Virtuele Maquette

3D-visualisatie met behulp van een database

Naam student	Johan de Koning (0421286)
Datum	15 augustus 2005
Opleiding	1-jarige Master Software Engineering Universiteit van Amsterdam, Hogeschool van Amsterdam en de Vrije Universiteit
Afstudeerdocent	Prof. Dr. D.J.N. van Eijck
Stagebegeleider	Drs. Valik Solórzano Barboza
Opdrachtgever	Geodan S&R

Inhoudsopgave

Inleiding **3**

Dankwoord **5**

Samenvatting **6**

1 Achtergrond en onderzoeksvraag **7**
1.1 Project Virtuele Maquette 7
1.2 Geodan S&R 7
1.3 Onderzoek 3D Visualisatie 7
1.4 Onderzoeksvraag en doelstellingen 9

2 GIS en visualisatie **11**
2.1 Wat is GIS? 11
2.2 Geografische informatie opslag 11
2.3 Visualisatie van 2D naar 3D 12
2.4 3D GIS, de stand van zaken 13
2.5 Opslag van 3D GIS volgens open standaarden 13
2.6 Opslag van 3D gegevens in de huidige praktijk 14

3 Stappenplan **16**

4 Uitvoering **18**
4.1 Keuze database systeem 18
4.2 Opslag 3D gegevens in een database 19
4.3 Selecteren en visualiseren van 3D gegevens 20
4.4 Opslag van kleuren en texturen in een database 24
4.5 Selecteren en visualiseren van kleuren en texturen 26
4.6 Importeren van bestaande 3D gegevens 28
4.7 Koppelen attribuut informatie 28

5 Resultaten **30**

6 Evaluatie **32**
6.1 Inzet database voor 3D gegevensopslag 32
6.2 PostGIS database 32
6.3 Opslagmethode 32
6.4 Selectiemethode 33
6.5 Visualisatie met OpenSceneGraph 33

7 Referenties **34**

Bijlage 1: Testresultaten PostGIS **36**

Bijlage 2: Managed C++ **38**

Inleiding

John Snow maakte rond 1850 gebruik van geografische informatie om zo de uitbraak van cholera te achterhalen. Dit staat bekend als het begin van het toepassen van geografische informatie voor analyse doeleinde. Tegenwoordig zijn systemen die gebruik maken van geografische informatie niet meer weg te denken.

Voor de analyse die John Snow uitvoerde maakte hij gebruik van eigen getekende kaartschetsen. Met de opkomst van de computer kwam hier verandering in. Kaarten werden door de computer getekend met daarop de verschillende informatie kenmerken weergegeven. Zelfs analyses worden hedendaags met de computer uitgevoerd.

Het gebruik van geografische informatie blijkt de laatste jaren in een stroomversnelling te zijn geraakt. Bedrijven kunnen met behulp van geografische informatie hun logistiek verbeteren, positie van medewerkers die onderweg zijn registreren en hebben sneller inzicht in sociaal-economische gegevens. Maar ook particulieren maken steeds meer gebruik van geografische informatie in de vorm van navigatiesystemen.

Zoals de overstap van handgemaakte kaarten naar computer gegeneerde kaarten heeft plaatsgevonden, zo zal er ook een stap worden gemaakt van tweedimensionale naar drie dimensionale geografische informatie. Deze derde dimensie zal dan ingezet kunnen worden voor complexere analyses en voor realistischere visualisatie.

Opslag van driedimensionale geografische gegevens is complexer dan de twee dimensionale geografische gegevens doordat er veel meer aspecten als kijkvrijheid, hoger detailniveau en het gebruik van texturen bij komen kijken. Daarnaast zijn er voor de opslag nog geen standaarden gedefinieerd. In deze scriptie wordt daarom onderzoek gedaan of de inzet van een huidig databasesysteem geschikt is voor de opslag van drie dimensionale geografische gegevens.

De opbouw van deze scriptie is als volgt. In hoofdstuk 1 wordt de context beschreven waarin dit onderzoek plaatsvindt. De inzet van driedimensionale modellen wordt nu al toegepast met veelal handgemaakte modellen, gemaakt in driedimensionale grafische softwareapplicaties. In dit hoofdstuk worden ook de beperkingen beschreven die aan het gebruik van deze handgemaakte modellen zitten en waarom de inzet van een database dit kan oplossen.

Hoofdstuk 2 bevat een literatuuronderzoek met betrekking tot geografische informatie systemen en de derde dimensie. Deze literatuurstudie wordt als basis genomen voor het onderzoek waarvan de stappen in hoofdstuk 3 worden beschreven. Uitvoering van dit onderzoek wordt beschreven in hoofdstuk 4.

De resultaten die behaald zijn met de inzet van een database voor de opslag van driedimensionale geografische gegevens is terug te vinden in hoofdstuk 5. In dit hoofdstuk worden ook aanbevelingen gegeven, om het selectieproces op de database te versnellen. Hierdoor wordt het real-time ophalen van gegevens tijdens een fly-over mogelijk.

Deze scriptie wordt afgesloten met een evaluatie op het uitgevoerde onderzoek. Deze evaluatie is terug te lezen in hoofdstuk 6.

In bijlage 1 worden de testresultaten beschreven met betrekking tot het ophalen van de gegevens uit een PostGIS database. Bijlage 2 beschrijft de mogelijkheid hoe bestaande C++ code gebruikt kan worden binnen de managed Microsoft .Net programmeertalen als C#. Voor het onderzoek, naar de inzet van een database voor driedimensionale gegevensopslag, is gebruik gemaakt van de OpenSceneGraph bibliotheek de gegevens te visualiseren. OpenSceneGraph is geschreven is C++ waardoor geheugen toewijzigen door de programmeur beheerd moeten worden. Met managed programmeertalen wordt geheugenbeheer uit handen genomen van de programmeur doordat dit automatisch geregeld wordt. Geheugenbeheer wordt binnen C++ als complex gezien. Om de OpenSceneGraph bibliotheek binnen bijvoorbeeld C# te kunnen gebruiken is dit opzichzelfstaande onderzoek uitgevoerd en worden de eerste behaalde resultaten daarvan beschreven in deze bijlage.

Dankwoord

Ook al is het schrijven van deze scriptie een individuele taak, de hulp en ondersteuning van verschillende mensen draagt sterk bij aan het uiteindelijke resultaat. Een resultaat waar ik met voldoening naar terugkijk. Niet alleen naar hetgeen wat in deze scriptie is beschreven maar ook naar de ervaringen die ik heb opgedaan tijdens de uitvoering van het onderzoek. Daarvoor wil ik dan ook een aantal mensen, in willekeurige volgorde, bedanken:

Ik wil mijn begeleider *Valik Solórzano Barboza* bedanken voor de begeleiding tijdens de uitvoering van mijn stageonderzoek binnen Geodan S&R. Vooral de enthousiasme reacties die ik kreeg bij het behalen van nieuwe resultaten heeft een positief effect gehad tijdens de uitvoering.

Jan van Eijck wil ik bedanken voor hetgeen hij heeft gedaan als begeleider vanuit de opleiding. De opmerkingen over de opbouw van de scriptie waardoor het een en ander is verschoven dragen volgens mijzelf erg goed bij aan een heldere structuur van deze scriptie.

Mijn vriendin *Monique de Haan* bedank ik voor de steun die ze mij gegeven heeft tijdens de gehele opleiding. Daarnaast heb ik vele malen adviezen gekregen over de inhoud en opbouw van de scriptie omdat ik er soms niet meer uit kwam. Dan te bedenken dat ik regelmatig discussies heb gehad over de inhoud terwijl ze geen informatica opleiding heeft gedaan.

Ella Steenman heeft de moeite genomen deze scriptie door te lezen en geprobeerd zoveel mogelijk spel- en grammaticafouten er uit te halen. Tijdens het herlezen van je eigen teksten lees je vaak over je fouten heen. Deze scriptie doorlezen is een behoorlijke klus geweest en daarom wil ik ook haar bedanken.

Uiteraard bedankt ik ook mijn collega's bij *Geodan* en dan met de nadruk op de mensen die werken bij *Geodan S&R*. Zij zorgden voor een gezellige en ontspannen sfeer tijdens mijn stage. Een aantal heb ik ook regelmatig om advies kunnen vragen op technisch gebied voor dit onderzoek.

Tot slot bedank ik de mensen die ik vergeten ben hier te noemen maar ook direct of indirect hebben bijgedragen aan dit eindresultaat.

Samenvatting

Het gebruik van driedimensionale modellen is een nieuwe stap binnen de wereld van Geografische Informatie Systemen (GIS). Het toevoegen van de derde dimensie maakt visualisatie realistischer, maar kan ook bijdragen aan het uitvoeren van analyses waarbij ruimte een rol speelt, bijvoorbeeld verspreiding van geluid.

Het ontbreekt aan een goede methode voor de opslag van deze driedimensionale gegevens. Modellen worden nu gemaakt binnen grafische ontwerpapplicaties als 3D Studio Max en Blender. Dit resulteert in een drietal beperkingen: geen selectiemogelijkheden, complexiteit bij beheersbaarheid van het model en het niet kunnen koppelen van attribuutinformatie wat bij tweedimensionale geografische informatiesystemen wel het geval is.

Voor de opslag van tweedimensionale geografische informatie wordt veelal gebruik gemaakt van een object gerelateerd database managementsysteem (ORDBMS). Daarvoor is een specificatie opgesteld door het Open Geospatial Consortium (OGC) waarin een database schema en functies beschreven zijn die door een ORDBMS geïmplementeerd dienen te worden. PostGIS (PostgreSQL met een uitbreiding voor geografische datatypes) is zo'n ORDBMS die deze tweedimensionale specificatie nastreeft. Een specificatie voor driedimensionale geografische gegevens opslag is op dit moment nog in ontwikkeling, maar PostGIS biedt nu al de mogelijkheid om de z-coördinaat (hoogte) van de geografische informatie op te slaan. Functionaliteit zoals die op tweedimensionale gegevens uitgevoerd kan worden ontbreekt hierbij, maar selectie functionaliteit op driedimensionale geografische informatie kan wel worden gerealiseerd.

In het onderzoek dat beschreven wordt in deze scriptie wordt het gebruik van PostGIS toegepast in combinatie met de visualisatie bibliotheek OpenSceneGraph. Door het inzetten van een databasesysteem voor de opslag van driedimensionale geografische informatie wordt aangetoond dat deze opslag nu al als een goede basis kan dienen voor verdere ontwikkelingen op het gebied van driedimensionale visualisatie in combinatie met GIS. Het onderzoek toont aan dat de drie gestelde beperkingen met behulp van PostGIS opgelost kunnen worden.

Met de komst van een door de OGC opgestelde specificatie, zal de inzet van een databasesysteem voor driedimensionale geografische informatie alleen maar worden versterkt. Daarnaast zal, zoals in de resultaten van dit onderzoek naar boven is gekomen, gewerkt moeten worden aan de snelheidsbeperking bij het ophalen van de gegevens. Aanbrengen van selectieniveaus kan dit probleem verhelpen waardoor on-the-fly weergave van een onbeperkt groot model mogelijk is. Buiten de GIS-wereld kan dit ook interessant zijn voor de computerspellenindustrie die op dit moment ook beperkingen hebben betreffende de omvang van het model waarin gespeeld wordt.

1 Achtergrond en onderzoeksvraag

In dit hoofdstuk wordt de achtergrond van het project Virtuele Maquette toegelicht. Vervolgens zal de onderzoeksvraag worden behandeld die in de rest van deze scriptie centraal staat.

1.1 Project Virtuele Maquette

Om op de vastgoedbeurs MIPIM in Cannes, Nederland als aantrekkelijk investeringsland voor (toekomstige) ondernemers te plaatsen, is het belangrijk dat het als klein land kan concurreren tegen de grote Europese steden als Berlijn, Parijs, Barcelona en London. Goede samenwerking kan daarbij de positie verstevigen. Vandaar dat de vier grote Randstadsteden Amsterdam, Rotterdam, Den Haag en Utrecht zich verenigd hebben in de groep Holland 4 Forces (H4F) en dat de behoefte is ontstaan om de presentaties te integreren.

Door inzet van de nieuwste technieken kan de houten statische maquette vervangen worden door dynamische presentaties, zoals een driedimensionale vlucht over de omgeving waarin de investeerder interesse heeft. Een bijkomend voordeel is dat belangrijke locaties en informatie van de omgeving die voor de investeerder van belang zijn uitgelicht kunnen worden. Voorbeelden hiervan zijn de afstand tot de dichtstbijzijnde haven of het vliegveld, gemiddeld gezinsinkomen in een gebied en positie van concurrenten. Deze gegevens worden bepaald door een vooraf opgesteld profiel.

1.2 Geodan S&R

Geodan System & Research (S&R), opgericht in 1995, is een van de bedrijven waaruit Geodan bestaat. Geodan S&R is verantwoordelijk voor het onderzoek naar, en onderhoud van, softwarecomponenten en de technische infrastructuur voor de Geodan groep. Voor de ontwikkeling van softwarecomponenten wordt veel gebruik gemaakt van open-source producten.

Bij de diverse projecten binnen Geodan is de rode draad het gebruik van geografische informatie en geografische informatie systemen (GIS) [1]. Binnen het project Virtuele Maquette voert Geodan S&R onderzoek uit naar de verschillende technieken die ingezet kunnen worden bij de H4F presentatie. Daarbij wordt het gebruik van geografische informatie op verschillende manieren toegepast. Aan de hand van de onderzoeksresultaten wordt de uiteindelijke software ontwikkeld.

Het onderzoek vindt plaats bij Geodan S&R en richt zich op de technieken voor gebruik van (real-time) 3D visualisatie.

1.3 Onderzoek 3D Visualisatie

Bij aanvang van het onderzoek zijn voor de driedimensionale visualisatie van de vluchtpresentatie al verschillende resultaten behaald. De behaalde resultaten worden als basis genomen voor het onderzoek en worden in deze paragraaf beschreven.

Als basis voor de driedimensionale visualisatie wordt gebruik gemaakt van geografische informatie die beschikbaar is van de omgeving zoals gebouwen en luchtfoto's. Deze informatie is echter beschikbaar in twee dimensies omdat er nog geen driedimensionale modellen van de verschillende objecten in Nederland beschikbaar zijn.

Om toch de derde dimensie toe te voegen, is er gebruik gemaakt van de (gemiddelde) hoogte gegevens van de objecten zoals die geregistreerd zijn in onder andere het gegevensmodel Top10NL [2], opgesteld door het Kadaster. Dit resulteert in een model waarvan de objecten driedimensionaal zijn, maar waarbij de details veelal missen en het object enkel een ruwe omvang bevat.

Deze basis objecten kunnen verbeterd worden door kleuren en texturen toe te voegen. Het plaatsen van foto's op de objecten maakt het geheel realistischer. Een andere mogelijkheid, als er geen foto's beschikbaar zijn, is het (her)gebruiken van standaard elementen zoals ramen en deuren.

