

Osm3dbuildingGenerator

MMT2 Verslag 2



20 april 2015

Stan Callewaert

2ICT3

Inhoudsopgave

**Opdracht 2**

**Onderzoek met bronvermelding2**

Data importeren van osm-bestand 2

Gebouwen van gebied 2D weergeven met polygonen 3

Gebouwen opsplitsen in driehoeken (polygon triangulation) 3

De oppervlakken van de gebouwen in 3D tekenen 3

De muren van de gebouwen in 3D tekenen 4

Landgebruik en checkboxes 4

Schaling en dynamische camera (roteren, verschuiven en zoomen) 4

**Praktische uitwerking 5**

Opbouwen van het drielagenmodel 5

Data importeren van osm-bestand 5

Gebouwen van gebied 2D weergeven met polygonen 6

Gebouwen opsplitsen in driehoeken (polygon triangulation) 7

De oppervlakken van de gebouwen in 3D tekenen 7

De muren van de gebouwen in 3D tekenen 7

Landgebruik en checkboxes 8

Schaling en dynamische camera (roteren, verschuiven en zoomen) 8

**Besluit 8**

**Bibliografie 9**

# Opdracht

De opdracht van het 2de project van het vak multimediatechnologie 2 was het grafisch weergeven in 3D van een gebied dat men kon specifiëren op de website van “OpenStreetMap” (OpenStreetMap, sd).

Concreet hield dit in dat er een gebied kon geselecteerd worden op de website van “OpenStreetMap”. Eenmaal dit gebied geselecteerd was kon men alle gegevens over dit gebied exporteren naar een osm-bestand. Dit bestand was de bron van data voor de applicatie zelf. Vanuit dit bestand moest data opgehaald worden en zo omgetoverd worden tot een 3D presentatie van het geselecteerde gebied. In dit gebied moesten de gebouwen en het landgebruik weergeven worden. Ook moesten van elk van deze verschillende subcategorieën gemaakt worden waardoor men zo kon filteren aan de hand van checkboxes wat men precies wou zien in de 3D omgeving.

Als laatste pijnpunt moest er dan ook nog een dynamische camera voorzien worden die zowel kon zoomen, draaien als verplaatsen. Dit om ervoor te zorgen dat de stad vanuit alle verschillende hoeken en afstanden kon geïnspecteerd worden.

# Onderzoek met bronvermelding

Vooreerst werden de nota’s en de slides die we meekregen via Toledo gelezen. Daarin stond er al een pak te implementeren code, vooral in het deel over OSM. Tijdens het lezen van de informatie op Toledo werd het al snel duidelijk dat dit geen kleinschalige, gemakkelijke opdracht zou worden. Het leek een pak werk en veel informatie werd je in een keer toegeworpen. Het besluit was dus dat er in verschillende stappen zou moeten gewerkt worden. Om deze reden gaat dit deel van het verslag verder met subtitels om de verschillende stappen duidelijk weer te geven.

Data importeren van osm-bestand

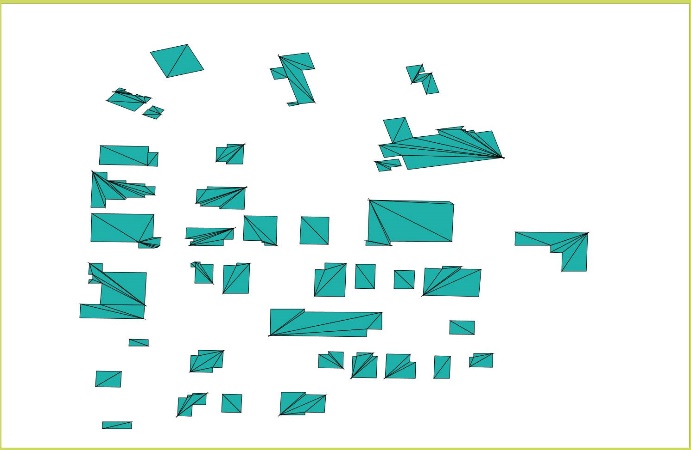
Om aan de volgende stappen te kunnen beginnen moest alle nodig data eerst opgehaald worden. De code hiervoor werd reeds voor een groot deel gegeven in de slides over OSM op Toledo. Om deze code echter te kunnen gebruiken moesten we eerst gebruik maken van de Package Manager Console om de OsmSharp package te kunnen ophalen. Hoe we dit konden doen kon ook terug gevonden in de slides. Echter moest er ook Visual Studio 2013 geïnstalleerd worden voor het pakket binnengehaald kon worden, dit om onbekende redenen. Vanaf dan kon het avontuur met het verwerken van de data beginnen.

