CES Softwareentwicklungspraktikum

Analyse- und Entwurfsdokument

Center for Computational Engineering Science
RWTH Aachen University

Stefan Jeske, Daniel Partida, Tom Witter und Chun-Kan Chow

Matr.-Nr. 334033, 335179, 333265, 333715
email:
[stefan.jeske|daniel.partida|tom.witter|chun-kan.chow]
@rwth-aachen.de

Inhaltsverzeichnis

Kapitel 1

Vorwort

1.1 Aufgabenstellung und Struktur des Dokuments

Sehr geehrte Damen und Herren,

Bitte entwerfen und implementieren sie eine Applikation, die die Interpolation von Werten zwischen festgelegten Punkten in einer X-Y-Ebene erlaubt. Die Interpolation soll genutzt werden, um die Punkte graphisch mit einer Linie zu verbinden.
Die Interpolationsroutine soll für m Kontrollpunkte die X- und Y-Koordinaten von n Stützstellen berechnen, die äquidistant (Abstand dx) im Intervall [xmin; xmax] verteilt liegen.
Ausschlaggebend für eine gute Integration der Software in unseren Arbeitsprozess ist, dass mithilfe einer graphischen Benutzerschnittstelle die Randbedingungen und die Interpolationsart eingestellt werden können.
Weiterhin soll eine Visualisierung der resultierenden Interpolation auf der Benutzeroberfläche möglich sein.

Wir freuen uns auf eine Zusammenarbeit.

1.2 Projektmanagement

Tom Witter:

- Benutzeranforderungen
- Implementation InterpolationControl

Stefan Jeske:

- Benutzerdokumentation
- Implementation MainWindow, User Interface

Daniel Partida:

- Aufgabenstellung und Struktur des Dokuments
- Implementation MainWindow

Chun-Kan Chow:

- Entwicklerdokumentation
- Implementation Interpolationsarten (CubicSpline, PolynomialInterpolation)

Alle:

- Aktivitätsdiagramme
- UC Beschreibungen
- funktionale und nicht funktionale Anforderungen
- Begriffsanalyse
- Sequenzdiagramme
- Klassendiagramm
- Check

1.3 Lob und Kritik

Danke Naumann für die Organisation des Projektes und die gewonnene Praxiserfahrung. ;-)

Kapitel 2

Analyse

2.1 Anforderungsanalyse

2.1.1 Benutzeranforderungen

Die Applikation ermöglicht es dem Benutzer, verschiedene Punkte in ein Koordinatensystem einzutragen und diese durch Interpolation graphisch zu verbinden.

Die Applikation gibt dabei zunächst ein 100x50 Koordinatensystem auf der X-Y-Ebene vor.
Auf der Benutzeroberfläche kann der Benutzer die Größe des Koordinatensystems, bzw. den Definitionsbereich verändern.
Mit der linken Maustaste kann der Benutzer Punkte im abgebildeten Koordinatensystem auf der Zeichenfläche einzeichnen und mit einem rechten Mausklick auf einen Punkt kann der Benutzer diesen löschen. Ist mehr als ein Punkt in dem angeklickten Löschradius, so löscht die Applikation den Punkt mit dem kleinsten x-Wert.
Sind mindestens zwei Punkte eingezeichnet, nachdem ein Punkt hinzugefügt oder gelöscht wurde, dann startet die Applikation automatisch die Interpolation und verbindet die eingezeichneten Punkte auf der Zeichenfläche mithilfe der gewählten Interpolationsmethode.
Der Benutzer kann auf einer Auswahlbox auf der Benutzeroberfläche zwischen einer Interpolation durch ein Polynom (Grad m-1 bei m zu Interpolierenden Punkten), oder durch lineare oder kubische Splines wählen.
Nach dem Start des Programmes oder nach dem Zurücksetzen ist die Interpolationsart standardmäßig auf lineare Splines eingestellt.
Der Benutzer kann zu jeder Zeit mit dem Button “Reset”das leere 100x50 Koordinatensystem mit den Standardeinstellungen wiederherstellen oder mit dem Button “Beenden”das Programm beenden.

2.1.2 Anwendungsfallanalyse

Anwendungsfalldiagramm:

Abbildung 2.1: Anwendungsfalldiagramm

Aktivitätsdiagramme:
Art der Interpolation ändern

- *Ziel:* Der Nutzer will die Interpolationsart ändern.
- *Einordnung:* Hauptfunktion
- *Vorbedingung:* Die Applikation ist geöffnet und zeigt die Zeichenfläche sowie die Benutzeroberfläche an.
- *Nachbedingung:* Die Applikation zeigt die neu berechnete Kurve auf der Zeichenfläche an.
- *Nachbedingung im Fehlerfall:*
- *Hauptakteure:* Nutzer
- *Nebenakteure:*
- *Auslöser:* Der Nutzer will die Interpolationsart ändern.
- *Standardablauf:*

  (1) Der Nutzer klickt mit der linken Maustaste auf die Auswahlbox mit der Überschrift “Interpolationsart”, welche sich auf der Benutzeroberfläche befindet.

(2) Die Applikation zeigt alle möglichen Interpolationsmethoden in der Auswahlbox an.

(3) Der Nutzer wählt mit der linken Maustaste eine Interpolationsart aus.

(4) Die Applikation führt die Interpolation mit der ausgewählten Interpolationsmethode aus.

(5) Die Applikation zeigt die neu berechnete Interpolation auf der Zeichenfläche an.

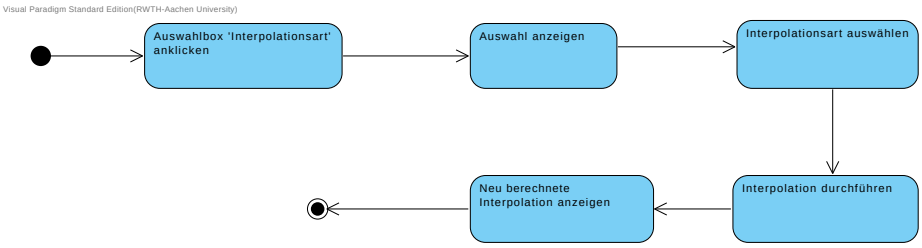

Abbildung 2.2: Interpolationsart aendern

Reset

- *Ziel:* Es sollen alle Stützstellen und die Kurve von der Zeichenfläche gelöscht, sowie der Definitionsbereich und die Interpolationsart auf den Defaultmodus (Lineare Spline Interpolation ([0,100]x[0,50]) zurückgesetzt werden.
- *Einordnung:* Hauptfunktion
- *Vorbedingung:* Die Applikation ist geöffnet und zeigt die Zeichenfläche sowie die Benutzeroberfläche an.
- *Nachbedingung:* Die Applikation hat alle Stützstellen auf der Zeichenfläche gelöscht und der Definitionsbereich sowie die Interpolationsart sind im Defaultmodus.
- *Nachbedingung im Fehlerfall:*
- *Hauptakteure:* Nutzer
- *Nebenakteure:*
- *Auslöser:* Der Nutzer möchte die Applikation auf den Defaultmodus zurücksetzen.
- *Standardablauf:*

  (1) Der Nutzer klickt mit der linken Maustaste auf den Button 'Reset', welcher sich auf der Benutzeroberfläche befindet.

(2) Die Applikation stellt den Defaultmodus wieder her und löscht die Stützstellen.

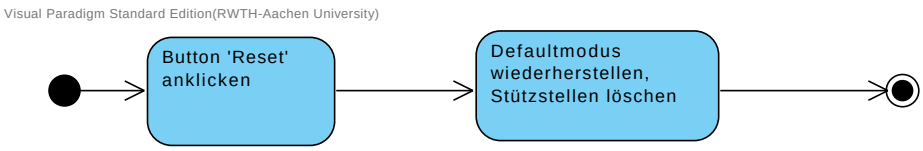

Abbildung 2.3: Reset

Definitionsbereich ändern

- *Ziel:* Der Definitionsbereich soll geändert werden
- *Einordnung:* Hauptfunktion
- *Vorbedingung:* Die Applikation ist geöffnet und zeigt die Zeichenfläche sowie die Benutzeroberfläche an.
- *Nachbedingung:* Die Applikation zeigt die aktualisierte Zeichenfläche sowie Benutzeroberfläche mit den neuen Grenzen an.
- *Nachbedingung im Fehlerfall:* Die Applikation zeigt eine Fehlermeldung an und verändert die Zeichenfläche nicht.
- *Hauptakteure:* Nutzer
- *Nebenakteure:*
- *Auslöser:* Der Nutzer will den Definitionsbereich ändern
- *Standardablauf:*
  (1) Der Nutzer klickt mit der linken Maustaste auf eins der Textfelder auf der Benutzeroberfläche unter der Überschrift “Definitionsbereich”.
  (2) Der Nutzer ändert den Wert im Textfeld.
  (3) Der Nutzer betätigt mit der linken Maustaste den Button ’Definitionsbereich setzen’.
  (4) Die Applikation prüft, ob Zahlen eingegeben wurden.
  (5) Die Applikation prüft, ob $x_{min} > x_{max}$ ist.
  (6) Die Applikation prüft, ob $y_{min} > y_{max}$ ist.
  (7) Die Applikation ändert den Definitionsbereich auf die gewünschten Werte.
- *Verzweigungen:*

(2a1) Der Nutzer will noch einen anderen Wert des Definitionsbereiches ändern.

(2a2) Der Nutzer geht zu Schritt '1' zurück.

(4a1) Der Nutzer hat eine ungültige Eingabe getätigt.

(4a2) Die Applikation gibt eine Fehlermeldung aus.

