



UNIVERSITÄT  
KOBLENZ · LANDAU

Fachbereich 4: Informatik

# **Berechnung und Visualisierung von Kavitäten in Molekülen während eines Docking-Prozesses**

## **Bachelorarbeit**

zur Erlangung des Grades Bachelor of Science (B.Sc.)  
im Studiengang Computervisualistik

vorgelegt von  
**Vanessa Schüller**

Erstgutachter: Prof. Dr.-Ing. Stefan Müller  
(Institut für Computervisualistik, AG Computergraphik)  
Zweitgutachter: Nils Lichtenberg

Koblenz, im Februar 2017





Institut für Computervisualistik  
AG Computergraphik  
Prof. Dr. Stefan Müller  
Postfach 20 16 02  
56 016 Koblenz  
Tel.: 0261-287-2727  
Fax: 0261-287-2735  
E-Mail: stefanm@uni-koblenz.de



UNIVERSITÄT  
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Bachelorarbeit  
Vanessa Schüller  
(213200567)

**Thema: Berechnung und Visualisierung von Kavitäten in Molekülen während eines Docking-Prozesses**

In der Medizin und Pharmazie ist es für viele Forschungsbereiche grundlegend wichtig, das Verhalten von Molekülen verstehen und vorhersehen zu können. So ist es besonders für die Entwicklung neuer Medikamente essentiell, die Interaktion von großen und komplexen Molekülen, wie Proteinen, untereinander nachvollziehen zu können und zu prognostizieren, ob und wie Liganden an solche andocken können.

Voraussetzung für das Andocken eines Liganden an ein Protein ist, dass dessen Oberfläche in die Kavitäten, also die Hohlräume, des Moleküls passt.

Während eines Docking-Prozesses kann es nun passieren, dass sich die räumliche Struktur des Moleküls und somit auch die Kavitäten verändern, sodass vermeintlich passende Liganden, nicht mehr andocken können.

Ziel dieser Arbeit ist es nun, Kavitäten von Molekülen und Protein-Ligand-Komplexen zu berechnen und den Einfluss eines Docking-Prozesses auf die inneren Strukturen zu beobachten und zu visualisieren. Dazu wird auf den Algorithmus des PyMol Plugins KVFinder aufgebaut, welcher mit Hilfe von geometrisch basierten Methoden Kavitäten in einzelnen Molekülen findet. Der Algorithmus soll so erweitert werden, dass Veränderungen durch den Dockingprozess, berechnet und zusammen mit dem Liganden visualisiert werden, um Andockmöglichkeiten vor und nach dem Dockingprozess vergleichen zu können.

Die inhaltlichen Schwerpunkte der Arbeit sind:

1. Recherche über chemische Eigenschaften und Protein-Ligand-Docking
2. Berechnung von inneren räumlichen Strukturen von Molekülen
3. Entwicklung einer Umgebung zur Visualisierung von Kavitäten
4. Implementierung von Berechnung und Visualisierung
5. Demonstration der Ergebnisse
6. Dokumentation und Bewertung der Ergebnisse

Koblenz, den 01.09.2016

Vanessa Schüller

Prof. Dr. Stefan Müller

## **Zusammenfassung**

Die vorliegende Arbeit beschreibt die Entwicklung eines OpenGL-basierten Tools zur Visualisierung von Hohlräumen in Proteinen, welche während eines statischen Dockings beobachtet werden können. Ziel ist es, anhand von Informationen über Abstände zwischen Proteinen und Liganden, Schlussfolgerungen über Interaktionen zu ziehen, um daraus Ansätze für die Entwicklung künstlicher Liganden zu gewinnen.

Zunächst wird auf chemische Grundlagen eingegangen, die das Thema motivieren und für das Verständnis der Thematik und der genutzten Algorithmen wichtig sind. Des Weiteren wird bestehende Software vorgestellt, die ähnliche Sachverhalte löst. Anschließend werden die Voraussetzungen zur Entwicklung des Programmes genannt, woraufhin dieses detailliert beschrieben wird. Zum Abschluss wird das Tool in Hinblick auf Performance und Nutzen evaluiert und ein zusammenfassendes Fazit getroffen, in dem sich das Programm als gute Hilfe für bestehende Forschungen und gute Basis für weitere, tiefergehende Forschungsprojekte erweist.

## **Abstract**

The present thesis describes the development of an OpenGL-based tool visualizing cavities of proteins, which can be observed during a static docking simulation. The goal is to achieve knowledge about interactions between proteins and ligands based on information about distances between them.

At first chemical basics, which motivate the topic and are important for understanding the topic and the used algorithms, are presented. Furthermore existing software, which deals with similar issues, is described. Next the prerequisites for the development of the program are presented and the tool is described in detail. Concluding the tool is evaluated concerning performance and usage and a summarizing conclusion is given. The program turns out as a helpful tool for current research and a good base for further and deeper research projects.

# Inhaltsverzeichnis

<b>1</b>	<b>Motivation</b>	<b>1</b>
<b>2</b>	<b>Einleitung</b>	<b>1</b>
<b>3</b>	<b>Chemische Grundlagen</b>	<b>2</b>
3.1	Proteine . . . . .	2
3.1.1	Strukturen von Proteinen . . . . .	2
3.1.2	Oberflächendefinitionen . . . . .	4
3.1.3	Funktionen von Proteinen . . . . .	5
3.1.4	Wechselwirkungen zwischen Proteinen . . . . .	5
3.1.5	Wechselwirkungen zwischen Proteinen und Liganden . . . . .	6
3.2	Proteine in der Pharmazie . . . . .	6
3.2.1	Beispiel eCD4-Ig . . . . .	7
3.3	Protein Docking . . . . .	8
<b>4</b>	<b>Visualisierung von Proteinen</b>	<b>9</b>
4.1	Protein Data Bank . . . . .	10
4.2	3D-Grafiksoftware . . . . .	10
<b>5</b>	<b>Berechnung von Kavitäten</b>	<b>11</b>
5.1	Auf evolutionären Prinzipien basierende Methoden . . . . .	12
5.2	Energie-basierte Methoden . . . . .	12
5.3	Geometrie-basierte Methoden . . . . .	13
5.4	KVFinder . . . . .	15
5.4.1	Definition von Kavitäten . . . . .	16
5.4.2	Algorithmus . . . . .	17
5.4.3	Vor- und Nachteile des Algorithmus . . . . .	19
<b>6</b>	<b>CavityDetector</b>	<b>20</b>
6.1	Kurzbeschreibung des Tools . . . . .	20
6.2	Programmierungumgebung . . . . .	21
6.2.1	Systemvoraussetzungen . . . . .	21
6.2.2	QTCreator . . . . .	22
6.2.3	CMake . . . . .	22
6.2.4	ProjectBase Framework . . . . .	23
6.3	Klassenhierarchie . . . . .	24
6.3.1	Benutzerabfragen . . . . .	24
6.3.2	PDBLoader . . . . .	26
6.3.3	CavityFinder . . . . .	28
6.4	Shader . . . . .	32
6.4.1	ImpostorShader . . . . .	33
6.4.2	Order Independent Transparency-Shader . . . . .	36
6.5	Ergebnis und Interaktion . . . . .	38

6.6	Konsolenausgaben . . . . .	42
6.7	Evaluierung der Ergebnisse . . . . .	43
6.7.1	Lade- und Berechnungszeiten . . . . .	44
6.7.2	Bewertung durch pharmazeutische Experten . . . . .	50
6.7.3	Fazit . . . . .	58
<b>7</b>	<b>Ausblick</b>	<b>59</b>
<b>A</b>	<b>Fragebogen</b>	<b>61</b>
<b>B</b>	<b>Auswertung des Fragebogens</b>	<b>71</b>
	<b>Glossar</b>	<b>81</b>



## Abbildungsverzeichnis

1	Das Schlüssel-Schloss-Prinzip . . . . .	2
2	Die vier Strukturebenen von Proteinen . . . . .	3
3	Verschiedene Oberflächendefinitionen . . . . .	4
4	Wirkungsweise von eCD4-Ig . . . . .	7
5	Protein-Ligand-Docking . . . . .	8
6	Aufbau einer PDB-Datei . . . . .	10
7	VMD User Interface . . . . .	11
8	JMOL User Interface . . . . .	11
9	PyMol User Interface . . . . .	11
10	Ergebnis ConSurf . . . . .	12
11	Ergebnis Q-SiteFinder . . . . .	13
12	Algorithmus POCASA . . . . .	13
13	Ergebnis KVFinder . . . . .	14
14	Ergebnis FPocket . . . . .	14
15	MetaPocket Algorithmus . . . . .	15
16	KVFinder's Definition von Kavitäten . . . . .	16
17	Algorithmus KVFinder . . . . .	17
18	Grid vor und nach Einfügen des Proteins . . . . .	17
19	Grid vor, während und nach Umlaufen mit Probe . . . . .	18
20	Grids für Probe Out (links) und Probe In (rechts) . . . . .	18
21	Markierung der nicht zu Kavitäten gehörenden Voxel . . . . .	19
22	Kavitätenberechnung . . . . .	20
23	OpenGL 4.3 Support führender Grafikkartenhersteller . . . . .	22
24	Grafische Benutzeroberfläche von CMake . . . . .	23
25	Klassenhierarchie von CavityDetector . . . . .	24
26	Erwartete Eingaben durch den Benutzer . . . . .	26
27	Occupancy-Wert in einer pdb-Datei . . . . .	26
28	Definition und Verwendung eines ESBTL-Classifiers . . . . .	27
29	Hierarchische Datenstruktur von ESBTL . . . . .	27
30	Bestimmung des Suchraumes . . . . .	29
31	Skizze der Tiefensuche zur Berechnung der Kavitätenvolumen	32
32	Schematische Darstellung des Impostor Shaders . . . . .	34
33	Mögliche Ergebnisse beim Schnitttest von Kugel und Linie . . . . .	36
34	Protein 5iks mit Impostor gerendert . . . . .	36
35	Ligand ohne und mit Transparenz . . . . .	37
36	Oberfläche CavityDetector . . . . .	38
37	Aminosäurefarbmodus . . . . .	39
38	Colorscale für Kavitätenpunkte . . . . .	40
39	Berechnete Kavitäten . . . . .	40
40	Transparenz weit entfernter Kavitätenpunkte . . . . .	41
41	Ligand in Kavitäten ein- und ausgeblendet . . . . .	42
42	Konsolenausgabe von CavityDetector . . . . .	43

43	Einfluss von Probe-Out Radius auf Lade- und Berechnungszeiten . . . . .	45
44	Einfluss von Probe-Out Radius auf das Ergebnis . . . . .	46
45	Einfluss von Probe-Out Radius auf Lade- und Berechnungszeiten . . . . .	47
46	Einfluss von Schrittweite auf Lade- und Berechnungszeiten .	48
47	Einfluss von Schrittweite auf Ergebnis . . . . .	49
48	Bindungssituation 1 - Ligand in einer falschen Bindungstasche	52
49	Bindungssituation 2 - Ligand in seiner korrekten Bindungstasche . . . . .	52
50	Aufbau des Fragebogens . . . . .	53
51	Bewertung des Abstandes von Ligand zu Protein . . . . .	54
52	Bindungssituation 1 . . . . .	55
53	Bindungssituation 2 . . . . .	55
54	Bewertung der räumlichen Einschätzung ohne Kavitätensvisualisierung . . . . .	56
55	Bewertung der räumlichen Einschätzung mit Kavitätensvisualisierung . . . . .	57
56	Bewertung des Abstandes von Ligand zu Protein . . . . .	58
57	Ligand in korrekter Bindungstasche . . . . .	59

# 1 Motivation

Das Verständnis von Proteinen ist in vielen Bereichen unseres Lebens von grundlegender Bedeutung. Ob es darum geht, die Ursache von Krankheiten zu verstehen und Ansätze für neue Medikamente zu finden oder die Wirkungsweise von Drogen oder anderen Schadstoffen zu verstehen - die Antworten auf die Fragen des „Warum?“ oder „Wie?“ können meistens nur dadurch gefunden werden, die Wechselwirkungen verschiedener Proteine und Liganden zu prognostizieren, zu erkennen und zu analysieren.

Da Proteine ihre Funktionen in der Regel immer erst durch die Bindung anderer Proteine oder Liganden aktivieren, ist es nicht nur wichtig ein Verständnis über die komplexen Strukturen eines einzelnen Proteins zu haben, sondern die Interaktion mehrerer Proteine zu verstehen und einen visuellen Eindruck der Auswirkungen eines solchen Prozesses zu erhalten.

Ob und auf welche Weise Proteine miteinander agieren, wird durch ihre Oberflächenstruktur bestimmt. Durch bestimmte Faltungen eines Proteins können Bindungstaschen entstehen, die durch ihre chemischen Eigenschaften potenzielle Andockstellen für spezifische Proteine oder Liganden bilden.

Diese Arbeit beschreibt die Entwicklung eines Tools zur Berechnung, Visualisierung und Analyse von Hohlräumen eines Proteins, welche Benutzer während eines Dockingprozesses beobachten können sollen. So soll es möglich sein, zum Einen die Oberfläche und innere Strukturen eines Proteins detailliert untersuchen zu können und zum Anderen auch die Auswirkungen des Anbindens eines Liganden auf eine Bindungstasche erkennen und Bindungssituationen analysieren und bewerten zu können.

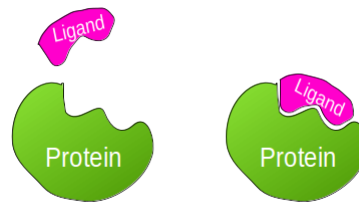
# 2 Einleitung

Proteine bilden einen der wichtigsten Grundbausteine des menschlichen Körpers. Dies ist spätestens seit dem Aufkommen von modernen Fitness-eiweißshakes und -riegeln jedem bekannt. Aber nicht nur beim Aufbau von Muskeln sind diese komplexen Moleküle beteiligt. Proteine machen bis zu 15% der Körpermasse aus[25] und sind auf etliche Arten und Weisen essentiell für unser Leben. In Form von Hormonen, Enzymen oder Antikörpern sind sie intensiv an den meisten Funktionalitäten des menschlichen Organismus beteiligt, indem sie Stoffwechselprozesse in Gang bringen, das Immunsystem unterstützen oder lebenswichtige Stoffe transportieren und bilden damit das Fundament für gesundes Leben.

Man kann sich somit aber auch leicht vorstellen, dass Proteine, aufgrund ihrer „Mächtigkeit“ nicht nur Segen, sondern auch Fluch sein können. Defekte Proteine, Viren oder Drogen können die Funktionalitäten der Proteine sehr schnell ausschalten oder sogar ins Negative umwandeln. Krankheiten

wie Diabetes mellitus, Mukoviszidose oder Krebs sind alle auf Störungen des Wechselspiels von Proteinen zurückzuführen.

Um die Prozesse eines gesunden Organismus, aber auch die Ursache von Krankheiten verstehen zu können, ist es essentiell zu wissen, wie Proteine funktionieren und wodurch bestimmte Funktionalitäten entstehen.



**Abbildung 1:** Das Schlüssel-Schloss-Prinzip

An dieser Stelle spielen die Oberfläche und die Struktur der Moleküle die entscheidenden Rollen, da Proteine und Liganden nach dem Schlüssel-Schloss-Prinzip<sup>1</sup> interagieren. Passt die Struktur eines Liganden in die Struktur eines Proteins, so können sie einen Protein-Ligand-Komplex bilden. Dies hängt allerdings auch von einigen chemischen Eigenschaften ab, welche in den folgenden Kapiteln vorgestellt werden.

## 3 Chemische Grundlagen

### 3.1 Proteine

Proteine sind biologische Makromoleküle, welche in allen Zellen auftreten und bis zu 50% ihres Trockengewichtes ausmachen. Sie setzen sich aus Aminosäuren zusammen, welche durch Peptidbindungen zusammengehalten werden. Insgesamt setzen sich die Proteine des Menschen aus 21 verschiedenen Aminosäuren zusammen, von denen acht nicht vom Menschen selbst hergestellt werden können. Daher ist die Aufnahme von Proteinen über die Nahrung auch besonders wichtig.

Die Aminosäuresequenz eines Proteins ist in der DNS codiert, anhand welcher Ribosomen, makromolekulare Komplexe, die jeweiligen Aminosäuren zu Proteinen zusammensetzen können (vgl. zu diesem Abschnitt [12]). Abhängig von der Abfolge der Aminosäuren entstehen verschiedene Strukturen, die im Folgenden beschrieben werden.

#### 3.1.1 Strukturen von Proteinen

Die dreidimensionalen Strukturen von Proteinen werden durch vier verschiedene Strukturebenen beschrieben.

Die Primärstruktur, die Sekundärstruktur, die Tertiärstruktur und die Quartärstruktur.

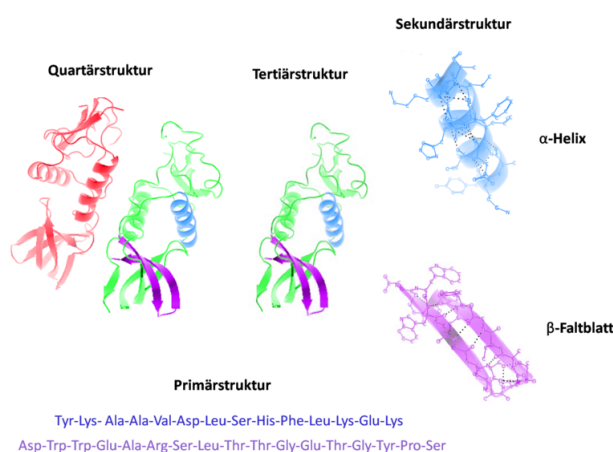
Diese Strukturen entstehen durch Wechselwirkungen zwischen NH- und CO- Gruppen innerhalb derselben oder zwischen verschiedenen Polypeptidketten und werden durch diese stabilisiert. Diese Wechselwirkungen sind die sogenannten Wasserstoff- und Disulfidbrücken.

Die Primärstruktur beschreibt die Aminosäuresequenz des Proteins, also die Abfolge der Aminosäuren in der Peptidkette.

Die Sekundärstruktur definiert auf einer nächst höheren Ebene die räumliche Struktur von lokalen Bereichen. Mögliche Strukturen sind beispielsweise die  $\alpha$ -Helix und das  $\beta$ -Faltblatt.

Die Tertiärstruktur beschreibt die räumliche Struktur einer Unterheit und die Quartärstruktur umfasst die dreidimensionale Struktur des vollständigen Proteinkomplexes mit allen Untereinheiten (vgl. zu diesem Abschnitt [14]).

Abbildung 2 zeigt eine Übersicht dieser Strukturen.



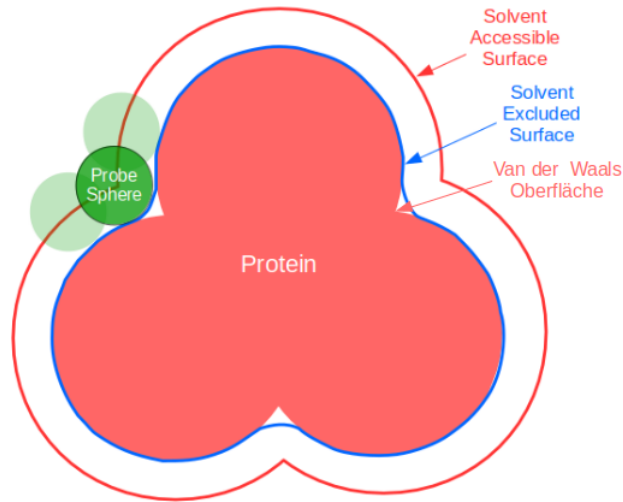
**Abbildung 2:** Die vier Strukturebenen von Proteinen

Von Holger87 - Eigenes Werk, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=20396685>

Die initiale Struktur eines Proteins erfolgt im Allgemeinen während des Translationsprozesses, der „Geburt“ eines Proteins. Anhand der Basensequenz der DNS werden die Proteine an den Ribosomen in den menschlichen Mitochondrien oder den pflanzlichen Chloroplasten synthetisiert.

Die Proteinstruktur kann aber auch nachträglich durch Wechselwirkungen mit Enzymen, Chaperonen und Liganden beeinflusst werden, wenn es zu Konformationsänderungen kommt. Hierbei kann es zu neuen Oberflächeneigenschaften und der Aktivierung neuer Funktionen kommen (vgl. zu diesem Abschnitt [14]). Wie die Proteinoberfläche definiert und beschrieben werden kann, wird im folgenden Abschnitt erläutert.

### 3.1.2 Oberflächendefinitionen



**Abbildung 3:** Verschiedene Oberflächendefinitionen

Schaut man sich Abbildungen in Schulbüchern oder ähnlichen Quellen an, fällt auf, dass häufig nur das Rückgrat [12] zur vereinfachenden Darstellung der komplexen Strukturen genutzt wird.

Viel essentieller für die Funktionalität eines Proteins ist jedoch die Oberfläche. Diese wird erst durch die Seitenketten der Aminosäuren des Proteins bestimmt.

Da auch die Oberfläche von Proteinen sehr komplex sein kann und feine Strukturen aufweist, gibt es verschiedene Methoden und Definitionen diese zu abstrahieren und zu beschreiben. Eine Übersicht über die Varianten illustriert Abbildung 3 .

1. *Van der Waals Oberfläche* Die Van der Waals-Oberfläche wird durch die individuellen Van der Waals-Radien aller Atome bestimmt. Diese Radien geben an, auf welchen Abstand sich nicht gebundene Atome potentiell annähern können. [15] Zur Visualisierung nutzt man dazu Kugeln mit den entsprechenden Radien, sodass die Hüllen dieser Kugeln die Van der Waals-Oberfläche bilden.
2. *Solvent Accessible Surface Area* Die SASA beschreibt die Oberfläche eines Moleküls, die zugänglich für einen bestimmten Stoff ist. Zur Berechnung wird der „Rolling Ball“ Algorithmus genutzt. [7] Eine Kugel mit dem Radius des gewünschten Stoffes wird um das Protein gerollt. Jeder Punkt, der von dieser Kugel besetzt werden kann, ohne dass das Protein geschnitten wird, gehört zur SASA.

