

Fractiles

MMT Verslag 3



28 mei 2015

Stan Callewaert

2ICT3

Inhoudsopgave

**Opdracht 2**

**Onderzoek met bronvermelding2**

Fractile tree 2

1D Midpoint displacement 3

2D Midpoint displacement 3

2D diamond-square 4

Implementatie in het 2de project 4

**Praktische uitwerking 5**

Fractile tree 5

Wall texture 5

1D Midpoint displacement 6

2D diamond-square 6

Implementatie in het 2de project 7

**Besluit 8**

**Bibliografie 9**

# Opdracht

Voor dit multimediatechnologie project werden 2 deelopdrachten gegeven in verband met fractalen. De eerste opdracht was een aparte opdracht, terwijl het in de tweede opdracht de bedoeling was dat je deze opdracht en project 2 van multimediatechnieken met elkaar in verband bracht en zo een mooi stad kon genereren.

De eerste opdracht van dit derde multimedia project is kort verwoord het experimenteren met fractalen. In de opgave die werd meegegeven stonden al meerdere algoritmes om mee te kunnen experimenteren. De bedoeling van dit eerste deel was dus een applicatie maken waarbij je kan experimenteren met deze algoritmes.

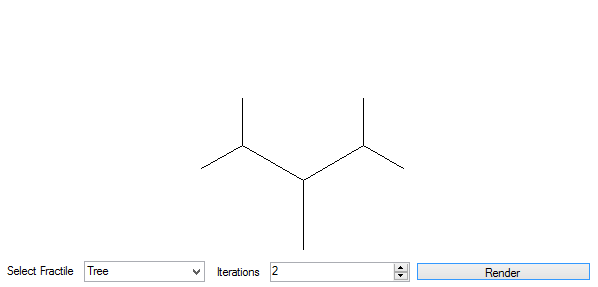
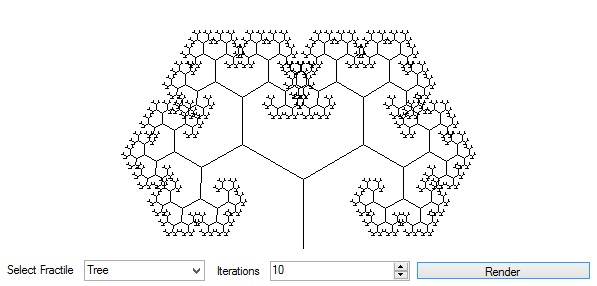
In het tweede deel van de opdracht was het dan de bedoeling de gegeneerde texturen in het eerste deel van de opdracht te gebruiken. Deze texturen konden dan geïmplementeerd worden in ons tweede multimediatechnologie project waar we een 3d stad moesten generen uit osm-data. Specifiek konden de texturen dan gebruikt worden als: achtergrond met wolken, grond met gras of muren van gebouwen met vensters in.

# Onderzoek met bronvermelding

Voor het maken van een opdracht is het verstandig na te denken wat de opdracht moet bevatten. In het geval van deze opdracht was dit relatief eenvoudig aangezien de algoritmes reeds gegeven werden. Toch kon deze opdracht wat meer de basis van fractalen ook behandelen en daarom heb ik ook een uitwerken van een ‘fractile tree’ uitgewerkt.

Fractile tree

Voor het uitwerken van deze boom werd de inspiratie vooral gehaald op YouTube in een filmpje over het maken van zo een boom in javascript (Kelleher, 2012). In dit filmpje werd uitgelegd hoe je fractalen kan programmeren a.d.h.v. recursieve methoden, waardoor code zich steeds herhaalt tot een bepaald opgegeven diepte. Hoe groter de diepte, hoe groter het aantal herhalingen en hoe groter de boom zal worden (zie afbeeldingen hieronder).

