BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Prof. Dr.-Ing. Kristian Hildebrand

Computergrafik 2 / Aufgabe 1.3
Parametrische Kurven und Bézierkurven

SoSo 2016

Bei dieser Aufgabe üben Sie die Darstellung allgemeiner parametrischer Kurven und Bézier-Kurven. Diese Aufgabe enthält **keinen weiteren Sourcecode**. Zur Lösung dieser Aufgabe erweitern Sie den Code aus den vorherigen Aufgaben.

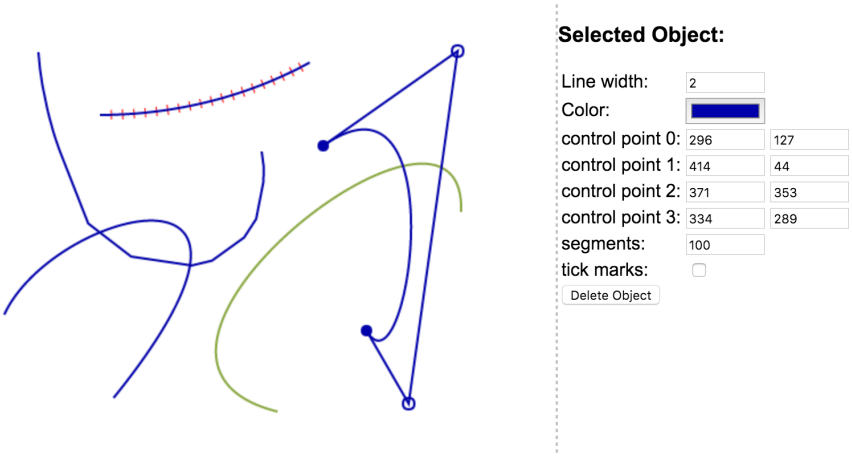

Abbildung 1: Ergebnisausgabe

Aufgabe 1: Aufgabenstellung

a) Implementieren Sie ein Objekt *ParametricCurve*, welches in der Lage ist, beliebige parametrische Kurven im Canvas darzustellen. Der Benutzer soll die Formel, mittels derer jeweils die x- und y-Koordinate für ein beliebiges t berechnet wird, als Text eingeben können (Siehe Screenshot oben auf dieser Seite). Neben der Berechnungsformel soll der Benutzer zwei Parameter t_{min} und t_{max} eingeben, die den Definitionsbereich der Kurve angeben. Die Anzahl der Liniensegmente, mit denen die Kurve angenähert wird, soll ebenfalls vom Benutzer festlegbar sein.

   i) Zur Umsetzung der Formel-Berechnung dürfen Sie die *gefährliche JavaScript-Funktion eval()* in dieser Aufgabe explizit verwenden. Fangen Sie jedoch Syntax-Fehler des Benutzers so ab, dass das Programm weiterhin funktioniert und der Benutzer den Fehler korrigieren kann. (Hinweis: Schauen Sie sich an, wie man *try/catch* in Javascript benutzt und geben Sie den Fehlerhinweis mit *alert* aus.)

   ii) Ihr *ParametricCurve*-Objekt benötigt wie schon der Circle die Methoden *draw()* und *isHit()*. Die Methode *createDraggers()* sollte eine leere Liste zurückliefern, da diese nur über die Funktionswerte der Eingabe verändert werden soll.

b) Implementieren Sie ein Objekt *BezierCurve*, welches basierend auf vier gegebenen Kontrollpunkten eine kubische Bézier-Kurve darstellt. *BezierCurve* funktioniert in vielen Fällen wie *ParametricCurve*.

   i) Verwenden Sie PointDragger zur Manipulation der Kontrollpunkte.

ii) Wenn das Objekt im Canvas selektiert ist, soll neben den vier Kontrollpunkten auch das Kontrollpolygon der Kurve visualisiert werden. Implementieren Sie dazu einen neuen Objekt-Typen, der lediglich das Polygon zeichnet und selbst keine Interaktivität implementiert.

Zusatzaufgabe (Voraussetzung für das Erreichen einer 1.0) Visualisieren Sie die Punkte, an denen die Kurve ausgewertet wurde, durch kleine Striche ("tick marks"), die senkrecht zur Kurve verlaufen. Machen Sie die Visualisierung dieser Striche im UI ein- und ausschaltbar.

Zusatzaufgabe (Optional / ohne Bewertung) Implementieren Sie die Darstellung der Bézierkurve zusätzlich mittels adaptiver Unterteilung durch den de-Casteljau-Algorithmus in einem weiteren .js Modul (z.B. AdaptiveBezierCurve.js). Schauen Sie sich dafür die Vorlesungsfolien nochmal an. Hinweis: Sie brauchen dafür einen rekursiven Aufruf, der die Punkte des Kontrollpolygons immer weiter unterteilt.

a) Eine maximale Unterteilungstiefe soll vorgebbar sein.

b) Ein maximaler Fehler (Abstand, Krümmung) soll vorgebbar sein.

c) Die Elemente der bei der Unterteilung konstruierten Kontrollpolygone sollen sichtbar zu machen sein.

Abgabe Dies ist *Teilaufgabe 1.3*; die Bearbeitungszeit der Teilaufgabe ist für ca. eine Woche ausgelegt. Die Abgabe der gesamten Aufgabe 1 soll via Git bis zu dem dort angegebenen Termin erfolgen. Der letzte Commit/Push zählt. Lokale Änderungen am SourceCode nach der Deadline sind nicht erlaubt. Verspätete Abgaben werden mit einem Abschlag von 1.0-Notenpunkten je angefangener Woche Verspätung belegt.

Demonstrieren und erläutern Sie Ihre Lösung in der nächsten Übung nach dem Abgabetag. Die Qualität Ihrer Demonstration ist, neben dem abgegebenen Code, ausschlaggebend für die Bewertung! Es wird erwartet, dass alle Mitglieder einer Gruppe anwesend sind und Fragen beantworten können.