sleutel
op je toekomst

3D-renderer

Welke wiskunde schuild er achter een 3D-renderer?

Liam Froyen

Scriptie onderzoekscompetenties
voor Wiskunde

Begeleidende leerkracht: Kris Kerkhofs

Schooljaar 2021-2022

www.sjbzonhoven.be

SJB Zonhoven
Kleine Hemmenweg 4A
3520 Zonhoven

+32 11 55 99 90
+32 11 81 71 38
info@sjbzonhoven.be

sleutel
op je toekomst

3D-renderer

Welke wiskunde schuild er achter een 3D-renderer?

Liam Froyen

Scriptie onderzoekscompetenties
voor Wiskunde

Begeleidende leerkracht: Kris Kerkhofs

Schooljaar 2021-2022

www.sjbzonhoven.be

SJB Zonhoven
Kleine Hemmenweg 4A
3520 Zonhoven

+32 11 55 99 90
+32 11 81 71 38
info@sjbzonhoven.be

Voorwoord

Voor mijn wiskunde eindwerk heb ik de keuze gemaakt om onderzoek uit te gaan voeren over 3D-renderers. Het maken hiervan heeft tegenwoordig niet meer veel nut, aangezien alles nu al klaargemaakt is in veel gebruikte systemen zoals Vulkan en OpenGL.

Toch kies ik er voor om dit te maken, dat doe ik omdat er heel veel uit valt te leren, het kan vaak interessant zijn om te weten hoe deze renderers werken. Later kan deze kennis dan verwerkt worden in shaders en custom render pipelines (bv: HDR, SRP, …).

Voor dit eindwerk heb ik dan ook een 3D-renderer gemaakt die u kan terug vinden op mijn Github-pagina: https://github.com/LiamFroyen/3D-Renderer. Ik heb echter niet een volledige renderer gemaakt, aangezien dat veel te veel werk zou zijn. Mijn programma mist nog een paar essentiële dingen zoals kleuren en schaduwen, maar tegen het einde is er al een duidelijk 3D beeld.

Ook heb ik ervoor gekozen om niet alles uit te leggen maar enkel de essentie van een 3D-renderer. Sommige delen hebben er namelijk niet veel mee te maken maar zijn wel nodig om het programma te laten werken (bv: het openen van een bestand).

Inhoudsopgave

Wat is een 3D-renderer?

1 Functie van 3D-renderers

Een 3D-renderer is een programma om een lijst met informatie om te zetten naar een schijnbaar 3D beeld.

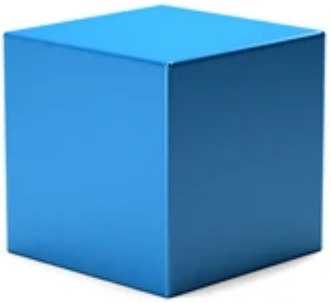
Er kan bijvoorbeeld een bestand ingevoerd worden waar elk punt beschreven staat met een x-, y- en z-coördinaat. De 3D-renderer gaat deze data dan interpreteren en deze punten in een 3D ruimte plaatsen. Daarna worden de punten geprojecteerd met bepaalde wiskundige technieken op het 2D beeldscherm (deze worden later besproken). Zodat er niet enkel een verschil in grootte maar ook in diepte kan gezien worden moeten er echter ook nog schaduwen worden toegevoegd. Zo gaat de 3D renderer een ‘dieptemap’ genereren, die wordt vermenigvuldigd met de kleuren van de punten en zo krijg je echt het gevoel van een 3D beeld op uw 2D beeldscherm.

Het lijkt veel op wat een camera doet, het trekt een foto vanuit 1 bepaald standpunt (de lens) en slaat alle kleuren dat het ziet op (de pixels). Een 3D-renderer gaat hetzelfde doen, enkel dan met een bestand als invoer.

```
cube: {
  p1: (0, 0, 0),
  p2: (5, 0, 0),
  p3: (5, 5, 0),
  p4: (0, 5, 0),

  p5: (0, 0, 5),
  p6: (5, 0, 5),
  p7: (5, 5, 5),
  p8: (0, 5, 5)
}
```

Voorbeeld invoer



Voorbeeld uitvoer

2 Toepassingen

2.1 Games

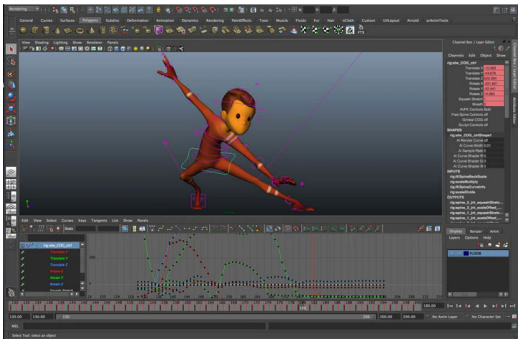
Een van de duidelijkste toepassingen zijn computerspellen, elk 3D-spel is gemaakt met behulp van een 3D-renderer. In het begin waren deze renderers heel simpel, een goed voorbeeld daarvan is het spel “Doom”. Maar nu is het zeer veel vooruitgegaan en bestaan er spellen die bijna echt lijken zoals de game “Cyberpunk 2077”.