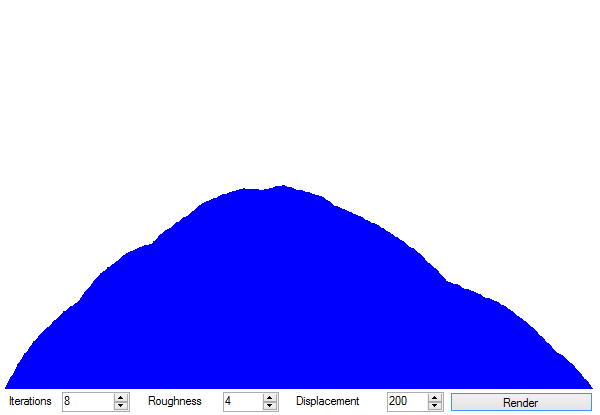
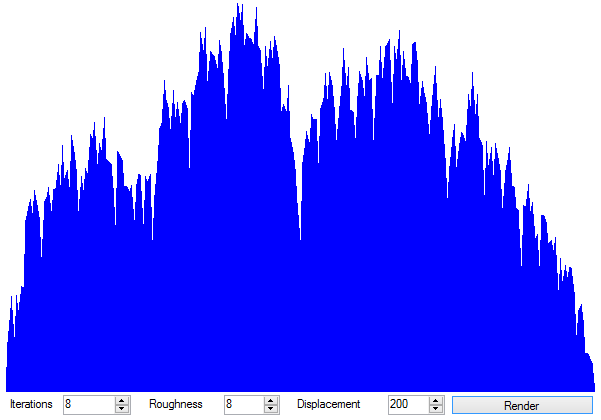
 

1D Midpoint displacement

Het genereren van een random grafiek met ‘midpoint displacement’ gebruikt ook het principe van fractalen. Hier wordt steeds een nieuw punt in het midden tussen 2 andere punten gezocht waardoor het algoritme zichzelf dus ook een aantal keer herhaalt en dus recursief werkt. Een steun bij het opmaken van deze code werd gevonden op de site van Bugfree Consulting (ronnieholm, 2009).

In deze code kon men vaststellen dat steeds de gemiddelde waarde van 2 punten moest genomen worden en deze moest opgeteld worden met een willekeurig getal. Dit willekeurig getal ervoor dat de grafiek steeds een andere vorm krijgt.

De aangemaakte klasse heeft ook nog 3 variabele parameters: het aantal iteraties, de ruwheid en de mate van de verplaatsing van een punt. Het aantal iteraties bepaalt het aantal punten in de grafiek dat per iteratie met een groter aantal vergroot. Als men de ruwheid laag zet zal men een grafiek verkrijgen met zachte overgangen, als men de ruwheid respectievelijk hoog zet dan zal men een grafiek met ruwe overgangen krijgen. De mate van verplaatsing bepaald ongeveer hoe hoog de grafiek zal zijn. Met nadruk op ongeveer want de grafiek blijft natuurlijk willekeurig gegeneerd.

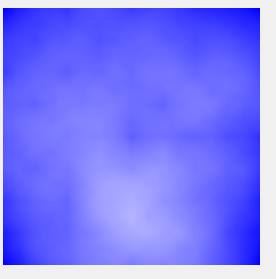
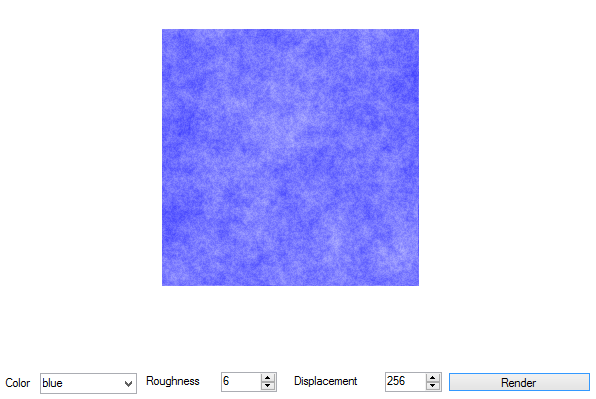
 