De meest gedetailleerde mogelijkheid is het handmatig maken van driedimensionale modellen in softwareapplicaties als 3D Studio Max [3] en Blender [4]. Hierdoor zijn de meeste details te bereiken qua vorm en texturen. Het maken van driedimensionale modellen op deze wijze is erg arbeidsintensief. Daarnaast blijkt het gebruik van de opgehoogde modellen met (fotografische) texturen voldoende te zijn om een goede indruk te kunnen geven van de omgeving.

Met behulp van een 'viewer' kunnen de modellen binnen de omgeving worden bekeken en kan er overheen worden gevlogen. Voor deze 'viewer' zijn twee alternatieve softwarepakketten vergeleken, Virtools [5] en Open Scene Graph (OSG) [6] Viewer. Virtools is een commerciële driedimensionale omgeving die oorspronkelijk voor computerspellen is ontwikkeld. De Open Scene Graph Viewer, met de daarbij behorende bibliotheken, is open-source software.

Voor de verdere ontwikkeling van het project is gekozen voor OSG. Dit is een keuze van het bedrijf dat als volgt werd gemotiveerd: ondersteuning van een groot scala van verschillende bestandsformaten; aanwezigheid van een krachtige level of details management (LOD) die bij bijvoorbeeld het gebruik van luchtfoto's zorgt voor een vloeiende overgang van de details; en de openheid van de software vanwege het feit dat het open-source is. Ook in de literatuur wordt de inzet van OSG genoemd bij de driedimensionale visualisatie van geografische informatie.

De manier waarop nu visualisatie plaatsvindt, door gebruik te maken van een volledig vooraf opgesteld model, brengt een aantal beperkingen met zich mee. Bij het weergeven van het model wordt eerst het gehele model ingeladen, dit met uitzondering van eventuele objecten die gebruik maken van de LOD-manager, zoals de luchtfoto's voor de ondergrond die pas in een gewenst detail worden getoond op het moment dat ze nodig zijn. Een model kan een zeer grote hoeveelheid aan gegevens bevatten, vooral op het moment dat er een vlucht van bijvoorbeeld Amsterdam naar Den Haag gemaakt gaat worden. Het ophalen van het model vereist op deze manier veel van de hardware. Daarnaast is een groot deel van het model niet nodig, wat daardoor overbodig wordt ingeladen en voor een vertraging kan zorgen.

Naast de selectiebeperking op het model, wordt ook het beheren van het model bemoeilijkt. Op het moment dat een model aangepast dient te worden vindt dit handmatig plaats binnen de grafische softwareapplicaties als 3D Studio Max of Blender. Modellen van gebouwen die door externe partijen worden gemaakt dienen op deze manier stuk voor stuk op de juiste plek neergezet te worden. Het beheren van het model is tijdrovend en het combineren van verschillende modellen is niet mogelijk.

De derde beperking heeft betrekking op de mogelijkheid om attribuutinformatie te tonen. Buiten de geografische objecten is het van belang dat er extra informatie getoond kan worden over objecten of de omgeving. Bijvoorbeeld het belichten van overheidsgebouwen en het weergeven van gemiddeld huishoudinkomen per gebied kan niet plaatsvinden als er geen relatie bestaat tussen het model en de attribuutinformatie. Gebruik van een met de handgemaakt model bevat deze informatie niet.

1.4 Onderzoeksvraag en doelstellingen

Als er in meer detail wordt gekeken naar het onderzoek dan staat hierbij de volgende vraag centraal:

"Kan de inzet van een database voor de opslag van driedimensionale geografische informatie de huidige modelbeperkingen als het ontbreken van selectiemogelijkheden, de complexe beheersbaarheid en het ontbreken van attribuutinformatie oplossen?"

Het inzetten van een database kan bijdragen aan een oplossing voor de gestelde beperkingen. Een databasesysteem maakt het mogelijk om een selectie uit te voeren op de beschikbare gegevens. Gegevens in een database kunnen als verschillende records opgeslagen worden waardoor de beheersbaarheid makkelijker wordt. Daarbij kunnen gegevens van verschillende bronnen geïmporteerd worden in een databasesysteem waardoor aanpassingen aan het model door meerdere partijen kan plaatsvinden. Tot slot biedt een databasesysteem de mogelijkheid tot het maken van relaties tussen de gegevens. Hierdoor is het mogelijk om attribuutinformatie aan het model te koppelen.

Op dit moment zijn er, zoals in de literatuur ook genoemd wordt, geen databasesystemen beschikbaar die functionaliteiten bieden voor driedimensionale gegevens, zoals de afstand tussen twee objecten waarbij de z-coördinaat (hoogte) wordt meegenomen. Wel is het mogelijk om in de huidige databasesystemen de driedimensionale gegevens op te slaan. Voor het onderzoek wordt er vanuit gegaan dat dit voldoende is.

Het onderzoek van de inzet van een database voor de opslag van driedimensionale geografische informatie resulteert in een aantal deelvragen.

- Welk databasesysteem is geschikt voor de opslag van de driedimensionale geografische gegevens?
- Hoe dienen de gegevens opgeslagen te worden in het databasesysteem, zoals de driedimensionale geografische gegevens, de texturen en de attribuutinformatie?
- Hoe kunnen de gegevens geselecteerd, opgevraagd en aangeboden worden voor de OSG viewer?
- Hoe kunnen bestaande objecten, zoals de objectgegevens en texturen, worden geïmporteerd in het databasesysteem zodat deze binnen het model beschikbaar zijn?

Doelstellingen van het onderzoek betreffen het opslaan van de geografische gegevens in een databasesysteem en het vervolgens opvragen van deze gegevens en het aanbieden voor OSG. Een verdere specificatie bevindt zich in hoofdstuk 3.

Het koppelen van attribuutinformatie en het importeren van bestaande modellen zullen wel behandeld worden, maar het implementeren hiervan binnen een prototype applicatie wordt niet gezien als een doelstelling tijdens dit onderzoek.

Om het geheel in een context te plaatsen volgt eerst een verslag van een literatuuronderzoek.

2 GIS en visualisatie

2.1 Wat is GIS?

Het toepassen van geografische informatie voor analyse gaat terug naar rond 1850 toen John Snow [1] met gebruik van een kaart kon achterhalen dat cholera verspreid werd door middel van de verschillende waterpompen in de stad Londen. Tegenwoordig wordt bij de inzet van geografische informatie een direct verband gemaakt met Geografische Informatie Systemen (GIS). [7] definieert een geografische informatie systeem als een computersysteem dat een relatie maakt tussen geografische informatie (*waar de dingen zijn*) met omschrijvende informatie (*wat de dingen zijn*). De inzet van computersystemen brengt extra mogelijkheden met zich mee. Geografische informatie kan eenvoudiger ingedeeld worden per thema of kenmerk van een kaart waardoor er verschillende kaartlagen ontstaan. Selectie op deze kaartlagen en het combineren van kaartlagen zorgt ervoor dat een bepaalde hoeveelheid aan informatie over een bepaald gebied getoond kan worden.

Buiten het indelen van geografische informatie in thema's en of kenmerken gaat GIS verder. [8] specificeert de functies en taken van GIS als gegevensopslag, gegevensstructurering, gegevensmanipulatie, gegevensanalyse en gegevenspresentatie. GIS heeft zijn kracht bewezen op het gebied van gegevensanalyse bij bijvoorbeeld analyse van epidemie-uitbraken, logistieke vraagstukken en het bepalen van toekomstige bestemmingsplannen.

In het onderzoek dat centraal staat in deze scriptie wordt gegevensanalyse verder buiten beschouwing gelaten en wordt er ingegaan op de gegevensopslag en de presentatie van geografische informatie.

2.2 Geografische informatie opslag

Geografische informatie heeft bij de opslag als belangrijkste voorwaarde dat er een locatie aan de gegevens gekoppeld is. Deze locatie kan een GPS locatie zijn, maar kan ook gerelateerd zijn aan een x en y coördinaat, zolang er maar een geografische referentie bekend is.

Voor het opslaan van geografische informatie zijn er twee verschillende methodes beschikbaar: vector en raster [1]. Bij de vector methode wordt de geografische informatie beschreven door gebruik te maken van punten, lijnen en polygonen. Bij een raster wordt er een vlak opgedeeld in cellen. Aan een raster of vector kunnen verschillende kenmerken worden gekoppeld.

Het voordeel van een raster is dat het gebruik ervan eenvoudig is. Nadeel is echter dat de nauwkeurigheid afhangt van de gekozen rasterschaal. Door het gebruik van vectoren kan de beschrijving van de geografische informatie nauwkeurig plaatsvinden maar is complexer. Rasters worden voornamelijk gebruikt in combinatie met (lucht)foto's.

De huidige GIS-systemen maken op dit moment voornamelijk gebruik van tweedimensionale geografische datatypes, maar coördinaten van driedimensionale geografische types worden wel voor opslag geaccepteerd. [9]

2.3 Visualisatie van 2D naar 3D

Voor de visualisatie van tweedimensionale geografische gegevens naar driedimensionale geografische gegevens komen er extra visualisatie-eigenschappen aan het licht. In [10] wordt een opsomming gegeven van kwesties die betrekking hebben op driedimensionale visualisatie:

- Driedimensionale objecten moeten ondersteuning bieden aan kleur, transparantie en generieke of fotorealistische texturen. Hierdoor kan een driedimensionaal object meerdere weergaven hebben afhankelijk van de applicatie en context waarbinnen het object wordt gebruikt;
- Uitzichtpunt informatie, zoals positie en hoek, voor gebruik binnen de visualisatie applicatie om zo de ideale positie te specificeren voor het bekijken van bepaalde objecten;
- Belichting kan binnen een driedimensionale omgeving zorgen voor extra detail als schaduw en sfeer;
- Animaties en simulaties binnen de visualisatie applicatie, wat betrekking heeft op het verloop van de objecten. Hierdoor kunnen tijdlijnen worden gevisualiseerd.

Aan de bestaande GIS-producten met tweedimensionale objecten wordt visualisatie van driedimensionale objecten toegevoegd. Volgens [10] worden er echter door deze wijze tekortkomingen gerealiseerd doordat de database geen of beperkte ondersteuning biedt aan driedimensionale objecten en object texturen:

- Model complexiteit: in werkelijkheid bestaande driedimensionale objecten zijn vele malen complexer dan tweedimensionale objecten. Een driedimensionaal object kan uit vele verschillende driedimensionale elementen bestaan die ieder een eigen semantiek hebben;
- Meerdere resoluties en representaties: ten opzichten van een tweedimensionale omgeving zijn er in een driedimensionale omgeving meerdere representaties en resoluties aanwezig. Het detailniveau van een object speelt een belangrijke rol;
- Kijkvrijheid en dynamiek: binnen een driedimensionale omgeving is er veel meer vrijheid met betrekking tot de positie van het gezichtspunt. Ook kunnen er in een driedimensionale omgeving zaken een rol gaan spelen als het binnen gaan van een gebouw of onder een brug doorgaan;
- Presentatie: tweedimensionale presentaties van GIS maken gebruik van symbolen of beschrijvingen die in een driedimensionale omgeving op een andere wijze getoond moeten worden.

Voor een realistische driedimensionale omgeving wordt volgens [9] door gebruikers een voorkeur gegeven aan het gebruik van fotorealistische texturen.

In dit onderzoek ligt de nadruk op het verkrijgen en opslaan van driedimensionale objecten. Aspecten als belichting en kijkvrijheid zijn hierbij niet van directe invloed omdat deze uiteindelijk door de visualisatie applicatie afgehandeld dienen te worden.

2.4 3D GIS, de stand van zaken

Als er wordt gekeken naar de huidige stand van zaken met betrekking tot het gebruik van driedimensionale objecten in een database, dan is vast te stellen dat er nog geen databasesystemen zijn die ondersteuning bieden voor het gebruik van driedimensionale objecten [11]. In de huidige database systemen die ondersteuning bieden voor geografische datatypes kan wel een z-coördinaat meegegeven worden. Daarnaast bieden de databasesystemen beperkte ondersteuning met betrekking tot operaties en functies in combinatie met driedimensionale objecten.

Beperkte ondersteuning van driedimensionale objecten in de huidige databasesystemen komt volgens [11] doordat bedrijven wachten op de open standaard specificatie voor driedimensionale objecten waaraan de Open Geospatial Consortium (OGC) [22] op dit moment nog werkt. Daarnaast is de vraag naar driedimensionale gegevensvoorzieningen nog te beperkt maar komt wel steeds meer in opmars.

Buiten het geografische model, waarvoor de specificaties nog door de OGC worden opgesteld, zal er ook een topologisch model moeten komen voor de relaties tussen driedimensionale objecten [9]. De daarop volgende stap is de implementatie van Constructive Solid Modeling waarbij driedimensionale objecten uit driedimensionale primitieven bestaan en doormiddel van boolean operators samengevoegd kunnen worden.

2.5 Opslag van 3D GIS volgens open standaarden

De OGC stelt standaarden op die gebruikt kunnen worden voor implementatie binnen GIS. Voor het gebruik van tweedimensionale geografische gegevens binnen een database heeft het OGC de Simple Spatial Features for SQL [12] specificatie opgesteld. Deze specificatie definieert een SQL schema voor het opslaan, selecteren,verkrijgen en bijwerken van eenvoudige geografische objecten. Database systemen als Oracle Spatial [13] en PostGIS [14] hebben deze specificatie geïmplementeerd.

Ontwikkeling van de specificaties voor driedimensionale objecten is bij de OGC in volle gang. Op dit moment is er een abstract model beschikbaar, Feature Geometry [15], voor implementatie van driedimensionale objecten. Deze open standaard wordt opgesteld in samenwerking met de International Standard Organization (ISO 19107) [16]. Het abstracte model is op dit moment nog niet geschikt voor implementatie. De specificatie bevat buiten de verschillende driedimensionale gegevenstypes diverse functies die geschikt zijn voor analyse mogelijkheden (zoals lengte en inhoud).

Omdat deze specificatie nog enige tijd op zich kan laten wachten is implementatie hiervan wel mogelijk, maar kan dit resulteren in het volledig herschrijven van een systeem op het moment dat de specificatie officieel is. Daarnaast wordt in [11] beschreven dat de huidige databasesystemen voor geografische gegevens beperkte ondersteuning bieden voor de opslag van driedimensionale gegevens waarbij de z-coördinaat meegenomen kan worden. Voor de uitvoer van dit onderzoek lijkt dit voldoende te zijn omdat er geen analytische activiteiten op de gegevens worden uitgevoerd. Wel wordt er gekozen voor een databasesysteem dat op dit moment gebruikt maakt van de specificatie voor tweedimensionale geografische gegevens. Dit om in de toekomst een overstap naar de specificatie voor de driedimensionale geografische gegevens te kunnen maken.

2.6 Opslag van 3D gegevens in de huidige praktijk

Voor de opslag van driedimensionale gegevens binnen een database is onderzoek gedaan of huidige database systemen hiervoor geschikt zijn. In [17] worden verschillende mogelijkheden beschreven voor de opslag van driedimensionale objecten. Hierbij wordt Oracle Spatial als voorbeeld genomen. Verderop in het onderzoek zal overigens gekozen worden voor PostGIS dat dezelfde functionaliteit biedt als Oracle Spatial maar dat ten opzichte van Oracle Spatial open source is. Daarnaast streeft PostGIS volledig de specificaties na die door de OGC zijn opgesteld.

