinaten van) vertices, en een tabel met de randen van het object en de begin- en eindpunten van deze randen [vin95].

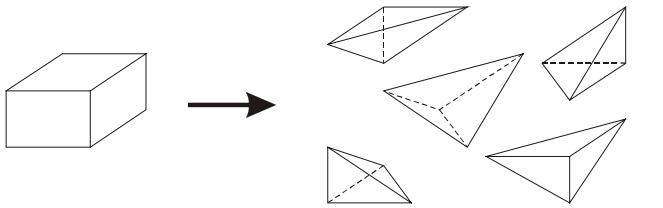
Een groot nadeel van dit model is, dat er geen informatie gekoppeld kan worden aan oppervlakten of aan volumes, waardoor dit model niet geschikt is voor gebruik in een 3D-GIS [bre96]. Omdat door de afwezigheid van vlakinformatie visualisatie van vlakken niet of via een omweg mogelijk is, is het draadmodel ook minder geschikt voor visualisatie.

Boundary Representation (B-Rep)

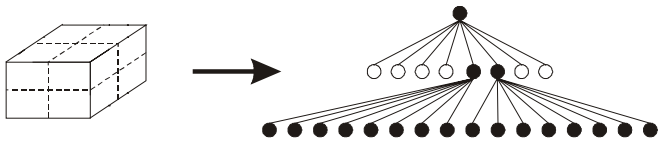
Bij de Boundary Representation (B-Rep) wordt een lichaam gerepresenteerd door de opslag van de grenselementen (eng.: *bounding elements*). Deze grenselementen kunnen de zijdes (eng.: *faces*) zijn van het object, maar ook functies. Een voorbeeld van dat laatste is de Function Boundary Representation, dat in paragraaf 5.3 beschreven wordt. Met B-Rep wordt meestal de Vector Boundary Representation (VBR), of Polygonal Vector Representation (PVR) bedoeld, waarbij opslag plaats vindt van de zijdes, de randen, en de vertices. Deze elementen kunnen opgeslagen worden in een graafstructuur, waardoor ook topologische informatie opgeslagen wordt. De belangrijkste nadelen van dit model zijn, dat geen informatie gekoppeld kan worden aan volumes, en dat operaties op objecten complex zijn. Visualisatie van objecten gemodelleerd met behulp van een B-Rep, is echter erg simpel omdat gemakkelijk gebruik gemaakt kan worden van standaard visualisatiemethodes zoals bijvoorbeeld z-buffering.

getesseleerde representaties

Als een object gemodelleerd wordt door dit object te verdelen in discrete eenheden, is er sprake van een getesseleerde representatie (eng.: *tesselating representation*). Een voorbeeld hiervan in 2D is de raster representatie. Voorbeelden in 3D zijn de in de volgende subparagrafen beschreven *Cell Decomposition*, en de *Spatial Occupancy Enumeration (SOE)* (afbeelding 2.2).



Cell Decomposition



Spatial Occupancy Enummeration (SOE)

afbeelding 2.2: Getesseleerde representaties. Het bovenste gedeelte van de afbeelding is een voorbeeld van Cell Decomposition, waarbij een object gerepresenteerd wordt door een vereniging van simpele, niet-overlappende en onregelmatige cellen. In het voorbeeld bestaan de cellen uit viervlakken., maar de cellen mogen verschillende vormen aannemen. Het onderste gedeelte van de afbeelding is een voorbeeld van Spatial Occupancy Enumeration, waarbij een object gerepresenteerd wordt door een onderverdeling in regelmatige cellen. Deze cellen worden geïndexeerd waardoor opslag in een boomstructuur plaats kan vinden.

Cell Decomposition

Objecten gerepresenteerd door cell decomposition, bestaan uit de vereniging van simpele, niet-overlappende en onregelmatige cellen (zie afbeelding 2.1, bovenste gedeelte). Deze cellen, of bouwstenen, kunnen bijvoorbeeld kubussen, blokken, cilinders of tetraëders zijn. Deze cellen mogen daarbij verschillende vormen en afmetingen aannemen, hetgeen gelijk een nadeel is, omdat de topologie van zo'n model niet sterk is en sommige analyses hierdoor complex kunnen zijn. Een voordeel van Cell Decomposition is dat praktisch elk lichaam, ook met gebogen oppervlakken, benaderd kan worden.

Spatial Occupancy Enumeration (SOE)

Spatial Occupancy Enumeration (SOE) is een specifieke vorm van cell decomposition, maar wordt vaak, en daarom ook in dit rapport, apart beschouwd. Het object wordt bij de SOE samengesteld uit identieke cellen, die in een vaststaand, regelmatig grid gerangschikt zijn (zie afbeelding 2.1, onderste gedeelte). Deze cellen zijn over het algemeen kubussen. Elke cel is zo te identificeren door middel van een unieke code, waardoor een efficiënte opslag in bijvoorbeeld een octree mogelijk is, analoog aan de quadtree decompositie van gebieden in 2D. Hierdoor kunnen operaties op objecten simpel uitgevoerd worden. Het nadeel van SOE is, dat de opslag weliswaar efficiënt is, maar wel veel ruimte in beslag neemt. Daarnaast worden sommige objecten, afhankelijk van de celgrootte, onvoldoende benaderd.

analytische representaties

Het analytisch representeren van een object houdt in dat de geometrie door functievergelijkingen of door parameters wordt gerepresenteerd (afbeelding 2.3). De belangrijkste vormen zijn:

- *Function Boundary Representation (FBR),*
- *Sweep Representation,*
- *Primitive Instancing (*of *Parameter Representation).*

Deze vormen worden in de volgende subparagrafen besproken, alhoewel bij voorbaat gesteld wordt, dat analytische representaties niet geschikt zijn voor toepassing in een 3D-GIS, omdat operaties op objecten te complex zijn, en daardoor te veel tijd kosten.

Function Boundary Representation (FBR)

De Function Boundary Representation (FBR) is een B-Rep, waarbij de grenselementen bestaan uit analytische functies, of uit oppervlakken die door interpolatie of schattingsmethoden gedefinieerd worden. De belangrijkste voorbeelden van dergelijke oppervlakken zijn Bilineaire oppervlakken, Bézier surface patches, B-Spline surface patches en Coons bicubic surface patches [vin95, bre96].

Sweep Representation

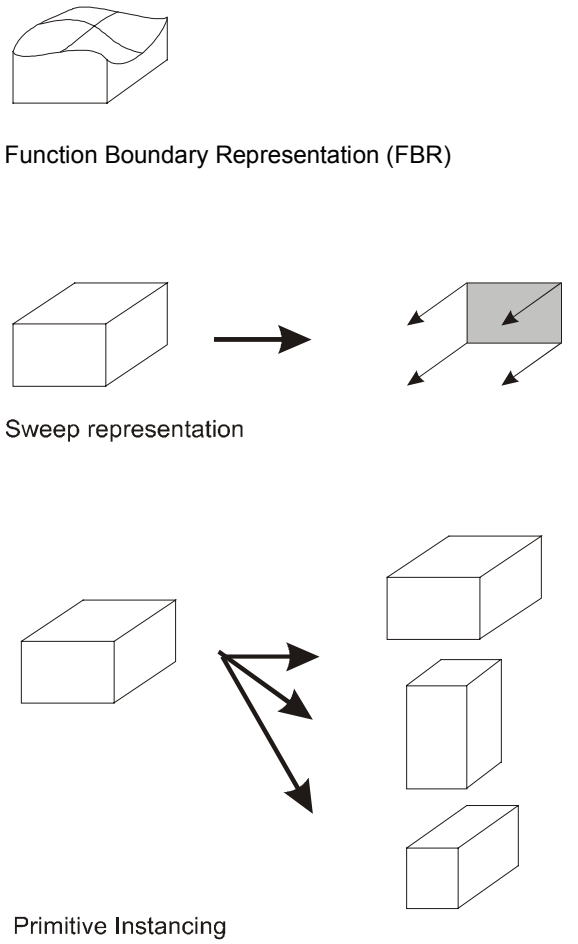
De Sweep Representation houdt in, dat de geometrie van objecten gerepresenteerd worden door een bewerking van een primitieve geometrische vorm, waarbij de volgende bewerkingen mogelijk zijn:

- rotatie rond een as (*rotation sweep*),
- translatie langs een lijn (*translation sweep*),
- gelijktijdige translatie en rotatie (*general sweep*).

Een simpel voorbeeld is bijvoorbeeld een kegel; deze is te beschrijven als een rechthoekige driehoek dat geroteerd wordt rond een as, waarbij deze rotatieas samenvalt met een rechte zijde.

Primitive Instancing

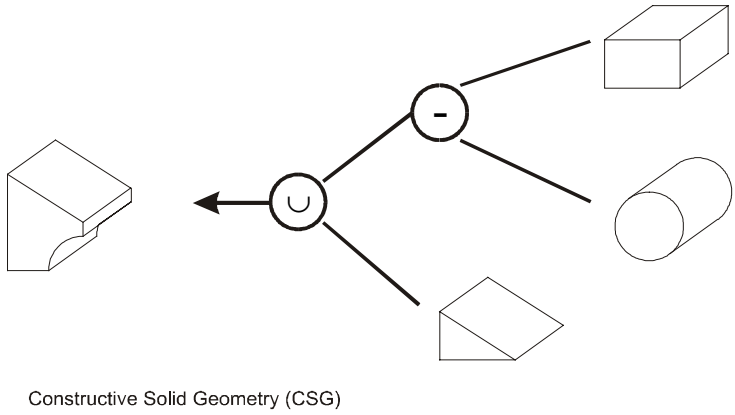
Primitive Instancing, of Parameter Representation, beschrijft een object aan de hand van een vast aantal parameters. Een blok kan bijvoorbeeld beschreven worden aan de hand van de parameters: lengte, hoogte, breedte. Een *instance*, of primitieve, is gedefinieerd door een set numerieke waarden, terwijl deze waarden gedefinieerd worden in een mathematische vergelijking dat een bepaald lichaam beschrijft.



Afbeelding 2.3: Analytische representaties. De belangrijkste varianten van analytische representaties zijn de Function Boundary Representation, de Sweep Representation, en Primitive Instancing. Bij de Function Boundary Representation (afbeelding, boven) worden grenselementen opgeslagen, die door interpolatie of schattingsmethoden gedefinieerd zijn. In de Sweep Representation (afbeelding, midden) worden objecten gerepresenteerd door een bewerking op een primitieve. In de afbeelding hiernaast wordt het blok bijvoorbeeld gerepresenteerd door een vierkant langs een rechte as 'uit te rekken' (zgn. translation sweep). Primitive Instancing (afbeelding, onder) houdt in, dat er een basisobject gedefinieerd wordt aan de hand van een aantal parameters, zoals bijvoorbeeld lengte, breedte en hoogte. Van dit basisobject worden instanties afgeleid die de geometrie vastleggen.

hybride representaties

Hybride representaties zijn representaties waarbij een combinatie wordt gebruikt van een vectorgebaseerde en van een getesseleerde representatie. Het meest relevante voorbeeld hiervan is de hieronder besproken *Constructive Solid Geometry (CSG)*.



Afbeelding 2.4: Hybride representatie. Het belangrijkste voorbeeld van een hybride representatie is Constructive Solid Geometry, waarbij een object gerepresenteerd wordt door een aantal booleaanse operaties (vereniging, doorsnede, etc.) op basisobjecten. In de afbeelding wordt het object bijvoorbeeld gevormd door eerst een cilinder van een blok af te trekken, en het resulterende object te verenigen met een prisma. CSG is een hybride representatie, omdat de basisobjecten natuurlijk zelf geometrisch gerepresenteerd worden.

Constructive Solid Geometry (CSG)

Objecten, gerepresenteerd met behulp van Constructive Solid Geometry (CSG), worden gevormd door het uitvoeren van transformaties (translatie, rotatie en schaling) en booleaanse operaties (vereniging, verschil, doorsnede) op basisobjecten, zoals kubussen, cilinders kegels, etc. Het te beschrijven object kan worden weergegeven in een binary tree, waarbij de basisobjecten opgeslagen worden in de eindpunten (eng.: *leaves*), en de operaties en transformaties worden opgeslagen in de knooppunten (eng.: *nodes*). CSG is een hybride representatie omdat de basisobjecten zelf door middel van een vectorgebaseerde representatie gemodelleerd worden.

Door de opslag van objecten in een binary tree is een efficiënte opslag mogelijk, maar om objecten weer te geven of analyses uit te voeren op objecten, moet steeds de hele boom doorlopen worden, waardoor CSG voor zowel analyse als visualisatie te traag kan zijn. Daarnaast kan een object op verschillende manieren samengesteld worden, waardoor geen éénduidige representatie mogelijk is en dus ook geen éénduidige topologie vast te leggen is.

objectrepresentaties en 3D-GIS

De meeste objectrepresentaties zijn ontwikkeld voor CAD-toepassingen. Een 3D-GIS stelt andere eisen aan de te kiezen representatievorm. Van belang is, dat een efficiënte opslag mogelijk is, en dat bewerkingen met objecten niet te complex zijn. Als visualisatie in de vorm van Virtual Reality gewenst is, moet bovendien een snelle visualisatie van de objecten mogelijk zijn. Een belangrijke overweging is de sterkte van de topologische structuur van een bepaalde representatie.

De analytische representaties zijn niet geschikt, omdat de operaties op objecten te complex zijn, en daardoor te veel (processor-)tijd in beslag nemen. Ook de hybride representatie, CSG in het bijzonder, is om dezelfde reden ongeschikt. CSG is laat bovendien geen snelle visualisatie toe. Van de vectorgebaseerde representaties is het draadmodel ongeschikt, omdat er geen informatie aan oppervlakken en volumes te koppelen is. De B-Rep, in de vorm van een Vector Boundary Representation, maakt snelle operaties op de objecten en een snelle visualisatie wel mogelijk, en bovendien is informatie te koppelen aan oppervlakken. Ook de Cell Decomposition en de SOE bieden het voordeel van snelle objectoperaties. Voor zowel de B-Rep als beide getesseleerde representaties geldt als nadeel, dat de opslag veel ruimte in beslag kan nemen.

De geschikte objectrepresentaties voor toepassing in 3D zijn, samenvattend:

- B-Rep / VBR,
- Cell Decomposition,
- Spatial Occupancy Enumeration.

De keuze uit voorgaande opsomming hangt af van het doel van de representatie. Als de nadruk op een snelle visualisatie ligt, dan is de B-Rep geschikt omdat deze representatie een snelle visualisatie mogelijk maakt. De Spatial Occupancy Enumeration (SOE) is, door de opslag van cellen in een boomstructuur, geschikt als de nadruk op operaties op objecten ligt.

Als meerdere doelen gesteld worden, dus bijvoorbeeld een snelle visualisatie naast een snelle gegevenstoegang, kan er voor worden gekozen om voor objecten verschillende representaties op te slaan. Een nadeel hiervan is wel, dat er meer opslagruimte nodig is, en dat de consistentie goed gewaarborgd moet worden. Er moet met andere woorden veel aandacht worden besteed aan de koppeling tussen de verschillende representaties. Bovendien is dan van een geïntegreerde ('echte') 3D-GIS geen sprake meer.

Tempfli [tem98] geeft aan, dat de B-Rep voldoende flexibiliteit biedt voor operaties op (topografische) objecten en bovendien geschikt is voor metingen en visualisatie. Op volume gebaseerde representaties, zoals CSG en SOE, noemt hij minder geschikt, omdat deze niet zowel voor visualisatie als objectoperaties geschikt zijn. Het uitgangspunt bij die constatering is, dat volumes voor topografische objecten niet relevant zijn, zo is het voldoende om een huis te modelleren als een aantal muren met een dak.

Dat een volume in een 3D-GIS een irrelevante primitieve is, geldt niet altijd en zal voor sommige toepassingen zelfs erg belangrijk zijn. Voorbeelden van dergelijke toepassingen zijn de mijnbouw, geologie, archeologie en maritieme wetenschappen; toepassingen waar een grens een andere grootheid is dan het volume die die grens omsluit. Maar ook voor GI-systemen waar het volume, of 0-extent, voor de geometrische of thematische objectbeschrijving van onderschikt of geen belang is, kan een volumegebaseerd model gewenst zijn omdat in een dergelijk model de topologie beter te beschrijven is dan in een Boundary Representation.

De topologie in een B-Rep is zwak vastgelegd. Omdat de grensvlakken verschillende vormen aan kunnen nemen (driehoeken, vierhoeken, etc.), is de relatie tussen deze grenselementen onderling, en met objecten van een lagere dimensie, moeilijk te bepalen. Een sterkere topologie wordt bereikt als de grenselementen gelijkvormig zijn. De grenzen van een object worden dan bijvoorbeeld opgeslagen als een verzameling onderling aan elkaar gerelateerde driehoeken. Maar ook dan zijn simpele vraagstukken, zoals het bepalen van het interieur (de binnenkant) van een object, moeilijk op te lossen, terwijl juist een driedimensionale GIS zich voor dergelijke vraagstukken dient te lenen. De bepaling van het interieur van een object is bijvoorbeeld van belang als alle objecten geselecteerd moeten worden, die zich (deels) in een ander object bevinden. Als een sterke topologie gewenst is, dan verdient een volumegebaseerde representatie de voorkeur.

Een geïntegreerd driedimensionaal Geografisch Informatiesysteem is een 3D-GIS, dat zowel faciliteiten biedt om objecten in drie dimensies op te slaan en te manipuleren, als om deze en gegenereerde objecten in drie dimensies te visualiseren. Een voorbeeld van een niet-geïntegreerd 3D-GIS is een GIS waarbij voor de visualisatie een aparte objectdatabase wordt bijgehouden dat gekoppeld is aan de ruimtelijke database. Integratie van opslag, manipulatie en visualisatie houdt onder meer in dat de objecten omwille van consistentie in één datastructuur worden opgeslagen. Een dergelijke datastructuur moet dan geschikt zijn voor de opslag van driedimensionale geometrie, thematiek en topologie, en zowel efficiënte operaties op de objecten als de snelle visualisatie van deze objecten.

Aan alle bovengenoemde eisen voldoet in feite alleen de Cell-Decomposition. De Cell-Decomposition maakt een snelle visualisatie, efficiënte objectoperaties, en opslag van thematiek en geometrie op basis van driedimensionale primitieven mogelijk. Slechts de topologie is in de Cell-Decomposition zwak, tenzij de niet-overlappende cellen alleen bestaan uit tetraëders. De topologie is dan juist zeer sterk omdat de relaties tussen de primitieven hoekpunten, randen, zijdes en cellen, allen bekend zijn. Zo is van een tetraëder bijvoorbeeld bekend, dat deze precies vier naburige tetraëders heeft en dat een tetraëder met een naburige tetraëder exact één driehoek deelt, die daardoor een oriëntatie krijgt. Een Cell-Decomposition waarbij de cellen alleen uit tetraëders bestaan, wordt een *Tetrahedral Network* (TEN) genoemd en wordt in hoofdstuk 3 nader uitgewerkt omdat de TEN voor een geïntegreerde 3D-GIS het meest geschikt is.

3. De opslag in een 3D-GIS toegepast op een case

Het uitgangspunt van dit rapport is, dat 3D-GIS pas een praktisch nut heeft bij grootschalige toepassingen en als de componenten opslag, bewerking en presentatie in drie dimensies met elkaar geïntegreerd zijn. Om te tonen hoe een 3D-GIS voor grootschalige toepassingen opgezet kan worden, wordt de datastructuur en een daarop gebaseerde analyse voor een dergelijke GIS beschreven. Dit hoofdstuk geeft aan hoe de datastructuur opgezet kan worden aan de hand van een case: de toepassing van een GIS op één gebouw.

In paragraaf 3.1 wordt de case in termen van functionaliteit beschreven. Paragraaf 3.2 beschrijft de criteria op basis waarvan het succes van dit onderzoek getest kan worden, waarna in paragraaf 3.2 een technische uitwerking van de case beschreven wordt, waarbij de aandacht ligt op de objectrepresentatie, en de opslag van deze representatie. Een analyse wordt in hoofdstuk 4 nader beschreven.

3.1 beschrijving van de case

In de literatuur zijn sporadisch voorbeelden te vinden van de toepassing van 3D-GIS. Over het algemeen betreft het dan modellen die stedelijk gebied beschrijven en vaak ligt de nadruk op het visuele aspect van 3D-GIS. In dit rapport wordt 3D-GIS echter toegepast op een grotere schaal, bijvoorbeeld op gebouwniveau. Ook op dit niveau zijn vraagstukken te bedenken, die geschikt zijn om met een GIS op te lossen, zoals bijvoorbeeld ten bate van gebouwbeheer, monitoren van persoonsstromen (bijvoorbeeld in een stadion), allocatie van winkels in een winkelcentrum. Overigens is op dit niveau de term 'Ruimtelijk Informatiesysteem' wellicht geschikter. Omwille van de duidelijkheid wordt in dit rapport de term GIS gehandhaafd.

In dit rapport wordt uit gegaan van de toepassing van 3D-GIS ten behoeve van de brandweer. De brandweer bezit voor belangrijke gebouwen zogenaamde aanvalsplannen, waarop vluchtroutes, aftappunten, etc. aangegeven worden. Daarnaast is informatie aanwezig over telefoonnummers, plaatsen van sleutels, etc. Deze kaarten zijn echter tweedimensionaal die, voor complex ingerichte gebouwen, moeilijk te interpreteren zijn. Daarnaast moet bij het opstellen van een dergelijk plan al rekening worden gehouden met alle mogelijk scenario's.