3. *Solvent Excluded Surface* Die SES, auch Connolly Surface genannt, hängt dicht mit der SASA zusammen. Der Algorithmus zur Berechnung basiert auf dem gleichen Rolling-Ball-Prinzip. Alle Punkte der van-der-Waals Oberfläche eines Proteins, die von einem Stoff berührt, aber nicht geschnitten werden, sowie die Punkte der zum Protein gewandten van-der-Waals Oberfläche eines Stoffes, wenn er mehr als ein Atom des Proteins berührt, gehören zur SES. Im Grunde ist diese Oberflächendefinition nichts anderes als das Inverse der SASA.[24]

Durch Faltungen der Oberflächen und daraus resultierenden chemischen Eigenschaften werden die verschiedenen Funktionalitäten von Proteinen realisiert. Welche Aufgaben Proteine übernehmen, erläutert der nächste Abschnitt.

### 3.1.3 Funktionen von Proteinen

Wie bereits angesprochen übernehmen Proteine eine Vielzahl an Aufgaben im Organismus. Die Funktionen der Proteine werden durch ihre Aminosäuresequenzen und die daraus resultierenden Strukturen bestimmt.

Membranproteine sind vorwiegend aus hydrophoben, also wasserabweisenden Aminosäuren aufgebaut und ermöglichen als Ionenkanal an den Membranen den Transport von Ionen. Ein Beispiel hierfür ist Porin.

Proteine treten des Weiteren in Form von Enzymen auf, wie zum Beispiel die Katalase in der Leber. Sie bestehen aus mehreren Untereinheiten und vielen Helix- und Faltblattanteilen. Charakteristisch ist das katalytische Zentrum, welches für die Wirksamkeit eines Enzyms verantwortlich ist. An dieser Stelle binden Substrate an.

Eine weitere Funktion, die Proteine übernehmen, ist die Festigung und Formgebung von Zellen, wie es zum Beispiel bei Kollagen der Fall ist. Strukturproteine sind oft hydrophob und besitzen sich wiederholende Aminosäuresequenzen (vgl. zu den letzten drei Abschnitten [5]).

Besonders bei Enzymen wird die Wichtigkeit der Oberfläche eines Proteins klar. Enzyme arbeiten nach dem Schlüssel-Schloss-Prinzip 1. Das bedeutet, dass Substrate, deren Konformation mit dem katalytischen Zentrum zusammenpasst, andocken können, um schließlich verarbeitet zu werden. Ändert sich die Struktur und damit auch die Bindungsstelle, kann ein und dasselbe Substrat nicht mehr andocken. Die Form und das Volumen der Bindungsstelle ist also entscheidend für die Funktion des Enzyms. Auf Wechselwirkungen, die diese Faktoren beeinflussen können, wird im Folgenden genauer eingegangen.

### 3.1.4 Wechselwirkungen zwischen Proteinen

Die komplexen Strukturen eines Proteins entstehen durch Wasserstoff- und Disulfidbrücken zwischen NH- und CO-Gruppen. Aber nicht nur inner-

halb eines Proteins wirken Kräfte, die wichtig für dessen Funktion sind. Viel wichtiger und interessanter sind die Wechselwirkungen verschiedener Proteine. Diese Interaktionen sind grundlegend für nahezu alle Prozesse eines Organismus, die Proteine benötigen.

Das Zusammenspiel von Proteinen ist auf nicht-kovalente Wechselwirkungen zurückzuführen. Das bedeutet, dass sich die Proteine keine Elektronenpaare teilen, sondern chemisch miteinander interagieren. Zu den nicht-kovalenten Bindungen gehören zum Einen die Wasserstoffbrückenbindungen, die bereits bei der Sekundärstruktur entscheidend sind, aber zum Anderen auch van-der-Waals-Kräfte, elektrostatische Wechselwirkungen oder hydrophobe Effekte von Aminosäureresten an der Proteinoberfläche (vgl. zu diesem Abschnitt [13]).

Um Wechselwirkungen zwischen Proteinen zu untersuchen gibt es bereits einige experimentelle biochemische und biophysikalische Methoden. Einige Beispiele hierfür sind das Hefe-Zwei-Hybrid-System[6], das molekulare Display[36], die Oberflächenplasmonenresonanzspektroskopie[18], der Ligandenbindungstest[4] oder das Thermal Shift Assay[27].

Darüber hinaus treten zudem auch Wechselwirkungen zwischen Proteinen und Liganden auf.

### **3.1.5 Wechselwirkungen zwischen Proteinen und Liganden**

Unter Liganden versteht man im Allgemeinen kleine Strukturen, wie kleine Moleküle, Atome, Ionen oder Radikale, die sich an ein Protein anlagern. Insbesondere Moleküle, die spezifisch an Oberflächenstrukturen eines Proteins binden und dadurch eine Konformitätsänderung veranlassen und Funktionalitäten realisieren, werden als Liganden bezeichnet.

Für gewöhnlich weist ein Protein auf seiner Oberfläche ein oder mehrere Bindungszentren auf, die für einen speziellen Liganden eine besonders hohe Bindungsaffinität haben. Die Wechselwirkungen zwischen Proteinen und Liganden sind somit spezifisch [31], weswegen ihnen auch in der Forschung ein besonders hoher Stellenwert zuteil wird. Das nächste Kapitel geht näher auf die pharmazeutische Forschung im Umgang mit Proteinen ein.

## **3.2 Proteine in der Pharmazie**

Aufgrund der Spezifität von Protein-Ligand-Komplexen, bilden sie in der pharmazeutischen und medizinischen Forschung einen besonders wichtigen Forschungsschwerpunkt.

Um verstehen zu können, wodurch Krankheiten ausbrechen oder das körpereigene Immunsystem angegriffen wird, muss der Ursprung gefunden werden, welcher fast immer bei einer fehlerhaften oder ausbleibenden Wech-



selwirkung von Proteinen liegt.

An diesem Punkt setzt auch die Medikamentenforschung an. Ist es möglich die dreidimensionalen Strukturen und die chemischen Eigenschaften einer Proteinoberfläche zu analysieren, können synthetische Liganden beispielsweise als Arzneimittel entwickelt werden, indem die Bindungsaffinität zu bestimmten Stellen ausgenutzt wird. Der nachfolgende Abschnitt liefert ein Beispiel, in dem eine solche Analyse erfolgreich war.

### 3.2.1 Beispiel eCD4-Ig

Ein aktuelles Beispiel für diese Form des Medikamentendesigns ist das von Forschern des Scripps Research Institutes<sup>1</sup> in Florida entwickelte Molekül eCD4-Ig, durch welches ein dauerhafter Schutz vor einer HIV-Infektion angestrebt wird.[30]

Bei einer HIV-Infektion docken die HI-Viren mit Hilfe von CD4-Rezeptoren, welche an der Oberfläche von Zellen des Immunsystems sitzen, oder CCR5-Rezeptoren, welche in Leukozyten und Leukoplasten verbreitet sind, an ihren Wirtszellen an.

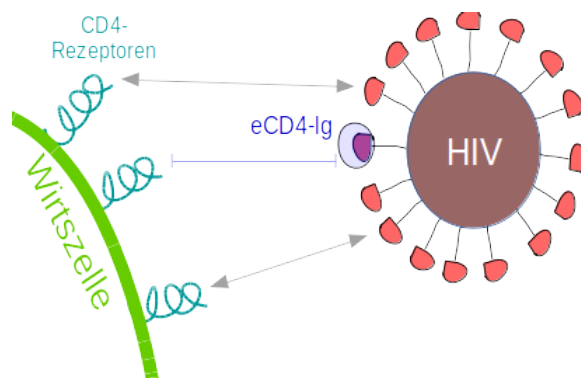


Abbildung 4: Wirkungsweise von eCD4-Ig

eCD4-Ig wurde so designt, dass es an die Oberflächenproteine des HI-Virus bindet, sodass das Andocken an die Wirtszellen verhindert wird (vgl. Abbildung 4). Getestet wurde dieses Molekül in einer 40-wöchigen Studie. Der Bauplan von eCD4-Ig wurde über einen Trägervirus in Zellen von vier Affen gesetzt, welchen anschließend hohe Dosen des tierischen Ursprungsvirus von HIV, des Simianen Immunschwäche-Virus (SIV) zugeführt wurden.

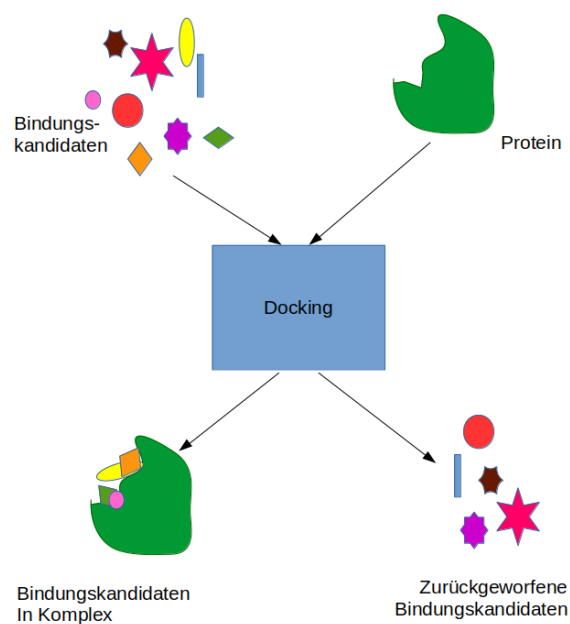
Im Gegensatz zu einer weiteren Gruppe Affen, welche keine eCD4-Ig Therapie bekamen, wies keines der Tiere am Ende der Testphase eine SIV-Infektion auf (vgl. zu diesem Kapitel [30]).

Dieses Beispiel zeigt deutlich, wie viel Potential in dem Verständnis von

<sup>1</sup><http://www.scripps.edu/>

Proteinen und deren Oberflächen steckt und wie hilfreich eine visuelle Darstellung solcher abstrakter und komplexer Moleküle sein kann, um zu erkennen ob, an welcher Stelle und mit welchem Effekt das Binden eines Medikaments möglich ist. Solche Probleme und Fragestellungen werden unter dem Begriff des *Protein Dockings* zusammengefasst, welcher Inhalt des folgenden Kapitels ist.

### 3.3 Protein Docking



**Abbildung 5:** Protein-Ligand-Docking

Da für ein erfolgreiches Binden eines Liganden an ein Protein sehr viele Faktoren eine Rolle spielen, ist es wichtig zu testen, ob und wie ein Ligand überhaupt binden kann.

Während praktische Methoden im Nasslabor sehr zeitaufwändig sind und auch hohe Kosten verursachen, bildet das sogenannte „Docking“ eine deutlich effizientere Möglichkeit, das Bindungsverhalten von Proteinen und Liganden zu untersuchen.

Docking ist ein computergestütztes Verfahren, welches versucht virtuell

den Bindungsmodus und die Bindungsenergie zwischen Molekülen zu prognostizieren. Anhand bekannter chemischer und räumlicher Strukturen soll ein Komplex zweier Moleküle vorhergesagt werden, der im besten Fall realen Verhältnissen sehr nah kommt [8]. Ist eine Verbindung dieser beiden Strukturen nicht möglich, so wird auch das prognostiziert. Abbildung 5 skizziert dieses Verfahren.

Da Proteine hochkomplex, Rechenkapazitäten jedoch begrenzt sind, ist es nötig, mit Annäherungen an die Realität zu arbeiten.

So gibt es Verfahren, wie das „starre Docking“, die annehmen, dass sich Moleküle während eines Bindungsvorgangs nicht verändern oder Methoden, die nur ein Molekül flexibel halten.

Ob es sich um Protein-Protein-Docking, Protein-Ligand-Docking oder DNA-Docking handelt, bestimmt, welche Algorithmen letztendlich genutzt werden.

Da für das Docking die Kenntnis der räumlichen Strukturen von Proteinen unabdingbar ist, ist eine gute Visualisierung dieser Moleküle hilfreich und auch nicht mehr wegzudenken. Hierfür gibt es einige etablierte Tools, welche in dem nächsten Kapitel vorgestellt werden.

## 4 Visualisierung von Proteinen

Um ein umfassendes Bild und ein Verständnis von Proteinen zu erlangen, ist es sehr wichtig einen visuellen Eindruck der komplexen Strukturen zu bekommen. Wie sieht ein Protein aus? An welchen Stellen liegen besondere Konformationen vor? Welche Konsequenzen können aus diesen Strukturen gezogen werden?

In der vorelektronischen Zeit war es kaum möglich, Antworten auf diese Fragen zu finden. Visualisierungen von chemischen Strukturen wurden mit der Hand angefertigt, erforderten dadurch viel Zeit und Arbeit und ermöglichten lediglich einen visuellen Eindruck eines einzigen Zustandes.

Die Entwicklung von Computern und Grafikkartenprogrammierung waren ein großer Zugewinn für die medizinische und pharmazeutische Forschung. Mit verschiedenen Tools war und ist es möglich, dreidimensionale Bilder von Proteinen und anderen Strukturen zu erzeugen, mit diesen zu interagieren und so das Verhalten solch großer Moleküle nachzuvollziehen.

Als Grundlage für genaue Visualisierungen von Proteinen dienen Datensätze, die beispielsweise durch Kristallstrukturanalysen gewonnen werden. Diese Datensätze werden in der *Protein Data Bank (PDB)* gespeichert, welche im folgenden Kapitel beschrieben wird.

## 4.1 Protein Data Bank

```

HEADER      PLANT PROTEIN                               30-APR-81  1CRN
TITLE       WATER STRUCTURE OF A HYDROPHOBIC PROTEIN AT ATOMIC RESOLUTION.
TITLE       PENTAGON RINGS OF WATER MOLECULES IN CRYSTALS OF CRAMBIN
COMPND     1 MOLECULE: CRAMBIN;
COMPND     2 MOLECULE: CRAMBIN;
COMPND     3 CHAIN: A;
COMPND     4 ENGINEERED: YES
SOURCE     1 MOL_ID: 1;
SOURCE     2 ORGANISM: SCIENTIFIC: CRAMBE HISPANICA SUBSP. ABYSSINICA;
SOURCE     3 ORGANISM_TAXID: 3721;
SOURCE     4 STRAIN: SUBSP. ABYSSINICA
KEYWDS     PLANT SEED PROTEIN, PLANT PROTEIN
EXPDTA     X-RAY DIFFRACTION
AUTHOR     W.A.HENDRICKSON, M.M.TEETER
REVDAT    7 11-JUL-12 1CRN 1 SCALE1 VERSN HEADER
REVDAT    6 24-FEB-09 1CRN 1 VERSN
REVDAT    5 16-APR-87 1CRN 1 HEADER
REVDAT    4 04-MAR-85 1CRN 1 REMARK
REVDAT    3 30-SEP-83 1CRN 1 REVDAT
REVDAT    2 03-DEC-81 1CRN 1 SHEET
REVDAT    1 29-JUL-81 1CRN 0
JRNLTITLE  M.M.TEETER
JRNLTITLE  TITL WATER STRUCTURE OF A HYDROPHOBIC PROTEIN AT ATOMIC
JRNLTITLE  TITL 2 RESOLUTION: PENTAGON RINGS OF WATER MOLECULES IN CRYSTALS OF
JRNLTITLE  TITL 3 CRAMBIN.
JRNLTITLE  REF PROC.NATL.ACAD.SCI.USA V. 81 6914 1984
JRNLTITLE  REFN ISSN 0027-8424
JRNLTITLE  PMID 16592516
JRNLTITLE  DOI 10.1073/PNAS.81.19.6914
REMARK     1 REFERENCE 1
REMARK     2 AUTHOR W.A.HENDRICKSON, M.M.TEETER
REMARK     1 TITL STRUCTURE OF THE HYDROPHOBIC PROTEIN CRAMBIN DETERMINED
REMARK     1 TITL 2 DIRECTLY FROM THE ANOMALOUS SCATTERING OF SULPHUR
REMARK     1 REF NATURE V. 290 107 1981
REMARK     1 REFN ISSN 0028-0836
REMARK     1 REFERENCE 2
REMARK     2 AUTHOR M.M.TEETER, W.A.HENDRICKSON
REMARK     1 TITL HIGHLY ORDERED CRYSTALS OF THE PLANT SEED PROTEIN CRAMBIN
REMARK     1 REF J.MOL.BIOL. V. 127 219 1979
REMARK     1 REFN ISSN 0022-2836
REMARK     2
ATOM       305 O TYR A 44 14.030 9.862 13.568 1.00 6.04 O
ATOM       306 CB TYR A 44 13.200 11.014 14.073 1.00 5.41 C
ATOM       307 CG TYR A 44 12.000 12.819 14.399 1.00 5.34 C
ATOM       308 CD1 TYR A 44 12.119 13.853 15.332 1.00 6.59 C
ATOM       309 CD2 TYR A 44 10.775 12.617 13.762 1.00 5.94 C
ATOM       310 CE1 TYR A 44 11.045 14.075 15.619 1.00 5.97 C
ATOM       311 CE2 TYR A 44 9.676 13.453 14.046 1.00 5.17 C
ATOM       312 CZ TYR A 44 9.892 14.456 14.906 1.00 5.06 C
ATOM       313 OH TYR A 44 8.740 15.265 15.269 1.00 8.60 O
ATOM       314 N ALA A 45 14.342 8.640 15.422 1.00 4.76 N
ATOM       315 CA ALA A 45 15.445 7.667 15.246 1.00 5.09 C
ATOM       316 C ALA A 45 15.171 6.533 14.280 1.00 6.67 C
ATOM       317 O ALA A 45 16.093 5.765 14.939 1.00 7.56 O
ATOM       318 CB ALA A 45 15.680 7.999 16.682 1.00 6.82 C
ATOM       319 N ASN A 46 13.966 6.562 13.739 1.00 5.90 N
ATOM       320 CA ASN A 46 13.512 5.395 12.878 1.00 6.15 C
ATOM       321 C ASN A 46 13.311 5.853 11.455 1.00 6.01 C
ATOM       322 O ASN A 46 13.733 6.929 11.026 1.00 7.18 O
ATOM       323 CB ASN A 46 12.266 4.769 13.591 1.00 7.27 C
ATOM       324 CG ASN A 46 12.538 4.304 14.022 1.00 7.98 C
ATOM       325 OD1 ASN A 46 11.982 4.849 15.886 1.00 11.00 O
ATOM       326 ND2 ASN A 46 13.407 3.298 15.025 1.00 19.22 N
ATOM       327 OXT ASN A 46 12.703 4.973 10.746 1.00 7.86 O
TER       328 ASN A 46
CONNECT   26 292
CONNECT   26 229
CONNECT  116 188
CONNECT  186 116
CONNECT  229 26
CONNECT  282 29
MASTER   224 0 0 2 2 1 0 6 327 1 6 4
END

```

Abbildung 6: Aufbau einer PDB-Datei

Mit über 100.000 Einträgen ist die Protein Data Bank die zentrale Datenbank für 3D-Strukturdatensätze von Proteinen.

Eine typische PDB-Datei, wie sie in Abbildung 6 zu sehen ist, ist eine Textdatei und speichert spaltenbasiert allgemeine Informationen, Positionen und Bezeichnungen einzelner Atome und Vorkommen strukturgebender Aminosäuren ab.

Seit dem Jahre 2003 ist die Protein Data Bank durch die Gründung der *Worldwide Protein Data Bank (wwPDB)*<sup>2</sup> international zugänglich und wird durch die Mitglieder in Europa (ePDB), der USA (RCSB) und Japan (PDBj) verwaltet.

Dieses Format bildet einen Standard, der von allen Tools einheitlich verarbeitet werden kann, um Visualisierungen von Proteinen zu generieren. Einige Beispiele für solche Visualisierungssoftware werden im nächsten Abschnitt vorgestellt.

## 4.2 3D-Grafiksoftware

Heutzutage gibt es einige etablierte Visualisierungstools, die neben hochaufgelösten Darstellungen auch die Interaktion mit Proteinen realisieren. Einige kostenfreie Beispiele hierfür sind *Jmol*<sup>3</sup>(Abb. 8), *UGENE*<sup>4</sup>, *VMD*<sup>5</sup>(Abb. 7), *BALL*<sup>6</sup> oder *QuteMol*<sup>7</sup>. Eines der verbreitetsten Tools ist jedoch das Open-Source Tool *PyMOL*<sup>8</sup>(Abb. 9). Neben den umfangreichen Visualisierungs-

<sup>2</sup><http://www.wwpdb.org/>

<sup>3</sup><http://jmol.sourceforge.net/>

<sup>4</sup><http://ugene.net/>

<sup>5</sup><http://www.ks.uiuc.edu/Research/vmd/>

<sup>6</sup><http://www.ball-project.org/>

<sup>7</sup><http://qutemol.sourceforge.net/>

<sup>8</sup><https://www.pymol.org/>

möglichkeiten und der Betriebssystemunabhängigkeit bietet PyMOL den Vorteil, dass ein Python-Interpreter integriert ist, wodurch es leicht möglich ist, die Funktionalität durch Plugins zu erweitern. Eine Liste verfügbarer Plugins kann dem PyMOL-Wiki entnommen werden. [35]

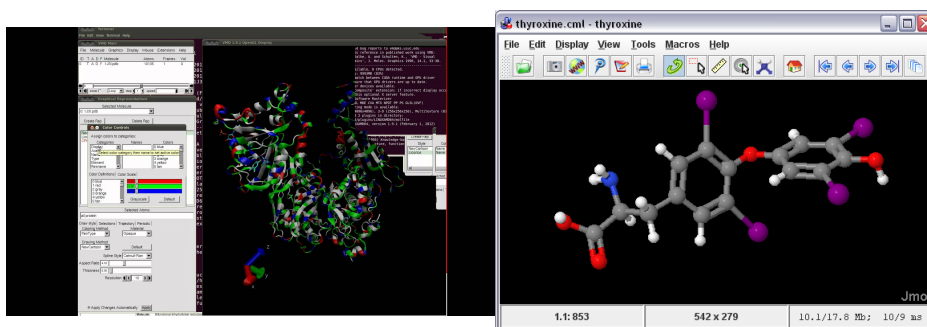


Abbildung 7: VMD User Interface

Abbildung 8: Jmol User Interface

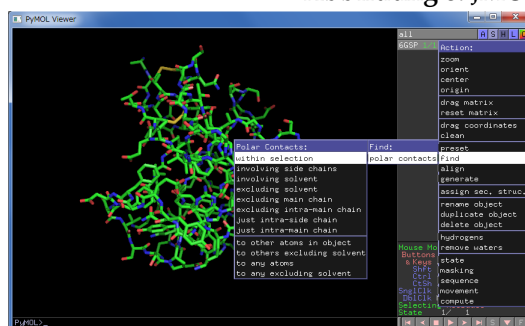


Abbildung 9: PyMol User Interface

Eines dieser Plugins ist KVFinder [29]. KVFinder ermöglicht es, die Hohlräume, und somit potentielle Bindungstaschen für Liganden in Proteinen zu berechnen und zu visualisieren. Dadurch soll es möglich sein, die Oberfläche eines Proteins noch besser zu veranschaulichen und mögliche Bindungsstellen leichter zu lokalisieren. Auf dieses Tool und seinen Algorithmus wird im weiteren Verlauf genauer eingegangen, da es Grundlage des in dieser Arbeit entwickelten Programmes bildet. Zunächst wird jedoch ein Überblick über bestehende Verfahren zur Berechnung von Hohlräumen gegeben.

