Aufgabe Kurven

4 Praxisaufgaben zu Kurven

Hinweise zur Implementierung: Der Übung liegen verschiedene Materialien bei. Dabei handelt es sich um eine Musterlösung (für Windows), in denen Sie die Ergebnisse der einzelnen Aufgaben betrachten können, ein lauffähiges Programm als Quelltext, dem die von Ihnen im Rahmen der Aufgaben zu erstellenden Algorithmen fehlen und eine allgemeine Erklärung zum Programmaufbau. Die Programmstellen, denen Sie Quelltext hinzufügen müssen, sind gesondert markiert und enthalten weitere Hinweise zur Lösung der Aufgabe. Des Weiteren finden Sie den Quellcode der Musterlösung unter `src_solution`.

Die Bedienung der Musterlösung (und Ihres Arbeitsprogrammes) erfolgt mittels eines Kontextmenüs, das Sie über die rechte Maustaste erreichen. Neben den einzelnen Menüpunkten finden Sie in Klammern den entsprechenden Tastatur-Shortcut. Vorhandene Kontrollpunkte können bei gedrückter linker Maustaste verschoben werden. Durch Klick auf einen leeren Bereich wird ein neuer Kontrollpunkt an das Ende der Liste von Punkten hinzugefügt. Wird auf einen Kontrollpunkt geklickt während die Shift-Taste gedrückt ist, so wird der angeklickte Kontrollpunkt entfernt.

Projektdateien für Visual-Studio 2012/2013, 2015 und 2017 finden Sie in den entsprechenden Unterverzeichnissen des Ordners `build`. Falls Sie nicht mit Visual-Studio arbeiten, finden Sie im Unterordner `build/cmake` entsprechende CMake-Dateien. Bitte beachten Sie, dass die Lösung auf den Rechnern im Rechenzentrum, auf denen sich Visual-Studio 2013 befindet, laufen muss! Laden Sie für die Abgabe *nur* die Quellcode-Dateien in einer einzigen `.zip`-Datei hoch.

In der Implementierung existiert eine Oberklasse `abstract_curve`, welche die Gemeinsamkeiten aller Kurventypen kapselt. Alle Kurven haben eine Methode `evaluate_basis` zum Auswerten deren spezifischer Basisfunktionen an der Stelle t, welche als Argument den Index des Kontrollpunktes und den Kurvenparameter erhält. Die Basisfunktionen bestimmen für jeden Kontrollpunkt, wie groß sein Einfluss auf den Verlauf der Kurve an einem bestimmten Kurvenpunkt ist. Zur Bestimmung eines Punktes auf der Kurve erbt jede Kurvenklasse die in `abstract_curve` implementierte Methode `evaluate`, in der die Berechnung durch Summenbildung über alle Kontrollpunkte, gewichtet mit deren Basisfunktionen, erfolgt. Für die Visualisierung existiert die Klasse `curve_renderer`, der eine Kurve übergeben werden kann. Nachdem sie in gleichmäßigen Abständen über t abgetastet wurde, werden die gewonnenen Punkte durch `render_curve` mit einem Linienzug verbunden.

4.1 Parametrische Kurven

In der Vorlesung wurden verschiedene Möglichkeiten besprochen, Kurven zu approximieren, wenn Kontrollpunkte vorgegeben sind. In diesen Praxisaufgaben sollen Sie nun lernen, Gemeinsamkeiten der einzelnen Verfahren zu erkennen und diese Techniken einzusetzen. Hierfür werden Sie BSpline- und Bézierapproximationen sowie Hermite-Interpolation durchführen.

4.1.1 Berechnung eines Kurvenpunktes

In der Klasse `abstract_curve` aus der Datei `abstract_curve.cpp` existiert die Methode `evaluate`, welche aus den Kontrollpunkten und deren Basisfunktionen an der Stelle t einen Punkt auf der Kurve bestimmt. Diese Methode wird von allen Kurventypen verwendet. Implementieren Sie ihre Funktionalität. Zum Testen können Sie die Lagrange-Kurve verwenden, für die die Berechnung der Basisfunktionen bereits implementiert ist. Da eine graphische Ausgabe der Kurve erst in den folgenden Teilaufgaben

implementiert wird, können Sie Ihre Lösung für die Lagrangekurve mit den vorgegebenen Kontrollpunkten durch die folgende Tabelle prüfen. Die Ausgabe der Kurvenpunkte auf der Konsole für die in der Tabelle vorkommenden Werte für t ist vorläufig in der Methode `curve_renderer::sample_curve` aus `curve_renderer.cpp` implementiert.

t	$0,0$	$0,2$	$0,4$	$0,6$	$0,8$	$1,0$
Punkt	$(100, 400)$	$(100, 290)$	$(110, 200)$	$(130, 130)$	$(160, 80)$	$(200, 50)$

t	$1,2$	$1,4$	$1,6$	$1,8$	$2,0$
Punkt	$(250, 40)$	$(310, 50)$	$(380, 80)$	$(460, 130)$	$(550, 200)$

4.1.2 Darstellung der Kurve

Zum Rendern einer Kurve dient die Methode `render_curve` der Klasse `curve_renderer`, deren Programmcode Sie in der Datei `curve_renderer.cpp` finden. Hier soll der Inhalt einer Liste von abgetasteten Kurvenpunkten (mit dem Namen `samples`) durch einen Linienzug dargestellt werden. Zum Füllen dieser Liste wird zuvor die Methode `sample_curve` aufgerufen, die sie ebenfalls implementieren sollen. Entfernen Sie zuerst den in dieser Methode vorhandenen Quellcode, welcher die in 4.1.1 beschriebene Ausgabe erzeugte. Beachten Sie bei der Implementierung den Definitionsbereich von t. Die Membervariable `sample_count` bestimmt die Anzahl der zu erzeugenden Punkte.

