Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Entwicklung und Implementierung eines Software-tools zur Einzelzellverfolgung.

Programm „CellTracker“

Bachelorarbeit

Leipzig, 1. Januar 2011

vorgelegt von Kunze, Michael

Studiengang Informatik

Betreuender Hochschullehrer: Prof. Dr. Ingo Röder
Professor für Medizinische Statistik und Biometrie und Direktor des Instituts für Medizinische Informatik und Biometrie (IMB), Medizinische Fakultät der TU Dresden.

Vorwort

Diese Bachelorarbeit entstand am Institut für Medizinische Informatik, Statistik und Epidemiologie (IMISE) der Universität Leipzig.

Durch die Tätigkeit als Studentische Hilfskraft am IMISE welche dieser Arbeit vorausging, befasste ich mich mit den Schwerpunkten Bildanalyse und Zelltracking. Inhalt meiner Aufgaben waren unter anderem die Anwendung eines Mathematica-Prototyps zum semi-automatischen Verfolgen von Zellen in Videomikroskopieaufnahmen unter verschiedensten Kulturbedingungen und die Nachbearbeitung sowie Fehleranalyse der automatisch erzeugten Daten.

Die damit im Vorfeld gesammelten Erkenntnisse über die konkreten Probleme der Zellverfolgung waren ausschlaggebend für die Wahl und Umsetzung meines Themas.

Besonders möchte ich mich bei Herrn Prof. Dr. Ingo Röder für die Möglichkeit bedanken, in seiner Arbeitsgruppe diese Arbeit anzufertigen. Bedanken möchte ich mich auch an dieser Stelle für die Hilfe und Unterstützung bei Nico Scherf und Ingmar Glauche.

Ganz besonderer Dank gilt meinen Eltern, die mich während meines ganzen Studiums unterstützt und immer an mich geglaubt haben. Ohne Sie wäre diese Arbeit nie zustande gekommen.

Abstract

Using time-lapse microscopy (i.e. *Live Cell Imaging* movies), it is possible to observe cells and ist progeny (offspring) within certain cell cultures. Sophisticated image processing methods admit to highly automate this process of cell observation to track cells for about hundreds of sequential images. But there are situations occuring quite frequently, where the choice of the automated tracking algorithm is not correct or ambiguous. At these points a manual correction step is necessary, as it currently presents the only way to obtain complete genealogical information from real experimental data.

The target of this work was to develop a software-tool that allows intuitive and interactive track editing via efficient user interaction, to complement the existing automated cell tracking. So this software has to be able to load and display the underlying image stacks (several thousand images per experiment). Additional metadata (obtained by automated object detection of cells and resulting automatically generated cell slices, i.e. fragments of cell tracks) has to be loaded and presented properly. But the main task of the development of this software was to create a platform for skilled users to add, correct and connect the cell slices obtained by the automated processing in an *efficient and user-friendly* way. Thus, usablity was the main focus of this work.

The developed tool is a fundamental and indispensible component of the analysis- and evaluation-pipeline for time extended movies of cell cultures. By this it contributes to an further understanding the organisation of cell development and cell migration.

Kurzzusammenfassung

Mittels Zeitrafferaufnahmen ist es möglich, einzelne Zellen und deren Nachkommenschaft innerhalb bestimmter Zellkulturen zu verfolgen. Fortgeschrittene Bilderverarbeitungsmethoden gestatten es, diesen Prozess der Zellverfolgung über Hunderte von aufeinanderfolgenden Bildern weitgehend zu automatisieren und baumartige Zelltrajektorien zu erzeugen. Allerdings kommt es dabei häufig zu Situationen, in denen die entsprechenden Algorithmen zu ungenau werden und ein Benutzereingriff erforderlich ist.

Ziel dieser Arbeit war ein entsprechendes Softwaretool zu entwickeln, dass diesen Benutzereingriff gewährleistet und damit die automatische Zellverfolgung komplementiert. Die entsprechende Software ist in der Lage , die zu Grunde liegenden Bildstapel zu laden und als Bildsequenz darzustellen. Darüber hinaus werden zusätzliche Informationen zu den erkannten Zellobjekten und den automatisch generierten Zelltrajektoren (vor-prozessierte Metadaten aus *Mathematica*) geladen und repräsentiert. Zentrale Aufgabe der Software ist es, eine Plattform zu schaffen, auf der ein geschulter Nutzer die erkannten Trajektorienabschnitte effizient und benutzerfreundlich erkennen und gegebenenfalls miteinander verknüpfen kann. Die entsprechend ergänzten Trajektorien können anschliessend als Metadaten für die weitere Verarbeitung und Auswertung zur Verfügung gestellt werden.

Das Softwaretool stellt einen wichtigen Bestandteil einer Auswertepipline für zeitlich ausgedehnte Videoaufnahmen von Zellkulturen dar und leistet damit einen Beitrag zum Verständnis von Zellorganisation und zellulärer Migration.

Inhaltsverzeichnis

1. Einleitung und Überblick
1.1 Fragestellung, Motivation
1.2 Problemstellung
1.3 Ziel der Arbeit
1.4 Gliederung

2. Grundlagen / Prinzipien
2.1 Phasen der Zellverfolgung
2.2 Struktur der vorhanden Daten

3. Planung und Konzeption (Programmdesign)
3.1 Ist-Analyse
3.2 Auswahl des geeigneten Frameworks
3.3 Die wichtigsten Problematiken im Vorfeld
3.4 Programmkonzept

4. Modellierung (UML-Diagramme)
4.1 Die Klassen `Project`, `Cell`, `Track`, `NewTrack`, `Time`

5. Implementierung
5.1 Import der Metadaten/Trackdaten
5.2 Interne Datenverwaltung
5.3 Externe Datenverwaltung
5.4 Algorithmen für Workflow

6. Programm CellTracker

7. Zusammenfassung und Ausblicke
7.1 Evaluierung der verschiedenen Strategien
7.2 Diskussion
7.3 Möglichkeiten der Weiterentwicklung / Ausblicke

Glossar

Literaturverzeichnis

1. Einleitung und Überblick

1.1 Fragestellung, Motivation

Die Untersuchung der Eigenschaften und des Verhaltens von Zellen, im Besonderen von Stammzellen ist wichtig um grundlegende Aspekte des Lebens zu verstehen. Dazu gehört z.B. die Entwicklung von Organismen (Embryologie), die Funktionsweise von regenerativen Geweben wie Blut oder Haut oder auch die Entwicklung und Ausbreitung von Tumoren.

Diese Dinge basieren auf einem koordinierten Zusammenspiel zwischen vielen einzelnen Zellen und ihrer Umgebung. Um das Ganze zu verstehen, sollte man demnach die Phänomene auch auf Einzelzellebene betrachten.
Eine besonders wichtige Aufgabe ist es dabei, die Stammbäume der Zellen zu rekonstruieren und auszuwerten. Weitere Eigenschaften der beobachteten Zellen (z.B. Ort, Grösse etc.) können benutzt werden, um weitere Aspekte des Zellverhaltens, wie Bewegungsmuster, Wachstum oder Zellkontakte in geeigneter Weise untersuchen zu können.

Experimente und Analyseverfahren müssen entworfen werden, um beispielsweise das Verhalten (und die Unterschiede im Verhalten) von Blutstammzellen unter normalen (homeostatischen) oder gestörten Bedingungen zu untersuchen. Dabei versucht man die Experimente so zu konzipieren, dass beispielsweise folgende Fragen beantwortet werden können:

Auf welche Art werden weitere Zellen aus den Stammzellen produziert? (z.B. verhalten/teilen sich die Nachkommen symmetrisch oder asymmetrisch)
Wie unterschiedlich ist diese Entwicklung bei verschiedenen Zellen einer Population?
Entstehen dabei räumliche oder zeitliche Muster?

Die gewählte Untersuchungsmethode sollte dabei die Zellen möglichst nicht beschädigen oder (zer-)stören und dabei in der Lage sein, dynamische Veränderungen über die Zeit zu dokumentieren. Hierbei ist Videomikroskopie (*Live Cell Imaging*) – dazu gehören verschiedene bildgebende Verfahren wie Lichtmikroskopie oder Laserscanning – besonders gut geeignet, da sie eine hohe zeitliche und räumliche Auflösung ermöglicht [Schroeder2008, Megason2007].

Unterschiedliche bildgebende Verfahren haben natürlich auch verschiedene Limitationen. Dadurch gibt es noch immer kaum Möglichkeiten, Langzeituntersuchungen in vivo[1] durchzuführen, da es schwer ist einen Organismus zu „durchleuchten“ – Mäuse sind beispielsweise nicht transparent – werden diese Experimente meist in vitro[2] durchgeführt. Diese Beobachtungen geschehen dann über einen längeren Zeitraum (mehrere Tage) unter mehr oder weniger strikt definierten Umgebungsbedingungen.

1.2 Problemstellung

Das Hauptproblem der manuellen Auswertung dieser Live Cell Imaging - Experimente ist die Masse der Daten. Die großen Datenmengen entstehen, da hunderte von Zellen in vielen tausend Bildern auszuwerten sind.

Es stellt sich die Frage, wie man soviel Objekte in den Filmen analysieren kann. Manuell ist dies weitestgehend unmöglich. Es sei denn man verzichtet beispielsweise auf exakte Bestimmung von Position, Größe und Form der Zellen. Doch selbst dann ist die Zahl auswertbarer Zellen stark limitiert.

Deshalb kommen hier automatische bzw. rechner-unterstützte Methoden zum Einsatz.

---

[1] in vivo – Prozesse, die im lebendigen Organismus ablaufen.

[2] in vitro – Organische Vorgänge, die außerhalb eines lebenden Organismus stattfinden.

Das Problem hierbei ist jedoch, dass die meisten automatischen Methoden nicht in der Lage sind fehlerfrei Zellen zu erkennen. Noch schwieriger ist es für automatische Ansätze jedoch, Zellen über einen längeren Zeitraum sicher zu verfolgen. Besonders problematisch sind hier zum Beispiel Situationen in denen die Zellen besonders dicht liegen (Überlappung) oder sich sehr schnell bewegen. Das stellt ein prinzipielles Problem in vielen Anwendungsgebieten dar, beim momentanen Stand der Forschung in *Künstlicher Intelligenz* (KI) und künstlichem Sehen (möglicherweise fehlt hier den automatischen Verfahren doch die gesammelte Erfahrung und die Adaptionsfähigkeit des menschlichen Sehapparates).
Auch bei 99 prozentiger Korrektheit hätte man bei einem Film mit 10 000 Bildern und 100 Zellen pro Bild 10 000 Fehler bei der Zellzuordnung zwischen zwei Bildern. Damit lassen sich keine robusten Zellgenealogien[1] erstellen. Momentan ist eine Einbeziehung des Menschen in die Auswertung der Zellfilme noch unumgänglich.

Deshalb ist eine kombinierte Lösung aus automatischer Detektion und Verfolgung von Objekten und einer manuellen Nachbearbeitung momentan vermutlich die effektivste Methode zur Extraktion von Einzelzelldaten aus Videodaten.

Daraus leitet sich nun auch das Ziel dieser Arbeit ab.

1.3 Ziel der Arbeit

In der Arbeitsgruppe von Ingo Röder gab es bereits Entwicklungen zur automatischen Detektion und Verfolgung von Zellen in Mikroskopievideos. Da jedoch komplette Stammbäume von Interesse waren bzw. weiterhin sind mussten die Daten noch von Hand nachbearbeitet werden (hier kam ein einfacher Interface-Prototyp zum Einsatz der in Mathematica implementiert wurde). Umgebungen wie Mathematica (oder auch Matlab und andere Skriptsprachen) eignen sich sehr gut für das sogenannte ‚Rapid Prototyping'. Die damit erstellten Lösungen haben jedoch meist deutliche Nachteile im Bezug auf Benutzbarkeit und Geschwindigkeit gegenüber C, C++ oder Java. Dies fällt bei einem recht komplexen User-Interface wie einem Zelltracking-tool ins Gewicht, besonders wenn eine große Menge an Daten vorliegt. Diese Lücke sollte nun von einem neuen und effizienten (hinsichtlich Rechnerkapazitäten und Benutzbarkeit) Tool geschlossen werden, das auch auf verschiedenen Plattformen lauffähig sein sollte. Wie bereits erwähnt sollte dabei besondere Aufmerksamkeit auf die effiziente Benutzbarkeit gelegt werden, um die benötigte Zeit für die Analyse eines Filmes möglichst gering zu halten. Dies stellt große Anforderungen besonders an das Design des Workflows (z.B. möglichst wenig „Klicks", eine effiziente Darbietung der nötigen Informationen etc.).

1.4 Gliederung

In dieser Arbeit wird die entwickelte Software (das Programm „CellTracker") vorgestellt.

Vom Bildstapel bis hin zu den fertig gestellten Baumstrukturen sind mehrere Schritte nötig. Die einzelnen Phasen der Vorverarbeitung, wie Objekterkennung und Tracking sowie die darauf folgende Phase der manuellen Bearbeitung werden im Vorfeld im Kapitel 2 beschrieben.

In Kapitel 3 wird auf die Planungsphase der entwickelten Software „CellTracker" eingegangen. Bestehende Lösungen / Applikationen (Ist-Analyse), bereits bekannte Probleme, sowie fundamentale Anforderungen werden hier aufgezählt und beschrieben.

---

[1] Zellgenealogien (Baumstrukturen, die eine „Stammzelle" und all ihre beobachteten Nachkommen in einer hierarchischen Weise darstellen)

Die wichtigsten Klassen werden dann unter Verwendung von UML-Klassendiagrammen in Kapitel 4 dargestellt.

Kapitel 5 enthält Details zu den wichtigsten Aspekten der Implementierung, wie dem Import der Metadaten, die interne Datenverwaltung sowie Algorithmen für den Workflow. Dabei wird unter anderem auf die Entwicklung effizienter Algorithmen zur Zusammenführung der einzelnen Trackfragmente eingegangen.

Die Benutzung und Erläuterung der grafischen Oberfläche, sowie allgemeines Handling des Programms und mögliche Vorgehensweisen werden dann im 6. Kapitel behandelt und schließlich findet sich in Kapitel 7 die Analyse und Zusammenfassung dieser Arbeit sowie abschließend einige Bemerkungen zu möglichen weiteren Schritten.

