Matte Oblig 1

Mathias Mohn Mørch

Januar 2023

1 Innledning

I de følgende oppgavene skal vi bruke open GL sin grafikk funksjonalitet sammen med c++ for å løse og visualisere mattematiske grafen. Det skal også vises hvordan man gjør disse grapfene om til triangler som OpenGL kan tegne, samt åssen skrive og lese disse dataene fra fil.

2 Oppgave 1

1

Se vedlegg 1 for datafil. Valgte en $h = \frac{1}{16}$ med en definisjons mendge $\mathbb{D} = (0,0),(1,1)$ Brukte $x^2 \cdot y$ som funksjon
kode snippet som lager grafen

```
float xmin = -0.f, xmax = 1.0f, ymin = -0.f, ymax = 1.0f;
float h = 1.f / (2 << 3);
for (float x = xmin; x < xmax; x += h)
    for (float y = ymin; y < ymax; y += h) {
        float z = myFunc(x, y);
        mVertices.push_back(Vertex{x, y, z, r:x, g:y, b:z});

        z = myFunc(x + h, y);
        mVertices.push_back(Vertex{x + h, y, z, r:x+h, g:y, b:z});

        z = myFunc(x, y + h);
        mVertices.push_back(Vertex{x, y + h, z, r:x, g:y+h, b:z});


        // second triangle
        // ---------------------------------------

        z = myFunc(x, y + h);
        mVertices.push_back(Vertex{x, y + h, z, r:x, g:y+h, b:z});

        z = myFunc(x + h, y);
        mVertices.push_back(Vertex{x + h, y, z, r:x+h, g:y, b:z});

        z = myFunc(x + h, y + h);
        mVertices.push_back(Vertex{x + h, y + h, z, r:x+h, g:y+h, b:z});
    }
```

Funksjoner som leser og skriver data

```
MM::TriangleSurface* tri1 = new MM::TriangleSurface();
tri1->construct();
tri1->toFile("C:/OFFLINE/BaseOpenGL/BaseOpenGL/Assets/Geometry/Franke.txt");


MM::TriangleSurface* tri = new MM::TriangleSurface();
tri->readFile("C:/OFFLINE/BaseOpenGL/BaseOpenGL/Assets/Geometry/Franke.txt");
mObjects.push_back(tri);
```

```
void TriangleSurface::readFile(std::string fileName) {
    std::ifstream inn;
    inn.open(_Filename: fileName.c_str());
    mVertices.clear();
    if (inn.is_open()) {
        int n;
        MM::Vertex vertex;
        inn >> n;
        mVertices.reserve(n);
        for (int i = 0; i < n; ++i) {
            inn >> vertex;
            mVertices.push_back(vertex);
            std::cout << vertex << std::endl;
        }
    }
}

void TriangleSurface::toFile(std::string fileName) {
    std::fstream o;
    o.open(_Filename: fileName.c_str(), _Mode: std::ios::out);
    o << mVertices.size() << std::endl;
    for (int i = 0; i ≤ mVertices.size(); ++i) {
        o << mVertices[i] << std::endl;
    }
    o.close();
```

2

Lage partiellderiverte lambda funksjoner, og lage en til som regner ut normaler basert på disse

```
auto funcX = [](float x, float y) ->float
{
    return 2.f * x * y;
};

auto funcY = [](float x, float y) ->float
{
    return x * x;
};

auto myFuncNormal = [funcX, funcY](float x, float y) ->vec<3, float, packed_highp>
{
    float xx = funcX(x, y);
    float yy = funcY(x, y);

    return normalize( x: glm::cross( x: glm::vec3( _x: 1, _y: 0, xx), y: glm::vec3( _x: 0, _y: 1, yy)));
```

```
float xmin = -0.f, xmax = 1.0f, ymin = -0.f, ymax = 1.0f;
float h = 1.f / (2 << 3);
for (float x = xmin; x < xmax; x += h)
    for (float y = ymin; y < ymax; y += h) {
        float z = myFunc(x, y);
        glm::vec3 n = myFuncNormal(x, y);
        mVertices.push_back(Vertex{x, y, z, n.x, n.y, n.z});

        z = myFunc(x + h, y);
        n = myFuncNormal(x+h,y);
        mVertices.push_back(Vertex{x + h, y, z, n.x, n.y, n.z});

        z = myFunc(x, y + h);
        n = myFuncNormal(x,y+h);
        mVertices.push_back(Vertex{x, y + h, z, n.x, n.y, n.z});


        // second triangle
        // ---------------------------------------

        z = myFunc(x, y + h);
        n = myFuncNormal(x,y+h);
        mVertices.push_back(Vertex{x, y + h, z, n.x, n.y, n.z});

        z = myFunc(x + h, y);
        n = myFuncNormal(x+h,y);
        mVertices.push_back(Vertex{x + h, y, z, n.x, n.y, n.z});

        z = myFunc(x + h, y + h);
        n = myFuncNormal(x+h,y+h);
        mVertices.push_back(Vertex{x + h, y + h, z, n.x, n.y, n.z});
    }
```