De eerste optie is het gebruik van twee tabellen: “body” en “face”. De “body” tabel bevat records met verwijzingen naar de verschillende vlakken van een driedimensionaal object. In de “face” tabel staan de verschillende vlakken beschreven als ruimtelijke polygonen. Dit resulteert voor een kubus tot de databaserecords die in Tabel 1 zijn weergegeven

BODY table	
BID	*FID*
1	1
1	2
1	3
1	4
1	5
1	6

FACE table	
FID	*sdo_ordinate_array*
1 (lower face)	(x4,y4,z4, ,x3,y3,z3, x2,y2,z2, x1,y1,z1, x4,y4,z4)
2 (side 1)	(x3,y3,z3, ,x4,y4,z4, x8,y8,z8, x7,y7,z7, x3,y3,z3)
3 (side 2)	(x4,y4,z4, ,x1,y1,z1, x5,y5,z5, x8,y8,z8, x4,y4,z4)
4 (side 3)	(x1,y1,z1, ,x2,y2,z2, x6,y6,z6, z5,y5,z5, x1,y1,z1)
5 (side 4)	(x3,y3,z3, ,x2,y2,z2, x6,y6,z6, z7,y7,z7, x3,y3,z3)
6 (upper face)	(x5,y5,z5, ,x6,y6,z6, x7,y7,z7, z8,y8,z8, x5,y5,z5)

Tabel 1: body en face tabellen voor het opslaan van een kubus object

Een andere mogelijkheid tot het opslaan van driedimensionale objecten is door de het object op te slaan in een record, weergegeven in tabel Tabel 2.

BODY table	
Body id	Geometry
1	SDO_GEOMETRY(3007, -- 3007 indicates a 3D multipolygon NULL, NULL, SDO_ELEM_INFO_ARRAY(-- the offset of the polygons is specified 1, 1003, 1, 16, 1003, 1, 31, 1003, 1, 46, 1003, 1, 61, 1003, 1, 76, 1003, 1), SDO_ORDINATE_ARRAY(x4,y4,z4, ,x3,y3,z3, x2,y2,z2, x1,y1,z1, x4,y4,z4, --end of 1st (lower) polygon x3,y3,z3, ,x4,y4,z4, x8,y8,z8, x7,y7,z7, x3,y3,z3, x4,y4,z4, ,x1,y1,z1, x5,y5,z5, x8,y8,z8, x4,y4,z4, x1,y1,z1, ,x2,y2,z2, x6,y6,z6, z5,y5,z5, x1,y1,z1, x3,y3,z3, ,x2,y2,z2, x6,y6,z6, z7,y7,z7, x3,y3,z3, x5,y5,z5, ,x6,y6,z6, x7,y7,z7, z8,y8,z8, x5,y5,z5 --end of last (upper) polygon))

Tabel 2: 3D object opslag binnen een database record

Het opslaan van een object in een record heeft als voordeel dat het uitvoeren van een selectie in verhouding met de losse records sneller is. Losse vlakken biedt extra mogelijkheden tot het maken van relaties met texturen. Beide implementaties hebben als nadeel dat er een overhead aan gegevens wordt opgeslagen. De hoeken van een kubus worden maar liefst drie keer opgeslagen.

In [18] wordt er een implementatie gemaakt die gebaseerd is op een echt driedimensionale object: de polyhedron primitief. Hierdoor wordt de overhead voorkomen doordat de datatypes binnen de database uitgebreid worden. Extra mogelijkheid is het opstellen van functies die voor analyse doeleinde gebruikt kunnen worden zoals het berekenen van de inhoud. De gekozen primitief heeft echter beperkingen en is niet gebaseerd op een standaard.

De verschillende opslag methoden zullen tijdens het onderzoek worden bekeken en er zal afhankelijk van een aantal factoren voor een van de implementaties worden gekozen. Binnen het onderzoek ligt de prioriteit op het opslaan van driedimensionale geografische gegevens en het verkrijgen ervan. Het uitvoeren van bijvoorbeeld analyses op een driedimensionaal model is niet het doel van dit onderzoek.

3 Stappenplan

Voor het onderzoek en de daarbij onderliggende deelgebieden zijn er een aantal stappen opgesteld die uitgevoerd dienen te worden. Deze stappen moeten bijdragen aan het behalen van de opgestelde doelstellingen, waarbij ruimtelijke informatie zal worden gerepresenteerd in de vorm van ruimtelijke polygonen die opgeslagen zijn binnen een databasesysteem. De volgorde van de stappen, waarbij eerst naar de opslag en het opvragen van de polygonen wordt gekeken en vervolgens pas de kleuren en texturen worden meegenomen, is gekozen omdat dit bijdraagt aan een heldere opzet van deze scriptie.

- *Onderzoek en kiezen van een databasesysteem.* Voor de opslag van de driedimensionale geografische gegevens binnen een databasesysteem moet een geschikt databasesysteem worden gekozen. Dit databasesysteem moet de ondersteuning bieden aan driedimensionale geografische datatypes en functionaliteit om hierop een selectie uit te voeren.

- *Opslag van geografische objecten in een databasesysteem.* In deze stap wordt beschreven hoe de daadwerkelijke implementatie plaatsvindt voor de opslag van de driedimensionale geografische gegevens. In hoofdstuk 2 werden de verschillende manieren voor het opslaan van driedimensionale objecten beschreven. Tijdens de uitvoering wordt de keuze van opslag bepaald aan de hand van deze verschillende mogelijkheden.

- *Opvragen en aanbieden voor visualisatie van geografische objecten verkregen uit een databasesysteem.* Op de opgeslagen driedimensionale geografische gegevens moet een selectie kunnen worden uitgevoerd. Deze gegevens moeten vervolgens omgezet worden, zodat het kan worden aangeboden voor visualisatie binnen OpenSceneGraph (OSG). De OSG bibliotheek biedt een architectuur die uitbreidbaar is met behulp van plugins. Tijdens de uitvoering zal deze architectuur meegenomen worden.

- *Opslag van kleuren en texturen voor de geografische objecten in een databasesysteem.* Om een betere indruk te kunnen geven van de omgeving is het gewenst om kleuren en texturen toe te voegen aan de geografische objecten. Deze kleuren en texturen dienen opgeslagen te worden in het databasesysteem. In de literatuur worden verschillende suggesties voorgesteld voor het gebruik van kleuren en texturen. Gebaseerd op de literatuur zijn er een aantal verschillende implementaties gekozen voor de opslag van kleuren en texturen. Bij de uitvoering wordt gekeken welke gevolgen dit heeft voor het databasemodel.

- *Opvragen en aanbieden voor visualisatie van kleuren en texturen voor de geografische objecten verkregen uit een databasesysteem.* Net als de geografische objecten dienen ook de kleuren en texturen opgehaald te worden uit de database. Om dit te realiseren zal er een uitbreiding moeten plaatsvinden op de functionaliteit die voor het ophalen van de geografische gegevens wordt ontwikkeld.

- *Importeren van bestaande modellen in een databasesysteem.* Bestaande modellen moeten geïmporteerd kunnen worden binnen de database. Bij het importeren komen een aantal aspecten kijken die bij de implementatie van een importeerapplicatie meegenomen dienen te worden. Importeren van de gegevens wordt niet gezien als een doelstelling dat tijdens dit onderzoek geïmplementeerd dient te worden in een prototype applicatie.

- *Koppelen van attribuutinformatie.* De inzet van een database biedt mogelijkheden tot het maken van relaties met attribuutinformatie zoals soort gebouw, bouwjaar en het gemiddeld aantal mensen per gebouw. Er zal gekeken worden hoe attribuut gegevens gekoppeld kunnen worden aan de geografische objecten. Ook het koppelen van attribuutinformatie wordt niet geïmplementeerd binnen een prototype applicatie.

Dit stappenplan zal in het volgende hoofdstuk worden uitgewerkt.

4 Uitvoering

In dit hoofdstuk wordt de uitvoering van de deeltaken beschreven volgens het stappenplan van hoofdstuk 3.

4.1 Keuze database systeem

In het uitgevoerde literatuuronderzoek blijkt dat huidige databasesystemen nog geen volledige ondersteuning bieden voor de opslag van driedimensionale gegevens. Wel blijkt het mogelijk te zijn om de z-coördinaat van de objecten in een database op te slaan, wat voldoende blijkt voor dit onderzoek. In deze paragraaf wordt een keuze gemaakt voor een databasesysteem dat voor het vervolg van het onderzoek zal worden gebruikt. Bij het kiezen van een databasesysteem moet er eerst worden gekeken naar de soorten databasesystemen die er bestaan.

Databasesystemen bestaan er in verschillende varianten [19]. Het meeste gebruikte type databasemodel dat hedendaags wordt ingezet is het relationele database managementsysteem (RDBMS). Voordeel van deze databasesystemen is dat er relatie tussen gegevens gemaakt kunnen worden en structuren eenvoudig aangepast kunnen worden. RDBMS's bieden echter beperkte ondersteuning voor complexere datatypes en vrijwel geen functies die uitgevoerd kunnen worden op deze datatypes.

Object georiënteerde database managementsystemen (OODBMS) ondersteunen het gebruik van object georiënteerde dataobjecten waardoor er een situatie ontstaat die ook terug te vinden is in object georiënteerde programmeertalen als C++ en Java. De objecten kunnen ieder eigenschappen en functies bevatten zoals dat bij object georiënteerd programmeren ook het geval is. OODBMS zijn complexere databasesystemen en hebben een totaal andere manier van implementatie dan bij een RDBMS. Dit resulteert bij het gebruik van bestaande RDBMS in een herschrijving van het database model. Daarnaast is er binnen OODBMS nog geen eenduidige manier van het opslaan van een database model wat voor compatibiliteit problemen kan zorgen tussen verschillende systemen.

Een database type tussen RDBMS en OODBMS is een object gerelateerd database managementsysteem (ORDBMS). De opslag van objecten wordt hierdoor gerealiseerd en er kunnen functies worden uitgevoerd op de verschillende objecten. Verschil met OODBMS is dat een object geen interne functies heeft maar dat functies binnen het databasesysteem worden geïmplementeerd. Hierdoor kunnen bestaande RDBMS eenvoudig uitgebreid kunnen worden naar een ORDBMS

Binnen geografische informatiesystemen is het vooral belangrijk dat er een relatie tussen de verschillende geografische objecten bestaat. Dit heeft als gevolg dat er gekozen wordt voor het inzetten van ORDBMS. Uiteraard moeten deze ORDBMS beschikken over geografische datatypes zoals punten, lijnen en polygonen. Oracle Spatial en PostGIS zijn ORDBMS's die beschikken over geografische datatypes.

Beide databasesystemen implementeren de Simple Feature voor SQL-specificatie die is opgesteld door de OGC. In [17] is onderzocht wat de mogelijkheden zijn voor de opslag en het uitvoeren van selectie op driedimensionale geografische gegevens. Het resultaat is dat beide databasesystemen opslag van de z-coördinaat ondersteunen en functionaliteit bieden voor het uitvoeren van selecties hierop. Voor de uitvoer van dit onderzoek zal dit voldoende zijn.

Voor dit onderzoek is er gekozen voor het PostGIS databasesysteem (een uitbreiding op Postgres voor het applicatiegebied van geografische informatie). Dit database systeem houdt zich zeer strik aan de standaarden opgesteld door het OGC. Daarnaast is PostGIS in tegenstelling tot Oracle Spatial een open-source databasesysteem. Eventuele extra functionaliteit zou hierdoor makkelijker geïmplementeerd kunnen worden door bestaande broncode aan te passen en of uit te breiden.

4.2 Opslag 3D gegevens in een database

Voor de opslag van driedimensionale geografische gegevens in een PostGIS database zijn er, zoals in de literatuurstudie en in [17] beschreven, drie verschillende manieren:

- Uitbreiden van de geografische datatypes met een nieuw driedimensionaal datatype;
- Verschillende vlakken van een driedimensionaal object als samengestelde polygoon;
- Ieder vlak van een driedimensionaal object afzonderlijk opslaan als polygoon.

De verschillende manieren om driedimensionale gegevens op te slaan binnen een databasesysteem hebben ieder hun voor en nadelen.

Het opstellen van een nieuw driedimensionaal datatype is vooruitlopend op de abstract Feature Geometry specificatie [15] van het OGC. Voordeel van deze aanpak is dat objecten als geheel opgeslagen kunnen worden binnen een database record. Daarnaast zal een driedimensionaal datatype voor minder overbodige gegevens zorgen doordat punten een keer opgeslagen dienen te worden, dit in tegenstelling tot de andere twee implementaties.

Beperkingen worden veroorzaakt door het driedimensionale model dat als basis wordt gekozen. In [18] is een implementatie gedaan van een nieuwe datatype met als basis een polyhedron. Met deze implementatie wordt het opslaan van een bolvormig object erg complex. Uiteraard kan hiervoor een nieuw datatype worden opgesteld maar dit vergt ten opzichte van de twee andere implementaties meer tijd en met de komst van de door de OGC opgestelde specificatie wellicht overbodig werk. Ook moet de selectie functionaliteit binnen PostGIS uitgebreid worden, omdat PostGIS niet weet hoe er met dit nieuwe datatype om moet worden gegaan. PostGIS zal het niet kunnen verwerken of als twee dimensionaal object beschouwen.

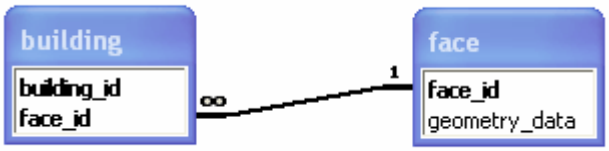
Het opslaan van driedimensionale objecten als samengestelde of losse polygonen heeft geen uitbreidingen nodig met betrekking tot de datatypes binnen de PostGIS database. Een samengestelde polygoon binnen een database record heeft als voordeel dat er een relatie blijft tussen de verschillende vlakken van een object. Bij een implementatie waarbij ieder vlak van een driedimensionaal object als los polygoon wordt opgeslagen in een record kan dit opgelost worden door gebruik te maken van een extra tabel die de relatie tussen vlakken en objecten onderhoudt. Een relatie tussen de verschillende vlakken is noodzakelijk om attribuutinformatie te kunnen koppelen aan het object.

Beide implementaties brengen als nadeel mee, dat punten overbodig meerdere malen opgeslagen worden voor ieder vlak. Voor een eenvoudige kubus (bestaande uit 8 hoekpunten) worden er in totaal 30 punten opgeslagen (ieder vlak bestaat uit 4 punten, plus het afsluitpunt). Uiteraard is met deze implementaties een bol ook moeilijk te construeren en enkel bij benadering met een grote hoeveelheid polygonen te bereiken.

Bij het onderzoeken naar de keuze voor opslag, is er gebleken dat er bij het uitvoeren van een selectie op een samengestelde polygoon niet het gewenste resultaat werd gehaald. Objecten werden in een selectie niet meegenomen doordat de bovenkant van een object buiten de selectie vielen. Hierdoor wordt het gehele record genegeerd. Bij de implementatie van losse polygonen werd in hetzelfde geval enkel de bovenkant niet meegenomen in het resultaat.

