Assignment in Computer Graphics II

– Assignment 5 –

Computer Graphics and
Multimedia Systems Group
David Bulczak, Christoph Schikora

Assignment 1 [2 Points] Catmull-Rom Approach

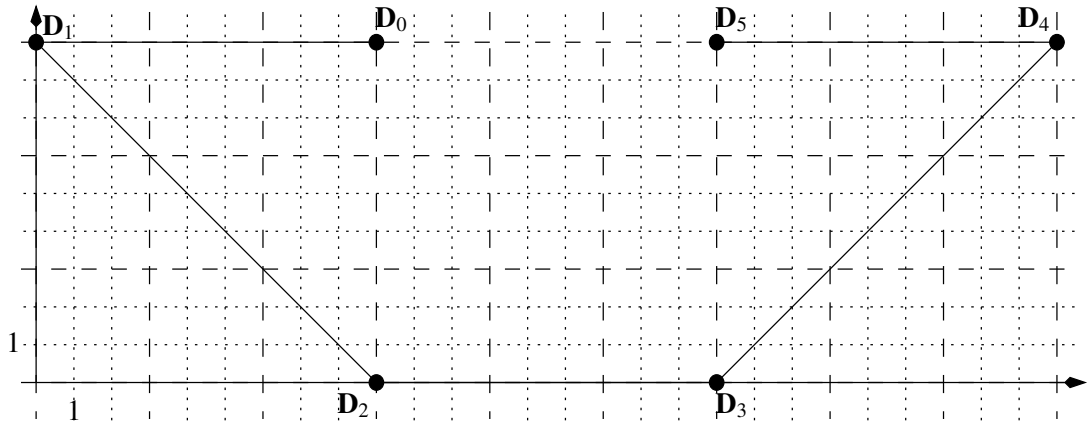
Calculate according to the Catmull-Rom approach, all control points for a cubic Bezier spline through the points $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$ whose tangents are constructed by the simple end tangent estimation. Additionally calculate the alternative tangents (with fitted parabola).

$$\mathbf{P}_0 = \begin{pmatrix} 2 \\ 0 \end{pmatrix} \quad , \quad \mathbf{P}_1 = \begin{pmatrix} 10 \\ 2 \end{pmatrix} \quad , \quad \mathbf{P}_2 = \begin{pmatrix} 20 \\ 6 \end{pmatrix} \quad .$$

Assignment 2 [2 Points] De Boor algorithm (uniform knot vector)

Given the following plotted de Boor points of a uniform, cubic B-Spline curve and the parameter $u = 4\frac{1}{3}$.

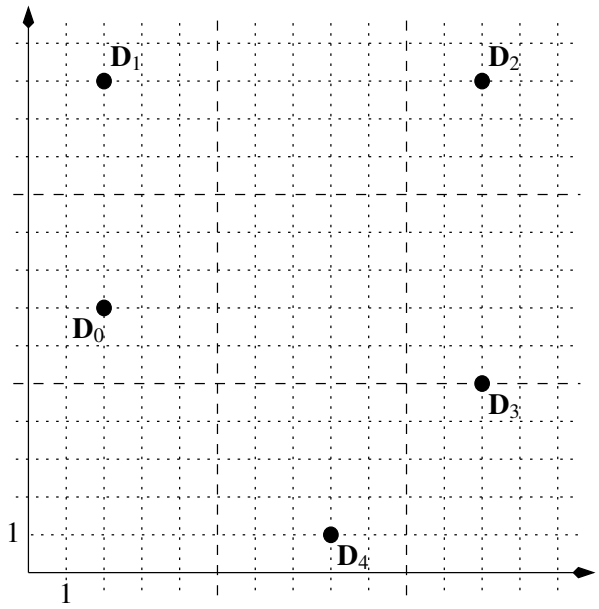
1. Which de Boor points are necessary for the evaluation of the curve at u.
2. Evaluate the curve geometrically and by calculation at u.