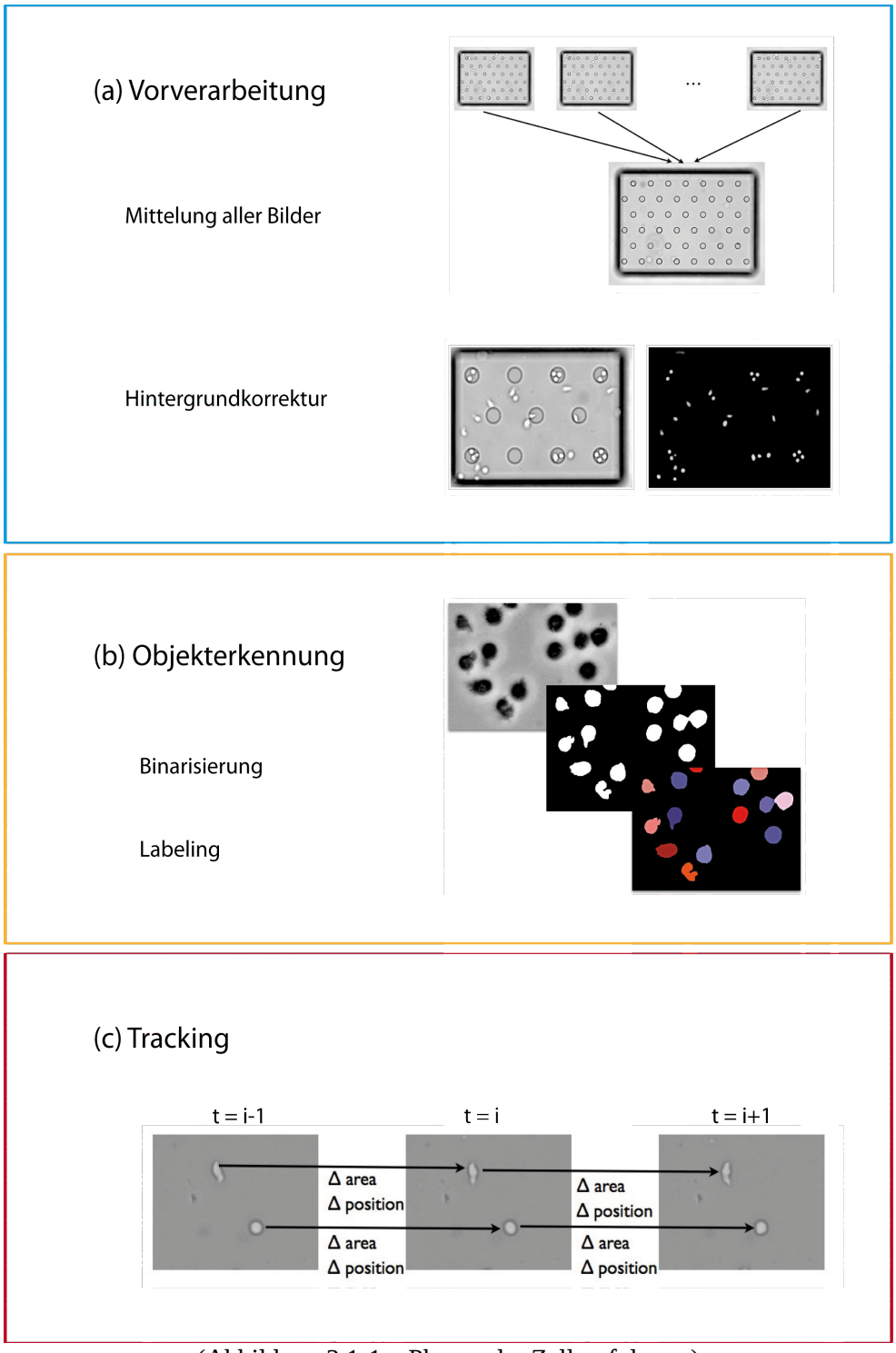
Darüber hinaus liegt der komplette Sourcecode für das Programm CellTracker auf einer CD-Rom bei.

2. Grundlagen / Prinzipien

Im Folgenden werden die technischen Vorbedingungen, d.h. die Methoden und Verfahrensweisen, welche die Basis der automatischen Zellverfolgung bilden, erörtert. Des Weiteren werden die zu Grunde liegenden Daten dargelegt, auf denen gearbeitet wird.

2.1 Phasen der Zellverfolgung

Der Weg von einem Bildstapel bis hin zu einem Film mit den erkannten Zellen und deren Trajektorien lässt sich prinzipiell in mehrere Phasen einteilen.



(Abbildung 2.1-1 – Phasen der Zellverfolgung)

Phase 1: Automatische Zellerkennung

Vorverarbeitung

In dieser Phase werden die Zellen aus den vorhandenen Bilddaten extrahiert. Dazu werden verschiedene Verfahren aus der digitalen Bildverarbeitung genutzt. Zuerst werden alle Bilder des Bildstapels aufeinander „geschoben“, sodass alle nichtbeweglichen Strukturen möglichst exakt aufeinanderpassen. Dies ist nötig, da während einer Aufnahme verschiedene Filme, d.h. verschiedene Stellen auf dem Objektträger, gleichzeitig aufgenommen werden. Dazu wird der Objekttisch des Mikroskops zwischen den Aufnahmen bewegt. Da die Tischsteuerung des Mikroskops nicht auf Mikrometer genau gesteuert werden kann, sind also die Bildaufnahmen aus einem Film nicht alle exakt an der gleichen Stelle aufgenommen. Diese Verschiebung kann mittels der sogenannten Bildregistrierung korrigiert werden [Gonzalez2008].

Die Bilddaten sind sehr verrauscht, da die Belichtungszeit für jedes Bild sehr klein gehalten werden muss, um die Zellen nicht zu schädigen (Phototoxizitaet). Dieses Rauschen sollte aus den Bildern entfernt werden, da es sonst zu vielen kleinen Fehlklassifikationen von Bildpixeln bei der Objekterkennung kommt. Hierfür wird ein nichtlinearer Glättungsfilter benutzt, der sogenannte Total-Variation Filter [Rudin1992].

Nun werden noch die störenden statischen Strukturen im Hintergrund entfernt, um eine robuste Zellerkennung zu gewährleisten. Dazu wird ein mittleres Hintergrundbild geschätzt, indem jeder Pixel im Bild über die gesamte Serie gemittelt wird. Dadurch erhält man ein Bild, das nur noch statische Strukturen enthält. Dieses wird dann benutzt, um den Hintergrund aus den Bildern zu entfernen, in dem der Hintergund von jedem einzelnen Bild abgezogen wird.

Objekterkennung

In den bearbeiteten Bildern sind jetzt nur noch die Zellen als helle Objekte vor einem dunklen Hintergrund zu erkennen. Dadurch können sie nun einfach mittels eines globalen Schwellwertverfahrens [Gonzalez2008] extrahiert werden. Die extrahierten Objektmasken werden dann gelabelt (nummeriert) und es werden verschiedene geometrische Maße für jedes Objekt bestimmt [Gonzalez2008]:

- Position im Bild
- Flächeninhalt
- Bounding-box
- Hauptachsen des Objekts
- Aussenlinie und Konvexe Hülle (werden zum Einzeichnen der Zellen benötigt) des Objekts.

Diese sogenannten Metadaten werden separat in einer Textdatei abgespeichert, d.h. eine Datei für jedes analysierte Bild, die alle erkannten Objekte und die dazugehörigen Maßzahlen beinhaltet.

Phase 2: automatische Objektverfolgung

In dieser Phase soll der Rechner die erkannten Objekte über die Zeit verfolgen, d.h. er soll jedem Objekt aus einem Bild i das (der selben Zelle entsprechende) Objekt im Bild i+1 zuordnen. Dazu können verschiedene Methoden angewendet werden. Im einfachsten Fall wird der Überlapp zwischen Objekt von einem Bild zum nächsten als Zuordnung verwendet, d.h. ein Objekt wird im nächsten Bild, dann einem neuen Objekt zugeordnet, wenn sich die beiden extrahierten Objektmasken überschneiden. Dies funktioniert sehr gut wenn die Zellen sich zwischen zwei Aufnahmen nur geringfügig bewegen. Es ist auch möglich kleinere Mehrdeutigkeiten bei der Zuordnung aufzulösen [Scherf2009]. Wenn die Zellen sich jedoch sehr schnell bewegen, ist diese einfache Zuordnung nicht mehr möglich, da es dann keine eindeutigen Überlappungen mehr gibt. Hier kann man nun zusätzlich weitere Heuristiken verwenden, um eine wahrscheinliche Zurodnung zu berechnen. Beispielsweise wird die Größenänderung des Objekts von einem Bild zum

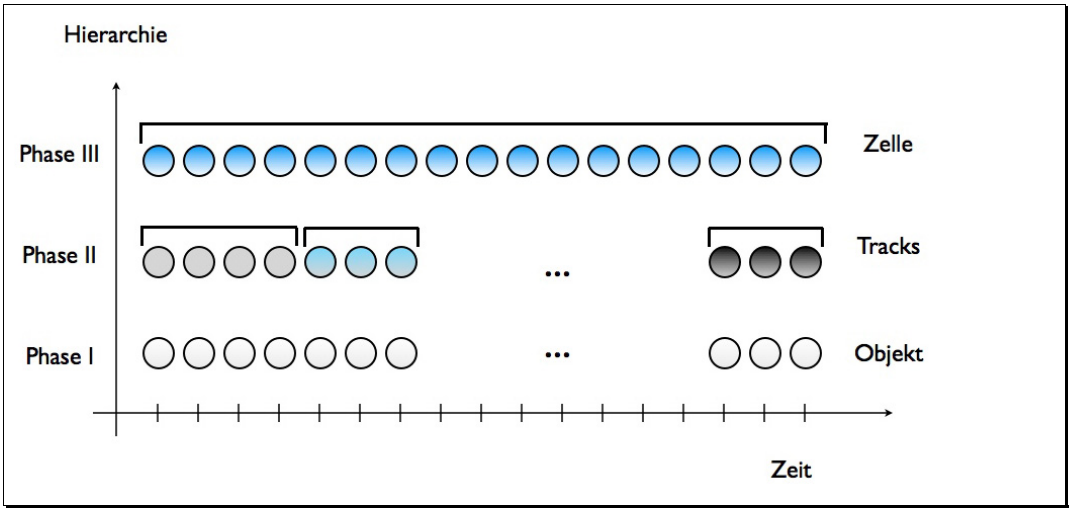
nächsten verwendet um vorherzusagen, welches das wahrscheinlichste Objekt im nächsten Bild ist. Zusätzlich wird meist noch das Bewegungsverhalten der erkannten Zelle in den letzten Zeitschritten als weiterer Faktor hinzugenommen, d.h. wenn sich eine Zelle schnell bewegt, kann man auch in einem größeren Umfeld im nächsten Bild nach ihr suchen.

Letztlich haben jedoch alle automatischen Verfahren große Probleme, wenn sich die Zellen sehr schnell bewegen und/oder sehr viele Objekte pro Bild vorhanden sind. Ein weiteres typisches Problem ist das Übereinanderlaufen von Zellen (manchmal über einige Bilder hinweg). Diese Situationen sind für den Rechner meist nicht auswertbar, obwohl der menschliche Betrachter hier meistens in der Lage ist, den „wahren“ Zellpfad zu erkennen. Das ist die Motivation eine weitere Bearbeitungsphase anzuschliessen.

Phase 3: manuelle Nachbearbeitung

Wenn man den kompletten (Lebens-)Weg einer Zelle nachvollziehen möchte, muss man sie verlässlich über mehrere Stunden (und Tage), d.h. über tausende Bilder hinweg erkennen und zuordnen. Da die vorhandenen automatischen Verfahren dazu nicht in der Lage sind und normalerweise auch nicht automatisch erkannt werden kann, wenn eine unklare Situation vorliegt, wird hier auf die Objekerkennungsfähigkeiten des menschlichen Betrachters zurückgegriffen. Der Mensch kann nun die vom Rechner vorbereiteten Tracks korrigieren und zu kompletten Zellpfaden zusammensetzen. Da der größte Teil der Arbeit (Erkennen der Zellgrenzen und Zuordnung von eindeutigen Situationen) bereits vom Rechner übernommen wurde kann man mit dieser halbautomatischen Vorgehensweise die Zellfilme in wesentlich geringerer Zeit auswerten als es komplett manuell möglich wäre. Weiterhin sind die erhaltenen Ergebnisse auch verlässlicher, als die Ausgaben von komplett automatisierten Methoden (bei denen auch manuell zumindest validiert werden muss).

Die folgende Darstellung (2.1-2) fasst die oben genannten Phasen noch einmal zusammen und soll die Hierarchie der Strukturen verdeutlichen, welche beim Erstellen der Zelltrajektorien aus den erkannten einzelnen Objekten und den daraus automatisch generierten Trajektorien entsteht. Die Daten bestehen nach der automatischen Bildverarbeitung aus einzelnen Objekten (erkannte Zellen pro Bild/Zeitpunkt). Beim automatischen Tracking werden diese dann zusammengefügt zu einzelnen Trajektorien (Tracks, bestehend aus mehreren Beobachtungen (eines Objektes) an aufeinanderfolgenden Zeitpunkten), die aber größtenteils noch unvollständig sind. Durch manuelle Nachbearbeitung werden diese dann zu kompletten Zelltrajektorien vervollständigt.



(Abbildung 2.1-2 – Hierarchie der Bearbeitungsschritte)

2.2 Struktur der vorhanden Daten

Im Folgenden werden die zu Grunde liegenden Daten der automatischen Zellerkennung und der automatischen Objektverfolgung dargestellt. Diese Daten liegen nach der Vorverarbeitung im XML-Format vor.



(Abbildung 2.2-1 – Auszug aus einer XML-Metadatei)

Diese Datei enthält verschiedene geometrische Maße der erkannten Objekte pro Bild. Es sind jedoch nicht Alle von Bedeutung für das Programm CellTracker. Es werden nur die Werte der rot markierten Tags importiert und verwendet.

```
</Track>
<Track>
 <TrackID>9286</TrackID>
 <object>
  <ObjectID>138080</ObjectID>
  <Time>4790</Time>
 </object>
 <object>
  <ObjectID>138182</ObjectID>
  <Time>4791</Time>
 </object>
 <object>
  <ObjectID>138277</ObjectID>
  <Time>4792</Time>
 </object>
 <object>
  <ObjectID>138387</ObjectID>
  <Time>4793</Time>
 </object>
 <object>
  <ObjectID>138490</ObjectID>
  <Time>4794</Time>
 </object>
 <object>
  <ObjectID>138592</ObjectID>
  <Time>4795</Time>
 </object>
 <object>
  <ObjectID>138704</ObjectID>
  <Time>4796</Time>
 </object>
</Track>
    ⋮
```

(Abbildung 2.2-2 – Auszug aus der XML-Trackdatei)

Diese Abbildung zeigt das Datenformat eines generierten Tracks (die Länge der Tracks variiert entsprechend der Qualität des automatischen Trackings, dieser besteht aus sieben erkannten Zellobjekten vom Zeitpunkt 4790 bis 4796).

