TF-Editor

Wartungshandbuch

Florian Kleber, 0025124
Informatikpraktikum I

Betreuer: Matej Mlejnek

Inhaltsverzeichnis

1 Einleitung

Dieses Wartungshandbuch soll Entwicklern eine Hilfe beim Warten und zukünftigen Entwickeln des TF-Editors sein. Anschließend werden sämtliche Klassen und deren Funktion erklärt.

Nicht näher eingegangen wird allerdings auf Basisklassen des Visualisation Toolkits und auf Klassen von Qt. Eine ausführliche Dokumentation befindet sich dabei auf den jeweiligen Websites.

VTK:

www.vtk.org

www.vtk.org/doc/nightly/html/

Qt:

www.trolltech.com

www.trolltech.com/developer/doc.html

2 KlassenÜbersicht

Im folgenden werden alle Klassen aufgezählt, die für den TF-Editor erstellt wurden, und welche Funktion sie besitzen. Dies soll lediglich eine Übersicht darstellen, da auf die einzelnen Klassen und deren Methoden im Folgenden noch genauer eingegangen wird.

- Main

  Main-Klasse. Wird beim starten des Programms erzeugt, erzeugt alle Fenster und initialisiert diese.
- NodeItem

  Entspricht einem Kurvenpunkt. Wird benötigt, um später Transferfunctions zu erzeugen.
- EdgeItem

  Entspricht einer Kante zwischen zwei Kurvenpunkten. Wird ebenfalls benötigt um später Transferfunctions zu definieren.
- Curve

  Stellt eine Transferfunction dar.
- FigureEditor

  Basisklasse um ein Diagramm zu erstellen, dass maximal drei Kurven beinhaltet, die editiert werden können.
- TFDesign

  Klasse für das „Hauptfenster“.
- TFStyle1

  Klasse die als „Behälter“ für den Transferfunction-Style1 fungiert.
- TFStyle2

  Klasse die als „Behälter“ für den Transferfunction-Style2 fungiert.
- CreadDate

  Klasse zum Einlesen Dateien. (zwei Dateiformate werden unterstützt)
- MyRenderer

  Klasse die eine Rendering-Pipeline realisiert
- vtkQtRenderWindow

  Wird benötigt um VTK in Qt zu integrieren
- vtkQtRenderWindowInteractor

  Wird benötigt um VTK in Qt zu integrieren
- WindowToImageFilter

Klasse um ein vtkWindow als Input einer Bild-Pipeline zu benutzen.

- GradSphere

  Klasse um ein Fenster mit einer Sphere zu erzeugen

- GradSphereHistogram

  Klasse, die ein Histogram auf eine Sphere mappt (Color Mapping)

- CellPicking

  Klasse, um interaktiv mit der Maus einzelne Zellen einer Sphere auswählen zu können („picken“).

3 Klassenbeschreibungen

Es folgt nun eine detailierte Beschreibung der einzelnen Klassen und ihrer Methoden.

3.1 NodeItem

Um die Transferfunction zu definieren benötigt man Klassen, um diese zu beschreiben. Eine Kurve setzt sich aus einzelnen Punkten und Kanten zusammen. Die Klasse NodeItem beschreibt einen einzelnen Kurvenpunkt.

```
class NodeItem: public QCanvasEllipse
{
public:
    NodeItem( QCanvas *canvas, QBrush *brush, QPen *pen );
    ~NodeItem() {}

    void addInEdge( EdgeItem *edge ) { inList = edge; }
    void addOutEdge( EdgeItem *edge ) { outList = edge; }

    void moveBy(double dx, double dy);

    EdgeItem *inList;
    EdgeItem *outList;
private:
    QBrush *tbb;
    QPen *tpp;
};
```

Die Klasse ist von der Klasse QcanvasEllipse aus der Qt Bibliothek abgeleitet. Im Konstruktor wird ein Punkt mit einem Durchmesser von 6 Pixel erzeugt. Weiters wird der Pen und und der Brush auf die jeweiligen Übergabeparameter gesetzt, und die eingehende/ausgehende Kante auf 0 gesetzt.
Weiters existieren noch Methoden, um eine ausgehende und eine eingehende Kante zu diesem Punkt zu definieren. Um einen Knotenpunkt relativ zu seinem ursprünglichen Ort zu verschieben, existiert die Methode moveby, die sich als Parameter zwei double Werte erwartet, die beschreiben, um welchen Wert der Knoten in x und y Richtung relativ verschoben wird. Eingehende/ausgehende Kanten werden ebenfalls mitverschoben.