4.1.3 Darstellung der Basisfunktionen

Zur Darstellung der Basisfunktionen einer Kurve existiert die Methode `render_basis_functions` in der Klasse `curve_renderer`. Implementieren Sie dessen Funktionalität indem Sie pro Basisfunktion über den gesamten Definitionsbereich von t einen Linienzug zeichnen. Weitere Hinweise können Sie den Kommentaren zu dieser Aufgabe entnehmen.

4.1.4 Bézier-Approximation

Implementieren Sie die Funktionen `evaluate_basis` in der Klasse `bezier_curve`.

4.1.5 Hermite-Interpolation

Implementieren Sie die Funktionen `evaluate_basis` und `evaluate` der Klasse `hermite_spline`. Die speziell für die Hermite-Interpolation vorhandene Neuimplementierung von `evaluate` ist notwendig, da Sie nicht nur die Kontrollpunkte benötigen, sondern auch Richtungsvektoren zwischen zwei Kontrollpunkten. Hier soll nur ein Hermite-Segment unterstützt werden und kein Hermite-Spline. Nutzen Sie den ersten und letzten Kontrollpunkt als Anfangs- und Endpunkt des Hermite-Segments und berechnen Sie Anfangs- und Endtangente aus den ersten bzw. letzten beiden Kontrollpunkten. Prüfen Sie zuvor, ob mindestens drei Kontrollpunkte vorhanden sind.

4.1.6 BSpline-Approximation

Implementieren Sie die Methode `evaluate_basis` in der Klasse `bspline`. Dafür existiert neben der bekannten Version von `evaluate_basis` eine weitere, welcher der Kurvengrad als Parameter übergeben werden kann. Es steht Ihnen frei, diese Methode zu verwenden, falls Sie eine rekursive Berechnung der Basisfunktionen durchführen möchten. Eine solche Berechnung hat allerdings eine im Grad des Splines exponentielle Laufzeit. Überlegen Sie, mit welchen Methoden man die Berechnung beschleunigen kann.

Zusätzlich müssen Sie die Methode `calculate_knot_vector` vervollständigen um einen nicht-uniformen Knotenvektor zu erzeugen, der offene Splines mit Start- und Endpunktinterpolation der Kontrollpunkte erlaubt. Schauen Sie sich dazu die Beispiele im Skript an.

4.2 Zusatzaufgaben

Hinweis: Die erreichbare Bonuspunktezahl für eine korrekt gelöste Zusatzaufgabe finden Sie am Ende der jeweiligen Aufgabenbeschreibung. Maximal können jedoch nicht mehr als 3 Bonuspunkte von diesem Übungsblatt eingebracht werden.

- Implementieren und visualisieren Sie das rekursive Auswerteschema der Bézierkurven nach De-Casteljau für einstellbares t. Dazu ist es notwendig die Klasse `curve_renderer` so zu erweitern, dass in der `render_curve`-Methode der Kurventyp der zu rendernden Kurve geprüft wird. Falls es sich um eine Bézier-Kurve handelt soll De-Casteljau verwendet werden, ansonsten ihre bereits vorher implementierte Darstellungsmethode. (1,5 Ptk.)
- Erweitern Sie die Anwendung, sodass eine graphische Anpassung des Stützstellenvektors für BSplines möglich ist. Beachten Sie dabei, dass immer $u_i \leqq u_{i+1}$ gelten muss. Die Implementierung kann in der Klasse `control_points_editor` erfolgen. Da keine vorgegebenen Methoden um etwa Slider zu erzeugen existieren, müssen Sie zunächst eine solche Funktionalität implementieren. Desweiteren müssen Sie die Klasse `bspline` um eine Methode zur Angabe des Stützstellenvektors erweitern. (2 Pkt.)
- Implementieren Sie den schnellen De-Boor-Algorithmus zur Auswertung der BSplines wie er zum Beispiel unter `http://www.cs.mtu.edu/~shene/COURSES/cs3621/NOTES/spline/de-Boor.html` beschrieben ist. Erstellen Sie dafür für die Klasse `bspline` eine eigene Version der Methode `evaluate`. (1 Pkt.)
- Erweitern Sie die Klasse `hermite_spline` derart, dass beliebig viele Kontrollpunkte zur Erzeugung eines Hermitesplines verwendet werden können. Orientieren Sie sich dazu am Catmull-Rom-Spline. (1 Pkt.)