GMCAD/CAE Programmieraufgabe 2

Abgabemodalitäten

- **Testatstermin:** Mittwoch, 11. Januar in der Übungsstunde.
- Zu diesem Termin stellen alle Gruppenmitglieder gemeinsam die Lösungen vor.

Ziel dieser Programmieraufgabe ist das Modellieren, Auswerten und Darstellen von Bézierkurven, Rationalen Bezierkurven und NURBS. Erweitern Sie das bereitgestellte Framework. Zu bearbeitende Stellen sind in unserem Framework mit `TODO` gekennzeichnet.

Aufgabe 2.1: Bézierkurven

Erweitern Sie ihr Framework, sodass Sie dreidimensionale Bézierkurven modellieren und visualisieren können. Verwenden und erweitern Sie die Klasse `BezierCurve`.

- Implementieren Sie den Algorithmus von de Casteljau, der eine Kurve beliebigen Grades an gegebener Stelle $u \in [0,1]$ in zwei Teilkurven aufteilt (`separateCurveAt`).
- Benutzen Sie diese Funktion, um den Auswertungspunkt und die Tangente an einer gegebener Stelle $u \in [0,1]$ zu berechnen (`evaluateCurveAt`).
- Implementieren Sie eine Funktion, welche die Bézier-Kurven an geeignet vielen Stellen auswertet und visualisiert. Visualisieren Sie ebenso das Kontrollpolygon (`evaluateCurve`).
  .
- Visualisieren Sie die Auswertung einer Bézier-Kurve an einem gewählten Parameter (`evalParamter`). Zeichnen Sie das Kontrollpolygon der Teilkurven. Der Parameter soll interaktiv veränderbar sein (über Tastatureingabe). Implementieren sie hierzu die Funktionen `drawBezier` und `drawBezierCtrlPolygon` in der Datei `CurveRendering.cpp`.

Aufgabe 2.2: Rationale Bézierkurven

Erweitern Sie ihr Framework, sodass Sie **zweidimensionale**, rationale Bézierkurven modellieren und visualisieren können.

- Verwenden Sie hierfür die homogene Darstellung

$$C^w(u) = \sum_{i=0}^{n} B_{i,n}(u)\mathbf{P}_i^w, \quad u \in [0,1]$$

  mit gewichteten Punkten $\mathbf{P}_i^w = (w_i P_i^x, w_i P_i^y, w_i)$.
- Nutzen Sie bereits implementierte Algorithmen aus Programmieraufgabe 2.1, um die Kurve auszuwerten und Tangenten zu bestimmen.
- Visualisieren Sie die Projektion auf die $w = 1$-Ebene. Implementieren sie hierzu die Funktionen `drawRationalBezier` und `drawRationalBezierCtrlPolygon` in der Datei `CurveRendering.cpp`.
- Verwenden Sie Rationale Bezierkurven, um analytische Geometrien wie beispielsweise eine Hyperbel oder ein Kreissegment zu modellieren.

Aufgabe 2.3: NURBS Kurven

Erweitern Sie das gegebene Framework, sodass Sie Non-Uniform Rational B-Splines (NURBS) modellieren und visualisieren können. Verwenden Sie hierfür die Klasse `NURBSCurve`, die NURBS Kurven mit beliebigem Polynomgrad und beliebiger Anzahl von Kontrollpunkten repräsentieren kann.

- NURBS Kurven werden durch den Polynomgrad p, den Knotenvektor $U = [u_0, \ldots, u_{m+1}]$ und die Menge an homogenen Kontrollpunkten $\{\mathbf{P}_0^w, \ldots, \mathbf{P}_{n+1}^w\}$ repräsentiert. Die Anzahl der Kontrollpunkte, die Länge des Knotenvektors und der Polynomgrad sind durch die Relation $m = n + p + 1$ voneinander abhängig. Verwenden Sie zunächst einen uniformen Knotenvektor bei der Erstellung einer NURBS Kurve.
- Implementieren Sie eine Methode zum Einfügen von Knoten in den Knotenvektor `insertKnot` .
- Implementieren Sie den in der Vorlesung vorgestellten de Boor-Algorithmus zur Auswertung der Kurven an beliebigen Stellen u `evaluteDeBoor` .
- Implementieren Sie eine Funktion, welche dieNURBS-Kurven an geeignet vielen Stellen auswertet und visualisiert. Visualisieren Sie ebenso das Kontrollpolygon (`evaluateCurve`). .
- Visualisieren Sie die Kurven - ähnlich zur Programmieraufgabe 2.1 - in dem Sie die Kurve und dessen Kontrollpolygon zeichnen `drawNURBS` und `drawNURBSCtrlPolygon`.
- Visualisieren Sie die Kurven homogenisiert in `drawNURBS_H` und `drawNURBSCtrlPolygon_H`.
- Demonstrieren Sie mit Ihrem Programm, welchen Einfluss Änderungen der Kontrollpunktgewichte auf den Verlauf der Kurve haben.
- Wie ändert sich der Verlauf der Kurve, wenn Sie (gültige) Änderungen an den Werten im Knotenvektor vornehmen?
- Verwenden Sie NURBS, um analytische Geometrien wie beispielsweise eine Hyperbel oder ein Kreis(-segment) zu modellieren.