Graz University of Technology
Institute of Computer Graphics und Knowledge Visualization

Version 1.0

Fundamentals of Geometry Processing

Lecturers: Assoc. Prof. U. Augsdörfer, Ass. Prof. O. Diamanti, Alexander Komar
Tutors: Dominik Krobath, Alexander Komar, Julian Rakuschek, Marco Riegler
Website: `http://www.cgv.tugraz.at`
Discord: `https://discord.gg/JasjaU5unh`
Email: fgp2022@cgv.tugraz.at

Please push the completed assignments to the main branch of your git repository before the respective deadline. Late submissions and submissions via email will not be considered. Have fun!

1 General Information

- Individual work - no groups.
- You need to achieve at least 50 % on each assignment to receive a positive grade.
- It is suggested to use a UNIX-based OS for the practicals.
  - You will need at least `CMake 3.5.1` and g++ with `C++14` standard (default since g++7).
- You are not allowed to use any additional libraries (e.g. math.h)/source code, meaning you have to implement all by yourself using the given framework.
- Please only modify files located inside the (`./src/student/`) directory!
  - Feel free to add private methods/members if needed.
  - However, make sure to use the already provided internal data structures and store data in those fields.
  - Do not change the structure/signature of the existing functions, as they are required by the test system!
- Your solution is tested on the assignment testing system after every push.
  - Notice, that the tests on the system are just basic Proof of Work tests.
  - For grading, more complex meshes are used by the FGP team.
- Efficiency plays a huge role in these practicals.
  - Every assignment is tested for efficiency on the test system.
  - You will not receive any points if your solution does not pass this test!
- Every solution is tested for plagiarism on an external server.
  - Please keep personal information out of the files in the `./src/student` directory!
- At the end of the course, each student is interviewed by the FGP team.
  - You will not have to know your code in detail.
  - You will be asked something about the concepts that are necessary to successfully implement the assignments.
- Following the announcement channel on the discord server is mandatory!
  - If you have any questions, feel free to ask them in the discord of the course.

1.1 Grading

This exercise consists of three assignments, each worth 20 Points. You need to score at least 50 % of the points on each assignment to achieve a positive grade.

There is a total of 100 Points for this course:

- 60 points: practicals
- 40 points: exam

In case you fail to score 50 % on an assignment, we provide you with a **second chance** for the respective assignment. You may hand in the failed assignment **one week after the regular deadline**. You may use the second chance on all three assignments.

For every second chance you consume, **we deduct 12.5 points on your overall grade.** For example, if you use the second chance on all three assignments, your overall grade computes as usual (exercise + exam), minus $3 \cdot 12.5 = 37.5$ points.

1.2 Theory

This exercise covers processing of **manifold triangle meshes** using the **half edge data structure**. You will learn how to use the half edge data structure based on:

- Loading a triangle mesh into the data structure.
- Performing queries on the loaded triangle mesh.
- Vertex decimation to reduce the complexity of meshes.
- Smoothing the colours of a triangle mesh.

You will implement all these tasks on your own.

1.2.1 Half Edge data structure

The half edge data structure is also presented in the lecture, but we focus on one specific version in the practicals.

A way to represent geometric objects in 2D or 3D is to use vertices and edges. Vertices are points in space and an edge connects two vertices. A so-called (triangle) face is the surface (or area) enclosed by three edges (and three vertices). In a regular grid every vertex is connected to every other vertex through the three edges. Figure 1 shows vertices (denoted by circles), edges (full, or dotted lines between the vertices), and faces (enclosed by three edges). The full lines highlight one specific face and the straight lined red arrows indicate which vertices belong to this face. The other red arrows point to the neighbouring triangle faces.

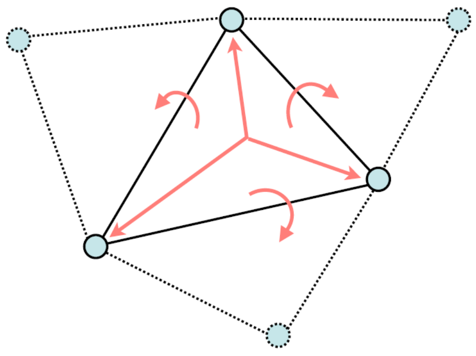

Figure 1: Vertices, edges, and faces as explained in section 1.2.1.