3.2 EdgeItem

Um die Transferfunction zu definieren benötigt man Klassen, um diese zu beschreiben. Eine Kurve setzt sich aus einzelnen Punkten und Kanten zusammen. Die Klasse EdgeItem beschreibt eine Kante zwischen zwei Kurvenpunkten.

```
class EdgeItem: public QCanvasLine
{
public:
    EdgeItem( NodeItem*, NodeItem*, QCanvas *canvas, QBrush *brush, QPen *pen);
    void setFromPoint( int x, int y ) ;
    void setToPoint( int x, int y );
    void moveBy(double dx, double dy);

    NodeItem *fromNode;
    NodeItem *toNode;
private:
    QBrush *tbb;
    QPen *tpp;
};
```

Die Klasse ist aus der Klasse QcanvasLine aus der Qt Bibliothek abgeleitet. Der Konstruktor erwartet sich zwei NodeItems: einen Startpunkt und einen Endpunkt der Kante. Bei diesen beiden Punkten werden durch das Erzeugen einer Kante die zugehörigen Informationen gesetzt. Beim Startpunkt wird für die ausgehende Kante ein Verweis auf die gerade erzeugte Instanz dieser Kante gesetzt. Mit dem Endpunkt wird ebenso verfahren.

Die Methoden setFromPoint und setToPoint erlauben es, den Start- und Endpunkt zu setzen und zu verändern. Die Methode moveBy überschreibt die moveBy Methode von QcanvasLine und beinhaltet keinen Code (notwendige Veränderungen der Lage einer Kante wird durch die Methode moveBy der Klasse NodeItem realisiert).

3.3 Curve

Um die Transferfunction zu definieren benötigt man Klassen, um diese zu beschreiben. Eine Kurve setzt sich aus einzelnen Punkten und Kanten zusammen. Die Klasse Curve definiert mit Hilfe der Klassen NodeItem und EdgeItem eine beliebige zusammenhängende Kurve.

```
class Curve
{
   public:
      Curve(QBrush *brush, QPen *pen);
      NodeItem *Insertp(int x, int y, QCanvas *canvas);
      void Removep(NodeItem *node, QCanvas *canvas);
      NodeItem *GetNodeItem(int x);
      NodeItem *GetStartNode(void);
      void setBrush(int r, int g, int b);
      void setPen(int r, int g, int b);
private:
      NodeItem *startPoint;
      QBrush *tbb;
      QPen *tpp;
};
```

Der Konstruktor setzt brush und pen, und erzeugt eine „leere“ Kurve.
Die Methode GetStartNode liefert den ersten Kurvenpunkt zurück. Die Methode GetNodeItem hat als Parameter einen integer-Wert, der der x-Koordinate eines Punktes entspricht, und liefert als Ergebnis jenen Kurvenpunkt, der an dieser Stelle liegt (aufgrund der Einschränkungen bei der Kurvenerstellung/editierung die durch die Klasse FigureEditor gegeben sind, ist es nicht möglich zwei übereinanderliegende Kurvenpunkte zu haben. Wäre dies der Fall, würde die Methode jenen zurück liefern, der bei Durchlauf der Kurve vom Startpunkt aus als erster gefunden werden würde.)
Mittels den Methoden setBrush und setPen ist es möglich die Farbeinstellungen beliebig zu verändern. Übergabeparameter ist ein RGB-wert.
StartPoint speichert den StartPunkt einer Kurve ab.

