

# Digital 3D Geometry Processing
# Exercise 4 – Curves

Handout date: Mon, 07.10.2019
Submission deadline: Thu, 17.10.2019, 23:00 h

## What to hand in

- From your modified source code, please upload **only** the file **curves.cpp**, not in a subfolder, and not compressed.
- We run an automated script to compile and test your homework, so it is necessary for us to find the file **curves.cpp** directly in your uploaded files.
- Optionally, if there are comments or problems in your code, upload a `readme.txt` with your comments.
- Submit your solutions to Moodle before the submission deadline. Late submissions will receive 0 points!

## 1 Theory Exercise (Not graded)

### 1.1 Curvature of Curves

Match each curve from Figure 1 with the corresponding curvature function:

$$k_1(s) = \frac{s^2-1}{s^2+1}, \quad k_2(s) = s, \quad k_3(s) = s^3 - 4s, \quad k_4(s) = \sin(s)s. \tag{1}$$

Write your solutions in a file named `TheoryExercise.pdf`. If, for example, the curvature function $k_1(s)$ corresponds to the curve **a**, write **1-a** as your solution. Briefly explain your answers.

### 1.2 Surfaces Area

King Archimedes wants to renovate his palace. The most striking structure is a spherical half-dome of 20m in diameter that covers the great hall. The king wants to cover this dome in a layer of pure gold. He has decided to split the work into two parts, each one covering a vertical slice of the dome of the same height (see Figure 2). For each part he hires different people and gives them 700kg of gold. The task is to cover the surface of one vertical slice with a layer of gold of 0.1mm thickness. The amount of gold that is left over is the salary for doing the job. Which slice should you pick if you want to make the most profit? Explain your answer in `TheoryExercise.pdf`.

How does your answer change when you have $n$ slices instead of just two?

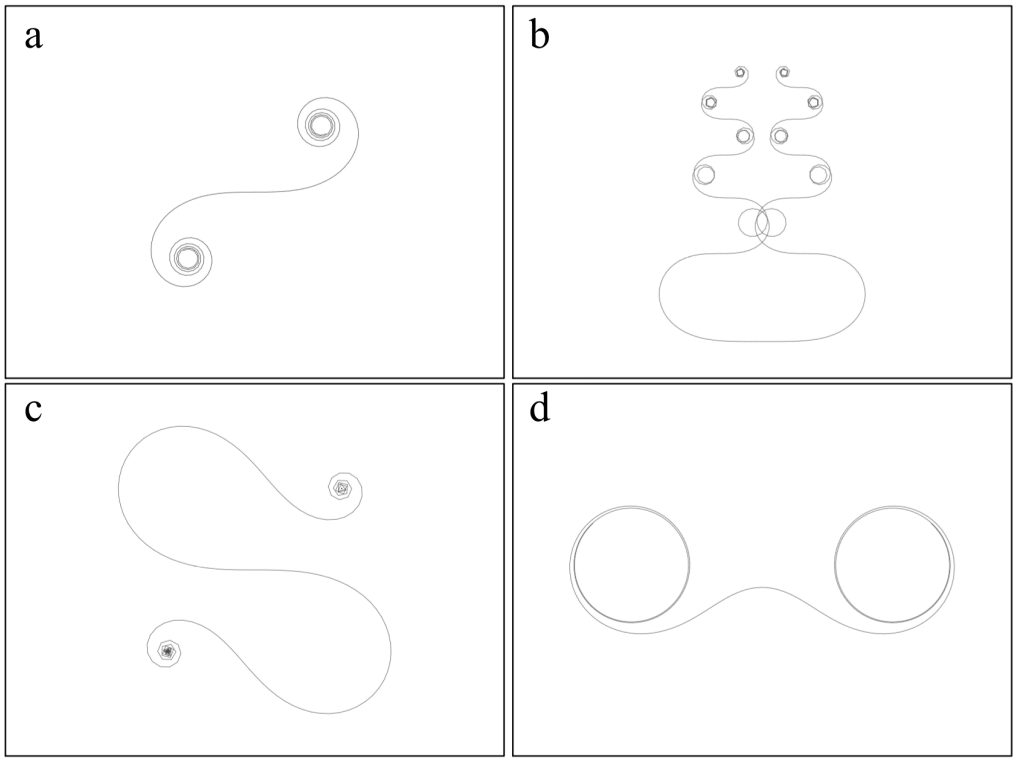


Figure 1: Curves reconstructed from given curvature functions $k_i(s)$.