Voor het uitvoeren van het onderzoek is voor de implementatie gekozen waarbij de vlakken van een driedimensionaal object als losse polygonen (en daardoor dus losse database records) opgeslagen worden. Dit is eenvoudiger te realiseren dan het opstellen van nieuwe datatypes. Bij het gebruik van losse polygonen kunnen kleuren en texturen ook eenvoudiger aan een model gekoppeld worden. De overbodige gegevens die opgeslagen en bij het opvragen meegegeven worden, worden wel gezien als mogelijk oorzaak voor snelheidsverlies maar dit blijft beperkt doordat de hoeveelheid aan gegevens gering blijft.

De opslag van losse polygonen resulteert in een datamodel dat weergegeven wordt in Figuur 4-1. Het veld geometry_data bevat de polygonen die uiteindelijke gevisualiseerd dienen te worden. Om een goed overzicht te kunnen geven is in dit datamodel nog niet het gebruik van kleuren en texturen toegevoegd. Dit vindt plaats in een latere stap binnen de uitvoering.



Figuur 4-1- Database model voor opslag van 3D gegevens

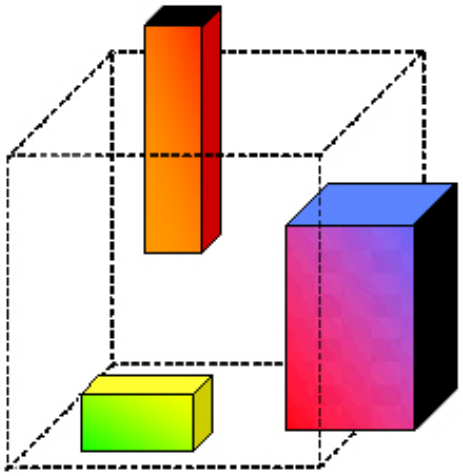
4.3 Selecteren en visualiseren van 3D gegevens

4.3.1 Selecteren van 3D gegevens uit een database

Bij het selecteren van driedimensionale geografische gegevens uit een database wordt er gebruik gemaakt van een selectie met behulp van een kubus. In Figuur 4-2 wordt dit principe weergegeven.

Het selectiecriterium is zo opgesteld dat alle polygonen die binnen de kubus zitten of deze kruizen worden opgehaald. Hierdoor kunnen er vlakken van de objecten buiten de selectiekubus steken maar toch worden meegenomen als selectieresultaat. Voor Figuur 4-2 heeft dit als gevolg dat de zwarte vlakken niet mee worden genomen in het selectieresultaat.

PostGIS kan voor een selectie gebruik maken van een kubus (Box3D) als argument [20]. Echter, de functie die de selectie uitvoert (intersect) transformeert het selectiecriterium naar de tweede dimensie, waardoor de z-coördinaat genegeerd wordt. Door de functie uit te breiden en ook de z-coördinaat (z-min en z-max) mee te nemen, wordt er een kubus selectie gerealiseerd.



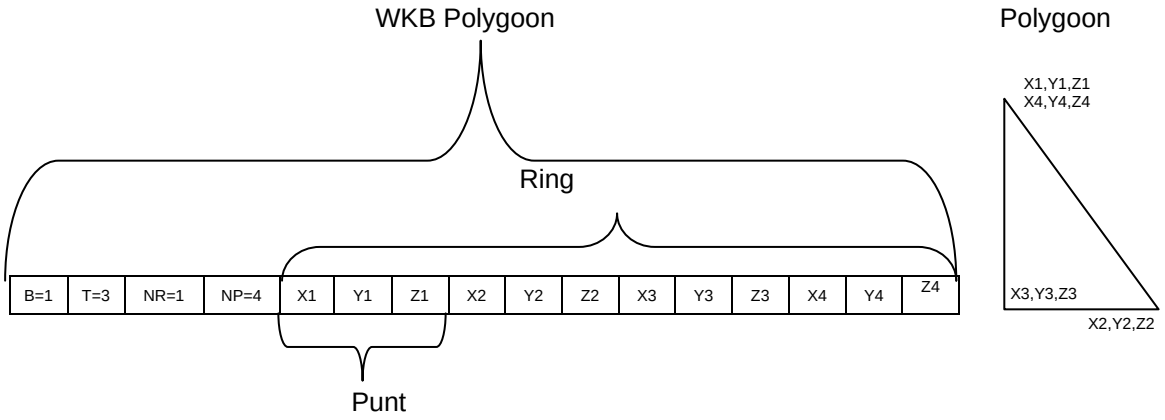
Figuur 4-2 - Selectie met behulp van een kubus

Om testresultaten te hebben over de hoeveel tijd die er nodig is om een selectie uit te voeren en vervolgens de resultaten terug te krijgen, is er een model geïmporteerd binnen de database. Dit model is het bestaande driedimensionale model van Amsterdam dat gebruikt maakt van de hoogte gegevens of met de hand gemodelleerde objecten. Dit model bestaat uit 142659 polygonen en 652961 punten. Over het importeren van bestaande gegevens zal verderop in de uitvoering worden gesproken.

De testresultaten zijn terug te vinden in bijlage 1. Indien alle polygonen opgevraagd worden zonder criterium blijkt het ophalen dezelfde tijdsduur te beslaan als met criterium. De selectie vertraagd het ophalen van gegevens dus niet en zal het zelfs versnellen indien de kubus verkleind wordt. Toch is het selecteren van de gegevens nu nog te langzaam als dit ingezet zou worden voor bijvoorbeeld het real-time ophalen tijdens een vlucht over het gebied. Hier zijn oplossingen voor te bedenken die het proces versnellen, deze worden in hoofdstuk 5 behandeld.

Selectieresultaten kunnen binnen PostGIS in verschillende formaten worden teruggegeven zoals tekst, GML (XML variant voor geografische objecten) [20] en binair als Well Known Binary (WKB) [12] formaat. De tekst en GML formaten geven als resultaat echter enkel tweedimensionale gegevens terug, waardoor de z-coördinaat te vervallen komt. De WKB bevat wel de gegevens van de z-coördinaat.

De samenstelling van het WKB formaat is schematisch weergegeven in figuur 3.



Figuur 1 - Schematische weergave van Well Known Binary formaat voor een polygoon [12]

De eerste bit (B) aan het begin van de WKB geeft aan in welke volgorde de bits zijn opgeslagen. Deze bit wordt de endian bit genoemd. Systemen die gebruik maken van een Intel architectuur slaan de bits in een little endian volgorde op, van links naar rechts. Systemen gebaseerd op een Macintosh of Sparc architectuur maken gebruik van een big endian volgorde, van rechts naar links. De volgerde van bits is een implementatie keuze die beide voor en nadelen hebben voor interne verwerking van gegevens. Voor het inlezen van het WKB formaat naar een formaat dat geschikt is voor de viewer betekend dat bits afhankelijk van de systeemarchitectuur en de opslag in de database omgedraaid dienen te worden. PostGIS biedt echter de mogelijkheid het resultaat van de gegevens te forceren naar een big endian (bit waarde 0) of naar een little endian formaat. (bit waarde 1).

Het tweede onderdeel (T) beschrijft het type van de WKB. In het geval van een polygoon zal het type een waarde hebben van 3. Voor een driedimensionaal object is aan het einde van de bit reeks een extra waarde toegevoegd wat aangeeft dat het een driedimensionaal object is.

Het NR gedeelte, bevat een waarde van het aantal ringen waaruit de polygoon bestaat (in dit voorbeeld dus 1). Iedere ring wordt begonnen met een NP, aantal punten, bit reeks waarin het aantal punten zijn opgeslagen (in het voorbeeld is dat 4 waarbij de laatste overeenkomt met de eerste om de polygoon af te sluiten). De Xn, Yn, Zn bevatten vervolgens de coördinatenwaarde van de punten.

4.3.2 Visualisatie van 3D gegevens

Voor de visualisatie wordt gebruik gemaakt van OpenSceneGraph (OSG) [6]. OSG is een open source driedimensionale toolkit geschreven in C++ [1] dat gebruikt maakt van OpenGL [21]. OpenGL is op zichzelf staand al een interface voor de aanroep van hardware die ingezet wordt voor twee en driedimensionale visualisatie. Het gebruik van OSG brengt voor de ontwikkelaar het ontwikkelen van driedimensionale applicaties tot een hoger abstractie niveau.
Het brede scala aan aanwezige methoden biedt diverse functionaliteit zoals gebruik van camera's,

[1] Binnen het bedrijf is er de wens ontstaan OSG te kunnen gebruiken binnen een Microsoft managed .Net programmeertaal als C#. In bijlage 2 wordt beschreven hoe bestaande C++ code gebruikt kan worden binnen de .Net omgeving.

interactie met de gebruiker door middel van het toetsenbord en of muis, en het toevoegen van speciale effecten als stofwolken en regen. Daarnaast wordt binnen OpenSceneGraph gebruik gemaakt van een boom structuur (SceneGraph) van de verschillende objecten binnen het model. Dit heeft als voordeel dat objecten snel gelokaliseerd kunnen worden, wat in het gebruik van geografische gegevens ervoor kan zorgen dat er relaties gemaakt kunnen worden tussen een object en eventuele attribuutgegevens. Deze gegevens kunnen dan op het moment dat de gebruiker het object selecteert worden getoond.

OSG biedt naast de boomstructuur ook een architectuur waarmee de ondersteuning van verschillende bestandsformaten kan worden uitgebreid. Deze architectuur leidt er toe dat er op dit moment al een groot aantal bestandsformaten worden ondersteund, zoals 3D studio max, JPEG en TIFF.

De uitbreiding van bestandsformaten wordt bereikt door een plugin architectuur. Iedere plugin heeft een aantal methoden voor registratie, het ophalen en wegschrijven van gegevens. Registratie van de plugins vindt run-time plaats aan de hand van de bestandsextensie zoals .3ds, .jpeg en .tiff. Iedere plugin heeft daarvoor een of meerdere extensies die worden ondersteund en tijdens het run-time proces geeft iedere plugin aan of deze de betreffende extensie ondersteund. Op het moment dat een plugin aangeeft dat de extensie ondersteund wordt is de koppeling gerealiseerd en zal er vervolgens een lees of schrijf actie worden uitgevoerd met gebruik van de plugin.

Voor de ondersteuning van de PostGIS database zal er een nieuwe plugin worden gemaakt die ondersteuning biedt voor het bestandextensie .postgis. Dit is geen bestaand bestandsformaat maar voor de OSG kern gaat het er enkel om dat er bij het lezen een OSG object wordt teruggegeven. Ditzelfde geldt voor het wegschrijven al zal dit in eerste instantie niet geïmplementeerd worden.

Voor het aanroepen van de postgis plugin zal gebruik worden gemaakt van een aanroep zoals deze in Figuur 4-3 is weergegeven.

server=<servername>\\user=<username>\\password=<password>\\dbname=<database>\\query=<query>)\\.postgis

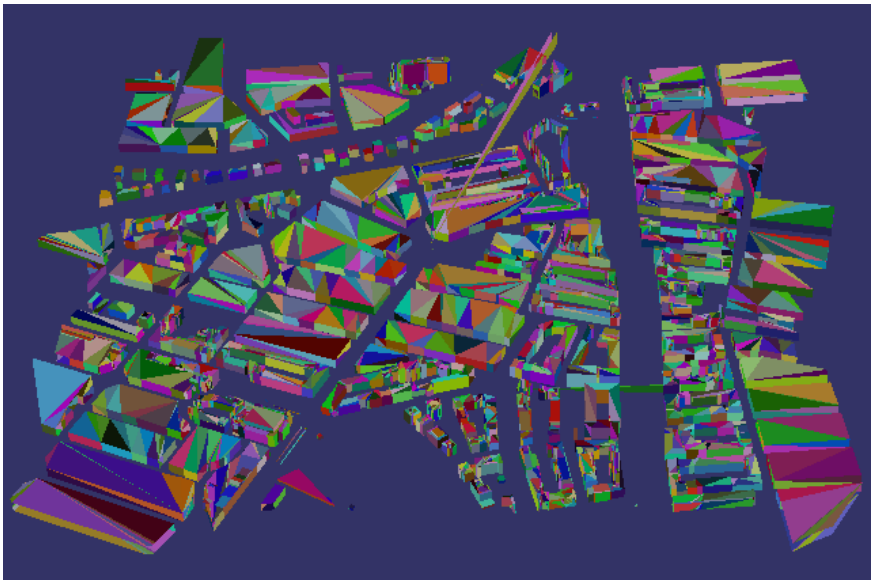
Figuur 4-3 – PostGIS plugin aanroep binnen OSG

De velden user en password bevatten de inloggevens voor de database ‘dbname’ die zich bevindt op de server ‘servername’. De query parameter bevat de volledige selectie die de PostGIS plugin dient uit te voeren. Voor het ophalen van driedimensionale objecten wordt een selectiecriterium meegegeven zoals dat in paragraaf 4.3 is beschreven. Vervolgens zal ieder geometrisch object wat in het WKB formaat wordt opgehaald omgezet worden naar OSG objecten. Deze omzetting zal voor dit onderzoek alleen plaatsvinden voor een polygoon. De OSG objecten worden vervolgens teruggegeven aan de OSG kern en ter visualisatie weergegeven.

Binnen de plugin is het belangrijk dat het omzetten van de WKB op de juiste manier gebeurd afhankelijk van de endian bit. Er wordt daarom gecontroleerd wat voor soort endian het computersysteem gebruikt en indien deze niet gelijk is aan de endian van het selectieresultaat dan zullen de verschillende bits eerst omgekeerd worden.

Het resultaat van de plugin voor het model dat in paragraaf 4.3 is ingeladen wordt in Figuur 4-4 weergegeven. Dit betreft een selectie van -500 (x-min), -500 (y-min), -150 (z-min) bij 500 (x-max),

500 (y-max), 150 (z-max). De kleuren die dit model bevat zijn willekeurig gekozen om zo een duidelijkere onderscheid te kunnen weergeven. In een latere fase van de uitvoering zal het datamodel en de plugin worden uitgebreid zodat kleuren en texturen worden meegenomen.



Figuur 4-4 – Weergave deel van model Amsterdam binnen OSG

4.4 Opslag van kleuren en texturen in een database

In [10] wordt het aanbrengen van detail op de driedimensionale objecten op verschillende wijzen mogelijk gemaakt. De eenvoudigste vorm is het aanbrengen van een kleur. Hierdoor kunnen daken onderscheiden worden van de muren, waardoor de derde dimensie in het model zichtbaarder wordt. Dit komt doordat vlakken tegen elkaar afsteken. Het gebruik van materiaaltexturen zoals bakstenen zorgt voor een extra detail niveau doordat duidelijk wordt uit welk materiaal een gebouw bestaat. De meest gedetailleerde vorm is het gebruik van fototexturen die gemaakt zijn van de gebouwen zoals deze in de werkelijkheid bestaan.