Aanvalsplannen zouden ook met gebruik van een 3D-GIS opgesteld kunnen worden. Hierdoor kan informatie gekregen worden over de locatie en aantal van de aanwezige bewoners, de bereikbaarheid van contactpersonen, de locatie van de dichtstbijzijnde aftappunten en brandblussers. Bovendien is simulatie mogelijk, bijvoorbeeld van rook- en vuurontwikkeling onder invloed van variabelen als bronlocatie, wind en in het gebouw aanwezige materialen. Ook vluchtroutes kunnen in drie dimensies berekend en gevisualiseerd worden. Het belangrijkste voordeel van een dergelijk systeem is, dat scenario's niet meer van tevoren ingeschat hoeven te worden, maar real-time berekend en gesimuleerd kunnen worden. Een dergelijke 3D-GIS leent zich goed om geïntegreerd te worden in programma's voor automatisch gebouwbeheer.

3.2 eisen aan de opslag in een 3D-GIS

Bij de keuze voor een bepaalde datastructuur moet aan een aantal eisen voldaan worden. Het opstellen van deze eisen is echter niet eenvoudig, omdat er geen vergelijkende cases zijn. Bovendien geldt dat de eisen afhankelijk zijn van het doel van de GIS; een 3D-GIS voor mijnwerkzaamheden moet aan andere voorwaarden voldoen dan een 3D-GIS dat vooral bedoeld is als presentatie- en communicatiemiddel bij stadsontwikkelingsprojecten. Aan een datastructuur voor een algemene 3D-GIS kunnen de volgende eisen gesteld worden:

- de datastructuur moet algemeen toepasbaar zijn;
- aansluiting aan visualisatie moet gewaarborgd zijn;
- de datastructuur moet snel te bevragen zijn;
- de datastructuur moet snelle objectoperaties toelaten;
- meerdere abstractieniveaus moeten gemodelleerd kunnen worden;

De eerste twee eisen zijn lijken wellicht niet relevant voor de toetsing, maar deze zijn het wel, omdat het gebruikte opslagmodel, dus de manier waarop de gegevens opgeslagen worden, belangrijk is voor de software die gebruikt en/of ontwikkeld moet worden. Als dat exotische programma's blijken te zijn, dan is het model (nog) niet praktisch toe te passen. De tweede eis heeft daar mee te maken. Een probleem bij de ontwikkeling van 3D-GIS is de koppeling tussen GIS-gegevens en visualisatiegegevens, omdat bevraging en analyse enerzijds en visualisatie anderzijds verschillende eisen stellen aan de opslag van gegevens. De geschiktheid van een 3D-GIS is dus te toetsen aan de manier waarop koppeling tussen GISgegevens en visualisatiegegevens plaats vindt. De snelheid van de toepassing is belangrijk, met name als de 3D-GIS, zoals in dit rapport, gebruikt wordt voor situaties, waarin snelheid van het grootste belang is. Het abstractieniveau is van belang om te kunnen beantwoorden, of het model voldoende detail bevat. Is het bijvoorbeeld van belang om te weten op welke locaties zich stopcontacten bevinden, of is slechts een locatie van de verdiepingen van belang voor het beantwoorden van 3D-GIS-vragen?

3.3 uitwerking van de case in 3D

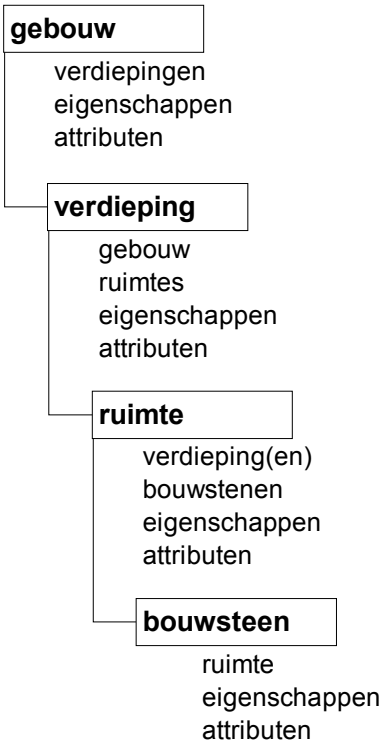
Een belangrijk aandachtspunt bij de uitwerking van de case, is de manier waarop de gegevens opgeslagen worden. Een belangrijke voorwaarde is, dat de gegevensstructuur verschillende abstractieniveaus (of *LoD: Levels-Of-Detail*) moet kunnen toelaten. Er bestaan twee soorten abstractieniveaus. Ten eerste duidt het op een 3D-visualisatietechniek, waarbij voor een object verschillende representaties worden opgeslagen, elk met een toenemende detaillering. Een huis wordt bijvoorbeeld opgeslagen als respectievelijk een blok, een blok met een dak, en als een blok met een dak en ramen. Op deze manier kunnen objecten die zich in de verte bevinden met minder detail, dus abstracter, weergegeven worden dan objecten in de nabijheid van de kijker. Met LoD wordt zo de visualisatiesnelheid verhoogd. Ten tweede duidt abstractieniveau op de mate van detail in de thematiek. Van een gebouw kan bijvoorbeeld informatie opgeslagen worden over de verdiepingen, maar ook over de ruimtes in het gebouw, of bijvoorbeeld de in het gebouw aanwezige brandblussers.

Deze paragraaf beschrijft de datastructuur op twee niveaus: Er wordt een beschrijving gegeven van de structurering van thematische informatie, en er wordt een beschrijving gegeven van de geometrische opslagstructuur. Beide beschrijvingen worden daarna samengevat in een geïntegreerd model.

thematische structuur: objecthiërarchie

Een gebouw is onder te verdelen in schachten en verdiepingen, die op hun beurt weer onder te verdelen zijn in aparte ruimtes, zoals kamers. Köninger en Bartel [kon98] stellen voor hun stadsmodel een objectgeoriënteerde structuur voor die, in aangepaste vorm, geschikt is voor toepassing in deze case. Hierbij heeft een gebouw eigenschappen, attributen, en bestaat uit één of meer verdiepingen. Het object 'verdieping' heeft eigenschappen en attributen, en bevat een verwijzing naar het gebouw en naar de ruimtes op de verdieping. Het object 'ruimte' bevat ook eigenschappen en attributen, en verwijzingen naar de verdieping(en) en naar de bouwstenen van de ruimte. In deze bouwstenen wordt de geometrie vastgelegd, en bevatten eigenschappen, attributen en verwijzingen naar de ruimtes waar zij deel van uit maken (afbeelding 3.1). In deze structuur is een onderscheid gemaakt tussen eigenschappen en attributen. Een eigenschap kan bijvoorbeeld de kleur van een gebouw zijn, terwijl een attribuut van een gebouw bijvoorbeeld het adres is.

Door objecten in een objecthiërarchie te rangschikken, worden verschillende abstractieniveaus opgeslagen. In het voorbeeld van het gebouw is het bijvoorbeeld mogelijk om ruimtelijke problemen zowel op gebouwniveau, als op kamerniveau te analyseren. Interactie tussen de verschillende niveaus is ook mogelijk. Zo kan in het voorbeeld als in een kamer brand uitbreekt met een bufferzone berekend worden welke andere kamers door rook beïnvloed worden, maar ook bijvoorbeeld welke verdiepingen geëvacueerd moeten worden.

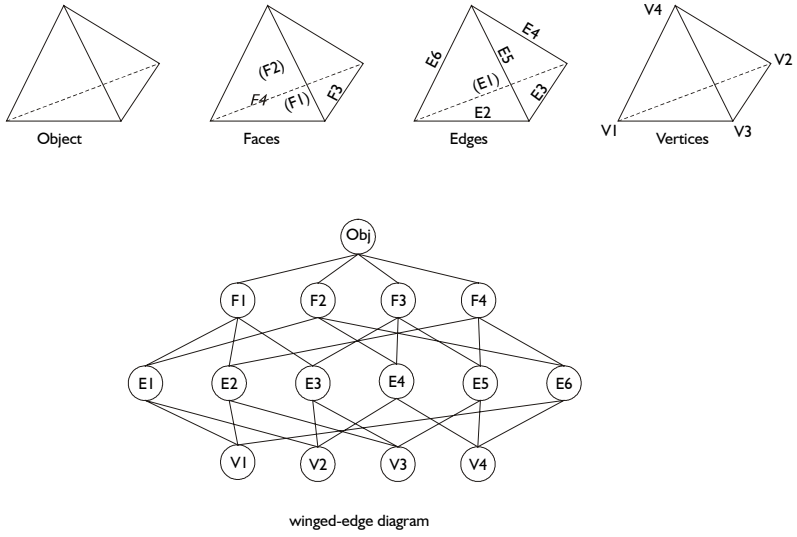


afbeelding 3.1: Objectgeoriënteerde structuur van een gebouw. Een gebouw leent zich om op te slaan in een objectgeoriënteerde structuur, waarbij een gebouw bestaat uit verdiepingen, en een verdieping uit ruimtes. Ruimtes zijn opgebouwd uit bouwstenen, waarin de geometrie vastgelegd is. Elk onderdeel van de structuur is een object met eigenschappen, attributen, en verwijzingen naar het ouderobject (eng.: parent) en de kinderobjecten (eng.: childs).

geometrische structuur: het Tetrahedral Network

Voor het geometrisch model is, conform paragraaf 2.3, een keuze te maken uit de B-Rep, Cell decomposition of de Spatial Occupancy Enumeration. In dit rapport wordt gebruik gemaakt van cell decomposition, waarbij de niet-overlappende cellen slechts bestaan uit irreguliere tetraëders. Deze tetraëders zijn de geometrische beschrijvingen van de hiervoor beschreven bouwstenen. Door het toestaan van slechts tetraëders wordt het grootste bezwaar tegen de cell decomposition, namelijk de veelvormigheid en daardoor lastige analyse, omzeild. De zwakke topologie die bij een cell-decomposition hoort, wordt door de toepassing van slechts tetraëders juist een hele sterke. De verbinding tussen de elementen van een enkele tetraëder zijn makkelijk op te slaan in een winged-edge structuur (afbeelding 3.2), maar ook de connectiviteit tussen verschillende elementen van de totale cell decomposition ligt eenduidig vast; een tetraëder heeft altijd vier naburige tetraëders, twee naburige tetraëders delen exact één vlak, etc.

afbeelding 3.2: Het viervlak en de opslag in een winged-edge diagram. Een viervlak bestaat uit vier vlakken (eng.: faces), zes ribben (eng.: edges) en vier hoekpunten (vertices). Het verband tussen deze elementen is vast te leggen in een winged-edge diagram.



Door de vorming van tetraëders ontstaat een driedimensionaal Triangular Irregular Network (3D-TIN), of Tetrahedral Network (TEN). Varianten die op de tweedimensionale TIN bestaan, bestaan ook voor de TEN. Zo is het mogelijk om een Delaunay Tetrahedral Network te genereren. Aan een tweedimensionale Delaunaytriangulatie wordt de eis gesteld, dat een cirkel door de drie hoekpunten van elke driehoek leeg is, voor een driedimensionale Delaunaytriangulatie geldt de eis, dat een bol door de vier hoekpunten van elke tetraëder leeg is. Aan de constructie van een TEN worden de volgende eisen gesteld (o.a. [ver92]):

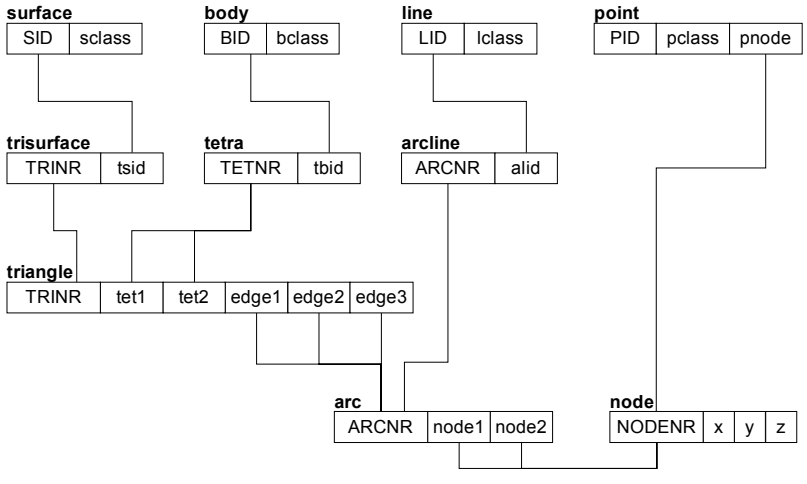
met TEN $T^3\{V\}$:
- De vier hoekpunten $\{v_a, v_b, v_c, v_d\}$ van tetraëder $T^3\{ v_a, v_b, v_c, v_d \}$ liggen niet in één vlak;
- $T^3\{ v_a, v_b, v_c, v_d \}$ bevat geen punten uit $T^3\{V\} - \{v_a, v_b, v_c, v_d\}$;
- Elk driehoeksvlak van een tetraëder ligt op het convexe omhulsel van $T^3\{V\}$, of wordt gedeeld door exact twee tetraëders.

Aan de Delaunay Tetrahedral Network ($DT^3\{V\}$) wordt de additionele eis gesteld:
- De bol door de vier hoekpunten van $T^3\{ v_a, v_b, v_c, v_d \}$ is leeg.

Een TEN kan in principe alleen voor convexe lichamen geconstrueerd worden. Door het omhullende lichaam van een concaaf lichaam te trianguleren, en vervolgens de overbodige, want buiten het lichaam bevindende, tetraëders te verwijderen na triangulatie, kan een TEN van een concaaf lichaam verkregen worden. een voorwaarde hierbij is, dat de grens van het concave lichaam als voorwaarde (eng.: *constraint*) wordt gesteld aan de triangulatie.

Pilouk [pil96] beschrijft een geschikt model om een TEN op te slaan (afbeelding 3.3). Dit model bevat de volgende *features*:
- Body feature: continue set tetraëders,
- Surface feature: continue set driehoeken,
- Line feature: continue set arcs, die de randen van tetraëders zijn;
- Point feature: hoekpunt van ten minste één tetraëder.

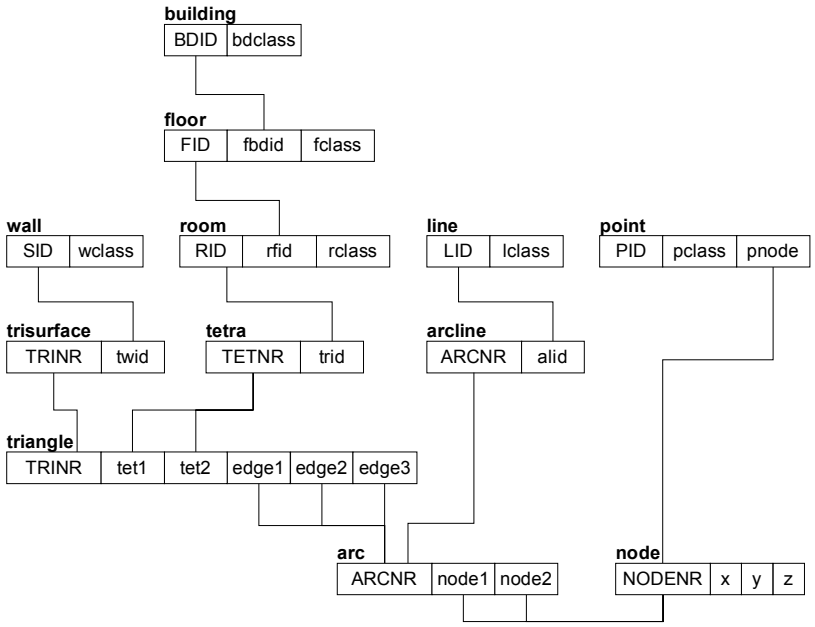
Aan de hand van de nummering van de driehoekszijdes, kan de oriëntatie van deze zijdes vastgelegd worden. Hierdoor kan bepaald worden, door welke twee tetraëders een driehoek gedeeld wordt.



afbeelding 3.3: TEN datamodel. In dit model wordt de TEN opgeslagen door surface, body, line en point features te ontleden in geometrische primitieven. De relatie tussen twee tetraëders wordt vastgelegd aan de hand van de oriëntatie van de driehoeksvlakken van deze tetraëders, in de cellen tet1 en tet2 van de triangletabel. *Afbeelding naar [Pil96].*

geïntegreerd datamodel

Het model van Pilouk [pil96] kan voor gebruik in de case aangepast worden, waardoor een geïntegreerde datastructuur ontstaat (afbeelding 3.4). Hierin wordt de line feature weggelaten, omdat dat overbodige, en voor toepassing in de case onnodige, informatie inhoudt. De surface feature, in het aangepaste model 'wall' genoemd, is wel aanwezig, om een onderscheid te kunnen maken tussen verschillende muurklassen. Een muur kan bijvoorbeeld een buitenmuur of een scheidingswand zijn. Een klasse bevat ook eigenschappen en attributen, zoals materiaal, sleutelbeheerder, bewoner, etc. Er is sprake van een integratie, omdat in het aangepaste model thematische gegevens (de gebouw-verdieping-ruimte hiërarchie) geïntegreerd worden met geometrische gegevens (de TEN).



afbeelding 3.4: Geïntegreerd datamodel. In dit model is de thematische en de geometrische informatie geïntegreerd tot één model.

Tenslotte dienen twee opmerkingen geplaatst worden. Ten eerste wordt het begrip objectoriëntatie in dit rapport vrij ruim geïnterpreteerd. Zo wordt er geen aandacht geschonken aan termen als inkapseling, overerving en polymorphisme. Daarnaast kennen de objecten in dit rapport geen methodes en events. Het in dit rapport gebruikte model kan echter wel gemakkelijk uitgebreid worden om 'echt' objectgeoriënteerd te worden. Ten tweede wordt in de literatuur de benaming 'tetraëder' vaak te ruim geïnterpreteerd. In feite is een tetraëder een van de platonische lichamen, en bestaat uit vier driehoeken met gelijke afmetingen. In een TEN is dit, omdat de cellen irregulier zijn, per definitie niet het geval. Omwille van de consistentie wordt ook in dit rapport de ruime interpretatie gehandhaafd.

Het toepassen van een objectgeoriënteerde structuur gecombineerd met een TEN om de geometrie vast te leggen heeft een aantal voordelen:

- aansluiting op 3D-modeleringsoftware;
- makkelijke representatie bouwstenen in winged-edge;
- uitvoeren van algoritmes op simpele elementen;
- zeer sterke topologie;
- representatie van verschillende abstractieniveaus.

De opslag van driedimensionale gegevens in een geïntegreerde objecthiërarchische structuur in combinatie met een TEN, is zowel voordelig voor de representatie van die gegevens, als voor de bewerking van de gegevens en verdient daardoor de voorkeur voor opslag van gegevens in een 3D-GIS.

4. Tetrahedral Networks in CGAL

In hoofdstuk 3 wordt geconstateerd dat voor een geïntegreerde 3D-GIS de Tetrahedral Network, of TEN, voor visualisatie en bewerking de meest geschikte opslagmethode is. Op basis van de TEN wordt in dit rapport een analysevorm in drie dimensies uitgewerkt en ter demonstratie geprogrammeerd. Daarbij wordt gebruik gemaakt van een programmabibliotheek en de keuzes, die gemaakt worden bij de uitwerking van de analyse, zijn onder meer afhankelijk van de wijze waarop de TEN en de bewerkingen daarop in deze bibliotheek beschreven is.

Voor het onderzoek is gebruik gemaakt van de programmabibliotheek CGAL. Paragraaf 4.1 beschrijft CGAL en motiveert de keuze hiervoor. In feite bestaat CGAL uit een aantal bibliotheken, waarvan in paragraaf 4.2 de voor de TEN belangrijke bibliotheek wordt toegelicht. Voor de beschrijvingen in beide paragrafen wordt gebruik gemaakt van de handleiding van CGAL [sch00].De resultaten moeten ook gevisualiseerd kunnen worden. Als CGAL, zoals voor dit onderzoek het geval was, toegepast wordt in een Windows NT omgeving, dan zijn hier geen directe voorzieningen voor. Paragraaf 4.3 beschrijft een oplossing voor tetrahedral networks.

4.1 CGAL: Computational Geometry Algorithms Library

Voor het programmeren van een analyse op basis van een TEN wordt gebruikt gemaakt van een programmabibliotheek. Deze bibliotheek moet de TEN als datastructuur ondersteunen en hiervoor in algoritmes voorzien. CGAL is zo'n bibliotheek. CGAL staat voor 'Computational Geometry Algorithms Library' en wordt ontwikkeld voor toepassing in geografisch informatiesystemen, visualisatie, simulatie, CAD/CAM, vormanalyse en -reconstructie.

CGAL bestaat uit drie bibliotheken:

- *kernel library:*
  geometrische primitieven en operaties hierop,
- *basic library:*
  geometrische datastructuren en algoritmes,
- *support library:*
  niet-geometrische ondersteuning, zoals bijvoorbeeld visualisatie.

Voor driedimensionale problemen worden in de kernel library de volgende primitieven gedefinieerd: Point_3, Vector_3, Direction_3, Line_3, Ray_3, Segment_3, Plane_3, Triangle_3 en Tetrahedron_3. Bovendien biedt de kernel operaties op de primitieven, zoals oriëntatie, coplanariteit en collineariteit van punten.