## 5 Berechnung von Kavitäten

Um Oberflächen und ihre Hohlräume, die Kavitäten, zu untersuchen und potentielle Bindungsstellen vorherzusagen und zu charakterisieren, wurde über die Jahre eine Vielzahl an Software entwickelt.

Einige Beispiele sind LIGSITE [2], PocketPicker [28], PocketFinder [1], Q-SiteFinder [20], MetaPocket [17], POCASA [37], DoGSite [26], ConSurf [3], VOIDOO [21], Fpocket[32] und KVFinder [29].

Die zugrunde liegenden Algorithmen lassen sich grob in drei verschiedene Kategorien einordnen und werden im Folgenden erläutert - Geometrie-basierte, Energie-basierte und auf evolutionären Prinzipien basierende Verfahren.

## 5.1 Auf evolutionären Prinzipien basierende Methoden

Bei Methoden, die auf evolutionären Prinzipien basieren, werden die Reste von Aminosäureketten untersucht, da diese eine wichtige Rolle für die Bindungsaffinität einer Oberfläche spielen. Dabei bedient man sich dem Wissen über bekannte Eigenschaften bestimmter funktioneller Gruppen. ConSurf ist ein Beispiel für diese Herangehensweise und visualisiert Bindungsstellen durch die Einfärbung von Resten, wie es Abbildung 10 zeigt. Wie dies im Detail funktioniert, wird im korrespondierenden Paper [3] beschrieben.

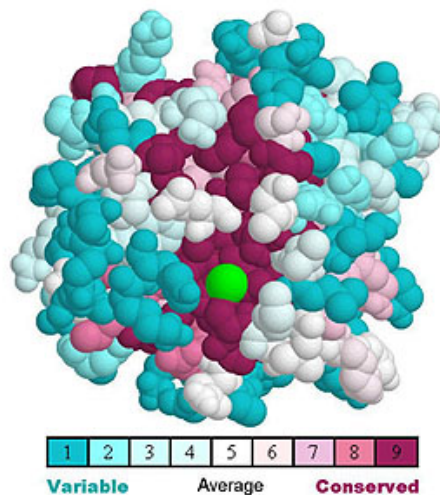


Abbildung 10: Ergebnis ConSurf

## 5.2 Energie-basierte Methoden

Bei energiebasierten Verfahren fokussiert man sich auf die Analyse von Wechselwirkungen zwischen einem Protein und Probe Spheres, um so Stellen zu finden, die hohe Bindungsaffinitäten aufweisen und somit interessante Kavitäten bilden. Diese Technik findet zum Beispiel bei Q-SiteFinder Einsatz. Wie genau der zugrunde liegende Algorithmus funktioniert, kann

dem entsprechenden Paper [20] entnommen werden. Ein mögliches Ergebnis dieser Anwendung zeigt Abbildung 11.

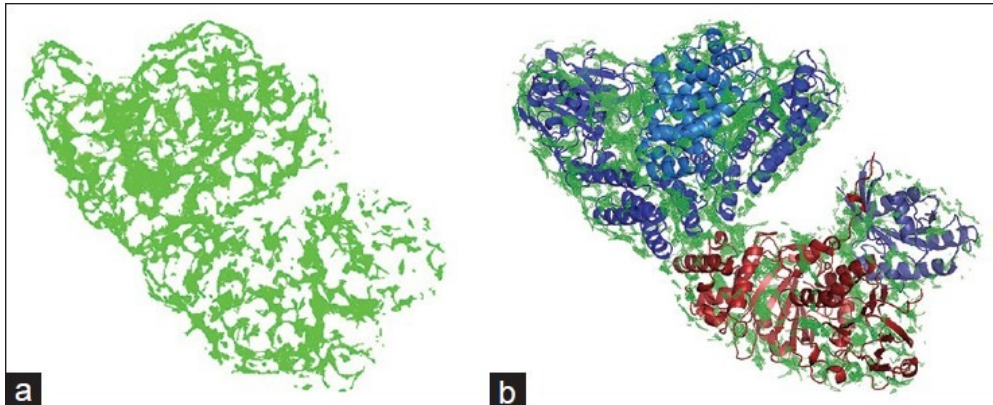


Abbildung 11: Ergebnis Q-SiteFinder

### 5.3 Geometrie-basierte Methoden

Geometrische Methoden untersuchen zur Detektion von Kavitäten die molekularen Oberflächen von Proteinen. Dazu bedient man sich häufig dreidimensionaler Grids, wie es beispielsweise bei POCASA (vgl.12), Pocket-Picker oder KVFinder der Fall ist.

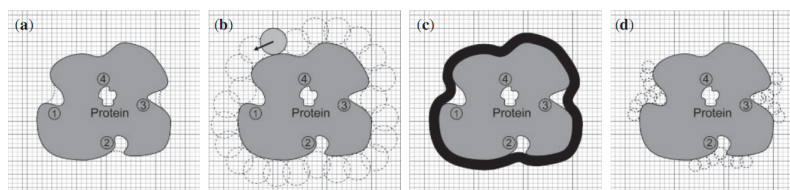


Abbildung 12: Algorithmus POCASA

Weitergehend werden oft sogenannte Probe Spheres mit individuellen Radien benutzt, die einmal um das Protein laufen, um so die Oberflächenstruktur zu analysieren. Beispiele dazu sind KVFinder(vgl.13) oder POCASA, dem Begründer dieses Roll-Algorithmus.

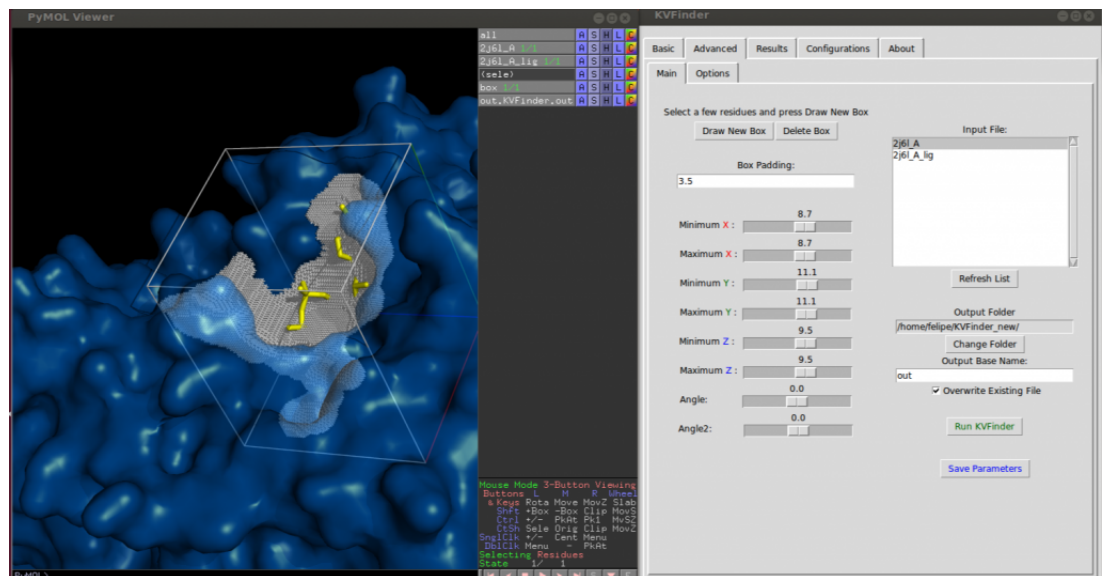


Abbildung 13: Ergebnis KVFinder

PocketPicker hingegen benutzt Gap-Spheres, die sofort in den Hohlräumen platziert werden und solange verkleinert werden, bis sie von keinen Atomen geschnitten werden.

Es gibt jedoch auch Beispiele, die nicht mit Grids arbeiten, sondern auf anderen geometrischen Prinzipien basieren. FPocket basiert beispielsweise auf Voronoi-Tessellierung (vgl.14) und DoGSite arbeitet mit einem Difference of Gaussian Verfahren, welches eigentlich aus der Bildverarbeitung bekannt ist.

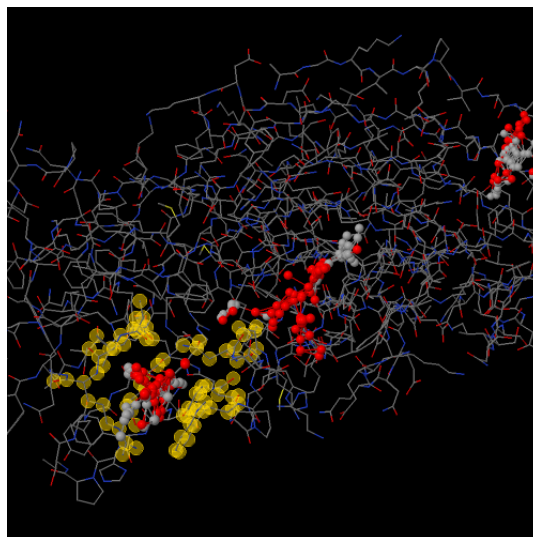


Abbildung 14: Ergebnis FPocket



Beschreibungen der Algorithmen und Ergebnisse sind auf den Abbildungen 12, 13 und 14 zu sehen und können in den dazugehörigen Papern genauer nachvollzogen werden.

Darüber hinaus gibt es Programme, die mehrere Algorithmen miteinander kombinieren, um ein möglichst optimales Ergebnis aus verschiedenen Methoden zu generieren. So arbeitet MetaPocket beispielsweise mit Prognosen von Kavitäten, die durch Programme wie LIGSITE, Q-SiteFinder oder POCASA generiert werden. Diese werden bewertet und verschieden gewichtet, um daraus die potentiellen und „besten“ Bindungsstellen und Bindungstaschen zu filtern. Abbildung 15 skizziert diese Prozedur.

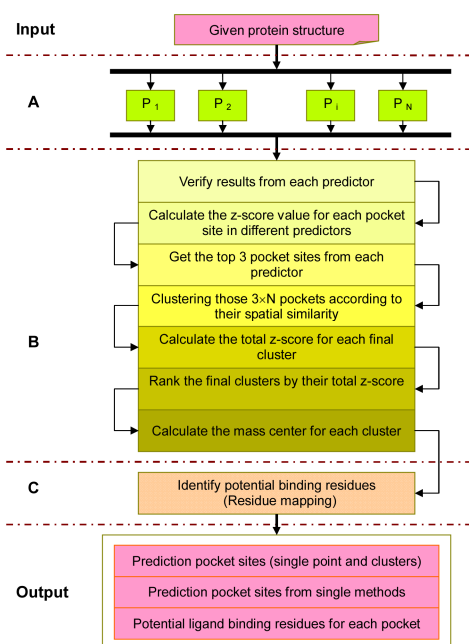


Abbildung 15: MetaPocket Algorithmus

Da sich geometrische Verfahren jedoch sehr gut veranschaulichen lassen und leicht nachvollziehbar sind, richtet sich der Fokus im Folgenden auf das Pymol Plugin KVFinder, welches die Grundlage des in dieser Arbeit entwickelten Tools *CavityDetector* bildet.

## 5.4 KVFinder

Wie bereits beschrieben, ist KVFinder ein Plugin für die Software PyMol und berechnet mit Hilfe von geometrischen Methoden Hohlräume von Proteinen. KVFinder bietet dem Benutzer viele Möglichkeiten, verschiedene Parameter anzupassen, um so verschiedene Arten von Kavitäten, wie Bindungstaschen, Tunnel oder schmale Einkerbungen zu finden. Desweite-

ren ist es möglich, nur bestimmte und für den Benutzer interessante Teile des Proteins zu untersuchen, sodass unnötiger Rechenaufwand vermieden wird.

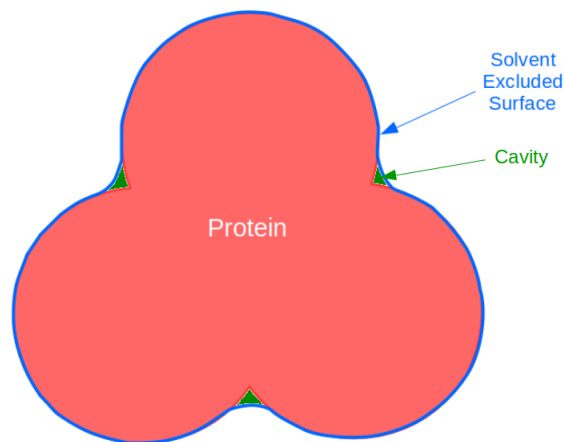
Welcher Algorithmus KVFinder zugrunde liegt und welche Vor- und Nachteile dieser bietet, wird in den folgenden Kapiteln beschrieben.

Zunächst ist es jedoch wichtig zu wissen, wie KVFinder Kavitäten überhaupt definiert.

#### 5.4.1 Definition von Kavitäten

Als Kavitäten bezeichnet man im allgemeinen Hohlräume. In Bezug auf Proteine versteht man darunter durch die Oberflächenfaltung entstehende Taschen, Tunnel oder Ritzen.

Um diese konkret berechnen zu können, ist es aber notwendig, zu definieren, wo eine solche Kavität anfängt und wo sie aufhört. Im Grunde definiert KVFinder nichts Anderes als Kavität, als die Differenz von einer Solvent Excluded Surface des Proteins und des Proteins selbst, wie es Abbildung 16 veranschaulicht.



**Abbildung 16:** KVFinder's Definition von Kavitäten

Dazu werden anschaulich zwei Probe Spheres mit verschieden großen Radien um das Protein bewegt.

Die kleinere Sphere, *Probe In*, definiert zunächst eine Solvent-Excluded-Surface des Proteins, die für kleine Moleküle erreichbar ist. KVFinder legt den Radius der Probe In auf 1.4 Å fest, was ungefähr dem Radius eines Wassermoleküls, dem kleinsten Molekül, entspricht, um so den maximal erreichbaren Bereich für einen Liganden zu beschreiben und die tatsächliche Oberfläche des Proteins anzunähern.

Der Radius der größeren Probe Sphere, *Probe Out*, kann durch den Benutzer bestimmt werden. Diese definiert eine weitere Solvent Excluded Surface des Proteins.

Aus der Differenz dieser beiden Oberflächenbeschreibungen können schließlich die Kavitäten generiert werden (vgl. zu diesem Abschnitt [29]). Abbildung 17 veranschaulicht dieses Verfahren. Wie dies im Detail von statten geht, wird im nächsten Abschnitt beschrieben.

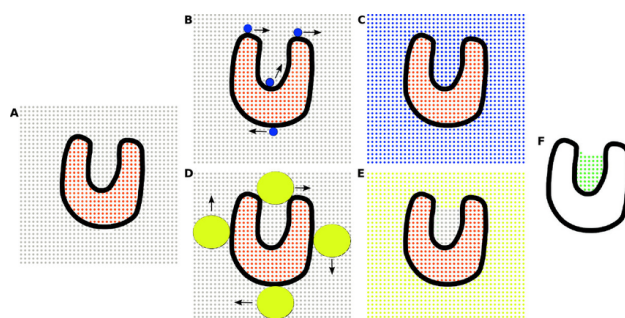


Abbildung 17: Algorithmus KVFinder

#### 5.4.2 Algorithmus

Um die Kavitäten berechnen zu können, benutzt KVFinder zwei dreidimensionale Grids.

Nachdem ein Protein erfolgreich geladen worden ist, wird das erste Gitter initialisiert. Jedes Voxel, das von einem Atom des Proteins eingenommen ist, wird markiert. Zusätzlich werden alle Voxel markiert, die einen Abstand kleiner des Probe Out-Radius zum Protein haben. Abbildung 18 zeigt die daraus resultierende Situation.

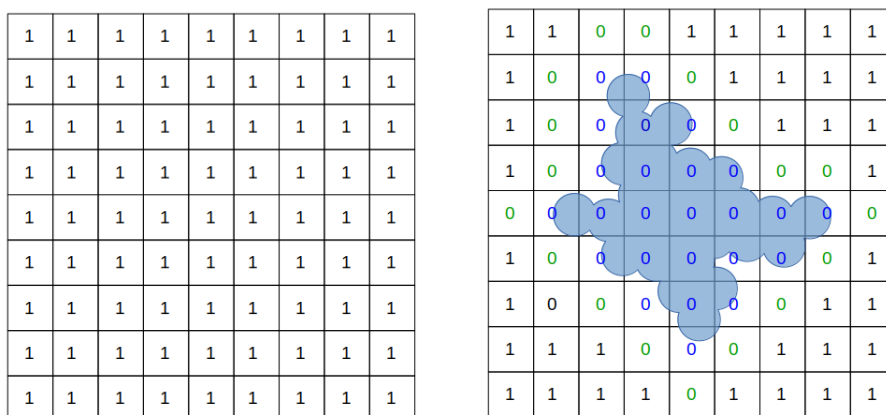
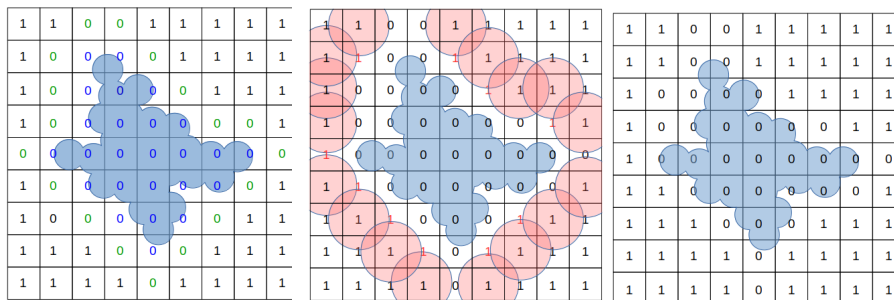


Abbildung 18: Grid vor und nach Einfügen des Proteins  
blau markiert Protein-Bereich, grün zusätzlichen Bereich im Probe Out-Radius

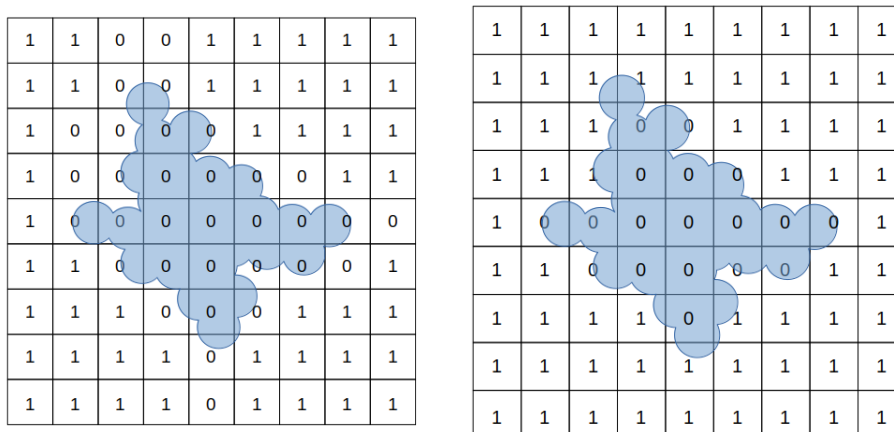
Im nächsten Schritt wird die Solvent Excluded Surface für die benutzerdefinierte Probe Out generiert. Dazu wird die Probe Sphere in jedes Voxel gesetzt, welches selbst nicht im zuvor markierten Bereich liegt, jedoch direkt zu diesem benachbart ist. Jedes zuvor markierte Voxel welches nun im Radius der Probe Sphere liegt wird wieder als nicht zum Protein gehörend markiert. Diesen Prozess veranschaulicht Abbildung 19.



**Abbildung 19:** Grid vor, während und nach Umlaufen mit Probe

Diese zwei Abläufe werden in einem zweiten Grid mit dem Probe In-Radius wiederholt, sodass zum Schluss zwei Voxelgrids mit zwei Oberflächenbeschreibungen vorliegen, eine Solvent Excluded Surface für die große Probe Out, und eine Solvent Excluded Surface für die kleine Probe In. Aus diesen Grids lassen sich schließlich die Kavitätenpunkte berechnen. Dazu wird die Probe Out erneut im Probe Out-Grid in alle Voxel gelegt, die nicht zum Protein gehören. Alle Voxel die im Radius der Probe Out liegen werden wieder markiert, diesmal jedoch im Probe In-Grid. Die übrig gebliebenen Voxel, die weder zum Protein gehören noch markiert wurden, bilden die Kavitäten.

Abbildungen 20 und 21 zeigen diese Operation.