(5a1) Die Applikation erkennt, dass '$x_{min} > x_{max}$' ist.

(5a2) Die Applikation gibt eine Fehlermeldung aus.

(6a1) Die Applikation erkennt, dass '$y_{min} > y_{max}$' ist.

(6a2) Die Applikation gibt eine Fehlermeldung aus.

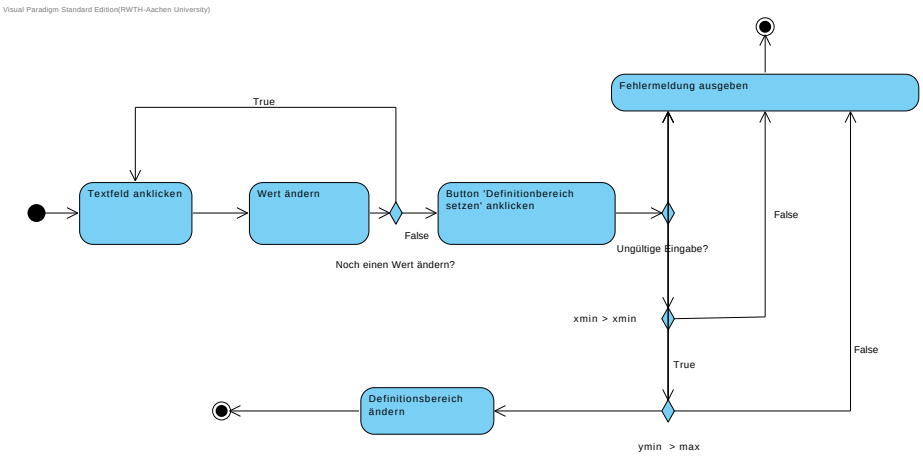

Abbildung 2.4: Definitionsbereich aendern

Beenden

- *Ziel:* Die Applikation soll beendet werden
- *Einordnung:* Hauptfunktion
- *Vorbedingung:* Die Applikation ist geöffnet und zeigt die Zeichenfläche sowie die Benutzeroberfläche an.
- *Nachbedingung:* Die Applikation wurde erfolgreich beendet.
- *Nachbedingung im Fehlerfall:*
- *Hauptakteure:* Nutzer

- *Nebenakteure:*
- *Auslöser:* Der Nutzer möchte die Applikation beenden.
- *Standardablauf:*
  (1) Der Nutzer betätigt auf der Benutzeroberfläche mit der linken Maustaste den Button 'Beenden'.
  (2) Die Applikation schließt.

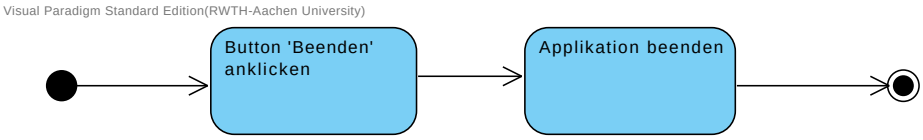

Abbildung 2.5: Beenden

Stützstelle löschen

- *Ziel:* Es soll eine vom Nutzer angeklickte Stützstelle von der Zeichenfläche gelöscht werden.
- *Einordnung:* Hauptfunktion
- *Vorbedingung:* Die Applikation ist geöffnet und zeigt die Zeichenfläche sowie die Benutzeroberfläche an.
- *Nachbedingung:* Die Applikation zeigt die neu berechnete Interpolation ohne die gelöschte Stützstelle auf der Zeichenfläche an.
- *Nachbedingung im Fehlerfall:* Die Applikation zeigt die ursprüngliche Interpolation an.
- *Hauptakteure:* Nutzer
- *Nebenakteure:*
- *Auslöser:* Der Nutzer möchte eine Stützstelle von der Zeichenfläche löschen
- *Standardablauf:*
  (1) Der Nutzer klickt mit der rechten Maustaste auf eine beliebige Stelle der Zeichenfläche.
  (2) Die Applikation löscht die im Löschradius befindliche Stützstelle mit dem kleinsten x-Wert.
  (3) Die Applikation führt die Interpolation mit den verbleibenden Stützstellen aus.

(4) Die Applikation zeigt die neu berechnete Interpolation auf der Zeichenfläche an.

- *Verzweigungen:*

(1a) Die Applikation kann keine Stützstelle im Löschradius identifizieren.

(2a) Der Use Case wird beendet.

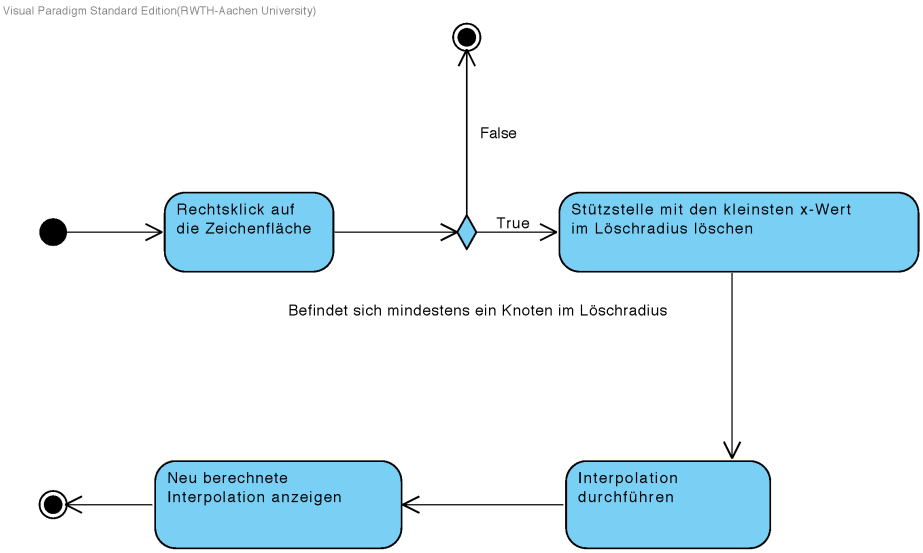

Abbildung 2.6: Stuetzstelle loeschen

Stützstelle hinzufügen

- *Ziel:* Die Interpolation soll um eine Stützstelle erweitert werden.
- *Einordnung:* Hauptfunktion
- *Vorbedingung:* Die Applikation ist geöffnet und zeigt die Zeichenfläche sowie die Benutzeroberfläche an.
- *Nachbedingung:* Die Applikation zeigt die neu berechnete Interpolation mit der hinzugefügten Stützstelle auf der Zeichenfläche an.
- *Nachbedingung im Fehlerfall:*
- *Hauptakteure:* Nutzer
- *Nebenakteure:*

- *Auslöser:* Der Nutzer möchte eine Stützstelle auf der Zeichenfläche hinzufügen.
- *Standardablauf:*
  (1) Der Nutzer klickt mit der linken Maustaste auf eine beliebige Stelle der Zeichenfläche.
  (2) Die Applikation erzeugt eine Stützstelle auf dem angeklickten Punkt.
  (3) Die Applikation führt die Interpolation mit den neuen Stützstellen aus.
  (4) Die Applikation zeigt die neu berechnete Interpolation auf der Zeichenfläche an.

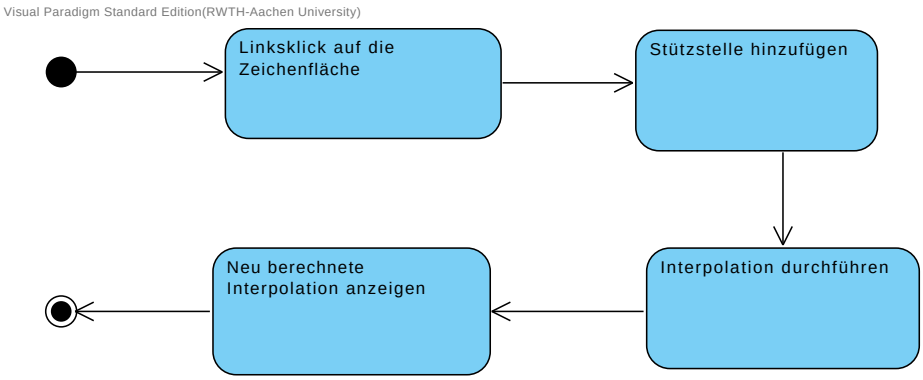

Abbildung 2.7: Stuetzstelle hinzufuegen

2.2 Systemanforderungen

2.2.1 Funktionale Anforderung

- Anzeigen einer Zeichenfläche
- Anzeigen einer Benutzeroberfläche, welche folgende Funktionen bereitstellt:
  1. Auswahl der Interpolationmethode
  2. Änderung des Definitionsbereichs
  3. Zurücksetzten der Zeichenfläche und Einstellungen auf default
  4. Beenden Button
- Anzeigen von Fehlermeldungen
- Eingabe von Gleitkommazahlen

- Anzeigen einer Interpolationskurve
- Darstellung von Punkten auf der Zeichenfläche
- Speichern der Punkte (Gleitkommazahlen, ganze Zahlen)
- Löschen und Hinzufügen von Punkten
- Test auf Gültigkeit des Definitionsbereichs
- Anzeigen von Gleitkommazahlen auf 2 Nachkommastellen
- Nach dem Starten des Programms werden Defintionsbereich und Interpolationsart standardmäßig fesgelegt

2.2.2 Nicht Funktionale Anforderung

- Hinzufügen und Löschen der Punkte durch Mausklick möglich
- Eingabe der Zahlen durch Tastatur
- Skalierung des Hauptfensters bei unterschiedlichen Auflösungen.