3. Planung und Konzeption (Programmdesign)

Da bei der Softwareentwicklung in der Planungs-, Definitions-, Enwurfs- und Implementierungsphase [Balzert2000] keine weitere Person mitwirkte, wurde hierfür ein Mittelweg zwischen formalen und pragmatischen Ansätzen gewählt.
Neben diesem Ansatz, der sich vorrangig an agiler Softwareentwicklung[1] orientiert, wurden große Teile auch mittels Extreme Programming[2] (XP) umgesetzt, d.h. dass weitere Designentscheidungen (welche über die zu Beginn der gestellen Analysen hinausgehen) erst im Laufe der Entwicklung gefällt wurden.

3.1 Ist-Analyse

Im Folgenden werden bestehende Lösungen kurz vorgestellt, deren Vor- und Nachteile konstruktiv bei der Formulierung der neuen Anforderungen berücksichtigt wurden.

Timm's Tracking Tool [TTT] – Dieses nicht kommerzielle Software-tool basiert auf vollständig manueller Analyse und ist momentan quasi das bekannteste Programm, das zur Auswertung von Zell-Stammbäumen verwendet worden ist [Eilken2009, Rieger2009]. Das grundlegende Prinzip des Programms beruht auf dem Speichern der aktuellen Mauspositionen, während der Film abgespielt wird. Damit braucht man die interessierende Zelle nicht exakt anklicken, sondern hält den Mauszeiger nur in der Nähe des Objekts und zeichnet die Position des Zeigers auf. Diese manuell gewonnen Daten werden dann zum generieren der Genealogien verwendet, d.h. es stehen Bedienelemente zur Annotation von Zellteilungen zur Verfügung. Das Programm unterstützt den Anwender dann bei dem geordneten Verfolgen der Tochterzellen, da es noch nicht bearbeitete Stellen im Baum markiert. Man kann zwischen einer Ansicht der Stammbäume und der eigentlichen Bilddaten wechseln, was das erstellen von Genealogien sehr vereinfacht (da man zwischen verschiedenen Abstraktionsebenen wechseln kann). Das manuelle Tracking bedeutet jedoch auch, dass keine exakte Zellerkennung, keine exakte Zellbewegung erkannt und analysiert werden kann, da keine automatische Vorverarbeitung benutzt wird. Es werden ausschließlich Stammbäume erstellt.
Des Weiteren sind keine offenen Definitionen der Input/Output Formate vorhanden und auch der Source-code ist nicht offen. Somit sind Erweiterungen und Anbindungen an andere Analysesoftware ausgeschlossen.
Es müssen im Voraus Ordner und Dateien manuell angelegt bzw. umbenannt werden. Es werden auch ausschließlich jpg oder tif Bild-Formate unterstützt. Andere Bildformate müssten vorher erst einmal konvertiert werden.
Außerdem ist die Anwendung des Programms plattformabhängig. Der TTTlogfileconverter (generiert log-files vom Bildstapel), welcher für die Funktionalität des TTT Programms benötigt wird, ist nur unter Windows lauffähig. Das eigentliche Cell Tracking Programm dagegen nur unter Linux.

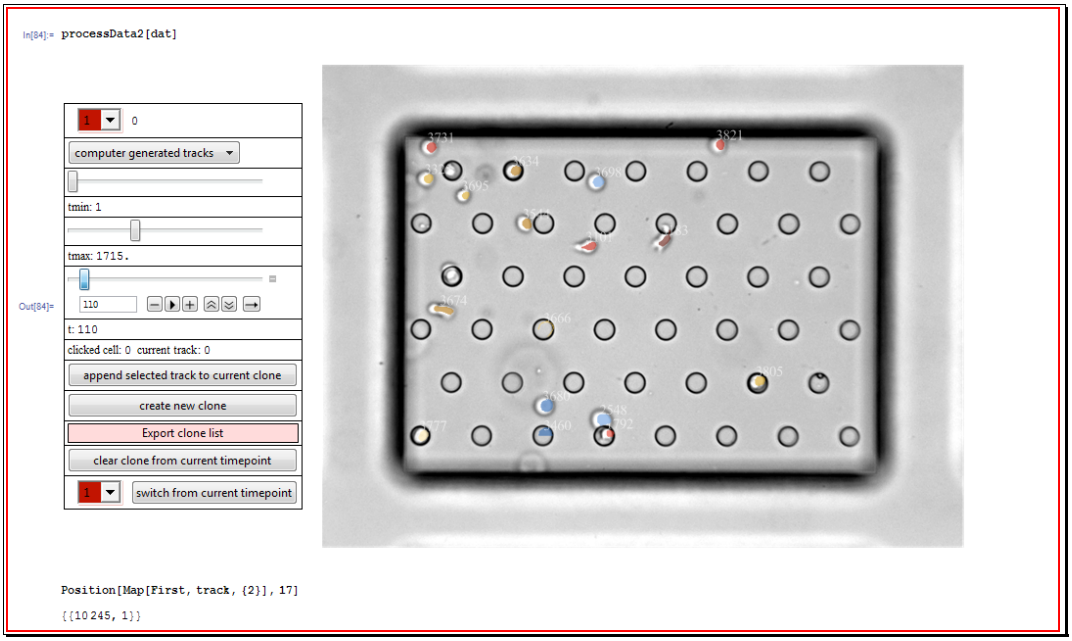
Mathematica Prototyp – Dieser Prototyp (entwickelt von Nico Scherf in der Arbeitsgruppe von Ingo Röder) enthält alle grundlegenden Funktionen zum Zusammenfügen und Nachbearbeiten der generierten Tracks. Die größte Schwachstelle des vorhandenen Systems ist die Geschwindigkeit, was eine effektive Auswertung erschwert. Auch ist dieses System nicht effizient im Bezug auf Ressourcen und es läuft

---

[1] Agile Softwareentwicklung – „Agile Softwareentwicklung versucht mit geringem bürokratischen Aufwand und wenigen Regeln auszukommen.“ … „Das Ziel Agiler Softwareentwicklung ist es, den Softwareentwicklungsprozess flexibler und schlanker zu machen, als das bei den klassischen Vorgehensmodellen der Fall ist. Man möchte sich mehr auf die zu erreichenden Ziele fokussieren und auf technische und soziale Probleme bei der Softwareentwicklung eingehen.“ [Wikipedia] (siehe auch [Wolf2008])

[2] Extreme Programming – ist ein Vorgehensmodell der Software-Entwicklung, das das Programmieren in den Vordergrund stellt und dabei einem formalisierten Vorgehen geringere Bedeutung zumisst. [Wikipedia] (siehe auch [Beck2003])

recht instabil, d.h. es treten gelegentlich Programmabstürze auf.
Desweiteren ist das Bearbeiten des Quellcodes (bzw. die Fehlersuche im Quellcode), auf Grund der verschachtelten funktionalen Programmierung, komplex. Ein weiterer ausschlaggebender Aspekt für die Neuentwicklung eines Software-tools war, dass dieser Prototyp ohne Mathematica Lizenz nicht lauffähig ist.



(Abbildung 3.1-1 – Oberfläche des Prototyps im Mathematica Notebook)

CellTrack [Sacan2008] – Frei verfügbares, plattformunabhängiges, Open Source Software Packet. Es funktioniert komplett automatisch und hat somit die gleichen Probleme wie die bereits erwähnte automatische Vorverarbeitung. Hinzu kommt, dass das Programm nur mit bestimmten Arten von Experimenten funktioniert. Nicht jedoch mit den uns vorliegenden Daten (Filmen). Das Software-Paket ermöglicht nicht die Rekonstruktion von Genealogien, welches für uns primäres Ziel ist. Jedoch erlaubt es Erweiterungen durch die offenen Schnittstellen (OpenSource), sodaß der Export der Daten in ein gewünschtes Textformat erfolgen könnte.

ExpertomicaCells [Levitner2009] - Auch hier ist nur eine manuelle Bearbeitung der Filme möglich, dadurch ist die Software nur eingeschränkt nutzbar bei Filmen mit wenigen, langsamen Zellen. Es ergeben sich die gleichen Nachteile wie bei TTT.

Des Weiteren gibt es verschiedene kommerzielle Software (z.B. Imaris Track, Definiens, etc.), die sehr teuer sind, keine offenen Strukturen aufweisen und meist nur für sehr spezifische Anwendungsfälle geeignet sind.

3.2 Auswahl des geeigneten Frameworks

Für die Umsetzung des Programms wurde die Programmiersprache C++ mit dem Framework Qt[1] verwendet. Diese Entscheidung wurde nicht zuletzt wegen den Vorbedingungen gewählt, dass das zu entwickelnde Programm plattformunabhängig und gut erweiterbar sein sollte. Weiterhin auch wegen der in Qt enthaltenden umfangreichen GUI-Klassenbibliothek und der guten und umfassenden Qt Referenz Dokumentation. [Wolf2007, Seite 12]

Nicht nur die beträchtliche Menge der Qt-internen Klassen, sondern auch die Unterstützung der Integration von OpenGL und die Erweiterbarkeit mit vielen anderen C++ Bibliotheken, wie zum Beispiel OpenCV[2] oder VTK[3] waren ausschlaggebend für die Verwendung von Qt mit C++.

Implementiert wurde unter der derzeit aktuellsten Vesion 4.7.1. Für diese Version stellt Qt drei Lizenzen zur Verfügung. Für kommerzielle Entwicklungen ist eine kommerzielle Lizenz erforderlich (Qt Commercial Developer Licence). Open-Source-Programme können unter einer Open-Source-Lizenz (Qt Gnu GPL v.3.0) verwendet werden. Zusätzlich bietet Qt noch die Möglichkeit eine proprietäre Software auch ohne kostenpflichtige Lizenz zu erstellen. Dazu wird eine Qt Gnu LGPL v.2.1 – Lizenz angeboten. [QtLicensing2010]

3.3 Die wichtigsten Problematiken im Vorfeld

Wie bereits erwähnt ist der kritischste Punkt die große Menge an Daten. Die Filme haben im Schnitt eine Länge von 5000-10000 Bilder, wobei in den meisten Fällen 300-500 Kilobyte pro Bild anfallen. Es gibt jedoch auch experimentelle Daten mit einer höheren Auflösung mit mehreren Megabyte pro Bild.

> **Beispiel:** Größenordnung der Datensätze von einem Beispielfilm mit 5000 Bildern.
>
> Insgesammt 160000 automatisch erkannte Zellobjekte (zwischen 20 und 50 erkannte Objekte pro Bild)
>
> - Image-Ordner ca. 1,3 Gigabyte
> - Meta-Ordner ca. 800 Megabyte (XML-Dateien mit den Metadaten zu jedem Bild)
> - Track-Datei 13 Megabyte (XML-Datei mit Listen aller generierten Tracks)

Die zu jedem Bild vorliegende XML-Datei mit den Metadaten (Informationen zu den Zellobjekten) kann von einigen Kilobyte bis zu einem Megabyte groß sein. Die Größe dieser Dateien ist natürlich abhängig von der Anzahl der Objekte pro Bild (also auch abhängig vom jeweiligen Film) sowie der Anzahl der erfassten Eigenschaften pro Objekt. Wobei abzusehen ist, dass die reinen Nutzdaten nur ein Teil davon ausmachen, da mehr Informationen in den Metadaten bereit stehen, als für die Bearbeitung und Visualisierung der Trajektorien benötigt werden (siehe Kapitel 2.2 und 5.1) und die Daten auf Grund des XML-Formats (Tags) einen erheblichen Overhead besitzen.
Da das Ziel eine schnelle und effiziente Bearbeitung ist, mußte hier eine geeignete Lösung für das schnelle Laden und Anzeigen der Bilder sowie Laden und Abspeichern der Metadaten gefunden werden.

---

[1] Qt – Qt ist eine von der norwegischen Firma Trolltech (2008 von Nokia übernommen) entwickelte C++ Klassenbibliothek für die Programmierung von graphischen Benutzeroberflächen (GUI – Graphical User Interface). Die erstellten Anwendungen lassen sich plattformübergreifend nutzen, d.h. auf den verschiedenen Betriebssystemen MS Windows, MacOS und Linux kompilieren. [Blanchette2009, Seite 11]

[2] OpenCV – Open-Source Programmbibliothek mit Algorithmen für die Bildverarbeitung

[3] VTK – (Vizualisation Toolkit) Open-Source-Programmbibliothek die besonders für die 3D-Computergrafik mit einem Augenmerk für die wissenschaftliche Visualisierung geeignet ist [VTK]

Ein weiterer Hauptaspekt ist die Gestaltung des Workflows. Wie soll der Benutzer an den Stellen, wo die Objekt- und Trackerkennungsalgorithmen abbrechen, effizient eingreifen? Wie kann ein unnötiger Eingabeaufwand verhindert werden? Wie können überhaupt problematische Stellen im Film schnell und ökonomisch gefunden werden? Zur Beantwortung dieser Fragen mussten produktive Strategien entwickelt werden, auf die in Kapitel 5.5 näher eingegangen wird.

Ausgehend von den in 3.1 gezeigten, bestehenden Lösungen und den soeben aufgeführten Problematiken, lassen sich nun die Anforderungen für das zu erstellende Programm formulieren. Hiermit beschäftigt sich der nächsten Abschnitt.

3.4 Programmkonzept

Im Folgenden werden die fundamentalen Anforderungen des Programms noch einmal konkret aufgezeigt. Das Gliederungsschema ist angelehnt an den groben Aufbau eines Lastenheftes [Balzert2000, Seite 62 ff].

Zielbestimmung:
Das zu entwickelnde Softwareprodukt *CellTracker*, soll in der Forschung zur Untersuchung von Zelldynamiken speziell zur Analyse von Stammbäumen (d.h. Zellteilungen aber auch Zellsterben) und Zellbewegung angewendet werden. Mit diesem Programm soll eine Nachbearbeitung der bei der automatischen Zellverfolgung entstandenen Fehler deutlich vereinfacht werden. Man soll die Möglichkeit haben, bei Unstimmigkeiten interaktiv einzugreifen und zu korrigieren.
Des Weiteren soll das nachbearbeitete Material in gewünschter Weise dargestellt und für weitere Forschungen und Analysen exportiert werden können.

Produkteinsatz:
Das Produkt soll wissenschaftlichen Mitarbeitern an Hochschulen und Universitäten oder in der industriellen Forschung dienen, die an zellbiologische Fragestellungen bearbeiten. Es wird demzufolge davon ausgegangen, dass geschulte User dieses Programm verwenden, für die die biologische Fragestellung hinter der Programmfunktionalität nicht explizit im Programm erläutert werden muß. Es ist daher wichtiger, dass das Programm ohne viel technisches (informatisches) Hintergrundwissen effizient benutzt werden kann.

Produktfunktionen:
Nachstehend werden typische Arbeitsabläufe genannt und die Hauptfunktionen des Produktes auf oberster Abstraktionsebene beschrieben.

/F10/ **Prozess:** Projekt erstellen
Beschreibung: Der User legt ein neues Projekt an, indem er das Verzeichnis des Bildstapels angibt und die Metadaten sowie die generierten Tracks importiert.

/F20/ **Prozess:** Projekt speichern
Beschreibung: Alle importierten, neu hinzugefügten und sonstige relevanten Daten eines Projektes werden abgespeichert.