3

Her brukte jeg en geometri shader steg for å tegne normal linjene basert på vertex dataen. Geometry Shader (Basert på learnopengl.com)

```
#version 330 core
layout (triangles) in;
layout (line_strip, max_vertices = 6) out;

in VS_OUT {
    vec3 normal;
} gs_in[];

const float MAGNITUDE = 0.4;

uniform mat4 projection;

void GenerateLine(int index)
{
    gl_Position = projection * gl_in[index].gl_Position;
    EmitVertex();
    gl_Position = projection * (gl_in[index].gl_Position +
                                vec4(gs_in[index].normal, 0.0) * MAGNITUDE);
    EmitVertex();
    EndPrimitive();
}

void main()
{
    GenerateLine(0); // first vertex normal
    GenerateLine(1); // second vertex normal
    GenerateLine(2); // third vertex normal
}
```

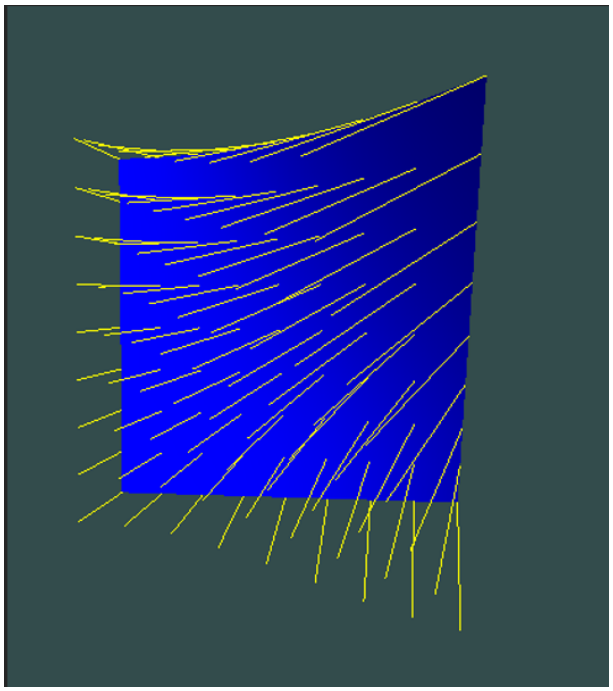
Tegner først objectet med normal shader, derreter normal (som har geometry shader del) shader.

```
leksjon2Shader.use();
leksjon2Shader.setMat4( name: "projection", projection);
leksjon2Shader.setMat4( name: "view", view);

for (VisualObject* object : mObjects) {
    object->draw();
}
/* for (std::vector<VisualObject*>::iterator it = mObjects.beg

normalGeoShader.use();
normalGeoShader.setMat4( name: "model", model);
normalGeoShader.setMat4( name: "view", view);
normalGeoShader.setMat4( name: "projection", projection);
for (auto m_object : mObjects)
{
    m_object->draw();
}
```

Dette ga følgende resultat



Oppengaven sa at man skulle modifisere draw funksjonen, så håper det er greit at jeg isteden bruker draw funksjonen to ganger.

3 Oppgave 2

1

Velger funksjonen $f(x) = cos(4x) * 1(e^x)$, med definisjonsmengde $\mathbb{D} = [0, 5]$

2

Velger 20 intervaller, det gir en h på:

$$h = dx = \frac{b-a}{n}$$
$$h = \frac{5-0}{20} = 0.25$$

3

Velger at fargen til vertexen skal være

$$rgb = f(x) * 0.5 + 0.5$$

Dette er slik at alle punktene på kurven får en farge verdi mellom 0 og 1.

4

Regnet ut funksjonsverdiene og pushet de til mVertices (arver fra Visual Object)

```
Graph2D::Graph2D(std::function<float(float)> f, int n, float from, float to)
{
    mVertices.clear();
    float h = (to - from) / (float)n;
    for (float x = from; x < to; x += h)
    {
        float y = f(x);
        float col = y * 0.5f + 0.5f;
        Vertex v(x, y, z:0, r:col, g:col, b:col);

        mVertices.push_back(v);
    }

    // debug values
    /* for (int i = 0; i < mVertices.size(); ++i) ... */

    mMatrix = glm::mat4x4(s:1.f);
}
```

Brukte derreter toFile funksjon (samme som i oppgave 1)

```
void Graph2D::toFile(std::string fileName) {
    std::fstream o;
    o.open(_Filename:fileName.c_str(), _Mode:std::ios::out);
    o << mVertices.size() << std::endl;
    for (int i = 0; i < mVertices.size(); ++i) {
        o << mVertices[i] << std::endl;
    }
    o.close();
}
```