2.3 Begriffsanalyse

Klassenkandidaten:

- **Zeichenfläche**
- **Benutzeroberfläche**
- Definitionsbereich
- **Interpolationsart**
- Button
- Textfeld
- Auswahlbox
- **MainWindow**
- Stützstelle
- Löschradius

Begriffsmodell:

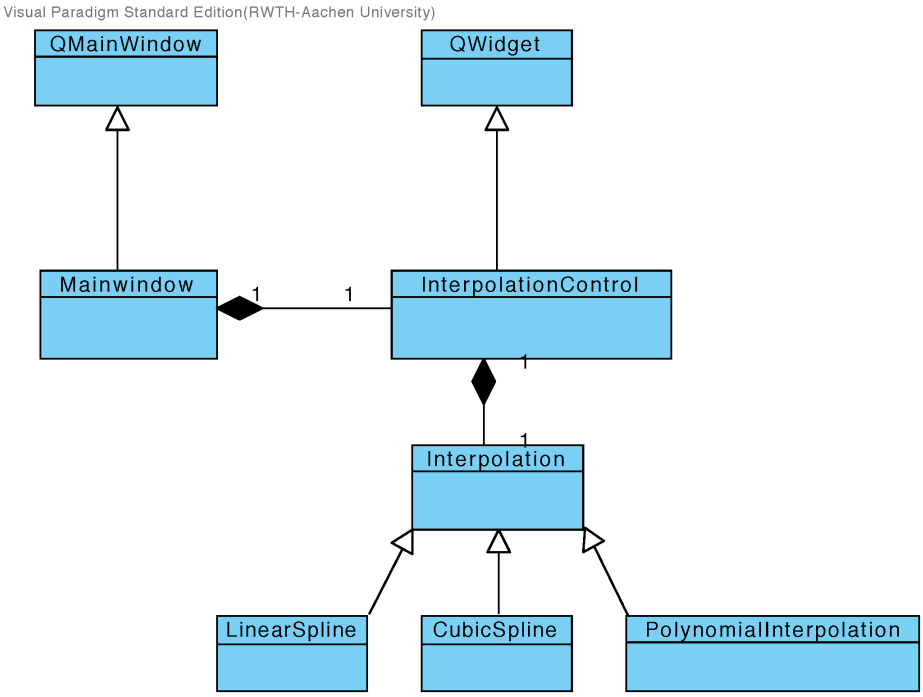

Abbildung 2.8: Begriffsmodell

Kapitel 3

Entwurf

3.1 Detailentwurf: Klassen

3.1.1 Statik

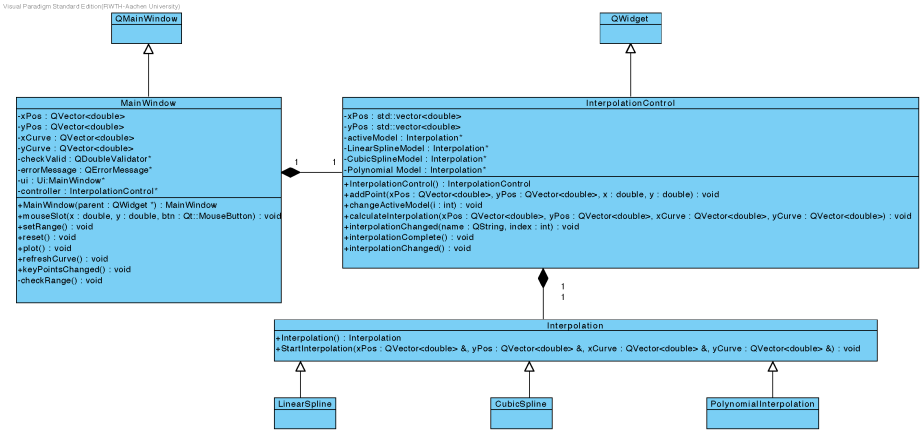

Abbildung 3.1: Klassendiagramm

3.1.2 Dynamik

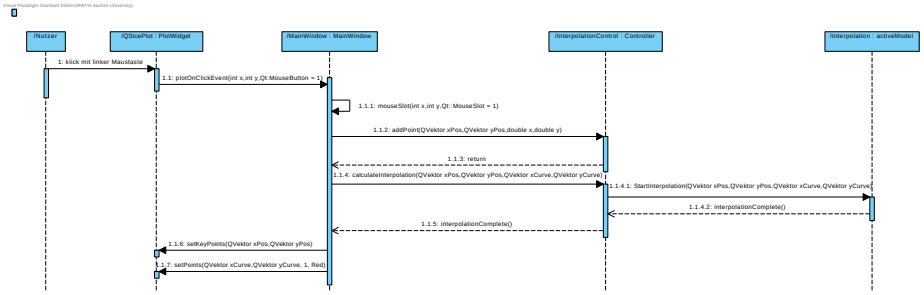

Abbildung 3.2: Sequenzdiagramm: Stuetzstelle hinzufuegen

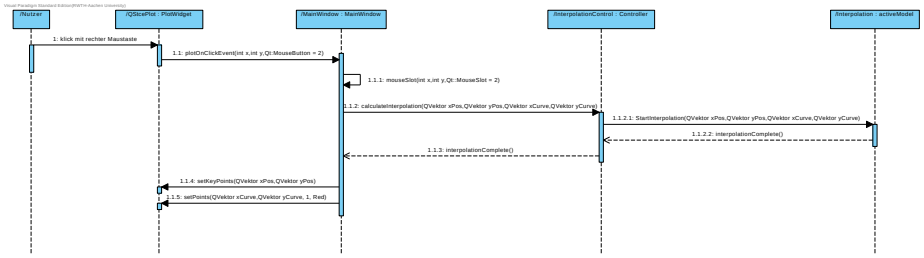

Abbildung 3.3: Sequenzdiagramm: Stuetzstelle loeschen

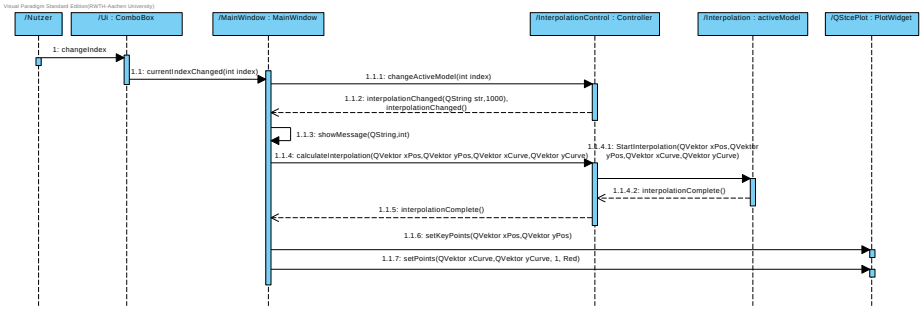

Abbildung 3.4: Sequenzdiagramm: Interpolationsart aendern

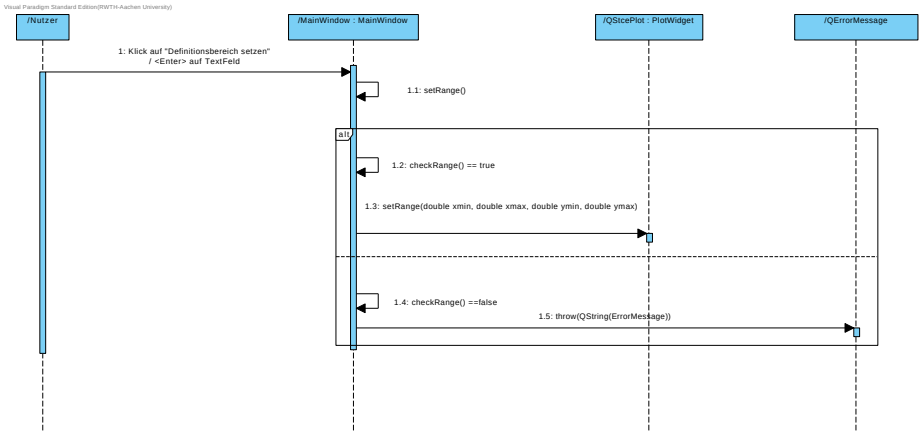

Abbildung 3.5: Sequenzdiagramm: Definitionsbereich aendern

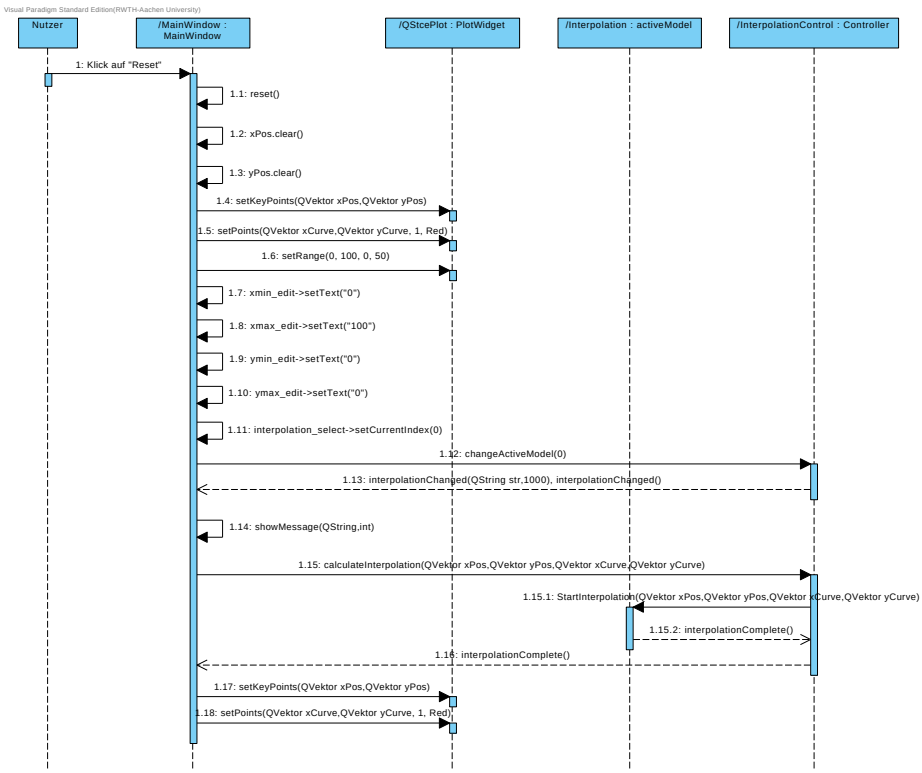

Abbildung 3.6: Sequenzdiagramm: Reset

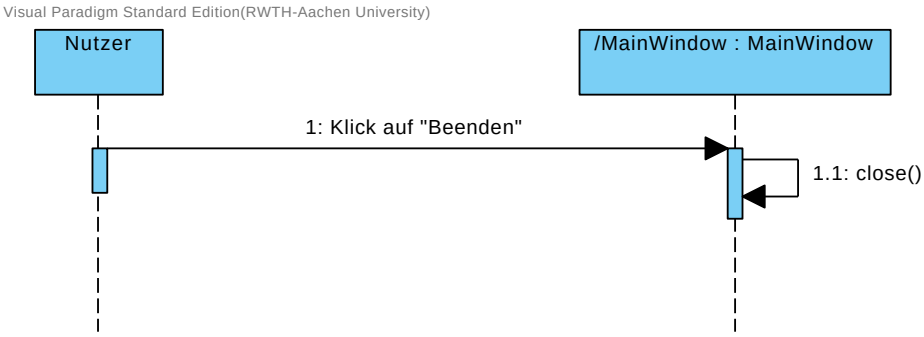

Abbildung 3.7: Sequenzdiagramm: Beenden

Kapitel 4

Benutzerdokumentation

4.1 Installation

4.1.1 Voraussetzungen

Für die Installation der Software ist ein Linux basiertes Betriebssystem, sowie die Vorinstallation von QT (Version 5) notwendig.

