H.C. Ørsted Gymnasiet

Opgaveformulering SOP 2022-23

Klasse:	3d2
Navn:	Alexander Hillerup Tuff
Fødselsdato	

Fag:	**Niveau:**	**Vejleder navn:**	**Vejleders mailadresse:**
Matematik	A	Lasse Rønbøg Møholt	lrm@tec.dk
Programmering	B	Kristian Krabbe Møller	kkm@tec.dk

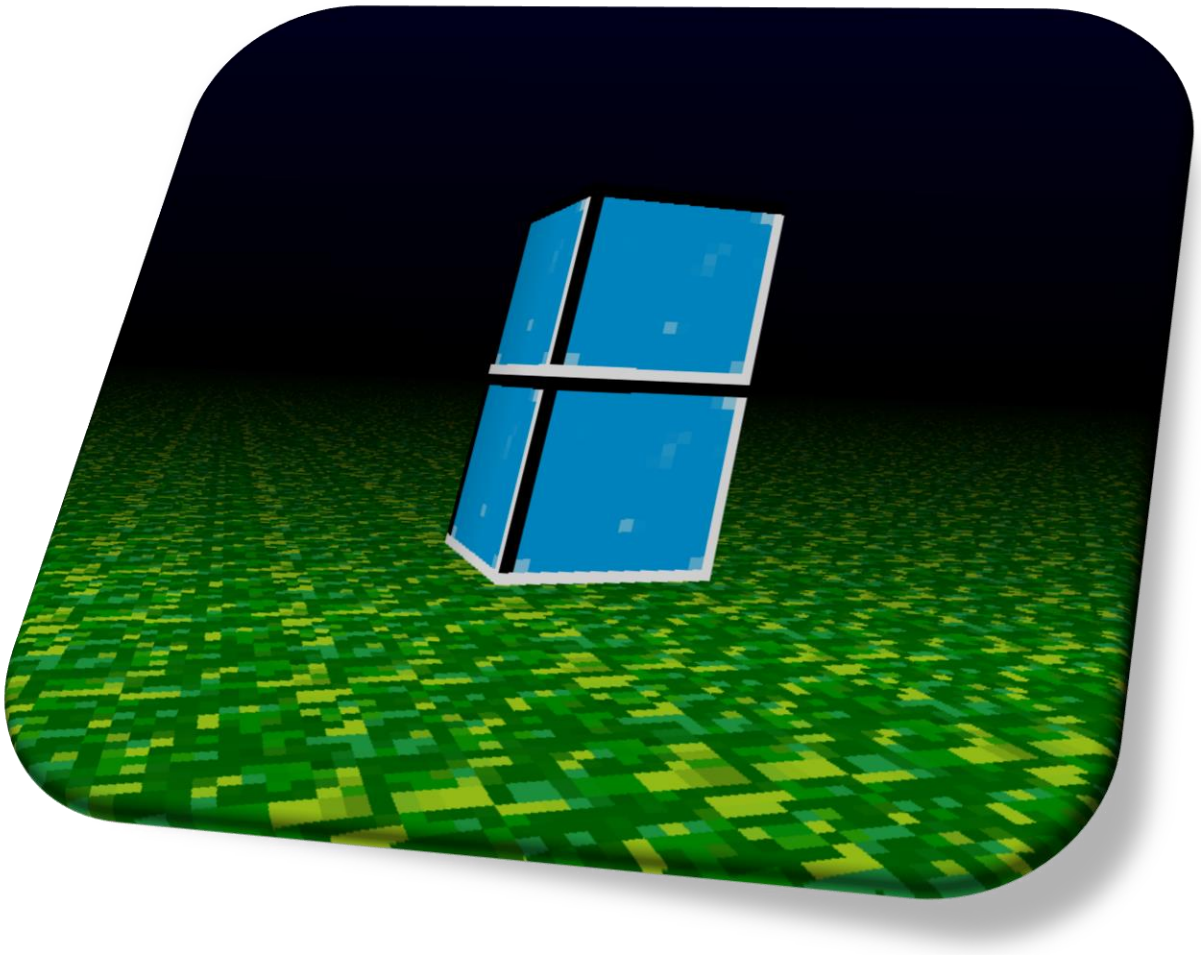
Problemformulering – hovedspørgsmål:
- Hvordan kan ray-tracing og AI upscaling hjælpe moderne spil og 3D renders til at give højere opløsning, med mere komplekse 3D figurer, uden uacceptable lave FPS?

Problemformulering – Underspørgsmål:

- Redegør for hvordan 3D render virker herunder kort om begrebet "raytracing".
- Redegør for vektor, matricer samt lineær algebra og transformation af matricer i relation til 3D rendering.
- Undersøg hvordan det er muligt at programmere en 3D render engine og muligvis optimere denne. Brug forskellige hardwareplatforme til at analysere hvorledes programmets ydeevne afhænger af disse. Vurder hvilke typer af hardwareplatforme der er bedst til at lave 3D render og hvorfor.
- Diskuter med udgangspunkt i dine resultater, hvordan brugen af grafikkort og udviklingen af dette, har gået hånd i hånd med udviklingen af 3D grafik. Kom gerne med et bud på hvordan udviklingen vil være i fremtiden og hvilke udfordringer der kan skimtes.

Udleverede bilag	
Opgaven udleveres	25. november 2022 kl. 15:00
Opgaven skal afleveres	9. december 2022 kl. 15:00

3D Render Projekt



Studie Område Projekt i Matematik og Programmering

Alexander Hillerup Tuff
3d2

9 December 2022

TEC H.C Ørsted Gymnasium Lyngby

Indholdsfortegnelse

Resume

En 3D render bruger verticies til udregning af position i 3D. Den fremstiller også et billede ved at bestemme en RGB-værdi til hver enkelt pixel, som bliver vist frem på skærmen flere gange i sekundet. En 3D render kræver transformations matricer, som for eksempel en rotationsmatrix som benytter trigonometriske metoder til at bestemme rotation i et koordinatsystem. Det kan ses i det 3D render program som jeg har skrevet, i for eksempel renderWall hvor væggen skal rykkes når der er bevægelse i spillet. Mit 3D render program har en benchmark funktion som tager gennemsnits FPS efter det har kørt igennem en bane, programmet er blevet kørt på forskellige hardware platforme hvorefter dataen er blevet analyseret og sat op som forskellige modeller. Data bliver perspektiveret til grafikkortets opfindelse og en forklaring kommer på tale hvorfor det skete og hvordan 3D grafik og grafikkort har udviklet sig gennem tiden. Til sidst kommer der et gæt på at fremtidige grafikkort kommer til at bruge udvikling af de nye teknologer og måske endda helt nye teknologer.

Indledning

Jeg har fået problemformuleringen:

Hvordan kan ray-tracing og AI upscaling hjælpe moderne spil og 3D renders til at give højere opløsning, med mere komplekse 3D figurer, uden uacceptable lave FPS?

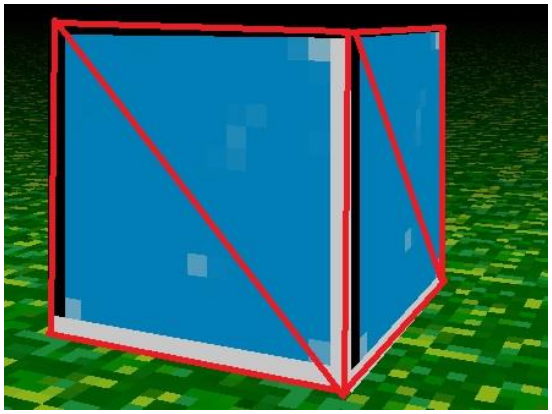
Jeg har i opgaven tænkt mig at redegøre for hvordan en 3D render virker herunder forskellige metoder inden for 3D fremvisning på 2D, rasterisation og ray-tracing, udover det vil jeg redegøre kort om vektorer hvorefter jeg vil redegøre for matricer og transformation af matricer samt forklare et bevis for rotations matrix i 2 dimensioner. Jeg har derefter tænkt mig at skrive en 3D render i java og forklare mit program ved hjælp af et flowchart. Hvorefter jeg vil foretage en undersøgelse af forskellige hardware platforme samle dataene herfra og analysere ved hjælp af regressions analyser, Jeg vil gentage programmet på samme hardware med andre indstillinger hvis nødvendigt og igen analysere dataene fra dette. Den analyse jeg foretager af dataene, vil jeg bruge som empiri til forklaring af grafikkorts udvikling, og vigtigheden i dette. Udover det vil jeg også komme med et gæt på fremtidige grafikkort ud fra hvad jeg har set i min data, men også ud fra undersøgelse af internettet. Jeg ender opgaven ud med at konkludere på problemformuleringen.

Problemformulering delt op

Som nævnt var min problemformulering: Hvordan kan ray-tracing og AI upscaling hjælpe moderne spil og 3D renders til at give højere opløsning, med mere komplekse 3D figurer, uden uacceptable lave FPS? - Jeg har delt den op i nogle mindre bider for bedre at forklare emnet.

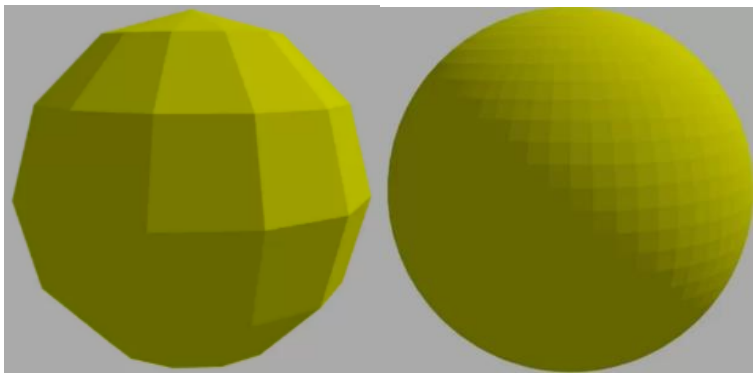
Hvordan virker en 3D render?