Gebouwen van gebied 2D weergeven met polygonen

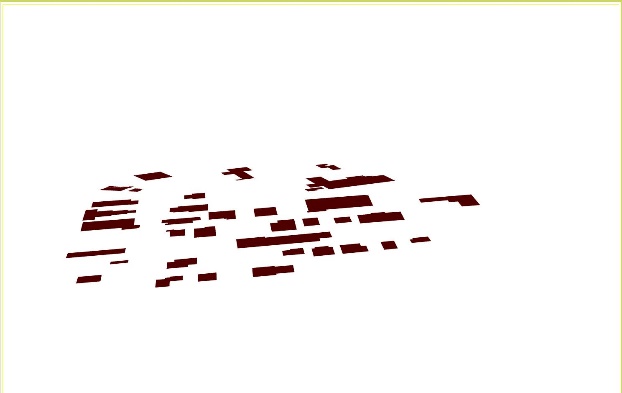
Door een simpele 3D driehoek te tekenen zoals in de voorbeelden van de slides gegeven op Toledo viel het op dat het direct in 3D weergeven van de gebouwen een veel te grote stap zou zijn. Dus werd er besloten de gebouwen weer te geven als polygonen (Polygon Class, sd).

Het onderzoek hiernaar was snel afgerond nadat er gevonden werd hoe een polygoon via C# in XAML kon geïmplementeerd worden (markmnl, 2013).

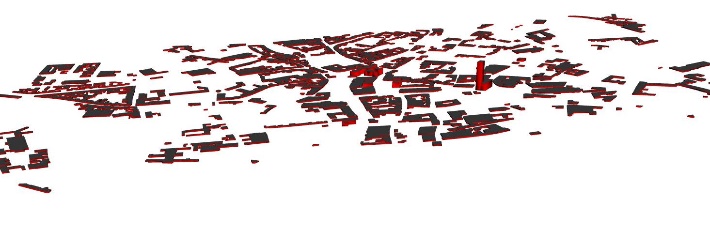
Gebouwen opsplitsen in driehoeken (polygon triangulation)

Dit was het moeilijkste deel van het onderzoek. Er moest gezocht worden naar een relatief simpele maar toch kwalitatieve methode om de reeds op voorhand gemaakte polygonen om te vormen in kleinere driehoeken. Mijn grootste hulp was hiervoor code die ik vond op Codeplex (BillOverman, 2008). In dit project moet er steeds naar een nieuwe driehoek (ear) gezocht worden doe dan kan afgezonderd worden van de polygoon. Dan begint herhaalt het algoritme zich om terug een nieuwe driehoek (ear) af te zonderen van de polygoon. Dit gebeurt tot de polygoon volledig verdeelt is in driehoeken.

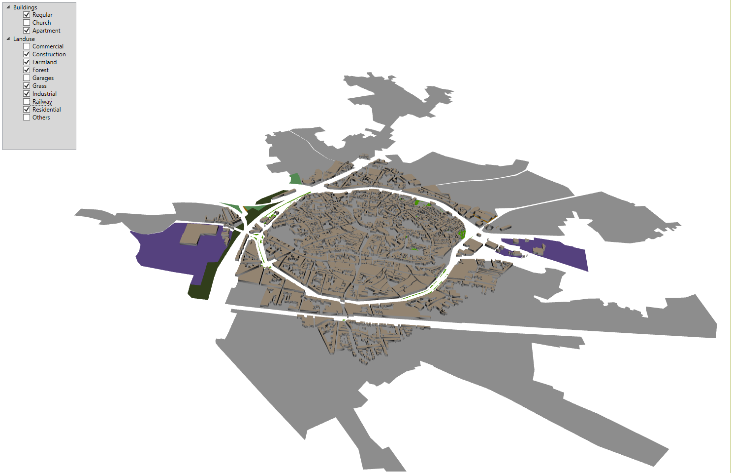
De oppervlakken van de gebouwen in 3D tekenen

Een stapje verder kunnen we nu gaan door deze formatie in 3D weer te geven. Het grote verschil hierbij is dat er een camera, licht, indices en een z-index moet meegegeven worden. Voor het toevoegen van deze 3D oppervlakken kwam het forum van Visual Studio goed van pas want daar was reeds een vraag gesteld hoe een oppervlak via C# in XAML kon geïmplementeerd worden (kaxxito, 2011). Met dit voorbeeld konden de 3D oppervlakken van de gebouwen geïmplementeerd worden met een juiste camera en het juiste licht.

