Prof. Dr. Thomas Schultz
Mohammad Khatami (khatami@cs.uni-bonn.de)
Ikram Jumakulyyev (ijumakulyyev@cs.uni-bonn.de)

University of Bonn
B-IT/Computer Science II
June 19, 2020

Summer term 2020

Visual Data Analysis

Assignment Sheet 9

Solution has to be uploaded by June 29, 2020, 8:00 a.m.
to `https://uni-bonn.sciebo.de/s/ktIVodWdEdra9eM` with password `vda.2020`

Please bundle the results (as PDF) and scripts (*.py/*.ipynb files) in a single ZIP file. Submit each solution only once, but include names and email addresses of all team members in the PDF and each script. Name the file `vda-2020-xx-names.zip`, where `xx` is the assignment sheet number, and `names` are your last names.

If you have questions concerning the exercises, please write to our mailing list: vl-scivis@lists.iai.uni-bonn.de.

Exercise 1 (B-Spline Kernels, *9 Points*)

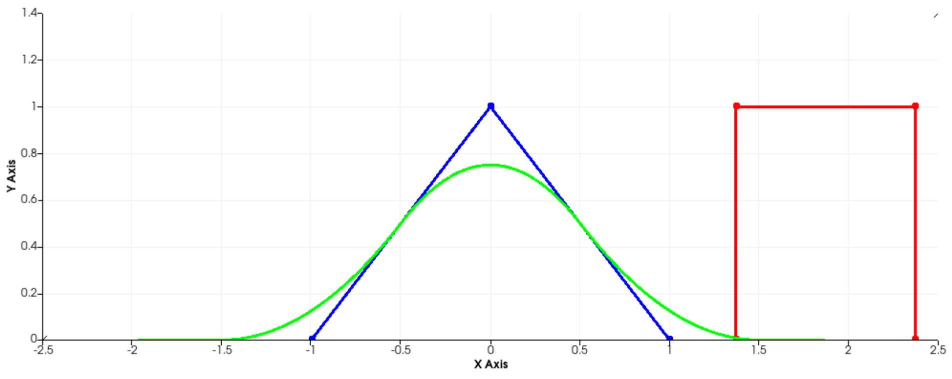

Figure 1: Frame from an animation that illustrates the construction of a quadratic B-spline.

In the lecture, we introduced convolution with B-Spline kernels as one way of reconstructing smooth functions from discrete samples, and we pointed out that B-Splines with higher order lead to smoother results. However, we did not present the formal definition of B-Splines. In this exercise, you will learn how to construct B-Spline kernels of arbitrary order, and you will implement an animation that illustrates their construction. A frame of that animation is shown in Figure 1.

Definition: The zero-order B-Spline kernel β^0 is defined as a unit width, unit height box function, centered around the origin:

$$\beta^0(x) := \begin{cases} 1 & \text{if } x \in [-0.5, 0.5] \\ 0 & \text{else} \end{cases} \tag{1}$$

Order-k B-Spline kernels β^k are defined recursively by repeated convolution with β^0:

$$\beta^k(x) := \beta^{k-1} * \beta^0 \tag{2}$$

Task: On the lecture homepage, we provide the framework `convolution.py`. It creates an animation that illustrates the convolution of two functions, similar to Slide 55 in Chapter 7. It is prepared to illustrate the two convolutions that create β^1 and β^2. The only thing that is missing is the code in `computeAreaBoxBox` and `computeAreaBoxTriangle` that should evaluate the respective convolution integral for a given shift value. Please insert that code and submit the result.

Hint: You can either formally solve the integrals, or use basic equations from geometry, since they only involve computing the areas of rectangles and triangles.

Exercise 2 (Mapping Between Index and World Space, *6 Points*)

Assume a uniform grid with origin at $\mathbf{p} = (1, -1, 2)^T$, axis directions $\mathbf{e} = 1/\sqrt{2}(1, 0, 1)^T$, $\mathbf{f} = (0, 1, 0)^T$, $\mathbf{g} = 1/\sqrt{2}(1, 0, -1)^T$, and axis spacings $dx = 1$, $dy = 2$, and $dz = 1$.

a) Specify the affine transformation matrix from index space coordinates (i, j, k) to world space coordinates (x, y, z). (2P)

b) Specify the corresponding mapping for transferring world space coordinates (x, y, z) back to index space coordinates (i, j, k). (2P)
Hint: You can use a computer (e.g., Python) to do the computation. It is acceptable if you specify rounded results.

c) How does the original transformation matrix from index to world space change if we upsample along the y-axis by a factor of 2 to achieve isotropic voxels? Keep in mind that the original sample points keep their world space positions. (2P)

Exercise 3 (Finite Differences, *7 Points*)