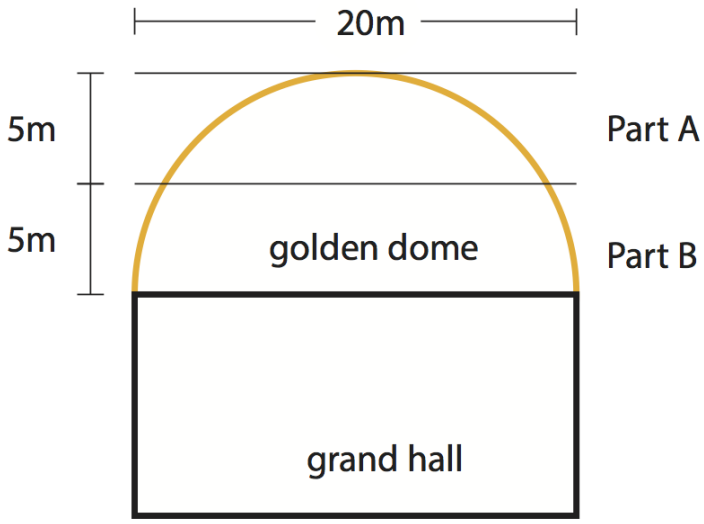


Figure 2: Sketch of King Archimedes dome.

# 2 Coding Exercise (100 pt)

## 2.1 The Framework

To set up the project, you will use CMake as in the exercises before. However, before performing the usual steps, you will need to have some external libraries installed in your system:

- On Ubuntu (15.04, 15.10, ...) follow these insturctions

```
~$ sudo apt-get install libglew-dev
~$ sudo apt-get install libglfw3 libglfw3-dev
```

- On Mac OSX (10.10.2, 10.11, ...) install `homebrew` from `http://brew.sh` and follow these instructions

```
~$ brew install glfw3
~$ brew install glew
```

- On Windows we provide the binaries for `glfw3` and `glew` (tested with Visual Studio 2019). This means you should be able to set up the project just like the previous exercises without any additional steps.
  If you encounter linking errors, you might need to build `glfw` and `glew` on your own. Download the source files from GLFW3 and GLEW, build, and copy the binaries to the folder: `dgp2018-exercise4/externals/{glfw3, glew}/lib`.

After compilation, you can test your smoothing on some curves by running the executable `curves/smoothing`.

## 2.1 The Task

For this exercise you will need to fill in the missing code in the file `curves.cpp`, in particular the functions `laplacianSmoothing()` and `osculatingCircle()`.
Given a curve in $\mathbb{R}^2$ as a closed polyline with vertices $\{V_i\}_{i=1}^{M}$, where $V_1 = V_M$, implement the following two examples of curve smoothing:

- Move every vertex towards the centroid of the two neighbors. More specifically, every vertex $V_i$ should shift to vertex $V_i'$ according to

$$V_i' = (1-\varepsilon)V_i + \varepsilon\frac{V_{i-1} + V_{i+1}}{2} \tag{2}$$

  where $\varepsilon$ is a small time step (in the code, this parameter is passed to the function). **After this, uniformly scale the curve back to its original length, in a way such that the centroid of the points doesn't move.**

  Implement this smoothing by filling in the `laplacianSmoothing()` function in the file `curves.cpp`. This function should perform one iteration of smoothing. In the GUI, press the key `S` to run this function.

- Move every vertex towards the center of the osculating circle. The osculating circle of vertex $V_i$ is a circumscribed circle $O$ of a triangle with the vertices $V_{i-1}$, $V_i$ and $V_{i+1}$. Let $C$ be center of the circumscribed circle $O$, then you should shift every vertex $V_i$ to $V_i'$ according to

$$V_i' = V_i + \varepsilon \frac{C - V_i}{\|C - V_i\|^2} \tag{3}$$

  where $\varepsilon$ is a small time step. **After this, uniformly scale the curve back to its original length, in a way such that the centroid of the points doesn't move.**

  Implement this smoothing by filling in the `osculatingCircle()` function in the file `curves.cpp`. This function should perform one iteration of smoothing. In the GUI, press the key `S` to run this function.

Both functions should modify the points of the curve by changing the Eigen matrix `MatMxN points`, which is a member of the struct. The position of each point is stored as a column in the matrix, and neighbouring points occupy neighbouring columns (the first and last point are also assumed to be neighbours, all curves in this exercise are *closed*).

For testing, use the keys `S` and `C` to perform smoothing, and keys `1-4` to display different curves.

**Note:** As for the last exercise, we will use automated testing to evaluate your code. You should make sure that you pass the test provided by the executable `curves/smoothing_test`. Note that this only evaluates a single test case for each smoothing type and relies on the fact that you changed no code, except for the two functions that do the smoothing.