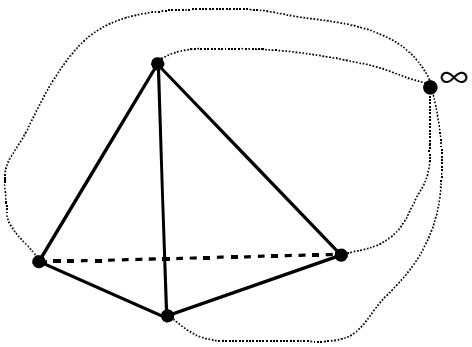
4.2 TEN datastructuur in CGAL

De TEN in CGAL bestaat uit expliciet gegeven cellen en vertices en de relaties hiertussen. Facetten en randen zijn hieruit af te leiden. Een TEN wordt in CGAL een driedimensionale triangulatie genoemd en bestaat uit drie lagen:

- *Triangulation:*
  de geometrische triangulatie van een set punten,
- *Triangulation Data Structure:*
  de relaties tussen cellen en vertices,
- *Cell- en Vertex Base:*
  de elementaire geometrische informatie.

Deze lagen zijn in CGAL in aparte bibliotheken ondergebracht en kunnen naar behoefte worden aangepast, zodat de definitie van een eigen triangulatie mogelijk is.
De triangulation laag is in feite de verdeling van de ruimte in cellen en vertices. Om de hele ruimte, dus ook het exterieur van een object, te kunnen trianguleren wordt een oneindige vertex geïntroduceerd (zie afbeelding 4.1) zonder coördinaten. Elk facet van de triangulatie verbindt daardoor twee tetraëders; als een facet deel uit maakt van de grens van een object dan is een van

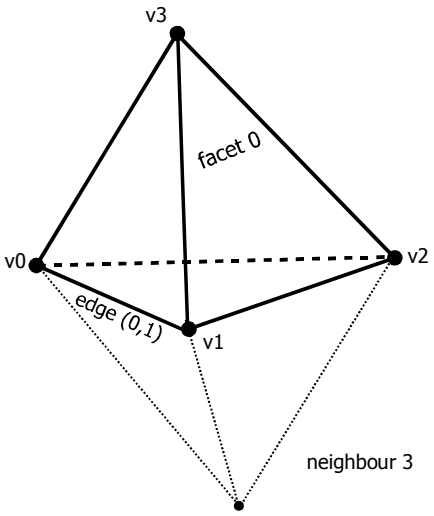
deze tetraëders een oneindige tetraëder waarvan een vertex de oneindige vertex is. Vertices en Cells zijn aan te spreken met behulp van *handles*, en de ten is te traverseren per cell, facet, edge, of vertex met *iterators*. Bij een gegeven edge zijn bovendien alle bijbehorende cellen of facetten te vinden met behulp van *circulators*.



afbeelding 4.1: De oneindige vertex en oneindige tetraëders. Bij triangulatie wordt de hele ruimte getrianguleerd. Om dit mogelijk te maken wordt een oneindige vertex, zonder coördinaten, geïntroduceerd en oneindige tetraëders waarvan een vertex de oneindige vertex is. Elk facet verbindt hierdoor altijd twee cellen; als een facet deel uit maakt van de grens van het getrianguleerde object, dan is een van die cellen een oneindige tetraëder.

De triangulation data structure laag beschrijft de relatie tussen cellen en vertices via de facets (zie afbeelding 4.2). De vertices van de een tetraëder zijn genummerd 0 t/m 3, waarbij de volgorde van vertices 0, 1, 2 het bijbehorende facet oriënteren en waarbij punt 3 aan de positieve zijde van dit facet ligt. De naburige cellen zijn ook genummerd: de naburige cellen tegenover vertex i is wordt ook i genoemd.

afbeelding 4.2: Relaties van de triangulatie in de triangulation data structure laag. Een tetraëder bestaat in CGAL uit vier genummerde vertices, waarbij vertex 3 zich aan de positieve zijde bevind van het facet dat door vertices 0, 1 en 2 wordt opgespannen. De volgorde van de nummering bepaald daarbij de oriëntatie van het facet. Een edge wordt beschreven aan de hand van de twee bijbehorende vertices en een facet aan de hand van de tegenoverliggende vertex. Een tetraëder heeft vier naburige tetraëders; deze worden genummerd aan de hand van de tegenoverliggende vertex.



De cell base en de vertex base beschrijft de geometrie van cellen en vertices aan de hand van primitieven uit de kernel. De cell base en vertex base beschrijven met andere woorden de losse bouwstenen van de triangulatie.

In de programmacode wordt een TEN als type bijvoorbeeld gedefinieerd met:

```
typedef Cartesian<double> Rep;
typedef Triangulation_geom_traits_3<Rep> Gt;
typedef Triangulation_vertex_base_3<Gt> Vb;
typedef Triangulation_cell_base_3<Gt> Cb;
typedef Triangulation_data_structure_3<Vb,Cb> TDS;
typedef Triangulation_3<Gt, TDS> Triangulation;
```

Eerst wordt de datatype, in dit geval double, gedefinieerd op basis waarvan de geometrische primitieven en operaties hierop uit de kernel worden gedefinieerd. De TEN wordt vervolgens gedefinieerd door eerst de Vertex- en de Cell Base te definiëren op basis van de primitieven. De relaties tussen de cellen en de vertices worden vastgelegd in de te definiëren Triangulation Data

Structure. Tenslotte wordt de geometrische triangulatie van de ruimte vastgelegd met behulp van Triangulation. Via deze laag is dus ook interactie mogelijk met geometrische primitieven, vandaar dat definitie van triangulatie op basis van de Triangulation Data Structure is, maar ook op basis van de primitieven en bijbehorende operaties uit de kernel. In de programmacode kan een triangulatie uiteindelijk op de normale manier worden gedeclareerd met:

```
Triangulation triMijnTriangulatie;
```

De topologische elementen van de triangulatie kunnen worden verwerkt in de code door te definiëren:

```
typedef Triangulation::Vertex Vertex;
typedef Triangulation::Cell Cell;
typedef Triangulation::Facet Facet;
typedef Triangulation::Edge Edge;
```

en de geometrische elementen met:

```
typedef Gt::Point Point_3;
typedef Gt::Segment Segment_3;
typedef Gt::Triangle Triangle_3;
```

De geometrische elementen worden daarbij uit de kernel gedefinieerd.
Tijdens de verwerking van de TEN worden cellen en vertices geïdentificeerd met zogenoemde 'Handles', waaraan deze elementen als het ware worden beetgepakt:

```
typedef Triangulation::Cell_handle Cell_handle;
typedef Triangulation::Vertex_handle Vertex_handle;
```

Ten slotte kan de elke edge, facet en elke vertex in de TEN worden bezocht met *iterators*, en kan elke bij een bepaalde edge behorende facet worden bezocht met een *circulator*:

```
typedef Triangulation::Edge_iterator Edge_iterator;
typedef Triangulation::Vertex_iterator Vertex_iterator;
typedef Triangulation::Facet_iterator  Facet_iterator;
typedef Triangulation::Facet_circulator Facet_circulator;
```

Circulators en Iterators komen voort uit de support library.

4.3 visualisatie van de datastructuur

Voor visualisatie van driedimensionale gegevens kan in CGAL gebruik gemaakt worden van Geomview. Geomview is een aparte bibliotheek dat echter niet beschikbaar is voor Windows NT. Voor dit onderzoek wordt gebruik gemaakt van CGAL op een Windows platform, waardoor een andere manier van visualiseren toegepast is. Er is voor gekozen om de uitvoer ten bate van visualisatie op te slaan in het RAW-formaat. Dit is een formaat waarin per regel de coördinaten van de drie hoekpunten van een driehoek worden weggeschreven. Bijvoorbeeld voor hoekpunten A, B, en C:

```
Ax Ay Az Bx By Bz Cx Cy Cz
```

Door de simpele structuur is het RAW-formaat een formaat dat zich leent voor conversie naar diverse andere 3D-formaten, zoals bijvoorbeeld VRML. Omdat vrijwel alle 3D-formaten maar één zijde van een driehoek tonen, moet elke driehoek twee keer, met tegengestelde oriëntatie, worden opgeslagen. Voor dit onderzoek is voor de conversie gebruik gemaakt van een vrij verkrijgbaar conversieprogramma Crossroads 3D v1.0. Voor de visualisatie van VRML-bestanden is Cortona VRML gebruikt.

5. Een driedimensionaal analysegereedschap: buffering in 3D

In dit hoofdstuk wordt een analysevorm, buffering, uitgewerkt in 3D. Buffering is een van de weinige analysevormen die voor toepassing in een driedimensionale omgeving een volledig andere benadering behoeft, en deze analysevorm sluit aan op de case, waarbij ten behoeve van de brandweer bijvoorbeeld rookontwikkeling gesimuleerd moet worden.

Worboys beschrijft het bufferen als 'het formeren van gebieden die locaties bevatten binnen een gegeven bereik van een gegeven set features'. Bufferzones zijn cirkelvormig of rechthoekig rond punten, en bestaan uit een uit corridors van constante breedte langs lijnen en oppervlakken [wor95, p.344]. De buffer rond een polygoon is in vorm een vergrootte versie van het gebufferde object [bur98, p.179]. Bij een dergelijke buffer ontstaat bij de hoeken een afstandsfout, de buffer is daar te groot en in een bufferoverlay kunnen daardoor teveel objecten op basis van de buffer geselecteerd worden. Een alternatieve methode is om een bij de hoeken afgeronde buffer te benaderen, maar ook hier ontstaat dan een afstandsfout door de benadering. De buffer is nu echter te klein. Het genereren van een zuiver afgeronde buffer is vaak niet mogelijk, omdat dan de opslag van cirkelbogen nodig is en dat wordt niet door elke datastructuur toegelaten.

Een driedimensionale buffer bestaat uit een volume dat locaties bevat binnen een gegeven bereik van gegeven objecten. Een afstandsfout treedt niet alleen op bij de hoekpunten, maar ook bij de randen van de buffer. Een driedimensionale buffer dat een vergrootte versie is van het gebufferde object, zal daarom, afhankelijk van de toepassing van de buffer, niet geschikt zijn. Ook de opslag van gekromde oppervlakken, bij een driedimensionale buffer zijn dat delen van bollen, is alleen mogelijk als gebruik wordt gemaakt van een analytische geometrische representatie. In dit hoofdstuk wordt daarom een driedimensionale buffer geschreven waarbij de hoeken en de randen afgerond zijn en daardoor een beter resultaat geven.

De benaderd afgeronde buffer wordt ook in twee dimensies toegepast. Daarom beschrijft paragraaf 5.1, na een kort overzicht van soorten buffers, de benaderde buffer in twee dimensies, en de benaderingsfout die bij toepassing van een dergelijke buffer optreedt. Op basis hiervan wordt in paragraaf 5.2 de afgeronde buffer uitgewerkt voor toepassing in drie dimensies. Hiertoe wordt een algoritme gegeven om een -simpele- benaderd afgeronde 3D-Buffer te genereren voor een object dat gemodelleerd is als een Tetrahedral Network.

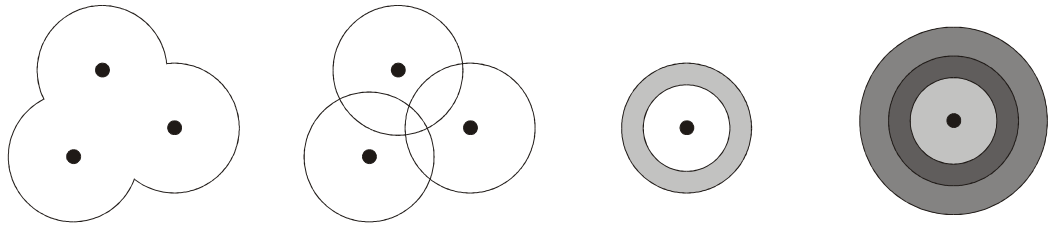
5.1 buffers in twee dimensies

Er bestaan verschillende soorten buffers en de keuze daaruit is afhankelijk van het doel van de bufferanalyse en het soort object dat gebufferd wordt. De paragraaf begint daarom met een overzicht van soorten buffers. De buffers in dat overzicht hebben gemeen dat zij ontstaan door een corridor langs het te bufferen object te leggen en het resultaat eventueel af te ronden, waardoor afrondingsfouten ontstaan. Een dergelijke fout heeft bij een vertaling naar drie dimensies een grotere invloed omdat een afrondingsfout zowel bij de hoeken als bij de randen van een buffer optreedt. Deze paragraaf geeft daarom een beschrijving van de opbouw van een afgeronde buffer en een maat voor de afrondingsfout. Daarnaast worden alternatieven beschreven om deze fout op te vangen.

soorten buffers

In een 2D-GIS kunnen punt-, lijn- en vlakelementen voorkomen, en voor elke soort bestaan verschillende soorten buffers. De volgende indeling van buffervarianten wordt gehanteerd in het GIS-pakket GeoMedia van Intergraph [lim98], en geeft een volledig overzicht:

- Puntelementen (zie afbeelding 5.1):
  - single merged:
    is een enkele cirkel per te bufferen puntelement, deze cirkels zijn samengevoegd tot één buffer.
  - single unmerged:
    is een enkele cirkel per te bufferen puntelement, deze cirkels zijn niet samengevoegd.
  - multiple-distance ('donuts'):
    wordt gevormd door een buffer met een bepaalde bufferafstand af te trekken van een andere buffer met een grotere bufferafstand, waardoor een ring als buffer ontstaat,
  - multiple-stack:
    is een samenstel van meerdere buffers met verschillende bufferafstanden.



Afbeelding 5.1: Buffers om puntelementen. Buffers rond puntelementen zijn onder te verdelen in single en in multiple buffers. Single buffers worden gevormd door per te bufferen puntelement een cirkel te vormen. Deze kunnen samengevoegd worden tot één buffer (a), of apart gehouden worden (b). Een multiple buffer kan een ring zijn, ontstaan door het verschil van twee single buffers (c), of een aantal buffers op elkaar gestapeld (d).

- Lijnelementen:
  - square (un)merged:
    ontstaat door een corridor met vaste breedte langs de lijnelementen te leggen, waardoor een vlak gevormd wordt. De buffers kunnen naar keuze samengevoegd worden.
  - rounded (un)merged:
    is een buffer om een lijnelement dat aan de uiteinden afgerond is, waardoor de buffer overal een gelijke bufferafstand heeft.

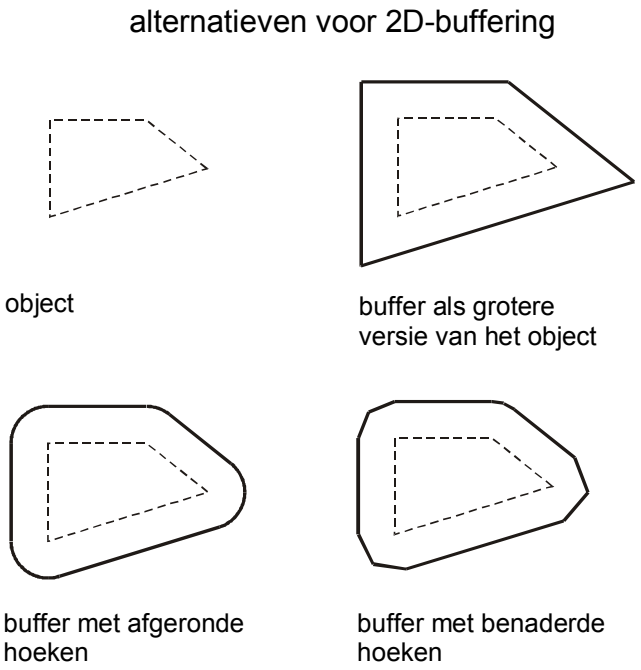
- Vlakelementen:
  - outside (un)merged ('vergroting'):
    ontstaat door een corridor met een vaste breedte langs de buitenkant van het element te leggen. De buffer is het oppervlak dat omsloten wordt dor de buitengrens van die corridor en is in feite een vergrootte versie van het element. Deze buffers kunnen samengevoegd worden tot één enkele buffer.
  - inside (un)merged ('verkleining'):
    is een buffer dat ontstaat door een buffer met een vaste breedte langs de binnenkant van het element te leggen, waardoor objecten binnenin het te bufferen vlak geselecteerd kunnen worden.

Samengevat bestaan buffers uit een vlak dat ontstaat door een corridor met constante breedte langs de te bufferen objecten te leggen. Als er meerdere objecten gebufferd worden, dan kunnen de resulterende buffers samengevoegd (*merged)* worden tot één buffer. Bovendien kan een buffer afgerond worden zodat de afstand tussen het object en de buffer overal even groot is. Deze afronding is echter een benadering, waardoor afrondingsfouten ontstaan. Een maat voor deze fout is echter niet gegeven, en de gebruiker kan hier ook geen invloed op uitoefenen.

verbeterde buffers: vier varianten

Het is belangrijk dat de afstand tussen de buffer en het gebufferde object overal gelijk is aan de gewenste bufferafstand. Een buffer dat een vergrootte versie is van het gebufferde object kent een fout bij de hoeken; de afstand tussen een hoekpunt van de buffer tot het corresponderende hoekpunt van het originele object is dan groter dan de gewenste bufferafstand. Als gevolg hiervan bevinden zich bij een bufferoverlay te veel objecten, of een te groot deel daarvan, binnen de bufferzone. Dit probleem zou opgevangen kunnen worden door de buffer af te ronden, waardoor de afstand tussen het gebufferde object en de buffer overal gelijk is aan de gewenste bufferafstand. Dit alternatief is echter niet toepasbaar, omdat de buffer bij de hoekpunten dan bestaat uit krommen, hetgeen onwenselijk is als de buffer opgeslagen wordt in een zelfde datastructuur als het gebufferde object en deze datastructuur de opslag van krommen niet toestaat. Een ander alternatief is het benaderen van een afgeronde buffer door extra hoekpunten toe te voegen. Dit alternatief verenigt de voordelen van beide voorgaande oplossingen, omdat de opslag in een zelfde datastructuur plaats kan vinden, en omdat de afstand tussen het object en de buffer overal zoveel mogelijk aan de gewenste bufferafstand gelijk is (zie afbeelding 5.2). De ontstane buffer is nu wel kleiner dan de ideale, afgeronde, buffer, maar het afstandsverschil is kleiner dan bij een buffer dat een vergrote versie van het gebufferde object is.

Afbeelding 5.2: Alternatieven voor 2D-buffering. Een simpele buffer is een vergrote versie van het gebufferde object, waardoor bij de hoekpunten fouten ontstaan. De buffer is te groot: dit kan opgelost worden door de hoekpunten af te ronden. Daarbij worden er echter krommen gegenereerd, waardoor de buffer eventueel in een andere datastructuur opgeslagen moet worden dan het gebufferde object. Een compromis wordt bereikt als een afgeronde buffer benaderd wordt door extra vertices toe te voegen.



Een benaderd afgeronde buffer ontstaat door elke vertex van het te bufferen object een aantal keer te verplaatsen. Een vertex is het begin- of eindpunt van twee randen van een object, dus per objectvertex worden twee buffervertices gegenereerd, loodrecht op de randen en met een afstand gelijk aan de gewenste bufferafstand. De hoek tussen de twee ontstane verplaatsingsvectoren bepaalt de positie van één of meerdere tussenpunten die de afronding benaderen.

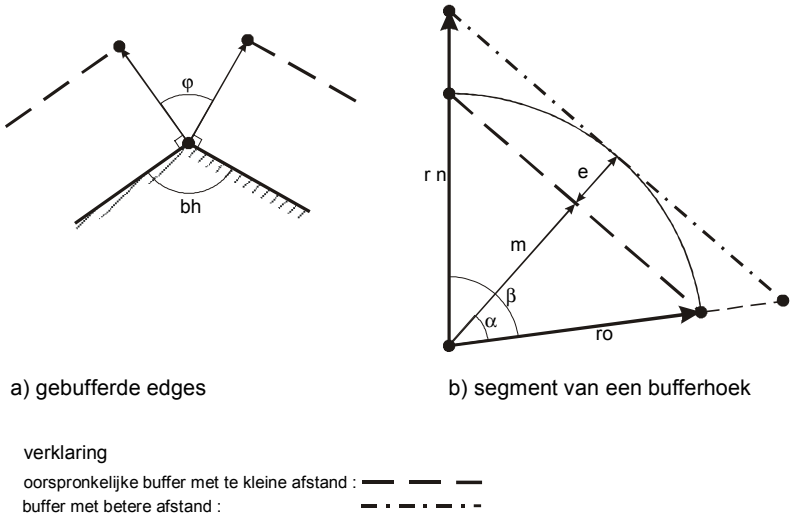
Doordat een dergelijke buffer een benadering is, blijven er nog steeds fouten optreden. De gegenereerde buffer is nu echter te klein in plaats van te groot en de fout is kleiner dan de fout bij een buffer dat een vergroting van het gebufferde object is. De benaderingsfout is afhankelijk van het aantal buffervertices dat bij een hoek ingevoegd wordt. Als er meer vertices worden berekend om de cirkelboog te benaderen, wordt de fout kleiner. De afstandsfout hangt daarnaast af van de grootte van de hoek tussen de twee randen die bij een vertex samenkomen, en de schaal van de toepassing. Om er zorg voor te dragen dat alle, op basis van de buffer, te selecteren objecten bij een bufferoverlayoperatie daadwerkelijk binnen de buffer liggen, moet de bufferafstand gecorrigeerd worden voor de afstandsfout.