/F30/ **Prozess:** Projekt laden
Beschreibung: Um die Bearbeitung fortzusetzen werden alle Daten eines bereits erstellten Projektes geladen.

/F40/ **Prozess:** Bildsequenzen und Metadaten darstellen
Beschreibung: Nachdem ein Projekt erstellt oder geladen wurde sollen die Bilder und Metadaten grafisch dargestellt werden, um die neuen Trajektorien erstellen zu können.

/F50/ **Prozess:** Trajektorien erstellen
Beschreibung: Die generierten Tracks werden in gewünschter Weise

miteinander verknüpft und zu einer neuen Trajektorie hinzugefügt.

/F60/ **Prozess:** Trajektorien exportieren
Beschreibung: Die neu erstellten Trajektorien werden in dem erforderlichem Format (für die Auswertung) gespeichert.

Produktdaten:
Die Hauptdaten des Produktes die permanent gespeichert werden.

/D10/ **Import von Bildstapel:** Die Bilder können in verschiedenen Formaten vorliegen (z.B. jpg, jpeg, bmp, png). Die Größenordnung liegt im Durchschnitt bei 5000-10000 Bilder mit ca. 300-500 Kilobyte pro Bild. Auf Grund des (unterschiedlich) großen Ausmaßes des Bildstapels werden die einzelnen Bilder nicht in den Arbeitsspeicher vorgeladen, es werden nur die Pfade der einzelnen Bildern in einer Liste gespeichert, welche im Arbeitsspeicher gehalten wird.

/D20/ **Import von Metadaten:** Die Daten liegen im XML-Format vor. Diese Metadaten bestehen aus den Daten für die einzelnen Zellobjekte, für die folgende Attribute zu speichern sind: die Zell-Identifikationsnummer (`cellID`), das umschließende Rechteck in dem sich das Zellobjekt befindet (`boundingBox`), die Kontur der Zelle (`polygon`) und die Identifikationsnummer des dazugehörenden generierten Tracks (`generatedTrackID`).

/D30/ **Import von Trackdaten:** Die Trackdaten liegen wie die Metadaten im XML-Format vor.
Diese Trackdaten enthalten Informationen über die einzelnen, vorgenerierten Trajektorien.
Zu speichern sind die Identifikationsnummer (`trackID`), ein Vektor der dazugehörenden Zellen (`cellIDs`), die Länge des Tracks (`trackLength`) sowie die Startzeit (`startTime`) und Endzeit (`endTime`).

/**D40/** **Speichern von Projekten:** Alle importierten, geladenen und neu berechneten Daten sollen binär in der CellTracker-Projekt-Datei (.ctp) für ein späteres Weiterbearbeiten abgespeichert werden.
Dazu müssen projektspezifische Daten gespeichert werden, wie Name des Projektes / Dateiname (`projectName`), das Arbeitsverzeichnis (`projectDirectory`), der Status des Projektes (`projectStatus`), die Pfade zu den importierten Bild- und Metadaten (`imgDirectory`, `metaDirectory`), sowie den Pfad zur generierten Track-Datei (`trackFile`).
Weiterhin sind zu einem Projekt die importierten Metadaten, folglich alle Zellobjekte (`cells`), die importierten Trackdaten also die generierten Tracks (`generatedTracks`) und die neu erstellten Trajektorien (`newTracks`) zu speichern.

/D50/ **Export von neu erstellten Tracks:** Vereinbart ist der Export in eine XML-Datei mit einer ähnlichen Struktur wie die XML-Datei der zu importierenden Trackdaten aus Mathematica. Jedoch erweitert um zusätzliche Informationen die für die Rekonstruktion der Genealogien erforderlich sind (ID der Mutterzelle, Ende des Trackings auf Grund von: Teilung/Tod/Verlust der Zelle etc.).

Produktleistungen:

Das Darstellen der Bildsequenzen (/F40/) soll flüssig ablaufen, sodaß eine benutzerfreundliche Bedienung gewährleistet werden kann.

Die Daten aus /D10/, /D20/ und /D30/ müssen im Arbeitsspeicher gehalten werden, da der Zugriff auf sie wahlweise ist und schnell erfolgen muß.

Die Daten aus /D50/ müssen in einem validen XML Format gespeichert werden.

Qualitätsanforderungen:
Die wichtigsten Qualitätsanforderungen und die geforderte Qualitätsstufe im Überblick.

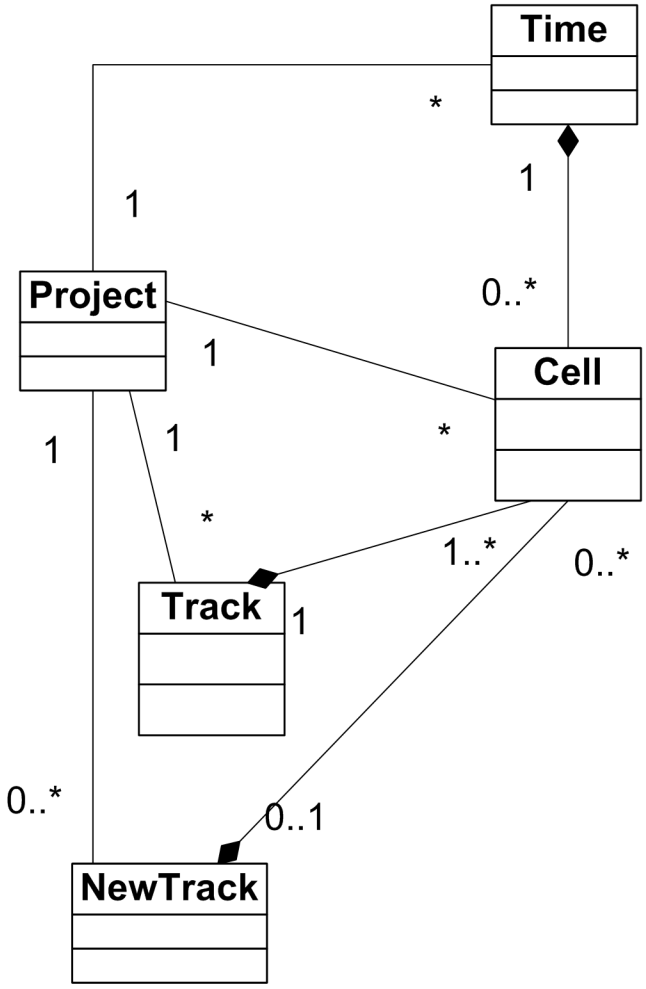
Produktqualität	entscheidend	wichtig	normal	nicht relevant
Funktionalität		**x**		
Zuverlässigkeit		**x**		
Erlernbarkeit			**x**	
Bedienbarkeit	**x**			
Effizienz		**x**		
Modifizierbarkeit		**x**		
Übertragbarkeit		**x**		
Datensicherheit				**x**

(Tabelle mit den wichtigsten Qualitätsanforderungen)

4. Modellierung (UML-Diagramme)

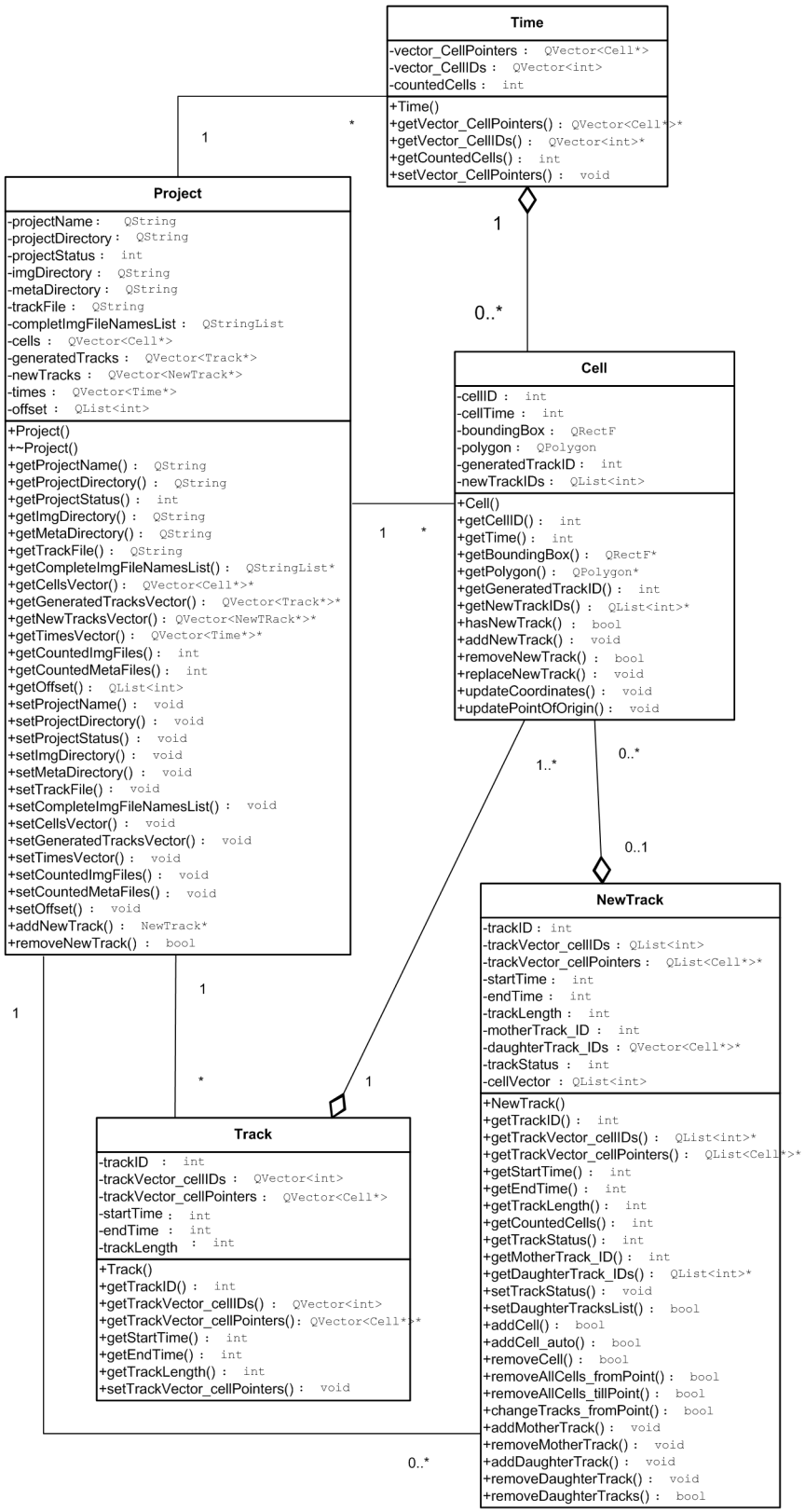
Um einen kurzen Überblick über den Entwurf der objektorientierten Software zu verschaffen, sind nachfolgend die wichtigsten Klassen des Programms *CellTracker* als UML[1]- Klassendiagramm graphisch dargestellt. Eine detaillierte Beschreibung dieser Klassen erfolgt im Kapitel 5.2.

4.1 Die Klassen `Project`, `Cell`, `Track`, `NewTrack`, `Time`



(Abbildung 4.1.-1 – Klassen und deren Assoziationen)

[1]UML – „UML (Unified Modeling Language) ist eine grafische Notation zur Erstellung objektorientierter Modelle für die Analyse und den Entwurf von objektorientierter Software.“ [Balzert2005, Vorwort]



(Abbildung 4.1-2 – Klassen mit Attribute und Methoden)

5. Implementierung

In diesem Kapitel wird näher auf die konkrete Implementierung des CellTracker Programms, den verwendeten Klassen und Algorithmen eingegangen. Die Klassen für die grafische Oberfläche werden hier jedoch nicht näher behandelt, da diese zum größten Teil von den in der Qt-Bibliothek befindlichen GUI-Klassen erben. Struktur und Implementierung dieser Klassen ist auf der beiligenden CD unter CellTracker/src/gui ersichtlich.

5.1 Import der Metadaten/Trackdaten

Der Import der Daten, wird mit Threads realisiert. Dies geschieht mit Hilfe der Klasse `QThread` aus der Qt-Bibliothek. Der Import beinhaltet neben dem Identifizieren der Bilddateien und deren Pfade, das Auslesen der zu jedem Bild gehörigen Meta-Datei (Kapitel 2 – Phase 1, automatische Zellerkennung / Objekterkennung) und das Auslesen der Track-Datei (Kapitel 2 – Phase 2, automatische Objektverfolgung). Da diese Operationen weitestgehend unabhängig voneinander sind kann mit Multithreads, bei Rechnern mit mehreren Prozessoren, ein Großteil dieser Berechnungen parallel ausgeführt werden.
Ein weiterer Vorteil der Verwendung von Threads an dieser Stelle ist, dass man bei der Ausführung der Operationen (welche auf Grund der großen Menge der zu importierenden Daten sehr umfangreich ist) das „Einfrieren“ der GUI (hier die ProgressDialog-Anzeige) vermeidet. [Wolf2007, Seite 554]

Um die Ressourcen des Arbeitsspeichers nicht unnötig auszulasten, werden nur die notwendigen Daten, die für die Programmfunktionalität erforderlich sind (Daten für Zelldarstellung, Daten für Genealogieerzeugung), aus den Meta-Dateien importiert (siehe auch Kapitel 2.2 vorhandene Daten und 3.4 Produktdaten /D20/). Falls nachträglich noch Daten benötigt werden (z.B. für Analyse, etc.) können diese hinterher immer noch aus den gegebenen Metadaten importiert werden.

Die Meta-Dateien und die Track-Datei liegen im XML-Format vor (siehe auch Abbildungen 2.3-1 und 2.3-2). Das Parsen der XML-Dateien wurde mit der – im QtXml-Modul enthaltenden – API DOM[1] realisiert.

Da für die Zukunft geplant ist den Import der Meta-Daten auf andere, besser geeignete Formate zu erweitern (siehe 8.3 Ausblicke – Andere Datenformate), wurde für die Implementierung des Imports die abstrakte Oberklasse `InputOutput` angelegt. Diese stellt die benötigten Methodensignaturen für die jeweiligen, unterschiedlichen Import-Formate bereit. Die von dieser Oberklasse abgeleiteten Klassen (z.Z. nur die Klasse `InputOutputXml`) überschreiben und implementieren dann die vererbten abstrakten Methoden.