4.1.2 Unkompilierte Version

Für die Installation der unkompilierten Version sind folgende Schritte zu befolgen:

1. Erstellen eines gewünschten Installationsverzeichnisses, z.B. `./Interpolation`
2. Speichern von `Interpolation.zip` in dem gewählten Verzeichnis.
3. Entpacken des Archivs mittels `unzip Interpolation.zip` oder einer entsprechenden Software.
4. Ausführen von `qmake-qt5` in dem gewählten Verzeichnis.
5. Anschließend ausführen von `make` in dem gleichen Verzeichnis.
6. 'make clean' stellt den Ausgangszustand des Programms wieder her, erhält aber die ausführbare Datei.
7. Ausführen des Programms durch das Starten der ausführbaren Datei 'Interpolation'.

4.2 Beispielsitzung

Das Programm startet mit der Anzeige des Hauptfensters.

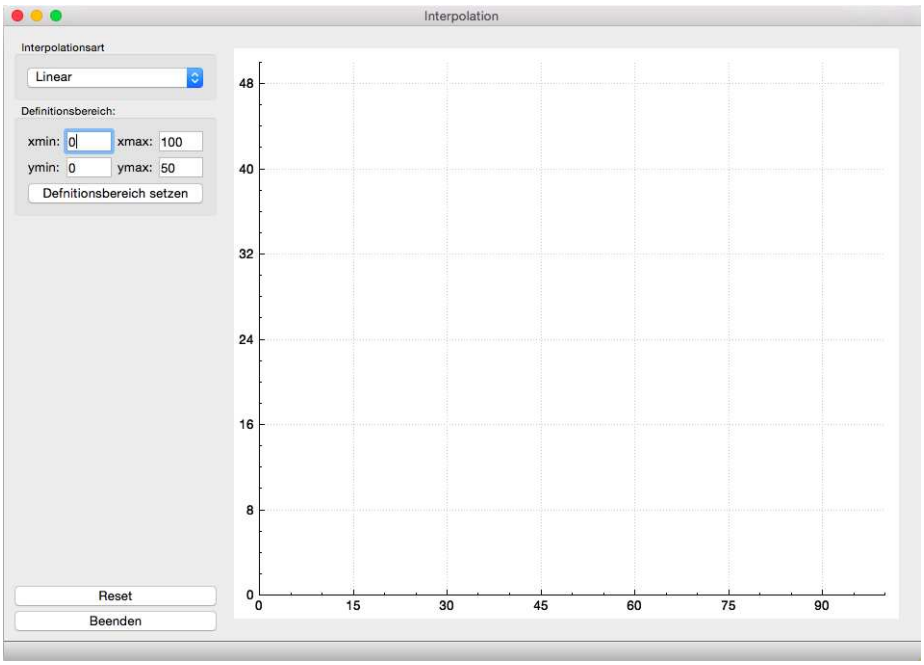

Abbildung 4.1: Startbildschirm

Der Nutzer klickt auf die Zeichenfläche mehrmals um Punkte hinzuzufügen. Diese werden auf der Zeichenfläche direkt angezeigt und durch gerade Linien (lineare Splines) verbunden.

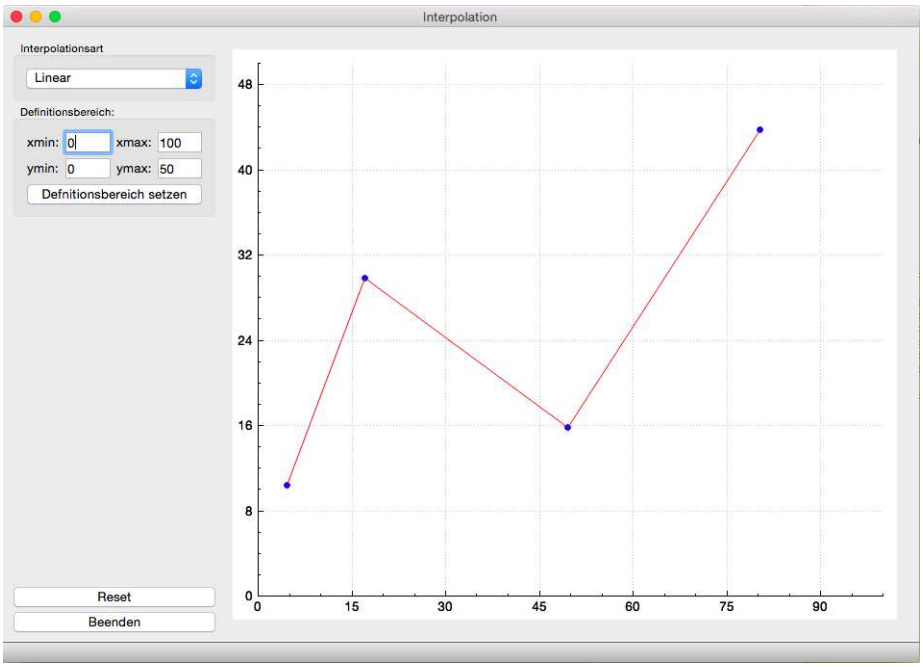

Abbildung 4.2: Lineare Interpolation zwischen Punkten

Der Nutzer wählt die Interpolationsart 'Polynominterpolation' aus. Die Punkte werden nun durch eine Polynomkurve miteinander verbunden.