2D Midpoint displacement

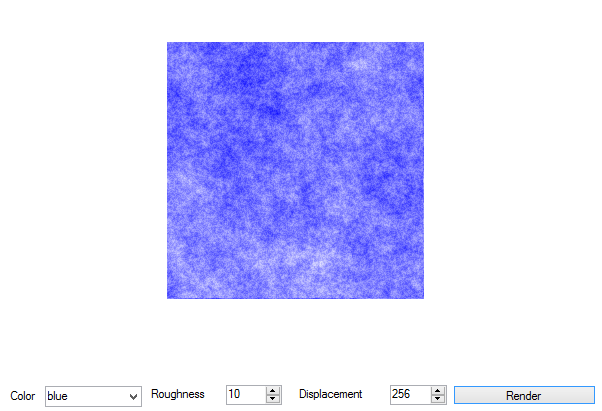
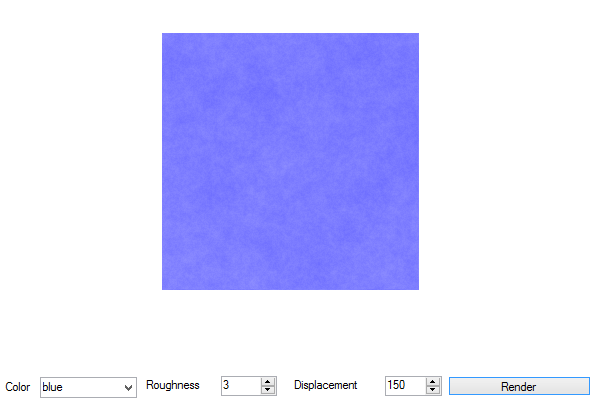
Dit algoritme werd niet uitgewerkt omdat ik merkte dat dit eigenlijk een minder goeie versie van het diamond-square algoritme is. Dit algoritme zal steeds vierkanten in de afbeelding creëren (zie afbeelding hieronder) terwijl dit bij het diamond-square algoritme niet het geval is. Omdat ze beide dezelfde doeleinden hebben en dit algoritme een minder goede versie ben ik rechtstreeks over gegaan naar het uitwerken van het diamond-square algoritme.



2D diamond-square

Dit algoritme heb ik eerst zelf proberen uitwerken maar toen ik er klaar mee was zag ik ook dat er vierkanten zichtbaar waren zoals in het 2D midpoint displacement algoritme. Dan heb ik eens gezocht naar het diamond-square algoritme op internet een mooi resultaat op de website van playfulljs (Loftis). Deze code werd wederom in javascript geschreven maar door deze code te bekijken kon ik afleiden dat mijn code verkeerd was en meer specifiek in het diamond deel (meer uitleg hierover straks) waar ik gebruik maakte van midpoint displacement tussen 2 punten (1D midpoint displacement) terwijl het de bedoeling is dat je net zoals bij het square deel het gemiddelde van 4 omliggende punten pakt en dit getal dan optelt met een willekeurig gegenereerd getal binnen een bepaalde range. Zo kon je het te zoeken punt bekomen en werden er geen vierkanten meer gevormd omdat het punt afhankelijk was van meer dan 2 omliggende punten.

Ook dit algoritme geeft enkele variabele parameters. In het diamond-square algoritme kunnen de ruwheid en de mate van de verplaatsing van een punt aangepast worden. Dit zijn dezelfde variabelen als het algoritme van 1D midpoint displacement, maar dan zonder het aantal iteraties omdat het aantal iteraties van het diamond-square algoritme afhangt van de grootte van de afbeelding. Met andere woorden in de afbeelding moeten alle pixels een kleur gekregen hebben en dan stopt het algoritme. Hieronder kunnen weer 2 afbeeldingen gevonden worden waar er gespeeld wordt met de variabele parameters.

Implementatie in het 2de project

Voor de implementatie van fractalen in het tweede project waren niet veel bronnen nodig. Het enige wat er gebruikt werd was Stack Overflow om een bitmap naar een bitmapImage te converteren (Hennig, 2011) en nog eens Stack Overflow om texture coördinaten te begrijpen (Schertler, 2013).