Er zijn vier mogelijkheden om een afgeronde buffer te benaderen. Ten eerste kan een normale benaderd afgeronde buffer gegenereerd worden, waarbij de gemaakte fout klein genoeg geacht wordt, of wordt verkleind door het toevoegen van extra tussenpunten. Het toevoegen van extra tussenpunten kan per vertex en al tijdens het genereren van de buffer gebeuren. Ten tweede kan de gewenste bufferafstand gecorrigeerd worden, zodat de hele buffer vergroot wordt en het zeker is dat alle te selecteren objecten daadwerkelijk geselecteerd worden. Ten derde kan een buffer gegenereerd worden waarbij de bufferafstand alleen bij de hoeken gecorrigeerd wordt. Ten vierde kan een buffer gegenereerd worden op een zelfde manier als het derde alternatief, maar waarbij de grootst mogelijke fout zo klein mogelijk gemaakt wordt. Hieronder worden deze vier alternatieven toegelicht.

alternatief I: normale benaderde afgeronde buffer

Het eerste alternatief is het genereren van een benaderd afgeronde buffer met de gewenste bufferafstand (afbeelding 5.4a). De fout is te berekenen met behulp van de volgende variabelen (zie afbeelding 5.3):

r_n:	oorspronkelijke, door de gebruiker gewenste, bufferafstand,		
m:	kleinst gegenereerde afstand,		
tp:	het aantal tussenpunten dat gebruikt wordt om de cirkelboog te benaderen,		
e:	afstandsfout;	$e = r_n - m$,	
bh:	de hoek van het te bufferen object bij een vertex;		$0 < bh < \pi$
φ:	door tussenpunten te verdelen hoek;	$\varphi = \pi - bh$,	$0 < \varphi < \pi$,
β:	hoek van een door tussenpunten gevormd segment;	$\beta = \varphi / (tp + 1)$,	$0 < \beta < \pi$,
α:	de helft van β;	$\alpha = ½\beta$,	$0 < \alpha < ½\pi$,



afbeelding 5.3: Correctie op de bufferafstand. Bij benadering van een afgeronde buffer wordt een fout e gemaakt dat per hoek verschilt, en afhankelijk is van de grootte van de hoek bh tussen de bufferranden, het aantal tussenpunten en de schaal van de toepassing. De grootste gemaakte fout is het verschil tussen de gewenste bufferafstand r_o en de kleinst mogelijke afstand m die door de afronding gegenereerd word, en kan gecorrigeerd worden door een nieuwe bufferafstand r_n te berekenen.

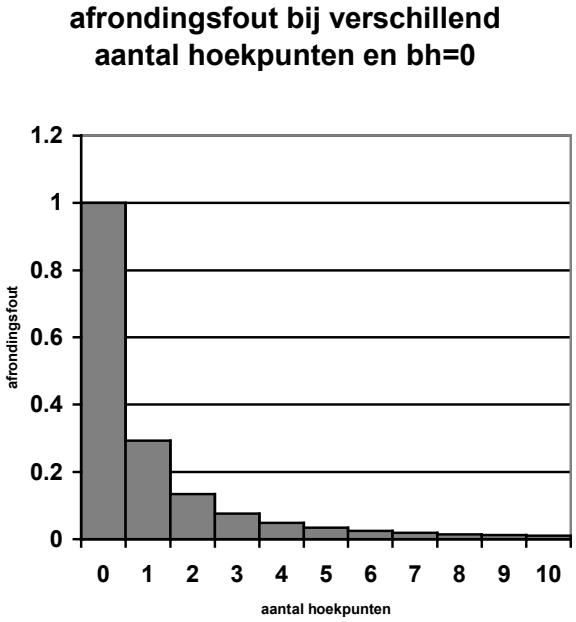
Met $m = r_o \cos \alpha$ is de fout e als functie van tp, bh, en r_o te berekenen als:

$$\mathbf{e(tp, bh, r_o)} = r_o - m = \mathbf{r_o(1 - \cos((\pi - bh) / (2tp + 2)))} .$$

De grootst mogelijke fout treedt op als de hoek van het object bh naar 0 gaat. Het effect van het toevoegen van extra hoekpunten op de fout neemt snel af (grafiek 5.1), Als één tussenpunt gebruikt wordt om de cirkelboog te benaderen dan is de grootste fout ongeveer een derde van de gewenste bufferafstand, bij twee tussenpunten is die fout ongeveer een tiende van de gewenste bufferafstand. Een beoordeling van dat effect kan echter pas gegeven kan worden als de schaal van de toepassing bekend is.

Het voordeel van dit alternatief is, dat het algoritme simpel en snel is terwijl het nadeel is, dat de afstand tussen de grens van de buffer en het gebufferde object kleiner zijn dan de gewenste bufferafstand en dat daardoor objecten of delen daarvan onterecht niet door de buffer worden geselecteerd.

grafiek 5.1: Afrondingsfout bij benadering van een afgeronde buffer. In de grafiek wordt aangegeven hoe groot de fout ten opzichte van de gewenste bufferstraal is, bij een bepaald aantal tussenpunten en bij een zo klein mogelijke hoek in het gebufferde object (bh=0). Als er bijvoorbeeld geen tussenpunten worden gebruikt, dan is de grootste fout natuurlijk gelijk aan de bufferafstand zelf, en bij één tussenpunt is de fout ongeveer 1/3 van de bufferafstand.



alternatief II: correctie op de bufferafstand

Om te verzekeren dat alle te selecteren objecten op basis van de te genereren buffer daadwerkelijk geselecteerd worden, mag de buffer niet te klein zijn en moet de bufferafstand gecorrigeerd worden. Een mogelijkheid is om de cirkelbogen met voldoende tussenpunten te benaderen, zodat de fout klein genoeg is, een andere mogelijkheid is er voor te zorgen dat de buffer groot genoeg is door een nieuwe, grotere, bufferafstand te berekenen en deze nieuwe bufferafstand toe te passen op de hele buffer (afbeelding 5.4b).

De nieuwe bufferafstand r_n is te berekenen met:

$$\mathbf{r_n = r_o / (\cos \alpha)}$$

Of, als functie van het aantal tussenpunten tp, de hoek van het object bh, en de oorspronkelijke bufferafstand r_o, met:

$$\begin{aligned} \mathbf{r_n(tp, bh, r_o)} &= r_o / \cos \tfrac{1}{2}\beta \\ &= r_o / \cos \varphi / (2tp + 2)) \\ &= \mathbf{r_o / (\cos((\pi - bh) / (2tp + 2)))} \; . \end{aligned}$$

Als een nieuwe bufferafstand voor de gehele buffer berekend moet worden, dan moet uitgegaan worden van de grootste fout die bij alternatief I zou ontstaan, en dus van de kleinste hoek bh. Als bh $\rightarrow$ 0 is de bufferafstand te berekenen met: r_n (tp, 0, r_o) = r_o / (cos (π / 2tp + 2)). Als bijvoorbeeld 1 tussenpunt wordt toegepast, dan geldt: $r_n(1, 0, r_o) = r_o / \cos ¼\pi \cong 1{,}4\ r_o$.

Voordelen van alternatief II zijn dat slechts een kleine aanpassing van alternatief I nodig is, en dat daadwerkelijk alle objecten in een bufferoverlay gevonden. Een nadeel van alternatief II is dat de corridor van de buffer ook bij de randen breder is dan de gewenste bufferafstand. Bovendien geldt de nieuwe bufferafstand voor de hele buffer en is afhankelijk van het aantal tussenpunten, dat daardoor van tevoren bekend en voor het hele object gelijk moet zijn.

alternatief III: slechts correctie van de bufferafstand bij de hoeken

Alternatief II verzekert dat de buffer groot genoeg is. De berekende bufferafstand geldt echter voor het gehele lichaam, dus ook voor de randen. Deze vergroting is bij de randen in feite overbodig, omdat daar bij alternatief I geen fout optreedt, en maakt de buffer onnodig groot, terwijl dat juist de reden was om een benaderd afgeronde buffer te berekenen. Een beter alternatief is, indien de buffer niet te klein mag zijn, om een buffer te genereren met twee bufferafstanden; de oorspronkelijke afstand van alternatief I wordt toegepast voor de randen, terwijl de hoeken gebufferd worden met een grotere, gecorrigeerde afstand, die te berekenen is als bij alternatief II. Het algoritme wordt wel complexer, omdat de vertices, behorende bij de randen van de buffer, niet loodrecht op de objectranden worden verplaatst, terwijl de afstand van de verplaatsing wel loodrecht bepaald wordt (zie afbeelding 5.4c).

Het voordeel van dit alternatief is, dat de grootste fout weliswaar even groot is als bij alternatief II, maar alleen optreedt bij de hoeken. Bovendien kan de alternatieve bufferafstand per vertex bepaald worden, en daardoor ook het aantal tussenpunten per vertex. Een nadeel is, dat het door de ongelijkmatige verdeling van fouten lastig is om een maat hiervoor te geven.

alternatief IV: correctie bij de hoeken en een minimale maximale fout

Alternatief III verkleint de totale fout die gemaakt wordt bij alternatief II, en is in feite een vervolg daarop. Alternatief IV is een vervolg op alternatief III en verkleint de fout verder. Zoals in afbeelding 5.4c te zien is, is de grootste fout die optreedt gelijk aan $r_n - r_o$, maar de gemaakte fout is echter niet overal even groot, en is ongelijk verdeeld. Door een kleine aanpassing is de grootst gemaakte fout zo klein mogelijk te maken (zie afbeelding 5.4d). Hierbij is de hoek tussen een lijn loodrecht op de rand van een object, en een lijn door het bijbehorende buffervertex en het hoekpunt, gelijk aan de helft van een segmenthoek. Met deze hoek is een kleinere gecorrigeerde bufferafstand voor de hoeken, r_a, te bepalen.

In vergelijking met alternatief twee wordt hoek φ afhankelijk van het aantal tussenpunten tp, nu niet over 2tp +2, maar over 2tp+4 halve segmenten verdeeld. De uiteindelijke bufferafstand is nu te berekenen met α = de hoek van een halve segment, bh = de hoek van het te bufferen object, r_o = de gewenste bufferafstand, en r_a=de berekende bufferafstand, als:

$$\mathbf{r_a} = r_o / (\cos \alpha) = \mathbf{r_o / (\cos((\pi\text{-bh}) / (2tp+4)))} .$$

Het volgende algoritme laat zien hoe de bufferpunten gevonden kunnen worden.

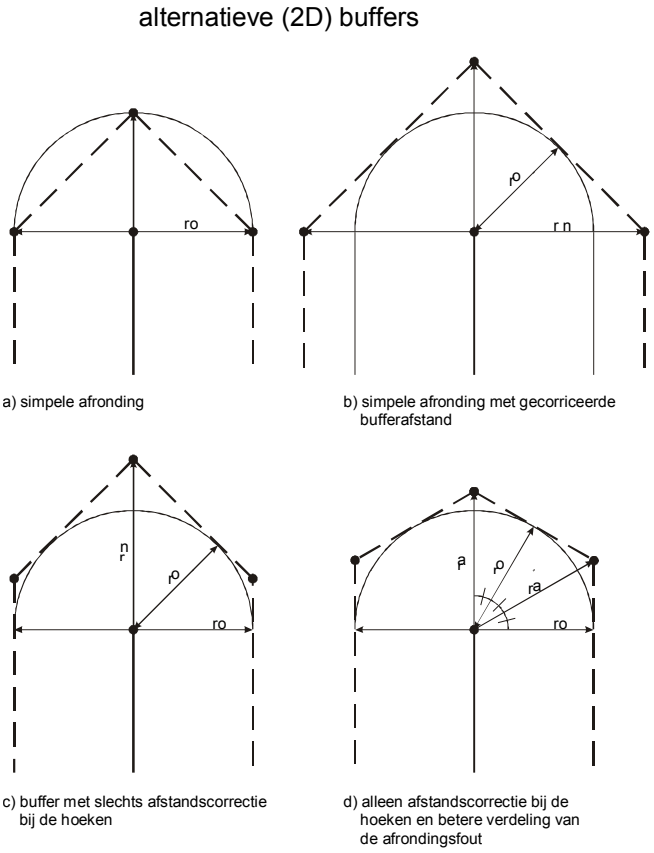
Gegeven: r_o = gewenste bufferafstand
X_i = coordinaten van objectvertex i
e = maximaal toegestane fout ra – ro

1. bepaal op basis van e en ro, bij welke hoeken hoeveel tussenpunten gebruikt moeten worden
2. zoek de coordinaten van de andere vertices die bij de incidentele randen van i horen: X_{i-1} en X_{i+1}
3. bereken de binnenhoek tussen de edges: bh
4. bepaal op basis van stap 1 en bh hoeveel tussenpunten gebruikt worden:tp
5. bereken α(tp, bh) en $r_a(\alpha, ro)$
6. bereken de positie van het 1e bufferpunt: $bp_1(X_i, X_{i+1}, \alpha, r_a)$
7. doe (tp+1) keer: bepaal het volgende bufferpunt j: $bp_j(X_i, 2\alpha, r_a, bp_{j-1})$

Voor het uitvoeren van het algoritme kan van tevoren bepaald worden hoe groot de maximale afwijking tussen de gegenereerde buffer en de gewenste buffer mag zijn en op basis hiervan wordt in stap 1 van tevoren bepaald bij welke hoekgroottes hoeveel tussenpunten nodig zijn. De bepaling hiervan kan met:

$$e = r_a - r_0 = r_o(\cos^{-1}\alpha - 1);$$
$$tp = ((\pi - bh) / (2 \text{ acos}(r_o/e))) - 2$$

De belangrijkste voordelen van dit alternatief zijn dat de grootst mogelijke fout zo klein mogelijk is, dat de buffer niet te klein is, en dat de bufferafstand en de straal per vertex te bepalen is.



afbeelding 5.4: Alternatieve 2D-buffers bij een zo klein mogelijke bufferhoek. Er zijn vier alternatieve buffers te onderscheiden. Afgebeeld zijn deze alternatieven bij een bufferhoek dat gelijk aan nul is, en waar dus de fout het grootst is. Een afgeronde buffer kan benaderd worden door tussenpunten toe te voegen (a). Hierdoor ontstaat een buffer dat te klein is, hetgeen opgevangen kan worden door de bufferafstand te corrigeren (b). De buffer wordt bij de randen hierdoor onnodig groot, zodat hier eventueel de oorspronkelijke bufferafstand gebruikt kan worden (c). Ten slotte kan de grootste fout zo klein mogelijk gemaakt worden (d).

5.2 buffers in drie dimensies

Als een 3D-GIS toegepast wordt voor relatief kleinschalige toepassingen, zal een tweedimensionale buffer voor de meeste vraagstukken voldoende zijn. De hoogtecomponent is in verhouding tot de planimetrische component dan namelijk relatief klein. In een grootschalige toepassing is de hoogtecomponent wel van belang en zal het gebruik van een conventionele, tweedimensionale, buffer niet volstaan en moet een driedimensionale buffer gebruikt worden. Zijn veel GIS-analysegereedschappen conform hoofdstuk 2 vrij gemakkelijk te vertalen voor gebruik in een 3D-omgeving, de buffer vereist meer inspanning, omdat de verschijningsvorm van de buffer verandert. Met andere woorden; een buffer in 2D is een vlak, in 3D is de buffer een volume.

Deze paragraaf beschrijft de voordelen van een TEN voor 3D-buffers, waarna de selectie van objecten op basis van een 3D-buffer verder uitgewerkt wordt. Daarop volgt een beschrijving van de constructie van een driedimensionale benaderd afgeronde buffer op basis van het Tetrahedral Network wordt. Ten slotte volgt een beschrijving van een praktijktest.

de voordelen van een buffer als Tetrahedral Network

De voordelen van een TEN als datastructuur voor toepassing in een driedimensionale GIS zijn in hoofdstuk 3 aangegeven, maar ook de TEN als basis voor GIS-analyses is een geschikte keuze. Een TEN heeft een sterke topologie, maar ook het feit dat een object in cellen, de tetraëders, wordt opgedeeld, is met name voor het genereren en de verdere verwerking van de driedimensionale buffer een groot voordeel.

De constructie van een buffer is een bewerking op de grens van een object. Een vlak wordt omgrensd door een polygoon en in twee dimensies wordt een buffer van een vlak bepaald door een corridor langs die polygoon te plaatsen. Een volume in drie dimensies wordt begrensd door een continue verzameling vlakken en een driedimensionale buffer bevat alle locaties die zich binnen een gegeven afstand tot de grens van het gebufferde object bevinden. Het bepalen van de grens van een object dat gemodelleerd is in een TEN, is een simpele bewerking. Een vlak in een TEN maakt alleen deel uit van de grens van het object, als een van de twee bijbehorende tetraëders bij het interieur van het lichaam hoort, en de andere tetraëder bij het exterieur. De grens van het object is door de sterke topologische structuur van een TEN zo gemakkelijk te vinden.

Een buffer wordt, nadat het geconstrueerd is, gebruikt om objecten uit de GIS te selecteren met een bufferoverlay. Het resultaat van een bufferoverlay is een verzameling van alle objecten of delen daarvan, uit een bepaalde laag, die zich binnen een gegeven reikwijdte, de bufferafstand, van het gebufferde object of gebufferde objecten bevinden. Een laag is daarbij een verzameling objecten met dezelfde kenmerken; in een model van een stad kunnen bijvoorbeeld een huizenlaag, een wegenlaag, een openbaar-gebied-laag, etc. gedefinieerd worden. De dimensie van kaartlaag is ondanks zijn naam niet per definitie beperkt tot twee dimensies. Voor elk object moet beslist worden, of deze zich binnen de bufferzone bevindt. Van een object dat zich geheel binnen of geheel buiten de bufferzone bevindt is dat duidelijk, maar voor een object dat zich gedeeltelijk binnen en gedeeltelijk buiten de bufferzone bevindt, moet beslist worden of deze geselecteerd of niet wordt, en als selectie plaats vindt, of het object in zijn geheel wordt geselecteerd of alleen het gedeelte van het object dat zich binnen de bufferzone bevindt. In het laatste geval wordt het object ‘doorgeknipt’ (eng.: *clipped*).

De TEN is geschikt voor zowel het bepalen of een object zich in de bufferzone bevindt, als voor het bepalen van welk gedeelte van een object zich binnen die bufferzone bevindt. Voor het bepalen van de objecten die zich in de bufferzone bevinden, kunnen Virtual-Realitytechnieken gebruikt worden, waarmee botsingen gedetecteerd worden (eng.: *collision detection*). Het probleem is hetzelfde: als een object zich geheel of gedeeltelijk in een ander object bevindt, is er sprake van een botsing. Als een object zich geheel of gedeeltelijk binnen een (3D-)buffer bevindt,

is er sprake van overlap. Door het bufferlichaam in te sluiten in een blok (eng.: *bounding box*) kan, door slechts coördinaten te vergelijken, een grove scheiding gemaakt worden tussen potentieel te selecteren objecten en objecten die definitief buiten de bounding box en dus buiten de bufferzone vallen. Voor het bepalen van een bounding box van een lichaam zijn de maximale coördinaten van dat lichaam nodig. Van de objecten die overblijven moet exact bepaald worden of deze zich geheel, gedeeltelijk of niet binnen de bufferzone bevinden.

Als alle objecten in TEN's opgeslagen zijn, kan per object bepaald worden of deze zich binnen de bufferzone bevindt, door dit object te doorlopen, of te *traverseren*, op tetraëderniveau en per tetraëder te bepalen of deze disjunct van de buffer is, met andere woorden: of deze zich in het geheel in het exterieur van de buffer bevindt. Als ook objecten geselecteerd moeten worden die zich gedeeltelijk binnen de buffer bevinden, dan hoeft niet van elke tetraëder bepaald te worden of deze los staat van de buffer; als van een tetraëder van een object berekend is dat deze zich geheel of gedeeltelijk in de buffer bevindt, dan geldt dat gelijk voor het hele object waar deze tetraëder een cel van is.

selectie van tetraëders op basis van topologie

Een object verdeelt de ruimte in drie subruimtes; een object bestaat uit zijn interieur, zijn exterieur en een grens die het interieur van het exterieur scheidt. Een dergelijke topologische benadering van ruimte wordt *point-set topology* genoemd, omdat de subruimten in verzamelingen punten zijn waarvoor geldt dat een punt dat deel uitmaakt van een bepaalde subruimte, geen deel uitmaakt van een andere subruimte. Een punt in het interieur van een object bevindt zich bijvoorbeeld niet tegelijk in het exterieur van een object. Verder wordt onderscheid gemaakt tussen open en gesloten verzamelingen, en een grens als *boundary* of als *frontier* [mol98]. Deze termen zijn voor het volgende niet van belang, omdat daar alleen de topologische relatie tussen twee tetraëders wordt beschreven.

De bepaling of een object zich binnen de buffer bevindt, komt neer op de bepaling van de topologische relatie tussen tetraëders. Per paar tetraëders moet bepaald worden of de twee tetraëders disjunct zijn of niet. Als twee tetraëders niet disjunct zijn dan geldt dat ook voor de buffer en voor het object waar de tetraëder deel van uitmaken. In paragraaf 2.2 is al aangegeven dat er vijf topologische relaties zijn te onderscheiden in drie dimensies: touch, in, cross, overlap en disjoint. Voor de bepaling hiervan worden doorsneden bepaald tussen de interieuren van de objecten en de interieuren verenigd met de grenzen. Deze beschrijving geldt voor relaties tussen objecten van verschillende dimensies, terwijl voor de selectie van objecten door een buffer slechts de relatie tussen alleen tetraëders bepaald hoeft te worden. Pigot [pig92] geeft een beschrijving van zes mogelijke topologische relaties tussen twee tetraëders, v_1 en v_2, door de relatie te bepalen van de grens ∂v_1, het interieur v_1° en het exterieur $\bar{v}_1$ van v_1 met de grens ∂v_2 van v_2. Met andere woorden: bepaald wordt of de volgende stellingen waar of onwaar zijn:

- E: $\partial v_2 \cap \bar{v}_1 \neq \emptyset$; 'de grens van v_1 maakt deel uit van het exterieur van v_2',
- B: $\partial v_2 \cap \partial v_1 \neq \emptyset$; 'de grens van v_1 maakt deel uit van de grens van v_2',
- I: $\partial v_2 \cap v_1^\circ \neq \emptyset$; 'de grens van v_1 maakt deel uit van het interieur van v_2'.

