**VIA University College**

**MSU-Rapport**

**Sofie Zhuk Sørensen - 343994**

**[Navne på studerende - Studienumre]**

**[Navn(e) på vejleder(e)]**

**[Antal tegn]**

**Software ingeniør**

**2. semester**

**[Dato]**

**Indhold**

[1. Resumé 2](#_Toc1355137138)

[2. Analyse af eksisterende transformationer 3](#_Toc768324276)

[2.1 Matricer og transformationer i vertex shaderen. 4](#_Toc1076358420)

[2.2 Formålet med - og effekten af transformationerne. 4](#_Toc159010913)

[3. Modifier – Matematisk model 5](#_Toc36991396)

[3.1 Twist 5](#_Toc1321954872)

[3.2 Translation 5](#_Toc1732876593)

[4. Udvikling af egen modifier 6](#_Toc274627130)

[5. Refleksion 7](#_Toc535459010)

[6. Referencer 9](#_Toc1954380742)

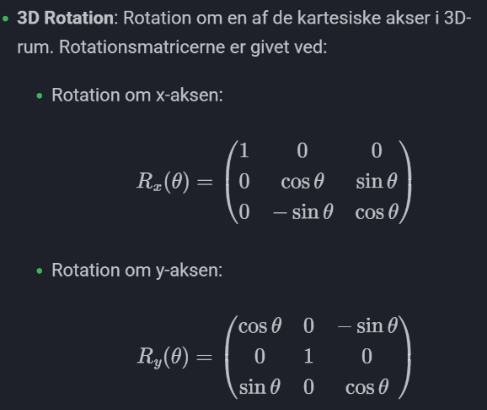
# Resumé

# Analyse af eksisterende transformationer

I vores web shader har vi fået udleveret kode som kan rotere et objekt baseret på mussemarkørens bevægelse på skærmen. I dette afsnit vil vi dykke ned i vertex shaderen, og hvordan linær algebra bliver benyttet til at transfomere vores objekter.

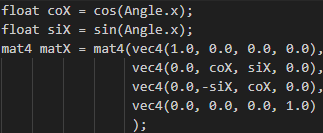
## 2.1 Matricer og transformationer i vertex shaderen.

Vertexshaderen er hvor vi kan manipulere vertexer, og det er her vi finder rotationsmatricerne. Der bliver anvendt henholdsvist en matrice for rotation omkring x- og y-aksen, hvilket kræver to forskellige matricer, som kan ses på Figur 1. Det er dog ikke nok til at kunne benytte direkte i vores vertex shader, for selvom den 4. dimension er ikke vigtig i forhold til rotationer, er uundværligt for translationer. Derfor, udvider vi vores kolonne og rækker med *w*, som vi sætter til identifikationen 1. Denne fremsynethed kommer os til gavn da vi senere vil beskrive hvordan vi har lavet vores egen spiralmodifier.

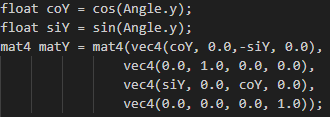


Figur 1: 3D Rotationer for x- og y-aksen

Efter at have omskrevet rotations matricerne til 4x4 vektorer, bliver de implementeret i vertexshaderen, se Figur 3 og Figur 2.



Figur 2: Rotation om x-aksen



Figur 3: Rotation om y-aksen

Det er vigtigt for at kunne rotere noget i 3D, at vi benytter 4 dimensioner. Grunden er den, at hvis vi udelukkende benyttede os af 3x3 matricer, så ville vi kun kunne lave rotationer og skalering. Vi vil altså ikke kunne translatere (flytte) objektet, hvilket er vigtigt for vores egen modifier – spiralen.

Disse to matricer bliver i vertexshaderen multipliceret med den originale position for derefter at kunne opnå en samlet rotation i vores kartesiske koordinatsystem.

## 2.2 Formålet med - og effekten af transformationerne.

Formålet med transformationerne er at kunne interagere med objektet, ved at rotere det i vores canvas.

Rotationerne er blevet implementeret med mat4 matricer og anvender differencen mellem 2 mussepositioner til at finde vinklen for rotationsgraden. Når mussetasten bliver klikket ned, til når musen ikke længere bevæger sig eller at mussetasten bliver sluppet igen, bliver musens bevægelser altså omregnet til thetra i rotationsmatricen, i forholdsvis x- og y-aksen.

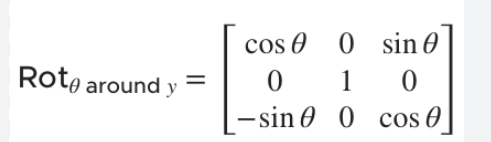


Figur 4: Eventlistner angiver musens position, så vinkel på rotation kan beregnes

# Modifier – Matematisk model

I dette kapitel vil vi gennemgå en twist modifier, og hvordan den kan implementeres

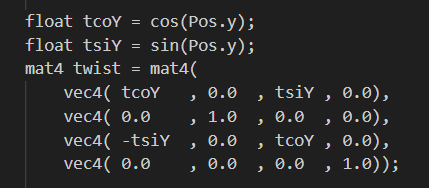
## 3.1 Twist



Figur 5: Formel for rotationsmatrix omkring y aksen

Den twist funktion vi har taget udgangspunkt i er baseret på en simpel rotation omkring y-aksen, baseret på en verteks y værdi. Ved at tage udgangspunkt i nedestående rotations matrix (Wikipedia, u.d.), for en rotation omkring y aksen.

For at implementere den i vores vertex shader, implementerede vi nedestående kode. Her baserede vi vinklen af rotationen på y koordinatets størrelse (Pos.y), for derfor at få en vertical twist.