De veranderingen in het project waren de achtergrond die nu de lucht geworden is met wolken. De grond die nu een grasveld is geworden. Ook de muren van de gebouwen zijn veranderd want nu staat er een texture op van muren met vensters.

# Praktische uitwerking

De praktische uitwerking werd stapsgewijs uitgebouwd door steeds een stap hoger in het fractalen proces te gaan. Het begon bij een fractal, de volgende stap was het midpoint displacement algoritme, daarna was het de beurt aan het diamond square algoritme en als laatste werd er een eigen implementatie van fractalen in project 2 geïmplementeerd.

De meeste problemen bij de praktische uitwerking ontstonden bij het diamond-square algoritme omdat dit fout werd geïmplementeerd. Meer uitleg hierover volgt bij het deel over diamond-square.

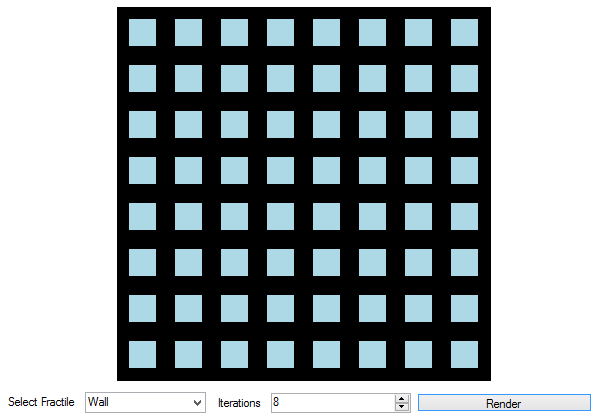
Fractile tree

Voor het maken van deze boom zijn er methodes nodig die zichzelf steeds opnieuw oproepen, iteratieve methodes dus. Deze methodes moet zichzelf een aantal keer terug oproepen. Een limiet instellen op de iteraties kan door een variabele diepte mee te geven aan de methode en in de methode te kijken of de diepte reeds nul is. Zo ja, stopt de methode met zichzelf terug op te roepen. Zo nee, roept de methode zichzelf weer op maar deze keer met diepte van 1 minder dan de vorige keer.

Natuurlijk moet er in de methode ook enige logica zitten die iets doet. Anders roept de methode zichzelf steeds op zonder enig resultaat te boeken. In dit voorbeeld roept de iteratieve methode 2 andere methodes op. Dit wil dus zeggen dat deze methode zichzelf niet oproept maar die 2 andere methodes, deze methode terug oproepen. In dit voorbeeld hoort dit zo omdat we uit iedere tak terug 2 takken willen laten groeien. Dat wil zeggen dat op iedere tak die gecreëerd wordt later terug moet kunnen verder gebouwd worden en daarom moeten voor de opsplitsing in twee takken, twee nieuwe methodes gecreëerd worden.

In deze methode zijn er vele variabelen die kunnen aangepast worden. Bijvoorbeeld de startlengte van de eerste tak, de verhouding van de lengte tegenover de volgende tak, de hoek die de takken maken als ze uit een andere tak groeien, …

Wall texture

Voor het patroon van venster op de muren van de gebouwen werd deze fractal gecreëerd. Als variabele kan je het aantal iteraties meegeven die dan bepaald hoeveel vensters er per zijde zijn.

Het algoritme van dit fractal berekent eerst hoeveel plaats alle ramen kunnen innemen met marges tussen de ramen meegerekend. Dit moet een natuurlijk getal zijn en natuurlijk deelbaar door het aantal ramen. Hierdoor zal er een rest ontstaan. Deze rest gedeeld door twee en vanboven en links van het eerste raam geplaatst, waardoor alle ramen samen ook in het midden zullen staan.

Op dit moment hebben we de breedte van ieder raam met de marge erbij gerekend. Als constante kan er meegegeven worden hoe de verhouding tussen venster en marge moet zijn. Door deze verhouding kan de breedte van het raam en de breedte van het marge berekend. Voor de hoogte zijn alle afmetingen dezelfde aangezien het vlak een vierkant is.