We now introduce half edges. The edges in Figure 1 are not oriented, but half edges are oriented. For every edge in Figure 1, we use two half edges. Two half edges form a pair of half edges, where each half edge points into the opposite direction. Figure 2 shows a triangle mesh using half edges.

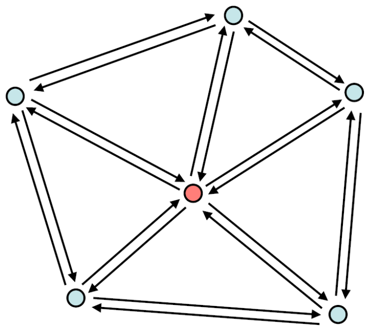

Figure 2: Basic idea of the half edge data structure.

As you can see in Figure 2, the mesh now becomes oriented based on how we choose our very first half edge. As soon as we have decided into which direction the half edges for one face point, we have defined the orientation for all other triangle faces as well.

In more detail, we will use the half edge data structure as shown in Figure 3. The framework provides us with three main classes:

- **Face**: Represents one face. Holds the following information:
  - Pointer to one adjacent half edge (does not matter which one).
- **Vertex**: Represents one vertex. Holds the following information:
  - Pointer to one **outgoing** half edge (does not matter which one).
- **HalfEdge**: Represents one half edge. Holds the following information:
  - Pointer to the opposite half edge (`pair_`). The red, dotted line with arrow in Figure 3.
  - Pointer to the next half edge in this face (`next_`).
  - Pointer to the previous half edge in this face (`prev_`). The red, dashed line with arrow in Figure 3.
  - Pointer to the vertex it origins from (`origin_`). The origin of the red highlighted half edge in Figure 3.
  - Pointer to the face it belongs to (`face_`). The red arrow pointing to a face in Figure 3.

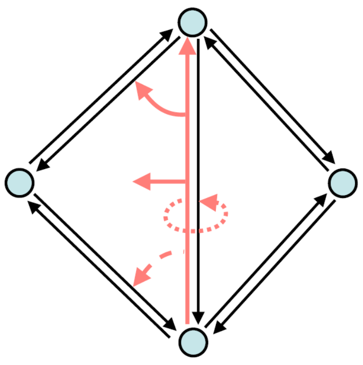

Figure 3: Half edge data structure detailed.

Make sure to use the half edge data structure described in this section!

1.3 Gitlab

The Gitlab repository which contains the framework for the assignments can be found here.
Additionally, the FGP team has provided a repository for each student.
The following steps should be executed to clone the framework repository and push it to your repository:

- Setup your git name: `git config --global user.name "<first name> <last name>"`
- Setup your git email: `git config --global user.email "<email>"`
- Create local folder and move to it: `mkdir <foldername> && cd <foldername>`
- Init git repository inside the folder: `git init`
- Add upstream-remote: `git remote add upstream <git link to framework>`
- Pull from it: `git pull upstream main`
- Add your repository as origin: `git remote add origin <git link to your repository>`
- Push to it: `git push -u origin main`

The necessary links:

- Framework (SSH): `git@student.cgv.tugraz.at:FGP.2022/framework.git`

- Framework (HTTPS): `https://student.cgv.tugraz.at/FGP.2022/framework.git`
- Your Repository (SSH): `git@student.cgv.tugraz.at:<user.name>/fgp2022.git`
- Your Repository (HTTPS): `https://student.cgv.tugraz.at/<user.name>/fgp2022.git`
- `<user.name>` : typically the local part of the TUG student email address.

1.4 *.ply format

The framework contains a couple of example meshes in the ***.ply** file format. The example meshes are all clean manifold meshes. Additionally, it reads such a ***.ply** file and provides you with a list of points in 3-dimensional space and a list of faces that you have to work with. You can visualize the given ***.ply** files with MeshLab, available for download here.