En 3D render et program som udregner 3D grafik og fremviser det på en 2D skærm. Det kan for eksempel være computerspil eller animationer.
Alle 3D renders starter et ”sted”, det sted hvor alt har default værdier, ofte i midten af det 3D rum man udregner, altså i (0,0,0) i et 3 dimensionalt kartesisk koordinatsystem. Det kaldes for ”frame zero”, og ofte i spil og andre 3D renders vises det ikke, men bliver bare regnet ud. Det vigtigste til at fremvise 3D objekter er en vertex eller verticies i flertal, her bruger man engelske begreber da de ikke eksistere på dansk. En vertex er et hjørne i en trekant, som bruges til at beskrive former. Hvis man har en hel flad væg i sin 3D render bruges der 4 vertecies, men hvis man derimod har en kube bruges der 8 verticies, som sammen er 12 trekanter. Alle 3D programmer bruger verticies til at udregne objekters positioner, både i programmet, men også i forhold til transformationer og kameraets position.[3.1]



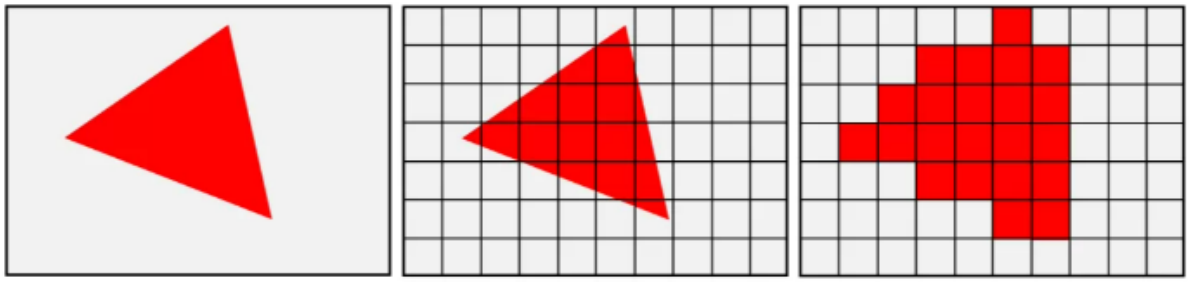
figur 1: et eksempel på 6 verticies (og 4 trekanter) fra mit program

En kube er en meget simpel figur, sammenlignet med en kugle. Du kan måske forestille dig at det er svært at lave en kugle med trekanter, men det er så her hvor man skal vælge det rigtige antal verticies, for at få en flot kugle, men også for ikke at gøre programme for langsomt kig evt. figur 2. på figur 2 kan der tydeligt ses en forskel, både på grund af lyset på figuren, men også fordi antallet af verticies er meget højere i den ene.[3.1] Hvis man så vil have figurer der skal ligne virkeligheden rigtig meget, så kan det være svært at køre programmet, med en god FPS. dog kan moderne computere fremvise spil med flere millioner af verticies, så for det meste er det ikke antallet af objekter og verticies, som kvæler en 3D renders hastighed.



figur 2: forskellen på antal af vericies vist på kugler.[3.1]

Når man har sin 3D verden så skal den selvfølgelig vises på skærmen, det gøres den ved at udregne hver enkelt pixels farve. Den laver de udregner ud fra hvor mange objekter der er i spillet, om de objekter er på skærmen, hvor langt væk de er (i mit tilfælde) og om der er en tekstur knyttet til det objekt. Det programmet så gør er den regner en RGB-værdi til hver pixel.[3.1] I mit program har jeg lavet lyskilden til kameraet, så når objekter kommer væk fra kameraet, bliver de mørkere, indtil de forsvinder helt. Figur 3 visualiserer farvevalget på pixelsne. Det kaldes rasterisation når man laver 3D verdenen til 2D.



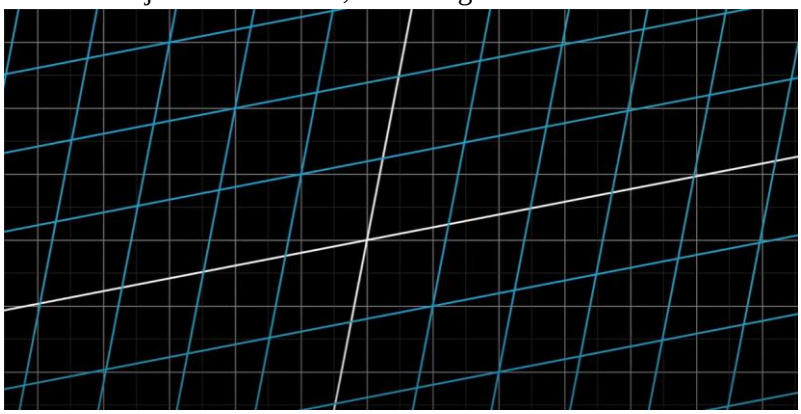
figur 3 viser hvordan pixels bliver fremvist på skærmen[3.1]

Hvis man gerne vil arbejde med 3D skal man normalt ikke skrive alt koden selv. Mange spil bruger biblioteker, som laver alle de her steps, så man ikke selv skal holde styr på millioner af vertices, men bare de objekter man tilføjer. Når det handler om spil bruger man ofte Direct3D, OpenGL eller Vulkan, som alle er biblioteker, som laver alt grafik-render delen, man vælger bare selv hvad der skal være.[3.1] Det smarte ved de biblioteker er at de benytter GPU'en altså Grafikkortet, hvor mit program laver alle de her steps som der er i en 3D render, men gør det med CPUen.

Matematikken bag en 3D render[2]

En 3D render bruger en masse matematik, men det mest interessante og det der står for hvor objekter er og bevæger sig, er transformationsmatrix. En matrix er i mit tilfælde en samling af vektorer. En vektor er en enhed som har en retning og en længde, og skrives på måden $\binom{x}{y}$. Hvor x er dens længde ud af x-aksen og y længden ud af y-aksen.

Før jeg kommer til den transformationsmatrix der sker i en 3D render, skal man først forstå en lineær transformation i 2D. Transformation betyder simpelt set at vi giver et input og for et output, lidt ligesom en funktion, dog er det forskelligt fordi når det er en transformation er der en bevægelse knyttet til, så det man kigger efter, er ikke bare outputtet, men også den bevægelse hen til outputtet. Når det er en lineær transformation, er det også vigtigt at origo ikke ændre sig, og at alle linjer forbliver linjer uden at kurve, evt. se figur 4.[1]



Figur 4: lineær transformation (blå linjer og fremhævede hvide) med originalt koordinatsystem i baggrunden

Måden hvorpå man regner en lineærtransformationen i to dimensioner er ud fra de to basisvektorer. Basisvektorer er enhedsvektorer der følger x-aksen og y-aksen. $\hat{\imath}$ som følger x-aksen og $\hat{\jmath}$ som følger y-aksen

$$\hat{\imath} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\hat{\jmath} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Man kan så efter en transformation aflæse deres position i det originale koordinatsystem og ud fra det se transformationen. Her er T transfomrationen

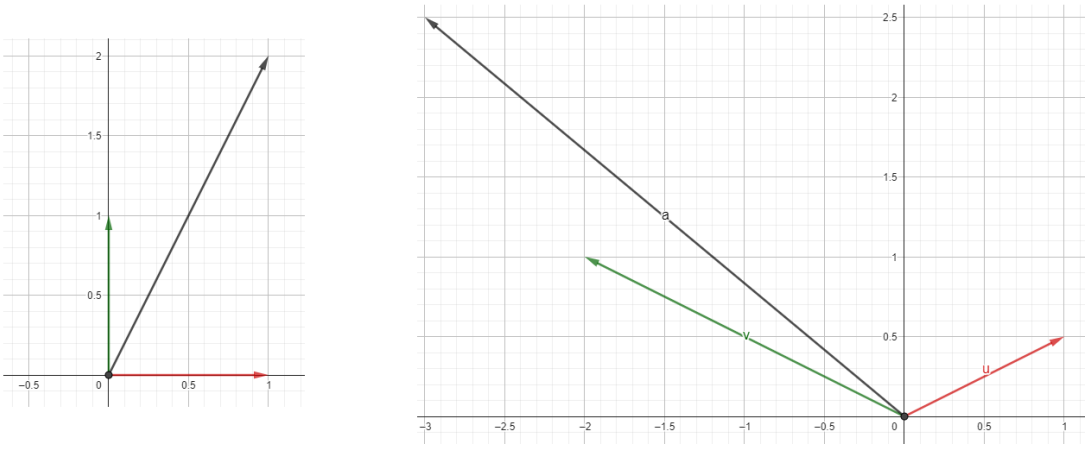
$$T(\hat{\imath})$$

$$T(\hat{\jmath})$$

Man kan ud fra de 2 basisvektorer udregne alle andre vektorer ved at skrive dem op som udtryk af basisvektor.

$$\bar{v} = \begin{pmatrix} v_x \\ v_y \end{pmatrix} = v_x \cdot T(\hat{\imath}) + v_y \cdot T(\hat{\jmath})$$

Grunden til man kan skrive den om som udtryk af basisvektorerne er fordi uden en transformation altså i et normalt koordinatsystem ville man bare skulle gange med 1 i henholdsvis x og y for $\hat{\imath}$ og $\hat{\jmath}$, fordi de 2 basisvektorer er enhedsvektorer.



Figur 6: en transformation hvor $T(\hat{\imath}) = \begin{pmatrix} 1 \\ 0{,}5 \end{pmatrix}$ og $T(\hat{\jmath}) = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$ og en vektor $v = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ som efter transformationen er $\begin{pmatrix} -3 \\ 2{,}5 \end{pmatrix}$ i det originale koordinatsystem.

Man kan skrive $\hat{\imath}$ og $\hat{\jmath}$ ind i en matrix ud fra deres koordinater, og her er matrixen ikke andet end en måde at samle de to vektorer på. Matrixen for transformationen $\begin{bmatrix} \hat{\imath}_x & \hat{\jmath}_x \\ \hat{\imath}_y & \hat{\jmath}_y \end{bmatrix}$. Denne 2x2 matrix beskriver vores todimensionelle lineære transformation, den første kolonne beskriver hvor $\hat{\imath}$ ender og den anden hvor $\hat{\jmath}$.[1]

For eksempel en transformationsmatrix $\begin{bmatrix} 1 & -2 \\ 0{,}5 & 1 \end{bmatrix}$ lavet på en vektor $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$

$$\begin{bmatrix} 1 & -2 \\ 0{,}5 & 1 \end{bmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} = 1 \cdot \begin{bmatrix} 1 \\ 0{,}5 \end{bmatrix} + 2 \cdot \begin{bmatrix} -2 \\ 1 \end{bmatrix} = \begin{pmatrix} 1 \\ 0{,}5 \end{pmatrix} + \begin{pmatrix} -4 \\ 2 \end{pmatrix} = \begin{pmatrix} -3 \\ 2{,}5 \end{pmatrix}$$

Her er den nye vektor så $\begin{pmatrix}-3\\2{,}5\end{pmatrix}$ sat ind i det gamle koordinatsystem.

