Komme i gang med OpenGL

Litt om

- Qt OpenGL eksempler og klasser.
- Arkitektur, pipeline, render loop, tegne en trekant og pyramide.
- OpenGL og OpenGL ES
- glDrawArrays() og glDrawElements() og initialisering for disse.
- Syntetisk kameramodell, lys, vertexshader og fragmentshader.
- primitive objekter i OpenGL

3DProg21

- Ferdig kompilerbart Qt prosjekt på Git
- Utviklet for 3d-programmering over flere år
- Last ned (og pakk ut)
- Behold navnet! Fjern endelsen **master**
- Bruk dette navnet hele semesteret
- Kompiler og test

Andre OpenGL eksempler

- OpenGL Window Example
- Cube OpenGL ES 2.0 example (samme ikon som kurset i Canvas)
- Velg welcome og let fram disse (filtrer med OpenGL)
- Kompiler og test gjerne disse også
- Finn igjen glDrawArrays() eller glDrawElements() funksjonene

GameSchoolOpenGraphicsLanguage2021

- Egen klasse Shader
- Nært OpenGL API - unngår Qt OpenGL implementeringen i stor grad
- glDrawArrays() heter gl.drawArrays() i JavaScript
- drawing context tilsvarer 3d-canvas i JavaScript
- OpenGL ES tilsvarer WebGL for JavaScript
- Lett å utvikle med JavaScript når en har lært API-et med C++

Eksperimenter

Vi skal gjøre noen små endringer i prosjektet på Git før vi går i gang med å skrive egen kode.

Øving 1 - tegne linjer

Finn igjen

```
  //actual draw call
  glDrawArrays(GL_TRIANGLES, 0, 3);
```

Erstatt i tur og orden med de tre alternativene nedenfor, kompiler på nytt og se hva som skjer:

```
  glDrawArrays(GL_LINES, 0, 3);
  glDrawArrays(GL_LINE_STRIP, 0, 3);
  glDrawArrays(GL_LINE_LOOP, 0, 3);
```

Øving 2 - tegne to trekanter

Finn igjen

```
//Simple global for vertices of a triangle - should belong to a class !
static GLfloat vertices[] =
{
  // Positions         // Colors
  -0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,  // Bottom Left
  0.5f, -0.5f, 0.0f,   1.0f, 0.0f, 0.0f, // Bottom Right
  0.0f,  0.5f, 0.0f,   0.0f, 0.0f, 1.0f   // Top
};
```

Legg til tre nye posisjoner med farger mellom 0 og 1 i vertices. Endre deretter siste parameter i glDrawArrays til 6 og kompiler

```
  //actual draw call
  glDrawArrays(GL_TRIANGLES, 0, 6);
```

Prøv deretter

```
  glDrawArrays(GL_TRIANGLES, 3, 3);
```

Øving 3 - tegne et tetraeder

Legg til ytterligere 6 nye posisjoner i vertices

Øving 6 - vertex shader

Åpne filen plainshader.vert, legg til linjen

```
 color = vec4(0,0,1,0);
```

som vist nedenfor, lagre og kjør programmet på nytt.

```
#version 410 core

layout(location = 0) in vec4 positionIn;   // 1st attribute buffer = vertex positions
layout(location = 1) in vec4 colorIn;   // 2nd attribute buffer = colors
out vec4 color;                         //color sent to rest of pipeline
uniform mat4 matrix;                    //the matrix for the model

void main() {
  color = colorIn;                      //passing on the vertex color
  color = vec4(0,0,1,0);
  gl_Position = matrix * positionIn;      //calculate the position of the model
}
```

Øving 8 - fragment shader

Åpne filen plainshader.frag, legg til linjen
   fragmentColor = vec4(1,0,0,0);
som vist nedenfor, lagre og kjør programmet på nytt.

```
#version 410 core

in vec4 color;          //color received from the pipeline (and vertex shader)
out vec4 fragmentColor;     //color sent to screen

void main() {
  fragmentColor = color;
  fragmentColor = vec4(1,0,0,0);
}
```

Øving 9 - vertex begrepet

Vi ser at det er ganske mye jobb å tegne bare en eller to triangler, og denne hardkodingen kan vi ikke fortsette lenge med. Vi skal begynne å strukturere prosjektet litt. Først, legg til en egen Vertex0 klasse i prosjektet. (Vi skal bruke en mer komplett vertex klasse snart).

```
#ifndef VERTEX_H
#define VERTEX_H
struct Vertex0 {
   float x;
   float y;
   float z;
```

```
    float r;
    float g;
    float b;
};
#endif // VERTEX_H
```

Øving 10 - mVertices, et anonymt std::vector<Vertex0> objekt

Legg til en vektor av vertices i RenderWindow klassen som nedenfor:

```
#include <vector>
#include "vertex.h"

private:
    void init();
    std::vector<Vertex0> mVertices;
```