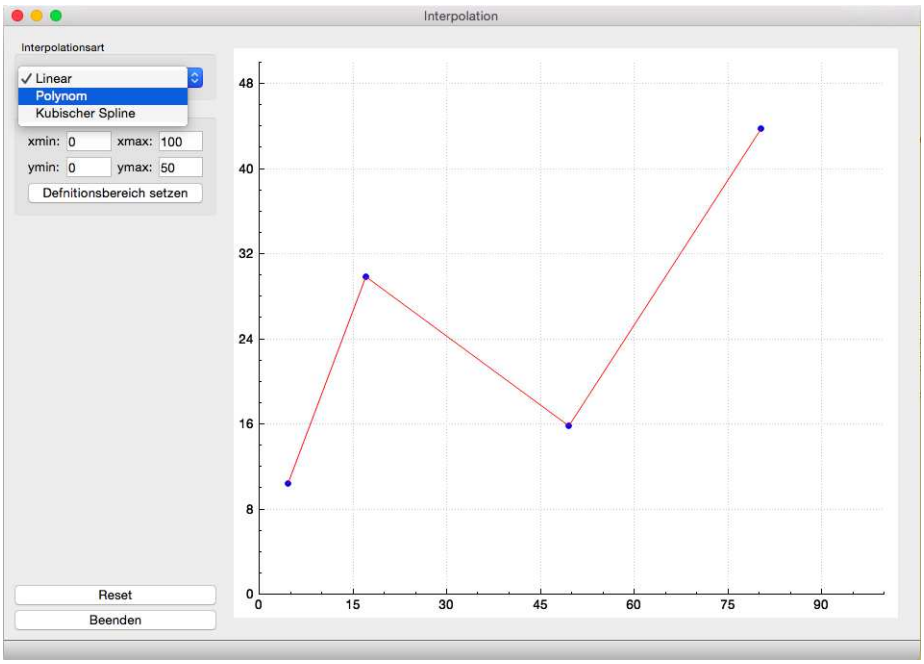

Abbildung 4.3: Ändere Interpolationsart

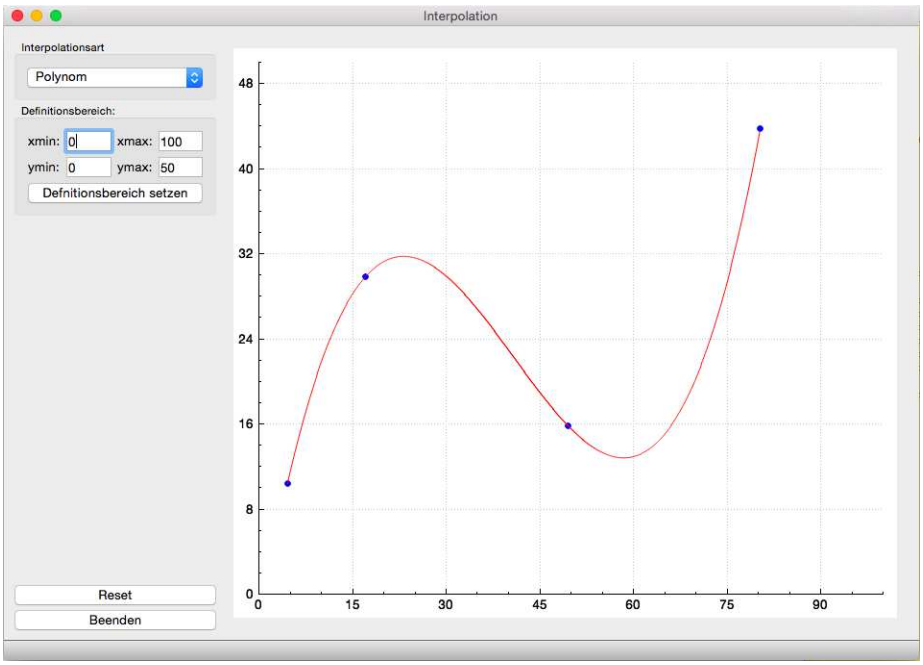

Abbildung 4.4: Polynominterpolation zwischen Punkten

Der Nutzer verändert den Definitionsbereich durch Eingeben der Grenzen in die entsprechenden Textfelder und Klicken auf die Schaltfläche 'Definitionsbereich setzen' . Sind die Grenzen jedoch nicht konform, d.h. ist xmin z.B. größer als xmax, so wird eine Fehlermeldung ausgegeben.

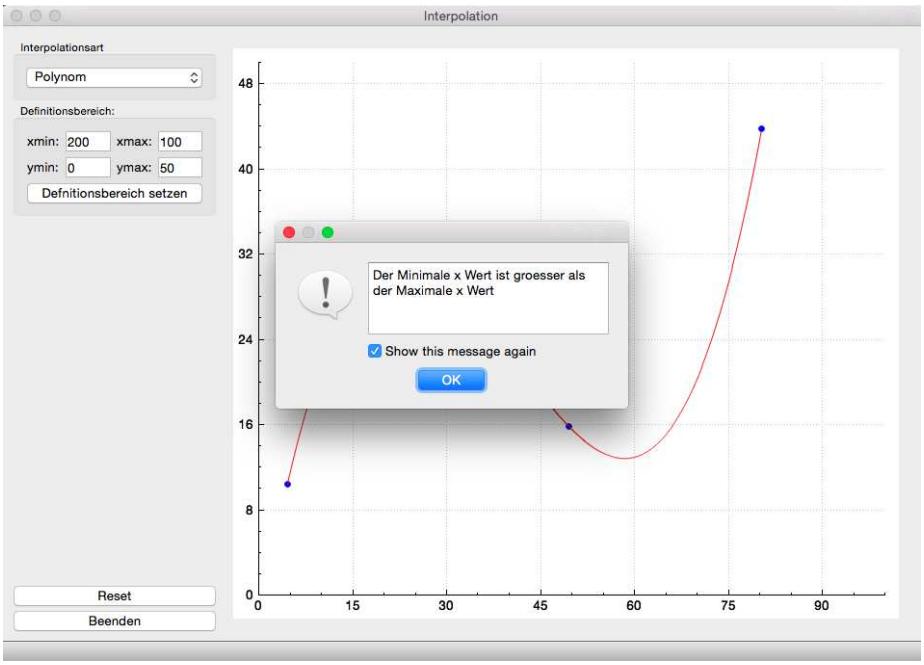

Abbildung 4.5: Anzeige einer Fehlermeldung

Der Nutzer klickt auf die Schaltfläche 'Beenden' und das Programm schließt.

4.3 Fehlersituationen

Das Programm gibt Fehlermeldungen aus. Diese können nur entstehen, wenn der Definitionsbereich verändert wird. Es wird eine Fehlermeldung ausgegeben, wenn ein Feld leer ist, einen Buchstaben enthält, die Grenzen außerhalb eines vordefinierten Intervalls oder die Anzahl der Nachkommastellen zu groß ist. Es sind alle anderen Ausnahmen und mögliche Fehlersituationen so behandelt, dass diese nicht angezeigt werden müssen und sich der Benutzer komplett mit der Bedienung des Programms beschäftigen kann. Sollten dennoch Fehler auftreten bitten wir diese an stefan.jeske@rwth-aachen.de zu melden.

Kapitel 5

Entwicklerdokumentation

5.1 Codestruktur

Der Code besteht aus mehreren Klassen. Die Quelldateien befinden sich alle im gleichen Verzeichnis. Siehe Kapitel 4 für Installation. Im Folgenden wird die Codestruktur im Groben beschrieben. Für eine detaillierte Dokumentation siehe Abschnitt 2 in diesem Kapitel.

5.1.1 Klasse MainWindow

Die Klasse `MainWindow` ist für die graphische Darstellung der Benutzeroberfläche zuständig. Der Quellcode findet sich in Sektion A.1.1 und in den Dateien `mainwindow.h` und `mainwindow.cpp`.

5.1.2 Klasse InterpolationControl

Die Klasse `InterpolationControl` verwaltet die zur verfügung stehenden Interpolationsarten. Der Quellcode befindet sich in Sektion A.1.2 und in den Dateien `interpolationcontrol.h` und `interpolationcontrol.cpp`

5.1.3 Klasse Interpolation

Diese Klasse stellt eine abstrakte Klasse zur Verfügung, welche abgeleitet wird um die einzelnen Interpolationsmethoden zu implementieren. Die Standardmäßig verfügbaren Methoden sind lineare Splines, kubische Splines und Polynominterpolation (Grad m-1, wobei m die Anzahl der Stützstellen ist.) Der Quellcode befindet sich jeweils im Anhang Sektion A.1.3 und in den Dateien `interpolation.h` und `interpolation.h`.
Der Quellcode der implementierten Verfahren befindet sich in Sektion A.1.4 bis Sektion A.1.6

5.2 Detaillierte Dokumentation des Codes

In dem Unterverzeichnis `./Interpolation/doc/` befindet sich die Konfigurationsdatei `Doxyfile` für die Erstellung der doxygen Dokumentation in `HTML` oder LaTeX Format. Zum Erstellen der Dokumentation muss der Befehl `doxygen Doxyfile` in dem Verzeichnis `./Interpolation/doc/` aufgerufen werden. Es werden zwei neue Verzeichnisse `./Interpolation/doc/html/` und `./Interpolation/doc/latex/` erstellt. Um die `HTML` Version anzuzeigen muss in den Ordner `html` gewechselt werden und die Datei `index.html` mit dem Standardbrowser geöffnet werden. Um die LaTeX Dokumentation anzeigen zu lassen muss der Befehl `make` in dem Ordner `latex` ausgeführt werden. Anschließend kann die PDF Datei mit einem entsprechenden Programm gelesen werden.

5.3 Hinzufügen einer neuen Interpolationsart

Das Hinzufügen einer neuen Interpolationsart lässt sich in die folgenden Schritte unterteilen.

5.3.1 Ableiten der Klasse Interpolation

Zunächst muss, um eine neue Interpolationsart zu implementieren, die Klasse Interpolation in der folgenden Form abgeleitet werden.

```
1 #include <interpolation.h>
2
3 class myInterpolation : public Interpolation
4 {
5 public:
6         void StartInterpolation(
7             const QVector<double>& xPos,
8             const QVector<double>& yPos,
9             QVector<double>& xCurve,
10            QVector<double>& yCurve
11            );
12 };
```

Die eigentliche Interpolation wird durch die Schnittstelle `StartInterpolation(...)` definiert, welche in den Variablen `xCurve, yCurve` die zu `xPos,yPos` gehörenden interpolierten Punkte speichert, die zum Zeichnen der Kurve verwendet werden. Diese Methode muss von dem Benutzer in folgender Form implementiert werden:

```
1
2 void myInterpolation::StartInterpolation(
3             const QVector<double>& xPos,
4             const QVector<double>& yPos,
5             QVector<double>& xCurve,
6             QVector<double>& yCurve
7             )
8 {
```

```
9           ...
10          <myImplementation>
11          ...
12 }
```

5.3.2 Hinzufügen der Interpolationsart in den Quellcode: InterpolationControl

Angenommen, die neue Interpolationsart wurde in den Dateien `myinterpolation.h` und `myinterpolation.cpp` gespeichert, muss zunächst die folgende Zeile zu den `includes` der `interpolationcontrol.h` und ein neues Objekt in den `private:` Teil der Klasse hinzugefügt werden.

```
1 ...
2 #include "myinterpolation.h"
3 ...
```

```
1 Interpolation * myInterpolationModel;
```

Danach muss in der Datei `interpolationcontrol.cpp` in dem Konstruktor ein Objekt vom Typ `myInterpolation` erzeugt und in der Funktion `changeActiveModel(...)` ein weiterer switch-case hinzugefügt werden:

```
1  InterpolationControl::InterpolationControl()
2  {
3          ...
4          myInterpolationModel = new myInterpolation();
5          ...
6  }
7
8  ...
9
10 void InterpolationControl::changeActiveModel(int _activeModel)
11 {
12         ...
13         switch(_activeModel)
14         {
15                 ...
16                 case 3 : activeModel = myInterpolationModel;
17                          str = "myInterpolation ausgewaehlt";
18                          break;
19         }
20 }
```

5.3.3 Hinzufügen der Interpolationsart in den Quellcode: MainWindow

Zuletzt muss noch eine Zeile am Ende des Konstruktors der Klasse `MainWindow` eingefügt werden.

```
1 MainWindow::MainWindow(QWidget *parent) :
2     QMainWindow(parent),
3     ui(new Ui::MainWindow)
4 {
5         ...
6         ui->interpolation_select->addItem("myInterpolation");
7 }
```

Nun befindet sich ein weiterer Eintrag `myInterpolation` in der Auswahlbox auf der Benutzeroberfläche und kann ausgewählt werden, um die Punkte auf der Zeichenfläche mit `myInterpolation` zu verbinden.

