May 4, 2021

General Information

You have to submit your solution via Moodle. We allow **groups of two students**. Please list all names in the submission comments and upload **one solution per group**. You have **one week of working time** (note the deadline date in the footer). To get the 0.3 bonus, you have to pass at least four exercises, with the fifth one being either completely correct or borderline accepted. The solutions of the exercises are presented the day after the submission deadline respectively. Feel free to use the Moodle forum or schedule a Zoom meeting to ask questions!

To inspect your result, you need a 3D file viewer like **MeshLab** (http://www.meshlab.net/). The exercise zip-archive contains the source templates, a CMake configuration file and the required data. To get the Eigen library, refer to exercise 1.

Expected submission files:

- ImplicitSurface.h
- MarchingCubes.h
- sphere.off
- torus.off
- hoppe.off
- rbf.off
- (optional) team_members.txt

Exercise 2 – Implicit Surfaces – Marching Cubes

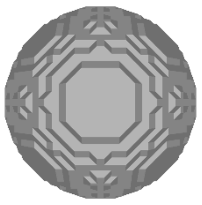
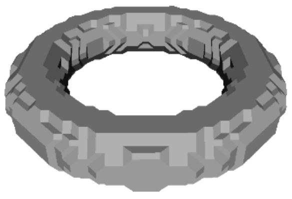
In this exercise we want to generate 3D meshes from implicit surfaces. **In the *main()* function** (main.cpp) **you can switch between the different implicit surface functions.**

Tasks:

1. Implicit Surface of a Sphere and a Torus

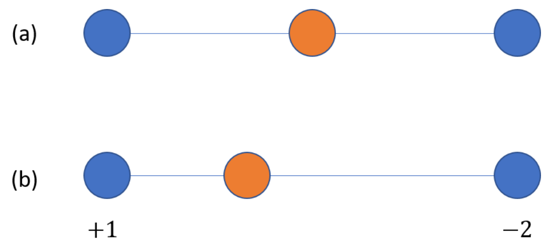
Implement the *Eval()* function of the two classes Sphere and Torus in ImplicitSurface.h.

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 meshes (result.off) of the sphere and the torus that look like the following:

2. Marching Cubes

We 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. A basic version of the algorithm is already implemented in MarchingCubes.h and your task is to improve it. The only thing you have to look at is the function *VertexInterp()*. This function is used to determine the location of 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).

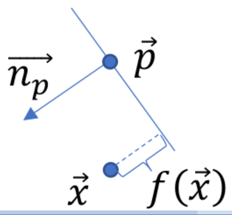


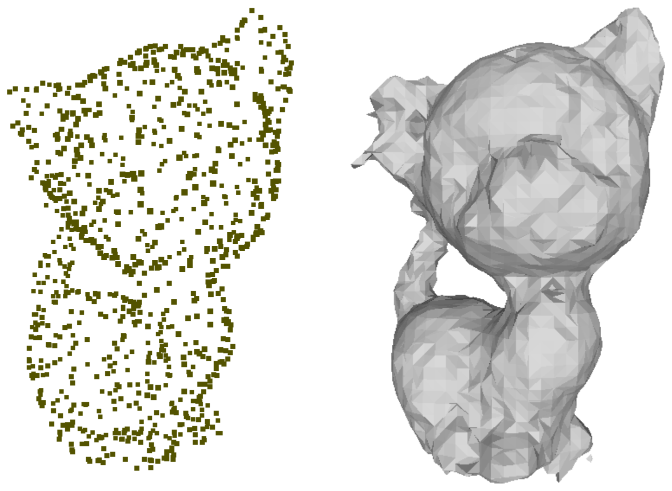
Your task is to compute the surface point via linear interpolation using the provided distances. (b) shows an example with $isolevel = 0, valp_1 =+ 1$ and $valp_2 =- 2$.

Using your improved Marching Cubes algorithm with the linear interpolation schema, the resulting sphere and torus meshes should look like the following:

3. Hoppe

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





May 4, 2021

4. Radial Basis Functions

Use Radial Basis Functions to find a smoother 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):

$$\begin{array}{c} \text{on surface points} \\ \\ \\ \text{off surface points} \end{array} \underbrace{\begin{bmatrix} \varphi_{1,1} & \cdots & \varphi_{1,n} & p_{1,x} & p_{1,y} & p_{1,z} & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varphi_{n,1} & \cdots & \varphi_{n,n} & p_{n,x} & p_{n,y} & p_{n,z} & 1 \\ \varphi_{n+1,1} & \cdots & \varphi_{n+1,n} & p_{n+1,x} & p_{n+1,y} & p_{n+1,z} & 1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \varphi_{2\cdot n,1} & \cdots & \varphi_{2\cdot n,n} & p_{2\cdot n,x} & p_{2\cdot n,y} & p_{2\cdot n,z} & 1 \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ b_1 \\ b_2 \\ b_3 \\ d \end{bmatrix}}_{\vec{c}} = \underbrace{\begin{bmatrix} h_1 \\ \vdots \\ h_{2\cdot n} \end{bmatrix}}_{\vec{b}}$$

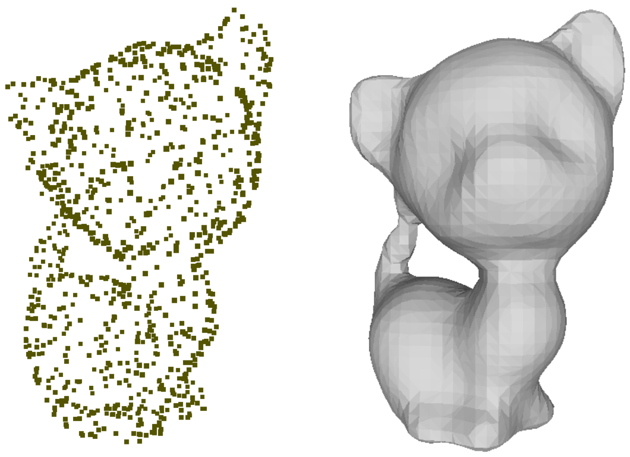
To evaluate the basis functions $\varphi_{i,j}$ with the corresponding points you can use the macro *phi(i,j)* with the indices i,j (note, that the indices start at 0). The sample points p_i can be accessed via *m_funcSamp.m_pos[i]* and the corresponding scalar distance values h_i are stored in *m_funcSamp.m_val[i]*.

The matrix A and the vector $\vec{b}$ are later used to build the normal equation $A^T A\vec{c} = A^T\vec{b}$. 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:

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

$$\varphi_{i,j} = \left\|\vec{p_i} - \vec{p_j}\right\|^3$$



5. Submit Your Solution

a. Submit the following code files: **ImplicitSurface.h**, **MarchingCubes.h**.
b. Upload the following four meshes:
   i. **sphere.off** and **torus.off**, both reconstructed with your improved Marching Cubes.
   ii. **hoppe.off** and **rbf.off**, results of Hoppe and RBF methods respectively.
c. If you worked in a group, upload **team_members.txt**.

>>> Submission Deadline: Wednesday, May 12, 2021, 23:55 <<<