3.4 CreadData

Diese Klasse ermöglicht das Einlesen von MR-Daten. Zwei unterschiedlich Datenformate werden unterstützt. Eine genaue Beschreibung der Formate ist dem jeweiligen Header bzw. dem Benutzerhandbuch zu entnehmen.

```
class CReadData
{
public:
      CReadData();
      virtual ~CReadData();
```

```
	void Read(QString file);
	short GetDataEl(int x, int y, int z);
	vtkImageData* GetData();
	char* GetCharData();
	int GetXdim();
	int GetYdim();
	int GetZdim();
	int* GetHistData();
	void SetFileName(QString file);
	int CReadData::GetType();


private:
	int m_x;
	int m_z;
	int m_y;
	int m_header;
	int type;   // =1 for *.dat =2 for *.vol
	float m_fx;
	float m_fy;
	float m_fz;
	char m_type[4];
	char *m_cdata;
	vtkImageData *vtk_data;
	vtkUnsignedShortArray *array;
	vtkUnsignedCharArray *array_type2;
	int histogram[4096];
	int drawhist[400];
	short *m_data;
	double max_range;
	QString filename;
};
```

Im Konstruktor werden alle Zeiger auf jeweilige Datenelemente mit NULL initialisiert.
Die Methode SetFileName ermöglicht es den Namen der Datei festzulegen, die gelesen werden soll (Durch den aufruf dieser Methode werden allerdings noch keine Daten gelesen!) Alternativ kann der Name der zu öffnenden Datei

auch der Methode Read übergeben werden.
Die Methode Read liest ein gegebenes MR-File. Wurde mittels SetFileName der Dateiname zuvor nicht festgelegt, oder nicht als Übergabeparameter dieser Methode definiert, so hat diese Methode keine Wirkung. Dateien die zufällig auf *.dat oder *.vol enden, und nicht den Dateiformaten entsprechen, können Fehler verursachen.
Die Dateiformate müssen folgender Formatierung entsprechen:

-) *.dat Dateien. Das Datenformat besteht aus einem 6-Byte großem Header, der die Dimensions in alle Raumrichtungen angibt (x-Dim.; y-Dim.; z-Dim) Dann folgen die eigentlichen Daten. Diese sind belegen für einen Voxel-Wert jeweils 2 Byte. Es werden aber immer nur die ersten 12 bit verwendet.

-) *.vol Dateien. Dieses Dateiformat besteht aus einem Header, dessen Größe 10 000 Bytes beträgt. Die ersten 5 Byte identifizieren das File als ein „Volume-File“ (mdvol). 1 Byte ist ein Versions Indikator. Die nächsten 4 Bytes (1 Integer) geben die Gesamtlänge des Headers an. Danach folgen die Dimensionen in x-, y-, und z-Richtung als drei Integerwerte (je 4 Byte, in Pixel). Es folgen drei Floatwerte, die die physikalischen Dimensionen beschreiben (3 byte, je ein byte für x-, y-, und z-Richtung in [mm]). Drei weitere Charakters spezifizieren die Farbinformation ('g08' 8-bit Grauwerte; 'g16' 16bit Grauwerte; 'c24' 24 bit Farbwerte). Nach dem Ende des Headers folgen die Datenwerte in der Ordnung x/y/z bzw. r/g/b/x/y/z.

Nach dem Aufruf der Methode Read sind die Daten und ihre Dimensionen in der Klasse gespeichert, und stehen der weiteren Verwendung zur Verfügung. Es werden ebenfalls die Gradienten und die ImageGradMagnitudes berechnet, die in gradData und in imageDa zur Verfügung stehen. Der Typ der gelesenen Datei wird in type abgespeichert. (type = 0 ...*.dat; type = 1 ... *.vol). Das ursprüngliche Datenset steht in vtk_data.

Die Methode GetType liefert den Typ der eingelesenen Datei (type = 0 ... *.dat; type = 1 ... *.vol).

Die Methoden GetXdim(), GetYdim() und GetZDim liefern die jeweilige Dimension der Daten in X-, Y- und Z-Richtung.

Die Methode GetData liefert die ImageDaten die in vtk_Data gespeichert sind (Zuvor mit Read gelesen).

Die Methode GetCharData liefert die ImageDaten nicht als vtkImageData, sondern als Array aus Charakters.

Die Methode GetDataElement erlaubt es durch Angabe der Koordinaten eines Voxels als Parameter den einzelnen Datenwert dieses Voxels als short zurückzugeben. Eine Fehlerabfrage ist nicht implementiert (bei ungültigen Dimensionswerten oder wenn eine Datei zuvor nicht gelesen wurde kommt es zu einem Fehler.)