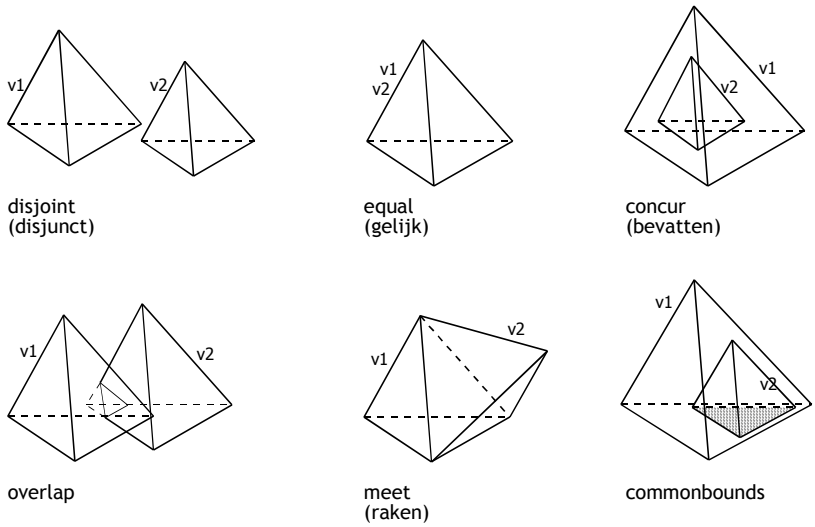
Er zijn in principe 8 mogelijkheden (E,B,I) (zie tabel 5.1),maar de uitkomst (0, 0, 0) is, althans in dezelfde dimensie, ongeldig en de uitkomst (1, 0, 1) is onmogelijk omdat beide tetraëders gesloten volumes zijn en de elementen daarvan onderling verbonden.

stellingen			relatie
E: $\partial v_2 \cap \bar{v}_1 \neq \varnothing$	B: $\partial v_2 \cap \partial v_1 \neq \varnothing$	I: $\partial v_2 \cap v_1^\circ \neq \varnothing$	
0	*0*	*0*	*ongeldig*
0	0	1	v1 bevat v2
0	1	0	v1 is gelijk aan v2
0	1	1	v1 bevat en raakt v2
1	0	0	v1 en v2 zijn disjunct
1	*0*	*1*	*onmogelijk*
1	1	0	v1 raakt v2
1	1	1	v1 overlapt v2

tabel 5.1: Relaties tussen twee volumes. Door van twee volumes v_1 en v_2 een vergelijking te maken tussen enerzijds de grens van v_2 en anderzijds de grens, het exterieur of het interieur van v_1, zijn 8 relaties te vinden, waarvan twee onmogelijk zijn binnen dezelfde dimensie, zodat 6 mogelijke relaties overblijven. *Tabel naar [pig92]*.

Zo blijven er zes mogelijke topologische relaties tussen v_1 en v_2 over (zie ook afbeelding 5.5):

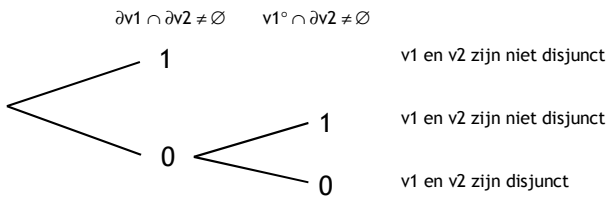
- v_1 en v_2 zijn disjunct (eng.: *disjoint*),
- v_1 en v_2 zijn gelijk (eng.: *equal*),
- v_1 bevat v_2 (eng.: *concur*),
- v_1 overlapt v_2 (eng.: *overlap*),
- v_1 raakt v_2 (eng.: *meet*),
- v_1 bevat en raakt v_2 (eng.: *commonbounds*).



afbeelding 5.5: De 6 mogelijke topologische relaties tussen twee tetraëders. Afgebeeld zijn de zes mogelijke topologische relaties tussen de tetraëders v_1 en v_2. *Afbeelding naar [pig92]*.

Bij het selecteren van objecten op basis van een buffer is van belang of twee tetraëders disjunct zijn. Dit kan op twee manieren. De eerste manier is rechtstreeks te bepalen of twee tetraëders disjunct zijn. Hiervoor moet bepaald worden of stelling E waar is, stelling B onwaar en stelling I onwaar. Hierdoor moeten op zijn minst 1 en op zijn hoogst 3 vergelijkingen gemaakt worden. De tweede manier is te bepalen of twee tetraëders juist niet disjunct zijn. Omdat twee tetraëders niet zowel wel disjunct als niet disjunct kunnen zijn worden hiermee ook disjuncte tetraëders gevonden. Voor niet-disjuncte tetraëders geldt: stelling B is waar is óf stelling I is waar óf zowel stelling B als stelling I is waar. Als beide stellingen niet waar zijn, dan zijn beide tetraëders disjunct omdat niet alle stellingen tegelijk onwaar kunnen zijn. Bij de tweede manier zijn nu ten minste 1 en ten hoogste 2 vergelijkingen nodig (zie afbeelding 5.6).

afbeelding 5.6: Beslissingsboom bij het vinden van niet-disjuncte tetraëders. Twee tetraëders v_1 en v_2 zijn niet-disjunct als de grens van v_2 ruimte deelt met de grens van v_1 en/of als de grens van v_2 ruimte deelt met het interieur van v_1. Totaal zijn er dus maximaal twee beslismomenten.



Bij deze beschrijving van Pigot is een kanttekening te plaatsen. De bewering dat twee tetraëders v_1 en v_2 disjunct zijn als alleen stelling E: $\partial v_2 \cap \bar{}v_1 \neq \emptyset$ waar is, dus als geldt (E, B, I) = (1, 0 ,0), is niet altijd correct: als v_1 bevat wordt door v_2 dan geldt stelling E namelijk ook. Dus als twee schijnbaar disjuncte tetraëders gevonden zijn, moet bepaald worden of de grens van v_1 zich in het exterieur van v_2 bevindt, dus $\partial v_1 \cap \bar{}v_2 \neq \emptyset$. Als die stelling waar is zijn v_1 en v_2 werkelijk disjunct.

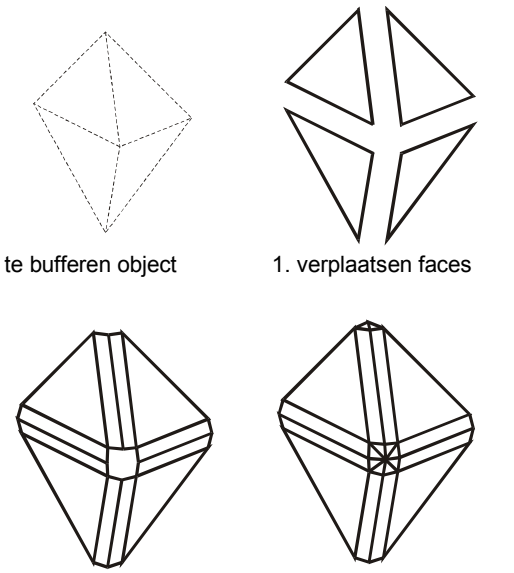
De bepaling, welke objecten geselecteerd worden door de buffer komt zo neer op de bepaling van de topologische relatie tussen tetraëders van de objecten en de tetraëders waaruit de buffer opgebouwd is. Per paar tetraëders zijn hier, inclusief de kanttekening bij Pigot, ten hoogste 3 beslismomenten voor nodig. Bovendien geldt dat als een niet-disjuncte paar tetraëders gevonden is, dat het hele object waar een van beide tetraëders deel van uit maakt, niet-disjunct is met de buffer, zodat van de andere objecttetraëders niet meer de relatie met de buffertetraëders bepaald hoeft te worden. Dankzij de opslag in een TEN is zo een snelle selectie mogelijk.

de opbouw van een afgeronde 3D buffer

Een simpele benaderd afgeronde tweedimensionale buffer met één tussenpunt (alternatief I) wordt gegenereerd door de vertices van de randen van het te bufferen object te verplaatsen in een richting loodrecht op de randen en met een afstand gelijk aan de gewenste bufferafstand. Voor de hoeken worden tussenvertices te gegenereerd op een afstand dat gelijk is aan de gewenste bufferafstand en een richting dat gelijk is aan de helft van de hoekgrootte. Een object in drie dimensies, gemodelleerd in een TEN, bestaat uit vertices (*nodes*), randen (*edges*), vlakken (*faces*) en cellen (*cells*). Met een extra stap is met deze elementen analoog aan de opbouw van een 2D-buffer een driedimensionale buffer te genereren. Bij een 3D-Buffer onstaan dan tussenedges, die de afronding tussen twee vlakken benaderen, en hoeknodes, die de afronding bij de hoeken benaderen.

Een simpele 3D-buffer kan opgebouwd worden (zie afbeelding 5.7) door elk vlak te verplaatsen in de richting van de normaal op dat vlak en met een afstand gelijk aan de bufferafstand. Van elke edge wordt bepaald welke vlakken bij die edge horen, en de som van de bij die vlakken behorende normaalvectoren wordt bepaald. Vervolgens worden tussenedges gevormd door de edges van het object te verplaatsen in de richting van de zojuist bepaalde somvector, en met de gewenste bufferafstand. Eventueel kunnen er meerdere tussenedges toegevoegd worden, om zo de benaderingsfout te verkleinen. Ten slotte wordt per node bepaald, welke vlakken daar samenkomen, en wat de som van de daarbijbehorende normaalvectoren is. Op basis van deze somvector en de bufferafstand worden hoeknodes aan de buffer toegevoegd. Het uiteindelijk verkregen puntenwolk wordt getrianguleerd, zodat de buffer in dezelfde datastructuur als het gebufferde object geretourneerd wordt.

afbeelding 5.7: De opbouw van een 3D-buffer. Een benaderd afgeronde buffer kan opgebouwd worden door de faces van het te bufferen object te verplaatsen met de gewenste bufferafstand (1), vervolgens tussenedges toe te voegen. In de afbeelding (2) wordt één tussenedge toegevoegd per objectedge, maar dat kunnen er ook meer zijn. Ten slotte worden de hoeknodes toegevoegd (3). Het aldus verkregen puntenveld wordt getrianguleerd om een TEN als uitvoer te geven.



5.3 het programma '3Dbuffer'

Om de opbouw van een driedimensionale buffer te demonstreren, is een programma geschreven dat een 3D-buffer genereert voor een convex object dat gemodelleerd is in een TEN. Een handzaam model om een computersysteem te beschrijven, is het COM/DNA model, waarbij een systeem bestaat uit een zogenaamde *datastore layer*, een *business rules layer*, en uit een *presentation layer*. Overigens is dit model een 3-laags, of *3-tier* model, en worden huidige systemen uit meer lagen opgebouwd.

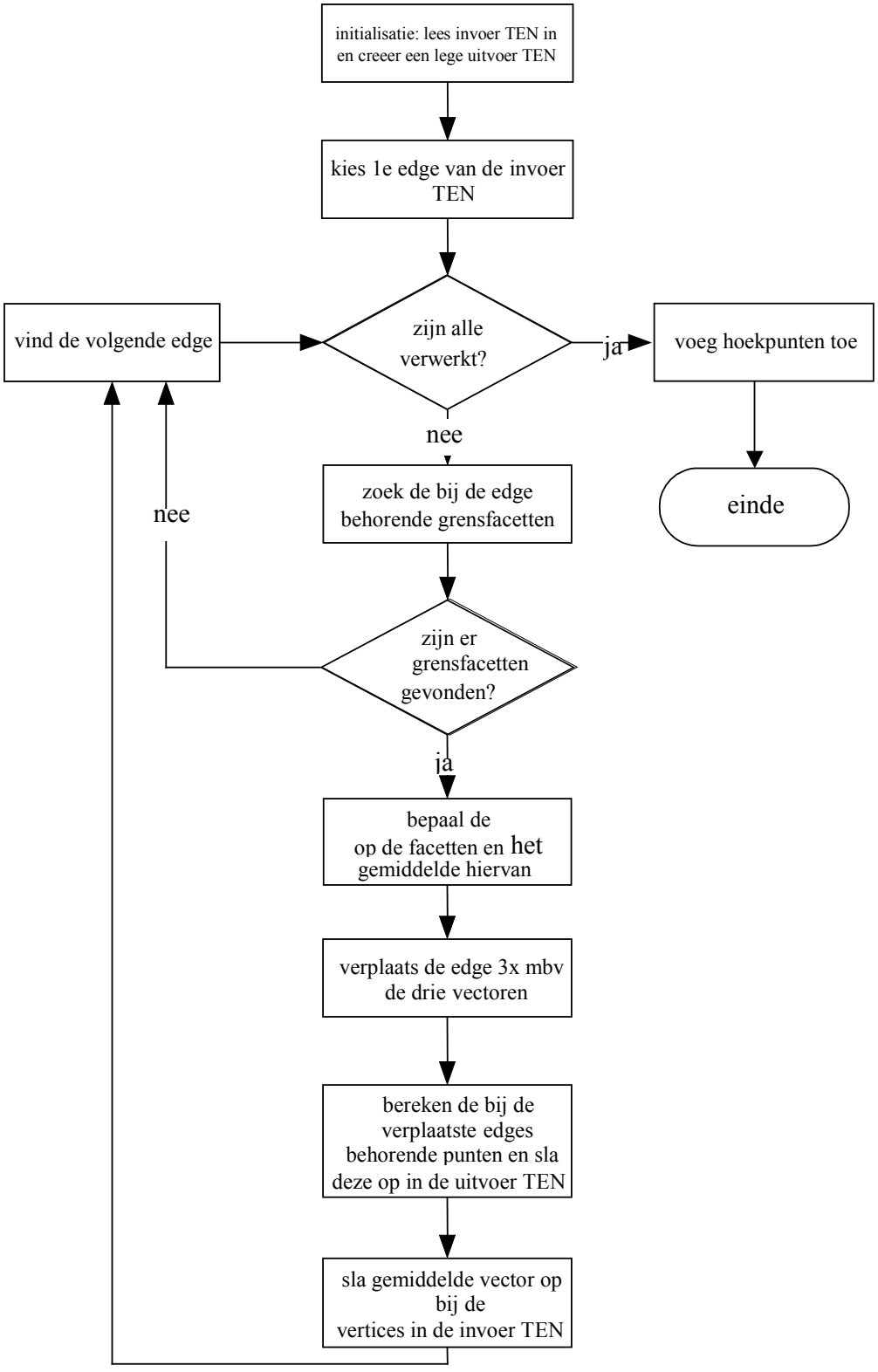
De presentation layer is de laag waarin interactie tussen de gebruiker en het systeem plaats vindt, bijvoorbeeld met behulp van visualisatie, maar ook in de vorm van tabellen, rapporten, etc. Omdat de nadruk bij dit programma ligt op het bufferalgoritme, is deze laag minimalistisch geïmplementeerd; de gebruiker hoeft alleen een bufferstraal in te voeren, en presentatie vindt plaats door de resultaten naar een raw triangle-formaat uit te voeren, dat ten behoeve van de visualisatie geconverteerd kan worden naar VRML-formaat. Bestanden in dit 3D-formaat zijn met een standaard webbrowser en een vrml-plugin te bekijken en te bestuderen.

De business rules layer is de laag waarin het algoritme uitgevoerd wordt. Hierbij is gebruik gemaakt van CGAL, versie 2.2. De wijze waarop de buffering uitgevoerd wordt is daarbij afhankelijk van de wijze waarop een TEN in CGAL beschreven wordt.

De datastore layer is de laag waar de opslag plaats vindt, meestal in de vorm van een database. In dit programma is gekozen voor opslag van het te bufferen object en van de buffer in een tekstbestand. Het gebruik van een database zou hier niet functioneel zijn, omdat de daarin opgeslagen gegevens voor CGAL weer geconverteerd moeten worden.

Het programma is geschreven in C++, en is gecompileerd met g++ op een Sparc sunOS 5.7 platform, alhoewel geen gebruik is gemaakt van platformspecifieke onderdelen als GeomView, zodat de code ook op een Windows NT platform gecompileerd kan worden. Deze paragraaf beschrijft de belangrijkste stappen in het programma aan de hand van de belangrijkste functies uit CGAL die in de code (zie bijlage 1 voor de volledige broncode) toegepast zijn. Om aan te sluiten bij de code wordt de engelse benaming gebruikt en omwille van de duidelijkheid worden de triangulatie 'mijnTEN', edge 'mijnEdge', vector 'mijnVector' etc. als invoer aangenomen.

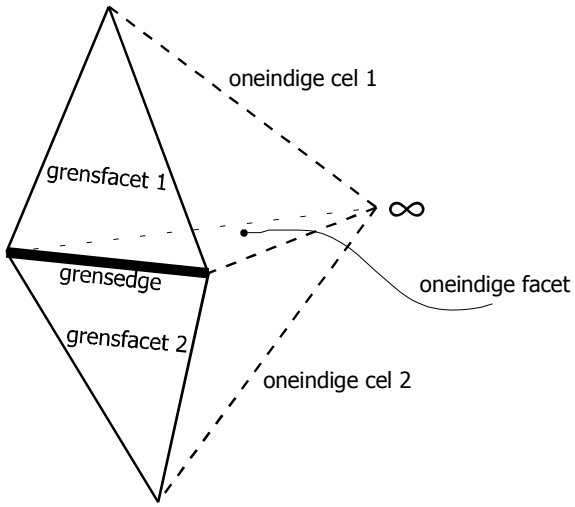
De keuze voor CGAL heeft voor deze toepassing het nadeel, dat de TEN, of triangulatiedatastructuur, alleen bestaat uit vertices en uit cellen (de tetraëders). Faces en edges zijn niet expliciet in de datastructuur opgenomen, hetgeen te verklaren is doordat deze twee elementen overtallig zijn. Het bufferalgoritme moest voor het programma daardoor echter aangepast worden (zie afbeelding 5.8). In plaats van een vlaksgewijze buffering, wordt nu een buffer per edge opgebouwd.



afbeelding 5.8: stroomschema van 3DBuffer. In tegenstelling tot de theorie, waarbij eerst de facetten worden verplaatst, daarna de tussenedges worden toegevoegd en ten slotte de hoekpunten, vindt de buffering in 3DBuffer plaats per edge; elke edge wordt drie keer verplaatst in verschillende richtingen waarna de hoekpunten worden toegevoegd.

stap 1: vindt de boundary edges en de boundary facets

Nadat een nieuwe triangulatie is aangemaakt, wordt per edge bepaald of deze deel uit maakt van de begrenzing (eng.: *boundary*) van het te bufferen lichaam en, als dat het geval is, welke *boundary faces* bij deze edge horen (zie afbeelding 5.9). Deze facetten moeten gevonden worden om uiteindelijk verplaatsingsvectoren te berekenen.



afbeelding 5.9: De bepaling van boundary edges en de bijbehorende boundary facets. Als een edge deel uit maakt van de grens van een object dan is precies één van de bij die edge samenkomende facetten een oneindige facet, d.w.z. een facet waarvan een vertex de oneindige vertex is. Bij die oneindige facet kunnen twee oneindige cellen gevonden worden en aan de hand van de index van de oneindige vertex in die cellen worden de grensfacetten gevonden, doordat deze zich tegenover de oneindige vertex bevinden.

Een edge behoort tot de grens van een object, als één van de facetten die bij die edge samenkomen, een oneindige facet is. Een facet is oneindig één van zijn vertices de oneindige vertex is, oftewel als de facet een facet van een oneindige cell is. Alle bij een edge behorende facetten kunnen gevonden worden met *facet circulators*:

```
Facet_circulator FC = mijnTen.incident_facets (*mijnEdge);
```

Een facet circulator bevat twee getallen; het eerste getal geeft de cell aan waartoe de facet behoort, en het tweede getal geeft de index van de facet binnen die cell aan en kan dus 0, 1, 2 of 3 bedragen. Van een facet kan vervolgens bepaald worden of deze oneindig is met:

```
mijnTen.is_infinite (*FC);
```

Als bij een edge geen van de met de facet circulators gevonden facetten oneindig is, dan bevindt de edge zich niet op de grens van het object. Als de edge deel uit maakt van de grens, dan wordt dus een oneindige facet gevonden, waarbij twee oneindige cellen gevonden worden. De cellen worden geïdentificeerd met *cell handles*, en de gezochte cellen worden gevonden aan de hand van de twee getallen bij de gevonden oneindige facet:

```
Cell_handle mijnCell1 = (*FC).first;
Cell_handle mijnCell2 = mijnCell1->neighbor((*FC).second);
```

In de tweede regel wordt gebruik gemaakt van het feit, dat de buur van een cell in CGAL dezelfde index heeft als de facet die beide cellen verbindt.