**Abbildung 20:** Grids für Probe Out (links) und Probe In (rechts)

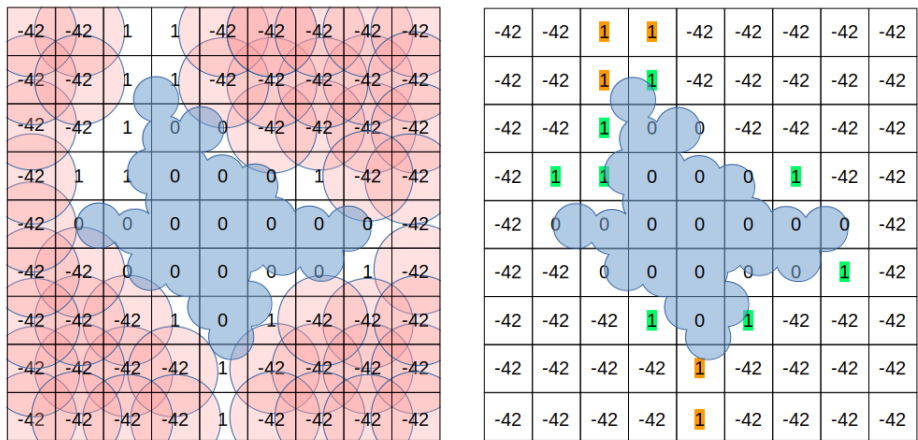


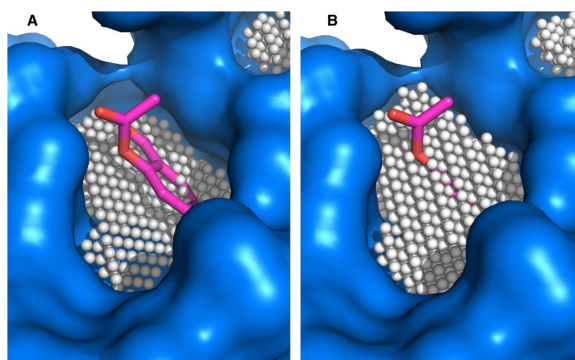
Abbildung 21: Markierung der nicht zu Kavitäten gehörenden Voxel

### 5.4.3 Vor- und Nachteile des Algorithmus

Wie zu erkennen ist, liefert dieser Algorithmus eine gute Annäherung an die Hohlräume, die man intuitiv vermuten würde. Da dieser Algorithmus jedoch mit Grids arbeitet, kann es leicht zu Rundungsfehlern kommen, sodass entweder Kavitätenpunkte gefunden werden können, die eigentlich keine sind oder auch Kavitätenpunkte fehlen, wo man sie erwarten würde. Dieses Phänomen wird auch in der 2D-Skizze 21 durch die orange markierten Kavitäten deutlich.

Dieses Problem kann durch eine feinere Abtastung des Grids gelöst werden, da das Ergebnis dadurch wesentlich präziser wird. Ein feiner unterteiltes Grid bedeutet aber gleichzeitig einen deutlich höheren Rechenaufwand, da auf Codeebene dreifach verschachtelte Schleifen durchlaufen werden. Es ist also notwendig eine Balance zwischen Genauigkeit und geringer Rechenzeit zu finden.

Ein weiterer Faktor, der das Ergebnis beeinflusst, ist der Radius der Probe Out. Je größer der Radius gewählt wird, desto näher befinden sich die Kavitäten an der Oberfläche. Möchte man also eher innere Hohlräume, wie Tunnel finden, muss der Radius entsprechend kleiner definiert werden, um das Innere des Proteins zu erreichen. Ist man hingegen an flacheren Taschen oder Kerben interessiert, ist ein größerer Radius von Vorteil. Abbildung 22 aus dem Paper KVFinder's visualisiert diesen Effekt.



**Abbildung 22:** Kavitätenberechnung mit Probe Out Radius von 3 Å (links) und 6 Å (rechts)

Es gibt also einige Parameter, die das Ergebnis positiv oder negativ beeinflussen können.

Gleichzeitig bieten diese Faktoren aber die Chance verschiedene Ergebnisse je nach Interesse des Benutzers zu generieren, was diesen Algorithmus sehr umfangreich und individuell macht. Daher ist es nur sinnvoll diese Parameter als Input-Variablen für den Benutzer zu definieren, um so effizient individuell zufriedenstellende Ergebnisse zu erzielen.

Des Weiteren sind Grid-Operationen leicht zu implementieren und erweiterbar. So ist es beispielsweise möglich, das Volumen einzelner Kavitäten zu berechnen und zu kleine und gegebenenfalls uninteressante Hohlräume herauszufiltern.

Ein weiterer Vorteil gegenüber anderen nicht-geometrischen Methoden ist, dass für diesen Algorithmus keine weiteren Datenbanken notwendig sind, die Informationen über die Wechselwirkungen zwischen funktionellen Gruppen oder Daten anderer Proteine speichern.

Aufgrund dessen baut das durch diese Arbeit entwickelte Programm *CavityDetector* auf diesen Algorithmus auf.

Das folgende Kapitel beschreibt die Entwicklung von *CavityDetector*, indem zunächst ein Überblick über die Funktionen des Programmes gegeben wird, daraufhin Grundlagen beschrieben werden und auf die Struktur eingegangen wird und abschließend generierte Ergebnisse bewertet und verglichen werden.

## 6 CavityDetector

### 6.1 Kurzbeschreibung des Tools

*CavityDetector* ist ein Tool zur Berechnung und Visualisierung von Hohlräumen eines Moleküls, welche während eines Dockingprozesses beobachtet werden können. Dazu erwartet das Programm als Eingabe zunächst

das Protein, dessen Kavitäten berechnet werden sollen, in Form einer pdb-Datei. Des Weiteren werden Parameter, wie der Probe-Out-Radius, die Schrittweite zum Abtasten des Grids, optimale Bindungsabstände und Toleranzbereiche, sowie Mindestvolumen der zu findenden Kavitäten durch den Benutzer definiert. Auf diese Eingaben wird im Kapitel 6.3.1 genauer eingegangen.

CavityDetector ermöglicht es dem Benutzer in Echtzeit das Anbinden eines Liganden an ein Protein zu simulieren. Dazu bewegt der Benutzer den Liganden mittels WASD-Steuerung in eine Kavität und ist in der Lage, anhand der Färbung der Kavitätenpunkte, abzuschätzen, ob ein Docking erfolgreich sein könnte.

CavityDetector ist ein OpenGL-basiertes Programm und wurde unter Zuhilfenahme des *ProjectBase* Frameworks mit dem QtCreator entwickelt. Das nachfolgende Kapitel beschreibt die Einrichtung der Entwicklungsumgebung.

## 6.2 Programmierumgebung

### 6.2.1 Systemvoraussetzungen

Um mit dem besagten Framework, welches durch die Forschungsgruppe der medizinischen Visualisierung der Universität Koblenz bereitgestellt wurde, arbeiten zu können, wird ein Linux-basiertes Betriebssystem vorausgesetzt. In diesem Fall wurde CavityDetector unter Ubuntu 16.10<sup>9</sup> entwickelt.

CavityDetector ist ein OpenGL-basiertes Programm. OpenGL ist eine Programmierschnittstelle, welche es ermöglicht mit Hilfe von Shadern direkt auf der GPU, der Grafikkarte, programmieren zu können und 2D- und 3D Computergrafikanwendungen zu entwickeln. [10]

Zur Entwicklung von CavityDetector wird mindestens OpenGL 4.3 benötigt, da die genutzten Shader mit sogenannten Shader Storage Buffer Objects arbeiten, welche erst mit der OpenGL-Shading-Language (GLSL) [11] Version 4.30 unterstützt werden. Diese ermöglichen es, dem Shader große Datenmengen bereitzustellen, ohne dabei eine fest definierte Größe zu haben. Des Weiteren sind Shader Storage Buffer Objects beschreibbar und bieten dadurch viele Möglichkeiten in der Shader Programmierung.

Welche Grafikkarten der führenden Hersteller OpenGL 4.3 unterstützen zeigt Abbildung 23.

---

<sup>9</sup><http://releases.ubuntu.com/16.10/>

### OpenGL 4.3 Support

<i>Nvidia</i>	> GeForce 400 series
<i>AMD</i>	> Radeon HD 5000 series
<i>Intel</i>	> Intel HD Graphics in Intel Haswell

Quelle: [https://en.wikipedia.org/wiki/OpenGL#OpenGL\\_4.3](https://en.wikipedia.org/wiki/OpenGL#OpenGL_4.3)

**Abbildung 23:** OpenGL 4.3 Support führender Grafikkartenhersteller

### 6.2.2 QtCreator

Erfüllt das System alle genannten Voraussetzungen muss eine Entwicklungsumgebung eingerichtet werden. Hierzu gibt es einige Möglichkeiten, wie beispielsweise Eclipse<sup>10</sup>, CodeLite<sup>11</sup> oder Netbeans<sup>12</sup>.

Im Falle von CavityDetector wurde die IDE QtCreator gewählt, da diese neben einem übersichtlichen Editor und einem hilfreichen Debug-Modus, einen eigenen Compiler besitzt und die Projekterstellung unter CMake unterstützt.

Alle benötigten Pakete zur Installation sind hier [34] beschrieben.

### 6.2.3 CMake

Neben einer Entwicklungsumgebung ist auch die Erstellung der Projektumgebung notwendig. Dazu sollte das Build-System *CMake* installiert sein. CMake erstellt anhand der vorliegenden Projektdateien ausführbare Projekte, welche in einem separaten Verzeichnis abgelegt werden. Dieser Prozess beruht auf Textdateien, den *CMakeLists.txt*, welche das Projekt definieren. Diese müssen im Root-Verzeichnis und in Unterverzeichnissen vorhanden sein.

Beim Erstellen des Projektes können verschiedene Build-Types gewählt werden. Der Modus „Debug“ ermöglicht beispielsweise das ordentliche Debuggen des Projektes. [33]

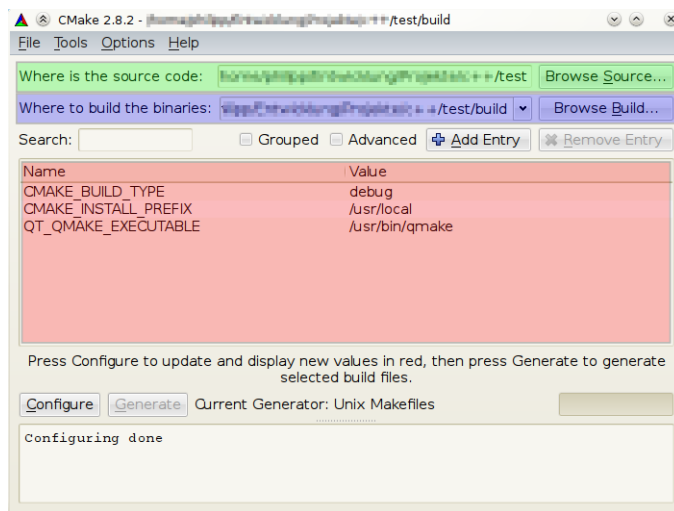
Abbildung 24 zeigt die grafische Benutzeroberfläche von CMake mit Quell- und Zielverzeichnissen, sowie den verschiedenen Parametern. CMake kann jedoch auch über die Konsole benutzt werden.

<sup>10</sup><https://eclipse.org/downloads/>

<sup>11</sup><https://codelite.org/>

<sup>12</sup><https://netbeans.org/>





**Abbildung 24:** Grafische Benutzeroberfläche von CMake  
<https://media-cdn.ubuntu-de.org/wiki/attachments/50/43/cmake-gui.png>

Um das Projekt nun im QtCreator zu bearbeiten, muss es geöffnet werden, indem die Basis-CMakeLists.txt importiert wird. Über die Funktion „Run CMake“ kann der beschriebene Build-Prozess dann initialisiert und das Projekt schließlich ausgeführt werden.

## 6.2.4 ProjectBase Framework

CavityDetector wurde mit Hilfe des bereits existierenden *ProjectBase* Framework entwickelt, welches durch die Medizinische Visualisierung der Universität Koblenz-Landau bereitgestellt wurde.

Dieses stellt bereits viele Klassen und Funktionen bereit, die für die Entwicklung von CavityDetector hilfreich gewesen sind. Dazu gehören beispielsweise Klassen für Kameras, Shadergenerierung, Shader Storage Buffer Objects oder auch Shader für Order-Independent-Transparencies. Auf einzelne Aspekte wird in den folgenden Sektionen näher eingegangen. Das Framework kann über die Versionsverwaltungsanwendung GitLab kostenfrei aufgerufen werden <sup>13</sup>.

Damit alle Komponenten richtig eingebunden werden, müssen zudem die Anweisungen, die in der Read Me-Datei des Projektes beschrieben werden, ausgeführt werden, sodass es schließlich als CMake-Projekt die Basis für die weitere Implementierung bilden kann.

Der nächste Abschnitt beschreibt den Aufbau von CavityDetector und die Funktionen der einzelnen Klassen.

<sup>13</sup>[https://gitlab.uni-koblenz.de/MedVis/Project\\_Base](https://gitlab.uni-koblenz.de/MedVis/Project_Base)

### 6.3 Klassenhierarchie

CavityDetector setzt sich aus drei Klassen zusammen. Die Hauptklasse *CavityDetector* ist das „Herzstück“ des Projektes. Hier werden alle Eingaben des Benutzers verwaltet. Darauf aufbauend werden alle nötigen Komponenten initialisiert. Dazu gehören zum Einen zwei Instanzen der *PDBloader*-Klasse, sowie eine Instanz der *CavityCalc*-Klasse, welche Protein und Ligand laden und die Kavitäten berechnen. Darüber hinaus enthält die *CavityDetector*-Klasse die *main*-Methode, den Eintrittspunkt eines jeden Programmes.

Abbildung 25 gibt einen reduzierten Überblick über die Klassenhierarchie des Projektes.

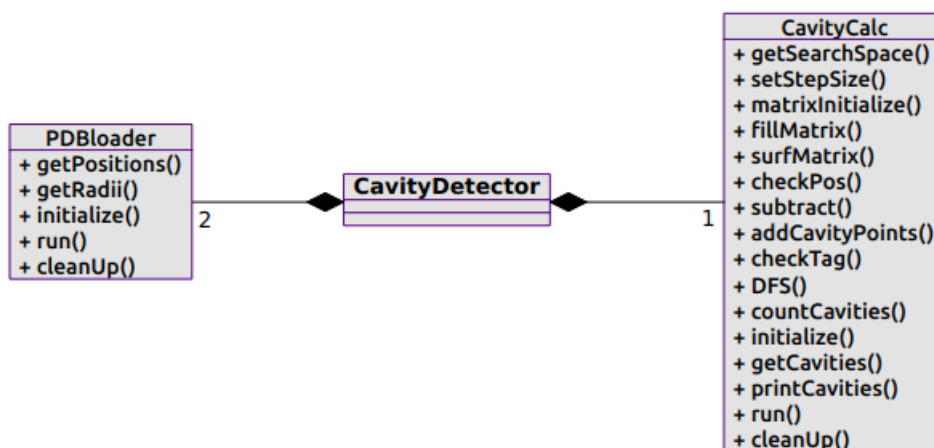


Abbildung 25: Klassenhierarchie von CavityDetector

Im Folgenden wird die Pipeline von CavityDetector beschrieben, beginnend bei den Eingaben durch den Benutzer, über das Laden von Protein und Ligand bis zu der Berechnung der Kavitäten.

#### 6.3.1 Benutzerabfragen

CavityDetector baut auf einen Algorithmus auf, der von vielen Parametern abhängt. Dadurch lassen sich viele verschiedene Ergebnisse erzeugen, die nicht per se in „besser“ und „schlechter“ kategorisiert werden können. Der Nutzen des Ergebnisses hängt ganz davon ab, woran der Benutzer interessiert ist. Daher werden die wichtigsten Parameter zu Beginn durch den Nutzer definiert. Folgende Eingaben werden erwartet:

- Name of Protein-file:  
Hier wird der Name des Proteins im pdb-Format erwartet. Die Datei muss sich dabei im resources-Ordner befinden.
- Name of Ligand-file:  
Gleichermaßen wird hier der Name des Liganden im pdb-Format erwartet. Auch diese Datei muss im resources-Ordner vorhanden sein.
- Stepsize:  
Hier wird ein float-Wert erwartet. Dieser beschreibt in welchem Abstand die Grids zur Berechnung der Kavitäten abgetastet werden. Einflüsse der Schrittweite auf die Ergebnisse werden im Evaluations- teil beschrieben.
- Size of outer probe:  
Auch hier wird ein float-Wert erwartet. Dieser beschreibt den Radius der ProbeOut-Sphere, die zur Berechnung der Kavitäten benötigt wird. Je größer der Radius, desto länger dauert die Berechnung und desto oberflächigere Kavitäten werden gefunden. Einflüsse dieses Wertes werden ebenfalls im Evaluationsteil erörtert.
- Minimum Volume of cavities:  
Mit Hilfe dieser float-Wert-Eingabe kann der Benutzer festlegen, wie groß die zu findenden Kavitäten sein sollen. Ist er nur an großen und tiefen Hohlräumen interessiert, können dadurch kleine Kavitäten herausgefiltert werden.
- Optimal binding distance:  
Damit ein Docking von Proteinen und Liganden erfolgreich ist, müssen diese miteinander interagieren. Wechselwirkungen, wie Salzbrücken oder Wasserstoffbrücken wirken in bestimmten Abständen. Diesen Abstand kann der Benutzer zu Beginn über die Eingabe eines float-Wertes frei definieren.
- Acceptable difference from optimal binding distance:  
Mit diesem float-Wert kann ein Toleranzbereich definiert werden, wie weit der Abstand vom zuvor definierten optimalen Wert abweichen darf, sodass ein erfolgreiches Docking noch möglich ist.

Abbildung 26 zeigt die Eingabeaufforderung in der Konsole.

```

Name of protein-file:
5iksNoRes.pdb
Name of ligand-file:
obj01.pdb
Stepsize:
0.5
Size of outer probe:
4
Minimum Volume of cavities:
10
Optimal binding distance:
2.7
Acceptable difference from optimal binding distance:
0.1

```

Abbildung 26: Erwartete Eingaben durch den Benutzer

Nachdem alle Eingaben erfolgt sind, kann das Protein geladen werden. Wie dies funktioniert, beschreibt der nächste Abschnitt.

### 6.3.2 PDBLoader

Das Laden eines Proteins erfolgt durch die *Easy Structural Biology Template Library*, kurz *ESBTL*. Diese header-only Bibliothek stellt einen pdb-Parser und einige Datentypen und Funktionen bereit, um alle Informationen aus einer pdb-Datei herauslesen und verarbeiten zu können.[19] Um über die Atome einer pdb-Datei iterieren zu können, müssen zunächst einige Grundlagen definiert werden und eine Datenstruktur aufgebaut werden.

Zunächst muss die sogenannte *Occupancy-Behandlung* definiert werden. Bei Seitenketten von Aminosäuren kommt es vor, dass sie, aufgrund von lokaler Flexibilität, unterschiedliche Anordnungen annehmen können. Dadurch ergeben sich verschiedene Elektronendichten, die bei Kristallstrukturanalysen mit Hilfe des sogenannten *Occupancy*-Wertes beschrieben werden. Der Occupancy-Wert der Atome, die Teil solcher zusätzlich möglichen Konformationen sind, ist immer kleiner 1 und beschreibt die Wahrscheinlichkeit des Auftretens. [23]

Die Occupancy-Behandlung muss bei ESBTL selbst definiert werden. Im Falle von CavityDetector werden neben allen Atomen, mit einem Occupancy-Wert von 1, also den in jedem Fall vorkommenden Strukturen, die Atome mit den maximalen Auftretswahrscheinlichkeiten geladen. Das geladene Protein entspricht also der wahrscheinlichsten Konformation. Abbildung 27 zeigt, wo diese Werte in einer pdb-Datei gespeichert werden.

ATOM	14	CD1	LEU	A	2	-14.195	10.241	-5.636	1.00	19.49	C
ATOM	15	CD2	LEU	A	2	-15.269	10.915	-7.793	1.00	19.75	C
ATOM	16	N	SER	A	3	-18.843	12.684	-4.437	1.00	18.98	N

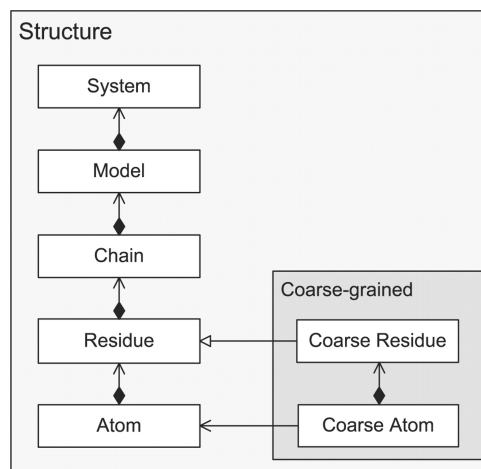
**Abbildung 27:** Occupancy-Wert in einer pdb-Datei

Des Weiteren muss ein *Atom-Classifier* definiert werden. Dieser stellt die Verknüpfung zwischen einer Eigenschaft und einem Objekt her. Diese Eigenschaften können zum Beispiel Farben oder Namen von Atomen sein. Im Falle des *CavityDetectors* wird der Classifier so definiert, dass ein Zugriff auf den Radius eines Atoms ermöglicht wird, da dieser sowohl zum Darstellen des Proteins, als auch für die Kavitätenberechnung wichtig ist. Abbildung 28 zeigt die Verwendung eines solchen Classifiers.

```
typedef ESBTL::Generic_classifier<ESBTL::Radius_of_atom<double,ESBTL::Default_system::Atom> > T_Atom_classifier;  
T_Atom_classifier atom_classifier;  
radius = atom_classifier.get_properties(*current_atom).value();
```

**Abbildung 28:** Definition und Verwendung eines ESBTL-Classifiers

ESBTL arbeitet mit einer eigenen hierarchischen Datenstruktur. Diese kann Abbildung 29 entnommen werden.



**Abbildung 29:** Hierarchische Datenstruktur von ESBTL

<https://academic.oup.com/bioinformatics/article/26/8/1127/206699/>  
ESBTL-efficient-PDB-parser-and-data-structure-for

Ein *System* bildet gewissermaßen einen Container für ein oder mehrere *Models*, die Proteine. Jedes *Model* umfasst *Aminosäureketten*, *Reste* und *Atome*. Um eine *pdb*-Datei nun lesen zu können, muss diese Datenstruktur aufgebaut werden.

Zunächst wird ein *Line Selector* benötigt, der definiert, woraus sich ein *System* zusammensetzt. Des Weiteren wird ein Container in Form eines einfachen Vectors für erstellte *Systeme* benötigt. Mit Hilfe eines *Builders* kann daraufhin ein *System* im zuvor definierten Container erstellt werden.

ESBTL stellt darüber hinaus einen *Iterator* bereit, welcher es ermöglicht über das Model, also das Protein, des erstellten Systems zu iterieren, um so für jedes Atom die spezifischen Koordinaten und Radien herauszufinden und zu speichern. Darüber hinaus bietet die Bibliothek viele Funktionen, die den Umgang mit einem Protein erleichtern.

So nutzt *CavityDetector* beispielsweise die Abfrage, ob es sich bei einem Atom um Teil eines Wassermoleküls handelt, um diese sofort beim Laden des Proteins herauszufiltern. Darüber hinaus werden auch die Namen der Aminosäuren abgefragt, um für die Atome sofort die entsprechenden Farben zu speichern.

*CavityDetector* realisiert dadurch zwei Farbmodi zur Darstellung des Proteins, zwischen welchen beliebig gewechselt werden kann, dem Mono-Farbmodus und dem Aminosäuren-Farbmodus für das Protein beziehungsweise dem Atom-Farbmodus für den Liganden.