Nogle af de transformationsmatricer som ofte forekommer, er identitet matrixen altså de originale værdier

$$i_2 = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$

Skalerings matrix, hvor s er den skalering der sker på enten x- eller y-aksen

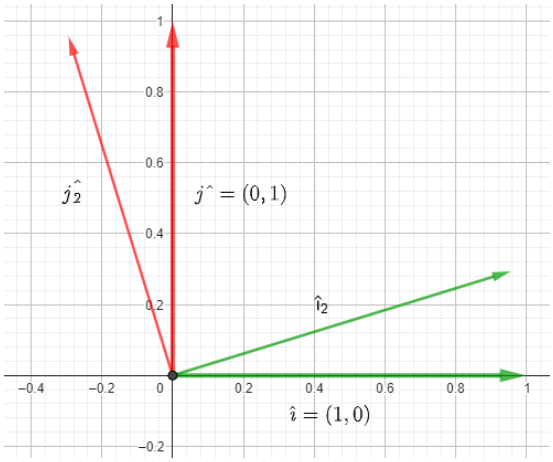
$$s_{\bar{s}} = \begin{bmatrix}s_x & 0\\0 & s_y\end{bmatrix}$$

Rotations matrix, som bruger trigonometri fordi de to basisvektorer følger enhedscirklen, derfor vil î skrives som $\begin{bmatrix}\cos(\theta)\\\sin(\theta)\end{bmatrix}$, hvor θ er vinklen mellem aksen og vektoren. for $\hat{\jmath}$ er det $\begin{bmatrix}-\sin(\theta)\\\cos(\theta)\end{bmatrix}$

Rotationsmatrixen er $R_\theta = \begin{bmatrix}\cos(\theta) & -\sin(\theta)\\\sin(\theta) & \cos(\theta)\end{bmatrix}$

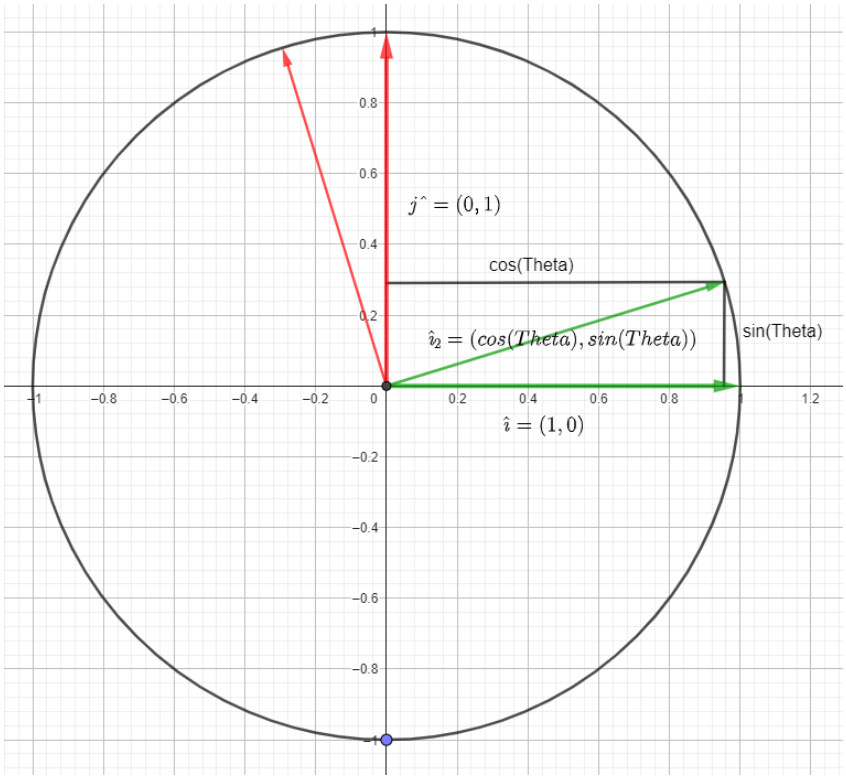
Bevis for rotationsmatrix

Formlen for rotationsmatrixen virker måske lidt tilfældig, men det er den ikke, hvis man tegner de 2 basisvektorer op i et koordinatsystem og derefter deres rotation i samme koordinatsystem så kan man måske gennemskue rotationen.



figur 7: rotationsmatrix i originalt koordinatsystem (î -> $\hat{\imath}_2$ osv.)

Rotationen følger nemlig enhedscirklen altså trigonometri, hvis man så skriver værdierne ind for î er det tydeligt hvorfor rotationsmatrixen ser ud som den gør (se figur 8).



Figur 8: enhedscirkel med basisvektor og deres rotation (î -> $\hat{i}_2$ osv.)

Man kan se at $\hat{i}$ kan beskrives ud fra sinus og cosinus, da den på x-aksen følger cosinus til vinklen θ (Theta) og på y-aksen følger sinus. Det gør at $\hat{i} = \begin{pmatrix}\cos(\theta)\\ \sin(\theta)\end{pmatrix}$. Det kan også tydeligt ses at det samme sker for $\hat{j}$, dog er det omvendte akser og sinus går i negativ på x-aksen derfor kan den beskrives som $\hat{j} = \begin{pmatrix}-\sin(\theta)\\ \cos(\theta)\end{pmatrix}$

De to udtryk for vektorerne kan samles i en matrix $R_\theta = \begin{bmatrix}\cos(\theta) & -\sin(\theta)\\ \sin(\theta) & \cos(\theta)\end{bmatrix}$ som er rotationsmatrixen.

translationsmatrix

Translations matrix hvor man flytter koordinatsystem, hvilket gør den ikke er lineær, men fordi man bare tilføjer en vektor som beskriver hvor det nye nulpunkt er, så er det stadigvæk en lineærtransformation. Den vektor hedder $\hat{t}$ og den er lig $\begin{pmatrix}t_x\\ t_y\end{pmatrix}$. Man har så matrixen der beskriver transformationen og en vektor som beskriver translationen. Man kan så sætte det sammen til en enkelt matrix

$T_{\hat{t}} = \begin{bmatrix}1 & 0 & t_x\\ 0 & 1 & t_y\\ 0 & 0 & 1\end{bmatrix}$ den her matrix får en ekstra række.

Det vil sige når man regner med en translation skal vektoren også augmenteres altså få en række mere. Med værdi 1 eksempelvis $\begin{pmatrix}x\\ y\\ 1\end{pmatrix}$.

Hvis man så laver to transformationer, så skal man gange de 2 matricer sammen.

Matrix multiplikation er faktisk meget simpelt, fordi man læser det bare som en matrix ganget med vektorer, for eksempel hvis vi har en transformeret matrix $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0{,}5 \\ 0 & 1 \end{bmatrix}$ så skrives den nye matrix op som $1 \cdot \begin{pmatrix} 0 \\ -1 \end{pmatrix} + 0 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$ hvor $\begin{pmatrix} 0 \\ -1 \end{pmatrix}$ er den første kolonne i produktet af de 2 matricer. Den næste $0{,}5 \cdot \begin{pmatrix} 0 \\ -1 \end{pmatrix} + 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ -0{,}5 \end{pmatrix}$ som er den anden kolonne

Derfor er $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0{,}5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -0{,}5 \end{bmatrix}$

Et krav til multiplikation af matricer er at der skal være lige mange kolonner i den venstre matrix som rækker i den højre matrix. Der er også altid lige mange rækker i den første matrix som i den resulterende matrix, og der er lige mange kolonner i den resulterende som i den anden matrix. Rækkefølgen man ganger matricerne I er også vigtig. Den transformation som bliver lavet først, er den man ganger på, altså den til højre.

3 dimensional transformations matrix

Den her lineære transformation som er nævnt før, er kun i 2 dimensioner. Hvis man skal lave det til 3D er det heldigvis meget det samme.
transformationen bliver til en 3x3 matrix som ser således ud

$$\begin{bmatrix} \hat{\imath}_x & \hat{\jmath}_x & \hat{k}_x \\ \hat{\imath}_y & \hat{\jmath}_y & \hat{k}_y \\ \hat{\imath}_z & \hat{\jmath}_z & \hat{k}_z \end{bmatrix}$$

Hvor $\hat{k}$ er basisvektor på z-aksen[1]

Hvis man vil lave en translation, ser den således ud:

$$\begin{bmatrix} \hat{\imath}_x & \hat{\jmath}_x & \hat{k}_x & T_x \\ \hat{\imath}_y & \hat{\jmath}_y & \hat{k}_y & T_y \\ \hat{\imath}_z & \hat{\jmath}_z & \hat{k}_z & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Igen kommer der en række mere på i bunden.

Identitetsmatrixen vil se således ud

$$i_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Skaleringsmatrix:

$$s_{\bar{s}} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

Rotationsmatrix kræver en matrix for hver akse, det vil sige der er 3 matricer, men de følger samme princip, som i 2 dimensional rotation.

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

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

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

når man roterer i 2D er det z-aksen man roterer omkring, derfor minder den om den fra 2 dimensioner.[2]

Transformation i 3D render

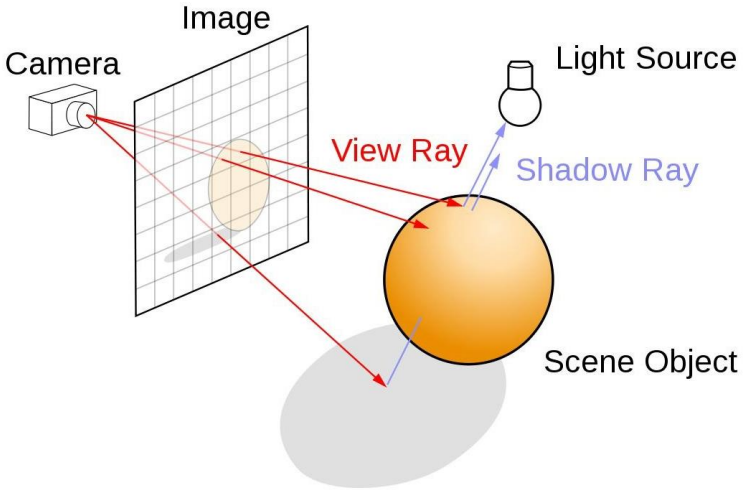
De her transformationer og udregninger bliver lavet af computeren ved hvert frame, for eksempel i mit program så laver den de her udregner for min renderWall funktion, fordi væggen skal blive på sin position, imens kameraet flytter sig. Funktionen virker rimelig uoverskuelig både på grund af de mange udregninger, syntaksen, men også fordi der lægges ekstra værdier til da bevægelse i spillet også ændre kameraets position (i y-aksen) en smule. Jorden og Loftet (funktionen floor) i spillet bliver der også regnet en transformation, her er det primært en rotation om y-aksen fordi man kan dreje kameraet til højre og venstre.

Hvad er acceptable FPS?