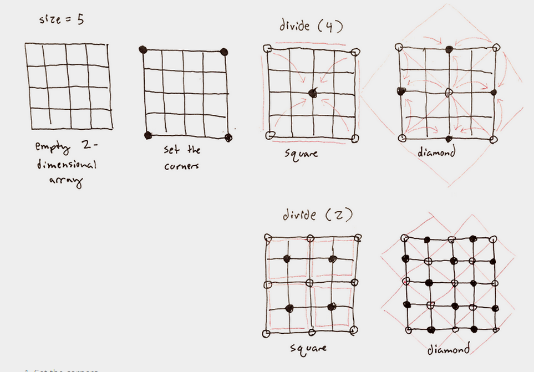
Nu we alle gegevens hebben is het tijd om de vierkanten uit te tekenen op een bitmap en die bitmap dan uiteindelijk terug te geven aan het form.

1D Midpoint displacement

Dit algoritme werkt zoals alle andere fractal algoritmen ook recursief. In dit algoritme wordt de hoogte de 2 aanliggende punten genomen. Daarvan wordt dan het gemiddelde genomen en dat getal wordt dan plus een willekeurig getal gedaan dat ligt tussen 0 en de grootte van de “displacementFactor” (deze variabele kan meegegeven worden met de constructor). De “displacementFactor” zal iedere iteratie vermenigvuldigd worden met de “roughness” (getal van 0 tot 1). Hierdoor zal de “displacementFactor” steeds kleiner worden waardoor de overgangen tussen verschillende punten vloeiender zullen zijn.

De gebruiker kan in het form door middel van een combobox beslissen hoeveel keer het voorgaande algoritme herhaalt wordt. Hoe meer keer dit algoritme herhaalt wordt, hoe meer punten er op de grafiek zullen liggen.

De ruwheid en de mate van verplaatsing van de punten kan ook door de gebruiker beslist worden maar het effect dat dit heeft op de grafiek is reeds voorheen in dit verslag uitgelegd.

2D Diamond-square

Op de foto hiernaast kan gezien worden dat er in de methode van dit algoritme 2 dingen moeten gebeuren: het square deel en het diamond deel. Vandaar de naam van het algoritme natuurlijk.

In het square deel wordt de waarde van het middelste punt van een vierkant berekent door het gemiddelde van de waarden van de vier hoeken te nemen en die plus een willekeurig getal van de negatieve “displacementFactor” tot de positieve “displacementFactor” te doen.

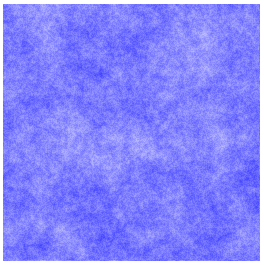
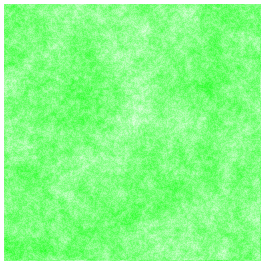
In het diamond deel wordt de gemiddelde waarde van de punten die een halve zijde boven, links, rechts en onder het punt liggen plus die plus een willekeurig getal van de negatieve “displacementFactor” tot de positieve “displacementFactor” te doen.

De “displacementFactor” zal iedere iteratie vermenigvuldigd worden met de “roughness” (getal van 0 tot 1). Hierdoor zal de “displacementFactor” steeds kleiner worden waardoor de overgangen tussen verschillende punten vloeiender zullen zijn.

Eens dit alles gedaan is zal deze methode zich herhalen en dit tot alle punten in het vierkant geïnitialiseerd zijn.

Na alle punten geïnitialiseerd zijn moeten ze omgezet worden naar een bitmap en daarvoor moeten de punten dus tussen de 0 en de 255 liggen (om naar een kleur om te vormen). Voordat de bitmap dus gecreeerd wordt zal er dus een transformatie van de punten plaatsvinden waardoor alle punten tussen 0 en 255 zullen liggen.