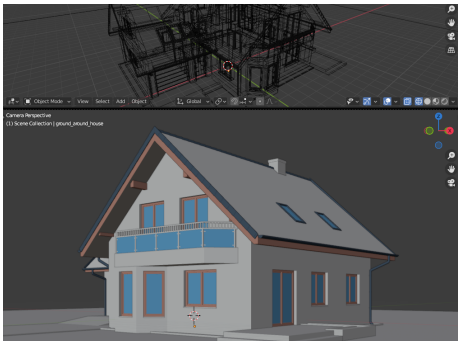
Doom

Cyberpunk 2077

2.2 Blender, Maya, ...

Ook een mooie toepassing zijn programma’s zoals Blender, Maya, … Hierin worden alle 3D-animatiefilms gemaakt, denk maar aan iconische films als “Finding Nemo” of “Shrek”. Ook CGI (Computer Generated Imagery) wordt hierin gemaakt, bijvoorbeeld als de camera in een film door een deurgat gaat. Tenslotte wordt er ook vanalles in de architectuur mee gedaan. Al de foto’s van een gepland huis worden gegenereerd door een 3D-renderer.





2.3 Besturingssystemen

Ook elk besturingssysteem gebruikt een 3D-renderer, een goed voorbeeld daarvan zijn folders. In Windows worden deze getekend met een paar foto's van bestanden die er in zitten. Maar u kunt duidelijk zien dat deze foto's er in het 3D in zijn gezet, dus om die icoontjes te genereren hebben ze een 3D-renderer nodig.

De geschiedenis

1 Begin van computer graphics

In 1960 begon de interesse in computer graphics voor grote bedrijven, een van de belangrijkste was IBM. Zij brachten een van de eerste home gaming console de “Odyssey” uit. Het kon witte punten over een scherm laten bewegen.

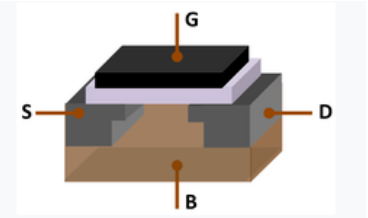
Zes jaar later (1966) kwam Ivan Sutherland met het idee van een head-mounted display (HMD). Dat is de voorganger van de VR-bril, het waren twee schermen op een bepaalde afstand van elkaar gezet zodat het beeld 3D leek. Maar dat idee was niet verder uitgewerkt door de grote hoeveelheid aan apparatuur die nodig was voor deze bril te maken.

Sutherland is dan wel uitgenodigd door Dave Evans om samen met de universiteit van Utah te gaan werken, wat later één van de belangrijkste onderzoek centrums naar computer graphics zou zijn.

Even later zou Arthur Appel met de eerste vorm van “ray casting” bekend maken. Dat was een algoritme om lichtstralen na te bootsen en het licht dat weerkaatst naar de camera te tekenen, zo ontstonden er meer realistische schaduwen. (Wikipedia, 28 januari 2022)

2 Introductie van de MOS

In 1970 kwam er een gigantische technologische verandering, de eerste halfgeleider “MOS” (Metal Oxide Semiconductor) werd uitgevonden. Hierdoor werd het mogelijk om al de gigantische computeronderdelen te verklein tot een simpele chip. Dat betekende ook dat de werkkracht enorm verbeterde.



Sutherland en Evans waren ook begonnen met les te geven en een van hun stunden, Edwin Catmull, heeft de eerste 3D animatie gemaakt. In 1972 had hij als eindwerk zijn hand nagemaakt in een zelfgemaakte 3D-renderer. Twee jaar later bracht hij ook de basis van “texture mapping” uit, wat één van de fundamenten van 3D-modeling is.

In 1976 kwam de eerste film met CGI uit, “Futureworld”. Daardoor is er veel meer attentie naar de graphische wereld gekomen. Even Later volgde de uitvinding van “flat shading”, een methode om licht en schaduw op elk vlak van een figuur te bepalen. Ook in de videospellen kwam er veel meer aandacht aan graphics en gingen we op enkele jaren van het spel “Pong” naar “Speed Race”, een volledig 3D racing spel. (Wikipedia, 28 januari 2022)

3 “Gaming-boom”

In 1980 begon de “gaming-boom”, er werd zeer veel geïnvesteerd in het maken van nieuwe 16-bit CPU’s en de eerste GPU werd uitgevonden. Dat was een revolutie die zorge voor high-resolution graphics. Spellen gingen van 8 kleuren en gemiddeld 32 sprites (de tekeningen waaruit een spel wordt opgebouwd) naar 16 tot 32 kleuren en het aantal sprites werd bijna 10 keer meer. Ook de introductie van VRAM door Texas Instruments was zeer belangrijk, dit zorgde ervoor dat bepaalde frames konden worden opgeslagen op de graphische kaart. Videogames evolueerden hierdoor zeer snel, op 10 jaar gingen ze van bewegende 2D prentjes naar volledig 3D gegenereerde werelden.

Vanaf 1990 begon ook de animatiewereld de 3D trend op te gaan zo kwamen de eerste films uit, zoals Toy Story, Shrek, … (Angel D, 28 januari 2022)

Het maken van een 3D-renderer

Om de wiskunde achter een 3D-renderer te onderzoeken ga ik zelf een renderer maken. Ik heb besloten om het in Java te maken, sinds ik het meest bekend ben met die taal.

Voordat we kunnen beginnen gaan we een venster moeten maken.