Die Methode GetHistData berechnet für einen zuvor eingelesenen Datensatz das Histogram. Der Parameter muss den Typ der gelesenen Datei angeben. Da das Canvas, in dem das Histogram dargestellt wird eine fixe Länge von 512 hat, werden die Daten auf ein entsprechendes Histogram mit 512 Datenwerten gemappt. Das Ergebnis wird in p2HistData gespeichert.

Die Methode GetGradHist berechnet auf die gleiche Weise wie GetHistData ein Histogram für die Gradienten der Bilddaten.

3.5 MyRenderer

Die Klasse MyRenderer implementiert eine Visualisierungspipeline zur Darstellung der Daten.

```
class MyRenderer : public vtkQtRenderWindow
{

public:
      MyRenderer(QWidget *parent, const  char *name);
      ~MyRenderer();
      void setOpTF(vtkPiecewiseFunction *func);
      void setColTF(vtkColorTransferFunction *func);
      void setData(vtkImageData *data, int x, int y, int z);
      int MyRender(int Diff, int Amb, int Spec, int SpecPower, int mode);
private:
      vtkRenderer *renderer;
      vtkVolume *volume;
      vtkVolumeRayCastMapper *volumeMapper;
      vtkVolumeProperty *volumeProp;
      vtkUnsignedShortArray *array;
      vtkUnsignedCharArray *array1;
      vtkImageData *imageData;
      vtkPiecewiseFunction *opacityTF;
```

```
    vtkColorTransferFunction *colorTF;
    vtkVolumeRayCastCompositeFunction *compositeFunction;

    int Status;

};
```

Im Konstruktor werden alle Datenelement auf NULL initialisiert, und eine Instanz der Klasse vtkRenderer und vtkQtRenderWindowInteractor erzeugt. Dokumentationen zu diesen Klassen findet man auf www.vtk.org.

Die Methode setOpTF setzt die Transferfunction für die Opazität.

Die Methode setColTF setzt die Color-Transferfunction.

Die Methode setData übergibt der Klasse MyRenderer die Bilddaten, die zu visualisieren sind.

Die Methode getCam liefert die aktuelle Kameraeinstellung.

Die Methode setCam setzt eine aktuelle Kameraeinstellung.

Die Methode MyRender implementiert die eigentliche RenderingPipeline. Als Parameter werden die Parameter für das Shading übergeben (ambient coefficient, specular coefficient, diffuse coefficient, specular power). Der Parameter mode bestimmt, ob sich die Transferfunction auf die Skalarwerte der Daten oder auf die Werte der Gradientmagnitudes bezieht.

3.6 vtkQtRenderWindow

Klasse von Matthias Koenig

http://www.isg.cs.uni-magdeburg.de

Wird benötigt um Qt und VTK gemeinsam zu „nutzen“.

3.7 vtkQtRenderWindowInteractor

Klasse von Matthias Koenig

http://www.isg.cs.uni-magdeburg.de

Wird benötigt um Qt und VTK gemeinsam zu „nutzen“.

3.8 FigureEditor

Diese Klasse implementiert die „View“ auf einen dem Konstruktor übergebenen Canvas. Sie stellt also eine Implementation zur Realisierung eines Diagramms mit Transferfunctions dar. Ebenfalls existierten Methoden um im Hintergrund ein Histogram einzufügen, und dieses in der y-Richtung zu zoomen. Es erlaubt auch ein Zoomen über die x-Richtung und ein scrollen über den gesamten Bereich.

```
class FigureEditor : public QCanvasView {
    Q_OBJECT

public:
    FigureEditor(QWidget* parent=0, const char* name=0, WFlags f=0);

    QSize sizeHint() const;
      QCanvas *getCanv(void) { return canv; }
      void createTF(void);
      void createColTF(void);
      void setCurveCol(int r, int g, int b);
      void setScale(int val);
      int getScale(void);
      void createHist(int *histdata);
      vtkPiecewiseFunction *getOpacTF(void);
      vtkColorTransferFunction *getColTF(void);
      vtkColorTransferFunction *getColTF2(void);


public slots:
      void setPointMode (int mode);
      void setRGBMode (int mode);

protected:
    void contentsMousePressEvent(QMouseEvent*);
    void contentsMouseReleaseEvent(QMouseEvent*);
    void contentsMouseMoveEvent(QMouseEvent*);
```

```
    void resizeEvent( QResizeEvent* );
    void showEvent( QShowEvent* );
    void hideEvent( QHideEvent* e);

private:
      void initialize();
      QPoint moving_start;
      QCanvas* canv;
      QCanvasItem* moving;
      QCanvasItem* old_moving;
      Curve *change;
      Curve *opacity;
      Curve *rCurve, *gCurve, *bCurve;
      QCanvasItem *histLines[400];
      int pointMode;
      int scale;
      int rgbMode;

};
```