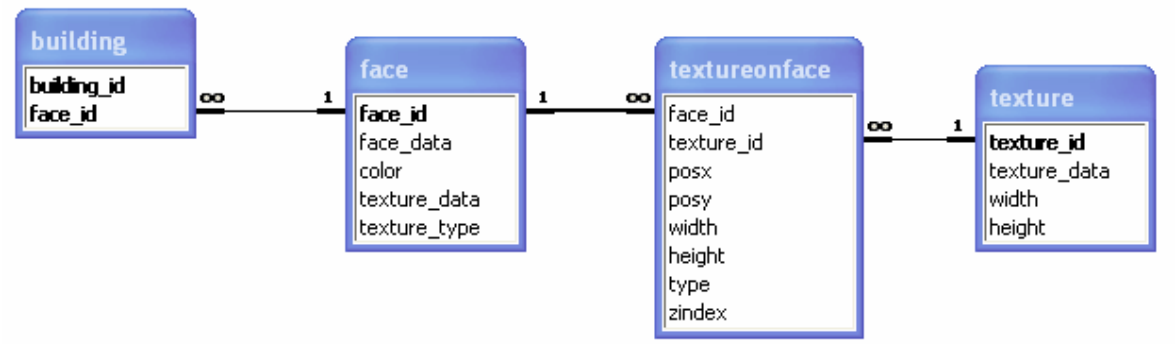
Fototexturen geven de meeste details aan het driedimensionale model. Er is van een groot deel van de gebouwen die in de werkelijkheid bestaan nog geen fototextuur aanwezig. Omdat kleuren en materiaaltexturen te weinig detail aanbrengen, zoals de aanwezigheid van deuren en ramen wordt, het gebruik van materiaaltexturen uitgebreid door middel van samengestelde texturen. Van gebouwen is eenvoudiger te bepalen en veelal bekend waar ramen, deuren maar ook bijvoorbeeld balkons zich bevinden. Het gebruik van samengestelde texturen wordt in Figuur 4-5 weergegeven.

De verschillende texturen, zoals verschillende ramen en deuren, worden in de database in een aparte tabel opgeslagen. Hierdoor ontstaat er een texturen bibliotheek die geschikt is voor hergebruik binnen het driedimensionale model. Uitbreiding op het datamodel zoals dat in paragraaf 4.2 is opgesteld wordt in figuur Figuur 4-6 weergeven.

De tabel texture bevat de verschillende afbeeldingen zoals ramen en deuren die in Figuur 4-5 zijn weergegeven. Met behulp van de textureonface tabel, worden de texturen aan een vlak van een driedimensionaal object gekoppeld. De verschillende velden in deze tabel zorgen voor de positie (posx, posy en z-index), de grootte (height en weight) en of de texture herhaald dient te worden (type).

Figuur 4-5 – Gebruik van een texturenbibliotheek

Indien er van de texturenbibliotheek geen gebruik wordt gemaakt kan er ook bij ieder vlak een kleur (color) of textuur (texture_data) worden opgeslagen. Daarbij kan ook bepaald worden of de textuur herhaald dient te worden (texture_type). Overigens is een combinatie van kleur, textuur op een vlak en de texturenbibliotheek ook mogelijk.



Figuur 4-6 - Datamodel met texturenopslag

4.5 Selecteren en visualiseren van kleuren en texturen

4.5.1 Selecteren van kleuren en texturen uit een database

Het gebruik van de kleuren en texturen heeft tot gevolg dat de selectie zoals die in paragraaf 4.3. plaatsvindt uitgebreid dient te worden. De texturenbibliotheek zorgt ervoor dat er meerdere database selecties plaats dienen te vinden om alle gegevens met betrekking tot samengestelde texturen op te halen. Dit werkt vertragend en zorgt ervoor dat de interface richting de PostGIS plugin niet meer transparant is en dat de plugin database model afhankelijk wordt.

Naast deze twee nadelen is er tijdens het onderzoek naar gebruik van texturen ook gebleken dat OpenGL [21] het gebruik van meerdere texturen wel ondersteund, maar met de beperking dat slechts een textuur een eigen textuur coördinaat kan meekrijgen. Overige texturen worden hierdoor altijd volledig over de gehele zijde uitgespreid of naast elkaar herhaald. Dit brengt met zich mee dat de gegevens van de samengestelde texturen van te voren plaats dient te vinden. Dit kan gebeuren binnen de database of binnen de plugin.

Door het samenstellen van texturen binnen de database plaats te laten vinden, worden de twee nadelen, het uitvoeren van meerdere selecties tussen database en plugin en het database model afhankelijk maken van de plugin, verholpen. Dit kan worden bereikt doordat de database met behulp van externe C bibliotheken afbeeldingen samenvoegt en hierdoor altijd maar één textuur in combinatie met de geografische data en eventueel een kleur aanbiedt aan de plugin. Om snelheidwinst te behalen tijdens het genereren van de samengestelde texturen kan er worden gekozen om deze samengestelde afbeeldingen op te slaan binnen de polygoon tabel.

4.5.2 Visualisatie van kleuren en texturen

Om de kleuren en texturen zichtbaar te maken zal de plugin aangepast dienen te worden. Het koppelen van kleuren en texturen is eenvoudig te realiseren omdat binnen OSG aan ieder gecreëerd object een kleur of textuur kan worden gehangen. Daarbij dient bij de texturen meegegeven te worden of deze over de hele zijde uitgestrekt of naast elkaar herhaald dienen te worden.

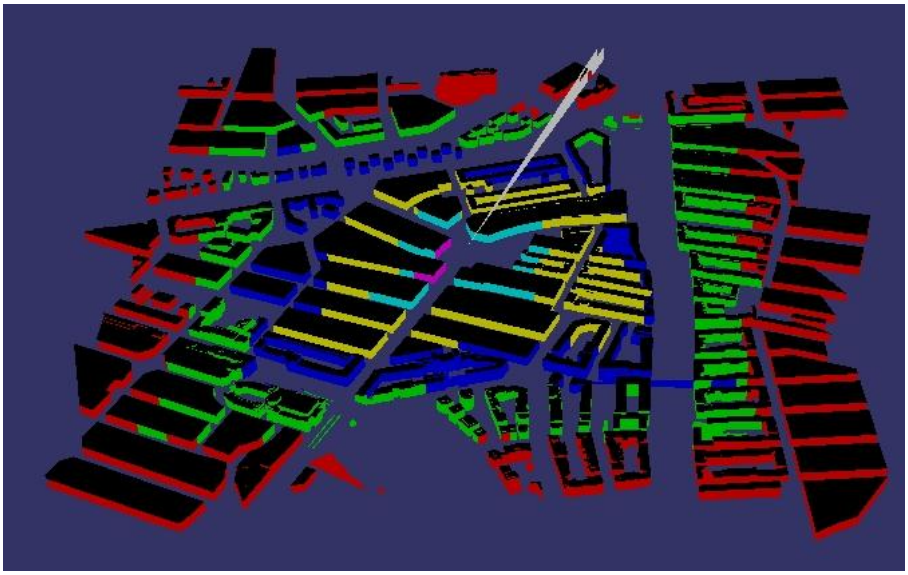
In tegenstelling tot het geografische datatype is voor de plugin niet direct duidelijk welke velden de kleur en textuur bevatten. Om dit probleem te verhelpen wordt bij de aanroep van de plugin bestandsnaam deze uitgebreid met beschrijvende gegevens (meta gegevens). De aanroep van de plugin veranderd hierdoor naar:

server=<servername>\\user=<username>\\password=<password>\\dbname=<database>
\\query=<query>)\\color=<colorfield>\\texture=<texturefield>
\\texturetype=<texturetypefield>\\.postgis

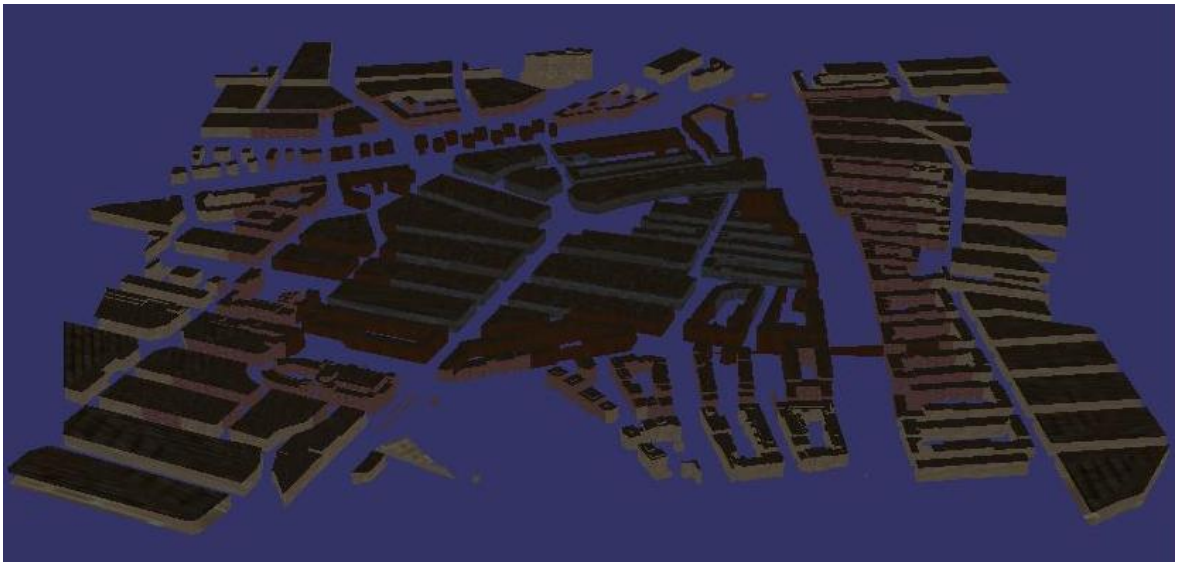
Figuur 4-7– PostGIS plugin aanroep met kleuren en texturen binnen OSG

Hierbij wordt van uit gegaan dat de samengestelde texturen onder dezelfde veldnaam worden aangeboden als de texturen die direct aan een vlak zijn gekoppeld. De toevoeging van kleur, textuur en textuur type aan de bestandsnaam geeft de relatie tot de database velden binnen het selectieresultaat aan.

Met behulp van deze uitbreiding in het databasemodel en de plugin is het mogelijk om kleuren en texturen te laten zien. Dit resultaat wordt getoond in figuur Figuur 4-8 en Figuur 4-9.



Figuur 4-8 – Weergave van model met kleuren



Figuur 4-9 – Weergave van model met texturen

4.6 Importeren van bestaande 3D gegevens

Voordat de driedimensionale gegevens uit de database geselecteerd en opgevraagd kunnen worden dient deze eerst in de database opgeslagen te worden. Het opslaan van de driedimensionale gegevens kan uiteraard volledig met de hand worden gedaan waarbij er geen gebruik wordt gemaakt van een bestaande model. Er zou ook een bestaand model geïmporteerd kunnen worden binnen de database. In beide situaties zijn er een aantal aspecten die bij het opslaan een belangrijke rol spelen.

- Schaal: vooral bij bestaande modellen dient de schaal van het model goed afgestemd te zijn op de aanwezige objecten binnen het model. Het schalen van een model kan met behulp van luchtfoto's worden ondersteund doordat de afdruk van het gebouw hierop zichtbaar is;
- Plaats: voor het plaatsen van het model kan gebruik worden gemaakt van een coördinaten stelsel, bijvoorbeeld GPS of een variant dat een kleiner gebied bestrijkt. Bij preciezere plaatsing kan ook hierbij het gebruik van luchtfoto's uitkomst bieden;
- Coördinatenstelsel: binnen PostGIS wordt er gebruik gemaakt van een XYZ stelsel waarbij X de diepte is, Y de breedte en Z de hoogte. Bij het importeren van bijvoorbeeld Virtual Reality Modeling Language (VRML) modellen, zoals heeft plaatsgevonden voor dit onderzoek diende een transformatie plaats te vinden bij het importeren. Dit omdat VRML met een XZY stelsel werkt. Indien er een verkeerd stelsel wordt gebruikt zullen vlakken gespiegeld worden;
- Kleuren: het toekennen van een kleur aan de zijden van een object kan bij gebruik van bestaande modellen vaak meegenomen worden doordat de kleurrelatie al aanwezig is;
- Texturen: het opslaan van texturen is het meest complexe onderdeel bij de opslag van driedimensionale modellen binnen de database. Indien er gebruik wordt gemaakt van zijde vullende texturen of texturen die over de gehele zijde herhaald dienen te worden, dan is de opslag vrij eenvoudig. Wordt er echter gebruik gemaakt van de texturenbibliotheek zoals die beschreven is in paragraaf 4.4 dan dient dit handmatig plaats te vinden. Wanneer de gegevens van ramen en deuren al reeds beschreven zijn in andere gegevensbronnen zoals een database of tekstbestand is dat niet nodig omdat deze gegevens geïmporteerd kunnen worden.

Bij het importeren van modellen is een applicatie die het plaatsen, schalen en koppelen van kleuren en texturen weergeeft zeer gewenst. Hiervoor zo gebruik gemaakt kunnen worden van OSG. Het wegschrijven van de gegevens naar de database zou hierbij geïmplementeerd kunnen worden binnen de PostGIS plugin.

4.7 Koppelen attribuut informatie

De inzet van de database voor de opslag van driedimensionale gegevens, heeft buiten de selectiemogelijkheid en de beheersbaarheid, ook de mogelijkheid om attribuutinformatie te koppelen aan de objecten.

Door het databasemodel uit te breiden met extra tabellen kunnen gegevens zoals soort gebouw, bouwjaar en aantal etages worden opgeslagen. Doordat binnen de database vervolgens een relatie kan worden gemaakt met het gebouw of enkel een zijde van het gebouw, wordt de koppeling gerealiseerd.

Het koppelen en ophalen van attribuutgegevens valt buiten het onderzoek maar biedt wel mogelijkheden voor de toekomst en ligt in de visie van GIS. Binnen het model dat gevisualiseerd wordt met behulp van OSG kan bijvoorbeeld door middel van een handeling die door de gebruiker via toetsenbord of muis wordt uitgevoerd, extra attribuutinformatie worden getoond. Zo kan er bij het klikken op een gebouw een nieuw venster komen dat gegevens bevat over het gebouw. Ook zouden er speciale spotlampen geplaatst kunnen worden boven gebouwen waarmee een onderscheid gemaakt kan worden tussen het soort gebouw.

5 Resultaten

Bij dit onderzoek stond centraal of het inzetten van een database de gestelde beperkingen kon verhelpen die bij het huidige model aanwezig waren. Als resultaat van dit onderzoek kan gesteld worden dat het gebruik van een database, waarin de driedimensionale objecten samen met de kleuren en texturen zijn opgeslagen, de beperkingen oplost en mogelijkheden biedt voor de toekomst.