5.2 Interne Datenverwaltung

Da die in der Qt-Bibliothek vorhandenen Methoden eine ausreichend schnelle Bidlanzeige erreichen, ist es nicht nötig die Bilder auf eine spezielle Art und Weise vorzuladen. Deshalb werden die Bilder nicht importiert, sondern nur die Pfade der Dateien sequentiell in der Klasse `Project` in einer Liste (`completeImgFileNamesList` vom Typ `QStringList`) gespeichert. Vorbedingung dafür ist, dass die zu importierenden Bilder geordnet in einem separaten Ordner vorliegen. Das heißt, dass die Dateinamen der Bilder keine Rolle spielen, nur die Position innerhalb des Ordners (zum Beispiel gehört die vierte Datei im Image-Ordner zur vierten

[1] DOM – „DOM (Document Object Model) repräsentiert ein XML-Dokument als Baumstruktur und gewährt wahlfreien Zugriff auf die einzelnen Bestandteile…“ [Wolf2007, Seite 705]

Datei im Meta-Ordner). Dies gilt immer als gegeben, wegen der in der Vorbearbeitungsphase (Kapitel 2.1 - Objekterkennung) getroffenen Vereinbarungen.

Um einen Zeitrafferfilm zu bearbeiten werden die benötigten Metadaten und die Trackdaten komplett in den Arbeitsspeicher geladen. Dies erfolgt einmal beim Anlegen eines neuen Projektes (Import) oder wenn ein bereits vorhandenes Projekt geladen werden soll.
Gespeichert werden die Metadaten (Daten für die Zellobjekte) in den jeweils dafür angelegten Objekten der Klasse `Cell` und die Trackdaten in den dafür angelegten Objekten der Klasse `Track` (siehe Abbildung 4.1-1). Um ein schnelles Zeichnen der Zellen und somit ein besseres Verhalten beim Navigieren im Bildstapel zu erzielen, wird zusätzlich zu jedem Zeitpunkt eine Liste mit Pointern auf die zum Zeitpunkt vorhandenen Zellobjekte angelegt. Eine solche Liste und die Operationen darauf wird jeweils durch ein Objekt der Klasse `Time` realisiert. Letztlich werden die neu gewonnen Daten, also die neu erstellten Trajektorien und deren Abhängigkeiten (Genealogien), als Objekte der Klasse `NewTrack` gespeichert. (Siehe auch Abbildung 4.1-1)
Im Folgenden wird die Strukturierung und Implementierung dieser Objekte näher beschrieben.

Objekte der Klasse `Cell` (Zellobjekte)

Ein Zellobjekt wird realisiert durch ein Objekt der Klasse `Cell`. Ein jedes Zellobjekt besitzt eine eindeutige Zell-Identifikationsnummer (`cellID`), das umschließende Rechteck, in dem sich das Zellobjekt befindet (`boundingBox`), die Kontur / Outline der Zelle (`polygon`) und die Identifikationsnummer des dazugehörenden vorgenerierten Tracks (`generatedTrackID`).
Wenn beim Erstellen einer neuen Trajektorie ein Zellobjekt hinzugefügt wird, wird zusätzlich die Track-Identifikationsnummer (`trackID`) der momentan bearbeiteten Trajektorie im Attribut `newTrackIDs` gespeichert (vom Typ `QList<int>`). Es handelt sich hierbei um eine Liste, damit eine Zelle mehreren Trajektorien zugewiesen werden kann, da bei der automatischen Zellerkennung bei sehr dicht liegenden Zellobjekten vereinzelt ein großes Zellobjekt an Stelle von zwei Objekten erkannt wird.

Die Methoden dieser Klasse lassen sich in drei unterschiedliche Aufgabenbereiche teilen. Zum einen Methoden die für die grafische Darstellung die Attribute des Objektes auslesen: `getCellID()`, `getTime()`, `getBoundingBox()`, `getPolygon()` und `getGeneratedTrackID()`. Zum anderen Methoden die Funktionen für Track-Operationen bereitstellen: `getNewTrackIDs()`, `hasNewTrack()`, `addNewTrack()`, `removeNewTrack()` und `replaceNewTrack()`. Letztlich noch zwei Methoden die Funktionen für eventuelle Koordinatenverschiebungen (Offset) oder Koordinatenursprungstransformationen bereitstellen: `updateCoordinates()` und `updatePointOfOrigin()`. Diese berechnen neue Koordinaten für die Attribute `boundingBox` und `polygon`. (Details siehe 5.4)

Um die Objekte dieser Klasse referenzieren zu können, werden die Pointer auf diese Objekte in der Qt eigenen Container-Klasse `QVector` gespeichert. Genau gesagt in der Klasse `Project` als Attribut `cells` mit dem Typ `QVector<Cell*>`. (siehe Abbildung 4.1-1)

Objekte der Klasse `Track` (vorgenerierte Tracks)

Diese Objekte stehen stellvertretend für die in der Phase der automatischen Objekterkennung entstandenen vorgenerierten Tracks. Zu einem solchen Objekt gehört jeweils eine Identifikationsnummer (`trackID`), ein Vektor der zum generierten Track dazugehörenden Zellen (`trackVector_cellPointers` bzw. `trackVector_cellIDs`), die Länge des Tracks (`trackLength`) sowie die Startzeit (`startTime`) und Endzeit (`endTime`).

Um auf die Objekte der Klasse `Track` zugreifen zu können, werden die Pointer im Projekt (`Project`) als Attribut `generatedTracks` mit dem Typ `Qvector<Track*>` gespeichert. (siehe Abbildung 4.1-1)

Objekte der Klasse `Time` (Zellobjekte zu einem Zeitpunkt)

Ein Objekt der Klasse `Time` repräsentiert einen bestimmten Zeitpunkt. Ein solches Objekt enthält eine Liste mit den Speicheradressen (`vector_cellPointers`) der zu diesem Zeitpunkt vorhandenen Zellobjekte. Diese zusätzliche Datenstruktur wurde geschaffen, um die zu einem bestimmten Zeitpunkt vorhandenen Zellobjekte schnell zu identifizieren, damit das Einzeichen dieser Zellobjekte ohne großartige Verzögerung geschehen kann.
Gespeichert werden die Pointer dieser Objekte in der Klasse `Project` als Attribut `times` mit dem Typ `QVector<Time*>`. (sieh Abbildung 4.1-1)

Objekte der Klasse `NewTrack` (neu erstellte Tracks)

Diese Klasse stellt eine Datenstruktur zum Speichern der neu erstellten Trajektorien samt ihren Genealogien bereit. Diese Objekte besitzen eine eindeutige Identifikationsnummer (`trackID`), eine Liste mit den dazugehörenden Zellen (`trackVector_cellPointers` bzw. `trackVector_cellIDs`), die Länge des Tracks (`trackLength`) sowie die Startzeit (`startTime`) und Endzeit (`endTime`).
Neben den allgemeinen getter-Methoden, die vorrangig von GUI-Klassen benutzt werden, werden Methoden für Track-Operationen (hinzufügen und löschen von Zellobjekten) zur Verfügung gestellt. Derartige Methoden sind `addCell()`, `addCell_auto()`, `removeCell()`, `removeAllCells_fromPoint()`, `removeAllCells_tillPoint()` und `changeTracks_fromPoint()`.
Für die Assoziationen (Genealogien) der neu erstellten Tracks untereinander werden folgende Methoden bereitgestellt: `setDaughterTrackList()`, `addMotherTrack()`, `removeMotherTrack()`, `addDaughterTrack()`, `removeDaughterTrack()`, `removeDaughterTracks()`.
Um die Objekte dieser Klasse referenzieren zu können, werden die Pointer im Projekt (`Project`) als Attribut `newTracks` mit dem Typ `QList<NewTrack*>` gespeichert. (sieh Abbildung 4.1-1)
Warum zum auflisten der dazugehörenden Zellen, wie auch für das Referenzieren der Objekte dieser Klasse die Container-Klasse `QList` an Stelle von `QVector` verwendet wurde liegt daran, das mit `QVector` die Elemente in einer Reihe, also im Speicher aneinanderliegende Adressen positioniert werden. Daraus folgt, dass Schreib-Operationen, die nicht am Ende einer solchen Liste durchgeführt werden, sehr langsam sind. `QList` hingegen ist Index-basiert, wodurch der Zugriff auf ein Element sehr schnell ist. `QList` verwendet ein Array von Zeigern, die auf die Objekte verweisen. Auf diese Weise geschieht das Einfügen insbesondere am Anfang und Ende einer solchen Liste sehr schnell. [Wolf2007, Seite 606]

5.3 Externe Datenverwaltung

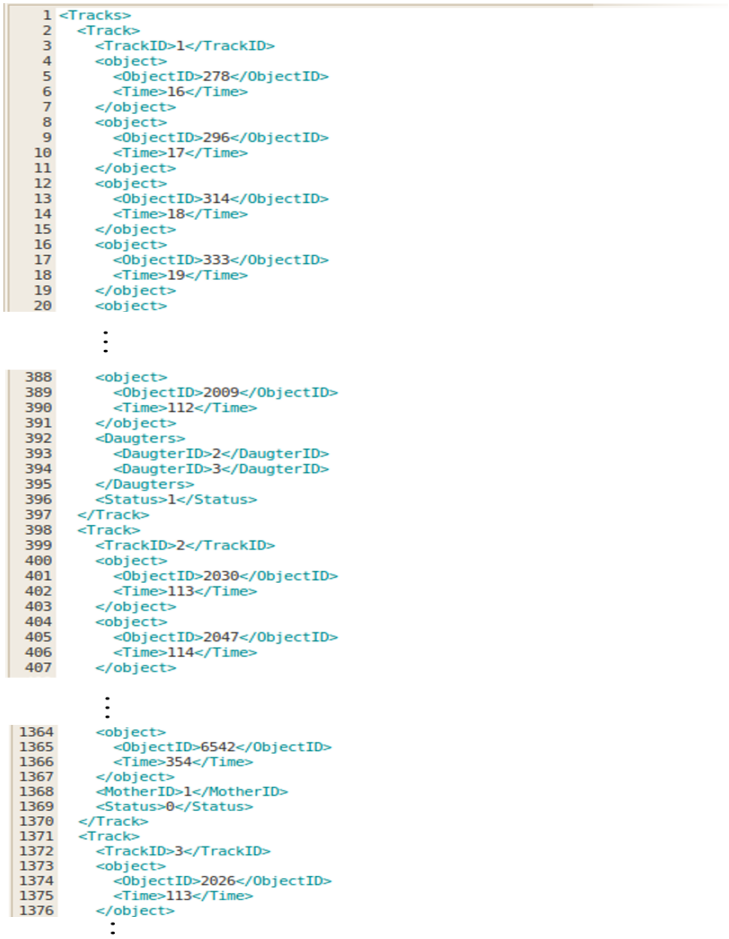
Speichern und laden des Projektes

Das gesamte Projekt (bzw. die Attribute der Klasse `Project`) wird binär in eine Datei gespeichert (CellTracker Projektdatei – „projektname.ctp“). Das Speichern/Schreiben erfolgt unter Verwendung eines Streams der Klasse `QDataStream`.
Das Laden/Lesen der Daten erfolgt analog zum Schreiben mithilfe der Klasse `QDataStream`. [siehe auch Blanchette2009, S. 113]
Da aus Effizienzgründen die Zuordnung von Zellobjekten (`Cell`) zu Zeitpunkten (`Time`), generierten Trajektorien (`Track`) oder neu erstellten Trajektorien (`NewTrack`) mit

Pointern realisiert wurde, müssen beim Speichern des Projektes an Stelle der Speicheradressen der Zellobjekte die `cellID`'s gespeichert werden.
Dieses Problem wurde durch Anlegen eines zusätzlichen Vektors (`vector_cellIDs`) gelöst. Kurz vor jedem Speichervorgang werden in diesem Vektor dann die IDs der Zellobjekte geschrieben, auf die im Vektor für die Zellobjekt-Pointer (`vector_cellPointers`) referenziert wird. Dies geschieht in den Klassen `Time`, `Track` und `NewTrack` (siehe Abbildung 4.1-1).
Da nach dem Laden eines Projektes die Speicheradressen dem Projekt wieder bekannt sein müssen, werden analog zum Speichern aus den `cellID`'s die Referenzen auf die Zellobjekte erzeugt. Folglich muß die Reihenfolge (Hierarchie) beim Laden beachtet werden. Erst die Objekte, dann die Pointer auf die Objekte.

Neu erstellte Tracks exportieren

Da die neu erstellten Trajektorien als Metadaten für die weitere Verarbeitung und Auswertung zur Verfügung gestellt werden sollen, wurde hierfür das XML-Format ausgewählt. Zumal diese Struktur austauschbar und für Menschen lesbar (Human-readable) also für andere Benutzer transparent ist.
Die zu exportierenden Daten wurden analog zum Import der Trackdaten mit der API DOM und unter Verwendung eines Streams der Klasse `QTextStream` in die gewünschte Datei geschrieben.
Struktur der XML-Datei ähnelt der zu importierenden Track-Datei (siehe Abb. 2.2-2). Allerdings erweitert mit den zusätzlichen Informationen die für die Rekonstruktion der Genealogien erforderlich sind.



(Abbildung 5.4-1 – Ausszug XML-Datei)

Die Abbildung 5.4-1 zeigt den typischen Aufbau einer exportierten Tracking-Datei. Man sieht die Erweiterung um die Tags <MotherID>, <DaughterIDs> und <Status>.

5.4 Algorithmen für Workflow

Einstellen des Offsets und des Koordinatenursprungs (KOU)

Ein Problem, welches sich nach dem Import der Metadaten ergeben kann, ist die eventuell auftretende Verschiebung der Koordinaten der Zellobjekte gegenüber den Original-Bilddaten. Dies kann während der Vorverarbeitungsphase durch das Abschneiden der Bildränder nach der Bildregestrierung entstehen.
Zusätzlich sind die Zellobjekte seitenverkehrt angeordnet, da Mathematica (mit dem die Berechnungen für die Objekterkennung durchgeführt wurden) und das Qt-Framework von unterschiedlichen Koordinatenursprüngen (KOU's) ausgehen.
Aus diesem Grund mussten effiziente Algorithmen zum Ausgleichen der Koordinatenverschiebungen und für die KOU-Transformationen entworfen werden.
Da im Vordergrund des Programms eine gute Benutzbarkeit steht, wird die Möglichkeit gegeben die Verschiebung und die Transformation der Koordinaten gleichzeitig einzustellen. Dies wiederum stellt einen höheren Anspruch an die verwendeten Algorithmen, da außerdem eine passende Datenstruktur zum Speichern der Veränderungen entworfen werden musste.

Die Koordinatenverschiebung und Koordinatenursprungstransformation werden nacheinander in einer Liste (`QList<int> offset`, siehe Klasse `Project`) gespeichert.

Index `i`	Value	
0	0	→ x-offset
1	53	→ verschieben der x-Werte um +53
2	1	→ y-offset
3	-30	→ verschieben der y-Werte um -30
4	2	→ Koordinatenursprungstransformation
5	3	→ KOU- Transformation (neuer KOU: links-unten)

(Abbildung 5.4-2 – Beispielliste)

Jeder Wert an den ungeraden Stellen ($i \bmod 2 = 1$ gehört zu dem Wert `i`-1. Die Werte an den geraden Stellen ($i \bmod 2 = 0$ entsprechen den einzelnen Operationen, wobei 0 die Verschiebung der x-Koordinaten, 1 die Verschiebung der y-Koordinaten und 2 die KOU-Transformation bedeutet. Die vier möglichen KOU's werden im Urzeigersinn angefangen, mit 0 (oben links) durchnummeriert.