For at besvare problemformuleringen skal jeg definere uacceptable FPS. Det er svært at definere da FPS er meget subjektiv, for nogle gør det meget hvis den er for lav og for andre er det lige meget. Generelt i ældre tid var det acceptabel at spille computerspil med en FPS på omkring 30, hvor i moderne tid stræber man oftest efter 60 FPS, i nogle titler vil man endda gå efter en endnu højere FPS, men for det meste vil man gerne have 60. Hvis man kigger på video og film er en acceptabel FPS 24, men da det ikke kræver bruger input, virker det stadig flydende med de lavere FPS.[14] Derfor har jeg endt med at definere en acceptabel FPS, som 60. Det vil sige at alt under 60 FPS er uacceptabel FPS, medmindre det kun er i få sekunder og i meget intense situationer i 3D renderen.

Ray-tracing og AI upscaling

Ray-tracing er en metode, man bruger i stedet for rasterisation, forskellen er bare at man som navnet siger følger de forskellige lysstråler. Det vil sige man laver en stråle fra kameraet igennem hver pixel hen mod lyskilden, og man følger så den stråle og tjekker om den rammer nogle trekanter. Ray-tracing giver derfor et meget mere realistisk billede (pga. lys og refleksioner), men det tager også meget mere computerkraft og tid, at gøre.[4]



Figur 5: viser 3 stråler, og deres vej til lys kilden aka ray-tracing[4]

Som det kan ses på figur 5, så går den ene stråle igennem objektet på scenen, hvilket vil gøre at jorden under den vil have noget af den gule farve og fremvise dens form (hvis jorden under er refleksiv). Den vil som det kan ses også vise skyggen[4]

Ray-tracing har ikke rigtig været i brug (i computerspil) fordi det er så langsomt, men Nvidea (Tech virksomhed, der producerer grafikkort) har udviklet nye kerner (tensor-cores) i grafikkort som er specialiseret til at lave ray-tracing, derfor er det blevet populært i moderne spil.

En anden ny teknologi, som Nvidea har udviklet er DLSS (deep learning super sampling), som er en måde grafikkortet kan producere flere billeder (øge FPS), ved forskellige AI.
Den første AI, som blev brugt, genererer pixels. Det vil sige at hvis man for eksemepl kørte sit spil i 1080p (fuld hd), så ville DLSS, kører spillet i 720p og lave de sidste pixels ved hjælp af AI, det vil gøre at computeren skal lave mange færre udregner hvilket giver bedre FPS. Den nyeste generation af DLSS (DLSS 3) kan gøre det endnu bedre, fordi den gør både det første step ved at render spillet i lavere opløsning og skalere op, men også kan fremstille 100% AI genererede billeder, hvilket ca. fordobler FPS, fordi hvert andet billede er lavet af AI, derfor kan computeren klare sig med kun at render hvert andet billede.[5] Dog har det så en konsekvens, hvis man bruger denne ”Frame Generation” så vil man i nogle tilfælde kunne mærke at spillet ikke er ligeså flydende fordi ens bevægelse (med mus eller tastatur) ikke sker med det samme.

Undersøgelse af en 3D render

Jeg har lavet en 3D render, som jeg vil undersøge på forskelligt hardware for at få en ide om hardware krav for en simpel 3D render, og for at understrege vigtigheden af et grafikkort til at lave udregningerne som bruges til fremvisning af 3D grafik. For at teste forskelligt hardware har jeg lavet en benchmark funktion ind i mit program, så når man starter programmet, kører den med det samme. Den gemmer FPS hvert sekund, og den går rundt på en bane jeg har lavet. Programmet stopper efter et minut og skriver gennemsnits FPS og gennemsnittet af 10% laveste FPS.

Min 3D render

Jeg har skrevet min 3D render i Java 19 og har skrevet render koden selv, jeg har brugt nogle standartbiblioteker som er i Java. Programmet er skrevet efter de principper nævnt til en 3D render. Jeg har også lavet et flowchart til at visualisere programmet (bilag 1). Mit program er meget inspireret af en YouTube tutorial (kilde 6). Jeg har lavet et flowchart for mit gameloop og som det kan ses i gameloopet så er der et par if-statements og 2 while loops. De gør forskellige ting på forskellige tidspunkter. Det første while loop tjekker om spillet kører (while running…) hvis det er sandt kører den koden inde i loopet og efter tjekker den igen. Den kode som der bliver kørt, er til at opdatere spillet, render grafik i spillet og opdatere og gemme FPS. det andet whileloop tjekker om delta >= 1, det gør den fordi koden i loppet skal kører 60 gange i sekundet, fordi her opdateres spillet (tick). Det gør at bevægelse ikke afhænger af FPS, men altid er ens. Så kommer et if-statement som igen tjekker om programmet kører, fordi her skal den begynde at render, og opdatere frames hver gang den render. Derefter kommer et sidste if-statement, som tjekker at der er gået et sekund siden den sidst kørte. Koden her skriver og gemmer frames (FPS over det sekund) og sætter så frames lig med nul så den kan tælle op igen. Hvis det første loop så giver false, altså hvis spillet ikke kører så skal den lukke alle threads og programemt terminere.

I min 3D render/3D spil, har jeg som nævnt før lavet en benchmark funktion, som går gennem en bane. Banen har jeg lavet med vægge og kuber af 4 vægge. Benchmarken tager 1 minut og den bruger en robot (java.awt.robot) til at klikke på forskellige taster, så den simulere den samme bevægelse hver gang. Programmet gemmer så alle FPS-værdier i en Arraylist, som den til sidst sorterer og regner

gennemsnittet af de 10% laveste og gennemsnittet af alle FPS. De FPS værdier vil jeg bruge som data og analysere ud fra.

Begrænsninger og Overvejelser

En af de problemer, som er ved computere, er decimaltal. Hvis man bruger floats har man en begrænset mængde decimaltal, fordi floats ikke er præcise. En float har 32 bit hvor den mest til venstre (mest signifikante) styrer fortegn, de næste 8 (fra venstre af) bestemmer eksponenten, de sidste 23 bit er "mantissa" som repræsenterer de stigende negative kræfter af 2. et eksempel fra teksten i kilde 7: "*if we assume that the mantissa is 11100000000000000000000, the value of this mantissa is calculated as follows:* $2^{-1} + 2^{-2} + 2^{-3} = 7/8$.". Det gør som sagt at ikke alle decimaltal kan repræsenteres, dog i mit program bruger jeg "double", som i bund og grund er det samme, dog er den lavet af 64 bit (doblet så mange altså navnet double), hvilket gør den mere præcis, men stadig vil dens værdi afvige. Dog vil det ikke betyde det store fordi der bliver lavet så mange billeder i sekundet at en enkelt pixels afvigelse ikke kan ses.[7]

En ting jeg har gjort for at forbedre ydeevne, var jeg skiftede /255 (division med 255) ud med bitwise operatøren >> (shift right), som flytter bitene så langt til høje som man skriver efter for eksempel *x >> 8;* når man flytter en værdi 8 bit til højre virker det som division med 255, men det kører meget hurtigere og da jeg skrev det ind i koden gik min FPS op med cirka 30%. Denne division med 255 var til udregning af RGB-værdier for hver enkelt pixel.

En anden overvejelse jeg har taget i forhold til optimering af mit program, er den if-statement (Render3D.java, linje 130) i min renderWall funktion, som stopper flere udregninger hvis væggen ikke er på skærmen, dog skal den stadig regne de første værdier, men den stopper derefter. Det vil sige der er steder i programmet med mange vægge, hvor det tydeligt kan ses, det kun er de vægge foran en der bliver udregnet.

Forsøg og resultater

Jeg har undersøgt programmet på 5 forskellige computere, jeg har kørt programmet et par gange på hver for at se om resultaterne afviger eller om de er nogenlunde ens. De forskellige hardwareplatforme er her:

Enhed \ Specifikationer	CPU	GPU	RAM
Computer 1	Ryzen 7 5800x (3,8GHz)	Nvidea RTX 3060 12GB	16GB 3200
Computer 2 (laptop)	intel I5-11300H (3,1GHz)	Nvidea GTX 1650 4GB	8GB 3200
Computer 3	Ryzen 5 3600 (3,6GHz)	Nvidea GTX 1060 2GB	8GB 3200
Computer 4 (laptop)	Intel i3-3227U (1,9GHz)	Intel HD Graphics 4000 (CPU)	4GB ????
Computer 5 (laptop)	Intel i5-2430m (2,4GHz)	Intel HD Graphics 3000	16GB 1600MHz

(computer 5 havde 3 IDE's åbne og mange faner på nettet)

Selvom Grafikkort ikke bliver brugt direkte i min 3D render, så bruger computeren stadig grafikkortet til at fremvise billedet på skærmen. Rammene bliver brugt i programmet til at holde på variable og derfor vil hurtigere ram også hjælpe med en højere gennemsnitlig FPS.

Resultater

Jeg har kørt programmet flere gange på alle maskiner og taget gennemsnittet. Efter hver kørsel varierede FPS'en meget lidt eller overhovedet ikke, hvilket vil sige at programmet giver et godt overblik.

Computer\FPS	Gennemsnits FPS	10 % laveste FPS
Computer 1	170	92
Computer 2	74	29
Computer 3	157	88
Computer 4	22	8
Computer 5	35	13

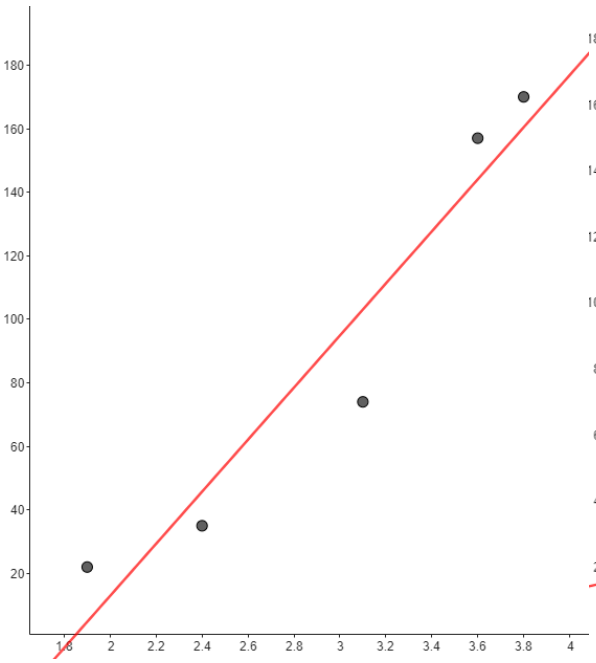
Resultaterne giver meget god mening, ud fra deres hardware, de computere med det nyeste og hurtigste hardware er bedre end dem, som har lidt ældre og langsommere hardware. Det kan også ses at alle 3 bærbare computere ikke kan følge med de 2 stationære. Det skyldes nok at de to stationære computere har en hurtigere CPU (GHz), men også fordi en bærbar ofte ikke kan køre helt optimalt på grund af temperatur, og fordi hardwaren ikke er optimeret, på samme måde. Man kan se at computer 1 og 3 var de eneste der havde en stabil og acceptabel FPS over 60, hvor computer 2 havde en gennemsnits FPS over 60, men i de mest intense dele af Spillet var FPS'en under 60. De to andre bærbare altså computer 4 og 5, kunne slet ikke følge med de andre computere, da de begge indeholder gammelt hardware og har meget langsommere CPU'er sammenlignet med de andre computere.

