Einführung in die Programmierung
GUI Programmierung Teil 3

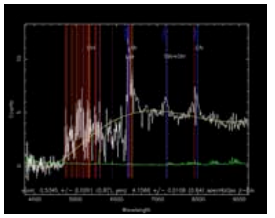
30.01.2014

Jan Quadflieg & Andreas Thom

Technische Universität Dortmund
Lehrstuhl 11 - Algorithm Engineering

Motivation





Bildnachweis: http://sdss.org

- Sloan Digital Sky Survey (SDSS) beinhaltet Daten über ca. 469 Millionen astronomische Objekte.
- Teleskope liefern photometrische und spektroskopische Daten.
- Datenvolumen des SDSS ca. 70 TB (Data Release 10 - Juli 2013)

Problemstellungen

- Identifikation „seltener“ Objekte.

Bildnachweis: http://sdss.org

Problemstellungen

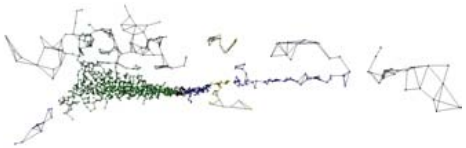
- Identifikation „seltener“ Objekte.
- Klassifikation „unbekannter“ Objekte.

Bildnachweis: http://sdss.org

Problemstellungen

- Identifikation „seltener“ Objekte.
- Klassifikation „unbekannter“ Objekte.
- Exploration zur Strukturerkennung.

Bildnachweis: http://sdss.org



Planung und Anforderungsanalyse

- Identifikation von Strukturen über einen großen Bereich des Himmels.

Planung und Anforderungsanalyse

- Identifikation von Strukturen über einen großen Bereich des Himmels.
- Astronomisches Vorgehen.

Planung und Anforderungsanalyse

- Identifikation von Strukturen über einen großen Bereich des Himmels.
- Astronomisches Vorgehen.
- Entwicklung eines Ablaufplans.
  1. Selektion von interessanten Objekten.
  2. Identifikation von Strukturen.
  3. Repräsentation der gefundenen Strukturen.
  4. Finde weitere Objekte, die dieser Repäsentation genügen.
  5. Interpretation der Ergebnisse.

Planung und Anforderungsanalyse

- Identifikation von Strukturen über einen großen Bereich des Himmels.
- Astronomisches Vorgehen.
- Entwicklung eines Ablaufplans.
  1. Selektion von interessanten Objekten.
  2. Identifikation von Strukturen.
  3. Repräsentation der gefundenen Strukturen.
  4. Finde weitere Objekte, die dieser Repäsentation genügen.
  5. Interpretation der Ergebnisse.
- Wie können Ergebnisse visualisiert werden?

Entstehungsprozess

- Wie können Ergebnisse interpretiert/bewertet werden?
- $\rightarrow$ Lösung durch Visualisierung der isolierten Strukturen.

hier: VTK 5.6.1 (The „Visualization Toolkit“)
Download: **http://www.vtk.org**
Aktuell: VTK 6.1.0 (abwärtskompatibel?)

