Practical information

Beskrive alle featurene som er implementer og hvordan dette er gjort.   
Maks 8 sider  
Husk kilder!  
Leveres som pdf  
Leveres på canvas sammen med alle ressurser som trengs for å bygge og kjøre koden (zip)

# Implementasjoner

For å enklere å kunne forklare implementasjonene ønsker jeg først å beskrive de tre matrisene model, view og projection.

MODEL - «MODEL TO WORLD»  
Denne matrisen plasserer objektet verdenen. Med denne matrisen kan man bestemme hvor objektet skal plasseres, samt bestemme rotasjon og skalering.

Objekt 🡪 Objekt på flate

VIEW - «WORLD TO VIEW» // DENNE KAN GJENBRUKES FOR ALLE!  
Denne matrisen bestemmer hvordan verden (med objetet) skal plasseres i forhold til kameraet. Kameraet, som er våre øyne, vil alltid se nedover den negative z-aksen. For at objektene skal syntes, må derfor verden bevege seg.

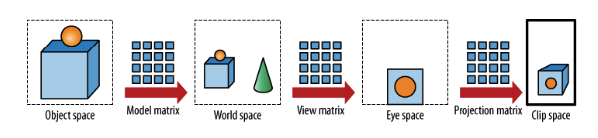
Objekt på flate med kamera

PROJECTION - VIEW TO PROJECTION»

Projectionmatrisen gjør om den objektene som nå er i 3D til 2D, slik at vår hardware kan vise de korrekt. Det er to typer projections. *Perpective* som tar med dypbe i betraktningen slik at objekter som er lenger unna vil se mindre ut, og *Ortohraphics* som ikke gjør det. Med orthographics vil objekter som er lenger unna se like ut og man vil derfor ikke få denne 3D-effekten man får med perspective.

Objekt forann flate i 2D

<https://www.youtube.com/watch?v=-tonZsbHty8>



## First Person Camera

For å kunne bevege seg rundt i verden er det implementert såkalt First Person Camera. Ved å flytte alle objekter i motsatt retning av det man trykker vil det skapes en illusjon av at man beveger seg rundt (<https://learnopengl.com/Getting-started/Camera>).

For bedre oversikt og enklere bruk er ulike Camera-funksjoner lagt i en egen header-fil kalt camera.h. Denne inneholder en Camera-klasse, med tilhørende konstruktør og funksjoner for bruk av instansiert Camera.

I main.cpp vil det opprettes en instanse av Camera. Her settes startposisjonen til 0.0f, 0.0f, 5.0f. Altså skal den settes midt i verdenen, men noe mot deg (langs den positive z-aksen) slik at vi kan se objektene. Ved instasieringen vil det også attributtene for retning, vinkel, hastighet, sensitivitet og zoom settes. Disse er forhåndsdefinerte i klassen.

PRODUSERER VIEW-MATRISE  
Camera-klassen inneholder funksjonen *GetViewMatrix.* Denne vil det ved hjelp av glm∷lookAt bygge en view-matrise basert på attributtene til Camera. Denne brukes for å sette view-matrise til alle objektene (utenom SkyBox).

PROSESSERER MUSEBEVEGELSER  
I main.cpp er det satt opp funksjonen *MouseCallback* som kalles hver gang musen beveger på seg. Denne igjen kalle funksjonen *ProcessMouseMovement* i Camera-klassen og sende med x- g y-posisjonene til mus. Med disse float-verdiene vil attributtene til Camera oppdateres og på den måten vil også view-matrisen man får i *GetViewMatrix*-funksjonen endres.

PROSESSERER TASTATURTRYKK

Main.cpp inneholder også funksjonen *DoMovement* som kalles i while-løkken i main-metoden. Altså vil denne funksjonen kalles «konstant» så lenge programmet kjører. Hver gang den kalles vil den sjekke om pilertaster eller A,S,W,D,Z,X er trykket ned på tastaturet. Om dette er tilfelle vil korrekt retning ut i fra disse sendes til *ProcessKeyBoard-*funksjoneni Camera-klassen. Deltatime, som hindrer hakkete bevegelser ved å bestemme hvor lenge hvert frame (bilde) skal vises på skjermen, sendes også som parameter. I *ProcessKeyBoard* vil posisjonen til Camera endres og dermed vil også view-matrisen fra *GetViewMatrix*-funksjonen endres.

BRUK:

Piltaster eller A,W,S,D = Venstre, frem, bak, høyre

Z = Ned

X = Opp

Musebevegelser = Endrer vinkel

INSPIRASJON TIL IMPLEMENTASJON:

Camera-funksjonen er implementert med hjelp av Sonar Systems sitt videoklipp: <https://www.youtube.com/watch?v=ns9eVfHCYdg>. Denne baserer seg på learnopengl.com sin guide. Mulighet for å også kunne bevege seg opp og ned er implementert selv.