De muren van de gebouwen in 3D tekenen

Hiervoor was er niet veel onderzoek nodig. Het was vooral het logische verstand dat hier voor een goed resultaat zorgde. Ook moesten de oppervlakken die hiervoor getekend werden een bepaalde hoogte krijgen zodat ze als dak van het gebouw konden functioneren.

Landgebruik en checkboxes

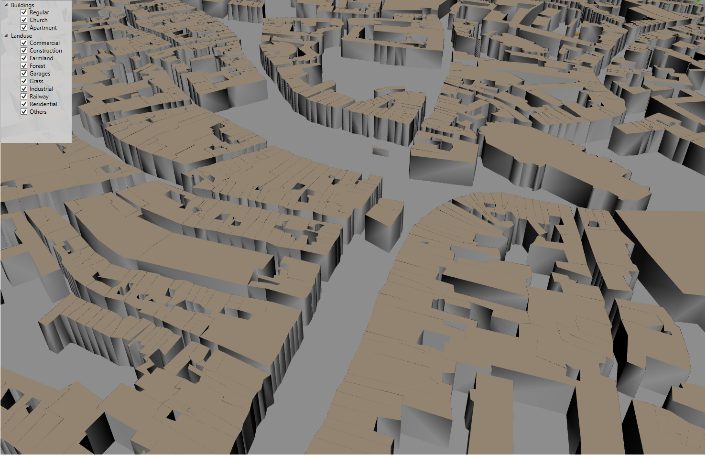
Voor het tekenen van het landgebruik was er ook niet veel onderzoek nodig. De data van het osm-bestand kon weer simpelweg opgehaald worden en zo kon met dezelfde methode als de daken van de gebouwen het landgebruik tekenen.

De checkboxes was nog een geval apart. Hiervoor moesten de verschillenden soorten gebouwen en landgebruiken opgesplitst worden in verschillende categorieën. Het opsplitsen gebeurde door de categorieën elk in een aparte “ContainerUIElement3D” te steken. Deze techniek kon terug gevonden worden op een pagina op “c-sharpcorner” waar men een volledige living in 3D weergeeft (Mike, 2010).

Om de checkboxes zelf vanboven links te plaatsen moest er dan weer gezocht worden op “codeproject” waar er een uitgebreide uitleg gegeven werd over treeviews en waar enkel de basis nodig was voor het gebruik van de applicatie (Iqbal, 2010).

De functionaliteit achter de checkboxes was da de categorie bij een aan gevinkte checkbox getoond moest worden en bij een uit gevinkte checkbox verborgen moest worden. Dit kon gebeuren met de “visibility” functionaliteit die gevonden werd op de Microsoft pagina (UIElement.Visibility Property).

Schaling en dynamische camera (roteren, verschuiven en zoomen)

Als laatste moest er dan ook nog geschaald worden (terwijl dit al veel eerder moest gedaan worden) en er moest ook nog een dynamische camera op het geheel geplaatst worden. Naar de schaling moest er niet veel onderzoek gedaan worden want in de slides die op Toledo stonden werd er duidelijk verwezen naar de “Web Mercator schalling” die dan ook gebruikt werd in de applicatie (Strebe, 2015).

Ook moest er nog een dynamische camera gebruikt worden die relatief snel gevonden werd op de website van Codeplex in het deel van 3DToools (een website waar open source tools gedownload kunnen worden) (corporation). Van dit project was enkel de “TrackBall” klasse nodig voor een dynamische camera te kunnen implementeren.

De startpositie van de camera was ook afhankelijk van de breedte en lengte van de stad en moest dus ook dynamisch geprogrammeerd worden. Een snelle zoekopdracht naar C# camera bracht mij al snel op een Microsoft pagina waar er uitgelegd werd hoe een camera vanuit C# een camera in XAML kon geplaatst worden (3-D Graphics Overview).