Der Konstruktor ruft die Methode initialize() auf. Hier wird eine Opacity Transferfunction und ein Canvas instanziert.

Alle Variablen werden auf 0 initialisiert, bis auf pointMode (gibt an, ob ein Punkt eingefügt, gelöscht, oder bewegt wird. Zu beginn auf Einfügen), relativeScal (bei gezoomten Diagramm ist hier der relative Skalierungswert zur vorherigen Ansicht gespeichert. Zu Beginnn auf 1.0), origScale (Originale skalierung. Zu Beginn auf 4096; dies entspricht dem *.dat format).

Die Methoden contensMouseMoveEvent, contentsMousePressEvent und contentsMouseReleaseEvent werden bei einer interaktiven Bearbeitung des Diagramms mit der Mouse aufgerufen. Hierbei können je nach Modus (Punkt einfügen, löschen, bewegen – kodiert in pointMode) die jeweiligen Aktionen durchgeführt werden.

Die Methoden CreateTF, CreateColTF erzeugen jene Transferfunction die zu Beginn angezeigt werden. Wird in einer späteren Implementation ein Algorithmus erzeugt, um eine Transferfunction zu erzeugen, müssen diese Methoden angepasst werden, um eine korrekte initiale Darstellung zu gewährleisten.

Die Methoden getColTF, getColTF2, getOpacTF erzeugen anhand der aktuellen Kurven im Diagramm die benötigten VTK-Transferfunctions (vtkColorTransferFunction, vtkPiecewiseFunction). Da die Klasse FigureEditor für alle Darstellungen (Transferfunction, Color-Transferfunction mit drei einzelnen Farbkanälen, Opacity-Color Darstellung gemischt – Style2) verwendet wird existieren die jeweiligen Methoden, die je nach Verwendungsart aufgerufen werden müssen.

Die Methoden saveOpTf und saveColTf dienen zum Speichern der Kurven in einem File, das als Parameter übergeben wird. Ein weiterer Parameter bestimmt den Style des geöffneten Registerblattes.

Die Methode setRgbMode bestimmt, welche der drei Kanäle durch die Methoden contensMouseMoveEvent, usw. interaktiv verändert wird.

Um beliebige Kurven in das Diagramm setzen zu können, existieren die Methoden setOpacity, setRCurve, setGCurve und setBCurve. Die Methode setChange bestimmt jene Kurve, die aktuell bearbeitet wird.

Die Methode createHist erzeugt im Hintergrund ein Histogram. Als Parameter werden die Histogramdaten für Datenwerte und ImageGradientMagnitude erwartet. Der dritte Parameter bestimmt, welches Histogram berechnet wird.

Die Methode actualizeHist implementiert das Zoomen in y-Richtung. Der Parameter ist dabei der aktuelle Wert des vertikalen Sliders im Hauptfenster.

Die Methode updateHist berechnet die Darstellung des Diagramms, wenn sich der Maßstab in x-Richtung ändert. Der Parameter ist der Wertebereich des geöffneten Datensatzes.

Die Methode scrollUpdate implementiert das Scrollen über den gesamten Wertebereichs des Diagramms. Als Paramter wird der Wert des Sliders übergeben.

Die Methode setScale setzt das Diagramm auf eine beliebige Skalierung, die als Parameter erwartet wird (gemeint ist der neue Wertebereich).

