Homework 1
Scattered data interpolation

Advanced computer graphics 2022/23

1 Introduction

Many tasks start with gathering the data by scanning the real world. Often, these data come in a scattered form, meaning that each data point has a location in the domain along with an associated value. These so-called *scattered data* must not be confused with unstructured data, which lack a specific structure altogether. Here are a few examples of scattered data:

- Weather data (temperature, atmospheric pressure, humidity, wind speed and direction etc.) are usually measured at weather stations, which are scattered across the globe, either on land or sea.
- Hydrological data (salinity, groundwater level, evaporation rate etc.) are of critical importance in water supply networks, agriculture and environmental engineering.
- 3D scanned data with applications including 3D modeling, land surveying, archaeology, robotic mapping, industrial design and autonomous vehicles. A common example is LIDAR data, where each data point may hold information about color, intensity, scan angle, scan direction and classification.

Scattered data represent just a small set of values of a spatially varying quantity, which we often want to evaluate not only at the data point locations but also at arbitrary locations between the data points. We therefore need to use some kind of interpolation to be able to get a full view of the data.

1.1 Formal problem statement

Formally, the interpolation problem is rather simple to state. For this assignment, assume that we are working in Euclidean space. We are interested in a quantity $f : \mathcal{D} \to \mathbb{R}$ defined on a domain $\mathcal{D} \subseteq \mathbb{R}^n$. Given N distinct data points $x_1, ..., x_N \in \mathcal{D}$ and their values $y_1, ..., y_N \in \mathbb{R}$, construct the function $\hat{f}$ such that $\hat{f}(x_k) = y_k$ for $k = 1, ..., N$. The function $\hat{f}$ should resemble f as closely as possible.

This last statement is intentionally vague, since any additional desired or required properties (e.g. continuity, differentiability, boundedness) of the function $\hat{f}$ are specific to the task at hand. Additionally, given a limited amount of computational resources, the function $\hat{f}$ should be easy to compute, fast to evaluate, and it should provide local control.

1.2 Basic Shepard's method

One of the simplest methods for scattered data interpolation is *inverse distance weighting* (IDW) with its basic form, the Shepard's method [She68]. Shepard's interpolation is the result of minimizing the following energy function:

$$E(x, y) = \left(\sum_{k=0}^{N} \frac{(y - y_k)^2}{d(x, x_k)^p} \right)^{\frac{1}{p}}. \tag{1}$$

Solving $\frac{\partial E}{\partial y} = 0$ yields the Shepard's interpolant:

$$\hat{f}(x) = \begin{cases} \frac{\sum_{k=1}^{N} w_k(x) y_k}{\sum_{k=1}^{N} w_k(x)}, & \text{if } d(x, x_k) \neq 0 \text{ for all } k, \\ y_k, & \text{if } d(x, x_k) = 0 \text{ for some } k, \end{cases} \tag{2}$$

where

$$w_k(x) = \frac{1}{d(x, x_k)^p} \tag{3}$$

for some distance metric d. The parameter $p > 0$ controls the shape of the interpolant, giving more or less weight to the nearest points.

1.3 Modified Shepard's method

The described basic form of the Shepard's method suffers from a major drawback: the data points act globally, so consequently all of them have to be taken into account when computing a single interpolated value. By limiting the radius of influence of each data point we can achieve local control. Additionally, it is possible to accelerate the search for the nearest neighbors by superimposing a space partitioning structure. This approach is known as Modified Shepard's method [FN80, Ren88].

The weights in this interpolation scheme have to be modified accordingly:

$$w_k(x) = \left(\frac{\max(0, R - d(x, x_k))}{Rd(x, x_k)} \right)^2, \tag{4}$$

where R is the radius of influence. If there are no data points in the radius R, we can output the value of the nearest data point.

2 Assignment

In this assignment you will be given a set of N data points $(x_k, f(x_k))$, from $\mathcal{D} = \mathbb{R}^3$, such that $x_k \in \mathbb{R}^3, f(x_k) \in \mathbb{R}$. Your task is to write a program that will implement **both the basic and the modified Shepard's method** in order to interpolate the data points and output a regular grid (a volume) of reconstructed values that can be visualized in a volume rendering application (e.g. VPT). The program will be given a bounding box $x_{\min}, x_{\max} \in \mathbb{R}^3$ along with the resolution $r \in \mathbb{N}^3$ in order to construct the volume, and the parameters $p, R \in \mathbb{R}$ for the interpolation itself. The distance metric should be Euclidean. To accelerate the search for the nearest neighbors, **you have to arrange the data points in an octree**. Technical details (octree maximum depth and bucket size) are up to you. As a rough guideline, your method should be fast enough to output a $256 \times 256 \times 256$ volume for 10^5 points in around 30 minutes maximum.

Your program must accept the following command-line arguments:

- `--input <path>` (input file path)
- `--output <path>` (output file path)
- `--method [basic|modified]` (basic or modified Shepard's method)
- `--p <float>` (parameter p of the basic Shepard's method)
- `--R <float>` (parameter R of the modified Shepard's method)
- `--min-x <float>` (min x coordinate of the output volume)
- `--min-y <float>` (min y coordinate of the output volume)
- `--min-z <float>` (min z coordinate of the output volume)
- `--max-x <float>` (max x coordinate of the output volume)
- `--max-y <float>` (max y coordinate of the output volume)
- `--max-z <float>` (max z coordinate of the output volume)
- `--res-x <uint>` (x-resolution of the output volume)
- `--res-y <uint>` (y-resolution of the output volume)
- `--res-z <uint>` (z-resolution of the output volume)

2.1 Input format

The input file will be binary. It will begin with a 32-bit unsigned integer (little endian) indicating the number of data points. Then, for each data point, there will be four single precision floats (little endian), describing the location (x, y, z coordinates) of the point and its value.

You can assume the input will always be valid, so do not waste time on validation and error handling.

2.2 Output format

The output will be binary. For each voxel, it will store its interpolated value as a single precision float (little endian). Since we are sampling on a regular grid, we do not have to store the sample locations, but the samples have to be sorted **lexicographically on** $\mathbb{R}$, that is, the x coordinate changes fastest, then the y coordinate, and finally the z coordinate.

2.3 Visualizing the data

This step is optional, but it will help with debugging.

To actually see your interpolated data, you can, for example, output each slice as an image. Hint: the PPM format is extremely easy to write. On the other hand, to visualize the whole volume in 3D, you can use volume rendering, e.g. VPT (`http://lgm.fri.uni-lj.si/ziga/vpt/`). In both cases, the values have to be encoded as an 8-bit unsigned integer instead of a single precision float, so you will have to convert them accordingly.

2.4 Grading

This assignment is worth 10 points:

- 4 points for the basic Shepard's method, and
- 6 points for the modified Shepard's method and octree acceleration.

References

[FN80] Richard Franke and Greg Nielson. Smooth interpolation of large sets of scattered data. *International Journal for Numerical Methods in Engineering*, 15(11):1691–1704, 1980.

[Ren88] Robert J. Renka. Multivariate interpolation of large sets of scattered data. *ACM Trans. Math. Softw.*, 14(2):139–148, June 1988.

[She68] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, page 517–524, New York, NY, USA, 1968. Association for Computing Machinery.