Eenmaal deze transformatie voltooid is kan de bitmap gecreëerd worden. Hoe de R, G en B waarde ingevuld worden kan de gebruiker kiezen in het form. De gebruiker kan kiezen tussen blauw en groen. Bij blauw zullen de R en de G waarde veranderlijk zijn terwijl bij groen de R en de B waarde veranderlijk zullen zijn (de niet veranderlijke waarde is 255). Hieronder staan afbeeldingen van het resultaat van het blauwe en het groene diamond-square algoritme.

Implementatie in het 2de project

De gegenereerde fractalen konden gebruikt worden als muren voor de gebouwen, als lucht en als gras.

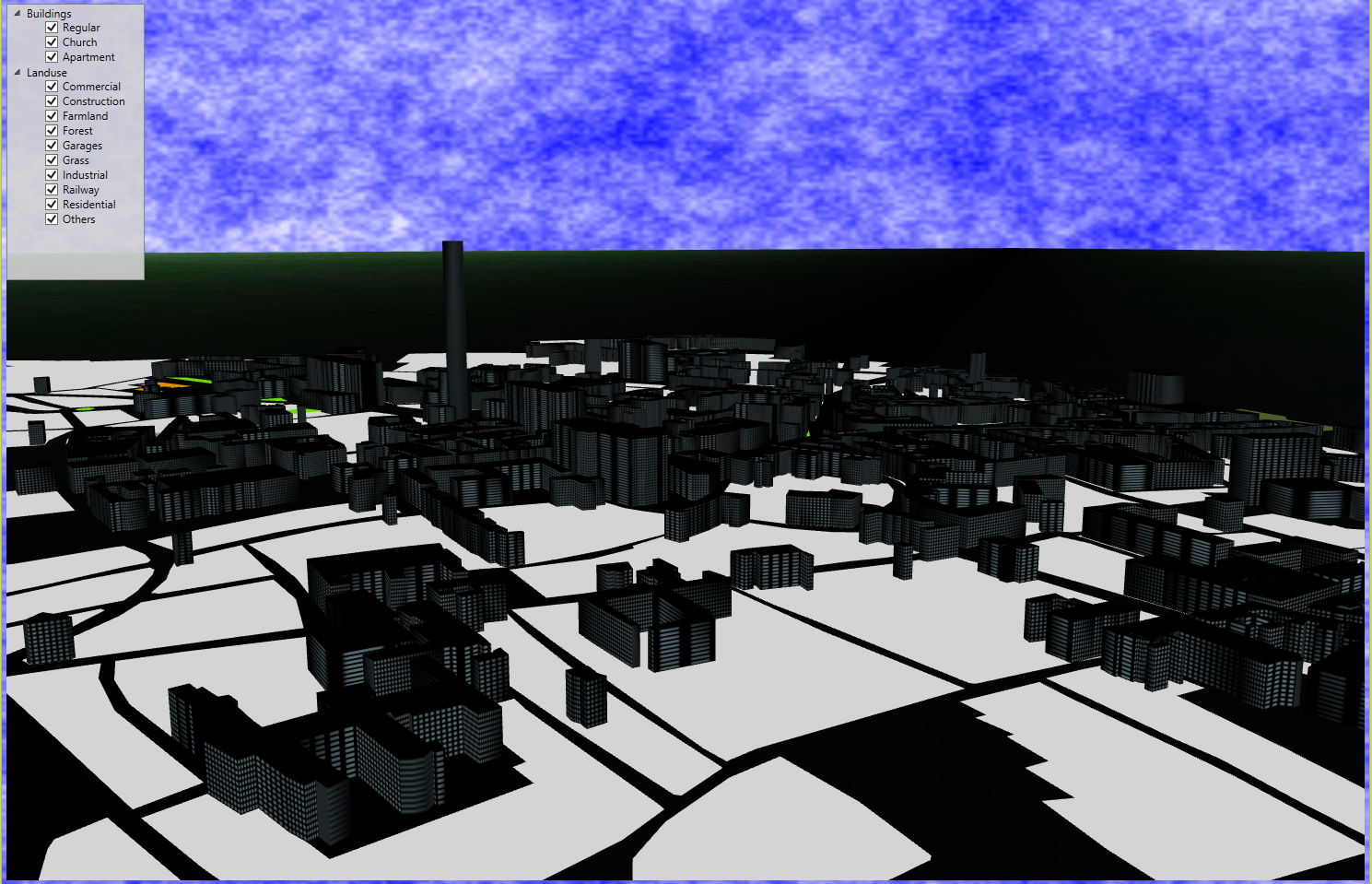
De muren van de gebouwen in de stad werden gecreëerd door het wall texture fractal van eerder. Men kan beslissen hoeveel ramen er in de texture komen. Dan wordt de texture gebruikt op de gebouwen. Om te zorgen dat de afbeelding niet uitgerokken werd moesten een aantal speciale