- Selectie mogelijkheden: door middel van het gebruik van een database kan er eenvoudiger een selectie worden uitgevoerd op zeer omvangrijke modellen. Dit heeft als resultaat dat er minder gegevens bij het weergeven van het model ingeladen dienen te worden;
- Beheersbaarheid: de inzet van de database maakt het mogelijk om bestaande modellen te importeren en zo de aanlevering van de driedimensionale gegevens te verspreiden over verschillende partijen. Voor het beheren van de gegevens is een ondersteunende applicatie gewenst omdat aspecten als schaal, plaatsing en het koppelen van kleuren en texturen met behulp van een visuele weergaven het eenvoudigst is;
- Koppelen van attribuutinformatie: de inzet van een ORDBMS biedt de mogelijkheid om het databasemodel uit te breiden met extra informatie en deze te koppelen aan de driedimensionale objecten.

Het ophalen van de gegevens uit de database brengt op dit moment wel een beperking met zich mee betreffende de snelheid. Dit blijkt uit de testresultaten met betrekking tot het ophalen van de polygonen. Voor het ophalen van de kleuren, texturen en de samengestelde texturen neemt deze tijd enkel toe. Indien het ophalen van polygonen en texturen sneller zou plaatsvinden zou dit mogelijkheden bieden tot het maken van rondvluchten over geheel Nederland of zelfs een groter gebied. Een aantal mogelijke oplossingen waarnaar verder onderzoek gedaan zou moeten worden zijn:

- Aanbrengen van selectieniveaus: door in de database bij de polygonen een prioriteit op te slaan dat aangeeft of een polygoon zichtbaar is van een bepaalde afstand, kan dit de selectie versnellen. Een selectie zal dan bestaan uit een aantal kubussen met ieder een eigen selectie niveau. Dit zal resulteren in een vermindering van het aantal polygonen en een vermindering van de hoeveelheid gegevens dat opgevraagd dient te worden uit te database. Daarnaast kan er vanaf een bepaald niveau voor gekozen worden om alleen de kleur mee terug te geven en de textuur weg te laten;
- Detail texturen: texturen die worden gebruikt voor visualisatie kunnen voordat deze zichtbaar worden gemaakt, automatisch een aantal detail niveaus toegekend krijgen. Dit heeft als voordeel dat de grafische kaart wordt ontlast doordat ver wegstaande texturen van een mindere kwaliteit zijn. Echter, dit wordt enkel bereikt als de meest gedetailleerde textuur eerst vanuit de database wordt opgehaald en verwerkt. Een oplossing zou zijn om texturen van verschillende kwaliteit vooraf in de database op te slaan, zodat de hoeveelheid aan gegevens die opgehaald wordt beperkt kan worden afhankelijk van het selectieniveau.
- Samengestelde texturen opslaan: texturen die gebruik maken van de texturenbibliotheek worden bij het opvragen samengevoegd. Het voordeel hiervan is, dat indien er een wijziging is doorgevoerd in de texturen, dit direct zichtbaar is. Nadeel is dat het samenvoegen tijdrovend kan zijn. Vandaar dat een samengevoegd textuur opgeslagen kan worden in de database, zodat deze hergebruikt kan worden. Deze samengestelde texturen dienen dan wel opnieuw gegenereerd te worden als er wijzigingen plaatsvinden.

Door de inzet van een database voor de opslag van driedimensionale geografische gegevens kan er binnen de GIS wereld een nieuwe richting ingegaan worden die van tweedimensionale visualisatie naar driedimensionale visualisatie gaat. Het gebruik van een database biedt echter ook de mogelijkheid om er nog een dimensie aan toe te voegen, de dimensie van tijd.

Indien de tijd meegenomen wordt in de database dan zal dit resulteren in het weergeven van verschillen in de omgeving vanuit het verleden, het heden en de toekomst. Daarnaast kan zelfs de tijdlijn tijdens een rondvlucht over de omgeving geanimeerd worden.

Gezien de voordelen die behaald worden door de inzet van een database voor de opslag van driedimensionale geografische gegevens, zal een database als goede basis dienen voor driedimensionale GIS. Daarbij zal de opslag van driedimensionale objecten eenvoudiger en uitgebreider kunnen worden met de komst van de open standaarden die door het OGC worden opgesteld, waarbij ook de mogelijkheden tot het uitvoeren van analyses wordt gerealiseerd.

6 Evaluatie

In dit hoofdstuk vindt een evaluatie plaats over het uitgevoerde onderzoek en de behaalde resultaten. Er wordt hierbij gekeken of bij het opnieuw uitvoeren van dit onderzoek andere keuzes zouden worden gemaakt.

6.1 Inzet database voor 3D gegevensopslag

Gezien de resultaten van het onderzoek blijkt de inzet van een database voor de opslag van driedimensionale geografische informatie een goede basis te zijn voor applicaties die gebruik willen maken van deze driedimensionale gegevens. Op dit moment blijkt een databasesysteem als PostGIS al geschikt te zijn voor opslag van driedimensionale geografische informatie die enkel gevisualiseerd dient te worden.

Over enige tijd wordt er een definitieve specificatie verwacht van het Open Geospatial Consortium (OGC) voor de opslag van driedimensionale geografische informatie binnen een database wat als opvolger kan worden gezien voor de tweedimensionale geografische informatie opslag. Met deze specificatie zal dan ook de mogelijkheid worden gecreëerd om analyses uit te kunnen voeren op de driedimensionale gegevens, omdat hiervoor diverse functies beschikbaar komen.

6.2 PostGIS database

In dit onderzoek is gekozen voor een PostGIS database. Als alternatief had er ook voor Oracle Spatial gekozen kunnen worden of een andere databasesysteem dat ondersteuning biedt voor geografische datatypes.

Indien het onderzoek opnieuw uitgevoerd zou worden en er weer een afweging gemaakt zou moeten worden dan zou er weer voor PostGIS gekozen worden. Een belangrijke overweging die hierbij een rol speelt is het feit dat PostGIS open-source is. Hierdoor zijn er geen aanschafkosten voor de software en is functionaliteit aan te passen doordat de broncode beschikbaar is. In de situatie dat een specificatie voor driedimensionale gegevens opslag nog ontbreekt, kan dit zeer bruikbaar zijn doordat eventuele functionaliteit eenvoudig geïmplementeerd kan worden. PostGIS streeft daarnaast volledig de OGC specificatie na voor de opslag van tweedimensionale gegevensopslag. Omdat Oracle Spatial een commercieel product is, wijkt Oracle van deze specificatie soms af om klanten beter in hun behoefte te voorzien. Deze afwijking kan resulteren in compatibiliteitsproblemen tussen verschillende applicaties.

6.3 Opslagmethode

De opslagmethode waarbij de verschillende vlakken als losse database records worden opgeslagen is gekozen omdat de andere twee methodes niet geschikt waren. Indien de specificatie van het OGC beschikbaar is voor de opslag van driedimensionale geografische informatie zal de opslagmethode opnieuw bekeken moeten worden en wellicht dient er overgestapt te worden naar de in de specificatie opgestelde opslagmethode. Hierbij zou het uiteraard gewenst zijn als er een conversie beschikbaar is, waardoor bestaande gegevens geconverteerd worden naar het nieuwe opslagformaat.

Op dit moment, met het ontbreken van de specificatie, is het gebruik van een databaserecord per

vlak een zeer geschikte methode met als nadeel dat er een kleine hoeveelheid aan overbodige gegevens wordt opgeslagen.

6.4 Selectiemethode

Op grond van de bereikte resultaten moet worden geconstateerd dat de snelheid nog niet voldoende is om on-the-fly gegevens op te halen en te visualiseren. De in hoofdstuk 5 beschreven suggesties om dit te verbeteren, zouden dit probleem kunnen oplossen. Echter, het implementeren van een level of detail (LOD) manager die gebruikt maakt van een database, is een onderzoek op zich. De nu gekozen selectiemethode was voor dit onderzoek voldoende om aan te tonen dat een database een goede basis zal zijn voor de opslag van de driedimensionale geografische gegevens.

6.5 Visualisatie met OpenSceneGraph

Voor de visualisatie binnen dit onderzoek was gekozen voor OpenSceneGraph. De kracht van deze open-source visualisatie bibliotheek waarmee het geografische data kan visualiseren is uitermate geschikt voor zeer grote modellen. Daarnaast maakt de opgestelde architectuur het eenvoudig uitbreidingen te maken voor nieuwe bestandsformaten zoals bij dit onderzoek is gedaan voor PostGIS.

Het gebruik van OpenSceneGraph kan echter in dit onderzoek vervangen worden door een andere driedimensionale visualisatie omgeving. Het zou daarbij zeer gewenst zijn als deze tevens beschikt over een flexibele plugin architectuur. Keuze voor een andere driedimensionale omgeving voor bijvoorbeeld Geodan zou directe ondersteuning kunnen zijn voor Microsoft .Net programmeertalen. OpenSceneGraph maakt gebruik van C++ waarvoor een schil moet worden gemaakt om de code te kunnen gebruiken in een Microsoft .Net programmeertaal als C#. Het genereren van een schil wordt beschreven in bijlage 2. De wens om C# te gebruiken in plaats van C++ vloert voort uit het feit dat C# beschikt over automatisch geheugenbeheer. Binnen C++ dient de programmeur het geheugen zelf te beheren wat als complex wordt ervaren en tijdrovend is.

Overigens moet de GIS wereld geen eigen omgevingen gaan ontwikkelen die visualisatie van driedimensionale geografische informatie mogelijk maakt. Dit omdat bijvoorbeeld computerspelfabrikanten op het gebied van driedimensionale visualisatie al zoveel bereikt hebben waardoor een bijna fotorealistische omgeving kan worden gerealiseerd. De toenadering van GIS tot computerspellen heeft voor de computerspelfabrikanten ook voordelen. Zo is er op dit moment een beperking voor de grootte van de omgeving waarin een spel zich afspeelt die opgelost zou kunnen worden door de inzet van een database. Ook kunnen spellen zich gaan afspelen in de werkelijke wereld. Een samenwerking tussen beide partijen kan dus de snelheidsproblemen die bij het ophalen van de gegevens voorkomt wellicht oplossen.

7 Referenties

[1] N. Schuurman, 2003. *GIS a short introduction*, Blackwell Publishing Ltd.

[2] Top10NL Gegevensmodel - http://www.kadaster.nl/top10nl/

[3] 3D Studio Max - http://www.discreet.com

[4] Blender - http://www.blender.org

[5] Virtools - http://www.virtools.com

[6] Open Scene Graph - http://www.openscenegraph.org

[7] ESRI. 2002. *Geography Matters*, An ESRI Whitepaper, September 2002
http://www.gis.com/whatisgis/geographymatters.pdf

[8] S. Zlatanova, A.A. Rahman and M.Pilouk, 2002. *Trends in 3D GIS development*, Journal of Geospatial Engineering, Vol.4, No.2, pp. 1-10

[9] S. Zlatanova, 2002. *Advances in 3D GIS,* DDD - Rivista trimestrale di Disegno Digitale e Design edita da Poli.Desing. Anno 1 n. 4, Milano

[10] S. Nebiker, 2003. *Support for visualisation and animation in a scalable 3D GIS environment,* International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXIV-5/W10

[11] J. Stoter and S. Zlatanova, 2003. *3D GIS where are we standing?*, Joint Workshop on Spatial, Temporal and Multi-Dimensional Data Modelling and Analysis, 2-3 October, Quebec city, Canada, 6p

[12] OpenGIS Consortium, 1999. *The OpenGIS Simple Features Specification for SQL*, Revision 1.1, OpenGIS Document Number 99-049.
http://www.opengis.org/docs/99-049.pdf

[13] Oracle Spatial, http://www.oracle.com/technology/products/oracle9i/index.html

[14] PostGIS, http://www.postgis.org (manual: http://postgis.refractions.net/docs/postgis.pdf)

[15] OpenGIS Consortium, 2001. *The OpenGIS Abstract Specification, Topic 1: Feature Geometry (ISO 19107 Spatial Schema)*, Version 5. OGC Document Number 01-101.
http://www.opengis.org/docs/01-101.pdf

[16] International Organization for Standardization - http://www.iso.org

[17] J. Stoter en S. Zlatanova, 2003. *Visualisation and editing of 3D objects organized in a DBMS,* Proceedings of the EuroSDR Com V. Workshop on Visualisation and Rendering, 22-24 January 2003, Enschede, The Netherlands, 16p

[18] C. Arens, J. Stoter en P.J.M van Oosterom, 2003. *Moddeling 3D spatial objects in a*

GeoDBMS using a 3D primitive, AGILE conference, April 2003, Lyon, France.

[19] Jagadish Chaterjee, 2005. *Introduction to RDBMS, OODBMS and ORDBMS*, ASPFree.com, Maart 2005.

[20] OpenGIS Consortium, 2003. *OpenGIS Geography Markup Language (GML) Implementation Specification,* Version 3. OGC Document Number 02- 023r4.
https://portal.opengeospatial.org/files/?artifact_id=7174

[21] OpenGL - http://www.opengl.org/

[22] Open Geospatial Consortium - http://www.opengeospatial.org

[23] G. Gröger, M. Reuter, L.Plümer, 2004. *Representation of a 3D city model in spatial object-relational databases*, Institute for Cartography and Geoinformation, University of Bonn, Germany

Bijlage 1: Testresultaten PostGIS

In deze bijlage staan de testresultaten beschreven van de selectiequeries zoals deze beschreven zijn in 4.3. Voor het uitvoeren van deze test is een model van Amsterdam in de database geïmporteerd. Dit model bestaat uit 142659 polygonen die 652961 punten bevatten.

Bij het uitvoeren van de verschillende tests is er een selectie op basis van een kubus uitgevoerd waarbij de omvang van de kubus telkens aangepast wordt. De kubus selectiecriteria staan vermeld in de eerste kolom 'criteria'. De kolom 'aantal polygonen' bevat het aantal teruggegeven polygonen die binnen het selectiecriterium vallen of het selectiecriterium kruizen. De overige twee kolommen bevatten de tijd die verstreken is bij het uitvoeren van de selectie tot en met het verkrijgen van de gegevens ervan en de gemiddelde tijd per polygoon om de verschillende resultaten met elkaar te kunnen vergelijken.

Deze selectietests zijn uitgevoerd binnen het programma pgAdmin III van de PostGIS database (PostGis 1.0 rc6 en PostgreSQL 8.0). Het computersysteem dat hiervoor gebruikt is betreft een Pentium III 800 Mhz met 512 MB Intern geheugen.