Een boundary facet maakt deel uit van een oneindige cell, waarvoor geldt dat één van zijn vertices de oneindige vertex is, en heeft in die cell dezelfde index als de tegenover liggende oneindige vertex. Deze index wordt gevonden met:

```
Vertex_handle OneindigeVertex = mijnTen.infinite_vertex();
int idx1OneindigeVertex = mijnCell1->index(OneindigeVertex);
```

De boundary facet wordt ten slotte gedefinieerd aan de hand van de oneindige cell en de gevonden index met:

```
mijnFacet.first = mijnCell1;
mijnFacet.second = idx1OneindigeVertex;
```

In de code wordt zo ook de boundary facet van de andere oneindige cell gevonden.

stap 2: bepaal de verplaatsingsvectoren

Als de bij een edge behorende boundary faces gevonden zijn, worden drie verplaatsingsvectoren berekend; twee vectoren loodrecht op de driehoeken die bij de facetten horen, en een vector dat het gemiddelde is van deze twee vectoren. De vectoren worden genormaliseerd. Eerst wordt bepaald, welke -geometrische- driehoek bij de -topologische- facet hoort en vervolgens het vlak door die driehoek:

```
Triangle_3 mijnDriehoek = mijnTen.triangle (mijnFacet);
Plane_3 mijnVlak = mijnDriehoek.supporting_plane();
```

Door de nummering van de vertices in een cell zijn twee facetten uitwaarts georiënteerd en twee facetten inwaarts. In de code wordt gewerkt met facetten die, conform stap 1, ten opzichte van oneindige cellen zijn gedefinieerd dus moet, voordat een normaalvector bepaald wordt, bepaald worden of de facet *in*waarts georiënteerd is en als dit niet het geval is moet het tegenoverliggende vlak gebruikt worden. Met andere woorden: aan de hand van de oriëntatie van de facetten wordt het juiste vlak bepaald:

```
int mijnIndex = mijnFacet.second;
if ((mijnIndex==0) || (mijnIndex==2)) mijnVlak = mijnVlak.opposite();
```

De loodrechte vector wordt ten slotte gevonden met:

```
Vector_3 mijnVector = mijnVlak.orthogonal_vector();
```

Door dit voor beide facetten uit te voeren en de gemiddelde vector te bepalen en vervolgens een lengte 1 te geven, worden zo de verplaatsingsvectoren gevonden. Om later hoekpunten te genereren, wordt de gemiddelde vector genormaliseerd en opgeteld bij al eerder opgeslagen vectoren bij de vertices die bij de edge, maar alleen als de twee normaalvectoren niet dezelfde richting hebben, met andere woorden: alleen als de twee facetten niet in hetzelfde vlak liggen.

De test of twee facetten niet in hetzelfde vlak liggen, is een zwak punt in de code. De vergelijking van de richting van twee normaalvectoren is niet ideaal omdat een kleine verschuiving, bijvoorbeeld door onnauwkeurigheid, kan leiden tot het onterecht optellen van een gemiddelde vector, waardoor het hoekpunt uiteindelijk verschuift. Dit probleem kan opgelost worden door bijvoorbeeld een grens te stellen aan het verschil tussen twee normaalvectoren.

Als er voor gekozen wordt om meerdere tussenedges in te voegen, moet dat in deze stap gebeuren. Het uitgangspunt is dan nog steeds de twee berekende vectoren loodrecht op de driehoeken die bij de facetten horen, maar nu wordt geen gemiddelde vector berekend, maar meerdere vectoren die de hoek tussen de twee berekende vectoren evenredig verdelen.

stap 3: verplaatsen van de edges

De genormaliseerde verplaatsingsvectoren krijgen een lengte gelijk aan de bufferafstand en de edges worden aan de hand van deze vectoren verplaatst. Eerst wordt de bij de edge behorende segment gevonden met:

```
Segment_3 mijnEdge = mijnTen.segment(mijnEdge);
```

Bij dit segment horen twee punten: een *source* en een *target*. Door hierbij een vector op te tellen worden de nieuwe bufferpunten gevonden. Deze bufferpunten worden toegevoegd aan een buffer dat opgeslagen is in een andere TEN dan het object zelf:

```
Vertex_handle Vh;
Point_3 mijnPunt1 = segEdge.source() + mijnVector;
Vh = mijnBufferTen.insert(mijnPunt1);
Point_3 mijnPunt2 = segEdge.target() + mijnVector;
Vh = mijnBufferTen.insert(mijnPunt2);
```

Het resultaat is dan uiteindelijk een getrianguleerd puntenveld dat alleen punten van de buffer bevat.

stap 4: voeg tussenpunten toe

Als alle edges verwerkt zijn worden ten slotte de hoekpunten gegenereerd aan de hand van de bij de vertices opgeslagen vectoren. Alle vertices, behalve de oneindige vertex, van het object worden verwerkt met behulp van een *vertex iterator*:

```
Vertex_iterator HuidigeVertex = mijnTen.finite_vertices_begin();
Vertex_iterator EindVertex = mijnTen.vertices_end();
```

De verwerking gebeurt in een loop, dat eindigt als EindVertex verwerkt is. Om de vectoren op te kunnen slaan is een klasse Hoekvertex gemaakt dat een afgeleide is van de vertex-klasse van CGAL. De bij een vertex behorende vector en de bij een vertex behorende punt worden nu opgevraagd met:

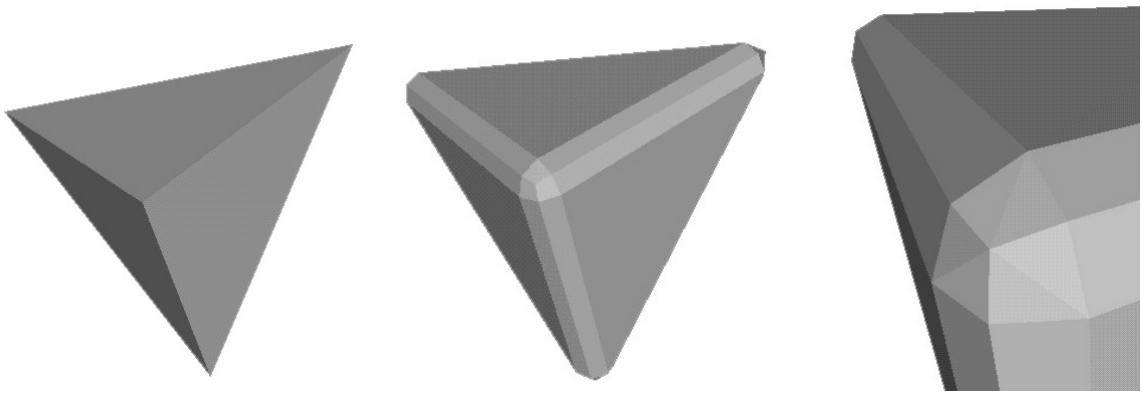
```
Vector_3 HoekVector = HuidigeVertex->get_vector();
Point_3 ObjectPunt = HuidigeVertex->point();
```

De gevonden vector krijgt een lengte dat gelijk is aan de bufferafstand en wordt opgeteld bij het objectpunt om een nieuw bufferhoekpunt te vormen. Dit nieuwe punt wordt vervolgens aan de buffer toegevoegd:

```
Point_3 BufferPunt = ObjectPunt + HoekVector;
mijnBufferTen.insert(BufferPunt);
```

resultaat

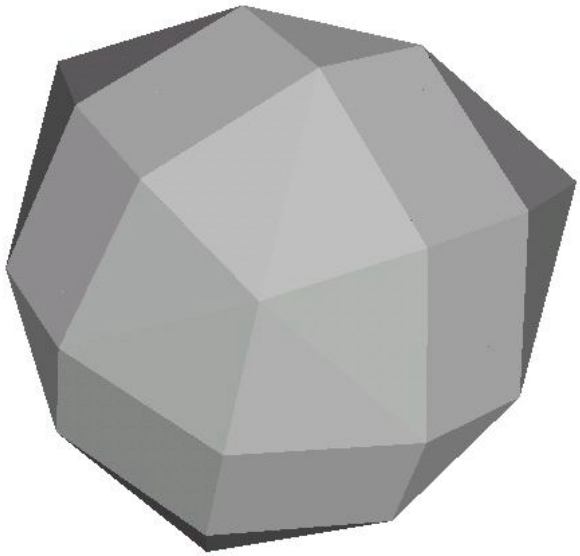
Afbeelding 5.10 laat het resultaat zien van de buffering van een tetraëder met een relatief kleine straal. duidelijk te zien is de benadering van de afronding van de randen en van de hoeken door verplaatste edges, tussenedges en een tussenpunt. Omdat de resulterende buffer ook weer in een TEN is gemodelleerd, bestaan de rechthoekige vlakken in de afronding in feite uit twee facetten van twee tetraëders (afbeelding 5.10).



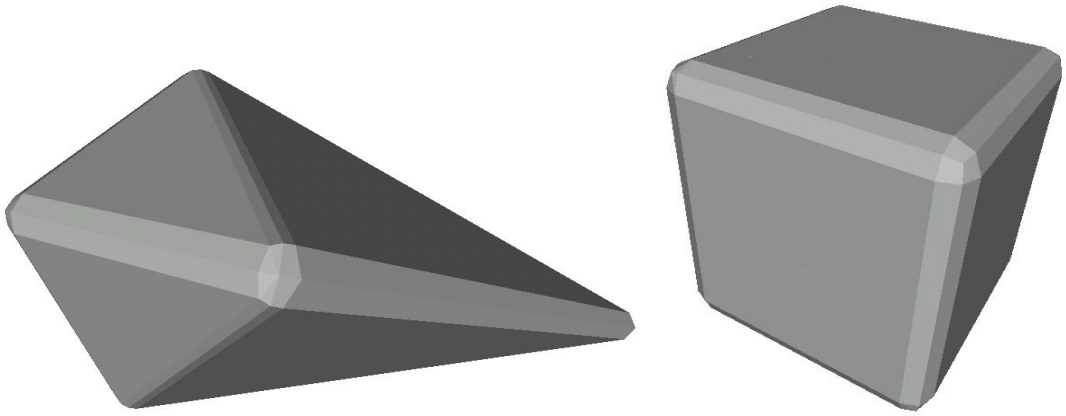
afbeelding 5.10: 3D-buffer van een tetraëder. Afgebeeld is een tetraëder (links) en een daarvoor gegenereerde buffer met straal 0,1 (midden). Op de close-up van een hoek van de buffer (rechts) zijn goed de elementen te zien waaruit de buffer bestaat: de verplaatste edges, de tussenedges, en een tussenpunt. Overigens is de linkerafbeelding gemaakt door een buffer met straal 0 te maken.

In afbeelding 5.11 is de tetraëder gebufferd met een straal die groter was dan de grootste zijdelengte van de tetraëder, om te demonstreren dat de keuze voor de afronding bij dergelijke buffers van groot belang is; de elementen die bij de afronding horen maken een groter deel uit van de buffer dan de facetten die onstaan door alleen de edges te verplaatsen. Laatstgenoemde facetten hebben immers dezelfde grootte als hun origineel.

afbeelding 5.11: afgeronde buffer met een grote straal ten opzichte van één tetraeder. Deze buffer is gegenereerd met een straal die groter was dan de grootse zijdelengte van de tetraëder, zodat de afronding het grootste deel van de buffer uitmaakt. De facetten die ontstaan door de facetten van de gebufferde tetraëder te verplaatsen zijn te onderscheiden doordat ze met elkaar verbonden zijn door de rechthoeken, die de afgeronde, gebufferde edges zijn.

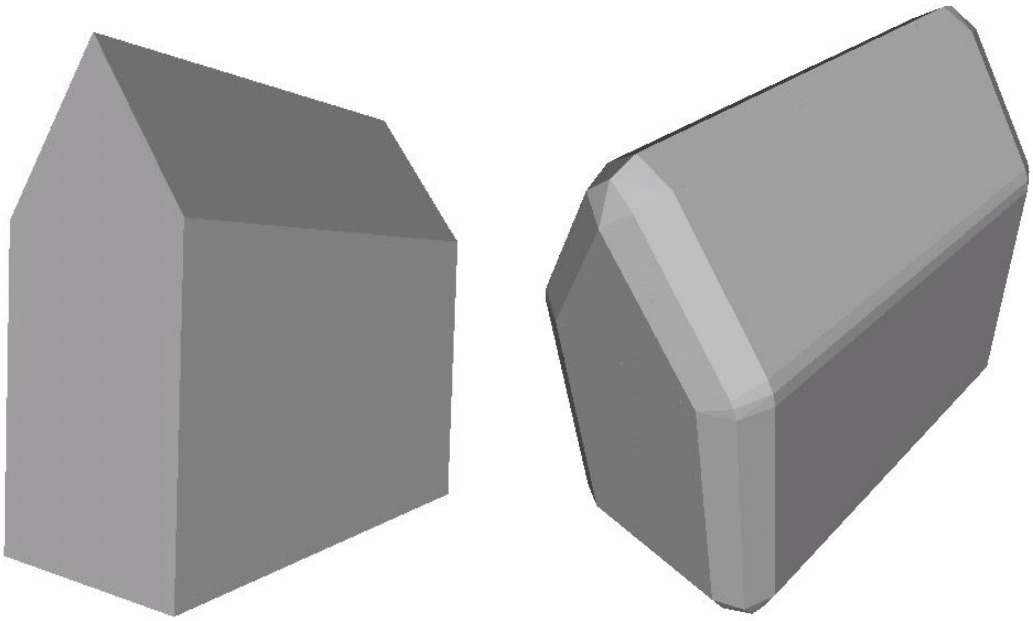


Ook complexere vormen zijn gebufferd (afbeelding 5.12), bijvoorbeeld voor een object dat uit twee tetraëders bestaat en bijvoorbeeld voor een kubus. Een kubus kan uit 5 tetraëders opgebouwd worden maar de wijze waarop CGAL een kubus in een TEN modelleert, hangt af van de volgorde waarop de hoekpunten toegevoegd worden aan de TEN, en het resultaat kan bestaan uit zes tetraëders. Dat is voor te stellen door de kubus te delen in twee prisma's en deze elk op te bouwen uit 3 tetraëders. Dit maakt de kubus wel interessant als testobject omdat bij een vertex tussen drie en zes facetten samen kunnen komen, terwijl de buffering van een kubus altijd hetzelfde resultaat moet geven.



afbeelding 5.12: buffering van complexere figuren. Afgebeeld zijn een buffers met bufferafstand 0,1 van twee tetraëders (links) en van een kubus (rechts). De gebufferde hoeken zijn het resultaat van buffering van meerdere tetraëders

Ten slotte is een minder abstract voorwerp gebufferd: een huis met een dak (zie afbeelding 5.11



afbeelding 5.13: Buffer voor een huis met een dak.

Een dergelijke buffer kan bijvoorbeeld toegepast worden als het huis te midden van flats staat en bepaald moet worden welke appartementen in die flats geluidshinder ondervinden veroorzaakt door de bewoners van dit huis.

5.4 driedimensionale buffering voor concave objecten

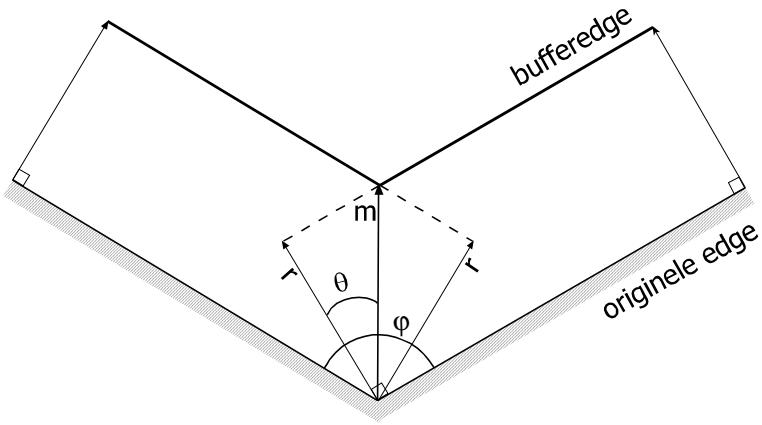
In dit onderzoek is de driedimensionale buffer beschreven voor convexe objecten. Dit heeft als reden dat de driedimensionale buffering op zich niet het doel van dit onderzoek was. Een andere belangrijke reden is de implementatie van de tetrahedral network in CGAL. Een getrianguleerd driedimensionaal puntenveld is altijd een convex object en om een concaaf object te modelleren, moet het mogelijk zijn om de grensobjecten van dat object in de triangulatie te behouden, het moet met andere woorden mogelijk zijn om *constraints* aan de triangulatie op te leggen. Dit is in CGAL (nog) niet mogelijk. bovendien moet van de genoemde grenselementen bijgehouden worden, dat deze weliswaar niet grenzen aan een oneindige cel, maar wel deel uit maken van de

feitelijke grens van het object. Dit is in CGAL wel mogelijk door van die elementen zelf een afgeleide klasse te definiëren.

Toch is het belangrijk om uiteindelijk een driedimensionale buffering van concave objecten te ontwikkelen, omdat veel te modelleren fenomenen een concave vorm hebben. Zo is een huis nog wel als convex object te modelleren, maar als er bijvoorbeeld een schoorsteen op gezet wordt dan is het object concaaf. In deze paragraaf wordt een opzet gegeven voor de buffering van concave objecten.

buffering van concave objecten in twee dimensies

In principe verloopt de buffering van concave objecten analoog aan de buffering van convexe objecten. Een hoek wordt concaaf genoemd als voor de hoek aan de binnenzijde van het object, bh, geldt: $bh > \pi$, en een hoek is convex als geldt: $bh < \pi$. In beide gevallen worden voor de buffering van edges deze met de bufferafstand verplaatst, maar als twee edges een concave hoek vormen, dan snijden deze bufferedges elkaar. Op dit snijpunt wordt een nieuw bufferpunt gevormd en de resulterende bufferedges hebben een kortere afstand dan de objectedges waar deze bij horen (zie afbeelding 5.12)

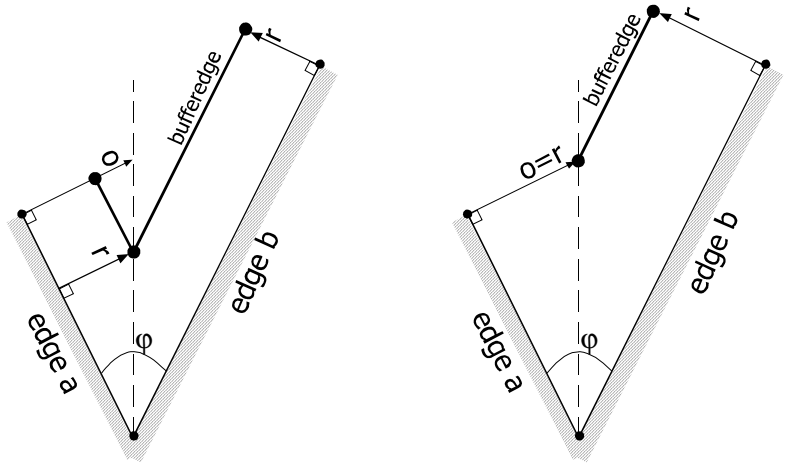


afbeelding 5.14: snijden van bufferedges bij concave hoek. Als de hoek tussen twee edges concaaf is (voor de hoek aan de buitenkant van het object, φ, geldt: $\varphi < \pi$) dan is de edge van de bijbehorende buffer kleiner, en de vector naar het bij het objecthoekpunt behorende bufferpunt is te bepalen uit r en θ.

De uiteindelijke positie van het bij een concave hoek behorende bufferpunt, en daardoor de lengte van de twee daar samenkomende bufferedges, is afhankelijk van de hoek tussen de edges, φ, en de bufferafstand, r. de afstand tussen het objecthoekpunt en het bufferpunt, m, is te berekenen met:

$$\theta = \tfrac{1}{2}\pi - \tfrac{1}{2}\varphi$$
$$\mathbf{m = r / \cos \theta .}$$

De berekende afstand m is overigens geen bufferafstand; de afstand wordt niet gebruikt voor een afronding, maar voor snijding van edges. Omdat de verplaatste edges, de bufferedges, kleiner worden dan de objectedges waar zij bij horen, is het nuttig te bepalen op welk moment de lengte van die edges gelijk aan nul wordt (zie afbeelding 5.11). In dit geval wordt de objectedge niet gebufferd.



a) o > r b) o ≤ r

afbeelding 5.15: verkleining van de bufferedge. Bij het bufferen van een concave edge wordt de resulterende edge kleiner; deze verkleining is begrensd en afhankelijk van de bufferafstand, de hoek van de edge met een volgende edge en de lengte van de objectedge. Hiermee is een lengte o te bepalen aan de hand waarvan bepaald wordt of een edge gebufferd wordt of niet. Als (a) o > r, dan heeft de bufferedge een lengte en wordt dus gegenereerd, als (b) o ≤ r, dan wordt er geen bufferedge gegenereerd.