Konklusion af forsøget

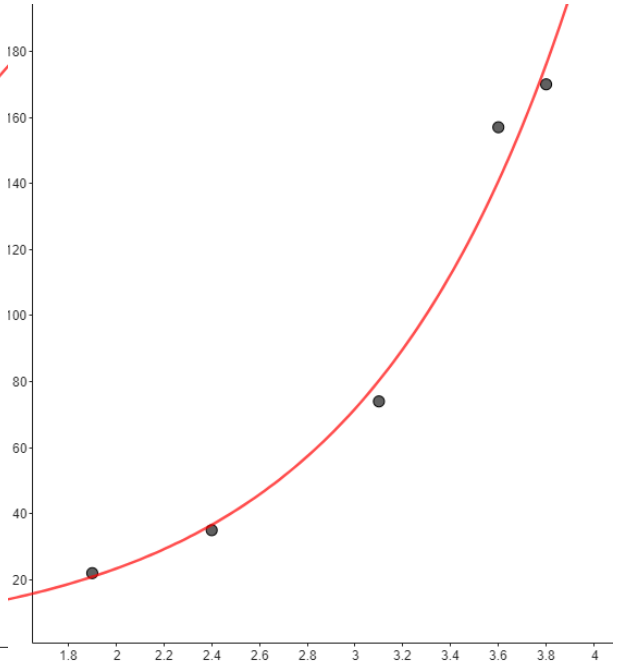
Ud fra resultaterne er det klart at det er svært at opretholde en stabil og acceptabel FPS, ved at bruge CPU til 3D grafik, dog kan det lade sig gøre hvis computerens CPU er hurtig nok og 3D verdenen ikke er for kompliceret, de undersøgte bærbare computere kunne ikke opretholde en acceptabel FPS, hvorimod de stationære computere godt kunne. Det kan også ses at nyere CPU'er med højere clock hastigheder og antal kerner nemmere kunne køre programmet og havde en højere gennemsnitlig FPS.

Diskussion og Perspektivering

Jeg har prøvet at lave en lineærregression af mine data for at få en ide om FPS kan skrives som en lineær funktion af GHz (CPU-GHz). Hvor GHz følger x-aksen og FPS følger y-aksen det kan ses på figur 9 at det godt lidt kunne ligne en lineær funktion, men funktionen giver ikke helt mening da den burde være tættere på (0,0). R^2 er her 0,92 hvilket ikke er helt dårligt.



Figur 9: gennemsnits FPS som funktion af GHz
$f(x) = 81{,}93x - 150$



Figur 10: gennemsnits FPS som funktion af GHz
$f(x) = 2{,}48 \cdot e^{1{,}12x}$

Dog indikere punkterne at det måske kunne være en eksponentiel funktion som beskriver FPS, så jeg testede også det. Den eksponentielle funktion (figur 10) passer rigtig godt med en R^2 på 0,98. det skal dog pointeres at mit antal af punkter er lavt, flere punkter med varierende GHz kunne muligvis give en bedre model, men ud fra mine data og mit program virker den eksponentielle model realistisk.

Flere GHz er svaret

Den eksponentielle model viser at man bare skal lave CPU'er med højere GHz for at få en bedre FPS i 3D renders, dog er der et lille problem med bare at få flere GHz, en af de problemer som der er ved fremstilling af CPU'er er de begrænsninger man støder på. De nye generationer af CPU'er har næsten samme "boost clock" som de tidligere, lige omkring de 5-6 GHz. Den nyeste intel i9-13900k med 5,8GHz og generationen før i9-12900ks med boost på 5,5 GHz, de kan også bruge op til 253 og 243 Watt for at køre på de hastigheder.[8+9] udvikling af CPU'er med højere boost tager tid og koster rigtig meget strøm. Derfor bruger computere også grafikkort, fordi de er speciallavet til de udregninger, som sker til 3D grafik. Hvilket betyder at man til 3D grafik ikke behøver at bruge de nyeste dyreste og mest strømkrævende CPU'er fordi man kan bruge et grafikkort og få meget bedre performance for pengene. Udover det betyder GHz ikke alt fordi GHz ikke direkte kan bruges til fremvisning af FPS, ud fra modellen ser det således ud, men modellen tager ikke højde for antal kerner, antal virtuelle kerner, ram hastighed og mængde, og meget andet.

Hvis man tager et kig på mit program, kan man ud fra den fundene model udregne GHz kravet for at køre med en gennemsnits FPS som er acceptabel, dog hvis man også skal have de lave FPS skal man

bruge en anden model. Man kan bestemme GHz kravet for at kører en acceptabel gennemsnits FPS ved at bestemme x i ligningen $f(x) = 60$

$$2{,}48 \cdot e^{1{,}12x} = 60$$

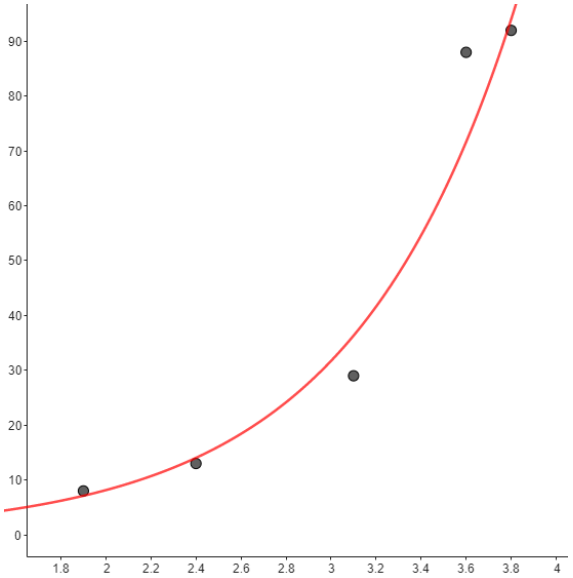
⇳ *ligningen er løst for x ved hjælp af WordMatt.*

$$x = 2{,}84472$$

Det vil sige at hvis man har en CPU med 2,85GHz hastighed burde man kunne køre min 3D render med en acceptabel gennemsnitlig FPS, dog er det ikke helt præcist fordi det ikke kun er GHz der afgør om programmet kører optimalt, men i denne sammenhæng og analyse giver det mening og det er et udmærket gæt for hastigheden. Jeg lavede derfor et forsøg mere for at undersøge om min model passer godt, derfor testede jeg igen med computer 1 (Ryzen 7 5800x, RTX 3060 12GB, 16GB3200), men hvor hastigheden blev låst til 2,85 GHz.

Det Resultat jeg fik var en gennemsnits FPS på 126 og en 10% laveste på 66, hvilket er meget langt fra det jeg havde gættet på, men det skyldes nok at CPU'en er mere moderne og har flere kerner. Det viser i hvert fald at det ikke kun er GHz som bestemmer FPS.

Jeg vil også undersøge om det samme gælder de 10% laveste FPS, og om jeg kan opstille en model for dem og finde en GHz som er optimal for altid at holde spillet kørende ved en acceptabel FPS. jeg lavede derfor en eksponentiel model for de 10% laveste FPS også se figur 11.



figur 12: eksponentiel model for de laveste 10% FPS $f(x) = 0{,}54 \cdot e^{1{,}36x}$

Modellen for de 10% laveste FPS passede ikke lige så godt, som gennemsnits FPS. Modellen har en R^2 på 0,95, som stadig er meget godt. Jeg opsatte samme ligning for at bestemme en den GHz der giver en acceptabel FPS for de 10% laveste,

$$f(x) = 60$$

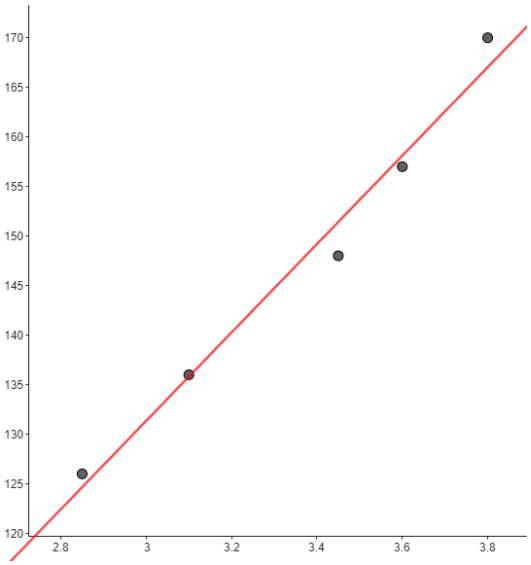
$$0{,}54 \cdot e^{1{,}36x} = 60$$

⇳ *ligningen er løst for x ved hjælp af WordMat.*

$$x = 3{,}463626$$

Hastigheden skal være 3,45 GHz for at have en acceptabel FPS i mine 10% laveste ifølge den fundne model. Jeg testede derfor igen computer 1, men hvor hastigheden var låst til 3,45GHz.
Det jeg kom frem til med en 3,45GHz var en gennemsnits FPS på 148 og en 10% laveste på 77 FPS.

Hvis jeg så laver en ny regressionsanalyse for kun resultater fra stationære computere kan jeg se at den er mere lineær hvilket giver mere mening end den eksponentielle, men stadig giver samme konklusion, fordi nyudvikling af CPU'er stadigvæk ikke kan komme op over på 6GHz. Jeg valgte også at lave et ekstra data punkt på 3,1Ghz hvor resultaterne var 136 gennemsnits FPS og 70 i de 10% laveste.



Figur 13: lineærregression af FPS som funktion af GHz $f(x) = 44{,}6x - 2{,}45$

Med den nye model, som primært kun er fra en hardware platform (computer 1) kan jeg igen undersøge GHz kravet for en acceptabel FPS, jeg udregner ligningen for $f(x) = 60$ igen.

$$44{,}6x - 2{,}45 = 60$$

$$44{,}6x = 62{,}45$$

$$\frac{44{,}6x}{44{,}6} = \frac{62{,}45}{44{,}6}$$

$$x \approx 1{,}4$$

Resultaterne fra computer 1, men på 1,4 GHz var 61 gennemsnits FPS og 30 FPS i 10% laveste, hvilket er meget tæt på det som modellen viser. Det betyder at jeg i teorien kan bestemme et cirka tal for FPS for en hver GHz, men hvis denne model også skal virke på andre hardwareplatforme, kræver det at det tilhørende hardware minder meget om det i computer 1, det vil sige antal kerner, ram hastighed, ram GB osv.