Die eigentlichen Algorithmen zur Berechnung der Verschiebung und der Transformation befinden sich in der Klasse `Cell`, da für jedes Zellobjekt die Umrechnungen der Koordinaten separat ausgeführt werden müssen. Die entsprechenden Methoden dazu sind `updateCoordinates()` und `updatePointOfOrigin()` (siehe Abbildung 4.1-2).

Unter anderem hat man somit auch die Möglichkeit die Transformationen (angewandten Operationen) Schritt für Schritt rückgängig zu machen. (Wiederherstellen des Anfangszustands)

Grafische Darstellung der einzelnen Zellobjekte

Um die einzelnen Zellobjekte eines Zeitpunktes `t` einzeichnen zu können, muss zu jedem Zellobjekt ein Grafik-Objekt angelegt werden. Diese Objekte, der Klasse `WorkingArea_View_CellGraphicsItem` (die von der Klasse `QGraphicsObject` erbt), werden zur Laufzeit angelegt. Zusammen mit dem zum Zeitpunkt gehörenden Bild werden diese Objekte dann zum Zeichenbereich (Scene) hinzugefügt.
Da ein schnelles Navigieren durch den Bildstapel gewünscht ist, wird der komplette Zeichenbereich mit einem Mal gelöscht. Dafür müssen jedoch jedes Mal die Grafik-Objekte der Zellen neu erstellt werden, d.h. Laufzteit angelegt werden.

Die Klasse `WorkingArea_View_CellGraphicsItem` bietet außerdem die Methoden, die das Auswählen/Selektieren der Zellobjekte mit der Maus erlauben. Hierfür werden die geerbeten Methoden `mousePressEvent()`, `hoverEnterEvent()` und `hoverLeaveEvent()` reimplementiert. Die Information, ob ein Zellobjekt selektiert wurde, wird im Attribut `itemSelected` (Typ `bool`) gespeichert.
Um effizienter arbeiten zu können ist ein Zellobjekt dann selektiert, wenn es entweder mit der Maus angeklickt wurde, oder sich der Mauszeiger über dem Objekt befindet (Mousehover).

Hinzufügen von Zellen (`addCell()`)

Die Implementierung für das Hinzufügen eines Zellobjektes zu einem neuen Track erfolgt in der Methode `addCell(Cell *cell) in` der Klasse `NewTrack`. Als Parameter wird der Methode ein Pointer des Zellobjekts (`Cell`) übergeben, der in die Liste `trackVector_cellPointers` (vom Typ `QList<Cell*>`) hinzugefügt werden soll.
Ein neuer Track wird repräsentiert durch eine Liste von Zellobjekten, d.h. durch eine Liste von Pointern, die auf die Zellobjekte zeigen. Das erste Zellobjekt (mit Zeitpunkt t_0) eines solchen Tracks steht immer am Anfang dieser Liste (Index `i=0`).
Beim Hinzufügen weiterer Zellobjekte wird jeweils unterschieden, ob das neue Zellobjekt vor dem ersten oder nach dem letzten Element zur Liste hinzugefügt werden soll. Wird ein Zellobjekt zum Zeitpunkt $t_n > t_0$ zu dieser Liste hinzugefügt, so geschieht dies an der Stelle $i = t_n - t_0$. Für die Stellen j (mit $i_{last} < j < t_n - t_0$) wird die Liste mit einem NULL-Pointer aufgefüllt. Für den Fall, das ein Zellobjekt mit Zeitpunkt $t_n < t_0$ zur Liste hinzugefügt werden soll, wird wenn $t_n < t_0 - 1$ ein Null-Pointer für jeden Zeitpunkt j (mit $t_0 - t_n > j > t_0$) an den Anfang der Liste angefügt. Anschließend wird noch der eigentliche Pointer auf das gewünschte Zellobjekt an den Anfang der Liste angehängt. Nicht definierte Stellen, d.h. Stellen an denen keine Zellobjekte zugewiesen wurden, sind also mit NULL-Pointern belegt. Falls in der Liste schon ein Wert an der gewünschten Stelle existiert wird dieser einfach überschrieben. Aus diesem Grund kann einer Trajektorie zum Zeitpunkt `t` immer nur ein Zellobjekt zugewiesen werden.

Diese Methode (`addCell()`) bietet die Grundfunktion zum Hinzufügen von Zellen und ist die Basis für die Strategien „click and step“, „hover and step“, „click and jump“ und „click and spin“. Mit dieser Methode könnten die Filme auch ohne die vorgenerierten Trajektorien manuell getrackt werden (es müssen jedoch die erkannten Einzelobjekte vorliegen).

Automatisches hinzufügen von Zellen (`addCell_auto()`)

Sobald ein Zellobjekt mit der Maus angeklickt wird, wird die zu diesem Objekt gehörende generierte Trajektorie farblich markiert. Alle Zellobjekte dieser ausgewählten generierten Trajektorie werden nun gelb dargestellt.

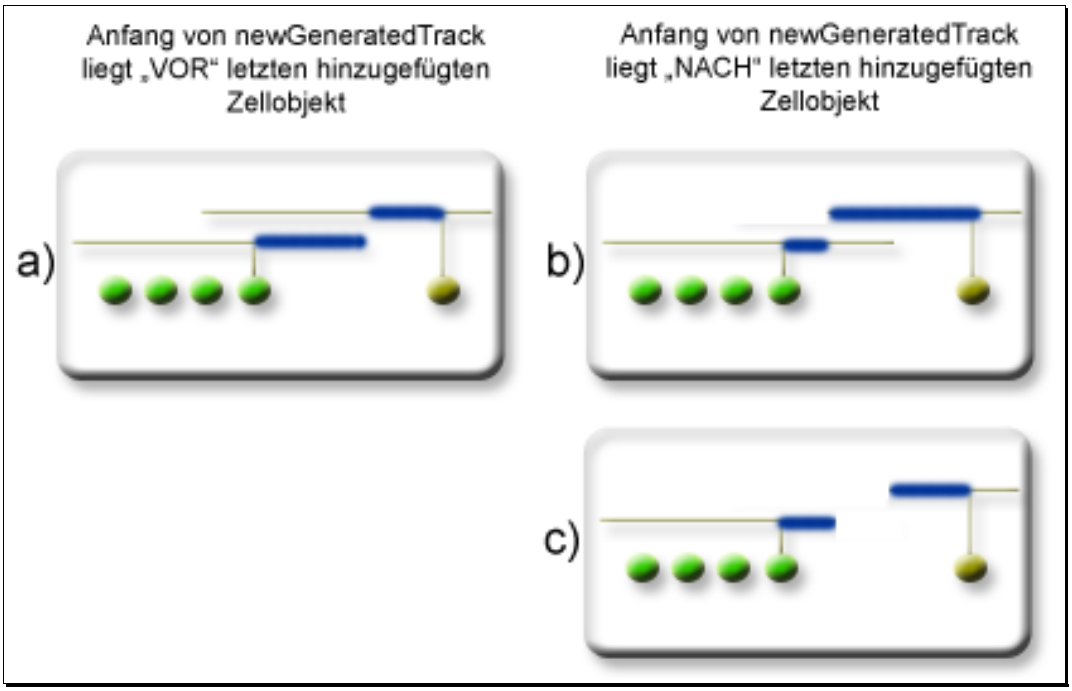
Die Methode `addCell_auto(Cell *cell,QVector<Track*> *generatedTracks)`, der Klasse `NewTrack` benutzt die generierten Trajektorien zum erstellen der Zelltrajektorien. Als Parameter werden der Methode ein Pointer des angeklickten Zellobjekts (`Cell`), und ein Pointer auf den Vektor (`generatedTracks`), der alle generierten Trajektorien enthält, übergeben.

Die generierten Trajektorien werden miteinander verkettet und der Liste `trackVector_cellPointers` (vom Typ `QList<Cell*>`) hinzugefügt. Voraussetzung dafür ist, dass mindestens ein Element in der Liste existiert. Im Falle einer leeren Liste wird nur das selektierte Zellobjekt hinzugefügt. Eine weitere Bedingung ist, dass das selektierte Zellobjekt zeitlich nach dem letzten Element (Zellobjekt) in der Liste steht.

Nun werden die generierten Trajektorien bestimmt, die miteinander verknüpft werden

sollen. Einmal die zum letzten Element gehörende und dann die zum ausgewählten Zellobjekt gehörende Trajektorie. Die generierte Trajektorie des letzten Elements wird nachfolgend als oldGeneratedTrack und die des ausgewählten Zellobjekts mit newGeneratedTrack bezeichnet.

Jetzt entscheidet der Algorithmus, ob der Anfang von newGeneratedTrack vor oder nach dem Vorkommen des letzten Elements in der Liste liegt. Bei Ersterem kann davon ausgegangen werden, dass eine Verwechslung der generierten Tracks (zwischen dem Zeitpunkt des letzten Elements und dem des ausgewählten Zellobjekts) vorliegt. Auf Grund der chronologischen Bearbeitungsweise (der Bildstapel wird in sequentieller Abfolge angeschaut) würde man eine Verwechslung bemerken, die weit vor dem Zeitpunkt des ausgewählten Zellobjekts geschieht. Deshalb werden in diesem Fall die restlichen Zellobjekte des oldGeneratedTrack (maximal jedoch bis zum ausgewähltem Zellobjekt) zur Liste hinzugefügt. Falls der Anfang von newGeneratedTrack nach dem Vorkommen des letzten Elements in der Liste liegt, werden die restlichen Zellobjekte des oldGeneratedTrack (maximal jedoch bis zum Anfang des newGeneratedTrack) zur Liste hinzugefügt. Eine eventuell auftretende Lücke (siehe Abbildung 5.4-3) wird mit Null-Pointern aufgefüllt. Anschließend werden die Zellobjekte von Anfang des newGeneratedTracks bis zum ausgewählten Zellobjekt zur Liste hinzugefügt. Dieser Unterschied erlaubt ein besseres Handling beim Aneinanderreihen der generierten Trajektorien.



(Abbildung 5.4-3 – Möglichkeiten des Hinzufügens)

Die grün dargestellten Objekte sind Zellobjekte, die bereits zur Liste hinzugefügt wurden. Das gelbe Objekt stellt das ausgewählte Zellobjekt dar. Mit Blau markiert sind die Abschnitte der generierten Trajektorien, die zur Liste hinzugefügt werden.

Löschen von Zellobjekten

Für das Löschen von Zellobjekten aus der Liste `trackVector_cellPointers` stehen verschiedene Methoden bereit (`removeCell()`, `removeAllCells_fromPoint()`, `removeAllCells_fromPoint()`), die alle einfache Listenoperationen ausführen.
Beim Löschen eines Zellobjektes Anfang / Ende einer Liste wird einfach das erste / letzte

Element der Liste entfernt. Geschieht das Löschen innerhalb einer Liste, wird an der Stelle der Pointer des zu löschenden Zellobjekts mit einem NULL-Pointer ersetzt. Wenn die neu erstellte Trajektorie eine Lücke (Stellen aufgefüllt mit NULL-Pointern) besitzt und das zu entfernende Zellobjekt direkt nach dieser Lücke kommt (und das letzte Objekt ist), dann werden die Null-Pointer auch aus der Liste entfernt. Analog dazu geschieht das Entfernen der Nullpointer, wenn das zu entfernende Zellobjekt das erste der Liste ist.

Effektives Arbeiten unter Verwendung entworfener Strategien

Im Folgenden werden Strategien vorgestellt, die für eine schnelle und effiziente Bearbeitung der Trajektorien entwickelt wurden. Im Vordergrund stand dabei immer ein effektives Handling unter der Prämisse: „So wenig Klicks wie möglich“.
Intern werden die einzelnen Strategien mittels Threads umgesetzt. Um bei rechenintensiven Operationen ein flüssiges Arbeiten zu gewährleisten.(Sourcecode siehe CellTracker/src/logic/strategie_logic_threads.cpp auf der beiliegenden CD)

Strategie 1: **„click and step“** – Das Zellobjekt welches hinzugefügt werden soll wird mit der Maus angeklickt. Das Zellobjekt wird in der Liste `trackVector_cellPointers` gespeichert und das Bild springt automatisch eins weiter. Bereits an dieser Stelle vorhandene Zellobjekte werden ersetzt.
Optional kann das gleiche Ergebnis erzielt werden, wenn der Mauszeiger über dem gewünschten Zellobjekt positioniert und anschließend die Space-Taste gedrückt wird.
Diese Strategie ist implementiert in der Klasse `ExecStrategy_1_Thread`. Intern wird die Methode `addCell()` der Klasse `NewTrack` benutzt, um die einzelnen Zellobjekte der gewünschten Trajektorie hinzuzufügen.

Da bereits Methoden existieren, die ein Zellobjekt selektieren wenn man den Mauszeiger über das Objekt bewegt (siehe auch 5.4 Grafische Darstellung der einzelnen Zellobjekte, `hoverEnterEvent()` und `hoverLeaveEvent()`), stellt sich folgende Frage: Wieso sollte man jetzt noch mit der Maus klicken? Den nächsten logischen Schritt zur Vereinfachung stellt die Strategie „hover and step“ dar, auf die im Folgenden näher eingegangen wird.

Strategie 2: **„hover and step“** – Hierbei wird nur der Mauszeiger über die Zelle die hinzugefügt werden soll positioniert. Automatisch wird dann das darunterliegende Zellobjekt in der Liste `trackVector_cellPointers` gespeichert und im Anschluss das nächste Bild angezeigt.
Auf Grund unterschiedlicher Rechnerleistungen und den daraus folgenden unterschiedlichen Zeiten, zum Laden des nächsten Bildes und Zeichnen der Zellobjekte , ist ein optionales Delay erforderlich. Hier kann eingestellt werden, nach wieviel Millisekunden ein Zellobjekt hinzugefügt werden soll.
Zusätzlich wurde extra für diese Strategie die Möglichkeit zum Einstellen der Maussensibilität für die einzelnen Zellobjekte implementiert. Damit hat man die Möglichkeit den Bereich um ein Zellobjekt zu vergrößern um dadurch das Selektieren eines Zellobjektes zu erleichtern.
Bereits zu einer anderen Zelltrajektorie hinzugefügten Zellobjekte werden zwar farblich anders dargestellt und mit einer Nummer markiert. Zellobjekte können jedoch zu mehreren Zelltrajektorien gehören (siehe auch 5.2 – Objekte der Klasse Cell). Dies und die Tatsache, dass diese Strategie sehr schnell hintereinander Zellobjekte zu einer Zelltrajektorie hinzufügen kann, kann eventuell zu Fehlern führen. Die Einstellungs-möglichkeit „no double“ läßt keine Doppelbelegung mit einem bereits zu einer anderen Zelltrajektorie hinzugefügten Zellobjekt zu. Damit wird auch ein versehentliches Vertauschen bei dicht beieinander liegenden Zellobjekten verhindert.
Diese Strategie ist implementiert in der Klasse `ExecStrategy_2_Thread`. Intern wird zum hinzufügen der einzelnen Zellobjekte wie bei der Strategie „click and step“ die Methode `addCell()` der Klasse `NewTrack` benutzt.