Please note: The ***.ply** files we provide use a dot as the decimal separator. In case you are using the German version of MeshLab, you will run into troubles, because the German version requires the decimal separator to be a comma. As a result, you will not be able to load some meshes (e.g. after performing smoothing). We recommend using the English version.

1.5 Command Line Interface

The provided `./fgp.sh` shell script provides different targets which handles stuff for you:
`./fgp.sh <target>`
Possible targets:

- `build`: builds the executable `./bin/fgp`
- `clean`: removes all build artifacts from the working directory
- `ass{1, 2, 3}`: executes the example call for the respective assignment
- `valgrind_ass{1, 2, 3}`: executes the example call with valgrind (helpful for debugging; valgrind needed)
- `analyze`: performs a static code analysis (optional; cppcheck needed)

You could also use an IDE of your choice and import the project via the cmake project file located in the root directory of your repository.

2 Implementation exercises

2.1 Assignment 1

Deadline: 17th of March 2022, 23:59 CET

2.1.1 Overview

The goal of this first exercise is to familiarise yourself with the data structure used for efficient geometry processing.

You will have to implement the half edge data structure in the C++ programming language and prove that it works by using it to perform some queries you will implement in this Assignment. You have to implement the function **Mesh::load** which fills the half edge data structure with data from ***.ply** files. It is sufficient to implement a data structure that only works with triangle faces. The framework already provides you with the data structure needed.

Reading the ***.ply** files is already done by the framework. The method **Mesh::load** provides you with three parameters:

- A list of points.
- A list of colours (one colour for each point).
- A list of triangles.

These lists are already given in sorted order as they appear in the ***.ply** files. Your task is to fill the data structures with the information from these two lists. The i-th element in the list of colours corresponds to the i-th element in the list of points.

For assigning new IDs for the different elements, use the provided functions:

```
Mesh::getNewFaceID()
Mesh::getNewHalfEdgeID()
Mesh::getNewVertexID()
```

Assign the IDs for the vertices in the same order as they appear in the list of points. The first element in the list of points should get assigned the first vertex ID.

You may test your implementation with the queries you have to implement in this Assignment. Additionally, a simple test is to load a triangle mesh and to save it again. The mesh should not have changed (inspect it with MeshLab). However, this very basic test does not tell if you have implemented the data structure correctly.

Please note: The working implementation of the load function is mandatory as it will be required in the further assignments!

Furthermore, you are asked to implement some queries that operate on the half edge data structure you have loaded in the first part of this assignment. You have to implement the following queries on the half edge data structure:

- Query 1: find all half edges adjacent to face f
  - TODO in **Face::getAdjacentHalfEdges**
- Query 2: find all half edges adjacent to vertex v
  - TODO in **Vertex::getAdjacentHalfEdges**
- Query 3: find all vertices adjacent to face f (basically the corners of the triangle)
  - TODO in **Face::getAdjacentVertices**
- Query 4: find all faces adjacent to vertex v
  - TODO in **Vertex::getAdjacentFaces**
- Query 5: find vertex v at given position (x, y, z)

  - TODO in **Mesh::getVertexAtPosition** and **Vertex::isVertexAtPosition**
  - Think about what a halfedge data structure is good for, and what it is not good for, when implementing this task ;)

- Query 6: find all vertices in the one-ring neighbourhood of given vertex v
  - TODO in **Vertex::getOneRingNeighbourhood**
  - Please implement the version presented in the lecture!

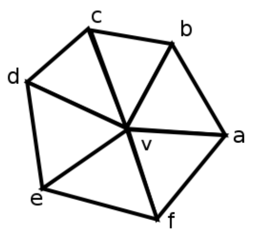

Figure 4: Vertices a, b, c, d, e, and f form the 1-ring-neighbourhood of vertex v.

2.1.2 Point Table

You need to achieve at least 50 % on this assignment to receive a positive grade.

- Load Function: 14 Points
- One point for every query (6 points in total)

You will not receive any points in case you do not pass the efficiency test on the Gitlab server.

In case you fail to achieve at least 50 % of the points at the deadline of Assignment 1, we provide you with a second chance for Assignment 1. You may hand in Assignment 1 one week after the deadline. Keep in mind that the second chance grading mechanism applies when using the second chance (see section 1.1).