# Praktische uitwerking

De praktische uitwerking zal net als het “onderzoek met bronvermelding” opgesplitst worden in verschillende stappen. Dit omdat het project ook echt gecreëerd werd in verschillende stappen en er zo een overzichtelijk geheel van het verslag kan gemaakt worden.

De praktische uitwerking verliep vrij vlot maar af en toe waren er toch enkele struikelblokken die meestal zeer gemakkelijk opgelost konden worden maar waar toch veel tijd in gekropen is door steeds in dezelfde richting te blijven denken en niet eens na te denken of er voor het probleem in kwestie geen gemakkelijke of betere oplossing is.

De praktische uitwerking begon bij het installeren van het “OSMSharp” pakket via de Console Packet Manager van Visual Studio 2013. Eens dit gebeurd was moest er een osm-bestand gecreëerd worden. Het leek mij best om te beginnen met een bestand met een klein aantal gebouwen waardoor het makkelijker zou worden om de grafische uitwerking te bestuderen. Dus werd er een bestand geëxporteerd van OpenStreetMap met een klein aantal gebouwen dat verder gebruikt zou worden als osm-bestand om de applicatie te creëren.

Opbouwen van het drielagenmodel

Het eerste wat moest gebeuren voor de applicatie echt geschreven zou worden was het opdelen van de applicatie in drie lagen: presentation layer, business logic layer en data layer. Er werd voor elke laag een map gemaakt waar de klassen van die laag dan zouden terecht komen.

Dit model heeft de bedoeling dat de data gescheiden wordt van de presentatie en dat de gebruikte klassen in een soort bibliotheek staan (business logic layer). Deze structuur zorg voor een gestructureerd overzicht van alle klassen in het project en zorgt voor een extra veiligheid.

Data importeren van osm-bestand

Dit verliep relatief eenvoudig aangezien een groot deel van de code reeds gegeven werd op de slides over OSM op Toledo. Deze data werd geïmporteerd in de OsmData-klasse, die de enige klasse van de data layer is en waar zowel alle data wordt opgehaald van het osm-bestand als verwerkt wordt om door te geven aan de presentation layer.

Het enige wat nog niet gedaan was in de code op Toledo was het opslaan van de verkregen data. De manier waarop de data zou opgeslagen worden zou logisch moeten zijn. Er werd gekozen voor het maken van een lijst van gebouwen. Een gebouw zou dan een object zijn die een aantal attributen heeft zoals: nodes (punten), hoogte, soort en zoom.

Er was echter nog een aparte dictionary voorzien waarin alle nodes worden opgeslagen. De keys van de dictionary waren de node id’s en de values waren de nodes zelf. Hierdoor was het enkel nodig om de id’s van de nodes op te slaan in de gebouw objecten in plaats van het hele node object.

Nodes zijn de punten van het oppervlak van het gebouw. Hoogte is natuurlijk de hoogte van het gebouw. Soort is de soort van het gebouw en in dit project wordt een gebouw slecht opgedeeld in 3 categorieën: gewoon gebouw, kerk of appartement. Als laatste heb je dan nog zoom dat wordt gebruikt wordt om de latitude en longitude te kunnen omzetten naar x – en y coördinaten (dit wordt later uitgelegd).

Het waren niet enkel gebouwen dat moesten weergegeven worden, het was ook het landgebruik van een gebied. Om dit landgebruik kunnen op te slaan werd dezelfde methode als de gebouwen gebruikt. Er werd een klasse “Landuse” aangemaakt om zo alle landgebruiken in een lijst van objecten op te kunnen slaan. Deze klasse heeft overigens dezelfde attributen en een deel dezelfde methoden als de klasse “Building”.

Het leek me daarom ook evident een overkoepelende klasse “Area” aan te maken met virtuele methodes zodat die door beide kind-klassen gebruikt konden worden. De klasse “Area” is overigens abstract omdat het niet de bedoeling is dat er een object van het type “Area” kan aangemaakt worden.

Gebouwen van gebied 2D weergeven met polygonen

Nu begon het grafische werk en kon er eindelijk een resultaat gezien worden. De bedoeling was dat er een soort 2D platte grond werd meegegeven met alle gebouwen erin. Ook dit ging relatief vlot na het vinden van de juiste bronnen (reeds vermeld in “onderzoek met bronvermelding”).