Da die Strategien „click and step“ und „hover and step“ zwar sehr wirkungsvoll und

erfolgreich sind, nicht jedoch die extra vorgenerierten Trajektorien nutzen, wurden zwei weitere Strategien umgesetzt.

Strategie 3: **„click and jump“** – Diese Vorgehensweise entspricht einem Aneinanderreihen der generierten Trajektorien. Das Zellobjekt, das zu der gewünschten Trajektorie gehört, wird angeklickt. Jetzt wird festgestellt, ob der Rest der ausgewählten Trajektorie weniger als 4 Zellobjekte enthält. Ist dies der Fall wird das nachfolgende Zellobjekt (vergleichbar der Strategie „click and step“) hinzugefügt. Für alle anderen Trajektorien, mit einem Rest größer 3 werden alle Zellobjekte der ausgewählten generierten Trajektorie zu der Liste `trackVector_cellPointers` hinzugefügt. Angefangen vom angeklickten Zellobjekt bis hin zum Letzten dieser Trajektorie. Realisiert wird dieses mit der Methode `addCell_auto()`. Nachfolgend wird zur Position `EndTime-3` dieser generierten Trajektorie gesprungen und die nachfolgenden Bilder, bis `EndTime+1`, mit einem Delay angezeigt. Hier kann dann eine neue Auswahl getroffen werden.

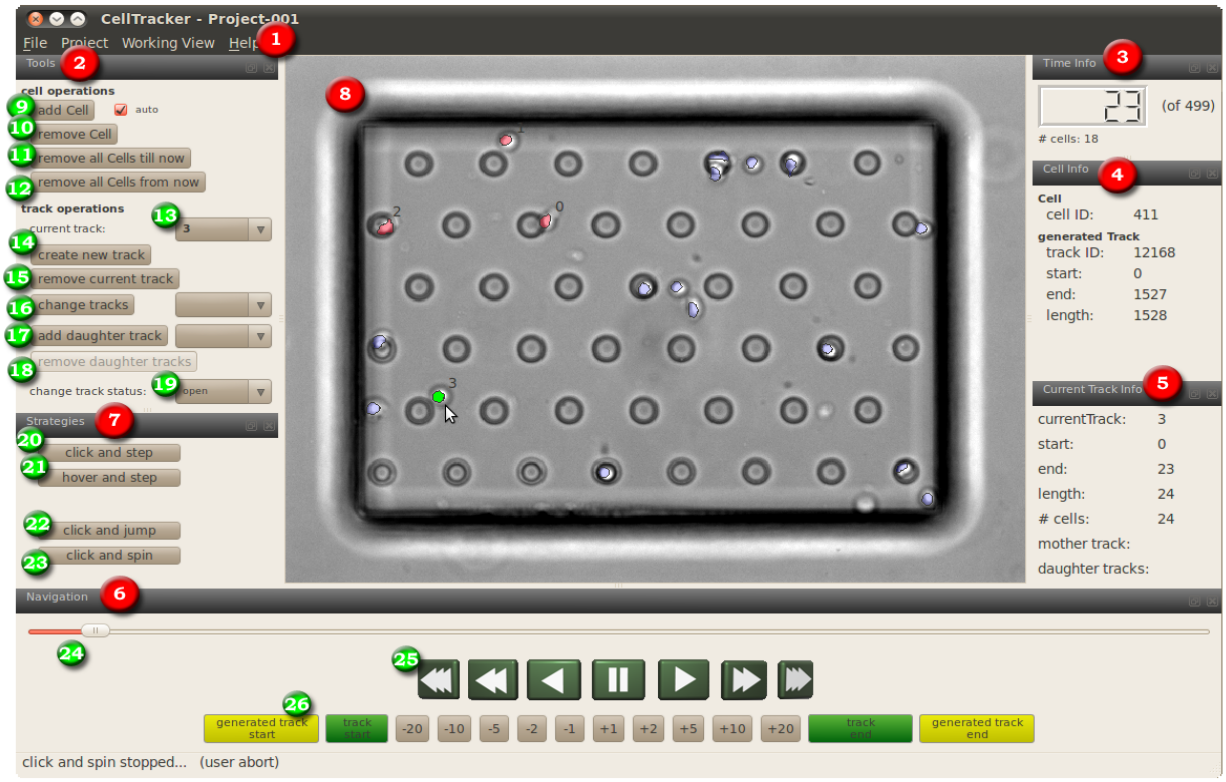
Da bis zum letzten Zellobjekt der generierten Trajektorie die Objekte schon zur Liste hinzugefügt wurden, werden beim Anzeigen der letzten 3 Bilder die dazugehörenden Zellobjekte farblich (grün) dargestellt. Dies erlaubt ein überschaubareres Arbeiten.
Diese Strategie ist implementiert in der Klasse `ExecStrategy_3_Thread`.

Mit dieser Verfahrensweise lassen sich zwar sehr schnell viele Trajektorien miteinander verknüpfen, Vertauschungen (Fehler während der Vorverarbeitung) werden damit jedoch nicht erkannt. Dies erfordert eine weitere Nachbearbeitung.
Dazu wurden Funktionen zum Abspielen des Films implementiert. Die Arbeit könnte somit in zwei unterschiedliche Phasen eingeteilt werden. Erst die generierten Trajektorien zusammenhängen und dann den „Film anschauen“, um die Richtigkeit der einzelnen Zelltrajektorien zu prüfen.
Um diese beiden Phasen zu vereinen wurde weiterhin die Strategie „click and spin“ implementiert.

Strategie 4: **„click and spin“** – Diese Technik ähnelt „click and jump“, erlaubt jedoch ein gleichzeitiges Anschauen des Films (time warp) zwischen gewähltem Zeitpunkt und Ende des vorgenerierten Tracks. Die generierte Trajektorie wird durch Anklicken des Zellobjekts ausgewählt. Darauf wird das nächste Zellobjekt der ausgewählten Trajektorie zur Liste `trackVector_cellPointers` hinzugefügt und das nächste Bild angezeigt. Dieses Hinzufügen und Anzeigen wird automatisch bis zum letzten Zellobjekt wiederholt. Da das Hinzufügen vor der Anzeige des nächsten Bildes geschieht, werden die zur gewünschten Zelltrajektorie dazugehörenden Zellobjekte farblich (grün) dargestellt. Dies erleichtert das Verfolgen der Zelltrajektorie während des „Vorspulens“.
Zusätzlich wird die Möglichkeit gegeben, das Abspielen des Films zu unterbrechen durch Drücken der Space-Taste. Bei Verwechslung von Trajektorien kann somit an dem gewünschten Zeitpunkt abgebrochen werden, um dann mit der „richtigen“ generierten Trajektorie vortzufahren.

Die Interaktionen mit dem Benutzer wurden mit Hilfe dieses Algorithmus auf ein Minimum reduziert. Mit dieser Strategie können Trajektorien sehr effektiv miteinander verknüpft werden und gleichzeitig die Richtigkeit dieser Zelltrajektorien überprüft wird.

6. Programm CellTracker



(Abbildung 6-1)

Das **Menü (1)** mit den Punkten File, Projekt, WorkingView, Help.

File:
- New Projekt (Erstellen eines neuen Projektes)
- Open Projekt (Öffnet ein bereits erstelltes Projekt)
- Save Current Project (Speichert das Projekt)
- Save Current Project As (Speichern unter neuem Namen)
- Export (Speichert die neu erstellten Trajektorien im XML-Format)
- Exit

Project:
- Project Details (Übersicht über das gesamte Projekt mit weiteren Einstellmöglichkeiten)
- Offset/ Point Of Origin (Einstellmöglichkeiten bei eventueller Verschiebung der Koordinaten (seihe 5.4))

Working View:
- show Cell Items (Zellobjekte werden nicht angezeigt, können jedoch trotzdem ausgewählt werden)
- show only Images (nur Bilder werden angezeigt, Zellobjekte werden nicht berechnet)
- base size (Setzt Bild auf Originalgröße zurueck)
- zoom in
- zoom out
- scroll hand drag (Scrollen mittels Klicken und „Ziehen“)
- scroll bar as needed (Zeigt Scrollbar bei entsprechender Zoomstufe)
- show WorkingView Info (zeigt Informationen über Skalierung, Fenstergröße, etc.)

Help: - About CellTracker (Informationen über das Programm)
- About Qt (Informationen über das verwendete Framework)

Das **Tools (2)** Docking Window stellt Funktionen zum erstellen neuer Trajektorien bereit.

- *add Cell* (9) fügt ein selektiertes Zellobjekt zu der bearbeitenden Trajektorie hinzu. Alternativ kann auch die Space-Taste gedrückt werden (siehe Tooltip).
- *remove Cell* (10) löscht ein eventuell zu diesem Zeitpunkt vorhandenes Zellobjekt aus der Liste der derzeit bearbeiteten Trajektorie.
- *remove all Cells till now* (11) löscht alle Zellobjekte vom Anfang bis zu aktuellem Zeitpunkt aus der Liste.
- *remove all Cells from now* (12) löscht alle Zellobjekte vom aktuellem Zeitpunkt bis zum Ende der Zelltrajektorie aus der Liste.
- *current track* (13) Auswahl der zu bearbeitenden Zelle/Zelltrajektorie
- *create new track* (14) legt eine leere Zelltrajektorie an.
- *remove current track* (15) löscht die ausgewählte ZellTrajektorie komplett.
- *change tracks* (16) Tauscht zwei Trajektorien vom aktuellen Zeitpunkt an miteinander (Im Falle einer Verwechslung notwendig).
- *add daughter track* (17) Hier werden die Tochterzellen assoziiert. Der track-status der Mutterzelle springt automatisch auf 'Zellteilung'.
- *remove daughter tracks* (18) Löscht Zuordnung der Tochterzellen.
- *change track status* (19) Legt Status der Bearbeitung einer Trajektorie fest (offen / Zellteilung / tot / verloren / Ende des Films).

Der aktuelle Zeitpunkt wird im **Time Info (3)** Docking Window angezeigt. Zusätzlich ist die Anzahl der Zellobjekte zum jeweiligen Zeitpunkt aufgeführt.

Cell Info Docking Window (4) zeigt Informationen über ein selektiertes Zellobjekt und der dazugehörenden generierten Trajektorie an.

Falls ein neuer Track erstellt wurde, werden im **Current Track Info (5)** Docking Window Informationen über diesen angezeigt.

Das **Navigation (6)** Docking Window stellt Funktionen zum Navigieren durch den Bildstapel bereit: Schieberegler (24), Abspiel-Buttons (25) und Buttons zum einfachen navigieren(26). Die einzelnen Tastaturbelegungen der Buttons werden in den Tooltips angezeigt.

Mit dem Docking Window **Strategies (7)** stehen Funktionen für eine schnelle und effiziente Bearbeitung der Trajektorien bereit. (siehe auch 5.4 - Effektives Arbeiten unter Verwendung entworfener Strategien).

- „click and step“ (20) Gewünschte Zelle anklicken, Bild springt eins weiter.
- „hover and step“ (21) Mit Mauszeiger über gewünschtes Zellobjekt, das hinzugefügt werden soll. Das Bild springt dann automatisch um Eins weiter und das Zellobjekt wird der Trajektorie hinzugefügt.
- „click and jump“ (22) Nach dem Anklicken wird an das Ende der gewünschten Trajektorie gesprungen, alle Objekte dazwischen werden hinzugefügt.
- *„click and spin“* (23) Anklicken und Film abspielen.

Siehe auch 5.4 – Effektives Arbeiten unter Verwendung entworfener Strategien für eine weitere Beschreibung der Strategien.

Im **Anzeigefenster (8)** werden die Bilder und die Zellobjekte dargestellt. Dies ist der eigentliche Arbeitsbereich. Hier werden die Zellobjekte selektiert / ausgewählt. Um ein überschaubareres Arbeiten zu gewährleisten werden die Zellobjekte mit drei

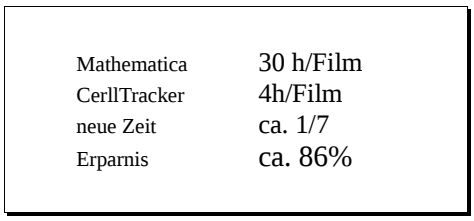
unterschiedlichen Farben dargestellt.

- *Blau (leicht transparent)*: Zellobjekte die weder selektiert noch einer Zelltrajektorie zugeordnet sind.
- *Rot (leicht transparent)*: Zellobjekte gehören bereits zu einer anderen Zelltrajektorie.
- Die Dazugehörende TrackID wird zusätzlich über dem Zellobjekt angezeigt (siehe Abbildung 6-1).
- Rot: Zellobjekt ist selektiert. (Wenn entweder mit der Maus angeklickt wurde, oder sich der Mauszeiger über dem Objekt befindet.)
- Grün: Zellobjekt gehört bereits zur aktuell bearbeiteten Zelltrajektorie.
- Gelb: Zellobjekte der ausgewählten vorgenerierten Trajektorie.

7. Zusammenfassung und Ausblicke

7.1 Zusammenfassung

Um den Zuwachs an Effektivität des neuen Programmes im Vergleich zu der bestehenden Lösung zu evaluieren, wurden alte Filme (die bereits mit dem Mathematica-Prototyp bearbeitet und ausgewertet worden waren) nochmals mit dem neuen CellTracker Programm bearbeitet. Die folgende Abbildung zeigt die benötigten (mittleren) Zeiten von sechs verglichenen Filmen.



(Abbildung 7.1-1)

Dies zeigt, dass jetzt die benötigte Zeit, um einen kompletten Film nachzubearbeiten, im Durchschnitt 1/7 der vorher benötigten Zeit beträgt. Das bedeutet eine durchschnittliche Zeitersparnis von über 80%.

7.2 Diskussion

Das Ziel dieser Arbeit – die vorgenerierten Trajektorien schnell, effizient und benutzerfreundlich miteinander zu verknüpfen – konnte mit dem Programm *CellTracker* erreicht werden.

Bereits durchgeführte Testdurchläufe zeigten, dass keine von den entworfenen Strategien unbedingt überlegen ist. Die Wahl der Strategien ist natürlich abhängig von den Daten. Es gibt Zeitrafferfilme wo sich die Zellen relativ ruhig verhalten und Filme wo sie viel „herumlaufen“, ja „springen“. Dies ist vorher nicht ersichtlich.