Entstehungsprozess

VTK

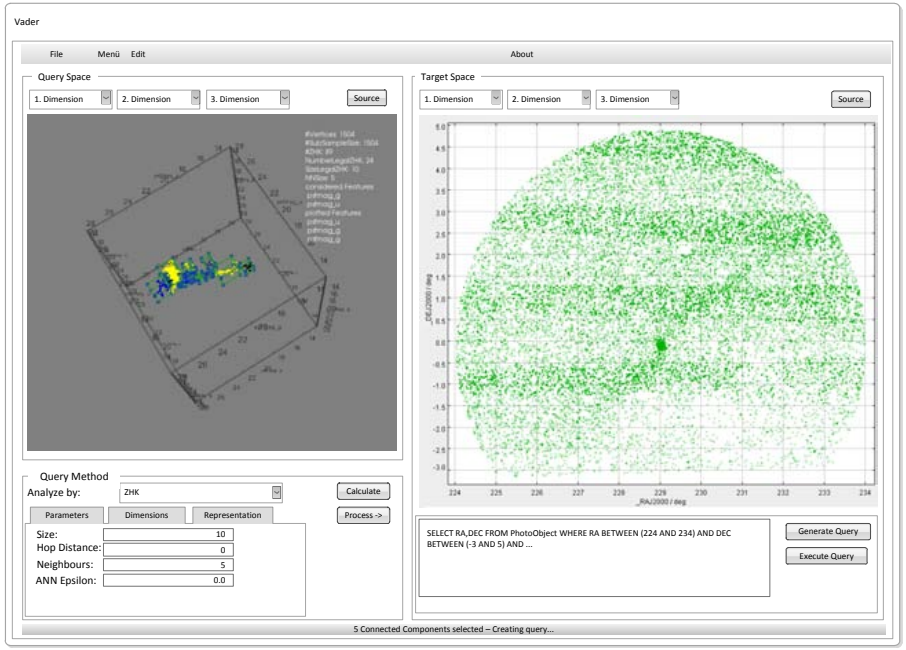
- Open-Source-C++-Klassenbibliothek.
- Geeignet für 3D-Computergrafik.
- Plattform-unabhängig: läuft unter Linux/Unix, Windows, MacOS, u.a.
- Ist gut in Anwendungen mit GUI die auf Qt basieren zu integrieren.
- VTK muss auf der Zielplattform speziell für diese übersetzt werden.
- Hilfsmittel cmake erzeugt build Umgebung für die entsprechende Plattform.

Refactoring

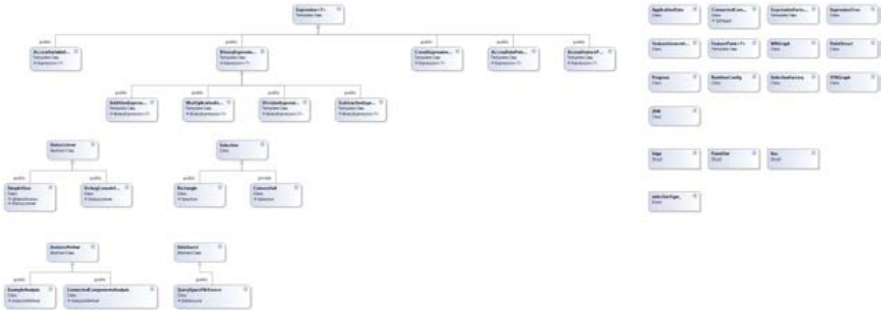
Was wollen wir?

- GUI.
- Keine Warnings mehr.
- Keine Speicherlecks mehr.
- Fehlertoleranz bzgl. Benutzereingaben.
- Modularität: Leichte Erweiterbarkeit um neue Analysemethoden.

GUI Mockup



Klassendiagramm



Details des Klassendiagramms

AnalysisMethod.h

```
class AnalysisMethod{
public:
  virtual void calculate(ApplicationData& app, StatusListener* listener = NULL)
       throw(std::string) = 0;
  virtual void replot(ApplicationData& app, std::vector<std::string>&
       plotFeatures, StatusListener* listener = NULL) throw(std::string) = 0;
  virtual void setHeader(const std::vector<std::string>& header) = 0;
  virtual std::vector<std::string> getPlotFeatures() = 0;
  virtual std::vector<std::string> getAnalysisFeatures() = 0;
  virtual void clear(StatusListener* listener = NULL) = 0;
  virtual void disableGui() = 0;
  virtual void enableGui() = 0;
  virtual std::string getName() = 0;
  virtual vtkSmartPointer<vtkRenderer> getRenderer() = 0;
  virtual vtkSmartPointer<vtkRenderer> getTargetSpaceRenderer() = 0;
  virtual QWidget* getWidget() = 0;
};
```

StatusListener.h

```
class StatusListener{
public:
  virtual void updateProgress(const std::string& message, double progress =
      -1.0) = 0;
};
```