De lengte van de bufferedge is afhankelijk van de twee hoeken tussen zijn aangrenzende edges. Als één concave hoek in beschouwing wordt genomen, dan is de lengte afhankelijk van deze hoek, φ, de bufferafstand, r, en de lengte van de objectedge, a. hiermee kan een afstand o berekend worden:

$$\mathbf{o = a \cdot \tan(\tfrac{1}{2}\varphi)\ .}$$

Alleen als geldt: o > r, heeft de bufferedge een lengte, als geldt: o ≤ r, dan heeft de bufferedge geen lengte en wordt in de uiteindelijke buffering overgeslagen.

buffering van concave objecten in drie dimensies

In drie dimensies moeten niet alleen holtes beschouwd worden, maar ook bijvoorbeeld zadelpunten. Voor holtes geldt, dat de hoek tussen elk facet aan de buitenzijde van het object kleiner dan π is, en het bufferen komt, geometrisch, neer op het vinden van snijlijnen van de met een bepaalde afstand verplaatste facetten. Tussenpunten worden gevonden door het snijden van de snijlijnen. Bij zadelpunten worden afronding en snijding van facetten gecombineerd. Als bij een punt bijvoorbeeld vier facetten samenkomen,waarbij twee edges twee facetten elk in een concave hoek verbinden, en twee edges twee facetten elk in een convexe hoek, dan worden eerst de facetten bij de convexe edges gebufferd, zodat zes nieuwe edges ontstaan. Deze edges worden vervolgens aan de hand van de concave edges gesneden. Het resultaat is dat voor het zadelpunt in de buffer 3 bufferpunten gegenereerd worden en 8 nieuwe bufferedges.

6. Conclusies en aanbevelingen

Een GIS is grofweg te beschrijven als een systeem voor opslag, bewerking en presentatie van ruimtelijk te lokaliseren gegevens. Voor een driedimensionale GIS geldt dat evenzeer, maar het begrip '3D' wordt vaak ruim geïnterpreteerd. Vaak wordt met 3D alleen de presentatie bedoeld; het gaat daarbij om een 2D-GIS met 3D-visualisatiemogelijkheden, zoals VR. GI-systemen waarmee ruimtelijke problemen op basis van terreinmodellen worden geanalyseerd, worden ook vaak driedimensionaal genoemd, terwijl een betere aanduiding '2,5-dimensionaal' is. Systemen waarbij ruimtelijke gegevens wel in een driedimensionale database worden opgeslagen, bestaan vaak uit een aparte GIS-database, gekoppeld aan een aparte visualisatiedatabase. In dit rapport is onderzocht, hoe een 'echte', want geïntegreerde 3D-GIS opgezet kan worden, waarbij de drie elementen opslag, bewerking en presentatie geïntegreerd zij tot één geheel. De nadruk bij het onderzoek lag daarbij, omdat visualisatietechnieken al uitgebreid onderzocht zijn, op opslag en bewerking.

Dit hoofdstuk geeft een overzicht van de belangrijkste conclusies, die naar aanleiding van het onderzoek worden getrokken, en doet aanbevelingen voor verder onderzoek.

6.1 conclusies

Om de opbouw van een geïntegreerde 3D-GIS te onderzoeken, is een inventaris gemaakt van GIS-analysegereedschappen, en naar modelleringtechnieken. Op basis hiervan is met behulp van een case uitgewerkt hoe gegevens in een geïntegreerde 3D-GIS opgeslagen kunnen worden, en is de constructie van een analysegereedschap, namelijk de buffer, in drie dimensies beschreven.

Bij de inventarisatie van de GIS-analyses is een overzicht gegeven van de belangrijkste bestaande analysevormen, en is bepaald of, en hoe deze toegepast kunnen worden in een 3D-GIS. Daarbij werd de volgende indeling gebruikt:

- directionele relaties
- transformaties
- objecteigenschappen
- metriek
- topologie
- attribuutoperaties
- thematische functies
- laagoperaties
- zichtbaarheidfuncties
- visualisatie

De meeste analysevormen kunnen met enige aanpassing of aanvulling toegepast worden in drie dimensies.

Bij de bouw van een 3D-GIS moet een keuze gemaakt worden uit beschikbare manieren om objecten in drie dimensies te modelleren. Daarom is in dit rapport een inventaris opgesteld van geometrische modellen, waaruit blijkt dat, voor een algemene driedimensionale GIS, een Tetrahedral Network (TEN) uitermate geschikt is. Een TEN is een datastructuur waarbij objecten opgebouwd worden uit niet-overlappende tetraëders, en is vergelijkbaar met triangulatie in 2D. De TEN is een consistente datastructuur met sterke topologische eigenschappen. Bovendien zijn vrije vormen te benaderen en zijn operaties op een TEN niet complex.

De opslag van ruimtelijke gegevens in een 3D-GIS is beschreven aan de hand van een case, waarbij een 3D-GIS ingezet wordt ten behoeve van de brandweer. Een belangrijke voorwaarde aan de datastructuur is, dat verschillende abstractieniveaus toegestaan worden. Een objecthiërarchie is daarbij de beste keuze. In de case kan een gebouw bijvoorbeeld gemodelleerd worden door een gebouw als object te beschouwen, dat verdiepingen bevat. Een verdieping als

object bevat vervolgens weer kamers, die op hun beurt geometrisch beschreven zijn. Een brandweer kan zo, als in een kamer brandt uitbreekt, bepalen welke andere kamers onder invloed van rook komen, maar ook welke verdiepingen geëvacueerd moeten worden.

In dit rapport is een analysegereedschap nader uitgewerkt. Gekozen is voor de bufferzone, omdat vertaling hiervan naar drie dimensies een aparte benadering behoeft. Een buffer in 2D is een vlak waarbij altijd afrondingsfouten ontstaan bij de hoeken. Een buffer in 3D is een volume. Er treden niet alleen fouten op bij de hoeken, maar ook bij de randen van de buffer. Een betere oplossing is een buffer te genereren dat bij de randen en de hoeken afgerond is. Met behulp van de programmabibliotheek CGAL is een programma geschreven waarmee buffers op basis van Tetrahedral Networks gegenereerd worden. Het programma bewijst, dat een dergelijke driedimensionale, afgeronde, buffer met behulp van een standaardgereedschap in de praktijk te genereren is en dat de theoretische opbouw van 3D Buffers in de praktijk ook geldt.

De Tetrahedral Network is voor bufferoperaties de meest geschikte modelleringtechniek. Een buffer bepaalt de locaties die zich binnen een bepaalde reikwijdte van een object bevinden, en is daarmee een operatie die op de grens van een object plaats vindt. Voor het bepalen van de grens in drie dimensies wordt gebruik gemaakt van de zeer sterke topologische structuur van een TEN. Het genereren van een bufferzone is geen doel op zich, maar wordt gebruikt om met een bufferoverlay objecten te selecteren. Per object komt dat neer op de bepaling per tetraëder, wat de topologische relatie met de buffer is. Alleen die objecten worden geselecteerd, die niet losstaan van de buffer. Per paar tetraëders moeten hiervoor hoogstens drie vergelijkingen gemaakt worden. Bovendien geldt dat als een tetraëder zich geheel of gedeeltelijk binnen de buffer bevindt, dat ook geldt voor het hele object waar de tetraëder een cell van is. Alle cellen die zich slechts gedeeltelijk binnen de buffer bevinden kunnen eventueel geclipped worden ten opzichte van de buffer.Hier bewijst de TEN zich ten opzichte van B-Rep en SOE, omdat de bufferoverlay op in TEN's gemodelleerde objecten zowel snel als nauwkeurig uitgevoerd wordt.

6.2 aanbevelingen

In dit rapport is het Tetrahedral Network nader uitgewerkt. Maar ook de Boundary Representation en de Spatial Occupancy Enumeration zijn, afhankelijk van de toepassing, geschikte geometrische datastructuren. Met name de Boundary Representation sluit aan op bekende visualisatiepakketten en is wellicht geschikt om zo'n pakket te koppelen aan een GIS. Nader onderzoek naar de toepassing van deze structuren in een driedimensionale GIS verdient aanbeveling.

Aangegeven is, dat er voor de tweedimensionale afgeronde buffer een aantal alternatieven bestaat. Deze zijn echter niet toegepast in de driedimensionale buffer, maar zijn bedoeld als voorzet naar verder onderzoek. Het kan bijvoorbeeld interessant zijn, om te onderzoeken met hoeveel hoekpunten een afgeronde buffer benaderd moet worden. Het vermoeden bestaat dat dit in ieder geval een oneven aantal moet zijn, maar dat vermoeden moet onderzocht worden.

Dit rapport beschrijft de constructie van driedimensionale buffers voor convexe objecten. Dit zou uitgebreid moeten worden voor concave objecten. In dit rapport is daarvoor een opzet gegeven, waarbij het mogelijk is, dat sommige delen van het te bufferen object zo klein worden in de buffer, dat deze in de constructie van de buffer worden overgeslagen.

De code van het programma waarmee 3D buffers gegenereerd worden, is voor optimalisatie vatbaar. Zo worden nu facetten verplaatst door de bijbehorende edges te verplaatsen. Omdat in een vertex van een facet twee van deze edges samenkomen, wordt uiteindelijk elk vertex twee keer in dezelfde richting verplaatst. Een verbetering kan zijn, de buffer per facet in plaats van per edge te genereren, maar in CGAL zijn facetten niet direct aan te spreken.

Een 3D-GIS is met name geschikt voor grootschalige toepassingen. Het is in dat kader beter om te spreken van een Ruimtelijk Informatiesysteem, omdat de schaal niet geografisch hoeft te zijn. Een '3D-GIS' kan bijvoorbeeld ingezet worden voor ruimtelijke medische problemen. Gelijk aan het in dit rapport beschreven gebouwmodel, zou een lichaam gemodelleerd kunnen worden aan de hand van het object 'orgaan', dat op zijn beurt beschreven wordt aan de hand van het object 'weefsel'. Voor een dergelijke toepassing leent de TEN zich, door de benadering van vrije vormen, perfect en is de beschikbaarheid 3D-analysegereedschappen zoals de 3D-buffer, maar ook andere driedimensionale analyses zoals bijvoorbeeld volumeberekening, erg belangrijk. Onderzoek naar deze toepassing van een geïntegreerde GIS op basis van een TEN viel buiten de doelstellingen van dit onderzoek, maar is wel toekomstig onderzoek waard, omdat hiermee niet alleen de techniek wordt beschreven, maar ook de noodzaak van een driedimensionaal geïntegreerde GIS wordt aangetoond.

Literatuur

[bat98] Batty, M. et al., *GIS and Urban Design.* Londen: CASA,UCL, 1998.

[bre96] Breunig, M., *Integration of Spatial Information for Geo-information Systems*. Berlijn: Springer-Verlag, 1996.

[bur98] Burrough, P.A. en A. McDonnell, *Principles of Geographical Information Systems.* New York: Oxford University Press, 1998.

[kön98] Köninger, A. en S. Bartel, '3D-GIS for Urban Purposes'. *GeoInformatica,* 2, nr.1 (1998), p.79-103.

[lim98] Limp, W.F. en Harmon, D., *Inside GeoMedia*. Santa Fe, NM: OnWord Press, 1998.

[loo98] Loo, J. van en J. Lawick - van Pabst, 'Geographical Information in Virtual Environments'. In: R.F. Erbacher en A. Pang (eds), *Proceedings of the SPIE - the International Society for Optical Engineering vol. 3298: Visual Data Exploration and Analysis V.* Washington: SPIE, 1998, p.63-69.

[mac99] MacEachren, A.M. et al., 'Cartographic Issues in the Design and Application of Geospatial Virtual Environments'. In: *Proceedings of the 19th International Cartographic Conference*. Ottawa: ICA, 1999, p.657-665.

[mol98] Molenaar, M., *An Introduction to the Theory of Spatial Object Modelling*.Londen: Taylor & Francis, 1998.

[oos94] Oosterom, P. et al., 'Integrated 3D Modelling Within A GIS'. In: M. Molenaar en S. De Hoop (eds), *Advanced Geographic Data Modelling, Spatial Data Modelling and Query Languages for 2D & 3D Applications.* Delft: Nederlandse Commissie voor Geodesie, 1994, p.80-95.

[orf98] Orford, S. et al., *Review of Visualization in the Social Sciences: a State of the Art Survey and Report.* Bristol: AGOCG, 1998.

[pig92] Pigot, S., 'Topological Models for 3D Spatial Information Systems'. In: *Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, Ohio*. Columbus, OH, IGU, 1992, p.368-392.

[pil96] Pilouk, M, *Integrated Modelling for 3D GIS*. Enschede: ITC, 1996 (proefschrift).

[ran97] Ranziger, M. en G. Gleixner, 'GIS datasets for Urban Planning'. *Computers, Environment and Urban Systems*, 21, nr.2 (1997), p.159-173.

[rap91] Raper, J.F. en B.Kelk, 'Three-Dimensional GIS'. In: D.J. Maguire, M.F. Goodchild en D.W. Rhind (eds), *Geographical Information Systems vol I: Principles*. Essex: Longman Scientific and Technical, 1991, p.299-317.

[sch00] Schirra, S et al., *CGAL Reference Manual. Release 2.1*. CGAL consortium, 2000.

[tem98] Tempfli, K, '3D Topographic Mapping for Urban GIS'. *ITC Journal*, 3/4 (1998), p181-190.

[ver92] Verbree, E., *Constructie en Toepassing van het 2-Dimensionale Delaunay Driehoeksmodel en het 3-Dimensionale Delaunay Tetraëdermodel*. Delft, 1992 (afstudeerverslag).

[vri99] Vries, J. de., *De Combinatie van GIS met Virtual Reality t.b.v. het Utrecht Centrum Project: Verslag van Stage bij Intergraph Benelux b.v.* .Delft / Hoofddorp, 1999 (stageverslag).

[vin95] Vince, J., *Virtual Reality Systems*. Wokingham: Addison-Wesley, 1995.

[wor95] Worboys, M.F., *GIS: A Computing Perspective*. herdruk. Londen: Taylor & Francis, 1997.

Gebruikte software

Computational Geometry Algorithms Library, release 2.2. (CGAL). CGAL consortium.

Cortona VRML client, versie 3.1 release 54. Parallelgraphics.

Crossroads 3D, versie 1.0. Keith Rule.

Bijlage 1: broncode 3Dbuffer

Om de vorming van een 3D Buffer te demonstreren is een programma geschreven dat een driedimensionale buffer genereert op basis van een object dat gemodelleerd is in een Tetrahedral Network (TEN). In de volgende code is gebruik gemaakt van CGAL 2.1 en Visual C++ 6.0. De invoer bestaat uit een tekstbestand, waarin de TEN opgeslagen is in het formaat van CGAL. De uitvoer bestaat uit een zelfde soort bestand, en uit een bestand in raw triangle formaat dat geschikt is voor conversie naar 3D bestandsformaten zoals VRML.