1 Een canvas maken

Om een venster te maken in java gebruiken we een canvas, deze maken we aan met enkele instellingen. Later gaan we dan via een bitmap dit canvas vullen met pixels en zo kunnen we beginnen met graphics te tekenen op het scherm.

```
Dimension size = new Dimension(width, height);
setPreferredSize(size);
```

```
//setup frame
frame = new JFrame();
frame.add(this);
frame.setResizable(false);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setTitle(title);

//show frame
frame.setVisible(true);
```

Vergeet niet, om een frame te maken moet de class ‘Canvas’ extenden:

```
public class Display extends Canvas
```

Voor meer extra informatie en instellingen voor JFrames kunt u kijken op https://docs.oracle.com/javase/tutorial/uiswing/components/frame.html.

2 Pixels opslaan

2.1 Een bitmap maken

Om de pixels te gaan kunnen tekenen op het canvas, moeten we deze kunnen opslaan, dat gaan we doen met behulp van een bitmap.

Van elke pixel gaan we de x en y coördinaten opslaan, samen met de rgba-waarde. Zo kunnen we per pixel de locatie, kleur en alpha opzoeken in de bitmap. We gaan dit opslaan in een matrix, de data van de pixel wordt opgeslagen in een matrix ‘P’ en deze matrix wordt dan zelf weer opgeslagen in een algemene matrix ‘B’.

$$P = \begin{bmatrix} a & g & b & r \end{bmatrix} \qquad B = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix}$$

De positie in matrix ‘B’ komt overeen met de x en y coördinaten van de pixel, met x gelijk aan de rijen en y gelijk aan de kolommen. Zo kunnen we makkelijk in onze code een specifieke pixel uitkiezen om mee te gaan werken.

$$B = \begin{bmatrix} \begin{bmatrix} a & g & b & r \end{bmatrix} & \begin{bmatrix} a & g & b & r \end{bmatrix} & \begin{bmatrix} a & g & b & r \end{bmatrix} \\ \begin{bmatrix} a & g & b & r \end{bmatrix} & \begin{bmatrix} a & g & b & r \end{bmatrix} & \begin{bmatrix} a & g & b & r \end{bmatrix} \\ \begin{bmatrix} a & g & b & r \end{bmatrix} & \begin{bmatrix} a & g & b & r \end{bmatrix} & \begin{bmatrix} a & g & b & r \end{bmatrix} \end{bmatrix}$$

Als we deze matrix zouden tekenen, krijgen wij het canvas met alle pixels zoals we het hebben opgeslagen.

2.2 De pixels tekenen

Om de positie te bepalen in de code wanneer we de x en y gegeven krijgen, gaan we de index moeten bepalen. We doen dat een de hand van de formule $index = x + (y \cdot width)$ met width de breedte van het canvas.

$$Index_B = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

Hiermee kunnen we een functie maken waar elk component wordt ingesteld voor de opgegeven pixel.

```
public void DrawPixel(int x, int y, byte a, byte b, byte g, byte r) {
    int index = x + y * width;
    pixels[index].components[0] = a;
    pixels[index].components[1] = b;
    pixels[index].components[2] = g;
    pixels[index].components[3] = r;
}
```

Een pixel tekenen wordt dan zeer gemakkelijk, bijvoorbeeld:

```
DrawPixel(x, y, (byte)0x00, (byte)0xFF, (byte)0xFF, (byte)0xFF);
```

Met behulp van enkele voorgemaakte java componenten kunnen we dan makkelijk ons canvas gelijkstellen aan onze bitmap.

```
private final BufferStrategy    bufferStrategy;
private final Graphics          graphics;
```

```
Bitmap frameBuffer = new Bitmap(width, height);
BufferedImage displayImage = new BufferedImage(width, height,
BufferedImage.TYPE_3BYTE_BGR);
```

```
graphics.drawImage(displayImage, 0, 0, frameBuffer.GetWidth(),
frameBuffer.GetHeight(), null);
bufferStrategy.show();
```

3 Demo: sterren bewegen in het 3D

3.1 Het maken van de sterren

Deze zogenaamde “sterren” gaan witte puntjes zijn die ik laat bewegen doorheen het canvas, dit in combinatie met een zwarte achtergrond zal het gevoel geven van sterren.

Ik begin door de x, y en z coördinaten van elke ster op te slaan in een lijst.

```
private final float starX[];
private final float starY[];
private final float starZ[];
```

Vanaf het moment dat een ster wordt aangemaakt zal hij dan een willekeurige waarde krijgen voor elke dimensie. Voor de x- en y-waarde zal dat $2 \cdot (k - 0,5) \cdot spread$ zijn en voor de z-waarde is dat $k \cdot spread$, met k een willekeurig getal tussen 0 en 1 en spread een waarde voor hoe verspreidt de sterren van elkaar moeten zijn.