Ein Überblick über alle Funktionen der Easy Structural Biology Template Library und hilfreiche Tutorials finden sich in der Dokumentation <sup>14</sup>.

Sobald das Protein und, auf gleichem Wege, der Ligand erfolgreich geladen wurden und alle Informationen zu allen Atomen, ihren Koordinaten und Radien vorliegen, kann die Berechnung der Kavitäten beginnen.

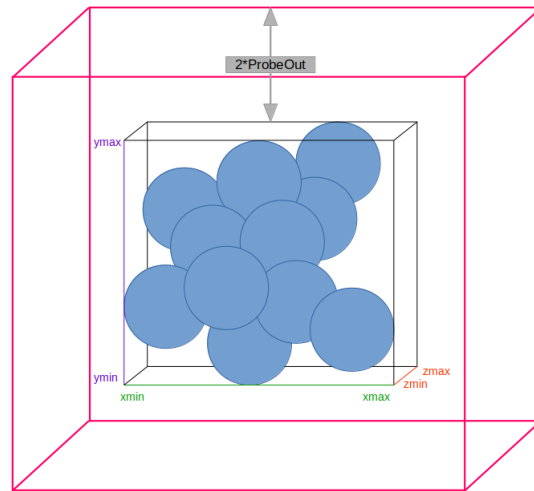
### 6.3.3 CavityFinder

Die Klasse *CavityFinder* umfasst sowohl die Berechnung aller Hohlräume, als auch die Berechnung der Volumen einzelner Kavitäten.

**Kavitätenberechnung** Dazu muss zu Beginn der Suchraum definiert werden. Dieser wird so gewählt, dass das Protein genau mittig im aufgespannten Grid liegt. Um dies zu gewährleisten, wird über alle Atome iteriert und mit Hilfe der minimalen und maximalen x-, y- und z-Koordinaten eine *Bounding Box* um das Protein berechnet. Diese wird daraufhin noch um den zweifachen *ProbeOut*-Radius in alle Dimensionen vergrößert, um sicherzustellen, dass im weiteren Verlauf während des *ProbeOut*-Rollings das Grid nicht verlassen werden kann und *Out-of-Range-Exceptions* vermieden werden. Diese Vorbereitung für den eigentlichen Algorithmus wird in Abbildung 30 veranschaulicht.

---

<sup>14</sup>ESBTL: <http://esbtl.sourceforge.net/refman/doc/html/index.html>



**Abbildung 30:** Bestimmung des Suchraumes

Aus den daraus neu entstandenen  $x$ -,  $y$ - und  $z$ - Minima und Maxima werden die Dimensionen berechnet, die zur Initialisierung zweier dreidimensionaler `std-Vektoren`, *OuterGrid* und *InnerGrid*, benötigt werden. Diese repräsentieren die Grids, auf welche der Algorithmus von *KVFinder* aufbaut, welcher in Kapitel 5.4.2 beschrieben wurde. Dieser wird mit Hilfe von drei Funktionen realisiert.

Die Funktion *fillMatrix* markiert zunächst das Grid im Bereich des Proteins, erweitert um den Radius der Probe Sphere. Für das *OuterGrid* wird dazu mit der *ProbeOut* gearbeitet, während für das *InnerGrid*, der feste Radius von  $1.4 \text{ \AA}$  genutzt wird.

Die Funktion *surfMatrix* iteriert daraufhin über alle an den markierten Bereich angrenzenden Voxel und markiert alle Voxel im Radius der *ProbeOut* für das *OuterGrid*, bzw. von  $1.4 \text{ \AA}$  für das *InnerGrid*. Diese Funktion implementiert somit anschaulich das Rollen der *ProbeSphere* um das Protein. Nachdem diese beiden Funktionen für beide Grids ausgeführt wurden, liegen zwei Oberflächenbeschreibungen des Proteins vor, welche im Folgenden miteinander verrechnet werden können.

Dieser Schritt erfolgt mit der Funktion *subtract*, welche die Kavitätenpunkte im *InnerGrid* markiert. Die Funktion *addCavityPoints* ermittelt daraufhin aus den markierten Voxeln, die genauen Kavitätenpunkte. Diese ergeben sich aus den Zentren der Kavitätenvoxel. Diese Positionen werden in einem Buffer abgespeichert. Darüber hinaus, wird auch die Farbe der Kavitätenpunkte bestimmt. Ist der Abstand zum Protein im optimalen Bereich, der durch den Benutzer definiert wird, so wird ein Grünton für diesen Kavitätenpunkt gespeichert, andernfalls ein Rot- bzw. Violettton.

**Volumenberechnung** Die Klasse `CavityCalc` beinhaltet zudem noch Funktionen, um zusammenhängende Kavitätenpunkte zu markieren, zu zählen und so die Volumen einzelner Kavitäten zu berechnen. Dies wird mit einer Tiefensuche realisiert, und wird so auch in `KVFinder` benutzt.

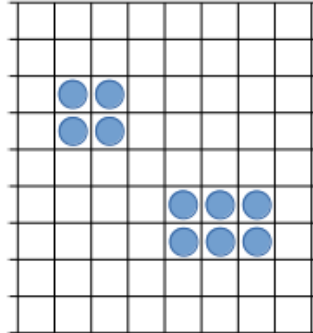
Die Funktion `countCavities` läuft über das gesamte `InnerGrid`, welches die Kavitäten beinhaltet, und führt für jedes Voxel die Tiefensuche `DFS` aus. Ist der aktuelle Punkt ein Kavitätenpunkt, so wird er „getagged“ und bekommt eine Nummer. Dies wird für alle benachbarten Voxel, und deren Nachbarn durchgeführt, die, sofern sie ebenfalls ein Kavitätenpunkt sind, mit der gleichen Nummer tagged werden. Kann kein Voxel mehr in der direkten Nachbarschaft gefunden werden, welches ein Kavitätenpunkt ist und noch nicht markiert wurde, wird die Nummer erhöht und es wird nach neuen Kavitäten gesucht, die tagged werden können. So besitzt jede Kavität zum Schluss eine andere Nummer und es lässt sich leicht zählen, aus wievielen Punkten die Hohlräume bestehen. Dieser Algorithmus wird in [Abbildung 31](#) vereinfacht dargestellt.



Ausgangssituation:

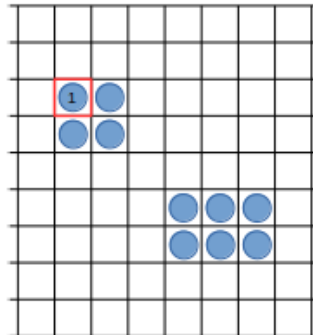
Zwei Kavitäten

Tag = 1  
Volume = 0



Step 1:

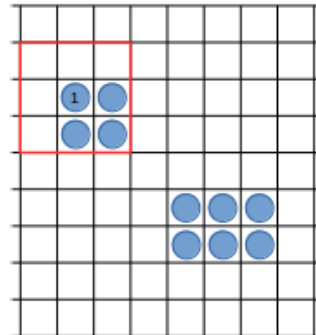
Tag = 1  
Volume = 1



Step 2:

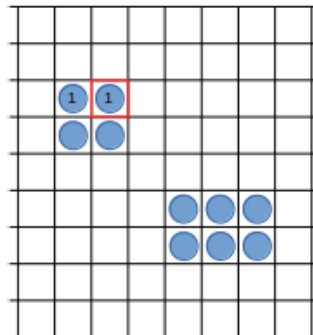
Tag = 1  
Volume = 1

Überprüfung aller  
Nachbarn



Step 3:

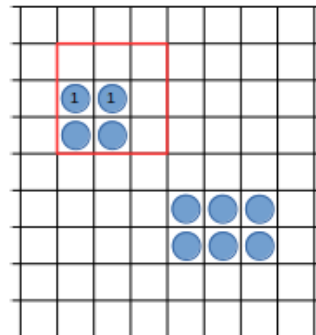
Tag = 1  
Volume = 2

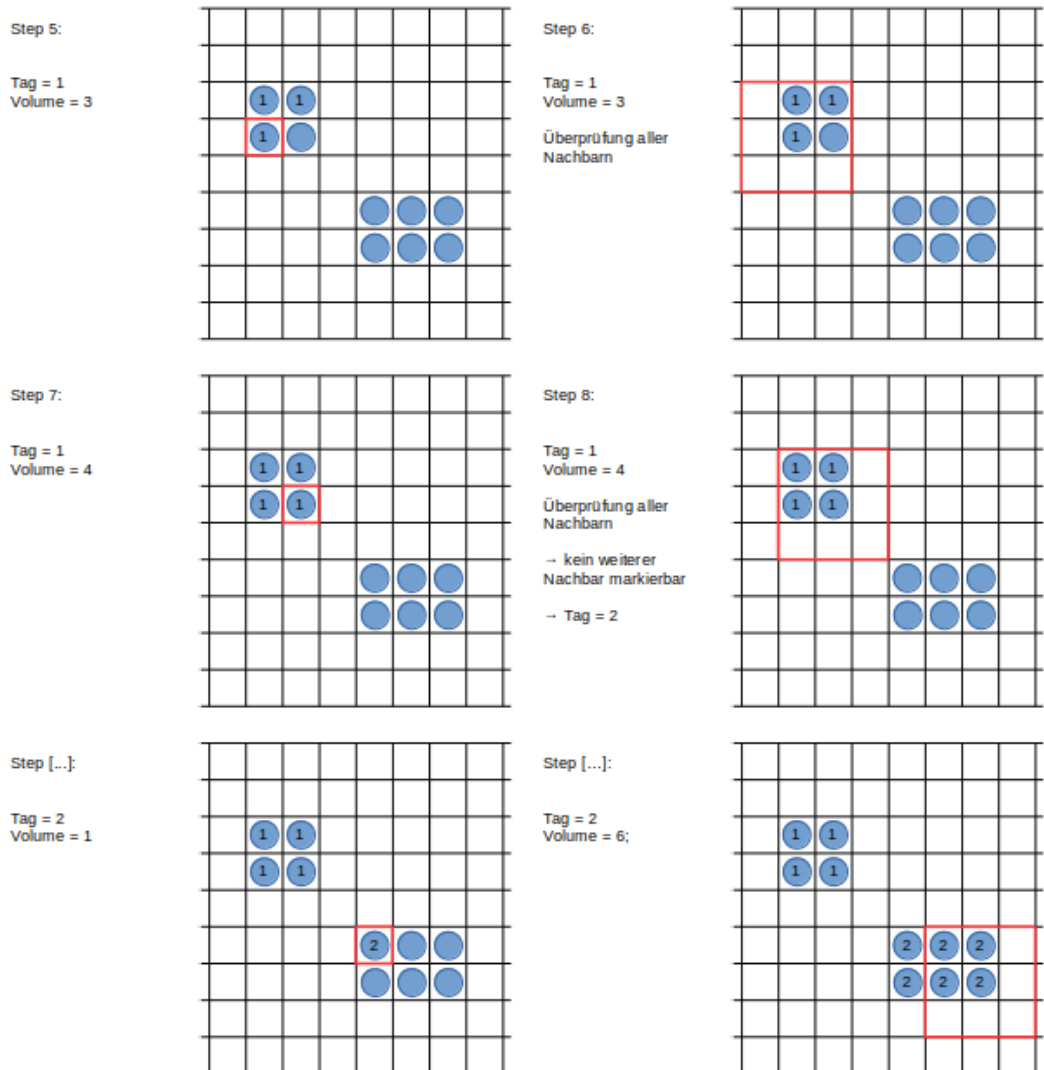


Step 4:

Tag = 1  
Volume = 2

Überprüfung aller  
Nachbarn





**Abbildung 31:** Skizze der Tiefensuche zur Berechnung der Kavitätenvolumen

Mit dieser Kenntnis kann nun auch das Filtern der Kavitäten umgesetzt werden. Bevor die Koordinaten der Punkte in Buffern gespeichert werden, wird überprüft, ob die zugehörige Kavität, das durch den Benutzer definierte Mindestvolumen besitzt. Falls nicht, werden diese Punkte einfach übersprungen und somit nicht gerendert.

## 6.4 Shader

Nachdem alle Informationen, wie Positionen von Atomen, Ligand und Kavitätenpunkten, sowie Farben vorliegen, ist der nächste Schritt, diese Da-

ten zu rendern. Dies sollte im besten Fall auf eine Weise geschehen, die eine echtzeitfähige Interaktion zulässt.

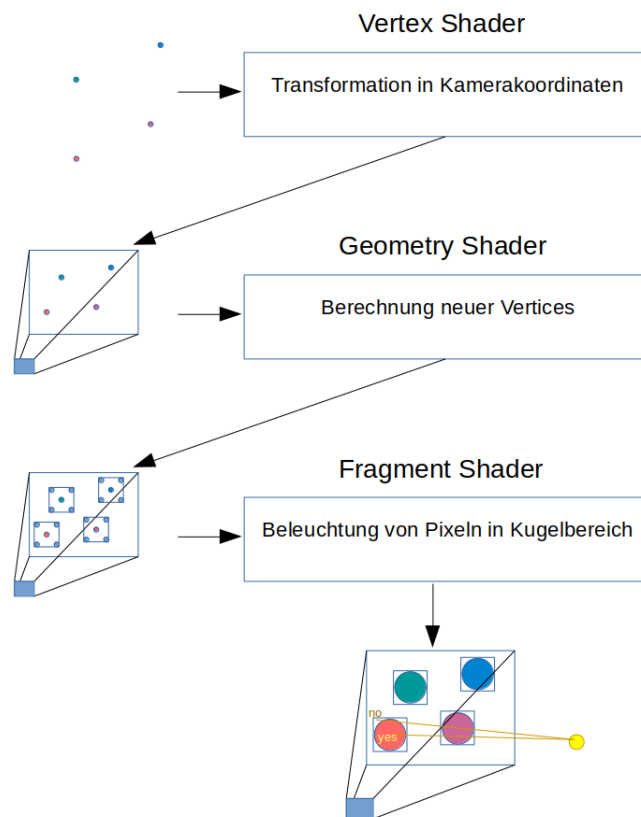
Proteine können riesige Datenmengen mit sich bringen. Das größte Protein Titin besteht beispielsweise aus etwa 30000 Aminosäuren.[16] Selbst wenn jede Aminosäure nur aus 10 Atomen bestünde, wie es bei der kleinsten Aminosäure Glycin der Fall ist, zählte Titin mindestens 300000 Atome. Darüber hinaus liefert die Kavitätenberechnung zusätzlich noch tausende Punkte, die als Kugel gerendert werden sollen. Selbst bei Proteinen mit nur wenigen tausend Atomen, sind sehr schnell Grenzen erreicht, sodass es nicht möglich ist, jedes Atom und jeden Kavitätenpunkt als echte Kugel zu rendern.

Als Lösung dieses Problems wurde CavityDetector mit Hilfe eines *ImpostorShaders*, entwickelt.

#### 6.4.1 ImpostorShader

Der Impostor Shader setzt sich aus drei Shaderprogrammen zusammen und bedient sich eines Tricks, der ein effizientes Rendern einer Kugel ermöglicht. Dazu werden lediglich die Zentren der Kugeln an den Shader gegeben, aus welchen dann Würfel mit entsprechenden Kantenlängen generiert werden. Indem aber nur bestimmte Pixel beleuchtet werden, entsteht der Eindruck einer echten Kugel. Abbildung 32 zeigt die Idee dieses Shaders.

Dieses Prinzip läuft in drei Shaderstufen ab.



**Abbildung 32:** Schematische Darstellung des Impostor Shaders

1. **Vertex Shader:**  
Der Vertex Shader bekommt als Input lediglich die Koordinaten der einzelnen Atome, beziehungsweise der Kavitätenpunkte, also die Zentren der später sichtbaren „Kugeln“. Diese werden im Vertex Shader lediglich in Kamerakoordinaten transformiert, also mit der view-Matrix multipliziert, und zusammen mit Farbe und Radius in die Shader Pipeline weitergeleitet.
2. **Geometry Shader:**  
Im Geometry Shader werden zusätzliche Vertices generiert, sodass ein Würfel entsteht, dessen Kantenlänge dem Durchmesser der finalen Kugel entspricht. Diese neuen Vertices, sowie Radius und Farbe werden an die nächste und letzte Shader Stufe weitergegeben.
3. **Fragment Shader:**  
Im Fragment Shader wird die finale Beleuchtung für jedes Pixel berechnet, so dass nur die Pixel der Würfel dargestellt werden, die in den Bereich einer Kugel fallen. Dazu wird getestet, ob es Schnittpunkte zwischen Sehstrahl und der theoretischen Kugel gibt, und anhand

dessen entschieden, ob ein Pixel zur Kugel gehört oder nicht. Hierbei handelt es sich um ein mathematisches Problem, die *Line Sphere Intersection*, welches sich mit einigen geometrischen Kenntnissen lösen lässt.

**Line Sphere Intersection** Um auf Schnittpunkte zwischen geometrischen Objekten zu testen, ist zu Beginn wichtig zu wissen, wie diese definiert werden. Im Falle des Impostor Shaders werden Linien und Kugeln benötigt. Die Gleichung für eine Kugel lautet wie folgt:

$$\begin{aligned} \|x - c\|^2 &= r^2, \\ x &= \text{Punkt auf der Kugel} \\ c &= \text{Zentrum der Kugel,} \\ r &= \text{Radius der Kugel} \end{aligned} \tag{1}$$

Die Gleichung für eine Linie sieht folgendermaßen aus:

$$\begin{aligned} x &= o + dl, \\ x &= \text{Punkt auf der Linie} \\ d &= \text{Distanz von Ursprung bis Punkt auf der Linie} \\ l &= \text{Richtung der Linie,} \\ o &= \text{Ursprung der Linie} \end{aligned} \tag{2}$$

Um Schnittpunkte zu berechnen, müssen beide Gleichungen kombiniert werden, da Punkte gesucht werden, die sowohl auf der Linie, als auch auf der Kugel liegen. Dies liefert, durch Einsetzen der Liniengleichung in die Kugelgleichung, folgendes Ergebnis:

$$\|o + dl - c\|^2 = r^2 \tag{3}$$

Multipliziert man diese Gleichung aus und formt sie um, ergibt sich eine quadratische Gleichung:

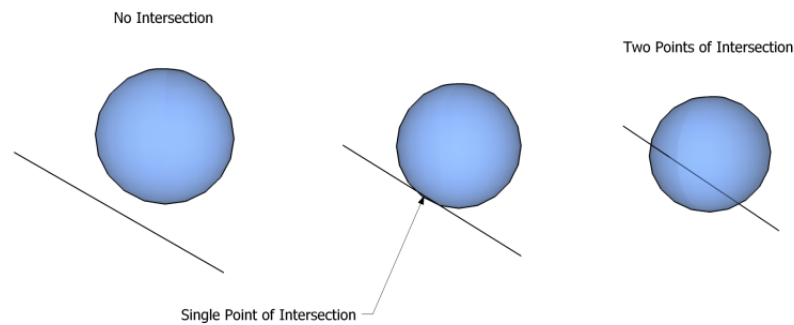
$$\begin{aligned} ad^2 + bd + c &= 0, \\ a &= l * l \\ b &= 2(l * (o - c)) \\ c &= \|o - c\|^2 - r^2 \end{aligned} \tag{4}$$

Mit Hilfe der pq-Formel und der Voraussetzung, dass  $l$  ein Einheitsvektor ist, kann eine Gleichung für  $d$  aufgestellt werden.

$$d = -(1 * (o - c)) \pm \sqrt{(l * (o - c))^2 - \|o - c\|^2 + r^2} \tag{5}$$

Anhand dieser Formel kann geprüft werden, ob ein Schnittpunkt vorliegt. Abbildung 33 zeigt die möglichen Situationen. Liefert das Einsetzen eines

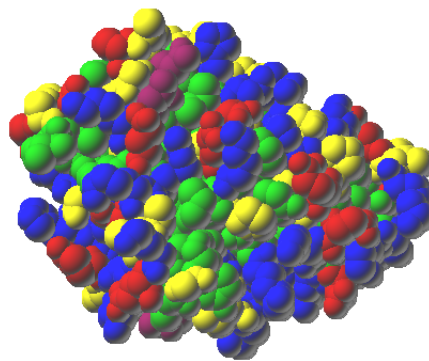
Punktes einen Wert kleiner 0 unter der Wurzel, so ist klar, dass kein Schnittpunkt vorliegen kann. Im Falle des Impostor Shaders, kann dieses Pixel also verworfen werden und wird nicht beleuchtet.



**Abbildung 33:** Mögliche Ergebnisse beim Schnitttest von Kugel und Linie  
[https://upload.wikimedia.org/wikipedia/commons/6/67/Line-Sphere\\_Intersection\\_Cropped.png](https://upload.wikimedia.org/wikipedia/commons/6/67/Line-Sphere_Intersection_Cropped.png)

Ist das Ergebnis unter der Wurzel 0 oder größer als 0, so liegt mindestens ein Schnittpunkt vor und das Pixel kann mit Phong beleuchtet werden, sodass ein Eindruck einer korrekten Kugel entsteht (vgl. zu diesem Kapitel [9]).

CavityDetector beruht auf einer bereits bestehenden Implementierung dieses Shaders, welcher ursprünglich für eine perspektivische Kamera geschrieben wurde und für eine orthographische Kamera abgeändert wurde. Abbildung 34 zeigt ein fertig auf diese Weise gerendertes Protein.

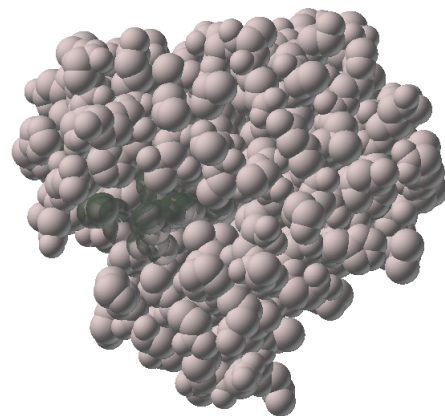
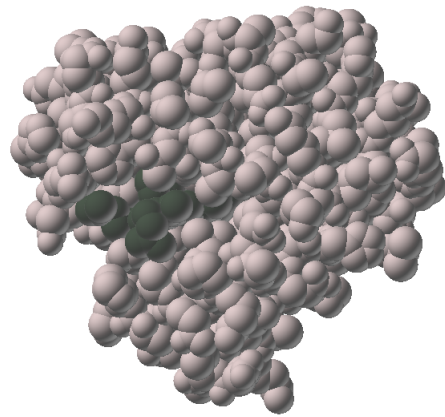


**Abbildung 34:** Protein 5iks mit Impostor gerendert

#### 6.4.2 Order Independent Transparency-Shader

Für eine bessere Orientierung und eine einfachere Nutzung werden sowohl die Kavitätenpunkte, als auch der Ligand während des Dockings teilwei-

se transparent gerendert. Dadurch ist es leichter zu erkennen, wo sich der Ligand aktuell befindet, wie in Abbildung 35 deutlich wird.