Assignment 3 [2 Points] De Boor algorithm (non-uniform knot vector)

Give a cubic B-Spline curve with $m = 4$, knot vector $T = \{0,0,0,0,1,2,2,2,2\}$ and control points

$$\mathbf{D}_0 = \begin{pmatrix} 2 \\ 7 \end{pmatrix} \quad , \quad \mathbf{D}_1 = \begin{pmatrix} 2 \\ 13 \end{pmatrix} \quad , \quad \mathbf{D}_2 = \begin{pmatrix} 12 \\ 13 \end{pmatrix} \quad , \quad \mathbf{D}_3 = \begin{pmatrix} 12 \\ 5 \end{pmatrix}, \quad \mathbf{D}_4 = \begin{pmatrix} 8 \\ 1 \end{pmatrix}$$



- Calculate $D(u)$ at $u = 1$. Use the de Boor algorithm.
- Name the knots and control points and draw them into the sketch below.

Assignment 4 [3 Points]

B-Spline curves (programming)

In this task you will extend the curve framework introduced with assignment 02. This time you have to implement B-Spline curves.

All relevant files for this and future programming tasks, related to curves, can be found in the `Curves` folder. In `Curve/Curve.hpp` you can find the abstract base class for all further curve classes. It provides three abstract member functions `eval, evalCurve, evalConstruct` which you will have to implement for all derived classes at least. Please study this class, read the comments and try to understand it. Everything else can be assumed to be a black box.

To build the project in your preferred development environment use the included CMake project (”CMakeLists.txt”). CMake can be downloaded from the following website: `http://www.cmake.org/`. Use the instructions on the page
`http://www.cmake.org/cmake/help/runningcmake.html` and the tutorial page to create the project.

1. Implement B-Spline curves in `Curve/BSplineCurve.hpp` and `Curve/BSplineCurve.cpp`.
   - `omega`: In this function you have to implement the weighting factor that is used to evaluate a B-Spline basis function. Use parameter `r` to distinguiish between $omega^r$ and $omega^l$.
   - `basisFunction`: In this function you have to implement the actual basis functions $N_i^n(u)$ that are used for B-Spline curve representation. Thus the parameter n represents the B-Spline

degree, i the i-th basis function. u is the parameter at which the basis functions has to be evaluated. Additionally this functions provides a parameter t that represents the knot vector (`std::vector`).

- `evalWithBasisFunction`: In this function you will have to evaluate the B-Spline curve for parameter u by using the basis functions.
- `alpha`: In this functions you have to implement the weighting factor that is used in de Boor's algorithm to compute points recursively.
- `evalNextLevel`: For a given array of points in this function you have to implement the recursion that determines the point of a next iteration.
- `eval`: In this function you have to evaluate the B-Spline curve by using de Boor's algorithm. You can assume that `controlPoints_` contains all de Boor points set by the user and that `knotVector_` contains the currently set knot vector. The class member variable `degree_` represents the current set B-Spline degree.
- `evalConstruct`: In this function you have to compute and store the intermediate results of the de Boor's algorithm. The results have to be stored in the `std::map` `contructedPoints_`.

HINT: To simplify some indexing we recommend to use `std::map` instead of `std::vector` to store and access points in algorithms you have to implement. This way you can use arbitrary indices while `std::vector` limits them by the current container size. For that purpose we have provided two aliases `PointMap` (`std::map<unsigned, glm::vec3`) and `IterationMap`(`std::map<unsigned, PointMap`). Feel free to use them for your algorithm implementation. Take a look on the corresponding documentation http://www.cplusplus.com/reference/map/map/.

Further explanations can be found in the comments of the code.

Hand in text assignment: Until 24.05.2017 12:00 o'clock in mailbox of our chair (next to room H-A 7107).

And hand in the programming assignment: Via e-mail until 31.05.2017 12:00 o'clock(`johnfr93@gmail.com`**).**