De gegenereerde buffer is een simpele buffer, met één tussenpunt per hoek, en één bufferafstand voor het hele object. De code is echter makkelijk aan te passen waardoor het invoegen van meerdere tussenpunten mogelijk is. De toepassing van meerdere bufferafstanden, zoals bij de in het rapport genoemde alternatieven III en IV, is met deze code niet mogelijk, evenmin als het bufferen van een concaaf lichaam. Het gebruik van een driedimensionale Delaunay-triangulatie is mogelijk, door de typedefinitie van de triangulatie-datastructuur aan te passen en een andere bibliotheek in de *include*-sectie aan te roepen.

```
/*********************************************************************************************
 ***                                                                                       ***
 *** 3DBuffer XP                                                                           ***
 ***                                                                                       ***
 *** auteur     : Jeroen de Vries                                                         ***
 *** e-mail     : jdevries@subdimension.com                                               ***
 *** begin      : september 2000                                                          ***
 *** laatste mod: 19 sep 2001                                                             ***
 *** libraries  : CGAL 2.2 (+ STLport); LEDA                                              ***
 ***                                                                                       ***
 *** versie 1.0                                                                           ***
 ***                                                                                       ***
 *** Doel programma:                                                                      ***
 ***    - genereren driedimensionale afgeronde buffer voor een door de gebruiker         ***
 ***      geselecteerde 3D-triangulatie (ten), en bufferafstand                          ***
 ***                                                                                       ***
 *** (c) 2001 JdV                                                                         ***
 *********************************************************************************************/

/*********************************************************************************************
 *** Voorwerk: declaraties etc                                                            ***
 *********************************************************************************************/

/*===========================================================================================*
 *== Include sectie                                                                        ==*
 *===========================================================================================*/
#include <CGAL/basic.h>
#include <CGAL/Arithmetic_filter.h>
#include <CGAL/leda_real.h>
#include <string>
#include <iostream>
#include <fstream>                                          // voor bestandoperaties
#include <math.h>                                           // voor sqrt
#include <cassert>                                          // voor assert
#include <list>
#include <CGAL/assertions.h>

#include <CGAL/Cartesian.h>

#include <CGAL/Vector_3.h>
#include <CGAL/Plane_3.h>

#include <CGAL/Triangulation_cell_base_3.h>
#include <CGAL/Triangulation_vertex_base_3.h>
#include <CGAL/Triangulation_data_structure_3.h>
#include <CGAL/Triangulation_geom_traits_3.h>
#include <CGAL/Triangulation_3.h>
//#include <CGAL/Delaunay_Triangulation_3.h>

#include <CGAL/squared_distance_3.h>
```

```
/*============================================================================================*
 *== Definities: Triangulatie datastructuur                                               ===*
 *============================================================================================*/

typedef CGAL::Filtered_exact<double,leda_real>  number_type;
typedef CGAL::Cartesian<number_type>            Rep;
typedef CGAL::Triangulation_geom_traits_3<Rep>  Gt;

typedef CGAL::Vector_3<Rep>                     Vector_3;

// ------------------------------
// Afgeleide class van triangulation_vertex_base_3 om een hoekvector in op te kunnen
// slaan:

template <class Gt>
class Hoek_vertex_base : public CGAL::Triangulation_vertex_base_3<Gt>{
private:
       Vector_3 Hoekvector;
public:
       // constructors (zie ook cgal\include\triangulation_vertex_base_3.h)
       Hoek_vertex_base() :
         CGAL::Triangulation_vertex_base_3<Gt>() {}
       Hoek_vertex_base(const Point & p) :
         CGAL::Triangulation_vertex_base_3<Gt>(p) {}
       Hoek_vertex_base(const Point & p, void* c) :
         CGAL::Triangulation_vertex_base_3<Gt>(p, c) {}
       Hoek_vertex_base(void* c) :
          CGAL::Triangulation_vertex_base_3<Gt>(void* c) {}

       // set en get functies
       void set_vector(Vector_3 vecIn) {
              Hoekvector = vecIn;
       }

       Vector_3 get_vector() {return Hoekvector;}
};
// ------------------------------


// DATASTRUCTUUR: Hoekvertexbase (Hvb), Cell Base (Cb)
//   samen in triangulation datastructure (TDS)
typedef Hoek_vertex_base<Gt>                            HVb;
typedef CGAL::Triangulation_cell_base_3<Gt>             Cb;
typedef CGAL::Triangulation_data_structure_3<HVb,Cb>    TDS;

typedef CGAL::Triangulation_3<Gt,TDS>                   Triangulation;
//typedef CGAL::Delaunay_Triangulation_3<Gt,TDS>        Triangulation;

/*============================================================================================*
 *== Definities: Handles                                                                  ==*
 *============================================================================================*/
typedef Triangulation::Cell_handle                      Cell_handle;
typedef Triangulation::Vertex_handle                    Vertex_handle;

/*============================================================================================*
 *== Definities: Iterators(traverse) en circulators(boundary check)                       ==*
 *============================================================================================*/
typedef Triangulation::Edge_iterator                    Edge_iterator;
typedef Triangulation::Vertex_iterator                  Vertex_iterator;
typedef Triangulation::Facet_iterator                   Facet_iterator;
typedef Triangulation::Facet_circulator                 Facet_circulator;


/*============================================================================================*
 *== Definities: Topologische elementen                                                   ==*
 *============================================================================================*/
typedef Triangulation::Vertex                           Vertex;
typedef Triangulation::Cell                             Cell;
typedef Triangulation::Facet                            Facet;
typedef Triangulation::Edge                             Edge;
```

```
/*============================================================================================*
 *== Definities: Geometrische elementen                                                     ==*
 *============================================================================================*/
typedef Gt::Point                                       Point_3;
typedef Gt::Segment                                     Segment_3;
typedef Gt::Triangle                                    Triangle_3;

/*============================================================================================*
 *== Definities: Elementaire 3d objecten                                                    ==*
 *============================================================================================*/
// typedef al gedaan, maar hoort eigenlijk hier:        Vector_3
typedef CGAL::Null_vector                               Null_vector;
typedef CGAL::Origin                                    Origin;
typedef CGAL::Plane_3<Rep>                              Plane_3;
typedef CGAL::Aff_transformation_3<Rep>                 Aff_transformation_3;

/*============================================================================================*
 *== Prototypes                                                                             ==*
 *============================================================================================*/
void Genereer_buffer_3          (const Triangulation&   tenIn,
                                 Triangulation&         tenUit,
                                 number_type            dblStraal);

bool Vind_facets                (const Triangulation&   tenIn,
                                 Edge_iterator          edgIn,
                                 Facet&                 Facet1,
                                 Facet&                 Facet2);

void Bepaal_normalen            (const Triangulation&   tenIn,
                                 const Facet&           Facet1,
                                 const Facet&           Facet2,
                                 Vector_3&              vec1,
                                 Vector_3&              vec2,
                                 Vector_3&              vecGem);

void Verplaats_edges            (const Triangulation&   tenIn,
                                 Triangulation&         tenUit,
                                 number_type            dblStraal,
                                 Edge                   edgIn,
                                 Vector_3&              vec1,
                                 Vector_3&              vec2,
                                 Vector_3&              vecGem);

void Corrigeer_hoekvector       (Vector_3&              vecIn,
                                 number_type            dblStraal);

void Normaliseer_vector         (Vector_3&              vecIn);

void Bepaal_normaalvector       (const Triangulation&   tenIn,
                                 const Facet&           facIn,
                                 Vector_3&              vecInUit);

void Sla_Buffer_Op              (const Triangulation&   tenSave);


/*********************************************************************************************
 *** Hoofdprogramma                                                                        ***
 *********************************************************************************************/
int main(int argc,char* argv[])
{
        Triangulation tenSelectie;

        /* inlezen bestand uit commandline en opgave gewenste bufferafstand (Straal) */
        std::ifstream iFileT(argv[1], std::ios::in);
        if (! iFileT) { std::cout << "Bestand input.ten niet gevonden.";
                        return 1;};

        iFileT >> tenSelectie;
        assert ( tenSelectie.is_valid() );
        std::cout << std::endl << "*** 3D Buffer XP 1.0"
                  << std::endl << "*** Geef straal. Nu! : ";
        number_type Straal;
        std::cin >> Straal;

        /* berekenen buffer */
        Triangulation tenBuffer;
        Genereer_buffer_3(tenSelectie, tenBuffer, Straal);              //functie f1

        /* buffer opslaan */
        cerr << "start Sla_Buffer_Op(tenBuffer)\n";
        Sla_Buffer_Op(tenBuffer);

        return 0;

} // einde main
```

```
/*****************************************************************************************
 *** Functiedefinities                                                                 ***
 *****************************************************************************************/


/*=======================================================================================*
 *==                                                                                   ==*
 *== Functie Genereer_buffer_3                                                         ==*
 *==                                                                                   ==*
 *== Doel  : vormen van een afgeronde buffer rond een TEN,                             ==*
 *== Output: de buffer wordt geretourneerd als TEN.                                    ==*
 *==                                                                                   ==*
 *== Uitgangspunten: - cellen en vertices zijn uniek gedefinieerd,                     ==*
 *==                 - er wordt gebruik gemaakt van zelf afgeleide                     ==*
 *==                   vertex base class.                                              ==*
 *==                                                                                   ==*
 *== JdV 2000 v0.3 f1                                                                 ==*
 *=======================================================================================*/
 void Genereer_buffer_3 (const Triangulation& tenIn,  Triangulation& tenUit,
                         number_type dblStraal){

       // *** Doe voor elke edge van tenIn (lus A):
       // *** verplaats elke edge apart (3x) en houd hoekvectoren bij

       Edge_iterator CurrentEdge = tenIn.finite_edges_begin();
       Edge_iterator EndEdge = tenIn.edges_end();
       for ( ; CurrentEdge != EndEdge; ++CurrentEdge ) {
              Edge edgCurrent = *CurrentEdge;

           // *** Zoek de bijbehorende twee boundary facets
              Facet Facet1;
              Facet Facet2;
              bool blnBoundary = Vind_facets (tenIn, CurrentEdge,Facet1, Facet2);
               if ( blnBoundary == false ) continue; // als edge geen boundary edge is:
                                                     // volgende edge

           // *** Bepaal de twee normaalvectoren en de gemiddelde vector
              Vector_3 vecNormaal1;
              Vector_3 vecNormaal2;
              Vector_3 vecNormaalGem;
              Bepaal_normalen (tenIn, Facet1, Facet2, vecNormaal1,vecNormaal2,
                     vecNormaalGem);

           // *** Verplaats de edge 3 keer en voeg de nieuwe vertices (6 dus) toe aan
           // *** tenUit
              Verplaats_edges (tenIn, tenUit, dblStraal, edgCurrent, vecNormaal1,
                    vecNormaal2, vecNormaalGem);

           // *** Tel de gemiddelde vector op bij de vectorgegevens van de originele
           // *** vertices (vertices zijn gebasseerd op een eigen vertex base met extra
           // *** vector_3)
              if (vecNormaal1.direction() != vecNormaal2.direction())
              {
                 Cell_handle EdgeCell = edgCurrent.first;
                 Vertex_handle Vertex1 = EdgeCell->vertex(edgCurrent.second);
                 Vertex_handle Vertex2 = EdgeCell->vertex(edgCurrent.third);

                 Vector_3 vecHoek1 = Vertex1->get_vector();
                 Vector_3 vecHoek2 = Vertex2->get_vector();

                 vecHoek1 = vecHoek1 + vecNormaalGem;
                 vecHoek2 = vecHoek2 + vecNormaalGem;

                 Vertex1->set_vector(vecHoek1); // hier wordt gebruik gemaakt van
                 Vertex2->set_vector(vecHoek2); // zelfgedef.functie set_vector van eigen
                                                // vertexbase class
              }
              else
              {
              cerr << "twee dezelfde normalen gevonden\n";
              } // einde if else (optellen hoekvectoren)

       }; // Einde lus A

// *** Doe voor elke vertex in tenIn (lus B):
    // *** voeg hoekpunten toe aan de buffer
       Vertex_iterator CurrentVertex = tenIn.finite_vertices_begin();
       Vertex_iterator EndVertex = tenIn.vertices_end();


       for ( ; CurrentVertex != EndVertex; ++ CurrentVertex ) {

       // *** Corrigeer de vector_3 voor de bufferafstand, tel op bij point_3 en voeg
       // *** nieuwe vertex met deze point_3 toe aan tenUit
       // *** Als een vertex in het lichaam ligt (hoekvertex is leeg), dan overslaan
              Vector_3 HoekVector = CurrentVertex->get_vector();
              Null_vector vecNull;
```

```
            if (HoekVector != vecNull) {
                    Corrigeer_hoekvector(HoekVector, dblStraal);
                    Point_3 OorspronkelijkPunt = CurrentVertex->point();
                    Point_3 NieuwPunt = OorspronkelijkPunt + HoekVector;
                    tenUit.insert(NieuwPunt);
            }; // end if

        }; // Einde lus B

} //einde Genereer_buffer_3


/*============================================================================================*
 *==                                                                                        ==*
 *== Functie Vind_facets                                                                    ==*
 *==                                                                                        ==*
 *== Doel  : het vinden van de twee bij een edge behorende boundary facets.                 ==*
 *== Output: als de edge geen boundary edge is, wordt false geretourneerd,                  ==*
 *==         anders wordt true geretourneerd en de twee facets die elk bij een              ==*
 *==         boundary cell horen.                                                           ==*
 *==                                                                                        ==*
 *== JdV 2000 v0.3 f2                                                                      ==*
 *============================================================================================*/
bool Vind_facets (const Triangulation& tenIn, Edge_iterator edgIn,
                  Facet& Facet1, Facet& Facet2){

        Facet_circulator FC = tenIn.incident_facets (*edgIn); // begin bij willek. facet,
        Facet_circulator Klaar(FC);                           // en onthoud deze in Klaar
        bool blnOneindig = false;                             // test op oneindigheid

        do {
            if (tenIn.is_infinite (*FC)) {               // is de facet oneindig?
               blnOneindig = true;                       //   flag omzetten,
               break;                                    //   en einde lus.
            };
        } while (++FC != Klaar);

        if (! blnOneindig) return false;                  // geen oneindige facets: einde.
        else {
               // de oneindige facet is facet met een index van een grenscell
               // de andere oneindige cell is de buurman van de gevonden cell met dezelfde
               // index als de facet
               Cell_handle hndlGrensCell1 = (*FC).first;
               Cell_handle hndlGrensCell2 = hndlGrensCell1->neighbor((*FC).second);

               // de boundary facets liggen tegenover de oneindige vertex in de eerste
               // cell, en tegenover de oneindige vertex in de naburige oneindige cell

               Vertex_handle hndlOneindigeVertex = tenIn.infinite_vertex();

               int idx1OneindigeVertex = hndlGrensCell1->index(hndlOneindigeVertex);
               int idx2OneindigeVertex = hndlGrensCell2->index(hndlOneindigeVertex);
```

```
                Facet1.first = hndlGrensCell1;
                Facet1.second = idx1OneindigeVertex;
                Facet2.first = hndlGrensCell2;
                Facet2.second = idx2OneindigeVertex;
               }
        return true;

} //einde Vind_facets


/*=============================================================================================*
 *==                                                                                         ==*
 *== Functie Bepaal_normalen                                                                 ==*
 *==                                                                                         ==*
 *== Doel: - het vinden van de normaalvectoren (lengte 1) op facet1 en op facet2,            ==*
 *==       - het bepalen van de gemiddelde vector (lengte 1).                                ==*
 *==                                                                                         ==*
 *== JdV 2000 v0.3 f3                                                                       ==*
 *=============================================================================================*/
void Bepaal_normalen (const Triangulation& tenIn,
                      const Facet& Facet1, const Facet& Facet2,
                      Vector_3& vec1, Vector_3& vec2, Vector_3& vecGem){

        Bepaal_normaalvector (tenIn, Facet1, vec1);
        Bepaal_normaalvector (tenIn, Facet2, vec2);

        vecGem = vec1 + vec2;

        Normaliseer_vector (vecGem);

} //einde Bepaal_normalen



/*=============================================================================================*
 *==                                                                                         ==*
 *== Functie Verplaats_edges                                                                 ==*
 *==                                                                                         ==*
 *== Doel  : het verplaatsen van de edge. Oftewel: het verplaatsen van de edge:              ==*
 *==         - in de richting van de 3 vectoren,                                             ==*
 *==         - met afstand dblStraal.                                                        ==*
 *==         Er ontstaan dus 3 nieuwe edges; per vector 1.                                   ==*
 *== Output: De gegenereerde punten worden in een triangulatie toegevoegd.                   ==*
 *==                                                                                         ==*
 *== JdV 2000 v0.3 f4                                                                       ==*
 *=============================================================================================*/
void Verplaats_edges (const Triangulation& tenIn, Triangulation& tenUit,
                      number_type dblStraal, Edge edgIn,
                      Vector_3& vec1, Vector_3& vec2, Vector_3& vecGem){


        cerr << "verplaats edges start\n";
        Segment_3 segEdge = tenIn.segment(edgIn);                 // van edge naar segment

        Vector_3 vecTemp1 = vec1*dblStraal;
        Vector_3 vecTemp2 = vec2*dblStraal;
        Vector_3 vecTempGem = vecGem * dblStraal;


        Vertex_handle Vh;

        std::cout << "---Edge---"<< std::endl;

        // eerste verplaatste edge
        Point_3 S1 = segEdge.source() + vecTemp1;   std::cout << "S1: " << S1 << " ";
        Vh = tenUit.insert(S1);                     std::cout << "inserted" <<  std::endl;
        Point_3 T1 = segEdge.target() + vecTemp1;   std::cout << "T1: " << T1 << " ";
        Vh = tenUit.insert(T1);                     std::cout << "inserted" <<  std::endl;


        // tweede verplaatste edge
        Point_3 S2 = segEdge.source() + vecTemp2;   std::cout << "S2: " << S2 << " ";
        Vh = tenUit.insert(S2);                     std::cout << "inserted" <<  std::endl;
        Point_3 T2 = segEdge.target() + vecTemp2;   std::cout << "T2: " << T2 << " ";
        Vh = tenUit.insert(T2);                     std::cout << "inserted" <<  std::endl;
```

```
        // gemiddelde verplaatste edge
        Point_3 Sg = segEdge.source() + vecTempGem;  std::cout << "Sg: " << Sg << " ";
        Vh = tenUit.insert(Sg);                      std::cout << "inserted" <<  std::endl;
        Point_3 Tg = segEdge.target() + vecTempGem;  std::cout << "Tg: " << Tg << " ";
        Vh = tenUit.insert(Tg);                      std::cout << "inserted" <<  std::endl;


} //einde Verplaats_edges


/*============================================================================================*
 *==                                                                                        ==*
 *== Functie Corrigeer_hoekvector                                                           ==*
 *==                                                                                        ==*
 *== Doel  : de lengte van de vector gelijk aan dblStraal maken.                            ==*
 *==                                                                                        ==*
 *== JdV 2000 v0.3 f5                                                                       ==*
 *============================================================================================*/
void Corrigeer_hoekvector (Vector_3& vecIn, number_type dblStraal){

              Normaliseer_vector (vecIn);
              vecIn = vecIn * dblStraal;

} //einde Corrigeer_hoekvector


/*============================================================================================*
 *==                                                                                        ==*
 *== Functie Normaliseer_vector                                                             ==*
 *==                                                                                        ==*
 *== Doel  : de lengte van een vector gelijk aan 1 maken.                                   ==*
 *==                                                                                        ==*
 *== JdV 2000 v0.3 f6                                                                       ==*
 *============================================================================================*/
void Normaliseer_vector (Vector_3& vecIn){

        Origin Oorsprong;
        Point_3 ptOorsprong(Oorsprong);
        Point_3 Punt = Oorsprong + vecIn;

        number_type dblLengte = CGAL::squared_distance(ptOorsprong, Punt);
        dblLengte = sqrt(dblLengte);

        vecIn = vecIn / dblLengte;

} //einde Normaliseer_vector


/*============================================================================================*
 *==                                                                                        ==*
 *== Functie Bepaal_normaalvector                                                           ==*
 *==                                                                                        ==*
 *== Doel  : de normaalvector (lengte 1) op de face retourneren.                            ==*
 *==         waarbij rekening wordt gehouden met de orientering van de face                 ==*
 *==                                                                                        ==*
 *== JdV 2000 v0.3 f7                                                                       ==*
 *============================================================================================*/
void Bepaal_normaalvector (const Triangulation& tenIn, const Facet& facIn, Vector_3&
vecInUit ){

        Triangle_3 tri = tenIn.triangle (facIn);               // van facet naar driehoek
        Plane_3 suppPlane = tri.supporting_plane();            // van driehoek naar vlak
        int intIndex = facIn.second;

        // het vlak uitwaarts orienteren aan de hand van de index van de oneindige vertex
        // de nummering van vertices (end dus facetindices) is consequent, en is bij index
        // 0 en 2 naar inwaarts gericht (rechtsdraaiend stelsel).

        if ((intIndex == 0) || (intIndex == 2)) suppPlane = suppPlane.opposite();
        Vector_3 vecTemp = suppPlane.orthogonal_vector();
        Normaliseer_vector (vecTemp);
        vecInUit = vecTemp;

} //einde Bepaal_normaalvector


/*============================================================================================*
 *==                                                                                        ==*
 *== Functie Sla_Buffer_Op                                                                  ==*
 *==                                                                                        ==*
 *== Doel  : opslaan van de buffer in twee bestandsformaten:                                ==*
 *==         - .ten : in formaat van CGAL                                                   ==*
 *==         - .raw : raw triangle formaat voor latere conversie naar VRML 2.0 (.wrl)       ==*
```

```
 *==                  hiervoor moeten de vlakken van de ten gevonden worden.           ==*
 *==                                                                                   ==*
 *== JdV 2000 v0.3 f8                                                                 ==*
 *=====================================================================================*/
void Sla_Buffer_Op (const Triangulation& tenSave){

        // opslaan in CGAL formaat
        std::ofstream oFileT("buffer.ten", std::ios::out);
        oFileT << tenSave;

        /* opslaan in RAW formaat. Hierin worden driehoeken opgeslagen per regel. Voor een
         * driehoek ABC ziet zo'n regel er zo uit: A(x) A(y) A(z) B(x) B(y) B(z) C(x) C(y)
         * C(z) <CR> uit de buffer moeten dus de vlakken gehaald worden, vertices bepaald,
         * en opgeslagen.*/

        std::ofstream oFileRaw("buffer.raw", std::ios::out);

        Facet_iterator CurrentFacet = tenSave.finite_facets_begin();
        Facet_iterator EndFacet = tenSave.facets_end();

         // per vlak worden twee zijdes opgeslagen:
        for ( ; CurrentFacet != EndFacet; ++CurrentFacet) {

                Triangle_3 tri = tenSave.triangle(*CurrentFacet); // van face naar triangle
                for ( int i = 0; i < 3 ; i++) {
                        Point_3 p = tri.vertex(i);               // van triangle naar punt
                        oFileRaw << p.x() << " " << p.y() << " " << p.z() << " ";
                }; // einde for
                 oFileRaw << std::endl;

                 // en nog een keer, om visualisatie van beide kanten van een face mogelijk
                 // te maken:
                for ( int j = 2; j > -1 ; j--) {
                        Point_3 p = tri.vertex(j);               // van triangle naar punt
                        oFileRaw << p.x() << " " << p.y() << " " << p.z() << " ";
                }; // einde for
                oFileRaw << std::endl;

        }; // einde for

} //einde Sla_Buffer_Op
```

Bijlage 2: het in- en uitvoerbestand van een buffer voor een kubus

Het programma 3Dbuffer, waarmee een buffer met een gegeven bufferafstand voor een gegeven object gegenereerd wordt, geeft als uitvoer een bestand dat de buffer, gemodelleerd als een Tetrahedral Network, bevat. De invoer is een zelfde soort bestand. Zowel het in- als uitvoerbestand bestaat uit een getal dat de dimensie aangeeft, een aanduiding voor het aantal punten, en de coördinaten van deze punten. Van een object kan eventueel alleen de topologie vastgelegd worden, door de coördinaten weg te laten. Na de coördinaten volgt een getal dat het aantal cellen aanduidt, waarna deze cellen beschreven worden aan de hand van hun hoekpunten. Een vlak dat een deel van de grens van het object is, verbindt toch twee tetraëders: een tetraëder van het lichaam, en een 'oneindige tetraëder', waarvan drie hoekpunten die van het grensvlak zijn, en de vierde een oneindig hoekpunt is. In de beschrijving van de cellen wordt dit oneindige hoekpunt met '0' aangegeven. Tenslotte wordt per tetraëder aangegeven, welke vier tetraëders zijn buren zijn.

In deze bijlage wordt een voorbeeld gegeven van een invoerbestand waarin een kubus gemodelleerd is en een uitvoerbestand dat een buffer voor de kubus bevat. Deze gegenereerde resultaten worden vervolgens vergeleken met berekende resultaten.

Een voorbeeld van een invoerbestand, waarin een kubus is gemodelleerd met grootte 2 is (met later toegevoegd commentaar):

```
3                   dimensie
8                   aantal vertices
0 1 1               coördinaten van de vertices (kunnen weggelaten worden om een zuiver topologische beschrijving
1 1 1               op te slaan)
0 0 1               (cell 3 heeft bijvoorbeeld de coördinaten (2, 2, 3)
1 0 1
1 1 0
0 0 0
1 0 0
0 1 0
18                  aantal cellen (tetraëders)
8 3 1 0             cellen (hier: cel 0 bestaat uit vertices 8, 3, 1 en de oneindige vertex
2 8 1 0
3 2 1 0
5 8 2 0
8 3 2 1
4 5 2 0
3 4 2 0
7 5 4 0
3 7 4 0
5 3 4 2
8 5 6 0
6 5 7 0
8 5 2 3
6 8 0 3
7 6 0 3
7 4 5 3
6 7 5 3
8 6 5 3
2 1 13 4            nabuurschap (hier: de naburige cellen van cell 0 zijn cell 2, cell 1, cell 13 en cell 4)
0 2 3 4
1 0 6 4
1 5 10 12
2 1 0 12
3 6 7 9
5 2 8 9
5 8 11 15
7 6 14 15
6 5 12 15
11 13 3 17
7 14 10 16
9 4 17 3
0 14 17 10
13 8 16 11
9 16 8 7
15 17 14 11
16 12 13 10
```

Als met het programma 3Dbuffer voor deze kubus een driedimensionale buffer met bufferafstand 5 gegenereerd wordt, dan wordt de volgende uitvoerbestand aangemaakt, waarbij omwille van de leesbaarheid de beschrijvingen van de cellen en hun nabuurschap weggelaten zijn:

```
3
56
-2.88675 3.88675 3.88675
-3.53553 1 4.53553
-2.88675 -2.88675 3.88675
-3.53553 0 4.53553
-3.53553 4.53553 1
-2.88675 3.88675 -2.88675
-3.53553 4.53553 0
-5 1 1
3.88675 3.88675 3.88675
0 4.53553 4.53553
1 4.53553 4.53553
0 6 1
4.53553 4.53553 1
3.88675 3.88675 -2.88675
4.53553 4.53553 0
0 4.53553 -3.53553
1 4.53553 -3.53553
0 1 6
1 6 1
0 6 0
1 6 0
3.88675 -2.88675 3.88675
4.53553 1 4.53553
4.53553 0 4.53553
6 1 1
1 -3.53553 4.53553
0 -3.53553 4.53553
1 1 6
1 0 6
0 0 6
3.88675 -2.88675 -2.88675
4.53553 -3.53553 1
4.53553 -3.53553 0
6 0 1
4.53553 1 -3.53553
6 1 0
1 -5 1
4.53553 0 -3.53553
6 0 0
-2.88675 -2.88675 -2.88675
-3.53553 0 -3.53553
1 1 -5
-3.53553 1 -3.53553
0 1 -5
1 -3.53553 -3.53553
-5 1 0
0 -3.53553 -3.53553
1 0 -5
0 0 -5
1 -5 0
-3.53553 -3.53553 1
-3.53553 -3.53553 0
0 -5 1
0 -5 0
-5 0 1
-5 0 0
- vanaf hier beginnen de beschrijvingen van de cellen en van nabuurschap.
```

Er wordt een blijkens de eerste regels een driedimensionaal object gegenereerd met 56 vertices, hetgeen klopt met de theorie: bij elke vertex in een kubus komen immers drie randen bijeen. Vertices worden gegenereerd door elke rand te verplaatsen (3x), deze verplaatsingsvectoren te middelen (3x), en het gemiddelde van deze verplaatsingsvectoren te bepalen (1x). Bij elk oorspronkelijke vertex worden dus 7 nieuwe (buffer)vertices gegenereerd, en de buffer bestaat daardoor uit 7 x 8 = 56 vertices.