5.4 Software-Tests

Alle Anwendungsfälle wurden manuell auf Funktionalität getestet. Die folgenden Anwendungsfälle wurden erfolgreich ohne Fehlermeldungen getestet:

- Interpolationsart ändern
- Reset
- Beenden
- Stützstelle löschen
- Stützstelle hinzufügen

Der Anwendungsfall 'Definitionsbereich ändern' wurde erfolgreich mit allen Fehlersituationen getestet, die in der Benutzerdokumentation ausführlich beschrieben sind.

Anhang A

Quellcode

A.1 Interpolation

A.1.1 Klasse MainWindow

Listing A.1: mainwindow.h

```
1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
4 #include <QMainWindow>
5 #include <cmath>
6 #include <QDebug>
7 #include <QDoubleValidator>
8 #include <QErrorMessage>
9 #include "interpolationcontrol.h"
10 #include "qStcePlot/qstceplot.h"
11
12 namespace Ui {
13 class MainWindow;
14 }
15
16 /*!
17  * \brief Die MainWindow Klasse
18  *
19  * Die MainWindow Klasse enthaelt alle Ui Elemente sowie die
      Funktionen und Variablen welche fuer das Zeichnen in das
      Plotfenster
20  * notwendig sind.
21  */
22
23 class MainWindow : public QMainWindow
24 {
25     Q_OBJECT
```

```
26
27 public:
28      //! MainWindow Ist der Standardkonstruktor fuer die
            Klasse MainWindow. Es werden die connect-Befehle
            definiert, das Ui und ein InterpolationControl Objekt
             erzeugt.
29     explicit MainWindow(QWidget *parent = 0);
30     ~MainWindow();
31
32 private:
33     /*!
34      * \brief xPos Definiert die KeyPoints auf der x-Achse
35      * \brief yPos Definiert die KeyPoints auf der y-Achse
36      * \brief xCurve Definiert die Punkte welche die x-
            KeyPoints verbinden
37      * \brief yCurve Definiert die Punkte welche die y-
            KeyPoints verbinden.
38      */
39     QVector<double> xPos,yPos,xCurve,yCurve;
40
41     //! Validator um die Definitionsbereich Grenzen zu pruefen
42     QDoubleValidator* checkValid;
43
44     //! Error Message um Fehler anzuzeigen
45     QErrorMessage* errorMessage;
46
47     //! ui Enthaelt alle ui Elemente
48     Ui::MainWindow *ui;
49
50     //! controller verwaltet die Interpolationsarten
51     InterpolationControl* controller;
52
53
54     //!checkRange ueberprueft die Eingaben in die Textfelder,
           zum Setzen des Definitionsbereichs
55     void checkRange() throw(QString);
56
57 public slots:
58      //! mouseSlot definiert das Verhalten bei einem Klick auf
            das PlotWidget.
59     void mouseSlot(
60             double x,
61             double y,
62             Qt::MouseButton btn
63             );
64
65     //! setRange veraendert die Grenzen des plotWidgets
66     void setRange();
67
```

```
68      //! reset setzt alle im Verlauf der Verwendung des
            Programms veraenderten Einstellungen auf die
            Standardeinstellungen zurueck.
69      void reset();
70
71      //! plot zeichnet xPos, yPos als KeyPoints, und xCurve,
            yCurve als Points (also Kurve) in den Plot
72      void plot();
73
74      //! refreshCurve berechnet den Plot komplett neu, z.B.
            wenn die Interpolationsart veraendert wird.
75      void refreshCurve();
76
77  signals:
78      //! keyPointsChanged signalisiert wenn ein Punkt
            hinzugefuegt oder geloescht wurde.
79      void keyPointsChanged(
80              const QVector<double>&,
81              const QVector<double>&,
82              QVector<double>&,
83              QVector<double>&
84              );
85
86  };
87
88  #endif // MAINWINDOW_H
```

Listing A.2: mainwindow.cpp

```
1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3
4  /*!
5   * \param parent Uebergibt den Besitzer von MainWindow.
6   */
7  MainWindow::MainWindow(QWidget *parent) :
8      QMainWindow(parent),
9      ui(new Ui::MainWindow)
10 {
11     checkValid = new QDoubleValidator(-10000000,10000000,3,
           NULL);
12     errorMessage = new QErrorMessage(this);
13
14     ui->setupUi(this);
15     controller = new InterpolationControl();
16     QObject::connect(ui->xmin_edit,SIGNAL(returnPressed()),
           this,SLOT(setRange()));
17     QObject::connect(ui->xmax_edit,SIGNAL(returnPressed()),
           this,SLOT(setRange()));
```

```
18      QObject::connect(ui->ymin_edit,SIGNAL(returnPressed()),
            this,SLOT(setRange()));
19      QObject::connect(ui->ymax_edit,SIGNAL(returnPressed()),
            this,SLOT(setRange()));
20
21      QObject::connect(ui->plotWidget,SIGNAL(plotOnClickEvent(
            double,double,Qt::MouseButton)),this,SLOT(mouseSlot(
            double,double,Qt::MouseButton)));
22      QObject::connect(ui->defbereich_button,SIGNAL(clicked(bool
            )),this,SLOT(setRange()));
23      QObject::connect(ui->reset_button,SIGNAL(clicked(bool)),
            this,SLOT(reset()));
24      QObject::connect(ui->exit_button,SIGNAL(clicked(bool)),
            this,SLOT(close()));
25      QObject::connect(ui->interpolation_select,SIGNAL(
            currentIndexChanged(int)),controller,SLOT(
            changeActiveModel(int)));
26      QObject::connect(controller,SIGNAL(interpolationChanged(
            QString,int)),ui->statusBar,SLOT(showMessage(QString,
            int)));
27      QObject::connect(controller,SIGNAL(interpolationChanged())
            ,this,SLOT(refreshCurve()));
28      QObject::connect(this,SIGNAL(keyPointsChanged(QVector<
            double>,QVector<double>,QVector<double>&,QVector<
            double>&)),controller,SLOT(calculateInterpolation(
            QVector<double>,QVector<double>,QVector<double>&,
            QVector<double>&)));
29      QObject::connect(controller,SIGNAL(interpolationComplete()
            ),this,SLOT(plot()));
30  }
31
32  MainWindow::~MainWindow()
33  {
34      delete ui;
35      delete checkValid;
36      delete errorMessage;
37  }
38
39  /*!
40   * \param x uebergibt die x-Koordinate des Mausklicks.
41   * \param y uebergibt die y-Koordinate des Mausklicks.
42   * \param btn uebergibt einen Index, welcher verwendeten
         MouseButton identifiziert.
43   */
44  void MainWindow::mouseSlot(double x, double y, Qt::MouseButton
         btn){
45
46      if(btn == 1){
47          QString str = "Click at (" + QString().setNum(x,'f',2)
                + " | " + QString().setNum(y,'f',2) + ") with
```

```
            Button " + QString().setNum(btn) +".";
48          this->statusBar()->showMessage(str,1000);
49
50          controller->addPoint(xPos,yPos,x,y);
51      }
52      if(btn ==2){
53          QString str = "Click at (" + QString().setNum(x,'f',2)
                + " | " + QString().setNum(y,'f',2) + ") with
                Button " + QString().setNum(btn) +".";
54          this->statusBar()->showMessage(str,1000);
55
56          double xmax = ui->xmax_edit->text().toDouble();
57          double xmin = ui->xmin_edit->text().toDouble();
58          double ymax = ui->ymax_edit->text().toDouble();
59          double ymin = ui->ymin_edit->text().toDouble();
60
61          double epsx = (xmax-xmin)*0.01;
62          double epsy = (ymax-ymin)*0.01;
63          double tmpx;
64          double tmpy;
65          QVector<double>::iterator xiter = xPos.begin();
66          QVector<double>::iterator yiter = yPos.begin();
67          for(;xiter!=xPos.end();)
68          {
69              tmpx = sqrt(pow(*xiter-x,2));
70              tmpy = sqrt(pow(*yiter-y,2));
71              if(tmpx < epsx && tmpy < epsy)
72              {
73                  xPos.erase(xiter);
74                  yPos.erase(yiter);
75                  break;
76              }
77
78              xiter++;
79              yiter++;
80          }
81      }
82      emit keyPointsChanged(xPos,yPos,xCurve,yCurve);
83  }
84
85  void MainWindow::setRange()
86  {
87      try
88      {
89          checkRange();
90      }
91      catch(const QString error)
92      {
93          errorMessage->showMessage(error);
94          return;
```

```
95      }
96
97      //Set Range
98      QString xmaxstr = ui->xmax_edit->text();
99      QString xminstr = ui->xmin_edit->text();
100     QString ymaxstr = ui->ymax_edit->text();
101     QString yminstr = ui->ymin_edit->text();
102
103     double xmax = xmaxstr.toDouble();
104     double xmin = xminstr.toDouble();
105     double ymax = ymaxstr.toDouble();
106     double ymin = yminstr.toDouble();
107
108     ui->plotWidget->setRange(xmin,xmax,ymin,ymax);
109 }
110
111 void MainWindow::reset()
112 {
113     xPos.clear();
114     yPos.clear();
115     ui->plotWidget->setKeyPoints(xPos,yPos);
116     ui->plotWidget->setPoints(xPos,yPos,1,"Red");
117     ui->plotWidget->setRange(0,100,0,50);
118     ui->xmin_edit->setText("0");
119     ui->xmax_edit->setText("100");
120     ui->ymin_edit->setText("0");
121     ui->ymax_edit->setText("50");
122     ui->interpolation_select->setCurrentIndex(0);
123     controller->changeActiveModel(0);
124     delete errorMessage;
125     errorMessage = new QErrorMessage(this);
126 }
127
128 void MainWindow::plot()
129 {
130     ui->plotWidget->setKeyPoints(xPos,yPos);
131     ui->plotWidget->setPoints(xCurve,yCurve,1,"Red");
132 }
133
134 void MainWindow::refreshCurve()
135 {
136     emit keyPointsChanged(xPos,yPos,xCurve,yCurve);
137 }
138
139 void MainWindow::checkRange() throw(QString)
140 {
141     int pos;
142     QString xmaxstr = ui->xmax_edit->text();
143     QString xminstr = ui->xmin_edit->text();
144     QString ymaxstr = ui->ymax_edit->text();
```

```
145     QString yminstr = ui->ymin_edit->text();
146     pos = ui->xmin_edit->cursorPosition();
147     if(checkValid->validate(xminstr,pos) < 2)
148     {
149         throw(QString("Ungueltige Eingabe in xmin Textfeld"));
150     }
151
152     pos = ui->xmax_edit->cursorPosition();
153     if(checkValid->validate(xmaxstr,pos) < 2)
154     {
155         throw(QString("Ungueltige Eingabe in xmax Textfeld"));
156     }
157
158     pos = ui->ymin_edit->cursorPosition();
159     if(checkValid->validate(yminstr,pos) < 2)
160     {
161         throw(QString("Ungueltige Eingabe in ymin Textfeld"));
162     }
163
164     pos = ui->ymax_edit->cursorPosition();
165     if(checkValid->validate(ymaxstr,pos) < 2)
166     {
167         throw(QString("Ungueltige Eingabe in ymax Textfeld"));
168     }
169
170     double xmax = xmaxstr.toDouble();
171     double xmin = xminstr.toDouble();
172     double ymax = ymaxstr.toDouble();
173     double ymin = yminstr.toDouble();
174
175     if (xmin > xmax)
176     {
177         throw(QString("Der Minimale x Wert ist groesser als
                der Maximale x Wert"));
178     }
179     if (ymin > ymax)
180     {
181         throw(QString("Der Minimale y Wert ist groesser als
                der Maximale y Wert"));
182     }
183 }
```