3.9 TFStyle1

Aufgrund der zuvor beschriebenen Klasse FigureEditor implementiert diese Klasse den ersten „Style“ des Transferfunction-Editors mit einem eigenem Diagramm für die Opacity-Transferfunction und einem Diagramm für die Color-Transferfunction.

```
Class TFStyle1 : public QWidget
{
      Q_OBJECT
public:
      TFStyle1( QWidget *parent=0, const char *name=0 );

      FigureEditor *opacity, *rgbtf;
      QVBoxLayout *vbox;
      QSpinBox *scalebox;

public slots:
      void scaleNew(int ind);
      void scrollTF(int ind);
};
```

Im Konstruktor werden beide Editoren für die Transferfunctions erzeugt.

3.10 TFStyle2

Aufgrund der zuvor beschriebenen Klasse FigureEditor implementiert diese Klasse den zweiten „Style“ des Transferfunction-Editors bei der die Farbwerte durch die Farben der einzelnen Punkte der Transferfunctions bestimmt wird.

```
class TFStyle2 : public QWidget
{
      Q_OBJECT
public:
    TFStyle2( QWidget *parent=0, const char *name=0 );

      FigureEditor *opacity, *rgbtf;
      QVBoxLayout *vbox;
```

```
      QHBoxLayout *hbox;
      QPushButton  *bColor;
      QSpinBox *scalebox;
      QLabel *lcol;
      QColor *col;
      int scalefact;


public slots:
      void scrollTf(void);
      void scaleNew(void);
      void setcolor(void);
};
```

Im Gegensatz zum ersten „Style“ gibt es hier zusätzlich einen Farbdialog zur Auswahl einer Farbe und eine Anzeige der gerade aktuell gewählten Farbe. Die Methoden scrollTf und scaleNew rufen wie beim ersten Style die zugehörigen Methoden des FigureEditors zum scrollen und zoomen auf.

3.11 TFDesign

Die Klasse TFDesign erzeugt das Hauptfenster. Sie kreiert jeweils eine Instanz von TFStyle1 und TFStyle2 um die notwendigen „Werkzeuge“ zum Erstellen von Transferfunctions bereit zu haben. Weiters werden alle notwendigen Klassen zum Rendern eines Datensatzes erzeugt.
Diese Klasse stellt auch die notwendigen Verknüpfungen zwischen den einzelnen Klassen her. Sie dient als eine „universelle Schnittstelle“.

```
class TFDesign : public QWidget
{
    Q_OBJECT

public:
      TFDesign( QWidget *parent = 0, const char *name = 0 );
      void ReadFile(QString file);

public slots:
      void render (void);
      void scalGrad (int renderM);

protected:
      QRadioButton *rbAdd, *rbMove, *rbRem;
      QRadioButton *rbR, *rbG, *rbB;
      QRadioButton *rbScalar, *rbGradient;
      CReadData data;
      MyRenderer *rendWin;          QHBoxLayout *box;
      QVBoxLayout *vbox;
      QVBoxLayout *vboxCan;
```

```
      QVBoxLayout *vView;
      QVBoxLayout *vtab1box;
      QButtonGroup *bgrpTF;
      QButtonGroup *bgrpRGB;
      QButtonGroup *bgrpScalGrad;
      QPushButton  *bRender;
      QTabWidget *tab;
      QLabel *labDiffuse;
      QLabel *labAmbient;
      QLabel *labSpecular;
      QLabel *labSpecularPower;
      QSpinBox *spinDiffuse;
      QSpinBox *spinAmbient;
      QSpinBox *spinSpecular;
      QSpinBox *spinSpecularPower;
      int renderMode;
public:
      TFStyle1 *tfstyle1;
      TFStyle2 *tfstyle2;
};
```

Im Konstruktor werden alle Klassen instanziert und die Verknüpfungen hergestellt.

3.12 GradSphere

Diese Klasse erzeugt ein neues Fenster mit einer Sphere und dessen Koordinatenachsen. Anhand dieser soll man die Gradienten jener Voxel auswählen können, die dann auch gerendert werden.