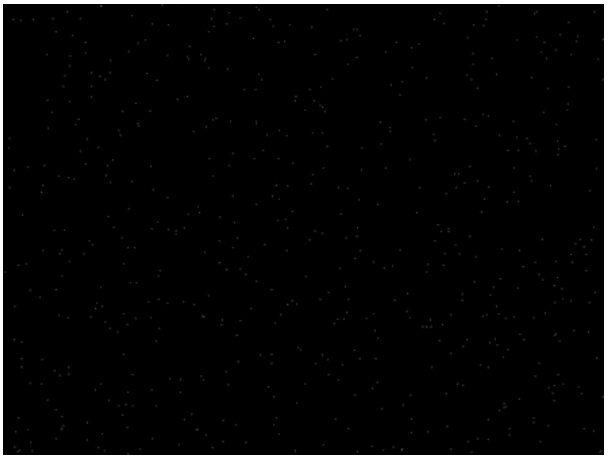
```
starX[i] = 2 * ((float)Math.random() - 0.5f) * spread;
starY[i] = 2 * ((float)Math.random() - 0.5f) * spread;
starZ[i] = ((float)Math.random()) * spread;
```

Daarna update ik mijn bitmap met witte pixels op de locaties van de sterren en de z locatie laat ik elke update met een bepaalde snelheid afnemen, zodat je het effect zou krijgen dat de sterren naar u toe komen.

```
starZ[i] -= delta * speed; //met delta de tijd sinds de vorige update
```

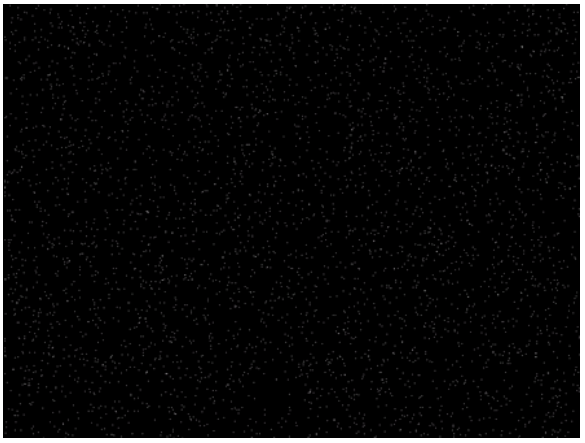
```
target.DrawPixel(x, y, (byte)0x00, (byte)0xFF, (byte)0xFF, (byte)0xFF);
```

Het resultaat hiervan is echter niet zo spectaculair, dat komt omdat we het nog niet 3D hebben gemaakt. De pixels veranderen in z-waarde, maar dat tekenen we niet op het canvas. Het enige wat we eigenlijk zien zijn sterren die te ver weg gaan verdwijnen en nieuwe sterren die worden aangemaakt.



We lossen dit probleem op door de x- en y-waarden van alle sterren te delen door hun z-waarde. Door deze eenvoudige bewerking veranderd het resultaat drastisch en ziet het er echt 3D uit.

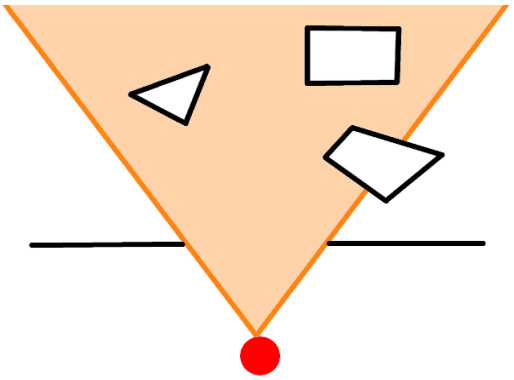
```
int x = (int)(starX[i]/starZ[i]);
int y = (int)(starY[i]/starZ[i]);
```

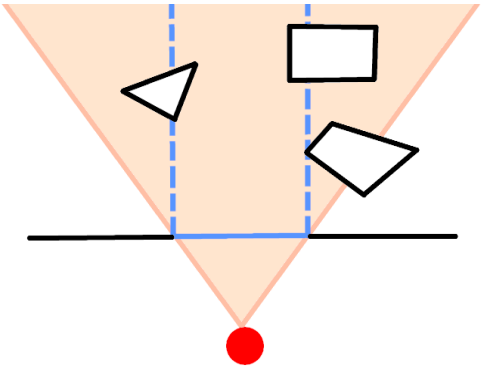


(Voor in het geval dat de gifs niet bewegen, is er een YouTube-link met beide resultaten: https://youtu.be/7_35KQsE97w)

3.2 Hoe werkt dit

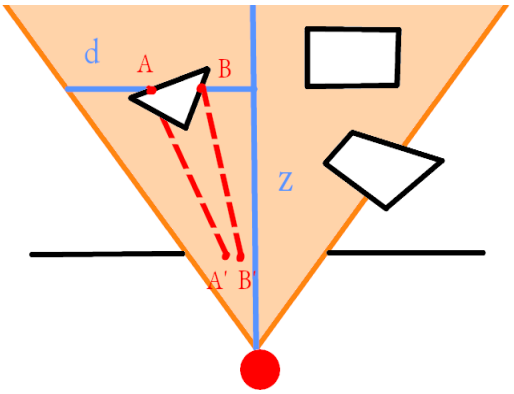
Stel u voor dat een persoon tussen 2 muren door kijkt, dan kunnen we duidelijk het zichtveld afbakenen. Als we dan dat beeld tussen die twee muren zouden opvullen met een foto van exact wat die persoon zag vanuit zijn standpunt, dan zou het lijken alsof de muren nooit waren opgevuld. Dat is wat wij proberen te bereiken, maar hoe gaan we dan dat namaken?

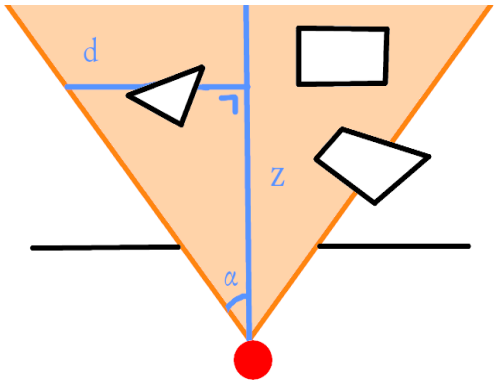




Je zou dan kunnen proberen alles wat achter het gat zit te projecteren op het gat, maar dan heb je al duidelijk het probleem dat het helemaal niet hetzelfde is als het gezichtsveld van de persoon.