Grafikkort er nødvendigt

Som jeg har nævnt meget nu, så er Grafikkort den som normalt laver 3D grafik, og hvis man kigger på hvornår 3D grafik først blev brugt og hvornår det første grafikkort blev lavet, ligger det meget tæt på hindanen, det giver også mening at det mest krævende grafik nogensinde kom med udvikling af noget der nemmere kunne lave grafik. Det første grafikkort der blev lavet, blev fremstillet af IBM i 1981.[10]
Og et første rigtige 3D grafik som blev brugt, i computerspil blev lavet i slut 1980'erne. Før det "rigtige" 3D grafik lavede man forskelligt falsk 3D grafik som lignede 3D, men ikke helt brugte de

3D metoder som bruges nu[11] det betyder at grafikkort og 3D grafik har gået hånd i hånd, mere avanceret 3D grafik har givet bedre og hurtigere grafikkort og omvendt. De to teknologier har udviklet sig sammen indtil nu, hvor grafikkort ikke længere bare kan øge hastigheden ligesom man ikke bare kan øge hastigheden for CPU'er. Nye moderne grafikkort bruger som sagt nye teknologier til at få bedre FPS. De nye teknologier er nødvendige fordi grafikken i moderne spil bliver mere og mere krævende hurtigere end de nye grafikkort kan blive mere kraftfulde, derfor må nye grafikkort bruge andre metoder til at fremvise en acceptabel FPS. Denne DLLS og Image-Generation, som Nvidea lavede bliver derfor en nødvendighed, da computere bare ikke kan lave udregningerne hurtige nok, for alle elementer i diverse 3D renders.

Hvis man kigger på lidt ældre grafikkort, kan man også se hvorfor Nvidea først lavede RTX-kortene (ray-tracing kort), det skyldes at de ikke kunne fremstille hurtigere grafikkort. Den nyeste serie af grafikkort, som ikke har ray-tracing og DLSS er GTX 16 serien, hvor det hurtigste grafikkort var GTX 1660-Ti, hvis man sammenligner det med det hurtigste fra den forrige GTX-serie et 1080-Ti, så er 1080-Ti'en hurtigere med 79%,[12] det er måske heller ikke en god sammenligning fordi den ene er en 60 og den anden en 80 (navne forskel pga. pris, antal kerner, Ram, OSV). Hvis man i stedet for sammenlignede en GTX 1060 med en GTX 1660 altså samme kort, men fra forskellig serie, giver det måske et bedre overblik over udviklingen. Her kan man tydeligt se at udviklingen er meget langsom med kun 8% forbedring over den ældre serie.[3]
Det er derfor klart at nye grafikkort havde brug for noget andet for at være bedre end de tidligere. Derfor blev RTX-serien lavet, som brugte ray-tracing og senere hen, efter en software opdatering også DLSS, som gjorde at de nye grafikkort var meget hurtigere, fordi de tidligere grafikkort ikke kunne lave de samme udregninger, som de nye. Nvidea startede ray-tracing som standard i computerspil, og som et krav til moderne grafikkort, det betyder også at nye og fremtidige grafikkort kommer til at bygge videre på de trends og teknologier, som Nvidea har startet. Mine forventninger til moderne grafikkort er at de kommer til at have ray-tracing, AI-upscaling, AI-image generation og måske endda andre nye teknologier og algoritmer, som kan hjælpe. eller ændre på måden hvorpå 3D grafik bliver udregnet. Nvidea har allerede på kort til kommet fra DLSS, som et eksperiment til den nye DLSS 3.0. Det udvikler sig meget hurtigt ligesom grafikkort gjorde efter dens opfindelse, og den udvikling kommer nok til at blive langsommere indtil man finder en ny teknologi som kan starte en ny udvikling forfra.

En ulempe ved AI til grafik-render

Som tidligere nævnt er en af ulemperne ved AI frame-generation at der kommer mere latency, som gør at spillet langsommere registrere brugers input, i mange spil gør det ikke en stor forskel, men for Esport og Esports-atleter kan det gøre en kæmpe forskel, selvom det bare er få millisekunder. Jeg har spurgt Oscar Hillerup Tuff en semi-professionel Counter-Strike spiller hvad han syntes om Ray-tracing, AI-upscaling og frame-generation som standard i moderne grafikkort. Han sagde: "Det ødelægger den competetive del i spillet, fordi jeg ikke kan bevæge mig ordentlig og skyde modstanderen, heldigvis er det ikke nødvendigt i Counter-Strike fordi spillet ikke er så krævende, men jeg syntes helt klart det er synd at udviklingen bevæger sig den retning." hvis man derimod kigger på mange andre computerspils entusiaster er det en god ide fordi det kan give bedre billeder og flere FPS, som youtuber 2kliksphillip siger det "best of both worlds".[14]

Konklusion af opgaven

En 3D render er baseret på et 3 dimensionalt kartesisk koordinatsystem og det bruger verticies og trekanter til at fremvise objekter i 3 dimensioner. Bevægelsen af de objekter kræver en matematisk forståelse inden for vektorer matricer og transformation af matricer, man har brug for meget matematik til at fremstille en 3D verden i en computer. Min undersøgelse af en 3D render har givet en bedre forklaring på hvorfor udvikling af grafikkortet var nødvendigt, men også hvordan den moderne udvikling ændrer sig da der er begrænsninger for fremstilling af hurtigere computer hardware. Resultaterne fra mit forsøg viser også at der er forskel på bærbare og stationære computers mulighed for at opretholde en acceptabel FPS på 60 i min 3D render. Den fremtidige udvikling af grafikkort peger også hen imod innovation og udvikling af smarte nye teknologier som kan mindske eller ændre på måden 3D grafik bliver udregnet. Det svarer også på min problemformulering som var: **Hvordan kan ray-tracing og AI upscaling hjælpe moderne spil og 3D renders til at give højere opløsning, med mere komplekse 3D figurer, uden uacceptable lave FPS?**
Dog viser det sig at ray-tracing ikke hjælper med højere FPS, men kan bruges i stedet for rasteresation, men ray-tracing udnytter smarte algoritmer og sammen med Ai-upscaling kan det give en højere FPS, som gør at moderne spil og 3D renders kan blive mere komplekse, selvom det kan give ulemper i brugerens oplevelse, kan det forbedre FPS.

Kilder

1. YouTube, two blue one Brown, Linear transformation and matricies
https://www.youtube.com/watch?v=kYB8IZa5AuE&t

2. YouTube, FloatyMonkey, Matricies and Transformations - Math for Gamedev
https://www.youtube.com/watch?v=HgQzOmnBGCo&t=2s

3. Techspot, Nick Evanson, 3D game Rendering 101:
3.1 https://www.techspot.com/article/1851-3d-game-rendering-explained/
3.2 https://www.techspot.com/article/1857-how-to-3d-rendering-vertex-processing/
3.3 https://www.techspot.com/article/1888-how-to-3d-rendering-rasterization-ray-tracing/

4. Nvidea, Nvidea, Ray Tracing:
https://developer.nvidia.com/discover/ray-tracing

5. Nvidea, Henry C Lin & Andrew Burnes, nvidia dlls3: AI-Powered performance…
https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations/

6. YouTube, The Cherno, 3D game programming in Java:
https://www.youtube.com/playlist?list=PL656DADE0DA25ADBB

7. ScienceDirect, Dogan Ibrahim, Floating Point Numbers
https://www.sciencedirect.com/topics/engineering/floating-point-number

8. Intel, intel, Intel, launches 13th gen intel core processers…
https://www.intel.com/content/www/us/en/newsroom/news/13th-gen-core-launch.html#gs.jzvqem

9. intel, intel, intel, core I9-12900KS Processer

https://ark.intel.com/content/www/us/en/ark/products/225916/intel-core-i912900ks-processor-30m-cache-up-to-5-50-ghz.html

10. Computerhope, Computerhope, computer video card history
https://www.computerhope.com/history/videocard.htm

11. itstillworks, Joshua Laud, History of the first 3D video games

https://itstillworks.com/12314899/history-of-the-first-3d-video-games

12. UserBenchmark, UserBenchmark, GTX1660-Ti vs GTX 1080-Ti
https://gpu.userbenchmark.com/Compare/Nvidia-GTX-1660-Ti-vs-Nvidia-GTX-1080-Ti/4037vs3918

13. UserBenchmark, UserBenchmark, GTX1660 vs GTX 1060

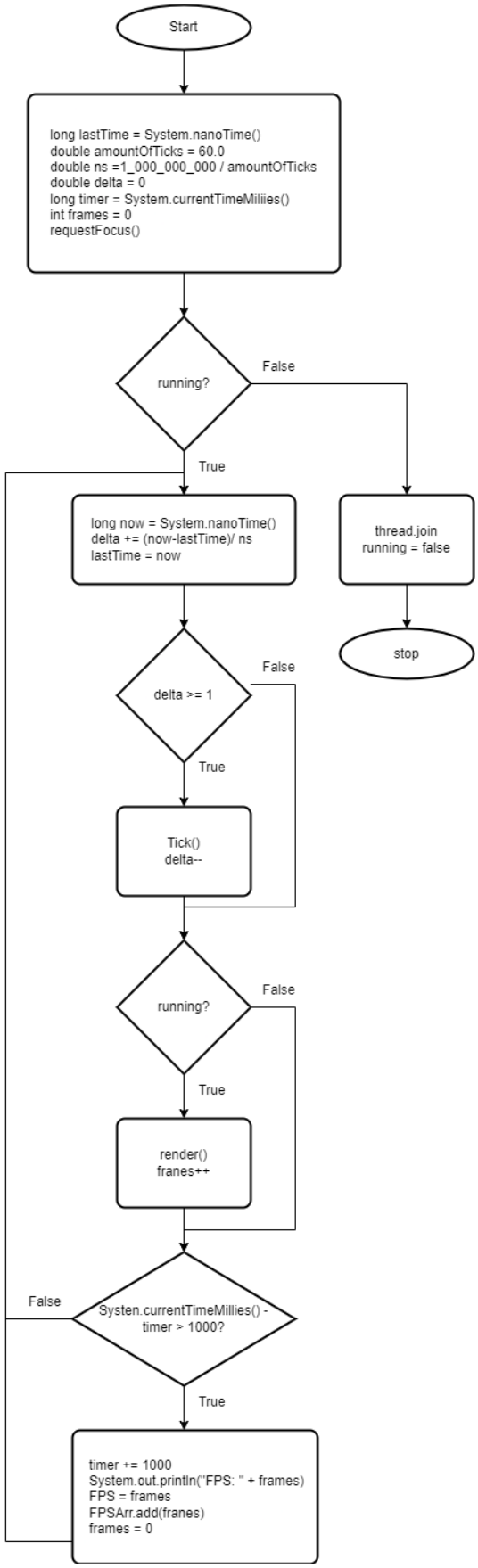
https://gpu.userbenchmark.com/Compare/Nvidia-GTX-1660-vs-Nvidia-GTX-1060-6GB/4038vs3639