Voordat we de gebouwen zouden kunnen weergeven zou er eerst geschaald moeten worden. Dit moest omdat de positie van de gebouwen voorlopig nog uitgedrukt stond in latitude en longitude. De schaling zouden we kunnen voltooien als we de minimum en maximum waarde hadden van de latitude en respectievelijk longitude.

De grenzen van de kaart konden gevonden worden in de bounds tag in het begin van het osm-bestand. Echter was er geen manier om deze op te halen via de “XmlSource”. Deze waarden moesten dus manueel via een “FileReader” uitgelezen worden en zullen dan opgeslagen worden in de nieuwe klasse “Map” die de gegevens over de grenzen van de kaart bijhoudt.

Nu we de minimum/maximum latitude/longitude hebben kan de schaling naar x/y punten doorgaan met de eenvoudige zelf geredeneerde formules:

en

Bij de y-waarde is het verloop in C# positief als de vector naar beneden loopt. Bij latitude is deze vector invers. Daarom moet de y waarde omgedraaid worden en is er dus een klein verschil tussen de formule van x en y.

Na de schaling van de latitude en longitude moeten de nieuwe punten enkel nog maar in een polygoon object gestoken worden en in de XAML code geplaatst worden. Het resultaat kan gezien worden bij “onderzoek met bronvermelding”.

Gebouwen opsplitsen in driehoeken (polygon triangulation)

Voor deze bewerking uit te voeren heb ik een goede bron op Codeplex (BillOverman, 2008). Deze bron heb ik volledig gekopieerd en later enkele aanpassingen aan aangebracht.

In deze code zijn er twee klassen: PolygonData en Triangulation2D. De PolygonData klasse is een alternatief voor de gewone Polygon klasse maar met een paar extra methoden en met punten van het type float. De Triangulation2D klasse is de plaats waar de magie gebeurt. Daar wordt de polygoon opgesplitst in driehoeken door steeds een nieuwe driehoek die aan de buitenkant van de polygoon ligt te vinden. Deze dan af te scheiden en het uiterste punt van deze driehoek te verwijderen uit de polygoon waardoor er steeds minder punten in de polygoon zitten tot de polygoon zelf een driehoek is geworden. Op dat moment is de polygoon volledig in driehoeken opgesplitst en kunnen deze driehoeken terug gegeven worden.

De oppervlakken van de gebouwen in 3D tekenen

Na wat opzoekwerk konden de gebouwoppervlakken in 3D weergegeven worden. De transformatie van 2D naar 3D verliep relatief vlot maar er was een klein probleem. Om een vlak met de voorkant naar de gebruiker weer te geven moeten de punten van de driehoek in linksom gedefinieerd worden in de indices. Echter was dit natuurlijk niet altijd het geval toen ik de gebouwoppervlakken wou weergeven. Er waren dus enkele gebouwen niet zichtbaar. Dit kon toen opgelost worden door een simpel algoritme dat ervoor zorgde dat de indices altijd linksom gedefinieerd werden. Echter kon dit veel simpeler opgelost worden maar meer info daarover bij “muren van gebouwen in 3D tekenen”.

De muren van de gebouwen in 3D tekenen

De muren in 3D tekenen ging wat moeizamer maar dit enkel door een het steeds in dezelfde richting blijven denken. De conclusie kwam al snel dat iedere muur een rechthoek was. Een rechthoek kan gevormd worden door 2 driehoeken. Dus het enige wat er moest gebeuren om de muren te vormen was het toevoegen van ieder punt zowel als punt op de grond (z-index 0) als punt op het dak (z-index groter dan 0). Waar men twee originele punten had, had men nu 4 punten waar 2 driehoeken uit gevormd kunnen worden. Zo gezegd zo gedaan, alle muren werden gevormd maar het probleem van het linksom definiëren van de indices stak weer op waardoor een deel van de muren niet zichtbaar was.

Om dit probleem op te lossen pakt ik de vectoren van iedere driehoek en zocht dan de cross vector van deze vectoren waardoor de richting van de indices kon bepaald worden maar bezig met al deze wiskundige problemen kwam het besef dat het veel eenvoudiger zou zijn om de achterkant van de driehoek gewoon hetzelfde kleur te geven als de voorkant.

De achterkant van ieder vlak een kleur geven werd dan ook gedaan waardoor de wiskundige formules om te bepalen of de indices van een vlak al dan niet linksom gedefinieerd zijn niet meer nodig waren en de code een stuk korter werd.