We trekken een lijn in het midden van het zichtveld van de persoon, genaamd z. Loodrecht op deze z wordt een lijn d getrokken, deze strekt tot het einde van het zichtveld. Als we dan de x- en y-waarden delen door d, zullen deze punten kunnen geprojecteerd worden tussen de 2 muren net zoals de persoon het ziet.





Deze d berekenen is dan heel simpel, want zoals je duidelijk kan zien werken we in een rechthoekige driehoek. Dus m.b.v. sin, cos en tan kan je makkelijk alle zijden berekenen zolang je de hoek hebt waaronder de persoon kan zien en de afstand. In het geval van onze sterren, weten we beiden.

Als we hier d gaan berekenen komen wij uit op $d = bgtan(\alpha) \cdot z$, we delen elk punt door d en dan krijgen we ons gewenst eindresultaat.

Maar klopt dit dan met wat we eerder hebben ontdekt?

```
int x = (int)(starX[i]/starZ[i]);
int y = (int)(starY[i]/starZ[i]);
```

$$\frac{x}{z} = \frac{x}{1 \cdot z} = \frac{x}{bgtan(90°) \cdot z}$$

$$\frac{y}{z} = \frac{y}{1 \cdot z} = \frac{y}{bgtan(90°) \cdot z}$$

Dit klopt dus inderdaad, wat wij eerder hebben gezien was een zichtveld onder een hoek van 90 graden.

4 Driehoeken maken

Natuurlijk, deze punten vormen nog geen objecten, daarvoor gaan we eerst nog figuren moeten kunnen tekenen. Dat gaan we doen door lijnen te tekenen, daarmee gaan we dan driehoeken maken, waarmee we op zijn beurt weer figuren kunnen vormen. Om dan het 3D-effect te bekomen, kunnen we gewoon weer dezelfde oplossing gebruiken als voor de punten.

4.1 Lijnen tekenen

Om een lijn te definiëren gaan we gebruik maken van een extra matrix, de 'Scan Buffer'. Dat is een ΔYx2-matrix met de kleinste en grootste X-waarde voor een bepaalde Y-waarde.

$$S = \begin{bmatrix} \mathrm{xMin}_1 & \mathrm{xMax}_1 \\ \mathrm{xMin}_2 & \mathrm{xMax}_2 \\ \mathrm{xMin}_3 & \mathrm{xMax}_3 \end{bmatrix}$$

We kunnen deze dan makkelijk gaan invullen, aangezien we een lijn willen tekenen. Een lijn kan namelijk altijd aangegeven worden volgens een eerstegraads functie: $y = ax + b$. Dus als we het startpunt en eindpunt hebben van de lijn, kunnen we de rico bereken: $a = \frac{\Delta x}{\Delta y}$.

Aan de hand van een for-loop gaan we deze functie aflopen startend bij de kleinste Y-waarde tot de grootste Y-waarde van de lijn. We maken een X-waarde aan, behorend tot de kleinste Y-waarde en aan deze voegen we elke iteratie de rico toe. Om de index te bepalen voor deze functie kunnen wij gewoon de iteratie vermenigvuldigen met twee aangezien er twee kolommen zijn.

```
public void ScanConvertLine(int yStart, int yEnd, int xStart, int xEnd) {
    int deltaY = yEnd - yStart;
    int deltaX = xEnd - xStart;

    float ricoX = (float) deltaX/deltaY;
    float currX = (float) xStart;

    for (int i = yStart; i < yEnd; i++) {
        scanBuffer[i * 2] = (int) currX;
        currX += ricoX;
    }
}
```

Daarna kunnen we de Bitmap gaan opvullen met de Scan Buffer met een gelijkende for-loop. We beginnen bij de kleinste Y-waarde en eindigen bij de grootste Y-waarde, daarna gaan we van de kleinste X-waarde tot grootste X-waarde en vullen we deze plaatsen op aan de hand van de handige 'DrawPixel' functie.

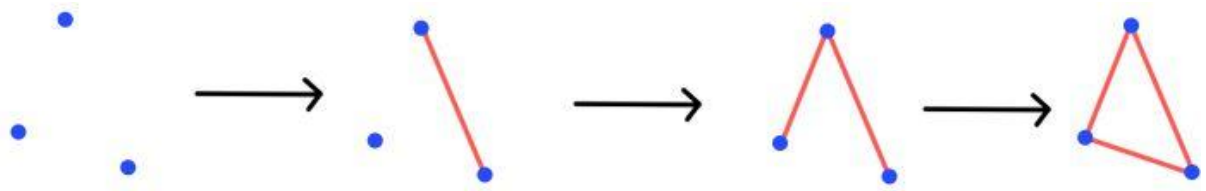
```
public void FillShape(int yMin, int yMax) {
   for (int i = yMin; i < yMax; i++) {

        int xMin = scanBuffer[i * 2    ];
        int xMax = scanBuffer[i * 2 + 1];

        for (int j = xMin; j < xMax; j++) {
            DrawPixel(j, i, (byte)0xFF, (byte)0xFF, (byte)0xFF, (byte)0xFF);
        }
   }
}
```

4.2 Driehoeken tekenen

Om de driehoeken te tekenen kunnen we dan heel simpel de drie randen maken in onze Scan Buffer aan de hand van de 'ScanConvertLine' functie. Het is echter zeer belangrijk dat we het volgens het hoogste tot laagste punt, dan het laagste tot middelste en tenslotte het middelste tot laagste punt ingeven.