I konstruktøren RenderWindow::RenderWindow kan du legge til vertex data - en ny trekant med hjørner og farger (burde brukt float konstantdata nedenfor):

```
    //Make the gameloop timer:
    mRenderTimer = new QTimer(this);

    mVertices.push_back(Vertex0{0,0,0,1,0,0});
    mVertices.push_back(Vertex0{0,1,0,0,1,0});
    mVertices.push_back(Vertex0{1,0,0,0,0,1});

}
```

Nå blir det litt mindre behov for hardkoding i RenderWindow::init(). Vi kan erstatte lokal variabel vertices med en objektvariabel (for klassen) mVertices:

```
    //Vertex Buffer Object to hold vertices - VBO
    glGenBuffers( 1, &mVBO );
    glBindBuffer( GL_ARRAY_BUFFER, mVBO );
    //glBufferData( GL_ARRAY_BUFFER, sizeof( vertices ), vertices, GL_STATIC_DRAW );
    glBufferData( GL_ARRAY_BUFFER, mVertices.size()*sizeof( Vertex0 ), mVertices.data(),
GL_STATIC_DRAW );
```

Vi erstatter hardkodet antall trekanter i render

```
    //actual draw call
    glDrawArrays(GL_TRIANGLES, 0, 3);
```

med

```
    //actual draw call
    glDrawArrays(GL_TRIANGLES, 0, mVertices.size());
```

(casting advarsel slik det står).
Legg nå til ytterligere 3 vertices (punkter med farge) i mVertices vektoren, kompiler og kjør på nytt. Hvis du har gjort dette riktig, skal du nå se to trekanter i stedet for bare en. Og vi er ett steg lenger unna hardkoding - med mVertices.size() i koden ovenfor slipper vi å holde styr på antall vertices hver gang vi gjør en endring.

(Eksemplene nedenfor er ikke generelle nok. Vi skal gå gjennom dette på nytt 21/1. Bruk eksempler fra 21/1 videre i kurset.)

XYZ

Eksempel på egen klasse hvor OpenGL kode er flyttet ut fra RenderWindow funksjonene. Kildekoden nedenfor kan legges til prosjektet, og med et xyz objekt i RenderWindow klassen skal dette kompilere og kjøre:

```
#ifndef XYZ_H
#define XYZ_H

#include <QOpenGLFunctions_4_1_Core>
#include <vector>
#include "vertex.h"

class XYZ : protected QOpenGLFunctions_4_1_Core
{
private:
  std::vector<Vertex0> mVertices;
  //QOpenGLContext *mContext;

public:
  XYZ();
  //void setContext(QOpenGLContext *context);
  void init(GLuint mVAO, GLuint mVBO);
  void draw();
};

#endif // XYZ_H

// xyz.cpp
#include "xyz.h"

XYZ::XYZ()
{
  mVertices.push_back(Vertex0{0,0,0,1,0,0});
  mVertices.push_back(Vertex0{1,0,0,1,0,0});
```

```
    mVertices.push_back(Vertex0{0,0,0,0,1,0});
    mVertices.push_back(Vertex0{0,1,0,0,1,0});
    mVertices.push_back(Vertex0{0,0,0,0,0,1});
    mVertices.push_back(Vertex0{0,0,1,0,0,1});
}

void XYZ::init(GLuint mVAO, GLuint mVBO)
{
    initializeOpenGLFunctions();

    //Vertex Buffer Object to hold vertices - VBO
    glBufferData( GL_ARRAY_BUFFER, mVertices.size()*sizeof( Vertex0 ), mVertices.data(),
GL_STATIC_DRAW );

    // 1rst attribute buffer : vertices
    glBindBuffer(GL_ARRAY_BUFFER, mVBO);
    glVertexAttribPointer(0, 3, GL_FLOAT,GL_FALSE,6 * sizeof(GLfloat), (GLvoid*)0);
    glEnableVertexAttribArray(0);

    // 2nd attribute buffer : colors
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE,  6 * sizeof( GLfloat ),  (GLvoid*)(3 *
sizeof(GLfloat)) );
    glEnableVertexAttribArray(1);
}

void XYZ::draw()
{
    //actual draw call
    //setContext(context);
    glDrawArrays(GL_LINES, 0, mVertices.size());
}

I RenderWindow::init()
    //Vertex Buffer Object to hold vertices - VBO
    glGenBuffers( 1, &mVBO );
    glBindBuffer( GL_ARRAY_BUFFER, mVBO );
    //xyz.setContext(mContext);//DN190111
    xyz.init(mVAO, mVBO);

I RenderWindow::render()
    glUniformMatrix4fv( mMatrixUniform, 1, GL_FALSE, mMVPmatrix->constData());

     //actual draw call
     xyz.draw();
```

3d-programmering leksjon 1 - 10/1/22