Aantal polygonen:	142659
Aantal punten:	652961 (minimale overhead van 142659 punten omdat elke polygoon in de database gesloten dient te zijn)
Punten/polygoon	4.57
X-Min:	-2152.0
X-Max:	2152.0
Y-Min:	-1936.0
Y-Max:	1936.0
Z-Min:	-40.56
Z-Max:	32.58

Criteria	Aantal polygonen	Tijd (ms)	Tijd/polygoon (ms)
(geen)	142659	24229	0.17
X- -2152.0 X+ 2152.0 Y- -1936.0 Y+ 1936.0 Z – 40.56 Z+ 32.58	142659	24773	0.17
X- -2000.0 X+ 2000.0 Y- -1900.0 Y+ 1900.0 Z – 40.56 Z+ 32.58	136331	21553	0.16
X- -1500.0 X+ 1500.0 Y- -1500.0 Y+ 1500.0 Z – 40.56 Z+ 32.58	95916	15350	0.16
X- -1000.0 X+ 1000.0 Y- -1000.0 Y+ 1000.0 Z – 40.56 Z+ 32.58	56226	9080	0.16
X- -500.0 X+ 500.0 Y- -500.0 Y+ 500.0 Z – 40.56 Z+ 32.58	20414	3657	0.18
X- -250.0 X+ 250.0 Y- -250.0 Y+ 250.0 Z – 40.56 Z+ 32.58	8653	1876	0.21
X- -250.0 X+ 250.0 Y- -250.0 Y+ 250.0 Z – 25.0 Z+ 25.0	7536	1733	0.23
X- -100.0 X+ 100.0 Y- -100.0 Y+ 100.0 Z – 10.0 Z+ 10.0	4684	1491	0.32
X- -50.0 X+ 50.0 Y- -50.0 Y+ 50.0 Z – 10.0 Z+ 10.0	4684	1476	0.32
X- -10.0 X+ 10.0 Y- -10.0 Y+ 10.0 Z – 10.0 Z+ 10.0	4638	1452	0.31
X- -5.0 X+ 5.0 Y- -5.0 Y+ 5.0 Z – 5.0 Z+ 5.0	908	818	0.90
X- -0.0 X+ 0.0 Y- -0.0 Y+ 0.0 Z – 0.0 Z+ 0.0	2 (twee polygonen snijden hierbij het 0 punt)	671	335.5

Bijlage 2: Managed C++

Binnen C++ wordt het geheugenbeheer als complex en tijdrovend ervaren. Daarnaast worden computersystemen steeds sneller en bevatten meer geheugen waardoor het efficiënt beheren van geheugen in veel gevallen niet meer de hoogste prioriteit heeft. Omdat de voordelen van het automatisch geheugenbeheer sterker wegen dan de nadelen heeft Microsoft in 2001 [1] de managed .Net omgeving gelanceerd. Binnen deze omgeving zijn diverse managed programmeertalen te gebruiken als C++, C# en Visual Basic.Net.

Ook al kan het gebruik van een managed omgeving leiden tot een hogere productiviteit, doordat de programmeur niet constant geconfronteerd wordt met het beheren van het geheugen, overstappen van unmanaged naar een managed omgeving zorgt ervoor dat code getransformeerd dient te worden om hergebruik mogelijk te maken.

In deze paper zal er gekeken worden hoe bestaande unmanaged C++ hergebruikt kan worden binnen een managed omgeving, waarbij de bestaande code niet volledig herschreven dient te worden.

Opbouw van deze paper is als volgt. Eerst zal er gekeken worden naar .Net, het .Net framework en de Common Language Runtime (CLR). De CLR is binnen de managed .Net omgeving het onderdeel wat zorgt voor de automatische geheugenafhandeling.

Vervolgens zal er gekeken worden hoe bestaande unmanaged C++ code hergebruik kan worden binnen de .Net omgeving. Hierbij wordt enkel gekeken naar implementaties die hergebruik mogelijk maken zonder de inzet van (gedistribueerde) architecturen als CORBA. Het resultaat van deze vergelijking blijkt de inzet van Managed C++ te zijn waarbij er een schil om de bestaande code geschreven wordt.

De keuze van Managed C++ zal in de rest van de paper verder worden uitgediept. Daarbij zal een voorbeeld worden gegeven hoe Managed C++ in combinatie met bestaande (unmanaged) C++ gebruikt kan worden. Daarnaast zullen de belangrijkste kenmerken worden behandeld die bij op opstellen van een conversieschil tussen managed en unmanaged c++ komen kijken.

Voor het maken van de schil is een concept generator ontworpen. Een beschrijving van deze generator is te vinden aan het einde van deze paper. De paper wordt afgesloten met een conclusie waarin geëvalueerd wordt of de inzet van Managed C++, en de daarvoor ontwikkelde concept generator, het hergebruik van unmanaged C++ in een managed omgeving mogelijk maken.

.Net, het .Net Framework en de CLR

Binnen de .Net omgeving, wat vergelijkbaar is met Java van Sun [2], wordt de code niet meer gepasseerd naar machinecode, maar naar een tussenformaat wat de Microsoft Intermidiate Language (MSIL) wordt genoemd. Tijdens uitvoering draait deze code binnen een omgeving dat de Common Language Runtime wordt genoemd. Deze CLR zorgt ook voor het automatisch beheren van het geheugen en gooit objecten automatisch weg als deze niet meer nodig zijn door gebruik te maken van een garbage collector.

De MSIL maakt het daarnaast mogelijk om code van verschillende programmeertalen te combineren zoals C# en Visual Basic.Net. Hierdoor wordt het slechts een keuze van de programmeur welke taal er wordt gekozen.

Buiten de CLR en de MSIL biedt .Net de programmeur direct een grote hoeveelheid aan klassen en functies die direct ingezet kunnen worden. Dit zorgt er tevens voor dat de productiviteit zal kunnen stijgen doordat er sneller ontwikkeld kan worden.

Hergebruik van unmanaged C++

Voor het hergebruiken van bestaande C++ code beschrijft [3] drie verschillende implementaties (deze implementaties worden verderop toegelicht):

- COM Interop
- Platform Invoke
- Managed C++

Deze implementaties maken gebruik van de bestaande code of binaire bestanden en maken een soort schil om zo de functionaliteit her te gebruiken binnen een managed omgeving. Daarnaast wordt er geprobeerd het aantal extra instructies die noodzakelijk zijn voor de conversie beperkt te houden. Hergebruik van bestaande C++ door middel van gedistribueerde architecturen wordt binnen deze paper buiten beschouwing gelaten omdat de implementaties hiervan te complex zijn.

In [4] worden de bovengenoemde implementaties met elkaar vergeleken. Deze vergelijking wordt weergegeven in tabel 1. In deze tabel worden de aanpakken vergeleken met een aantal kenmerken.

Approach / Feature	COM Interop	P/Invoke	Managed C++
Type system	Good	Limited	Good
CF support	Poor	Limited	None
Callbacks	Fair	Poor	Good
Stability	Good	Good	Good
Useability	Fair	Poor	Limited
Maintainability	Fair	Poor	Good

Tabel 1 – Vergelijking verschillende methode voor hergebruik C++

De kenmerk Type system, geeft aan of er een grote hoeveelheid aan datatypes aanwezig is. Compact Framework support is voor de ondersteuning op mobiele apparaten waarop het .Net compact framework kan draaien. Callbacks betreft het aanroepen van functies vanuit de unmanaged code terug naar het managed gedeelte. Dit is noodzakelijk als er gebeurtenissen plaatsvinden binnen de code. Stability, Useability en Maintainability geven respectievelijk aan of deze implementatie, voor hergebruik van bestaande unmanaged C++ code, stabiel, eenvoudig toepasbaar en goed onderhoudbaar is.

Het gebruik van COM Interop waarbij de schil tussen managed en unmanged C++ plaatsvindt op basis van Component Object Model is een van de eerste aanpakken om hergebruik van unmanaged C++ mogelijk te maken. COM is een Microsoft Object georiënteerd programmeermodel dat definieert hoe objecten met elkaar communiceren binnen een applicatie of tussen applicaties. Voor het gebruik van COM Interop moeten de C++ objecten geconverteerd worden naar COM objecten. Voordeel van COM is dat het een uitgebreide type ondersteuning aanbiedt waardoor een goede overdracht van objecten tussen unmanaged en managed c++ kan plaatsvinden. Nadelen van COM is dat deze aanpak resulteert in veel conversie code waardoor het aantal instructies tussen unmanaged en managed c++ groot is. Dit zorgt voor een vertraging. Daarnaast zijn ook callbacks moeilijk te implementeren waardoor het complex is om aanroepen vanuit unmanaged C++ terug naar managed C++ plaats te laten vinden.

De tweede aanpak is om basis van Platform Invoke waarbij directe aanroepen plaatsvinden op de Dynamic Linked Libraries (DLL). Platform Invoke werkt alleen met flat API's en niet met klasse bibliotheken. Type ondersteuning is hierdoor beperkt. Er is geen conversie nodig van de C++ objecten omdat er gebruik wordt gemaakt van aanroep van de dll's. Bijkomend nadeel van Platform Invoke is dat deze aanroepen dynamisch zijn en niet worden gecontroleerd tijdens het compileren maar pas runtime. Ook de ondersteuning aan callbacks is beperkt.

De laatste beschreven aanpak is het gebruik van de Managed C++, dat beschikbaar is binnen .Net. Managed C++ is de enige .Net taal die de ondersteuning biedt om managed en unmanaged code door elkaar heen te gebruiken binnen een broncodebestand. Met de Managed C++ extensie ingeschakeld wordt alle code gecompileerd naar de Microsoft Intermediate Language (MSIL) waardoor deze her te gebruiken is binnen .Net (deze techniek waarbij unmanaged code en managed code gecombineerd kunnen worden heet It Just Works [3]). Op deze wijze is echter niet alle unmanaged code direct gemanaged waardoor er nog steeds conversiecode voor een deel van de methodes moet worden geschreven. Voor het gebruik van deze extensie is het dus wel van belang dat er header files en lib bestanden beschikbaar zijn omdat anders implementatie niet mogelijk is.

Het implementeren van callbacks is ten opzichte van de twee voorgaande keuzes eenvoudiger. Geheugenadressen kunnen vastgezet worden waardoor unmanaged code deze managed methodes kan aanroepen om een callback te registreren. Dit betekend wel dat binnen de conversieschil het localiseren en vrijgeven van geheugen dient plaats te vinden. Daarnaast biedt Managed C++ de mogelijkheid om de bestaande klassen structuur samen met de hiërarchie te behouden (met uitzondering van enkele beperkingen als overerving van meederde klassen).

Als alle drie de aanpakken worden vergelijken dan kan worden geconcludeerd dat Managed C++ de meeste mogelijkheden heeft. Het is qua performance het snelst gevolgd door Platform Invoke en daarna COM Interop doordat het aantal instructies tussen managed en unmanaged code beperkt blijft. Nadeel van het gebruik van Managed C++ is dat er geen applicaties beschikbaar zijn die de code kunnen laten genereren die als conversiebrug dient tussen managed en unmanaged C++.

Om conversie toch mogelijk te kunnen maken zal er een generator ontwikkeld dienen te worden die deze code genereert. Om te weten wat voor code gegenereerd dient te worden zal er eerst een voorbeeld worden gegeven van het gebruik van unmanaged C++ code binnen Managed C++. Vervolgens zullen een aantal kenmerken beschreven worden die een rol spelen bij het maken van een conversieschil met Managed C++.

Unmanaged C++ binnen Managed C++

Managed C++ gebruikt in tegenstelling tot de andere twee implementatie technieken gebruik van bestaande C++ header files.

In figuur 1 wordt een voorbeeld gegeven van een header bestand van een unmanaged C++ class Foo. Indien we deze class willen gebruiken binnen een managed programmeertaal als C# dan moet hiervoor een conversieschil worden geschreven in Managed C++.

```
class Foo
{
public:
  // constructor
  Foo(void) {}
  // destructor
  ~Foo(void) {}

  // some method
  void DoSomeFoo(){}
};
```

Figuur 1 – Voorbeeld unmanaged C++ header bestand [5]

Bij het opstellen van een conversieschil tussen unmanaged en managed c++ komen een aantal aspecten naar boven waarvan de belangrijkste in de volgende paragraaf worden beschreven. Volgens [5] komt het maken van een conversieschil in grote lijnen overeen met de volgende uit te voeren stappen, hierbij zijn specifieke unmanaged C++ taal implementaties achterwegen gelaten:

- Maak een managed class, waarin een van de member variable een pointer bevat naar de unmanaged class
- Voor elke constructor in de unmanaged class dient een overeenkomende constructor the worden gemaakt in de managed class. Binnen deze constructor worden instanties gemaakt met de new operator waarbij de originele constructor aangeroepen wordt.
- Indien de managed class de enige class is die een referentie heeft met het unmanaged object dan dient er ook een destructor geïmplementeerd te worden. Binnen deze destructor wordt er gebruik gemaakt van de delete operator om de geheugenadressering naar het unmanaged object vrij te geven.
- Voor iedere publieke methode in de unmanaged class, dient er een overeenkomende methode opgesteld te worden binnen de managed class waarbij de unmanaged methode wordt aangeroepen. Eventuele parameters en retourwaardes dienen geconverteerd te worden naar managed en unmanaged versies of andersom. Private methode dienen niet geimplementeerrd te worden omdat deze toch niet benaderbaar zijn van buitenaf.

Voor de unmanaged class Foo resulteert dit in managed code zoals deze is afgebeeld in figuur 2. De code __gc geeft aan dat de code onder beheer is van de garbage collector waardoor het geheugen automatisch beheert wordt. Indien dit object niet meer gebruikt wordt zal het automatisch de destructor aanroepen van het object en vervolgens zal het opgeruimd worden.

De __nogc* geeft aan dat het hier een pointer betreft naar een object waarvan het geheugen niet automatisch beheert mag worden. Hierdoor blijft het object altijd op hetzelfde adres in het geheugen staan en kan het unmanaged object zijn methodes uitvoeren. Het is wel van belang dat binnen de conversieschil het unmanaged object wordt vrijgegeven. Dit gebeurt in dit voorbeeld in de destructor.

```
__gc class MFoo
{
private:
  Foo __nogc* _foo;

public:
  // constructor
  MFoo() { _foo = new Foo();}

  // destructor
  ~MFoo() { delete _foo; }

  // method
  void ManagedDoSomeFoo() { _foo->DoSomeFoo(); }

};
```

Figuur 2 – Managed class Foo conversieschil code

Met deze code is MFoo volledig gemanaged en te gebruiken binnen iedere .Net programmeertaal.

Conversies tussen unmanaged en managed C++

Het voorbeeld dat in de vorige paragraaf geschetst is bevat slechts enkele aspecten die bij het opstellen van een conversieschil met behulp van Managed C++ komen kijken. In deze paragraaf worden de belangrijkste kernmerken behandeld, waarbij sommige zeer taalspecifieke C++ implementaties zijn. Voor deze implementaties is niet direct een oplossing beschikbaar voor implementatie binnen een managed omgeving.

Abstracte classen en interfaces

In unmanaged C++ worden abstracte classen en interface niet specifiek aangeven bij de class definitie. Binnen unmanaged C++ kan een class niet worden geinstantieerd als minimaal een methode in de header definitie gevolgd wordt door = 0; Indien dit voor alle methoden het geval is en de class bevat geen overige onderdelen als member variabelen, dan betreft het een interface.

Binnen Managed C++ worden abstracte classen gekenmerkt met het woord __abstract. Indien in het vorige voorbeeld de class abstract was geweest dan zou dat geresulteerd hebben in __gc __abstract class Foo. Voor interface is het soortgelijk maar dan wordt __gc class vervangen door __gc interface.

Methoden die geïmplementeerd moeten worden bij overerving worden aangegeven met het woord virtual en eindigen ook met = 0, net zoals in unmanaged C++.