Als we dit doen en daarna pas de Scan Buffer tekenen, zal (door het overschrijven van bepaalde waarden dankzij deze volgorde) de volledige driehoek worden opgevuld.

Het sorteren van deze punten zal ik niet uitleggen, maar kunt u wel terug vinden in de source code[1].

[1] De Github-pagina vermeld in het voorwoord.

5 Transformeren

Om beweging te creëren in de driehoeken kunnen we de input dat laten doen, zoals bij de sterren demo, maar dat is zeer onhandig. We kunnen ook ons programma dat automatisch laten doen. Om hieraan te beginnen moeten we de punten van een driehoek kunnen transformeren.

5.1 De transformatiematrix

Het simpelst is om een 4x4-matrix 'T' te maken met de x, y, z en w componenten. De som van een rij stelt het getransformeerde coördinaat en elke kolom stelt voor de factor met het originele coördinaat voor. Zo is de getransformeerde $x_E = x_1x + y_1y + z_1z + w_1w$ voor:

$$\mathrm{T} = \begin{bmatrix} \mathrm{x}_1 & \mathrm{y}_1 & \mathrm{z}_1 & \mathrm{w}_1 \\ \mathrm{x}_2 & \mathrm{y}_2 & \mathrm{z}_2 & \mathrm{w}_2 \\ \mathrm{x}_3 & \mathrm{y}_3 & \mathrm{z}_3 & \mathrm{w}_3 \\ \mathrm{x}_4 & \mathrm{y}_4 & \mathrm{z}_4 & \mathrm{w}_4 \end{bmatrix}$$

Stel we nemen als invoer het punt P(1, 0, 5, 2) en we gaan het transformeren volgens de transformatiematrix T':

$$\mathrm{T'} = \begin{bmatrix} 2 & 1 & 0 & 5 \\ 0 & 3 & 5 & -15 \\ 5 & 2 & -1 & 0 \\ 1 & 1 & 3 & 15 \end{bmatrix}$$

Dan krijgen we als uitvoer:

$$x_E = 2 \cdot 1 + 1 \cdot 0 + 0 \cdot 5 + 5 \cdot 2 = 12$$
$$y_E = 0 \cdot 1 + 3 \cdot 0 + 5 \cdot 5 + (-15) \cdot 2 = -5$$
$$z_E = 5 \cdot 1 + 2 \cdot 0 + (-1) \cdot 5 + 0 \cdot 2 = 0$$
$$w_E = 1 \cdot 1 + 1 \cdot 0 + 3 \cdot 5 + 15 \cdot 2 = 46$$

$$P'(12, -5, 0, 46)$$

5.2 Verplaatsen

In mijn programma ga ik echter nooit de w-component vragen, maar deze ga ik altijd als ‘1’ ingeven. Zo heb ik in mijn transformaties altijd een neutraal component om mee te werken.

Voor een verplaatsing kan ik dan een functie maken die een x_i, y_i en z_i component als invoer neemt. Dan maak ik een transformatiematrix volgens:

$$\mathrm{T} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Zo krijgen we bijvoorbeeld voor $x_E = x + 0 + 0 + x_i = x + x_i$ (onze verplaatsing).

5.3 Vergroten

Het vergroten is ook zeer simpel. We kunnen gewoon elk component vermenigvuldigen met zijn toebehorende input. De transformatiematrix ziet er dan als volgend uit:

$$\mathrm{T} = \begin{bmatrix} x & 0 & 0 & 0 \\ 0 & y & 0 & 0 \\ 0 & 0 & z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Zo krijgen we bijvoorbeeld voor $x_E = x \cdot x_i + 0 + 0 + 0 = x \cdot x_i$ (onze vergroting).

5.4 Roteren

Om een punt te roteren gaan we deze eerst opsplitsen in drie transformatiematrices:

$$T_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(x) & -sin(x) & 0 \\ 0 & sin(x) & cos(x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_y = \begin{bmatrix} cos(y) & 0 & -sin(y) & 0 \\ 0 & 1 & 0 & 0 \\ sin(y) & 0 & cos(y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

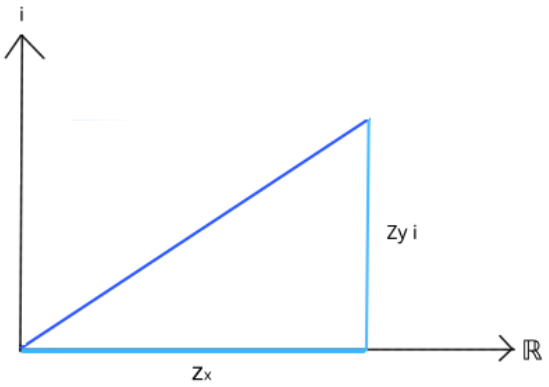
$$T_z = \begin{bmatrix} cos(z) & -sin(z) & 0 & 0 \\ sin(z) & cos(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De keuze van deze transformatiematrices is het makkelijkst uit te leggen aan de hand van de T_z, als we deze weer uitschrijven verkrijgen we:

$z_{E_x} = x \cdot cos(z) - y \cdot sin(z)$ en

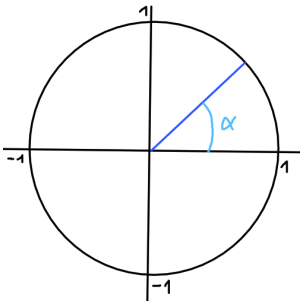
$z_{E_y} = x \cdot sin(z) + y \cdot cos(z)$

We krijgen dus een vector $z_E = (z_{E_x}, z_{E_y}, z, w)$, dat doen we om een rotatie mogelijk te maken. We kunnen deze namelijk vergelijken met een punt in het complexe vlak. Als reëel getal nemen we z_{E_x} en als imaginair getal $z_{E_y} i$. Als we dat ook doen voor de hoek waarmee alles moet draaien en deze twee vermenigvuldigen krijgen we een rotatie in het complexe vlak (eigenschap complexe getallen):



$$(z_x + z_y i) \cdot (r_x + r_y i) = -z_y r_y + r_x z_y i + z_x r_y i + z_x r_x$$
$$= (z_x r_x - z_y r_y) + (z_x r_y + z_y r_x)i$$

Omdat we werken in de transformatiematrix van z kunnen we zeggen dat $z_x = x$ en $z_y = y$. We kunnen r_x en r_y bepalen aan de hand van een goniometrische cirkel.



Stel we zouden z willen roteren met de hoek α, dan kunnen we het punt dat overeenkomt met deze hoek op de goniometrische cirkel als coördinaat gebruiken voor r(cosα, sinα).

Als we dit invullen krijgen we:
$(x \cdot cos(\alpha) - y \cdot sin(\alpha)) + (x \cdot sin(\alpha) + y \cdot cos(\alpha))i$

Deze splitsen we dan weer op in z_{E_x} en z_{E_y}, zo komen we aan:

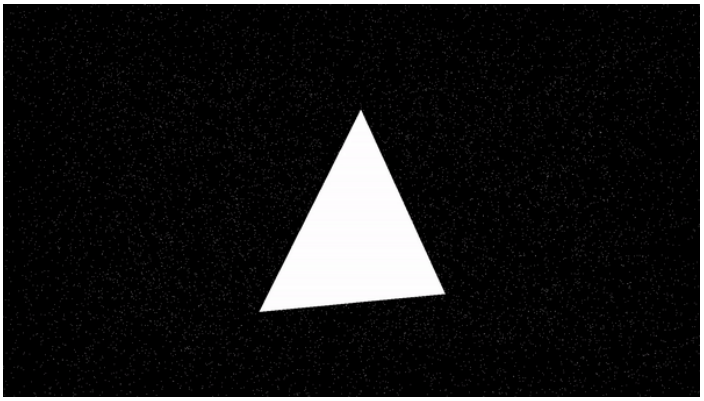
$z_{E_x} = x \cdot cos(\alpha) - y \cdot sin(\alpha)$ en

$z_{E_y} = x \cdot sin(\alpha) + y \cdot cos(\alpha)$

Dat gaan we voor elk component doen en als we deze op het einde dan gaan vermenigvuldigen met elkaar in de volgorde $x_E \cdot y_E \cdot z_E$ krijgen we terug ons eerste punt maar dan geroteerd over een bepaalde hoek.

5.5 Resultaat

Om deze berekeningen uit te proberen heb ik een gelijkzijdige driehoek laten draaien over onze sterren, het resultaat ziet er uit als volgt:

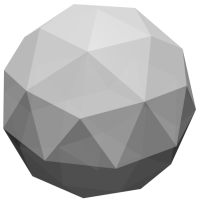


(Voor het geval dat de gif niet werkt is er een YouTube-link: https://youtu.be/dcsBcAaC_Dw)

6 Meshes

6.1 Invoer

Als laatste stap ga ik zorgen de we meshes kunnen maken, dat zijn 3D-figuren opgebouwd uit verschillende driehoeken. Als voorbeeld ga ik met een ico-sphere werken, te zien op de afbeelding.

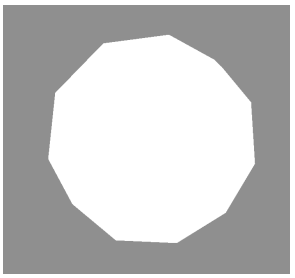


Om een meshes als input te geven maak ik gebruik van .obj bestanden, die van de ico-sphere ziet er als volgt uit:

```
v 0.000000 -1.000000 0.000000
v 0.723607 -0.447220 0.525725
v -0.276388 -0.447220 0.850649
v -0.894426 -0.447216 0.000000
v -0.276388 -0.447220 -0.850649
v 0.723607 -0.447220 -0.525725
v 0.276388 0.447220 0.850649
v -0.723607 0.447220 0.525725
v -0.723607 0.447220 -0.525725
v 0.276388 0.447220 -0.850649
v 0.894426 0.447216 0.000000
…
```

Het lezen van deze data zal ik niet uitleggen aangezien het niet veel met een 3D-renderer te maken heeft. Maar eenmaal dat we de data hebben geanalyseerd kunnen we zeer simpel driehoeken maken met deze coördinaten.

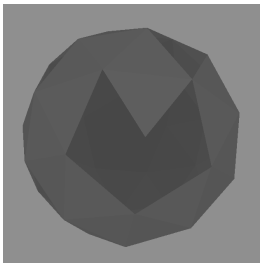
Het resultaat hiervan is echter enkel de contour van de mesh, dat komt omdat we nog geen kleuren of schaduwen hebben toegevoegd.