Het resultaat van het blauwe diamond-square algoritme wordt gebruikt als achtergrond zodat het lijkt of het de lucht is.

Het resultaat van het groene diamond-square algoritme wordt gebruikt als ondergrond zodat het lijkt of het gras is. Om te zorgen dat de startpositie bij het openen van de applicatie er uit ziet alsof je een stad in de voorgrond hebt met gras errond en lucht op de achtergrond moest de oppervlakte van het gras heel groot gezet worden, zodat er zeker geen lucht langs de zijkanten getoont wordt in plaats van gras.

Ook moest het aantal graden van de camera tegenover de grond zeer klein worden zodat men niet alleen maar gras ziet maar ook een horizon met daarna lucht in de achtergrond. Dit was makkelijk realiseerbaar aangezien ik in het tweede project een variabele maakte waarin je de hoek van de camera moet ingeven.

Het resultaat van het gehele proces kan je zien op de afbeelding hieronder.



# Besluit

Fractalen hebben meer toepassingen dan ik aanvankelijk verwachtte. Ik merkte dat fractalen eigenlijk voor iedere virtuele 3D wereld worden gebruikt en dus een belangrijke stap zijn naar het worden van een game developper.

Ook bekijk ik fractalen nu helemaal anders. Vroeger dacht ik altijd dat dit vreemde vormen waren die een soort vreemde virtuele wereld kunnen vormen zoals je vaak als slaapstandbeeld in oudere pc’s ziet. Nu weet ik dat fractalen ook gebruikt kunnen worden voor alledaagse dingen weer te geven en dat fractalen duizend en een vormen kunnen innemen.

Ik heb genoten van deze opdracht omdat ik graag werk met 3D werelden, je voelt je een soort god die een wereld schept. Je kan deze wereld veranderen hoe jij wil en iedere kleine aanpassing jij maakt zal die wereld ook veranderen.

# Bibliografie

Hennig, S. (2011, June 26). *Converting BitmapImage to Bitmap and vice versa.* Opgehaald van Stack Overflow: http://stackoverflow.com/questions/6484357/converting-bitmapimage-to-bitmap-and-vice-versa

Kelleher, C. (2012, December 11). *Tree Fractal.* Opgehaald van Youtube: https://www.youtube.com/watch?v=mYeCV\_jYWGU

Loftis, H. (sd). *Realistic terrain in 130 lines.* Opgehaald van PlayfulJS: http://www.playfuljs.com/realistic-terrain-in-130-lines/

ronnieholm. (2009, Februari 23). *Generating 2D random fractal terrains with C#.* Opgehaald van Bugfree Consulting: http://bugfree.dk/blog/2009/02/23/generating-2d-random-fractal-terrains-with-csharp/

Schertler, N. (2013, September 13). *WPF 3d — understanding texture coordinates.* Opgehaald van Stack Overflow: http://stackoverflow.com/questions/18776406/wpf-3d-understanding-texture-coordinates