Auch das Feedback über die Benutzbarkeit ist positiv. Besonders Vorteilhaft sind die verschiedenen Bedienmöglichkeiten („click and step“, „hover and step“, „click and jump“ bzw. „click and spin“). Dadurch kann man in problematischen Fällen die geeignete Strategie wählen und so flexibel auf Situationen reagieren.

Dies zeigt, dass die aufgestellten Qualitätsanforderungen (wie Funktionalität, Zufverlässigkeit, Benutzbarkeit, Effizienz, etc.) erfolgreich umgesetzt wurden.

Das Programm CellTracker ist zur Zeit das einzige Tool, was sowohl vollständig manuell, das heißt jede Zelle einzeln, als auch mit Hilfe von automatisch vorgenerierten Tracks, Genealogien aus Filmen von Zellkulturen erstellen kann.

7.3 Möglichkeiten der Weiterentwicklung / Ausblicke

Bei dem Programm CellTracker handelt es sich um eine Neuentwicklung. Durch die objektorientierte Softwareentwicklung – und der daraus folgenden guten Strukturierung und Modularisierung – bietet das Programm viele Möglichkeiten, an denen Verbesserungen und Weiterentwicklungen angesetzt werden können.

Diese Erweiterungen / Weiterentwicklungen sind zum Teil direkt umsetzbar mittels Plugins. Qt bietet solche Schnittstellen. [QtReference]

Erweiterung der Applikation:

Derzeit beschränkt sich der wesentliche Teil des Programmes auf – das interaktive Zusammenfügen einzelner Trajektorien – die Phase 3 (siehe Kapitel 2). Zusätzlich könnte

die Vorbearbeitung und die Analyse als Teil der Applikation (eventuell realisiert durch Plugins) mit eingebunden werden. Für die Vorbearbeitung heißt das, dass zu den importierten Bildstapel die Metadaten (mittels bereits vorhandenen Objekterkennungs-algorithmen) dann automatisch innerhalb des Programms berechnet werden sollen.

Falls diese automatisierten Objekterkennungsverfahren implementiert sind, könnte man auch den „Grad“ der Automatisierung vor der Berechnung der Metadaten wählen. (Wie viel will ich selber machen, wie viel soll der Rechner automatisch rechnen?)

Grundlegende Auswertungen sollen in Zukunft im Programm berechnet werden können. (Beispielsweise Geschwindigkeitsprofile einzelner Zellen anzeigen/berechnen, Anzahl der Teilungen anzeigen, Symmetrien in den Teilungsbäumen berechnen (dazu bestehen bereits Formeln), Aufenthaltswahrscheinlichkeiten an den einzelnen Orten im Bild berechnen/darstellen etc.)

Desweiteren wäre eine Darstellung der bereits erstellten Zellbäume mit Verbindung zu den Bilddaten hilfreich. Dies könnte beispielsweise realisiert werden, indem man die Trajektorien als Bäume auf einer Zeitlinie in einem eigenen/neuen Fenster visualisiert.

Erweiterung der Programmoberfläche mit weiteren Sprachen ist dank Qt Linguist kein Problem, da die nötigen Vorkehrungen im Source-Code für die Übersetzung der Anwendung implementiert wurden. [Wolf2007, S. 729] [Blanchette2009, S. 420]

Andere Speicherformate:

Der Import und Export der Metadaten soll auf andere maschinenlesbare Formate erweitert werden (z.B. n3) um die Verwendung formaler Beschreibungssprachen aus der Ontologie zu ermöglichen. Dadurch könnten die Daten zwischen verschiedenen Programmen konsistent ausgetauscht und verglichen werden. Eine Speicherung und semantische Suche in geeigneten Datenbanken waere damit ebenfalls moeglich.

Projekte sollen auch in nicht binären Formaten importiert werden können (zum Beispiel XML-Format), um auch Projekte zu laden und weiter zu bearbeiten, die eventuell in einem externen Programm erstellt worden sind.

Optimierung der Bearbeitungszeit:

Des Weiteren könnte eine *Neuauswertung/Reorganisierung* der noch nicht bearbeitenden computergenerierten Tracks nach der Fertigstellung eines neuen Tracks iterativ Mehrdeutigkeiten aus Daten entfernen. Somit könnte ein Projekt, gerade am Ende, wenn schon viele Trajektorien erstellt wurden, schneller abgeschlossen werden. Dies setzt jedoch eine Implementierung des automatischen Trackings in das bestehende Programm voraus.

Treffen einer *automatischen Vorauswahl* eines neuen Kandidaten bei Trackende. Dies würde beim Verfolgen der Strategie „click and jump“ und „click and spin“ zu einem weiterem Geschwindigkeitsvorteil hinsichtlich der Bearbeitungszeit führen.
Dies könnte dann für Zellen in einer gewissen Umgebung und auch im Zusammenhang mit der vorher genannten Reorganisation geschehen. (Geschwindigkeit und Richtung der Zellen in der Umgebung)

Multithreads beim Anzeigen des Zeitrafferfilms, um das Laden und Anzeigen der Bilder, sowie das Berechnen der Metadaten und das Einzeichnen der Zellobjekte gleichzeitig ausführen zu können. Das gewährleistet einen schnellerer Bildaufbau. Dies ist auch von Vorteil, wenn in naher Zukunft eventuell 3D Aufnahmen von Live Cell Imaging Experimenten analysiert werden sollen. (Einer Tiefenauswahl steht von Qt's Seite nichts im Wege, da die verwendete Klasse QGraphicsItems bereits Operationen für die z-Values mitbringt.)
Auch Erweiterungen für 3-D Animationen sind mit Qt dank OpenGL oder VTK kein Problem.

Optimierung der Bedienbarkeit:

Abspielgeschwindigkeit mittels Mausgesten regeln. Beim abspielen des Zeitrafferfilms oder bei der Strategie „click and spin“ könnten Mausbewegung dafür verwendet werden um die Abspielgeschwindigkeit zu steuern.

QTouchEvent für Touch Screens. Um das Verfolgen der Zellen auch auf portablen Geräten mit Touch Screen zu gewährleisten bietet Qt neben der `QMouseEvent` Klasse die Klasse `QTouchEvent` für Touch Screen Eingabegeräte.

Erweiterung der Dokumentation:

Damit auch Aussenstehende zur Fehlersuche oder zum Zwecke von Änderungen und Erweiterungen einen Überblick über das Projekt bekommen können, muss der Sourcecode noch umfangreicher kommentiert werden. Als Kommentarformat soll die Syntax der Doxygen-kommentare verwendet werden, um mit dem Open-Source-Dokumentationstool *Doxygen* eine HTML-Dokumentation erstellen zu können.

Science-Fiction oder Zukunft?
Viele Hände machen der Arbeit bald ein Ende!

Für das oberste Ziel, die Nachbearbeitungszeit zu minimieren, werden die Algorithmen für Objekterkennung und Trackerkennung optimiert und benutzerfreundliche Tracking-Tools implementiert. Was für Möglichkeiten bestehen aber nun, wenn man selbst gar keine Arbeitszeit mehr investieren möchte? Fest steht, dass das Nachbearbeiten trotz bester Algorithmen unverzichtbar ist.
Das Schlüsselwort in diesem Falle ist *Crowdsourcing*. Damit ist die Auslagerung von Aufgaben auf „die Intelligenz und die Arbeitskraft einer Masse von Freizeitarbeitern im Internet“ gemeint [Wikipedia, siehe Crowdsourcing] (Vergleichbar mit Open Innovation, Kollektive Intelligenz oder Swarm Intelligence).
Die Realisierung dazu könnte auf verschiedenden Wegen laufen. Zum einen könnte man die gesamten Metadaten eines Films in ein hübsches 3D-Spiel packen. Wobei die einzelnen Trackstückchen als approximierte Raumkurve dargestellt werden (Postitionen und Zeitpunkte einer Zelle bilden diese Raumkurve).
Oder man benutzt das „Divide and Conquer“ (Teile und herrsche) Prinzip. Hierzu müsste man die Problemestellen, bei denen die Objekt- / Trackerkennungsalgorithmen aufhören, in elementare Teilprobleme zerlegen. Diese dann soweit abstrahieren, sodass sie dann ähnlich Google's Crowdsourcing-Konzept von reCAPTCHA[1] [heiseOnline] auf einer Internetplattform im Austausch zu einer Dienstleistung dem User zur Lösung präsentiert werden.
Eine Schätzung für die Genauigkeit für die Ergebnisse bekommt man, indem man dem User bereits gelöstes und noch zu lösendes Material gibt und dessen Resultate durch mehrfaches Wiederholen mit unterschiedlichen Usern abgleicht. Das Ergebnis was am häufigsten übereinstimmt wird als richtig angenommen. Hierfür ist die Anzahl der Iterationen maßgeblich für den Grad der gewünschten mathematischen Wahrscheinlichkeit.

[1] reCAPTCHA – Ein System, welches ursprünglich an der Carnegie Mellon University von Luis von Ahn entwickelt wurde. Es benutz CAPTCHAs um bei Buch-Digitalisierungen die eingescannten Wörter, die die Texterkennungsalgorithmen nicht erkannt haben, zu finden und gleichzeitig begrenzte Bereiche auf Webseiten vor Bots zu schützen.

Glossar

Ein **Bot** ist ein weitgehend autonom arbeitendes Computerprogramm, was wiederholende Aufgaben abarbeitet, ohne dabei auf eine Interaktion mit einem menschlichen Benutzer angewiesen zu sein. [Lee2008]

CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Aparat) bedeutet wörtlich übersetzt „Vollautomatischer öffentlicher Turing-Test zur Unterscheidung von Computern und Menschen“. Mit CAPTCHAs versucht man zu unterscheiden, ob das Gegenüber ein Mensch oder eine Maschine ist.

Genealogie ist hier der komplette Verwandschaftsbaum einer Zelle und ihrer Nachkommen (siehe auch [Glauche2009]).

Eine **generierte Trajektorie** (Klassenname `Track`), darunter versteht man die vom Computer zu einer „Teil-Trajektorie“ zusammengefassten Zellobjekte.

Metadaten bestehen aus zusätzlichen Informationen für die einzelnen Zellobjekte, z.B. Größe, Form, Intensität etc. (abhängig von den verwendeten Bildverarbeitungsverfahren).

Mit **Rapid Prototyping** bezeichnet man Verfahren zur schnellen und direkten Herstellung von Produkten.

Trackdaten enthalten Listen von erkannten Objekten (mit entsprechenden Zeitpunkten), die zu ein und derselben Zelle gehören.

Unter einer **Zelle** auch Zelltrajektorie (Klassenname `NewTrack`) versteht man die *vollständig* verfolgte Trajektorie (von erster Beobachtung bis zu einem definierten Endpunkt wie Teilung / Zelltod / Verlieren der Zelle etc.). Tochterzellen werden als neue Zellen angesehen und sind somit eigene Trajektorien.

Ein **Zellobjekt** (Klassenname `Cell`) ist ein *einzelnes* automatisch erkanntes Objekt in *einem* Bild.

Literaturverzeichnis

[Balzert2000] Helmut Balzert (2000). Lehrbuch der Software-Technik – Software-Entwicklung (2. Auflage). *Spektrum Akademischer Verlag Heidelberg – Berlin*. ISBN 3-8274-0480-0

[Balzert2005] Heide Balzert (2005). UML 2 kompakt, mit Checklisten. (2. Auflage) *Elsevier, Spektrum Akademischer Verlag*. ISBN 3-8274-1389-3

[Beck2003] Kent Beck (2003) Extreme Programming – Das Mainfest. *Addison-Wesley Verlag*. ISBN 3-8273-2139-5

[Blanchette2009] Blanchette J, Summerfield M (2009) C++ GUI-Programmierung mit Qt 4. *Addison-Wesley Verlag*. ISBN 3-8273-2186-7

[Bleek2008] Bleek W-G, Wolf H (2008) – Agile Softwareentwicklung: Werte Konzepte und Methode. *Dpunkt-Verl.* ISBN 978-3-8986-4473-0

[Eilken2009] Eilken HM, Nishikawa S-I, Schroeder T (2009) Continuous single-cell imaging of blood generation from haemogenic endothelium. *Nature* 457:896-900.

[Glauche2009] I. Glauche, R. Lorenz, D. Hasenclever, I. Roeder (2009):
A novel view on stem cell development: Analyzing the shape of cellular genealogies
Cell Proliferation 42(2):248 – 263

[Gonzalez2008] Gonzalez RC, Woods RE (2008) *Digital Image Processing* (Prentice Hall series Pearson Education, New Jersey). 3rd Ed.

[heiseOnline2009] Artikel vom 16.09.2009 22:08 „Google gliedert führenden Captcha-Spezialisten ein“ http://www.heise.de/newsticker/meldung/Google-gliedert-fuehrenden-Captcha-Spezialisten-ein-788988.html

[Lee2008] Wenke Lee (Hrsg.): *Botnet detection: countering the largest security threat*. Springer, New York 2008. ISBN 978-0-387-68766-7

[Levitner2009] Levitner T, Timr S, Stys D (2010) Expertomica Cells: analysis of cell monolayer development. *Bioinformatics (Oxford, England)* 26:278-9.

[Megason2007] Megason SG, Fraser SE (2007) Imaging in systems biology. *Cell* 130:784-95.

[QtLicensing2010] http://qt.nokia.com/products/licensing (stand 10.11.2010)

[QtReference] http://doc.qt.nokia.com/

[Rieger2009] Rieger M, Hoppe PS, Smejkal BM, Eitelhuber AC, Schroeder T (2009) Hematopoietic cytokines can instruct lineage choice. *Science (New York, N.Y.)* 325:217-8.

[Rudin1992] Rudin L, Osher S, Fatemi E (1992) Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena* 60:259-268.

[Sacan2008] Sacan A, Ferhatosmanoglu H, Coskun H (2008) CellTrack: an open-source software for cell tracking and motility analysis. *Bioinformatics (Oxford, England)* 24:1647-9. Available at: http://www.ncbi.nlm.nih.gov/pubmed/18511469.

[Scherf2009] Scherf N et al. (2009) Spatio-temporal Analysis of Unstained Cells In-vitro. *Bildverarbeitung fuer die Medizin 2009*:292–296.

[Schroeder2008] Schroeder T (2008) Imaging stem-cell-driven regeneration in mammals. *Nature* 453:345-351.

[TTT] Timm's Tracking Tool. Herausgegeben von Michael A. Rieger und Dr. Timm Schroeder. *Institut für Stammzellforschung, Helmholtz Zentrum München Forschungszentrum für Gesundheit und Umwelt.* http://www.helmholtz-muenchen.de/isf/haematopoese/software-download/index.html

[VTK] http://vtk.org/

[Wikipedia] http://www.wikipedia.de

[Wolf2007] Jürgen Wolf (2007). Qt 4 - GUI-Entwicklung mit C++. *Galileo Press.* ISBN 978-3-8362-1030-0

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann.

Leipzig, 1. Januar, 2011

Michael Kunze