A.1.2 Klasse InterpolationControl

Listing A.3: interpolationcontrol.h

```
1 #ifndef INTERPOLATIONCONTROL_H
2 #define INTERPOLATIONCONTROL_H
3 #include <QVector>
4 #include <QWidget>
```

```
5 #include "interpolation.h"
6 #include "cubicspline.h"
7 #include "linearspline.h"
8 #include "polynomialinterpolation.h"
9
10     /*!
11     * \brief Die InterpolationControl Klasse
12     *
13     * Die InterpolationControl Klasse regelt den Wechsel
           zwischen den einzelnen Interpolationsmethoden und kann
14     * die Kurve mit der jeweils ausgewaehlten
           Interpolationsmethode starten.
15     */
16 class InterpolationControl : public QWidget
17 {
18     Q_OBJECT
19 public:
20          /*!
21          * \brief InterpolationControl Ist der
                Standardkonstruktor.
22          */
23         InterpolationControl();
24
25         /*!
26          * \brief addPoint F gt den Punkt (x,y) nach der
                Groe e sortiert in den xPos und yPos QVektor
27          */
28         void addPoint(
29                 QVector<double>& xPos,
30                 QVector<double>& yPos,
31                 double x,
32                 double y
33                 );
34 private:
35         /*!
36          * \brief xpos Enthaelt die x Werte von den
                bestehenden Knoten
37          * \brief ypos Enthaelt die zu den x-Werte die
                dazugehoerigen y-Werte
38          */
39         std::vector <double> xpos,ypos;
40         /*!
41          * \brief activeModel Zeiger auf die aktuelle
                Interpolationsmethode
42          */
43         Interpolation *activeModel;
44         /*!
45          * \brief LinearSplineModel Zeiger auf die lineare
                Spline Interpolation
46          */
```

```
47          Interpolation *LinearSplineModel;
48          /*!
49           * \brief CubicSplineModel Zeiger f r die Kubische
                 Spline Interpolations
50           */
51          Interpolation *CubicSplineModel;
52          /*!
53           * \brief PolynomialModel Zeiger auf die
                 Lagrangeinterpolationsmethode
54           */
55          Interpolation *PolynomialModel;
56  public slots:
57          /*!
58           * \brief changeActiveModel kann man zwischen den
                 einzelnen Interpolationsmethoden herumwechseln
59           */
60          void changeActiveModel(int);
61          /*!
62           * \brief calculateInterpolation ruft in der Klasse
                 Interpolation die Funktion StartInterpolation()
                 auf
63           */
64          void calculateInterpolation(
65                  const QVector<double>&,
66                  const QVector<double>&,
67                  QVector<double> &,
68                  QVector<double> &
69                  );
70  signals:
71          /*!
72           * \brief interpolationChanged Signalisiert durch eine
                  Nachricht, wenn der User die
                 Interpolationsmethode aendert
73           */
74          void interpolationChanged(
75                  QString,
76                  int
77                  );
78          /*!
79           * \brief interpolationComplete Signalisiert, wenn die
                  Berechnung der Stuetzstellen fertig ist
80           */
81          void interpolationComplete();
82          /*!
83           * \brief interpolationChanged Signalisiert dem
                 Programm die Kurve neu zuplotten
84           */
85          void interpolationChanged();
86  };
87
```

```
88 #endif // INTERPOLATIONCONTROL_H
```

Listing A.4: interpolationcontrol.cpp

```
1 #include "interpolationcontrol.h"
2 #include <QString>
3
4 /*!
5  * Es werden beim Initialisieren Objekte
6  * der verschiedenen Interpolationsmethoden, sowie ein
       activeModel, welches auf die aktuelle
7  * Interpolationsmethode zeigt, erzeugt. Dieses ist beim
       Initialisieren die lineare Spline Interpolation.
8  */
9 InterpolationControl::InterpolationControl()
10 {
11     CubicSplineModel = new CubicSpline();
12     LinearSplineModel = new LinearSpline();
13     PolynomialModel = new PolynomialInterpolation();
14     activeModel = LinearSplineModel;
15 }
16
17 void InterpolationControl::changeActiveModel(int _activeModel)
18 {
19     QString str;
20     switch(_activeModel)
21     {
22         case 0 : activeModel = LinearSplineModel;
23             str = "Lineare Interpolation ausgewaehlt";
24             break;
25
26         case 1 : activeModel = PolynomialModel;
27             str = "Polynominterpolation ausgewaehlt";
28             break;
29
30         case 2 : activeModel = CubicSplineModel;
31             str = "Kubische Spline Interpolation ausgewaehlt";
32             break;
33     }
34
35     emit interpolationChanged(str,1000);
36     emit interpolationChanged();
37 }
38
39 void InterpolationControl::calculateInterpolation(const
      QVector<double> &xPos, const QVector<double> &yPos,QVector
      <double>&xCurve, QVector<double>&yCurve)
40 {
41     activeModel->StartInterpolation(xPos, yPos, xCurve, yCurve
          );
```

```
42      emit interpolationComplete();
43  }
44
45  /*!
46   * \param xPos Enthaelt die x Werte von den bestehenden Knoten
47   * \param yPos Enthaelt die zu den x-Werte die dazugehoerigen
         y-Werte
48   * \param x x-Koordinate des neuen Punktes
49   * \param y y-Koordinate des neuen Punktes
50   */
51  void InterpolationControl::addPoint(QVector<double>&xPos,
        QVector<double>&yPos, double x, double y)
52  {
53      for(QVector<double>::iterator xit = xPos.begin(), yit =
            yPos.begin(); xit != xPos.end(); xit++, yit++)
54      {
55          if(*xit == x)
56          {
57              return;
58          }
59
60          if(*xit > x)
61          {
62              xPos.insert(xit,x);
63              yPos.insert(yit,y);
64              return;
65          }
66      }
67
68      xPos.push_back(x);
69      yPos.push_back(y);
70      return;
71
72  }
```