DataSource.h

```
class DataSource{
public:
  // reads one line of data
  virtual std::string getLine() throw(std::string) = 0;

  // is another line of data available?
  virtual bool hasNextLine() throw(std::string) = 0;

  // approximate number of points available from this data source
  virtual size_t getRoughPointCnt() = 0;

  // is the exact number of points available from this data source known?
  virtual bool hasPointCnt() = 0;

  virtual std::vector<std::string> getHeader() = 0;

  // the exact number points available from this data source,
  // returns std::numeric_limits<size_t>::max() if the exact number is not yet
      known
  virtual size_t getPointCnt() = 0;

  virtual std::string getName() = 0;
};
```


Exkurs: Dynamische GUIs

```
#include <QApplication.h>
#include "DynamicGui.h"

int main(int argc, char *argv[]) {
  QApplication app(argc, argv);
  DynamicGui gui;
  gui.show();
  return app.exec();
}
```


Exkurs: Dynamische GUIs

```
#include <QObject.h>
#include <QDialog.h>
#include <QPushButton.h>
#include <QLineEdit.h>
#include <QLabel.h>
#include <QComboBox.h>

class DynamicGui : public QDialog {
  Q_OBJECT
private:
  std::vector<QComboBox *> boxes;
  std::vector<QLabel *> labels;
  QPushButton* qbAdd;
  QPushButton* qbRemove;
  QLabel* l1;
  QLabel* l2;
public:
  DynamicGui();
  ~DynamicGui();
public slots:
  void addButtonClicked();
  void removeButtonClicked();
};
```

Exkurs: Dynamische GUIs - Konstruktor

```
DynamicGui::DynamicGui(){
  setWindowTitle("DynamicGui");
  resize(600, 400);

  l1 = new QLabel(QString("#"), this);
  l1->setGeometry(0, 0, 20, 20);

  l2 = new QLabel(QString("Beschreibung"), this);
  l2->setGeometry(20, 0, 100, 20);

  qbAdd = new QPushButton(QString("+"), this);
  qbAdd->setGeometry(120, 0, 20, 20);

  qbRemove = new QPushButton(QString("-"), this);
  qbRemove->setGeometry(140, 0, 20, 20);
  qbRemove->setEnabled(false);

  connect(qbAdd, SIGNAL(clicked()), this, SLOT(addButtonClicked()));
  connect(qbRemove, SIGNAL(clicked()), this, SLOT(removeButtonClicked()));
}
```


Exkurs: Dynamische GUIs - Destruktor

```
DynamicGui::~DynamicGui() {
  while(labels.size() > 0){
    removeButtonClicked();
  }

  delete qbAdd;
  delete qbRemove;
  delete l1;
  delete l2;
}
```


Exkurs: Dynamische GUIs - Hinzufügen von Widgets

```
void DynamicGui::addButtonClicked(){
  QLabel* label = new QLabel(QString::number(labels.size()+1).append("."), this)
      ;
  label->setGeometry(0, (labels.size()+1)*20, 20, 20);
  label->show();

  QComboBox * box = new QComboBox(this);
  box->setEditable(true);
  box->setGeometry(20, (labels.size()+1)*20, 100, 20);


  box->addItem(QString("Auswahl 1"));
  box->addItem(QString("Auswahl 2"));
  box->addItem(QString("Auswahl 3"));

  box->show();

  boxes.push_back(box);
  labels.push_back(label);

  qbRemove->setEnabled(true);
}
```


Exkurs: Dynamische GUIs - Entfernen von Widgets

```
void DynamicGui::removeButtonClicked(){
  if(labels.size() > 0){
    labels[labels.size()-1]->hide();
    labels[labels.size()-1]->setParent(0);

    boxes[boxes.size()-1]->hide();
    boxes[boxes.size()-1]->setParent(0);

    delete labels[labels.size()-1];
    labels.erase(labels.end()-1);

    delete boxes[boxes.size()-1];
    boxes.erase(boxes.end()-1);
  }

  if(labels.empty()){
    qbRemove->setEnabled(false);
  }
}
```