Geover

Lab 1

Aufgabe 1 Bézier-Kurven Auswertung

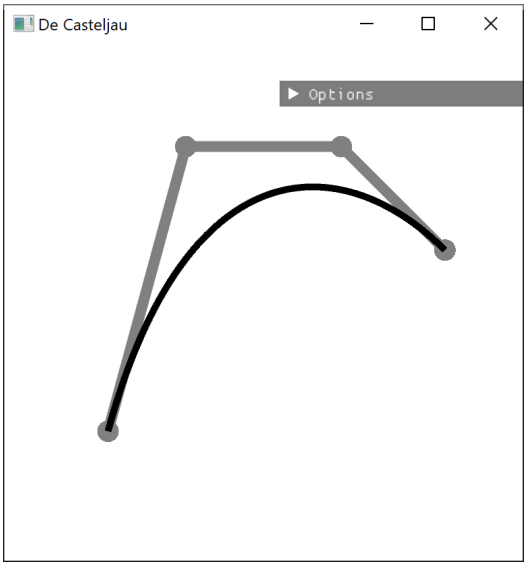
(a) Machen Sie sich mit der Klassenhierarchie von `ParametricCurve`, `PolynomialCurve` und `BezierCurve` vertraut.

(b) Implementieren Sie die Methode `ParametricCurve::sample` gemäß der Beschreibung im Kommentar.

(c) Realisieren Sie die Methode `BezierCurve::computeBinomialCoefficients`, welche die Binomialkoeffizienten für den angegeben Grad als Array zurück gibt.

(d) Nutzen Sie die Binomialkoeffizienten um die Bézierkurve an der Stelle t in der Methode `BezierCurve::evaluate` auszuwerten. Verwenden Sie dazu das Horner-Bézier-Schema.

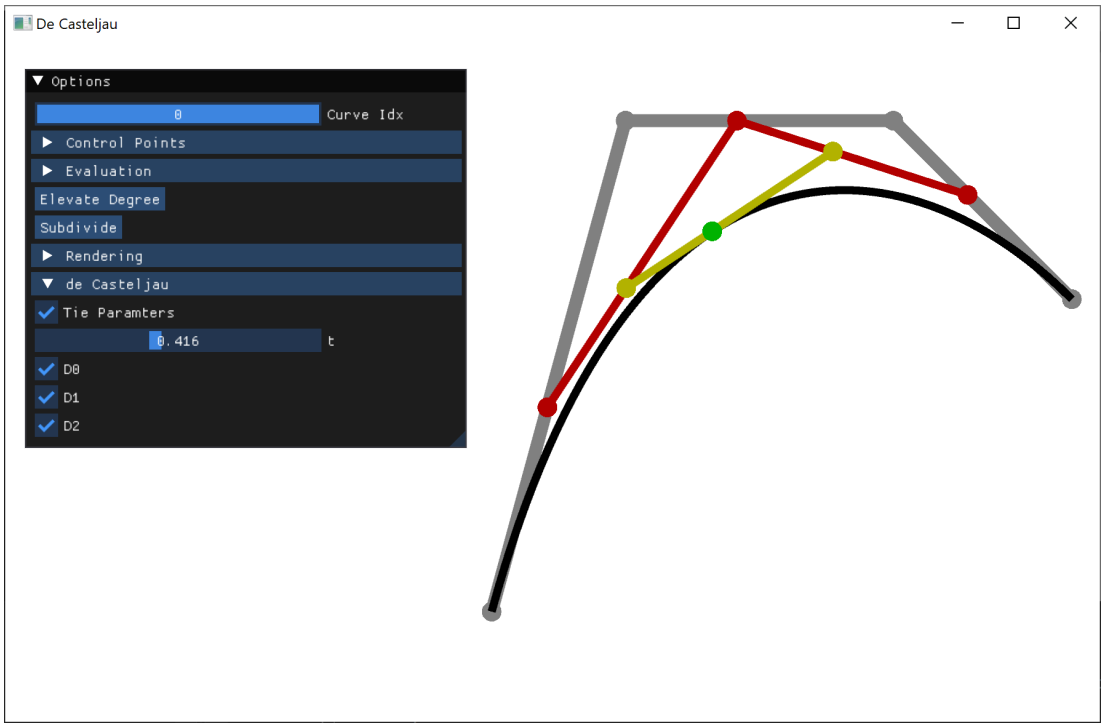
Ihre Belohnung ist folgendes Bild!