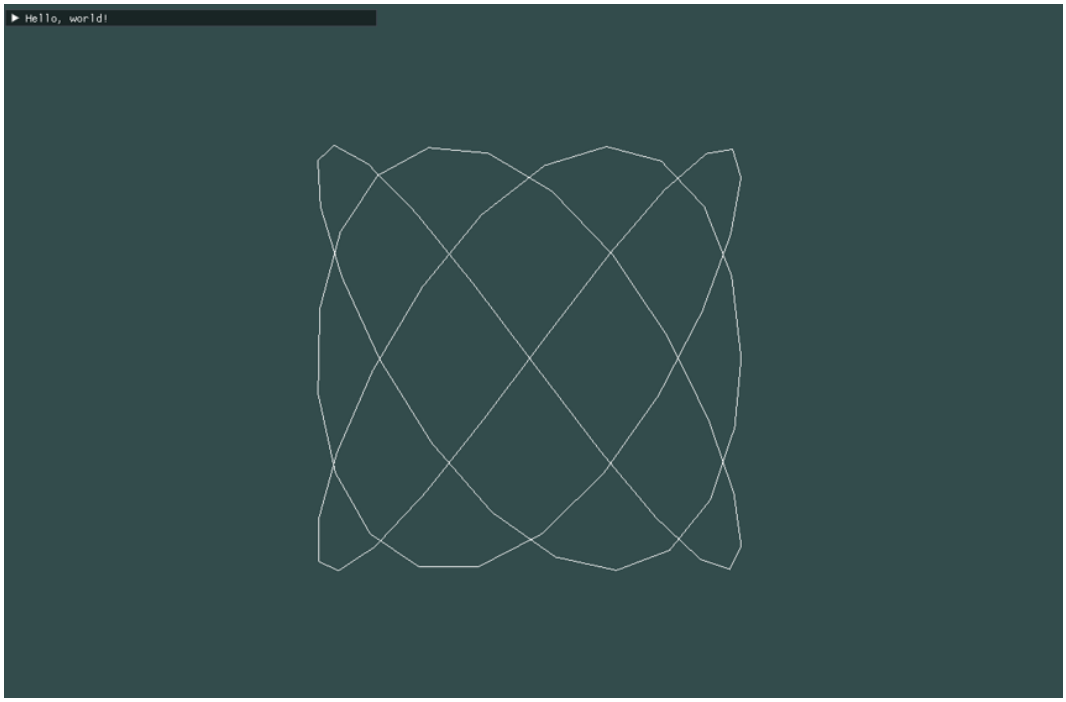
5

```
auto lissa = [](float t)->Vertex
{
    float d = glm::pi<float>()/2.f;
    float a = 3.f;
    float b = 4.f;
    float x = 1.f * sin(_Xx:a * t + d);
    float y = 1.f * sin(_Xx:b*t);

    return MM::Vertex(x,y,z:0.f, r:1.f,g:1.f,b:1.f);
};
std::vector<MM::Vertex> lissaVerts{};
for (float i = 0; i < glm::pi<float>()*2.f; i+= 0.1f)
{
   lissaVerts.push_back(_Val:lissa(i));
}

MM::Graph2D* lissaGraph = new MM::Graph2D(lissaVerts);
// lissaGraph->toFile("LissaGraph.txt");
// lissaGraph->readFile("LissaGraph.txt");
mObjects.push_back(lissaGraph);
```

Valgte parameterene $d = pi/2, a = 3, b = 4, A = 1, B = 1 ogh = 0.1$ Fikk følgende kurve



4 Oppgave 3

1

Funksjonene brukt

```
auto oblig1_3Func = [](float x,float y) ->float
{
    return (1 - x - y);
};


// getting the integral
float lower, upper, step;
lower = 0.f;
upper = 1.f;
step = 0.05f;

float total{};
// reads from the center of each square we evaluate
for (float x = lower + 0.5f*step; x < upper; x+=step) {
    for (float y = lower + 0.5f*step; y < 1.f-x; y+=step) {
        total += oblig1_3Func(x,y) * step * step; // height * length * length
    }
}

std::cout << total << std::endl;
```

Dette ga "total" lik 0.16625. Fasit er $\frac{1}{6} = 0.16666...$

2

Igjentok med halvert avstand h, fikk resultatet 0.166661)

5 Diskusjon

Generelt fant vi at datamaskinen, gitt liten nokk oppdelig h, kan med bra presisjon regne ut intergraler til grafer av to variable. I tilleg klarte vi å tegne flere grafer, både i 3d og 2d i et 3d rom med hjelp fra OpenGL
Jeg har lært åssen man konstruerer en 3d flate med triagnler, hvor inndataen er en graf av to variable, samt mer åssen OpenGL generelt funker med sine VBO VBO osv. Itilleg har intergral oppgaven (Oppgave 3.1) hjulpet med å få en bedre intuitiv forståelse av åssen intergraler av funkksjoner av to variable fungerer.

6 Lenker

Github lenke : https://github.com/Skyress-s/BaseOpenGL/tree/Math3-Compulsory-1