**Abbildung 35:** Ligand ohne und mit Transparenz

Beim Rendern von transparenten und opaquen Objekten wird in der Regel mit sogenanntem *Blending* gearbeitet. Das bedeutet, dass wenn man durch ein transparentes Objekt schaut, auch die Farbe dahinterliegender Objekte wahrgenommen wird. Wie diese Farben miteinander verrechnet werden, bestimmt die *Blend-Funktion*, die anhand der Alpha-Werte der Objekte  $(r,g,b,\alpha)$  die finale Pixelfarbe berechnet.

Werden mehrere transparente Objekte hintereinander gerendert, so macht es einen Unterschied in welcher Reihenfolge sie gerendert werden. Um ein korrektes Bild zu generieren, müssen diese Objekte nach ihren Tiefenwerten sortiert werden.

Da dieses Verfahren komplex ist, und auch einige Zeit dauert, nutzt *CavityDetector* ein Verfahren, welches unabhängig von der Reihenfolge der Objekte funktioniert.

*Order Independent Transparency* sortiert die Objekte nach der Rasterisierung

pixelweise, weshalb alle Fragmente bereits vor der Sortierung vorliegen müssen. Um dies zu realisieren wird Render-To-Texture benötigt und somit ein zweimaliges Rendern der transparenten Objekte.

Im ersten Renderdurchlauf werden die Atome bzw. Kavitäten mit dem Impostor Shader gerendert, wobei die Pixelwerte in einer Textur abgespeichert werden.

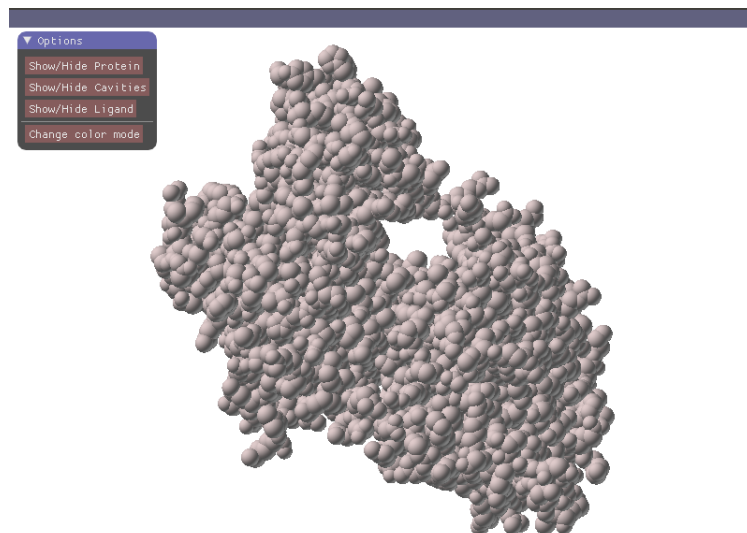
Diese Textur wird im zweiten Durchlauf mit dem Order-Independent-Transparency-resolve Shader gerendert, der bereits im ProjectBase Framework integriert ist. Dieser sortiert die Pixelwerte und berechnet schließlich die korrekte Farbe.

Wie das fertige Resultat aussieht und wie der Umgang mit CavityDetector funktioniert, wird in den nächsten Abschnitten beschrieben.

## 6.5 Ergebnis und Interaktion

Die folgenden Ergebnisse zeigen ein  $G\alpha_q$ -Rezeptorprotein. Dieses wird, wie Pharmazeuten aus Bonn berichten, aktuell in der Forschung intensiv untersucht, wobei besonderer Fokus auf der Interaktion mit dem Liganden YM-254890 liegt, da dieser aufgrund seiner Eigenschaften als „perfekter“ Ligand beschrieben wird.

Als Grundlage für den Algorithmus wurde ein Radius der ProbeOut von 4 Å und eine Schrittweite von 0.5 gewählt. Mit Hilfe der beschriebenen Techniken ergibt sich das folgende Bild, nach Laden des Proteins 36.

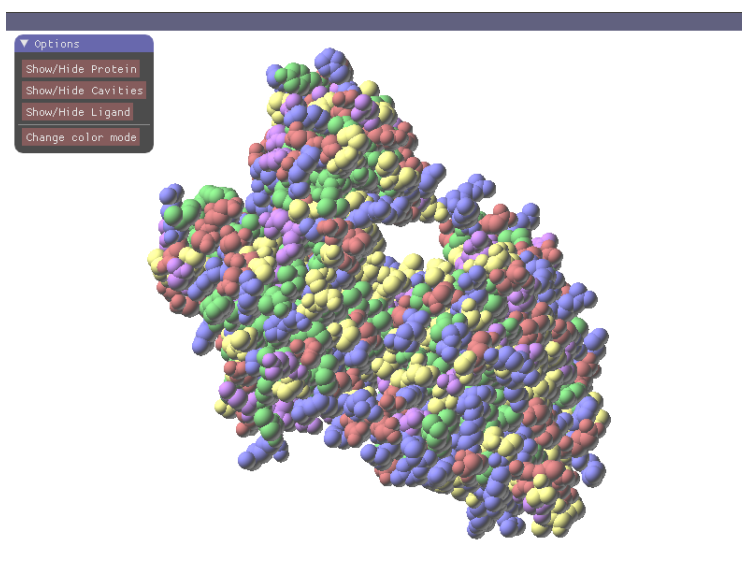


**Abbildung 36:** Oberfläche CavityDetector

Die Darstellung im Gesamten ist sehr aufgeräumt. Neben vier Button, ist lediglich das geladene Protein zu sehen. In der Default-Ansicht besitzt das



Protein eine einheitliche Farbe, sodass später eingeblendete Kavitätenpunkte gut hervorgehoben werden. Dieser Farbmodus kann jedoch über den Button „Change color mode“, oder die Taste „o“ geändert werden, sodass die Farben daraufhin die entsprechenden Aminosäurezugehörigkeiten veranschaulichen.

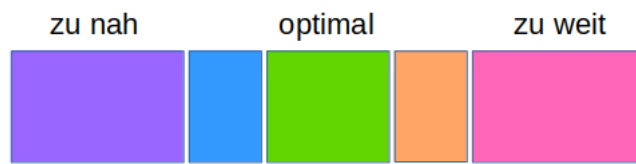


**Abbildung 37:** Aminosäurefarbmodus

Mit Hilfe der Maus kann die Ansicht des Proteins rotiert, sowie heran- und herausgezoomt werden. Dies erfolgt über die gedrückte linke Maustaste und das Bewegen der Maus, beziehungsweise das Scrollen des Mausekranzes. Über die Tasten 1,2,3 und 5 des Nummernblocks kann die Kamera außerdem nach links/rechts und oben/unten bewegt werden.

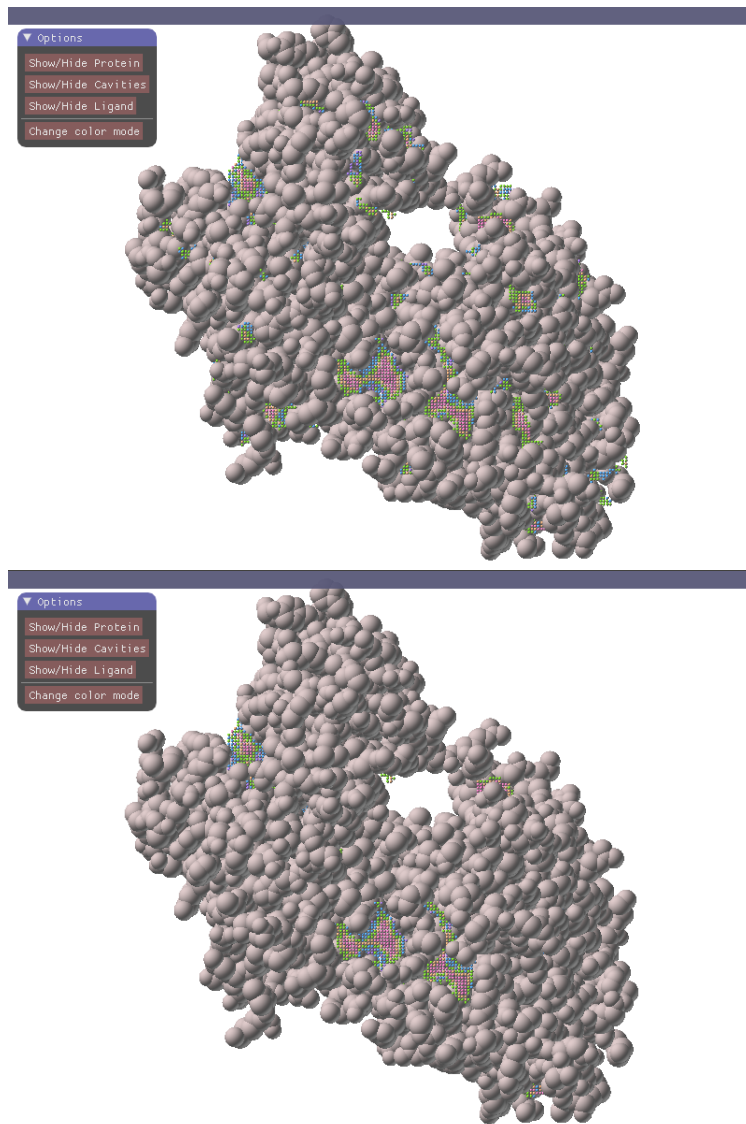
Über den Button „Show cavities“ oder die Taste „c“ werden die anhand der durch den Benutzer definierten Parameter berechneten Kavitätenpunkte eingeblendet.

Die Farben der einzelnen Punkte geben hierbei Aufschluss über den Abstand zum nächsten Atom des Proteins. Hierbei werden diskrete Farbabstufungen genutzt, die die Abstände zum Protein kodieren. Rot-Pink wird für weit entfernte Kavitätenpunkte genutzt, grün symbolisiert den optimalen Abstand und violett zeigt sehr nahe Kavitätenpunkte an. Die Farbskala kann Abbildung 38 entnommen werden.



**Abbildung 38:** Colorscale für Kavitätenpunkte

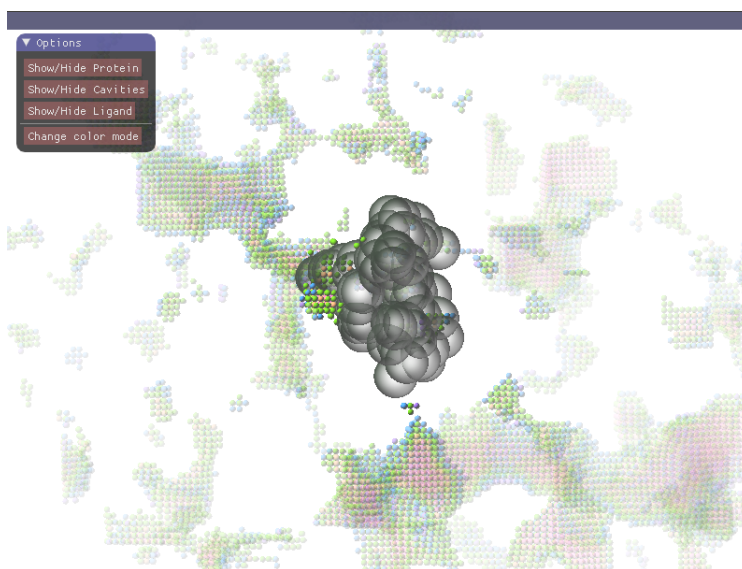
Im aktuellen Beispiel wurde ein optimaler Abstand von 3 Å mit einer Toleranz von 0.2 Å definiert, da dies in etwa dem Abstand entspricht, bei dem eine Wasserstoffbrücke zustande kommen kann (2.8-3.2 Å)[22].



**Abbildung 39:** Berechnete Kavitäten

Durch die Eingabe eines Mindestvolumens kann der Benutzer zu Beginn festlegen, welche Kavitäten er sehen möchte. Abbildung 39 zeigt verschiedene Ergebnisse, bei der einmal ein Mindestvolumen von 10 Kavitätenpunkten und einmal ein Mindestvolumen von 500 Kavitätenpunkten gewählt wurde.

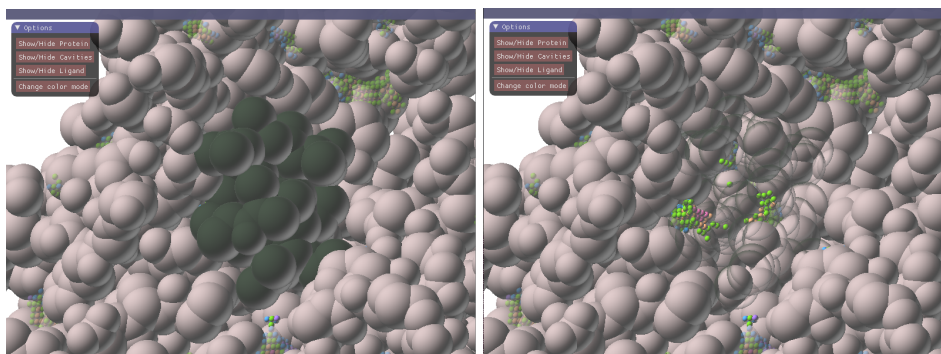
Mit Hilfe des Buttons „Add/Hide Ligand“, oder die Taste „l“ kann der Ligand hinzugefügt oder ausgeblendet werden. Beim Einblenden des Liganden werden alle Kavitätenpunkte transparent dargestellt, bis auf jene, die in sich in der Nähe des Liganden befinden. Die Transparenz nimmt dabei zu, je weiter ein Kavitätenpunkt vom Liganden entfernt ist. Dies sorgt dafür, dass der Benutzer nicht von weit entfernten und somit uninteressanten Kavitätenpunkten abgelenkt wird und immer nachvollziehen kann, an welcher Stelle er sich aktuell befindet. Ein Beispiel zeigt Abbildung 40, wobei das Protein in diesem Fall ausgeblendet ist.



**Abbildung 40:** Transparenz weit entfernter Kavitätenpunkte

Mit den WASD- und RF-Tasten kann der Ligand entlang aller Achsen bewegt und mit Hilfe der Pfeiltasten rotiert werden. Dadurch kann der Benutzer ein Docking simulieren, indem er den Liganden dem Protein annähert und Kavitäten betritt. Sobald der Ligand in eine Kavität eindringt, werden die betroffenen Kavitätenpunkte ausgeblendet. Über die +- und - - Tasten kann die Transparenz des Liganden gesteuert werden, um die Einschätzung der aktuellen Position und dahinter liegender Atome zu optimieren. Durch das vollständige Ausblenden des Liganden wird ersichtlich, ob und wie der Ligand die Bindungstasche betreten hat, da dieser durch das Betreten einer Kavität betroffene Punkte „ausschneidet“ und eine Art „Gipsabdruck“ hinterlässt. Diesen Effekt zeigt Abbildung 41.

Die an der Kante dieses „Gipsabdruckes“ übrigen Kavitätenpunkte zeigen nun durch ihre Farbe an, welchen Abstand der Ligand aktuell zum Protein besitzt und ob dieser dem durch den Benutzer definierten optimalen Abstand entspricht.



**Abbildung 41:** Ligand in Kavitäten ein- und ausgeblendet

Des Weiteren kann auch das Protein selbst ausgeblendet werden, um einen Gesamteindruck über alle Kavitäten zu bekommen, beziehungsweise eine uneingeschränkte Sicht auf die zu untersuchende Bindungsstelle zu bekommen, wie es bereits in Abbildung 40 deutlich wurde.

Weitere Informationen zu den berechneten Kavitäten werden durch die Konsole ausgegeben. Diese werden im nun folgenden Teil beschrieben.

## 6.6 Konsolenausgaben

Nach vollständiger Berechnung aller Kavitäten wird in der Konsole eine Liste über alle gefundenen Hohlräume ausgegeben. Für jede gefundene Kavität wird das entsprechende Volumen angegeben, welches der Anzahl aller zu dieser Kavität gehörenden Punkte entspricht. Dementsprechend ist diese Ausgabe auch von dem definierten Mindestvolumen abhängig und umfasst nur jene Hohlräume, die der Mindestgröße gerecht werden. Abschließend wird die Zahl aller detektierten Hohlräume ausgegeben.

Neben diesen Informationen werden zudem Debuginformationen, wie Anzahl der geladenen Atome, ausgegeben, die durch die Einbindung der ESBTL bereitgestellt werden und zu Evaluierungszwecken genutzt werden können.

Eine beispielhafte Konsolenausgabe kann Abbildung 42 entnommen werden.

```

(ESBTL-DEBUG) Lines read 5731
protein loading time = 0.136273 sec.
cavity calculation time = 82.9721 sec.
cavity counting time = 0.140463 sec.
Cavity 1: 31
Cavity 2: 32
Cavity 3: 64
Cavity 4: 341
Cavity 5: 42
Cavity 6: 413
Cavity 7: 226
Cavity 8: 347
Cavity 9: 11396
Cavity 10: 81
Cavity 11: 22
Cavity 12: 1017
Cavity 13: 18
Cavity 14: 410
Cavity 15: 38
Cavity 16: 93
Cavity 17: 308
Cavity 18: 16
Cavity 19: 16
Cavity 20: 18
Cavity 21: 475
Cavity 22: 14
Cavity 23: 13
Cavity 24: 14
Cavity 25: 36
Cavity 26: 100
Cavity 27: 65
Cavity 28: 12
Cavity 29: 10
Cavity 30: 225
Cavity 31: 62
Cavity 32: 102
Cavity 33: 2421
Cavity 34: 27
Cavity 75: 51
Cavity 76: 84
Cavity 77: 195
Cavity 78: 10
Cavity 79: 271
Cavity 80: 76
Cavity 81: 11
Cavity 82: 20
Cavity 83: 500
Cavity 84: 18
Cavity 85: 1579
Cavity 86: 12
Cavity 87: 10
Cavity 88: 53
Cavity 89: 22
Cavity 90: 43
Cavity 91: 11
Cavity 92: 227
Cavity 93: 39
Cavity 94: 32
Cavity 95: 11
Cavity 96: 12
Cavity 97: 46
Cavity 98: 53
Cavity 99: 20
Cavity 100: 40
Cavity 101: 10
Cavity 102: 15
Cavity 103: 17
Cavity 104: 193
Cavity 105: 24
Cavity 106: 15
Cavity 107: 28
Cavity 108: 45
Cavity 109: 41
Cavity 110: 73
Cavity 111: 17
111 Cavities were found.
(ESBTL-DEBUG) Lines read 68

```

**Abbildung 42:** Konsolenausgabe von CavityDetector

Das folgende Kapitel umfasst die Evaluierung der generierten Ergebnisse und bewertet zum Einen die Berechnungszeiten auf Grundlage verschiedener Datensätze, sowie die Nutzbarkeit und die Sinnhaftigkeit der Visualisierungsmethoden.

## 6.7 Evaluierung der Ergebnisse

Bei der Beurteilung von Software ist es im Allgemeinen zunächst wichtig, klar zu definieren, welche Ziele und Kriterien ein Programm zu erfüllen hat.

Im Falle von CavityDetector können sechs grundlegende Anforderungen formuliert werden:

1. Die Visualisierung soll den Benutzer dabei unterstützen, die Bindungssituation eines Protein-Ligand-Komplexes besser erkennen zu können.
2. Die Visualisierung soll den Benutzer dabei unterstützen, die Bindungssituation eines Protein-Ligand-Komplexes bewerten zu können.

3. Das Programm soll vielseitig sein und individuelle Ergebnisse für verschiedene Benutzereingaben generieren.
4. Die Berechnung der Kavitäten soll korrekt sein.
5. Die Berechnung der Kavitäten soll in einer angemessenen Zeit geschehen.
6. Das Programm soll echtzeitfähig und interaktiv sein.

Um zu untersuchen, ob und in welchem Maße die genannten Kriterien erfüllt werden, wird die Evaluation in zwei Abschnitte unterteilt. Zunächst werden Lade- und Berechnungszeiten zu verschiedenen Datensätzen gegenübergestellt und bewertet. Anschließend wird eine Bewertung durch pharmazeutische Experten wiedergegeben und ein abschließendes Fazit getroffen.

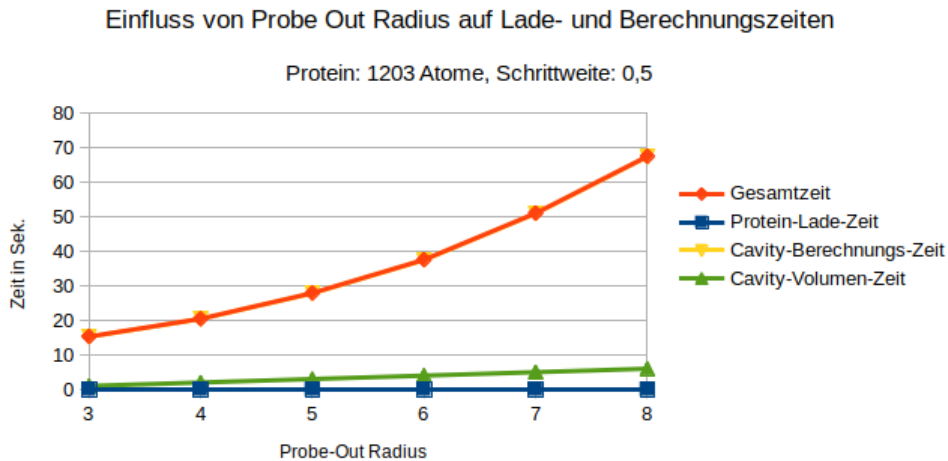
#### 6.7.1 Lade- und Berechnungszeiten

Im Umgang mit Software ist es wichtig, dass Berechnungs- oder Ladezeiten und dadurch generierte Ergebnisse in einem Verhältnis stehen, welches den Benutzer zufrieden stellt. In einem idealen Szenario würde eine Programm in minimaler Zeit ein optimales Ergebnis erzielen.

