3D Scanning & Motion Capture

Exercise 2 – Implicit Surfaces – Marching Cubes

Submission Deadline: see moodle

3D Graphics & Vision
TECHNICAL UNIVERSITY OF DARMSTADT

General Information:

You have to submit your solution via Moodle. We allow **groups of two students**. Please list all names in the submission comments, upload one solution per group. You have **two weeks of working time** (note the deadline date in the header). To get the 0.3 bonus, you have to pass all exercises. The solutions of the exercises are presented the day after the submission deadline respectively. **Feel free to use the Moodle forum to ask questions!**

To inspect your results, you need a 3D file viewer like **MeshLab** (http://www.meshlab.net/). The exercise zip-archive contains a Visual Studio 2017 solution and the required libraries.

Expected submission files: ImplicitSurface.h, MarchingCubes.h, sphere.off, torus.off, hoppe.off, rbf.off

Exercise 2 – Implicit Surfaces – Marching Cubes

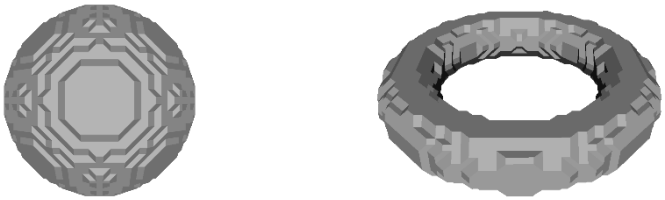
In this exercise we want to generate 3D meshes from implicit surfaces. **In the *main()* function** (main.cpp) **you can switch between the different implicit surface functions.**

Tasks:

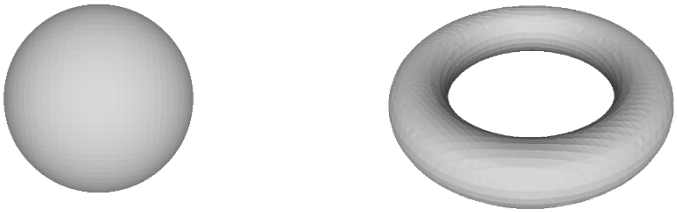
1. Implicit Surface of a Sphere and a Torus

Implement the *Eval()* function of the two classes Sphere and Torus in ImplicitSurface.h.

We provide a default Marching Cubes implementation that is used to extract the iso-surface of the implicit functions. After implementing the functions you should get a mesh (result.off) of the sphere and the torus that looks like that:



Using the Marching Cubes with the linear interpolation schema (Task 2), the sphere and the torus look like:



Exercise 2 – Implicit Surfaces – Marching Cubes

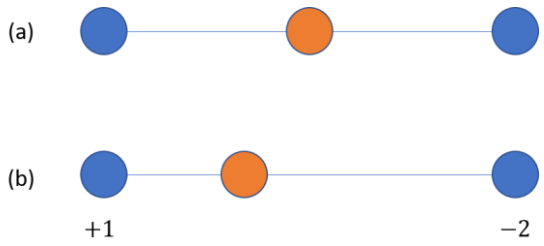
Submission Deadline: see moodle

3D Graphics & Vision
TECHNICAL UNIVERSITY OF DARMSTADT

2. Marching Cubes

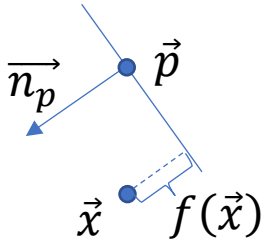
Use the Marching Cubes algorithm to convert a regular 3D grid that stores the distance values (sampled from an implicit surface) to a triangular mesh. Most of the algorithm is already implemented in MarchingCubes.h. The only thing you have to look at is the function *VertexInterp()*. This function is used to determine the surface point, if a zero crossing has been detected on an edge. The parameters of the function are the start point p_1 and end point p_2 of the edge and the corresponding distance values $valp_1$ and $valp_2$. The iso-level of the surface is defined by *isolevel*. In our basic implementation we just return the midpoint of the edge (as depicted in (a)).