A.1.3 Klasse Interpolation

Listing A.5: interpolation.h

```
1  #ifndef INTERPOLATION_H
2  #define INTERPOLATION_H
3  #include <QVector>
4  /*!
5   * \brief Die Interpolation Klasse
6   *
7   * Die Klasse Interpolation ist die abstrakte Klasse fuer die
        verschiedenen Interpolationsmethoden
8   */
9  class Interpolation
10 {
```

```
11 public:
12     /*!
13      * \brief Interpolation Standardkonstruktor
14      */
15     Interpolation();
16     /*!
17      * \brief StartInterpolation Virtuelle Methode, welche die
            Berechnung der Stuetzstellen startet
18      */
19     virtual void StartInterpolation(const QVector<double>&,
           const QVector<double>&, QVector<double>&, QVector<
           double>&)=0;
20 };
21
22 #endif // INTERPOLATION_H
```

Listing A.6: interpolation.cpp

```
1 #include "interpolation.h"
2
3 Interpolation::Interpolation()
4 {
5
6 }
```

A.1.4 Klasse CubicSpline

Listing A.7: cubicspline.h

```
1 #ifndef CUBICSPLINE_H
2 #define CUBICSPLINE_H
3 #include <interpolation.h>
4 #include "spline.h"
5 #include <vector>
6
7 /*!
8  * \brief Die CubicSpline Klasse erbt von der Klasse
       Interpolation und implementiert die Funktionitaet
       zwischen Punkten mithilfe
9  * Kubischer Splines zu Interpolieren.
10 *
11 * Die Klasse verwenden ausserdem die Bibliothek spline.h der
       Uni Chemnitz
12 */
13 class CubicSpline : public Interpolation
14 {
15 private:
16     /*!
17      * \brief x Enthaelt lokale Kopie des xPos QVector als std
           ::vector
```

```
18      * \brief y Enthaelt lokale Kopie des yPos QVector als std
            ::vector
19      */
20     std::vector<double> x,y;
21 public:
22     /*!
23      * \brief Standardkonstruktor fuer die Klasse CubicSpline
24      */
25     CubicSpline();
26
27     /*!
28      * \brief StartInterpolation interpoliert die Punkte xPos
            und yPos mithilfe Kubischer Splines und speichert die
             berechnete Kurve in xCurve
29      * und yCurve
30      */
31     void StartInterpolation(
32             const QVector<double>& xPos,
33             const QVector<double>& yPos,
34             QVector<double>& xCurve,
35             QVector<double>& yCurve
36             );
37
38     /*!
39      * \brief qVectorToVector erstellt in lokalen Variablen x
            und y Kopien der QVektoren xPos und yPos
40      */
41     void qVectorToVector(
42             const QVector<double>& xPos,
43             const QVector<double>& yPos
44             );
45 };
46
47 #endif // CUBICSPLINE_H
```

Listing A.8: cubicspline.cpp

```
1 #include "cubicspline.h"
2
3 CubicSpline::CubicSpline()
4 {
5
6 }
7
8 /*!
9  * \param xPos enthaelt die x Stuetzstellen
10  * \param yPos enthaelt die y Stuetzstellen
11  * \param xCurve speichert die interpolierten x Punkte
12  * \param yCurve speichert die interpolierten y Punkte
13  */
```

```
14 void CubicSpline::StartInterpolation(const QVector<double> &
       xPos, const QVector<double> &yPos, QVector<double> &xCurve
       , QVector<double> &yCurve)
15 {
16     xCurve.clear();
17     yCurve.clear();
18     if(xPos.size()>2)
19     {
20         int n = 1000;
21         xCurve.resize(n);
22         yCurve.resize(n);
23         double xmin = *std::min_element(xPos.begin(),xPos.end
               ());
24         double xmax = *std::max_element(xPos.begin(),xPos.end
               ());
25         double h = (xmax-xmin)/n;
26
27         qVectorToVector(xPos,yPos);
28         tk::spline s;
29         s.set_points(x,y,true);
30
31
32         for(int i=0;i<n;i++)
33         {
34             xCurve[i] = xmin + i*h;
35             yCurve[i] = s(xCurve[i]);
36         }
37     }
38 }
39
40 /*!
41  * \param xPos enthaelt die zu interpolierenden x Punkte
42  * \param yPos enthaelt die zu interpolierenden y Punkte
43  */
44 void CubicSpline::qVectorToVector(const QVector<double>& xPos,
       const QVector<double>& yPos)
45 {
46     x.clear();
47     y.clear();
48     int n = xPos.size();
49     x.resize(n);
50     y.resize(n);
51     for(int i=0; i<n; i++)
52     {
53         x[i] = xPos[i];
54         y[i] = yPos[i];
55     }
56
57 }
```

A.1.5 Klasse PolynomialInterpolation

Listing A.9: polynomialinterpolation.h

```
1 #ifndef POLYNOMIALINTERPOLATION_H
2 #define POLYNOMIALINTERPOLATION_H
3 #include <interpolation.h>
4
5 /*!
6  * \brief Die Klasse PolynomialInterpolation Interpoliert
      zwischen gegebenen Punkten x und y mithilfe der Lagrange-
      Multiplikator
7  * Methode.
8  *
9  * Es wird also bei n Punkten ein Polynom vom Grad n-1 zur
      Interpolation verwendet. Die Klasse erbt ausserdem von
      der abstrakten Klasse
10 * Interpolation.
11 */
12 class PolynomialInterpolation : public Interpolation
13 {
14 public:
15     /*!
16      * \brief PolynomialInterpolation ist der
          Standardkonstruktor.
17      */
18     PolynomialInterpolation();
19
20     /*!
21      * \brief StartInterpolation fuehr die Interpolation aus.
22      */
23     void StartInterpolation(const QVector<double>& xPos,const
          QVector<double>& yPos,QVector<double>& xCurve,QVector<
          double>& yCurve);
24 };
25
26 #endif // POLYNOMIALINTERPOLATION_H
```

Listing A.10: polynomialinterpolation.cpp

```
1 #include "polynomialinterpolation.h"
2 #include <algorithm>
3
4 PolynomialInterpolation::PolynomialInterpolation()
5 {
6
7 }
8
9 /*!
10 * Es wird mit den Punkten xPos und yPos interpoliert.
11 *
```

```
12  * \param xPos enthaelt die x Stuetzstellen
13  * \param yPos enthaelt die y Stuetzstellen
14  * \param xCurve speichert die interpolierten x Punkte
15  * \param yCurve speichert die interpolierten y Punkte
16  */
17 void PolynomialInterpolation::StartInterpolation(const QVector
       <double> &xPos, const QVector<double> &yPos, QVector<
       double> &xCurve, QVector<double> &yCurve)
18 {
19     xCurve.clear();
20     yCurve.clear();
21     int n = 1000;
22     xCurve.resize(n);
23     yCurve.resize(n);
24     int k = xPos.size();
25     double xmin = *std::min_element(xPos.begin(),xPos.end());
26     double xmax = *std::max_element(xPos.begin(),xPos.end());
27     //double ymin = *std::min_element(yPos.begin(),yPos.end())
           ;
28     //double ymax = *std::max_element(yPos.begin(),yPos.end())
           ;
29
30     double lj;
31
32     double h = (xmax-xmin)/n;
33
34     for (int i=0; i<n; i++)
35     {
36         xCurve[i] = xmin + i*h;
37         yCurve[i] = 0;
38         for(int j=0; j<k;j++)
39         {
40             lj =1;
41             for(int m=0; m<k;m++)
42             {
43                 if(m==j)
44                 {}
45                 else
46                 {
47                     lj *= (xCurve[i] - xPos[m])/(xPos[j] -
                           xPos[m]);
48                 }
49             }
50             yCurve[i] += yPos[j]*lj;
51         }
52     }
53 }
```

A.1.6 Klasse LinearSpline

Listing A.11: linearspline.h

```
1 #ifndef LINEARSPLINE_H
2 #define LINEARSPLINE_H
3 #include <interpolation.h>
4
5 /*!
6  * \brief Die LinearSpline Klasse
7  *
8  * Die LinearSpline Klasse erbt von der Klasse Interpolation
       und implementiert die Funktionitaet zwischen Punkten
       mithilfe
9  * linearer Splines zu Interpolieren.
10 */
11 class LinearSpline : public Interpolation
12 {
13 public:
14     /*!
15      * \brief LinearSpline Ist der Standardkonstruktor
16      */
17     LinearSpline();
18     /*!
19      * \brief StartInterpolation interpoliert die Punkte xPos
           und yPos mithilfe linearer Splines und speichert die
           berechnete Kurve in xCurve
20      * und yCurve
21      */
22     void StartInterpolation(
23             const QVector<double>& xPos,
24             const QVector<double>& yPos,
25             QVector<double>& xCurve,
26             QVector<double>& yCurve
27             );
28 };
29
30 #endif // LINEARSPLINE_H
```

Listing A.12: linearspline.cpp

```
1 #include "linearspline.h"
2
3 LinearSpline::LinearSpline()
4 {
5
6 }
7
8 /*!
9  * \param xPos enthaelt die zu interpolierenden x
       Stuetzstellen
10 * \param yPos enthaelt die zu den x Stuetzstellen gehoerigen
       interpolierten y Stuetzstellen
```

```
11  * \param xCurve speichert die interpolierten x Punkte
12  * \param yCurve speichert die interpolierten y Punkte
13  */
14 void LinearSpline::StartInterpolation(const QVector<double>&
       xPos, const QVector<double>&yPos, QVector<double> &xCurve,
        QVector<double> &yCurve)
15 {
16     xCurve.clear();
17     yCurve.clear();
18     xCurve = xPos;
19     yCurve = yPos;
20     return;
21 }
```

A.1.7 Main Datei

Listing A.13: main.cpp

```
1 #include "mainwindow.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     MainWindow w;
8     w.show();
9
10    return a.exec();
11 }
```