Aufgabe 2 DeCasteljau

(a) Die Methode `BezierCurve::deCasteljau(const std::vector<value_type>& t)` soll eine verallgemeinerte DeCasteljau-Pyramide berechnen. Dabei wird ein Array von Arrays `result` zurückgegeben. In `result[i]` soll dabei die i-te Ebene der DeCasteljau Pyramide zurückgegeben werden. Für jede Ebene soll aber ein extra Parameter `t[i]` verwendet werden.

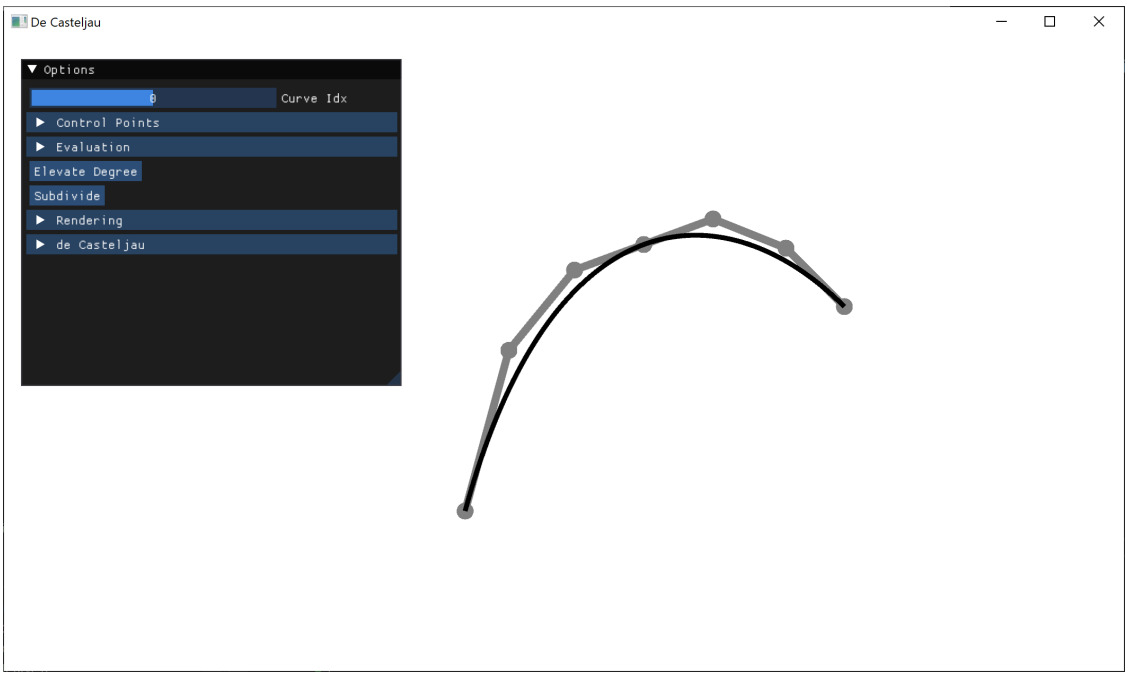
Experimentieren Sie UI-Bereich „de Casteljau“ mit der Visualisierung:



(b) Nutzen Sie die eben implementierte Methode um `deCasteljau(const value_type t)` zu programmieren, welche für alle Ebenen der DeCasteljau den gleichen Parameter `t` verwendet.

(c) Nutzen Sie die DeCasteljau-Pyramide um aus einer Bézier-Kurve zwei neue Kurven erzeugen, in dem Sie die alte Kurve genau in der Mitte teilen. Setzen Sie dies in `subdivide() const` um.

Nach einer Runde Subdivision (erreichbar über den UI-Knopf Subdivision) bekommen Sie folgendes Bild:



Aufgabe 3 Graderhöhung

Fügen Sie in der Klasse `BezierCurve` eine Methode `void elevateDegree()` hinzu, welche die aktuelle Bézierkurve durch eine neue Bézierkurve ersetzt, die einen Grad höher ist als die aktuelle Kurve, aber genau so aussieht.