14.gpumag, Branko Gapo, what is a good FPS for gaming
https://www.gpumag.com/good-fps-for-gaming/

15. YouTube, 2kliksphillip, DLSS - the past, the present, the future
https://www.youtube.com/watch?v=rmd6X3JE0MY

Bilag

Bilag 1 GameLoop Flowchart:



```
public class Display extends Canvas implements Runnable {
    public static final int WIDTH = 1080, HEIGHT = 720;
    public static final String TITLE = "3D World";
    public static double distance = 0;
    private int FPS=0;
    private Thread thread;
    private boolean running = false;
    private Screen screen;
    private Game game;
    private BufferedImage img;
    private int[] pixels;
    private InputHandler input;
    private int TotalFPS,CountFPS;
    private ArrayList<Integer> FPSArr;

    public static void main(String[] args) {
        Display game = new Display();
        JFrame frame = new JFrame();
        frame.add(game);
        frame.pack();
        frame.setTitle(TITLE);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
        frame.setVisible(true);
        game.start();
    }

    public Display(){
        FPSArr = new ArrayList<>();
        Dimension size = new Dimension(WIDTH,HEIGHT);
        setPreferredSize(size);
        setMaximumSize(size);
        setMinimumSize(size);

        screen = new Screen(WIDTH,HEIGHT);
        game = new Game();
        img = new BufferedImage(WIDTH,HEIGHT,BufferedImage.TYPE_INT_RGB);
        pixels = ((DataBufferInt)img.getRaster().getDataBuffer()).getData();

        input = new InputHandler();
        addKeyListener(input);
        addFocusListener(input);
        addMouseListener(input);
        addMouseMotionListener(input);
```

```
}

private void start(){
    if(running)
        return;
    running=true;
    thread = new Thread(this);
    thread.start();
}

@Override
public void run() {
    long lastTime = System.nanoTime();
    double amountOfTicks = 60.0;
    double ns = 1_000_000_000 / amountOfTicks;
    double delta = 0;
    long timer = System.currentTimeMillis();
    int frames = 0;
    requestFocus();
    while(running){
        long now = System.nanoTime();
        delta += (now - lastTime) / ns;
        lastTime = now;
        while(delta >= 1){
            tick();
            delta--;
        }
        if(running) {
            render();
            frames++;
        }
        if(System.currentTimeMillis() - timer > 1000){
            timer += 1000;
            System.out.println("FPS: " + frames);            //prints frames
            FPS=frames;                                     //shows frames top left in stats
            FPSArr.add(frames);
            frames = 0;
        }
    }
    stop();
}

public synchronized void stop() {
    try{
        thread.join();
```

```
            running = false;
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    private void tick(){
        game.tick(input.key);
        game.benchmark(FPSArr);      //runs benchmark
    }

    private void render(){
        BufferStrategy bs = this.getBufferStrategy();
        if(bs==null){
            createBufferStrategy(3);
            return;
        }

        screen.render(game);

        System.arraycopy(screen.pixels, 0, pixels, 0, WIDTH * HEIGHT);

        Graphics g = bs.getDrawGraphics();

        //draws calculated image
        g.drawImage(img, 0, 0, WIDTH,HEIGHT,null);

        //draws fps counter
        g.drawString("FPS: " + FPS, 5, 15);

        //draws walking speed
        g.drawString("WalkSpeed: "+Controller.walkspeed , 5, 30);

        //draws xyz pos on screen (same as boxes (16 cuz box = 16 units (texture is 16x16)))
        String xyzposition = String.format("%.2f, %.2f, %.2f",
game.control.x/16,game.control.y/16,game.control.z/16);
        g.drawString("x , y , z: "+xyzposition, 5, 45);

        g.dispose();
        bs.show();
    }
}

public class Game {
    public int time;
```

```
private Robot r;
public Controller control;
public Game(){
    try {
        r = new Robot();
    } catch (Exception e){
        System.out.println("robot doesnt work (game.java)");
    }
    control = new Controller();
}

public void tick(boolean[] Key){
    time++;
    boolean forward = Key[KeyEvent.VK_W];
    boolean back = Key[KeyEvent.VK_S];
    boolean left = Key[KeyEvent.VK_A];
    boolean right = Key[KeyEvent.VK_D];
    boolean turnleft = Key[KeyEvent.VK_LEFT];
    boolean turnright = Key[KeyEvent.VK_RIGHT];
    boolean jump = Key[KeyEvent.VK_SPACE];
    boolean sprint = Key[KeyEvent.VK_SHIFT];
    boolean crouch = Key[KeyEvent.VK_CONTROL];

    control.tick(forward, back, left, right, turnleft, turnright, jump, sprint, crouch);
}

public void benchmark(ArrayList<Integer> FPS){
    //enhanced switch case
    switch (time-1) {
        case 0 -> {       //press n' hold w
            r.keyPress(KeyEvent.VK_W);
            }
        case 240*2, 1558*2, 940*2 -> {     //release w turn right
            r.keyRelease(KeyEvent.VK_W);
            r.keyPress(KeyEvent.VK_RIGHT);
            }
        case 320*2, 1640*2, 1015*2 -> {     //release turn right press w
            r.keyRelease(KeyEvent.VK_RIGHT);
            r.keyPress(KeyEvent.VK_W);
            }
        case 590*2 -> {     //turn right
            r.keyPress(KeyEvent.VK_RIGHT);
            }
        case 668*2 -> {     //stop turn
            r.keyRelease(KeyEvent.VK_RIGHT);
```

```
            }
        case 1800*2 ->{      //stop press w, terminate program print avg frames and 10% low after 30
secs (without frame zero)
            r.keyRelease(KeyEvent.VK_W);

            Collections.sort(FPS);

            int TotalFPS=0;
            int TotalLowFPS=0;
            int k=0;

            for (Integer n: FPS) {
               TotalFPS +=n;
            }

            for (int i = 0 ;i<=FPS.size()/10-1;i++){
               TotalLowFPS += FPS.get(i);
               k = i+1;
            }

            int AVGFPS = TotalFPS/FPS.size();
            int LOWFPS = TotalLowFPS/k;

            System.out.println("Average Fps: "+AVGFPS);
            System.out.println("10% low Fps: "+LOWFPS);

            System.exit(0);
         }
      }
   }
}

public class InputHandler implements KeyListener, FocusListener, MouseListener,
MouseMotionListener{
   public boolean[] key = new boolean[68836];
   public static int mouseX;
   public static int mouseY;


   @Override
   public void mouseDragged(MouseEvent e) {
      // TODO Auto-generated method stub

   }
```

```
@Override
public void mouseMoved(MouseEvent e) {
   mouseX = e.getX();
   mouseY = e.getY();

}

@Override
public void mouseClicked(MouseEvent e) {
   // TODO Auto-generated method stub

}

@Override
public void mousePressed(MouseEvent e) {
   // TODO Auto-generated method stub

}

@Override
public void mouseReleased(MouseEvent e) {
   // TODO Auto-generated method stub

}

@Override
public void mouseEntered(MouseEvent e) {
   // TODO Auto-generated method stub

}

@Override
public void mouseExited(MouseEvent e) {
   // TODO Auto-generated method stub

}

@Override
public void focusGained(FocusEvent e) {
   // TODO Auto-generated method stub

}

@Override
public void focusLost(FocusEvent e) {
```

```
        for(int i =0;i<key.length;i++){
            key[i]=false;
        }
    }

    @Override
    public void keyTyped(KeyEvent e) {
        // TODO Auto-generated method stub

    }

    @Override
    public void keyPressed(KeyEvent e) {
        int keyCode = e.getKeyCode();
        if(keyCode>0 && keyCode<key.length){
            key[keyCode] = true;
        }
    }

    @Override
    public void keyReleased(KeyEvent e) {
        int KeyCode = e.getKeyCode();
        if(KeyCode > 0 && KeyCode < key.length){
            key[KeyCode] = false;
        }
    }
}

public class Render {
    public final int width;
    public final int height;
    public final int[] pixels;

public Render(int width, int height){
    this.width = width;
    this.height = height;
    pixels = new int[width * height];

    }

    public void draw(Render render, int xOffset, int yOffset){
        for(int y=0;y<render.height;y++){
            int ypix = y + yOffset;
            if(ypix<0 || ypix>=height){
                continue;
```

```
            }

            for(int x=0;x<render.width;x++){
                int xpix = x + xOffset;
                if(xpix<0 || xpix>=width){
                    continue;
                }

                int alpha = render.pixels[x+y*width];
                if(alpha>0){
                    pixels[xpix+ypix*width] = alpha;
                }
            }
        }
    }
}

public class Controller {
    public double x , y, z , rotation , xa , za , rotationa;
    public static boolean walk = false;
    public static boolean isSprinting=false;
    public static double rotationspeed;
    public static double walkspeed;

    public void tick(boolean forward, Boolean back,Boolean left,Boolean right,Boolean
turnleft,Boolean turnright, boolean jump, boolean sprint, boolean crouch){
        rotationspeed = Display.distance;
        walkspeed = 0.75/2;
        if(!walk)
            walkspeed=0;
        double crouchheight = 0.6;
        double xmove = 0;
        double zmove = 0;
        isSprinting=false;

        if(forward){
            zmove++;
            walk = true;
        }
        if(back){
            zmove--;
            walk = true;
        }
        if(left){
            xmove-=0.5;
```

```
            walk = true;
        }
        if(right){
            xmove+=0.5;
            walk = true;
        }
        if(turnleft){
            rotationspeed=0.01/2;
            rotationa -= rotationspeed;
        }
        if(turnright){
            rotationspeed=0.01/2;
            rotationa += rotationspeed;
        }
        if(crouch){
            y -=  crouchheight;
            walkspeed = 0.5;
            sprint=false;
        }
        if(sprint){
            walkspeed = 1.5;
            isSprinting=true;
        }
        //only bobbing when walking
        if(xmove==0 && zmove==0)
            walk=false;

        xa += (xmove * Math.cos(rotation) + zmove*Math.sin(rotation)) * walkspeed;
        za += (zmove * Math.cos(rotation) - xmove*Math.sin(rotation)) * walkspeed;

        x += xa;