Die Häufigkeit mit der Gradienten in eine bestimmte Richtung zeigen wird mittel Color Mapping auf die Oberfläche dieser Kugel gemappt. Das erzeugen des Histograms für die Gradienten auf der Oberfläche übernimmt der Filter GradientSphereHistogramFilter. Dieser Filter braucht die Klasse GradSphere als Basisklasse.

```
class GradSphere : public vtkQtRenderWindow
{
public:
      GradSphere(QWidget *parent, const  char *name);
      void setGradData(vtkImageData *data);
      void upDate(void);
      ~GradSphere();



protected:
    void closeEvent( QCloseEvent* );
```

```
public:
      vtkSphereSource *sphere;
      vtkActor *aSphere;
      vtkRenderer *renderer;
      vtkQtRenderWindowInteractor *iren;
      vtkActor2D *textActor;
      vtkTextMapper *textMapper;
      vtkCellPicker *picker;
      CellPicking *cellPicking;
      GradientenSphereHistogramFilter *histoFilter;
      vtkImageData *gradData;
      vtkLookupTable *lut;
      vtkDataSetMapper *map;
};
```

Die Methode setGradData übergibt das Datenarray mit den Gradienten des geöffneten Dateisatzes.

Die Methode upDate wird benötigt, wenn auf einen anderen Datensatz gewechselt werden soll.

3.13 GradientenSphereHistogramFilter

Um das Histogram der Gradienten auf die Sphere der Klasse GradSphere zu mappen, wird dieser Filter benötigt. Anhand des Datensatzes berechnet er die Skalarwerte der einzelnen Punkte der Sphere.

```
class GradientenSphereHistogramFilter : public vtkDataSetToDataSetFilter
{

public:
      static GradientenSphereHistogramFilter *New() {return new
GradientenSphereHistogramFilter(); }

public:
      GradientenSphereHistogramFilter();
      ~GradientenSphereHistogramFilter() {};
      void setGradientenSource(vtkImageData *data);
```

```
      void setSphereRes(int res) {sphereResolution = res;}
      float getRange() {return ScalarRange[1];}

      void Execute();

      vtkImageData *gradientenSource;
      float ScalarRange[2];
      int sphereResolution;
};
```

Die Methode New erzeugt einen neue Instanz dieser Klasse.
Der Konstruktor setzt die Auflösung der Sphere auf 18°.

Die Methode getRange liefert die höchste Anzahl von Gradienten die innerhalb eines Bereichs in die gleiche Richtung zeigen.
Die Methode setSphereRes erlaubt es die Auflösung der Sphere auf eine beliebige zu verändern. Man sollt sich jedoch dessen bewusst sein, dass der Rechenaufwand bei einer kleineren Auflösung um ein erhebliches steigt.

Die Methode setGradientenSource übergibt dem Filter die zugrunde liegenden Bilddaten.

Die Methode Execute berechnet schlussendlich die Skalarwerte der einzelnen Punkte der Sphere die dem Histogram für die Gradienten entsprechen.

3.14 CellPicking

Um die Bereiche dessen Gradienten gerendert werden sollen interaktiv wählen zu können wurde diese Klasse implementiert.

Sie erlaubt durch Positionierung der Maus und durch Drücken der Taste „p“ das picken einer Zelle. Dabei wird von der Position der Maus ein Strahl weggeschickt, der mit der Sphere geschnitten wird. Um genauere Informationen über diese Methode zu erfahren sollte man sich die Dokumentation der Klasse vtkPointPicker auf www.vtk.org anschauen.

```
class CellPicking : public vtkCommand
{
public:
            vtkCellPicker *picker;
            vtkIdType cellId;
```

```
            float pickedPoint[3];

            CellPicking() {
              picker = 0;
                  cellId=-1;
            }

            static CellPicking *New() {return new CellPicking; }
            virtual void Execute(vtkObject *caller, unsigned long eventId, void
*callData) {
                  picker = reinterpret_cast<vtkCellPicker*>(caller);

                  if (picker->GetCellId() >= 0) {
                        cellId = picker->GetCellId();
                        picker->GetPickPosition(pickedPoint);
                  } else {
                        cellId = -1;
                  }
            }
            virtual void StartInteraction() {}
            virtual void EndInterAction() {}

            vtkIdType getCellid() { return cellId; }
            float *getPoint() { return pickedPoint; }

};
```

Die Methode Execute wird durch Drücken der Taste „p“ aufgerufen, und bestimmt einen aktuellen Punkt der Sphere.

Mit der Methode getPoint erhält man den „gepickten“ Punkt.

Die Methode getCellId liefert die Id der zugehörigen Zelle.