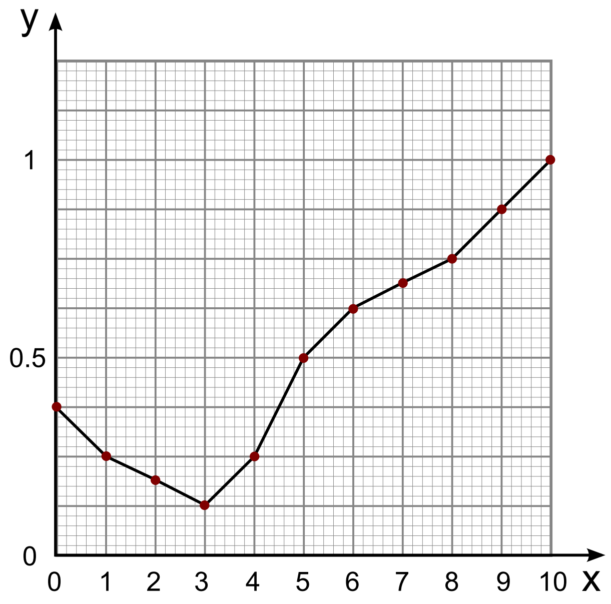

Figure 2: A uniformly sampled and piecewise linearly interpolated function.

a) For the uniformly sampled function shown in Fig. 2, compute the derivative at $x = 3$. Provide the results from using forward differences, central differences, and backward differences. For each answer, also draw the corresponding tangent into the plot. (3P)

b) Use the Taylor expansion to show that the forward finite difference scheme on a uniform grid with stepsize h is first-order accurate (i.e., the approximation error decreases linearly with decreasing stepsize h). (2P)

c) Use the same strategy to show that the central finite difference scheme is second-order accurate (i.e., the approximation error decreases with h^2). (2P)

Exercise 4 (Visualization for Neurosurgery Planning, *10 Points*)

This week's reading, `beyer-neurosurgery-planning-2007.pdf`, should motivate some of the visualization techniques that we will cover during the next few weeks. You should not expect to fully understand all technical details (yet), but this will not be required to answer the following questions. As always, we do not give credit for copy-pasted text.

a) The system that is described in the paper fuses 3D images from several modalities. In the first case study, MR, PET, and fMRI were used. Briefly describe the purpose of each of these three. (3P)

b) The system fuses different modalities mostly by switching to the most informative one for the object at the current spatial location. However, in certain cases, modalities get blended together: DSA, fMRI, and PET can be blended with anatomical MR or CT images. Briefly describe how the blending is done for DSA, and for PET. (2P)

c) The need for a registered CT dataset is mentioned as a drawback of the system. Briefly explain why it is required. (2P)

d) To achieve nicer results, the system implements object boundary smoothing as shown in Fig. 7. However, this reduces the framerate. How does the system ensure fluent interactions despite this? (2P)

e) Why do the authors decide to use direct volume rendering (DVR) instead of surface-based rendering? (1P)

Exercise 5 (Marching Squares Implementation, *18 Points*)

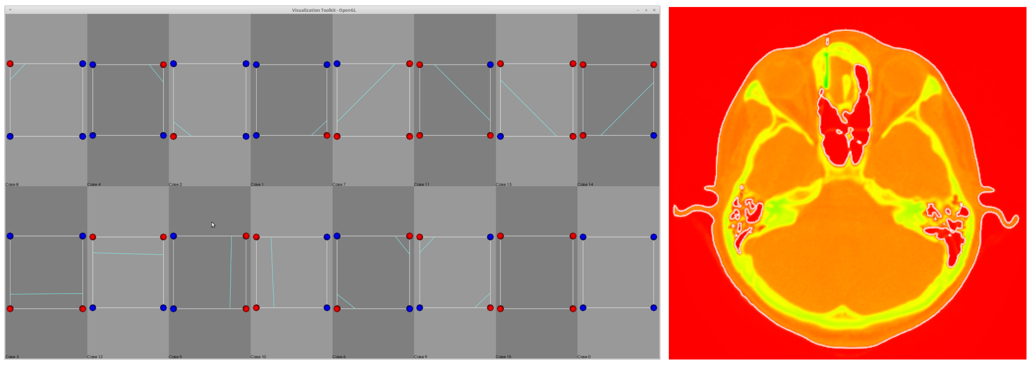

Figure 3: Left: Different cell cases. Right: Isocontour Visualization with Marching Squares.

In this task, you will implement the marching squares algorithm you learned about in the lecture. First, you will write code that handles all the different cases, and you will visualize them. To this end,

a) Implement determineCase() in `functions.py` (2P)

b) Implement createCaseEdgeTable() in `functions.py` (4P)

c) Implement findIntersectionPoint() in `functions.py` (2P)

d) In order to handle the ambiguous cases you have to implement two functions in `functions.py`:
   - identifySubcase() (2P)
   - asymptoticDecider() (1P)

e) Implement computeIntersectionPoints() in `functions.py` (2P)

Hints and descriptions for the functions are given as docstrings in each function. At this point, running `cellTest.py` should produce an image similar to the left one in Fig. 3.

In the second part, we will apply the algorithm to a real-word dataset. We have provided a framework for you that takes as input parameters the dataset and the isovalue and then displays the isocontour.

f) Complete the framework by implementing computeLineSegments() in `functions.py`. This function should iterate through all the cells in the image and add the intersection points. (5P)

At this point, running `isoLines.py` on `fullhead15.png`, which is provided on the lecture webpage, with iso value 700 should produce an image similar to Fig. 3 right. Please submit your code and screenshots of the results.

Good Luck!