Figur 6: Implementering af vertex shader code for twist matrix

## 3.2 Translation

A number with white letters

AI-generated content may be incorrect.En translation er at forskyde noget fra et sted til et andet. For at kunne foretage dette med en lineær translation, bliver vi nødt til at benytte homogene koordinater. Dette betyder at vi tilføjer en ekstra dimension til vores objekt, og benytter dette til forskydningen.

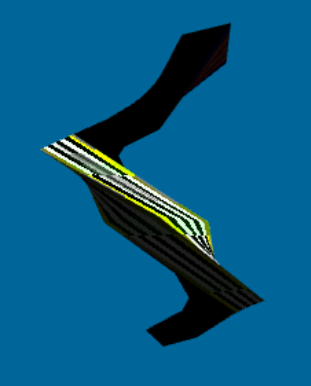
Figur 6: Translations matrice for søjlevektorer

I vores tilfælde arbejder vi med 3 dimensionelle objekter, så vi benytter derfor den 4. dimension til forskydningen. Derved får vi matricen illustreret som en 4x4 matrix.

I matricen ovenfor definerer tx, ty og tz hvor meget vi vil flytte et objekt i heholdvis i x-, y- og z-retningen.

# Udvikling af egen modifier

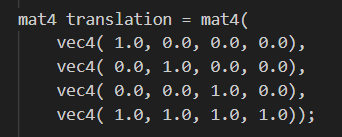
Ved implementering af denne modifier, er det nødvendigt at anvende to forskellige matricer; en rotationsmatrice (twist om y-aksen) og en translations matrice. Denne udvikling af egen modifier har vi tilladt os selv at kalde for en Spiral. Dette gør at vi får en afin modifikation.



Figur 7: Vores egen modifier implementerert i WebGL.

Implementeringen af rotationsmatricen omkring y-aksen kan ses på Figur 6, angående modifieren, twist.

Vi har i vores kode implementeret, så at vores Box flytter alle punkter én enhed i hver retning. Vi har dermed implementeret translationsmatricen i 3D således:



For at udføre denne vores modifier, er det vigtigt at translationen udføres først, så vores Box flyttes væk fra origo, og derefter rotere sig rundt om den origo (z-aksen), og dermed får vores Box til bevæge sig i cirkler rundt om vores z-akse.

Hvis vis vore Box derimod roterede først, ville den rotere om sin egen origo, og derefter ville hele objektet bevæge sig væk fra origo, eftersom rotationen ikke har en effekt på translationen. Det er derfor vigtigt translationen udføres først, således:



I WebGL (GLSL-shader) arbejdes der med søjlevektorer, hvor at transformationerne udføres fra højre til venstre, hvilket ville være omvendt hvis der var tale om rækkevektorer. Derfor bliver transformationerne udført fra venstre af, og vil sige, at den matrice der står tættest på vektoren, bliver udført som det første.

Det er derfor vigtigt at have translationen til venstre vore vektor, så translationen udføres før vores rotation.

# Refleksion

* Forklar tydeligt, hvordan lineære transformationer anvendes praktisk inden for områder som 3D-grafik, spiludvikling eller visuel simulering.
* Tag udgangspunkt i Blender

Linære transformationer bruges oftes i 3D-grafik, spiludvikling og visuel simulering. I denne refleksion vil vi bruge eksempler fra Blender, Unity og det vedlagte projekt fra WWW undervisningen på dette semester.

**Blender**

Blender er et 3d modelerings program der bruger vertecies og meshes til at oprette 3d modeller. Her spiller linær algebra en stor rolle i den måde man interagere og manipulere modellen på. Den mest benyttede, men nok også mest oversete form for linær transformation i blender er nok translation. Det føles nemt og intuativt at gribe en 3d model, og så trække den rundt på skærmen. Det samme gør sig gældende for rotationer og skaleringer. Disse funktier er så simple at man ikke ænser hvordan de fungerer.

Blender tilbyder også andre modfiers af forskellige visuelle kompleksiteter. Fx er mirror modifieren enorm nyttig, hvor modifiers som ”hook”, kan fremstille komplicerede former. En mirror modifier bruges til at spejle vertecies over en akse, mens hook modifieren imitere en krogs påvirkning af en form. En enorm nyttig sidegevinst af at bruge linære transfomationer, er at de altid kan inverteres. Det vil sige at man nemt kan komme tilbage til sit originale udgangspunkt, uden at have lavet permanente ændringer til sin model, hvis man fortryder nogle modifikationer. En anden gevinst er at linære transformationer kan kombineres.

Når alle disse simple transformationer kombineres til alverdens komplicerede former, er det umuddelbart ikke særligt meget mere beregningstungt, da alle de linære transformationer kan samles til en enkel transformations matrix, og derfor kun behøves udregnes en enkelt gang før den benyttes.

I det vedlagte projekt er der brugt linær algebra til at konstruere 2d repræsentationer af 3 dimensionelle former, og herefter ændret med linære transformations matricer. Vertecies positioner, UV maps og normals bliver alt sammen beskrevet med matricer, og ved at konstruere dem som ligesidede matricer, kan de nemt manipuleres matematisk.

Et andet eksempel på brugen af linær algebra i visuelle repræsentatinoer er i billede redigering. Programmer som photoshop og Krita har begge mange filtre man kan putte på, som på en eller anden måde forvrænger, strækker, flytter, roterer eller på anden måde manipulerer pixels.

# Referencer

[Indsæt din tekst her]

Find mere information i “Guide til projektarbejde - Projektrapport”.