Multiple inheritance

Managed C++ ondersteund in vergelijking met unmanaged C++ niet de mogelijkheid tot het maken van classen die afgeleid zijn van meerdere basisclassen. Classen kunnen wel afgeleid worden van een basis class in combinatie met diverse interface classes.

Voor de conversie heeft dit als gevolg dat de classen hiërarchie niet volledig gehandhaafd kan blijven. Echter blijft het wel mogelijk om alle methoden te ondersteunen door een kopie van deze managed methode voor alle extra basis classsen te implementeren binnen de class. Hierdoor wordt de methode niet meer afgeleid. Intern wordt nog steeds dezelfde methode uitgevoerd.

Managed argumenten en retourwaarden

Methoden binnen een class kunnen gebruik maken van argumenten en of retourwaarden teruggeven. Op het moment dat argumenten en retourwaarden binnen een managed omgeving worden gebruikt dienen er conversies plaats te vinden.

```
//Class Garage

void AddCarToGarage(Car * car);
```

Figuur 3 – Unmanaged methode met unmanaged object als argument

Voor standaard datatypes als een boolean, integer, float en double vind conversie tussen unmanaged en managed code automatisch plaats. Wordt er echter als argument meegeven of als retourwaarden terug gegeven wat een object betreft dan is conversie noodzakelijk.

In figuur 3 wordt een voorbeeld gegeven van een unmanaged methode, binnen de class Garage, met een argument van het type Car wat ook een unmanaged class is. Als deze twee classes gebruikt zouden willen worden in een managed omgeving dan zal er eerst voor beide classen code moeten worden geschreven zoals eerder al beschreven is.

Om vervolgens de methode AddCarToGarage in de managed omgeving te implementeren worden beide classen uitgebreid met een extra methode GetUnmanagedPointer(), dat de pointer naar het unmanaged object teruggeeft. Dit betreft de pointer die in de managed class is gemaakt naar de unmanaged variant van de class. De inhoud van de managed methode AddCarToGarage staat beschreven in figuur 4.

Voor conversie van retourwaarden kan dit opgelost worden door extra constructors te implementeren die een unmanaged pointer als argument verwachten.

De methoden die gebruik maken van unmanaged argumenten of deze teruggeven zoals GetUnmanagedPointer zijn niet te gebruiken in andere .Net programmeertalen als C# of Visual Basic .Net. Dit is geen probleem omdat deze methoden enkel gebruikt worden voor interne afhandelingen.

Argumenten binnen managed programmeertalen worden voor de meeste datatypes en voor objecten standaard als pointer meegegeven. Het is mogelijk om argumenten als reference mee te geven. Binnen een programmeertaal als C# dat geen gebruikt maakt van & voor een reference wordt dit vervangen door ref voor het argument te zetten.

```
__gc class MCar
{
private:
   Car __nogc* _car;

public:
   Car * GetUnmanagedPointer(){return _car};
   …
};

__gc class MGarage
{
private:
  Garage __nogc* _garage;

public:
  void AddCarToGarage(MCar * car){
      _garage->AddCarToGarage(MCar->GetUnmanagedPointer());};
  …
};_
```

Figuur 4 – Managed versie van methode AddCarToGarage

Methode met abstracte retourwaarde

Bij het vorige voorbeeld wordt er vanuit gegaan dat de objecten niet abstract zijn. In het geval dat er in de garage niet alleen auto's maar ook fietsen geplaatst kunnen worden zal er gebruik worden gemaakt van een abstracte class TransportItem waarvan de class Car en Bike worden afgeleidt. Bevat garage vervolgens een methode om een transportmiddel uit de garage te halen wat als retourwaarde het transportmiddel teruggeeft dan ontstaat er een probleem doordat de managed variant van de abstract Transport class niet geinstantieerd kan worden.

Dit probleem kan opgelost worden als de abstracte class afgeleid is van een concrete class of door in de managed versie de class af te laten leiden van een concrete dummy class. Nadat deze geïnstantieerd is kan deze eenvoudig worden gecast naar het abstracte type. Dit resulteert in de code weergegeven in figuur 5.

```
//Class Garage

TransportItem * GetTransportItemFromGarage(int index)
{
  return(TransportItem*)
      new MDummy(_garage->GetTransportItemFromGarage(index));
};
```

Figuur 5 – Voorbeeld managed methode met abstracte retourwaarde

Het instantiëren van een dummy class wat de unmanaged pointer meekrijgt als argument zorgt er niet voor dat er informatie verloren gaat doordat enkel de adres verwijzing wordt doorgegeven. Bij eventuele methoden die gebruik gaan maken van de abstracte retourwaarde moeten wel mogelijk controles worden ingebouwd om na te gaan om wat voor unmanaged type het gaat.

Argumenten met standaard waarden

Unmanaged C++ beschikt over de mogelijkheid om argumenten een standaard waarden te geven. Indien een methode met deze argumenten wordt aangeroepen dan kunnen deze argumenten achter wegen gelaten worden.

Managed C++ biedt niet de mogelijkheid om argumenten een standaardwaarde mee te geven. Om dit toch te kunnen realiseren dienen er verschillende versies van dezelfde methode gemaakt te worden met een verschillend aantal argumenten. Dit is weergegeven in figuur 6.

```
//Unmanaged Class Car

void setColor(ColorOptions color = ColorOptions::BLACK);

//Managed Class MCar

void setColor(){_car->setColor();};

void setColor(ColorOptions color){_car->setColor(color);};
```

Figuur 6 – Methode overloading

Methoden buiten een class

Binnen unmanaged en managed C++ kan er gebruik worden gemaakt van statische methoden. Deze methode kunnen direct aangeroepen worden zonder eerst de betreffende class te instantiëren.

Unmanaged C++ ondersteund ten opzichte van managed C++ ook de mogelijkheid om deze statische methoden buiten een class te zetten. Deze methoden komen dan binnen de namespace terecht. De namespace zorgt er voor om classen bij elkaar te groeperen en zo een onderscheid te kunnen maken. Indien er meerdere classen voorkomen met dezelfde naam maar met verschillende namespace dan kunnen deze classen gebruikt worden voorafgaand met de namespace omschrijving.

Statische methoden direct binnen een namespace worden niet ondersteund in Managed C++ maar kan wel opgelost worden door een extra class aan te maken met bijvoorbeeld de naam StaticMethods. De methoden binnen deze class worden vervolgens gekenmerkt dat ze statisch zijn. Voor het aanroepen van deze methoden dient de class StaticMethods niet geïnstantieerd te worden. Methode aanroepen worden wel voorafgegaan met de naam StaticMethods.

Templates

Om code efficiënter te kunnen gebruiken wordt hergebruik binnen C++ ondersteund met behulp van templates. Templates kunnen gezien worden als sjablonen die een beschrijving geven van een methode of class zonder in details te gaan. Een voorbeeld van een class template wordt weergegeven in figuur 7.

Bij het compileren van de code worden er klassen gegenereerd waarbij de template-argumenten vervangen worden door de expliciete datatypes.

Binnen de .Net omgeving is er geen ondersteuning voor templates waardoor templates niet volledig te implementeren zijn binnen de conversieschil tussen unmanaged en managed C++. Dit geldt niet voor template classen waarvan al bekend is welke datatypes gebruikt gaan worden. Voor deze classen kan een managed variant worden gemaakt waarbij de methode de juiste managed versie hebben van de bekende datatypes voor de template argumenten.

```
//Template class PERSONAL

template <class T>
class Person {
   private:
     char name[20];
     T information;           //Template argument T;
   public:
     Person(char * s, T info);
};

Person <int> P1("Adam", 23);   //Person <int> can be wrapped.
                               //The Person class not because T is
                              //templates are not supported and T is
                              //not known
```

Figuur 7 – Voorbeeld van een template class

Voor template classen, die bijvoorbeeld binnen een classen bibliotheek worden aangeboden, voor gebruik in applicaties kunnen er geen managed varianten worden gemaakt omdat de template argumenten niet expliciet bekend zijn. Deze beperking kan wellicht in de toekomst worden verholpen doordat binnen .Net 2.0 (de opvolger van de .Net omgeving die eind 2005 verwacht wordt) het gebruik van template classen binnen de .Net omgeving zal worden ondersteund.

Callbacks

De voorgaande behandelde punten voor het maken van een conversieschil tussen unmanaged en managed C++ hadden voornamelijk betrekking op het aanroepen van unmanaged code vanuit Managed C++. Toch is het in veel situaties gewenst om ook aanroepen vanuit unmanaged code te krijgen die managed code aanroepen. Dit mechanisme wordt callbacks genoemd.

Binnen C++ worden callbacks geïmplementeerd door aan een methode een pointer mee te geven naar een andere methode. Door het doorgeven van de plaats in het geheugen kan de betreffende methode uitgevoerd worden.

Bij Managed C++ wordt het geheugen automatisch beheerd waardoor het niet mogelijk is om het geheugenadres van een managed methode als pointer mee te geven aan een unmanaged methode. Het geheugenadres kan namelijk automatisch leeg worden gehaald of worden verplaatst.

Echter, binnen Managed C++ is het mogelijk om geheugenadressen vast te zetten door het gebruik van de functie __pin. Op deze wijze wordt het geheugenadres vastgepinned en kan een unmanaged methode, de managed methode aanroepen voor de callback implementatie.

Het geheugen dient wel weer vrijgegeven te worden binnen de conversieschil op het moment dat de callback niet meer nodig is.

Generator

In de vorige paragraaf zijn de belangrijkste kenmerken genoemd die komen kijken bij het maken van een conversieschil tussen unmanaged en managed C++. Het maken van deze conversieschil kan met de hand gebeuren maar bij een groot aantal classen is dit een arbeidsintensief karwei.

Om te onderzoeken of het mogelijk is een conversieschil te ontwikkelen is er een concept generator gemaakt. Deze generator is voor een groot gedeelte generiek ontwikkeld waardoor het voor ieder C++ project ingezet kan worden. Echter, bleek er toch specifieke code noodzakelijk te zijn om bepaalde datatypes en constructies te kunnen ondersteunen die per project afhankelijk zijn. Deze afhankelijkheden kunnen in een toekomstige versie met behulp van configuratiebestanden worden geïmplementeerd.

De concept generator is ingezet voor het ontwikkelen van een conversieschil voor de OpenSceneGraph [6] bibliotheek. Met behulp van de conversieschil zou het mogelijk moeten zijn de OpenSceneGraph functionaliteit ook te kunnen gebruiken in de .Net omgeving in combinatie met C# of Visual Basic.Net.

De informatie die nodig is voor het maken van de conversieschil bevindt zich in de beschikbare unmanaged C++ header bestanden. Voor het interpreteren van deze header bestanden moet er gebruik worden gemaakt van een soort parser die op basis van de C++ taalspecificaties een abstract formaat kan genereren. Dit abstracte formaat kan vervolgens gebruikt worden voor de invoer van de generator.

Om dit te realiseren kan er gebruik worden gemaakt van een applicatie als ASF/SDF Meta Environment [7], dat op basis van de C++ taalspecificaties een abstracte boomstructuur van de code kan genereren. Echter dient hiervoor de C++ taalspecificatie eerst gedefinieerd te worden binnen ASF/SDF. Om een sneller resultaat te behalen is er gebruik gemaakt van DoxyGen [8]. Overigens kan ASF/SDF in theorie ook gebruikt worden voor het genereren van de conversieschil code.

DoxyGen is een applicatie dat op basis van de header bestanden, documentatie kan generen zoals classdiagrammen. Als uitvoerformaat kan er ook gekozen worden voor XML. Doordat XML een gestructureerd formaat is kan deze eenvoudig worden doorlopen binnen de generator applicatie.

Met de eerste versie van de conceptgenerator is gebleken dat het genereren op basis van DoxyGen XML uitvoer inderdaad mogelijk is. Toch blijkt bij het gebruik van de managed versie veel functionaliteit te ontbreken. Dit komt met name doordat zeer taalspecifieke C++ implementatie moeilijk te implementeren zijn binnen de conversieschil. Daarnaast bleek ook dat DoxyGen nog niet alles volledig kan interpreteren waardoor belangrijke informatie van de C++ header bestanden ontbreekt.

Conclusie

In deze paper is er gekeken naar hoe bestaande C++ code hergebruikt kan worden binnen een managed omgeving als Microsoft .Net. Dit om het geheugenbeheer uit handen van de programmeur te nemen, hetgeen als tijdrovend en complex wordt gezien.

Op basis van Managed C++ blijkt een conversieschil tussen unmanaged en managed C++ mogelijk te zijn. Met behulp van een concept versie van een generator is aangetoond dat ook op basis van de C++ header bestanden, de conversieschil gegenereerd kan worden. Toch blijkt een volledige dekking van de functionaliteit nog niet volledig mogelijk te zijn, doordat bijvoorbeeld templates niet ondersteund worden en DoxyGen de header bestanden nog niet volledig kan interpreteren. Het is moeilijk om aan te geven hoeveel functionaliteit er daadwerkelijk in de eerste conceptversie ondersteund wordt en daar zal dan ook verder onderzoek naar gedaan moeten worden.

Als er gekeken wordt naar de snelheid dan geldt ook hiervoor dat verder onderzoek moet uitwijzen of er een groot snelheidsverlies optreed. Managed C++ zorgt in ieder geval voor de kortste lijnen tussen unmanaged en managed code waardoor het aantal instructies beperkt blijkt. Daarnaast wordt de bestaande code als het ware ingepakt waardoor interne berekeningen binnen de verschillende methoden nog steeds even snel worden afgehandeld. Vertraging zal daardoor alleen optreden als er veel geconverteerd moet worden tussen unmanaged en managed objecten.

Verder onderzoek zal de generator kunnen verbeteren waardoor wellicht een 99% dekking kan worden gehaald van de unmanaged functionaliteit. Met behulp van ASF/SDF zou een betere interpretatie kunnen worden gerealiseerd van de C++ header bestanden. De komst van de opvolger .Net 2.0, waardoor templates structuren ook ondersteund zullen gaan worden, zal in ieder geval bij gaan dragen aan een betere ondersteuning van taalspecifieke C++ functionaliteiten.

Referenties

[1] J. Templeman en Andy Olson, 2002. *Microsft Visual C++ .Net Step by Step.* Microsft Press.

[2] Sun Java - http://java.sun.com/

[3] Richard Grimes, 2003. *Programming with Managed Extensions for Microsoft Visual C++ .Net.* Microsoft Press.

[4] Rajesh Shenoy and Keith Moore, 2004. *Subtaining the integration of Long-Lived Systems with .Net.* HP Laboratories, http://www.hpl.hp.com/techreports/2004/HPL-2004-133.pdf

[5] Sam Gentile, 2004. *Writing Managed Wrappers with Managed C++*
http://www.ondotnet.com/pub/a/dotnet/2004/03/29/mcpp_part3.html

[6] OpenSceneGraph – http://www.openscenegraph.org

[7] ASF/SDF Meta Environment
http://www.cwi.nl/htbin/sen1/twiki/bin/view/SEN1/MetaEnvironment

[8] DoxyGen – http://www.doxygen.org