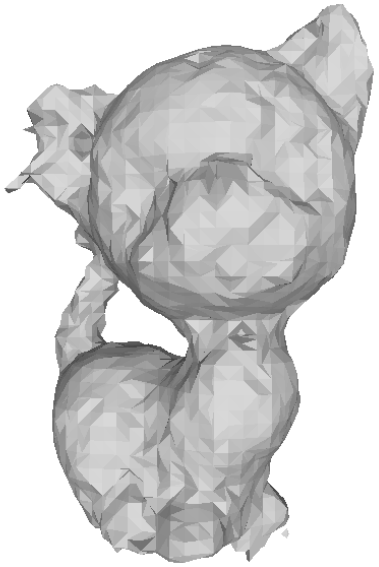
Your task is to compute the linear interpolated point using the provided distances. (b) shows an example with $isolevel = 0$, $valp_1 = +1$ and $valp_2 = -2$.



3. Hoppe

Use the method of Hoppe to generate an implicit surface function. Implement the corresponding *Hoppe::Eval(x)* function. The closest point and its normal is already provided in the source code.





Exercise 2 – Implicit Surfaces – Marching Cubes

Submission Deadline: see moodle

4. Radial Basis Functions

Use Radial Basis Functions to find a smooth implicit surface function.

a. Build the system of linear equations in function *RBF::BuildSystem()*. Use the following formula known from the lecture (adapted to the implementation):

on surface points

off surface points

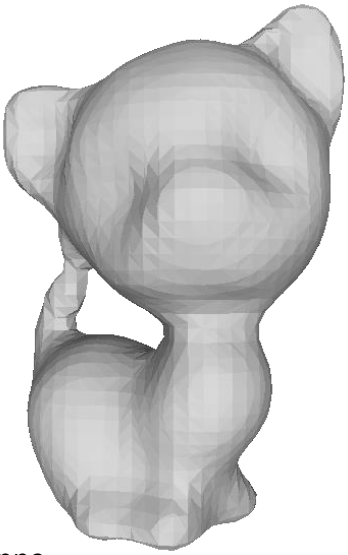
$$\begin{bmatrix} \varphi_{1,1} & \cdots & \varphi_{1,n} & p_{1,x} & p_{1,y} & p_{1,z} & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varphi_{n,1} & \cdots & \varphi_{n,n} & p_{n,x} & p_{n,y} & p_{n,z} & 1 \\ \varphi_{n+1,1} & \cdots & \varphi_{n+1,n} & p_{n+1,x} & p_{n+1,y} & p_{n+1,z} & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varphi_{2\cdot n,1} & \cdots & \varphi_{2\cdot n,n} & p_{2\cdot n,x} & p_{2\cdot n,y} & p_{2\cdot n,z} & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ b_1 \\ b_2 \\ b_3 \\ d \end{bmatrix} = \begin{bmatrix} h_1 \\ \vdots \\ h_{2\cdot n} \end{bmatrix}$$

$$A \cdot \vec{x} = \vec{b}$$

To evaluate the basis functions $\varphi_{i,j}$ with the corresponding points you can use the macro *phi(i,j)* with the indices i,j (note, that the indices start at 0). The sample points p_i can be accessed via *m_funcSamp.m_pos[i]* and the corresponding scalar distance values h_i are stored in *m_funcSamp.m_val[i]*.
The matrix A and the vector $\vec{b}$ is later used to build the normal equation $A^T A\vec{x} = A^T\vec{b}$. The normal equation is solved via a LU decomposition in *RBF::SolveSystem()*.

b. Implement the evaluation function *RBF::Eval(x)* using the solution of the normal equation that is stored in *m_coefficients*. The formula is also known from the lecture:

$$f(\vec{x}) = \sum_i \boldsymbol{\alpha_i} \cdot \|\vec{p_i} - \vec{x}\|^3 + \vec{\mathbf{b}} \cdot \vec{x} + \boldsymbol{d}$$



5. Submit your solution

a. Upload a mesh of a reconstructed sphere, torus, the result of the Hoppe method and the result of the RBF fitting.
b. Submit your ImplicitSurface.h and MarchingCubes.h files.