Model: Model to World: Hvordan og hvor boksen skal vises i verden

View: World to View: Hvordan verden skal være I forhold til kameraet

Projection: View to Projection: Hvordan vise det 2-dimenesjonale bildet som 3D

# Skybox/Cubemap

Det første jeg implementerte i mitt prosjekt var Skybox(Også kalt cubemap). Dette er en enkel måte å få et inntrykk av et landskap eller verden rundt kameraet. Skyboxen vil bestå av 6 bilder som hvert for seg representerer en side inne i en kube. Disse 6 2D-bildene vil brettes rundt kameraet og på den måten vil det oppleves som en verden man kan se seg rundt i. <https://learnopengl.com/Advanced-OpenGL/Cubemaps>

## Implementere Skyboxen

Som nevnt over består Cubmap av 6 2D-bilder, altså 6 textures. For hvert bildet må derfor dette lastes inn og spesifiseres med glTexImage2D. I TextureLoading.

For Skyboxen har jeg valgt å opprette egne shadere (skybox.vert og skybox.frag).

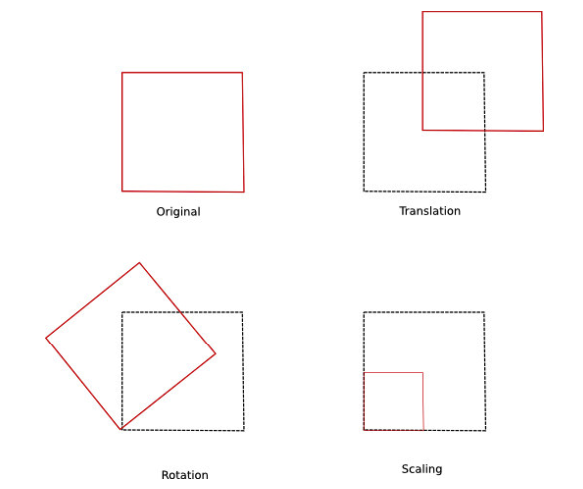
I vertex-shaderen:

* Med layout (location 0) hentes det data med attributten 0 og legger dette i variabelen position.
* Returnerer en vec3 til neste steg i pipeline som inneholder texture-koordinater (sendes til fragment-shaderen)
* Det er deklarert to uniform variabler som brukes til å kommunisere med shadern utenifra. (Er read-only). Vi definerer ikke model-matrise da vi ikke ønsker å påvirke posisjonen til skyboxen.
  + Uniform mat4 view: Mottar informasjon om hvordan verden skal være I forhold til kameraet
  + Uniform mat4 projection: Mottar informasjon om hvordan vise det 2-dimenesjonale bildet som 3D på skjermen

Note that the interesting part of the vertex shader is that we set the incoming position vectors as the outcoming texture coordinates for the fragment shader. The fragment shader then takes these as input to sample a samplerCube:

I fragment-shaderen:

* Mottar TexCoords fra vertex-shaderen
* Sender ut en vec4 med info om color til neste steg i pipeline
* Henter inn data fra koden med info om skybox (SamplerCube)



# INSPIRASJON:

OPPSETT: Simple lighting eksempel!

CUBEMAP, OG CUBE : <https://www.youtube.com/watch?v=_EjsL3B9CRY>

LYS: Blanding av Simple Lighting og <https://learnopengl.com/Lighting/Lighting-maps>

CAMERA: <https://www.youtube.com/watch?v=ns9eVfHCYdg>

# UTFORDRINGER UNDERVEIS:

Mye i begynnelsen før jeg forstod hva kodelinjene gjorde.

**🡪 Deklarterte buffer på nytt. Hadde satt dette globalt:**

// Oppretter Vertex-array og buffer -object for KUBE

GLuint cubeVAO, cubeVBO;

// Oppretter Vertex-array og buffer -object for SKYBOX

GLuint skyboxVAO, skyboxVBO;

Men opprettet nye når de skulle benyttes for å tegne. Brukte ikke de som var satt tidligere og var knyttet til vertecisene.

🡪 Det samme gjaldt cubeTexture

Jeg initialiserte denne på nytt!

**🡪 Hadde satt feil viewmatrise for skybox**

Hadde: glm::mat4 viewSkybox = camera.GetViewMatrix( );

Skulle være glm::mat4 viewSkybox = glm::mat4( glm::mat3( camera.GetViewMatrix( ) ) );

**🡪 Få lys sammen med texture**

**HJELP:** <https://learnopengl.com/Lighting/Lighting-maps>

Måtte gange texture med ambient, diffuse og specular

**vec3** ambient = ambientStrength \* texture( cubeTexture, cubeTextureCoordinates ).rgb;

* Feil med normalverdier

Rekke følge på vertexer feil

Hjalp å kalkulere de for hver isteden for å sette de fast.

# Bibliotek

SOIL

LYS!

<http://www.chai3d.org/download/doc/html/chapter16-lighting.html>