        y *= 0.9;
        z += za;
        xa *= 0.1;
        za *= 0.1;
        rotation += rotationa;
        rotationa *= 0.5;
    }
}

public class Render3D extends Render {

    public double[] zBuffer;
    public double[] zBufferWall;
    private final double renderDistance=20000;
```

```
private final double floorPosition = 16;
private final double ceilingPosition = 100;
private double forward, rightward, upward, cosine, sine, walking;

public Render3D(int width, int height) {
   super(width, height);
   zBuffer = new double[width*height];
   zBufferWall =  new double[width];
}

public void floor(Game game){
   for(int i =0;i <width;i++){
      zBufferWall[i] = 0;
   }

   forward = game.control.z;
   rightward = game.control.x;
   upward = game.control.y;
   walking=0;

   double rotation = game.control.rotation;
   cosine = Math.cos(rotation);
   sine = Math.sin(rotation);

   for(int y=0;y<height;y++){

      double ceiling= (y - height / 2.0) / height;

      double z = (floorPosition+upward) / ceiling;
      if(Controller.walk){
         walking = Math.sin((game.time/3.0)*0.5)/3;
         z = (floorPosition+upward+walking) / ceiling;
      }

      if(Controller.isSprinting && Controller.walk){
         walking = Math.sin((game.time/3.0)*0.5)/1.5;
         z = (floorPosition+upward+walking) / ceiling;
      }

      if(ceiling < 0){
         z = (ceilingPosition-upward) / - ceiling;
         /* if i want roof to move as well
         if(Controller.walk)
            z = (ceilingPosition-upward - walking) / -ceiling;
         */
```

```
        }

        for(int x=0;x<width;x++){
            double depth = (x - width / 2.0) / height;
            depth *= z;
            double xx = depth * cosine + z * sine;
            double yy = z * cosine - depth * sine;
            int xPix = (int) (xx + rightward);
            int yPix = (int) (yy + forward);
            zBuffer[x+y*width]=z;

            //for different roof and floor
            if(y>height/2){
                pixels[x + y * width] = Texture.MCgrass.pixels[(xPix & 15) + (yPix & 15) * 16];
            } else{
                pixels[x + y * width] = Texture.sky.pixels[(xPix & 15) + (yPix & 15) * 16];
            }
        }
    }
}

  public void renderWall(double xLeft, double xRight, double zDistanceLeft, double zDistanceRight,
double yHeight){
    double correction = 1/(floorPosition*2);

    double xcLeft =  ((xLeft) - (rightward*correction))*2;
    double zcLeft = ((zDistanceLeft) - forward*correction)*2;


    double rotLeftSideX = xcLeft * cosine - zcLeft * sine;
    double yCornerTL = ((-yHeight)-((-upward)*correction-(walking*correction)))*2;
    double yCornerBL = ((+0.5 - yHeight)-((-upward)*correction-(walking*correction)))*2;
    double rotLeftSideZ = zcLeft * cosine + xcLeft * sine;

    double xcRight =  ((xRight) - rightward*correction)*2;
    double zcRight = ((zDistanceRight) - forward*correction)*2;

    double rotRightSideX = xcRight * cosine - zcRight * sine;
    double yCornerTR = ((-yHeight)-(-upward*correction-(walking*correction)))*2;
    double yCornerBR = ((+ 0.5 - yHeight) - (-upward*correction-(walking*correction)))*2;
    double rotRightSideZ = zcRight * cosine + xcRight * sine;

    double tex20 = 0;
    double tex30 = 16;
    double clip = 0.05;
```

```
//clip if you walk into the wall it stops render, (removes visual bugs)
if(rotLeftSideZ < clip && rotRightSideZ < clip){
   return;
}

if(rotLeftSideZ < clip){
   double clip0 = ((clip-rotLeftSideZ)/(rotRightSideZ-rotLeftSideZ));
   rotLeftSideZ = rotLeftSideZ + (rotRightSideZ - rotLeftSideZ) * clip0;
   rotLeftSideX = rotLeftSideX + (rotRightSideX - rotLeftSideX) * clip0;
   tex20 = tex20 + (tex30-tex20) * clip0;
}

if(rotRightSideZ < clip) {
   double clip0 = ((clip - rotLeftSideZ) / (rotRightSideZ - rotLeftSideZ));
   rotRightSideZ = rotLeftSideZ + (rotRightSideZ - rotLeftSideZ) * clip0;
   rotRightSideX = rotLeftSideX + (rotRightSideX - rotLeftSideX) * clip0;
   tex30 = tex20 + (tex30 - tex20) * clip0;
}

double xPixelLeft = (rotLeftSideX / rotLeftSideZ * height + width / 2.0);
double xPixelRight = (rotRightSideX / rotRightSideZ * height + width / 2.0);

if(xPixelLeft>=xPixelRight){
   return;
}

int xPixelLeftInt = (int) (xPixelLeft);
int xPixelRightInt = (int) (xPixelRight);

if(xPixelLeftInt < 0){
   xPixelLeftInt = 0;
}
if(xPixelRightInt > width){
   xPixelRightInt = width;
}

double yPixelLeftTop =  (yCornerTL/rotLeftSideZ*height+height/2.0);
double yPixelLeftBot =  (yCornerBL/rotLeftSideZ*height+height/2.0);
double yPixelRightTop =  (yCornerTR/rotRightSideZ*height+height/2.0);
double yPixelRightBot =  (yCornerBR/rotRightSideZ*height+height/2.0);

double tex0 = 1 / rotLeftSideZ;
double tex1 = 1 / rotRightSideZ;
double tex2 = tex20 / rotLeftSideZ;
```

```
    double tex3 = tex30 / rotRightSideZ-tex2;

    for(int x=xPixelLeftInt;x<xPixelRightInt;x++){
        double pixelrotation = (x-xPixelLeft)/(xPixelRight-xPixelLeft);
        double zWall = (tex0+(tex1-tex0)*pixelrotation);

        if(zBufferWall[x] > zWall)
            continue;
        zBufferWall[x] = zWall;

        int xTexture = (int) ((tex2+tex3*pixelrotation)/zWall);

        double yPixelTop = yPixelLeftTop + (yPixelRightTop - yPixelLeftTop) * pixelrotation;
        double yPixelBot = yPixelLeftBot + (yPixelRightBot - yPixelLeftBot) * pixelrotation;

        int yPixelTopInt = (int) yPixelTop;
        int yPixelBotInt = (int) yPixelBot;

        if (yPixelTopInt<0){
            yPixelTopInt=0;
        }
        if (yPixelBotInt>height){
            yPixelBotInt=height;
        }

        for(int y = yPixelTopInt;y<yPixelBotInt;y++){
                double pixelrotationY = (y - yPixelTop) / (yPixelBot - yPixelTop);
                int yTexture = (int) (16*pixelrotationY);
                pixels[x+y*width] = Texture.bricks.pixels[(xTexture & 15)+(yTexture & 15)*16];
                //pixels[x+y*width] =xTexture*80 + yTexture*80;
                zBuffer[x+y*width]= 1/(tex0+(tex1-tex0)*pixelrotation)*16;
        }
    }
}
//fade away using light aka render distance
public void RenderDistanceLimit(){
    for(int i=0;i<width*height;i++){
        int color = pixels[i];

        int brightness = (int) (renderDistance/(zBuffer[i]));

        if(brightness < 0)
            brightness = 0;

        if(brightness>255)
```

```
            brightness=255;

            int r=(color>>16)&0xff;
            int g=(color>>8)&0xff;
            int b=(color)&0xff;

            // >>> 8 or >> 8 idk signed or unsigned??...  it is the same as /255
            r = r*brightness >> 8;
            g = g*brightness >> 8;
            b = b*brightness >> 8;

            pixels[i]=r<<16 | g<<8 | b;
        }
    }
}

public class Screen extends Render {
    private Render3D render;

    public Screen(int width, int height) {
        super(width, height);
        render = new Render3D(width, height);
    }

    public void render(Game game) {
        for(int i = 0 ; i < width * height ; i++){
            pixels[i]=0;
        }

        render.floor(game);

        for (double i = 0; i <= 5; i+=0.5){
            renderbox(1,i,0);
            renderbox(1,i,0.5);
        }
        for (double i = 1; i <= 6; i+=0.5){
            render.renderWall(i,i+0.5,7.5,7.5,0);
            render.renderWall(i,i+0.5,7.5,7.5,0.5);
        }
        for (int i = 0; i <= 6; i++){
            renderbox(10,i,0);
            renderbox(10,i,0.5);
            renderbox(10,i,1);
            renderbox(10,i,1.5);
            renderbox(10,i,2);
```

```
            renderbox(10,i,2.5);
        }
        for (double j = 0;j <= 5;j += 0.5){
            for (double i = -1; i >= -2.5; i-=0.5){
                renderbox(j,i,0);
                renderbox(j,i,0.5);
                renderbox(j,i,1);
            }
        }
        for (double j = 0;j <= 5;j += 0.5){
            for (double i = -4; i >= -5.5; i-=0.5){
                renderbox(j,i,0);
                renderbox(j,i,0.5);
                renderbox(j,i,1);
            }
        }
        renderbox(4,-3,0);
        renderbox(4,-3.5,0);
        renderbox(4,-3,0.5);
        renderbox(4,-3.5,0.5);

        renderbox(2,-3,0);
        renderbox(2,-3.5,0);
        renderbox(2,-3,0.5);
        renderbox(2,-3.5,0.5);

        for (int j = -2; j >= -5;j--){
            for (int i = 0; i <= 6; i++){
                renderbox(j,i,0);
                renderbox(j,i,0.5);
                renderbox(j,i,1);
                renderbox(j,i,1.5);
                renderbox(j,i,2);
                renderbox(j,i,2.5);
            }
        }

        render.RenderDistanceLimit();
        draw(render, 0,0);

    }
    private void renderbox(double x,double z,double y){
        render.renderWall(x+0.5, x, z+0.5, z+0.5, y);
        render.renderWall(x, x, z+0.5, z, y);
        render.renderWall(x, x+0.5, z, z, y);
```

```
            render.renderWall(x+0.5, x+0.5, z, z+0.5, y);
        }
    }

    public class Texture {
        public static Render floor = LoadBitmap("resources/textures/floor.png");
        public static Render grass = LoadBitmap("resources/textures/grass.png");
        public static Render bricks = LoadBitmap("resources/textures/bricks.png");
        public static Render piller = LoadBitmap("resources/textures/pillar.png");
        public static Render MCgrass = LoadBitmap("resources/textures/MCgrass.png");
        public static Render test = LoadBitmap("resources/textures/test.png");
        public static Render test32 = LoadBitmap("resources/textures/test32.png");
        public static Render sky = LoadBitmap("resources/textures/sky.png");

        public static Render LoadBitmap(String fileName){
            try {
                BufferedImage image = ImageIO.read(new FileInputStream(fileName));
                int width = image.getWidth();
                int height = image.getHeight();
                Render result = new Render(width, height);
                image.getRGB(0, 0, width, height, result.pixels, 0, width);
                return result;
            } catch(Exception e){
                System.out.println("textures not loading");
                throw new RuntimeException(e);

            }
        }
    }
```