Im Falle von CavityDetector lässt sich der gesamte Initialisierungsprozess in drei Stufen aufteilen: das Laden eines Proteins, die Berechnung der Kavitäten und die Berechnung der Volumina. Die dafür aufgewendeten Zeiten wurden für verschiedene Datensätze mit einer Hilfsfunktion gemessen und untersucht. Dabei wurden zudem verschiedene Parameter, wie die Größe des Probe-Out Radius, die Anzahl der Atome eines Proteins und die Schrittweite zum Abtasten des Grids, variiert, um den Einfluss dieser auf die aufgewendeten Zeiten zu untersuchen und zu bewerten.

Alle Zeiterhebungen wurden auf einer HP Z420 Workstation mit einem Intel Xeon Prozessor, einer NVidia Quadro 2000 Grafikkarte und 32 GB Arbeitsspeicher durchgeführt.

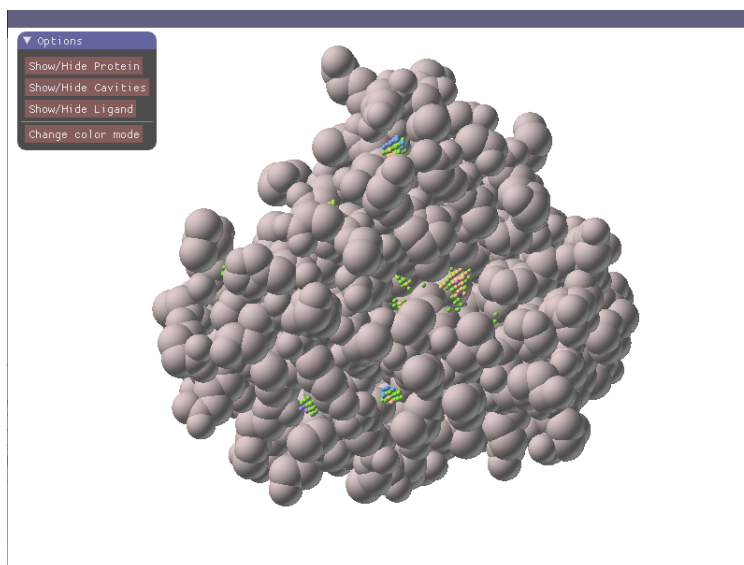
**Probe-Out Radius** Die folgende Datenerhebung wurde mit einem Myoglobin Protein (5iks.pdb) durchgeführt, welchem Liganden und umliegendes Wasser bereits entfernt wurden. Das Protein besteht demzufolge aus 1203 Atomen. Das Grid wurde mit einer Schrittweite von 0.5 abgetastet.

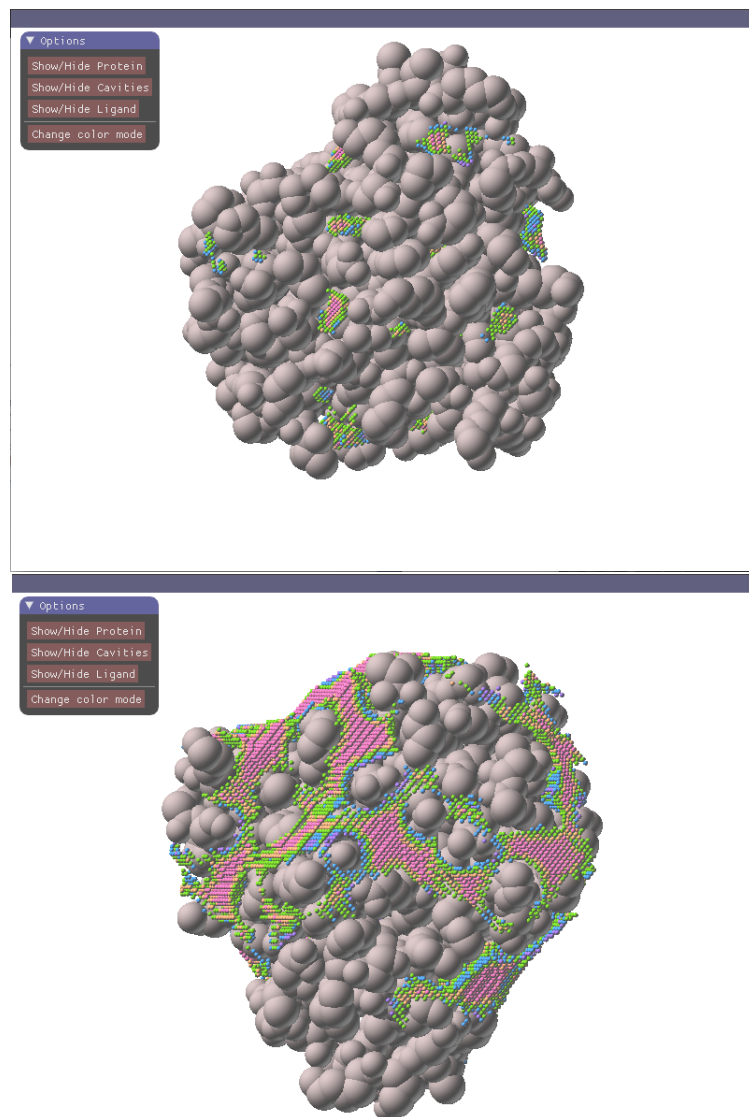


**Abbildung 43:** Einfluss von Probe-Out Radius auf Lade- und Berechnungszeiten

In dieser Untersuchung wurden sechs verschiedene Radien getestet. Betrachtet man das Diagramm der verschiedenen Zeiten fällt sofort auf, dass die Gesamtzeit im Grunde nur durch die Berechnungszeit der Kavitäten ausgemacht wird. Die Zeit, um das Protein zu laden und die Volumina zu berechnen, sind vom Radius der ProbeOut-Sphere kaum bis gar nicht beeinflusst und liegen stets unter 0.1 Sekunden.

Die Berechnungszeit der Kavitäten hingegen steigt mit Zunahme des Probe-Out Radius exponentiell an. Beispielhafte Ergebnisse für verschiedene Radien werden in Abbildung 44 gegenübergestellt.





**Abbildung 44:** Einfluss von Probe-Out Radius auf das Ergebnis

Die minimale Berechnungszeit bei einem Radius von 3 Å beträgt 15,2597 Sekunden, während die maximale Gesamtzeit bei einem Radius von 8 Å bei 67,455 Sekunden liegt. Dies ist dadurch zu begründen, dass die Berechnung durch dreifach verschachtelte for-Schleifen realisiert wird. Erfahrungsgemäß liefert der Algorithmus bei einem Probe-Out Radius von 4 Å meistens die „besten“ und anschaulichsten Ergebnisse. Hier beträgt die Gesamtzeit etwa 20,468753 Sekunden.

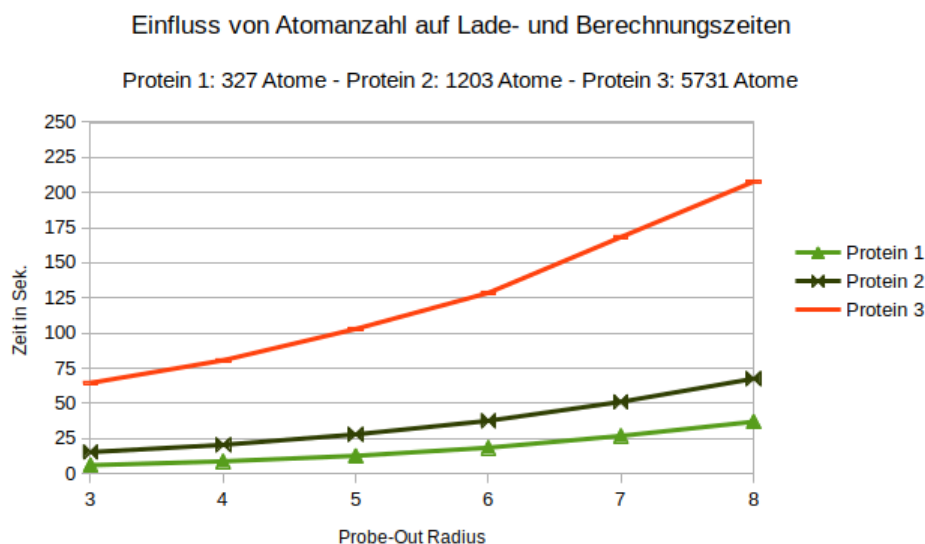
Unter Berücksichtigung der Tatsache, dass diese Berechnung nur einmal durchgeführt werden muss und anschließend eine echtzeitfähige Interaktion möglich ist, kann diese Zeit, aber auch die Zeiten für die übrigen Radien,



als schnell und zufriedenstellend bewertet werden.

**Atomanzahl** Für diese Datenerhebung wurden lediglich die Gesamtzeiten drei verschiedener Proteine betrachtet, da diese, wie im vorherigen Abschnitt beschrieben, im Grunde den Berechnungszeiten der Kavitäten entsprechen und das Laden eines Proteins in allen Fällen unter 0,15 Sekunden passierte.

Bei den untersuchten Proteindatensätzen handelt es sich um ein sehr kleines einkettiges Protein (1crn.pdb) mit 327 Atomen, um das Myoglobin Protein mit 1203 Atomen, welches auch bei der Untersuchung des Einflusses des ProbeOut-Radius genutzt wurde, und einem  $G\alpha_q$  Protein, aus welchem ebenfalls die Liganden extrahiert wurden, mit 5731 Atomen.



**Abbildung 45:** Einfluss von Probe-Out Radius auf Lade- und Berechnungszeiten

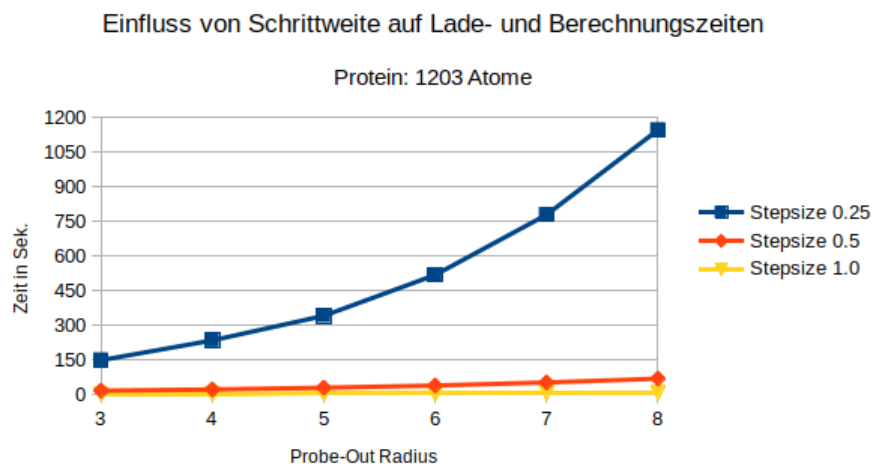
Abbildung 45 zeigt deutlich, dass sich die Anzahl der Atome direkt auf die Berechnungszeit auswirkt. Bei einer fünffachen Anzahl an Atomen muss der Algorithmus fünf mal längere Schleifen durchlaufen. Betrachtet man wieder die Werte für einen Probe-Out Radius von 4, benötigt die Initialisierung des größten Atoms schon 80,58 Sekunden, also 1 Minute und 20 Sekunden und ist damit schon deutlich langsamer als bei kleineren Proteinen.

Da es im Bereich der Forschung in der Regel darum geht, langfristige Erfolge zu erzielen und bestimmte Sachverhalte gründlich und in Ruhe zu untersuchen, ist diese Berechnungszeit sicher noch akzeptabel, was auch durch Gespräche mit Pharmazeuten bestätigt wurde, jedoch keinesfalls als effizient zu bezeichnen und bietet somit großes Optimierungspotential. Durch

Multithreading und Multiprocessing ließe sich der zeitliche Rechenaufwand sehr gut verkürzen, sodass auch noch größere Proteine und ihre Kavitäten in einer angemessenen Zeit geladen und berechnet werden könnten.

**Schrittweite** Die letzte Zeitmessung wurde ebenfalls an dem Myoglobin Protein 5iks.pdb durchgeführt, wobei verschiedene Schrittweiten zur Abtastung des Grids definiert wurden.

Wie fein das Grid abgetastet wird hat nicht nur unmittelbar Einfluss auf die aufgewendete Zeit, sondern ebenso auf das visuelle Ergebnis.



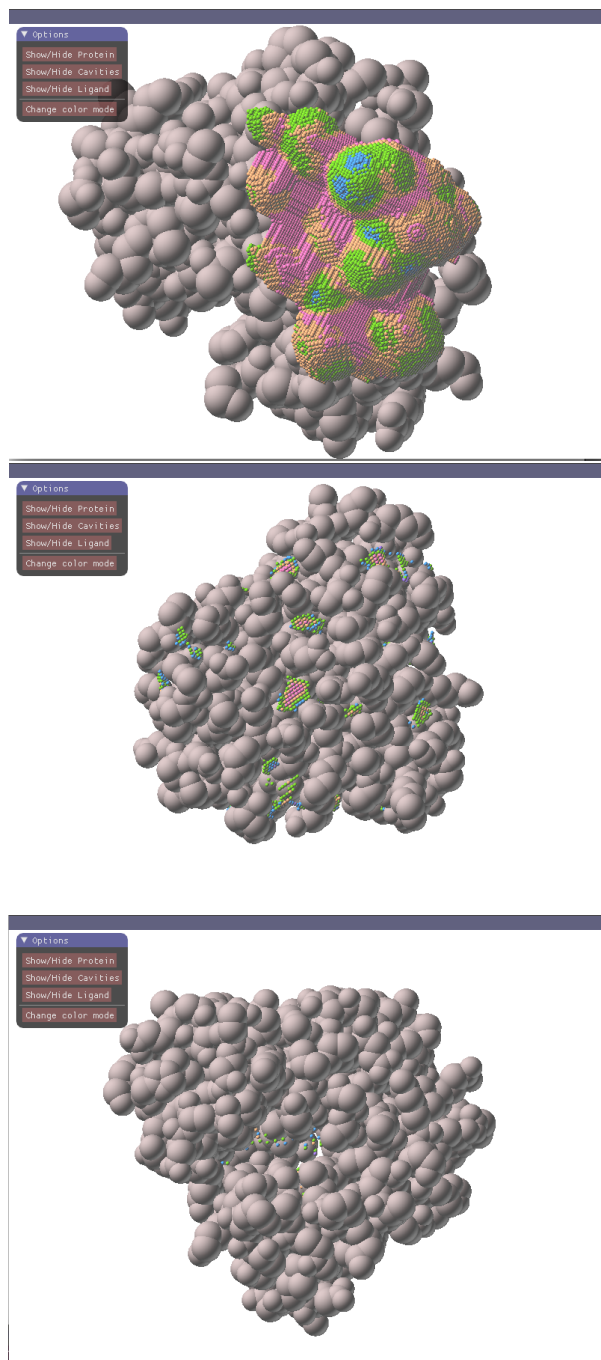
**Abbildung 46:** Einfluss von Schrittweite auf Lade- und Berechnungszeiten

Abbildung 46 zeigt die Gesamtzeiten für Schrittweiten von 0.25, 0.5 und 1.0. Während die Berechnung bei einfacher Schrittweite maximal 6 Sekunden benötigt, dauert die Initialisierung bei einer Schrittweite von 0.25 und einem Probe-Out Radius von 4 schon drei mal so lang wie die Berechnung bei dem fünf mal größeren  $G\alpha_q$ -Protein bei halber Schrittweite.

Die Schrittweite hatte demzufolge einen viel größeren Einfluss auf die Kalkulationszeiten.

Wie bereits erwähnt ist es wichtig, die Berechnungszeiten stets in Verhältnis zu den visuellen Ergebnissen zu setzen. In diesem Fall konnte zum Einen durch eine grobere Abtastung kein ausreichend gutes Ergebnis und zum Anderen durch eine feinere Abtastung kein sinnvolles Ergebnis erzielt werden, wie in Abbildung 47 zu sehen ist. Bei einer Schrittweite von 0.25 werden keine korrekten Kavitäten gefunden, was unter Umständen an Rundungsfehlern liegt. Dieses Ergebnis ist dementsprechend nicht brauchbar. Ebenso ist das Resultat bei einer einfachen Schrittweite nicht hilfreich, da

diese Schrittweite zu groß ist, um Kavitäten zu finden. Die einzige sinnvolle Schrittweite ist also 0.5.



**Abbildung 47:** Einfluss von Schrittweite auf Ergebnis  
(Oben: Schrittweite 0.25, Mitte: Schrittweite 0.5, Unten: Schrittweite 1.0)

Daher ist es nicht notwendig, die langen Berechnungszeiten für kleinere

Schrittweiten zu berücksichtigen oder besonders zu gewichten, da die besten Resultate mit einer halben Schrittweite generiert werden können und eine tolerable Zeit benötigen.

Zusammenfassend kann gesagt werden, dass CavityDetector keinesfalls auf einem schnellen Algorithmus basiert. Mit größeren Datensätzen und größeren Probe-In Radien steigt die gesamte Berechnungszeit, aufgrund der vielen Gridoperationen, exponentiell an.

CavityDetector erlaubt es jedoch, wichtige Parameter, die die Berechnungszeit beeinflussen, selbst zu definieren, sodass der Benutzer in der Lage ist, diese so zu wählen, dass er ein zufriedenstellendes Ergebnis in einer angemessenen Zeit erhält. Darüber hinaus bietet der Algorithmus viel Potential zur Optimierung, sodass der Zeitaufwand noch reduziert werden kann und auch gut für sehr große Proteine eingesetzt werden kann.

### 6.7.2 Bewertung durch pharmazeutische Experten

Eines der größten Ziele von CavityDetector ist es, Pharmazeuten in ihrer Forschung zu unterstützen und einen Mehrwert gegenüber bereits existierenden Methoden zu besitzen. Um dies zu überprüfen wurde die Kooperation zwischen der Universität Koblenz-Landau und pharmazeutischen Experten aus Bonn dazu genutzt, eine fachkundige Meinung über den Nutzen und die Tauglichkeit des Tools zu erhalten. Dazu stellten die Pharmazeuten ausgewählte Datensätze bereit, die in ihrer aktuellen Forschung eine große Rolle spielen. Darauf aufbauend wurde ein Fragebogen entwickelt, welcher durch die Experten beantwortet werden sollte.

**Datensätze** Die Pharmazeuten baten darum, die Bindungssituation zwischen einem  $G\alpha_q$  Protein und dem Ligand YM-254890 zu untersuchen. G-Proteine sind sehr wichtige Moleküle, die für vielerlei Signaltransduktionen verantwortlich sind. Damit G-Proteine diese Funktionen erfüllen können, müssen sie durch Rezeptoren aktiviert werden. Diese Rezeptoren müssen ebenfalls in einem aktiven Zustand vorliegen, was erst durch die Bindung eines Liganden und die dadurch verursachte Konformationsänderung realisiert wird. Nach einer solchen Aktivierung kann der Rezeptor als GTP-Austauschfaktor dienen und so die Funktionen des G-Proteins aktivieren.

Ziel aktueller Forschungsprojekte ist es, Liganden so zu synthetisieren, dass sie in der Lage sind, die sogenannten G-Protein-gekoppelten Rezeptoren und somit auch die G-Proteine zu aktivieren, um daraus möglicherweise neue Medikamente entwickeln zu können.

Ein Gespräch mit den Pharmazeuten ergab, dass besonders die Bindungstasche für den Liganden YM-254890 interessant sei, da dieser aufgrund seiner Stabilität und seiner chemischen Eigenschaften als „perfekter“ Ligand

bezeichnet werden könne. Aus diesem Grund sei es das Ziel, diese besondere Bindung zu untersuchen und herauszufinden, wie der Ligand in seiner Bindungstasche liegt und ob es möglich ist, einen Stoff herzustellen, der genau so an den Rezeptor binden kann.

**Fragebogen** Um die Korrektheit und den Nutzungsfaktor von CavityDetector zu überprüfen, wurden das Protein und der Ligand geladen und mit Hilfe der beschriebenen Möglichkeiten visualisiert. Dazu wurde ein Fragebogen entwickelt, der sich aus drei Teilen zusammensetzte - einer Einleitung und einem zweigeteilten Hauptteil.

Zunächst wurde eine Übersicht über das Tool CavityDetector und seine Funktionen, sowie Hinweise zum Aufbau des Fragebogens gegeben. In der ersten Hälfte des Hauptteils wurde der Ligand YM-254890 in eine beliebige Bindungstasche gelegt, die nicht seiner natürlichen entsprach. In der zweiten Hälfte wiederum wurde der Ligand in seiner korrekten, natürlichen Bindungstasche dargestellt.

Diese Bindungssituationen wurden aus vier verschiedenen Blickwinkeln dargestellt. Aufgabe der Pharmazeuten war es nun, zu bewerten wie gut oder schlecht sie diese Protein-Ligand-Komplexe räumlich einschätzen können und ob der Ligand gut in seiner Bindungstasche liegt.

Dazu wurden die Bindungen jeweils auf vier verschiedene Weisen dargestellt.

1. Darstellung von Protein und Ligand, beide opaque, ohne Zusatzinformation
2. Darstellung von Protein und Ligand, Protein opaque und Ligand transparent, keine Zusatzinformation
3. Darstellung von Protein und Ligand, Protein opaque und Ligand transparent, Kavitätenpunkte
4. Darstellung von Ligand, Protein ausgeblendet und Ligand transparent, Kavitätenpunkte

Die Abbildungen 48 und 49 zeigen beispielhaft beide Bindungssituationen aus einem Blickwinkel auf die vier verschiedenen Weisen.