Landgebruik en checkboxes

Dit deel werd bijna volledig uitgelegd bij “onderzoek met bronvermelding”. Het enige wat er nog aan toegevoegd moet worden is dat na het toevoegen van de Trackball klasse om de camera dynamisch te maken, de treeview niet meer werkte omdat de eventsource naar de Trackball klasse werd verplaatst en de events van de checkboxes niet meer werden waargenomen. Daarom werd de treeview in een overkoepelende grid gestoken zodat de events wel nog geregistreerd werden.

Schaling en dynamische camera (roteren, verschuiven en zoomen)

Ook hier is niet veel meer over te vertellen. De schaling gebeurde door de “Web Mercator Projectie” (Strebe, 2015) en de dynamische camera kwam er door de Trackball klasse die ik haalde van Codeplex (corporation).

# Besluit

Voor het maken van een statische 3D omgeving is er meer wiskunde nodig dan gedacht. Vooral de transformatie van longitude/latitude naar x/y is vrij wiskundig. Ook was het vrij verassend dat er in 3D enkel met driehoeken gewerkt kan worden. Het is algemeen geweten dat er in 3D vrij veel met driehoeken gewerkt wordt maar ik had wel gedacht dat er ook nog andere figuren zouden kunnen gebruikt worden (zoals polygonen).

Voorlopig vond ik dit het meest plezante MMT project. Dit komt vooral omdat je de uitwerking van het project grafisch kan waarnemen en zelf niet van 1 standpunt maar van alle mogelijke standpunten. Ik ben ook zeer blij dat we iets gezien hebben over XAML waardoor er weer een portie kennis toegevoegd is aan mijn brein.

Mijn voornaamste les na dit project is dat je soms even moet stilstaan tijdens het programmeren en nadenken of je wel de juiste weg aan het opgaan bent. Ook heb ik geleerd dat het belangrijk is dat je een project stap voor stap opbouwt en niet meteen naar het eindresultaat werkt. Als laatste heb ik nog geleerd dat het gebruiken van andere mensen hun code helemaal geen schande is. Integendeel, het kan je project een heuse meerwaarde geven.

# Bibliografie

*3-D Graphics Overview.* (sd). Opgehaald van Microsoft: https://msdn.microsoft.com/en-us/library/ms747437%28v=vs.110%29.aspx

BillOverman. (2008, April 18). *Simple Polygon Triangulation*. Opgehaald van Codeplex: https://polygontriangulation.codeplex.com/

corporation, M. (sd). *Trackball.cs.* Opgehaald van Codeplex: http://3dtools.codeplex.com/SourceControl/latest#3DTools/3DTools/Trackball.cs

Iqbal, S. (2010, November 4). *Basic Understanding of Tree View in WPF.* Opgehaald van codeproject: http://www.codeproject.com/Articles/124644/Basic-Understanding-of-Tree-View-in-WPF

kaxxito. (2011, Februari 1). *please tell me what's wrong with my code.* Opgehaald van Microsoft: https://social.msdn.microsoft.com/forums/vstudio/en-US/0593949c-76de-4498-a06d-04e3ba7f133a/please-tell-me-whats-wrong-with-my-code

markmnl. (2013, December 9). *Drawing dynamic polygons in WPF*. Opgehaald van Stack Overflow: http://stackoverflow.com/questions/20465322/drawing-dynamic-polygons-in-wpf

Mike. (2010, Maart 4). *Creating a simple 3D scene in WPF.* Opgehaald van c-sharpcorner: http://www.c-sharpcorner.com/uploadfile/mheydlauf/creating-a-simple-3d-scene-in-wpf/

*OpenStreetMap*. (sd). Opgehaald van OpenStreetMap: www.openstreetmap.org

*Polygon Class*. (sd). Opgehaald van Microsoft: https://msdn.microsoft.com/en-us/library/system.windows.shapes.polygon%28v=vs.110%29.aspx

Strebe. (2015, Maart 18). *Web Mercator.* Opgehaald van Wikipedia: http://en.wikipedia.org/wiki/Web\_Mercator

*UIElement.Visibility Property.* (sd). Opgehaald van Microsoft: https://msdn.microsoft.com/en-us/library/system.windows.uielement.visibility%28v=vs.110%29.aspx