6.2 Diepte maken

Omdat ik niet een volledig licht- of kleurensysteem ga maken heb ik een omweg gevonden. Om toch het gevoel van diepte te creëren neem ik de gemiddelde z-waarde van elk punt van elke driehoek en daarop gebaseerd geef ik ze een grijswaarde.

Maar er valt snel op dat er nog steeds iets mis is, er zitten gaten in de mesh. Dat komt doordat we de achterkant van de ico-sphere na de voorkant tekenen. Aangezien deze op dezelfde x- en y-waarde staan gaan ze overlappen en zal de laatst getekende altijd zichtbaar zijn.



Om dat te vermijden kan ik testen als de driehoek wegkijkt van de camera. Als dat het geval is moet de driehoek niet getekend worden.

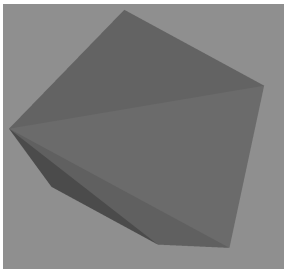
Het is misschien wel al duidelijk dat dit niet werkt voor alle meshes, het is dus een tijdelijke oplossing. Een betere oplossing zou zijn om nog een variabele aan te maken die alle z-waarden opslaat en als er twee zouden overlappen wordt de dichtstbijzijnde gekozen. Maar ik heb besloten om het toch hierbij te laten aangezien er al goede resultaten uit komen.

7 Eindresultaat

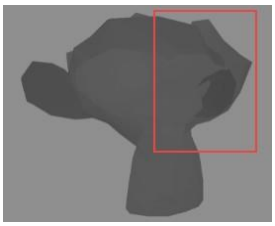
Om een mooi eindresultaat te creëren heb ik verschillende meshes laten draaien, deze staan in de bijlage.

7.1 Problemen

Ze zien er al geloofwaardig 3D uit, maar er zijn duidelijk nog een paar dingen mis. Mijn omweg om diepte te creëren werkt niet altijd even goed omdat er geen rekening wordt gehouden met stukken die in de schaduwen liggen van andere delen. Daardoor zie je bijvoorbeeld bij het apen hoofd niet duidelijk het verschil tussen het oog en de omliggende driehoeken.



Ook zorgt deze manier niet voor vloeiende overgangen waardoor je (vooral duidelijk bij de kubus) verschillende grijswaarden in één vlak krijgt.



Zoals eerder besproken is mijn oplossing voor het voorste driehoek te bepalen niet de beste manier. Dat valt vooral op bij de complexere meshes, zoals het apen hoofd en de boom.

7.2 Verdere uitwerking

Als ik het programma verder af zou maken, zou ik het volgende doen:

- Aparte variabele maken om enkel de voorste z-waarde te tekenen
- Kleuren/gradiënten toevoegen
- Interpolatie tussen kleuren (mooie overgangen)
- Texture mapping (foto's op de vlakken kunnen tekenen)
- Clipping (zorgen dat meshes ook buiten het venster kunnen staan)
- Een lichtsysteem
- Muis beweging (met een muis de meshes kunnen rondbewegen)

Literatuurlijst

Computer Graphic. (13.01.2022). [28.01.2022, Wikipedia: https://en.wikipedia.org/wiki/Computer_graphic].

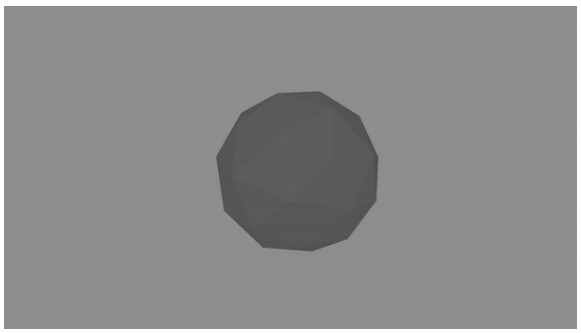
Angel D. (19.04.2021). *THE HISTORY OF 3D CGI [From 1965 to 2021]*. [28.01.2022, YouTube: https://www.youtube.com/watch?v=_DNzTCd7-jc].

TheBennyBox. (26.11.2014). *3D Software Rendering*. [08.09.2021, YouTube: https://www.youtube.com/playlist?list=PLEETnX-uPtBUbVOok816vTl1K9vV1GgH5].

Bijlagen

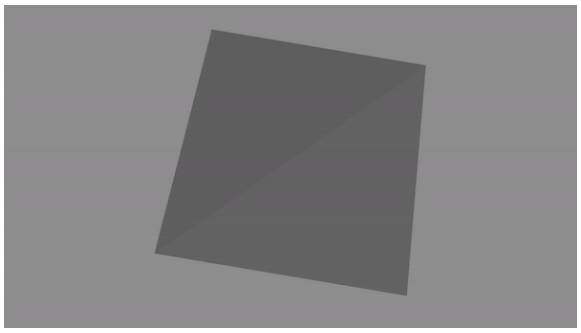
Bijlage 1: Resultaten

Ico-sphere



Een apen hoofd

kubus



Een boomstam

Voor het geval dat de gifs niet bewegen is er een YouTube-link met alle resultaten: https://youtu.be/dYnnftQWAM0)