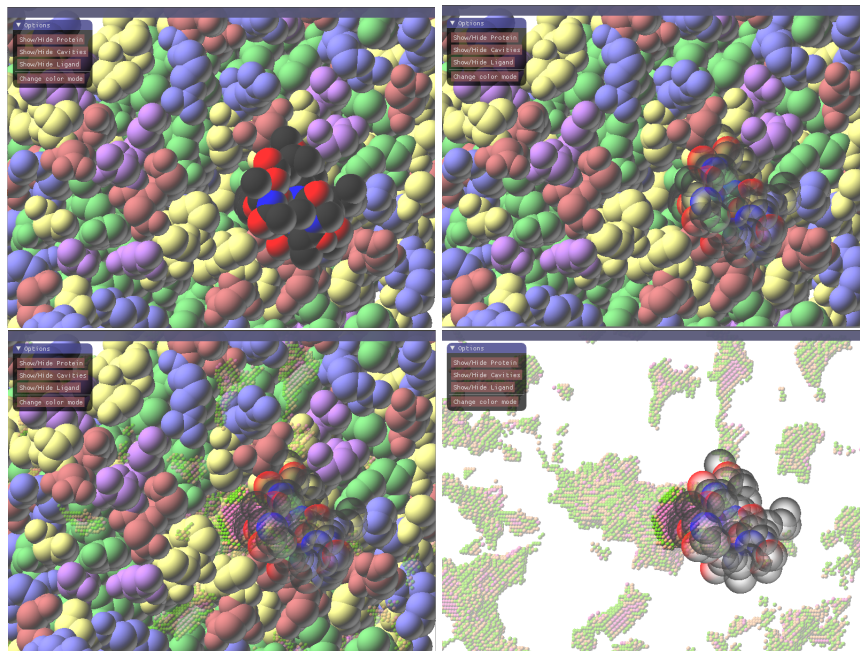


Abbildung 48: Bindungssituation 1 - Ligand in einer falschen Bindungstasche

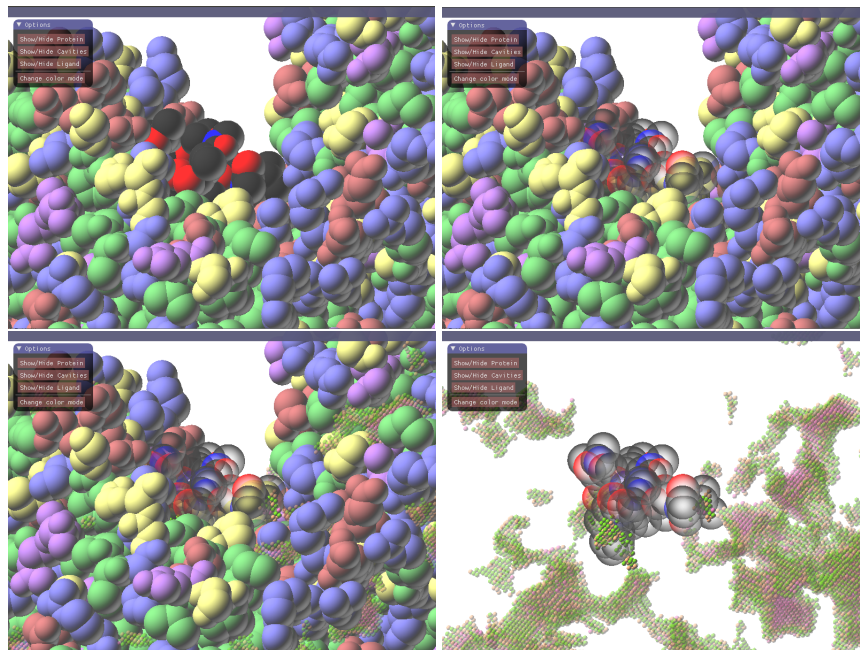


Abbildung 49: Bindungssituation 2 - Ligand in seiner korrekten Bindungstasche

Zu jeder Darstellungsvariante wurden die gleichen vier Fragen gestellt.

1. Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

- Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt?
- Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?
- Besitzt der der Ligand einen optimalen Abstand zum Protein?

Diese mussten auf einer Skala von „sehr schlecht“ bis „sehr gut“, beziehungsweise von „Nein“, über „nicht ersichtlich“, bis „Ja“ beantwortet werden.

CavityDetector - Evaluation  
Vanessa Schüller

CavityDetector ist ein Tool, welches im Rahmen einer Bachelorarbeit entwickelt wurde, um Hohlräume eines Proteins zu visualisieren und zu hochrechnen, um Docking-Prozesse zwischen diesem und einem Liganden zu visualisieren und besser bewerten zu können.

Dann werden die Kavitäten des Proteins mit Hilfe eines geometrischen Algorithmus berechnet und durch Punkte visualisiert, die durch ihre Farbgebung Aufschluss über den Abstand zum Protein liefern. Für verschiedene Bindungsstellen gibt es jeweilige optimale Abstände, ab welchen eine solche Bindung zustande kommen kann. Diese werden durch den Benutzer definiert und durch grüne Punkte symbolisiert. Nicht-optimale Abstände wiederum werden durch rote Punkte dargestellt. Übergänge werden durch orangefarbene Punkte visualisiert.

Durch verschiedene Tastaturbefehle kann ein Ligand hineingefügt werden und beliebig verschoben werden. Betritt dieser eine Kavität, so wird der betroffene Bereich ausgeschaltet.

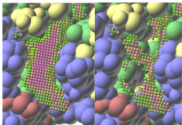


Abbildung 1: Kavität vor und nach Betreten eines Liganden

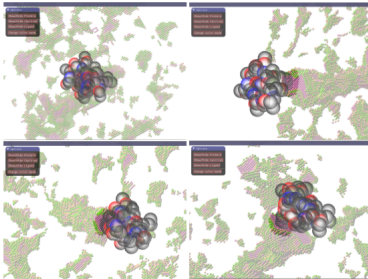
Hat ein Ligand, wie in der Abbildung, um eine Position erreicht, in welcher er überwiegend an grüne Kavitätspunkte stößt, so kann davon ausgegangen werden, dass ein Docking erfolgreich sein könnte.

Im Folgenden wird Ihnen eine Bindungsoptionen aus verschiedenen Blickwinkeln auf verschiedene Weisen gezeigt. Sie werden gebeten, diese zu bewerten und eine Aussage darüber zu treffen, ob die dargestellte Konformation eine Bindung ermöglichen könnte oder nicht. Als optimal wurde in diesem Beispiel ein Abstand von 3,0 Å mit einer Toleranz von 0,2 Å angenommen, da dies ungefähr dem Abstand zwischen Atomen entspricht, die eine Wasserstoffbrücke bilden (2,8 - 3,2 Å)<sup>1</sup>.

Durch den definierten Abstand ergibt sich folgendes Bild einer Kavität:  
 Grüne Punkte symbolisieren optimale Abstände  
 Orangefarbene Punkte symbolisieren Abstände nahe dem optimalen Abstand  
 Rote Punkte symbolisieren nicht-optimale Abstände

<sup>1</sup>Gebhard Kliche - Wirkstoffdesign: Entwurf und Wirkung von Arzneistoffen

1.4 Protein(ausgeblendet) und Ligand, Kavitätenpunkte



1.4.1 Wie gut können Sie die Bindungsoptionen zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

1.4.2 Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

1.4.3 Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

1.4.4 Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja
<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

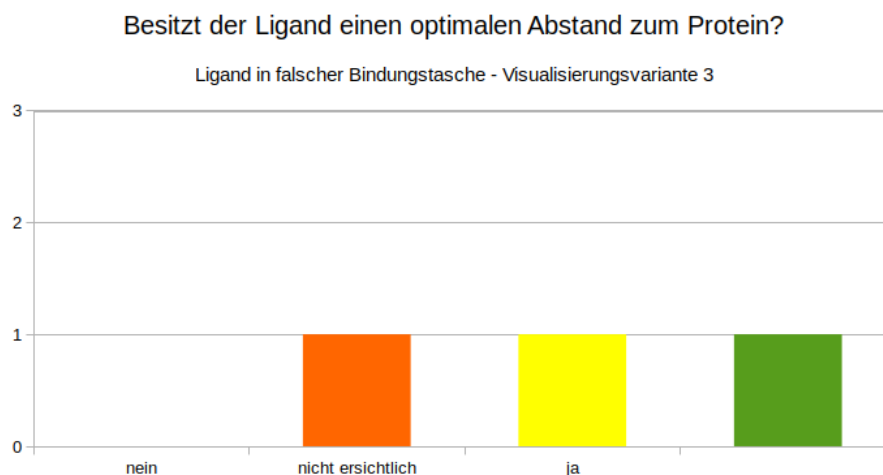
1.4.5 Persönliche Anmerkungen:

**Abbildung 50:** Aufbau des Fragebogens

Abbildung 50 zeigt den Aufbau des Fragebogens. Der vollständige Fragebogen kann im Anhang eingesehen werden. Auf die Antworten und das Ergebnis der Umfrage wird im Folgenden eingegangen.

**Auswertung** Die Auswertung des Fragebogens ergab im Gesamten, dass die Kavitätenberechnung und -visualisierung durch CavityDetector die räumliche Wahrnehmung von Proteinen und Liganden gut unterstützen und nur kleine Änderungen an Farbgebungen vorgenommen werden sollten. Zum Zeitpunkt der Befragung besaß das Protein noch keine einheitliche Farbe, sondern jedes Atom war entsprechend seiner Aminosäurezugehörigkeit gefärbt. Dies ist aufgrund der formulierten Kritik in der aktuellen Version nur noch über einen Button möglich. Die Pharmazeuten wiesen

darauf hin, dass die Darstellung der Bindungssituationen unter Zuhilfenahme der Kavitätenpunkte zu voll und zu bunt sei und dass dies die Analyse des Bindungsabstandes erschweren würde. Dies spiegelt sich auch in den Antworten auf die Frage, ob der Ligand in Bindungssituation 1 einen optimalen Abstand zum Protein hat, wieder, die in Abbildung 51 dargestellt sind.



**Abbildung 51:** Bewertung des Abstandes von Ligand zu Protein

Hier gab es trotz der Darstellung der Hohlräume kein klares Ergebnis, da jede Möglichkeit die gleiche Stimmenanzahl hatte. Ein möglicher Grund für dieses Ergebnis ist die fehlende Erläuterung dazu, wie die Kavitätenpunkte interpretiert werden sollten. Eine weitere Ursache ist die bereits erwähnte Farbgebung sowie die Anzeige von weiter weg liegenden und uninteressanten Kavitäten, die auch durch freie Kommentarooptionen angesprochen wurden.

*Hintergrund blasser, Interaktion hervorheben [...]*

*monocolored for protein/ ligand*

*Andere Kavitäten ausblenden*

Diese Kritikpunkte sind in der aktuellen Version von CavityDetector optimiert und entsprechend umgesetzt.

Besonders der Vergleich der ersten, einfachen Darstellungsvariante, bei welcher lediglich Protein und Ligand ohne weitere Informationen dargestellt wurden, und der vierten Variante, bei welcher das Protein ausgeblendet war, zeigt jedoch deutlich, dass die Visualisierung der Kavitätenpunkte sehr hilfreich für die Einschätzung der Bindungssituation sein kann. Abbildungen 52 und 53 zeigen die entsprechenden Bilder.



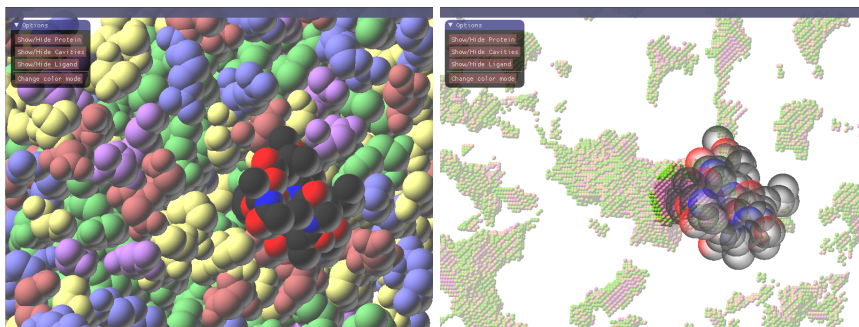


Abbildung 52: Bindungssituation 1

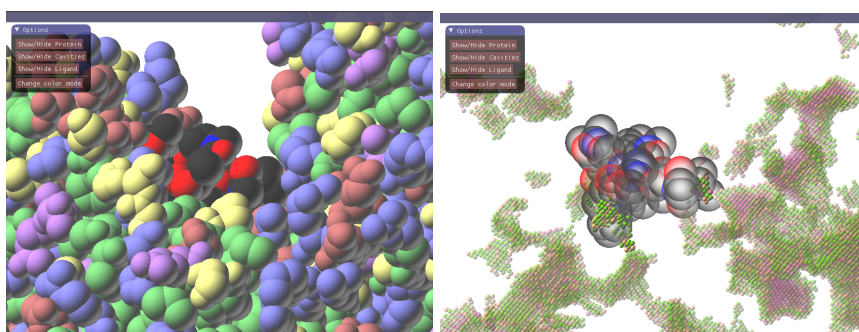
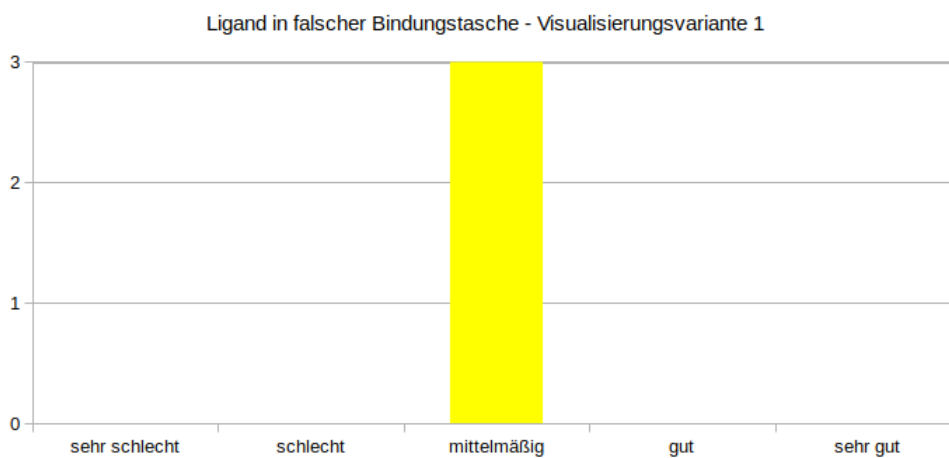


Abbildung 53: Bindungssituation 2

Auf die Frage, wie gut der Abstand zwischen Protein und Ligand eingeschätzt werden kann, war die Antwort bei beiden Bindungssituationen für die einfache Darstellung nur durchschnittlich mittelmäßig, wie in Abbildung 54 zu sehen ist.

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?



Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

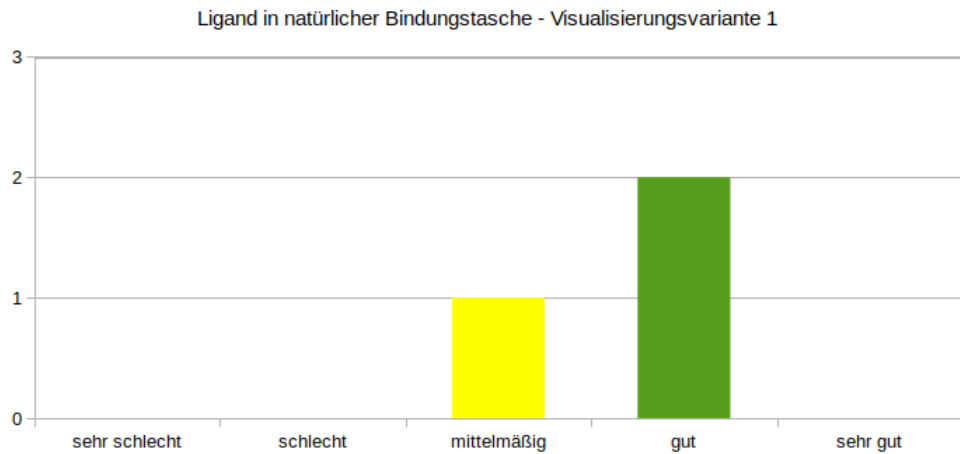
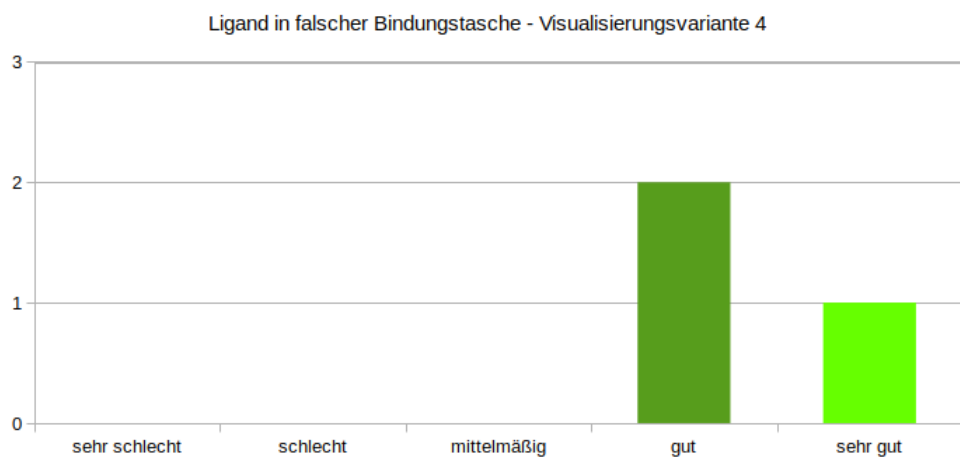


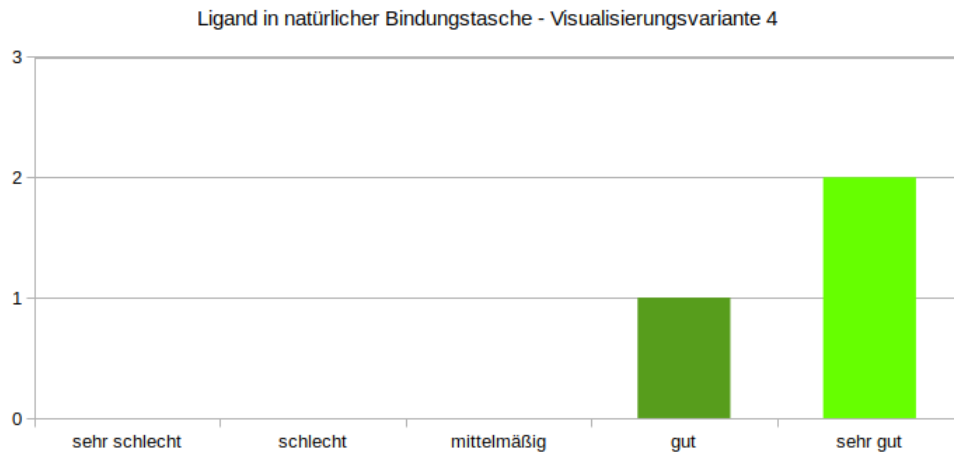
Abbildung 54: Bewertung der räumlichen Einschätzung ohne Kavitätenvisualisierung

Durch die Hinzunahme der Kavitätenpunkte und das Ausblenden des Proteins war eine bessere Einschätzung des Protein-Ligand-Komplexes möglich und die Frage wurde mit „gut“ bis „sehr gut“ beantwortet, wie es Abbildung 55 zeigt.

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt?



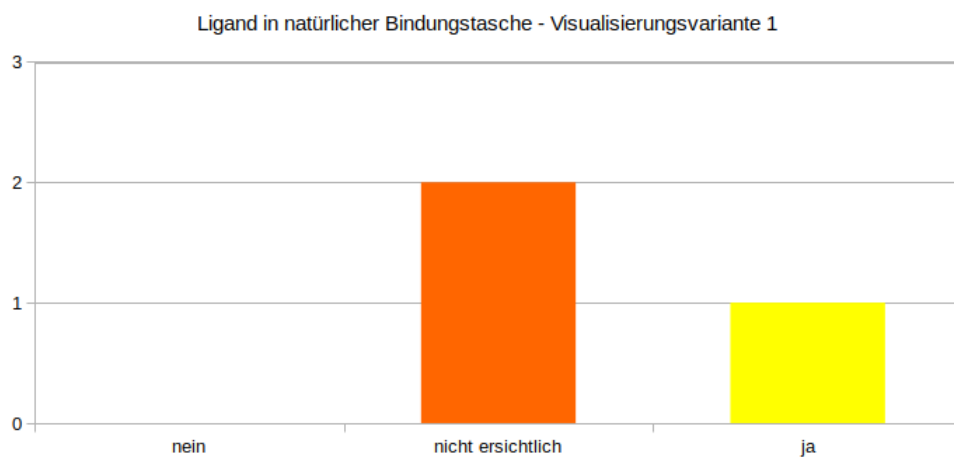
### Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt?



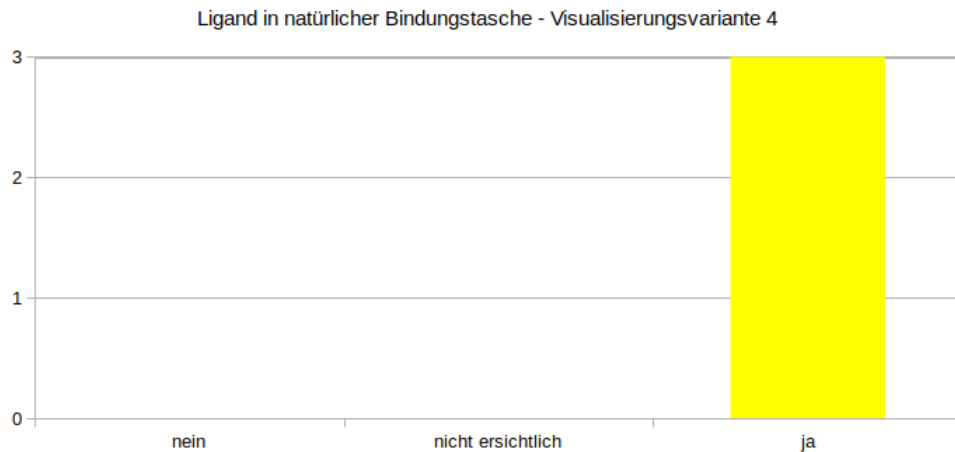
**Abbildung 55:** Bewertung der räumlichen Einschätzung mit Kavitätenvisualisierung

Im Falle der zweiten Bindungssituation wurde die Frage über den optimalen Abstand zudem einheitlich mit „Ja“ beantwortet, während auf die gleiche Frage bei anderen Darstellungsmethoden geantwortet wurde, dass der Abstand nicht ersichtlich sei. Dieses Ergebnis zeigt Abbildung 56 .

### Besitzt der Ligand einen optimalen Abstand zum Protein?



## Besitzt der Ligand einen optimalen Abstand zum Protein?



**Abbildung 56:** Bewertung des Abstandes von Ligand zu Protein

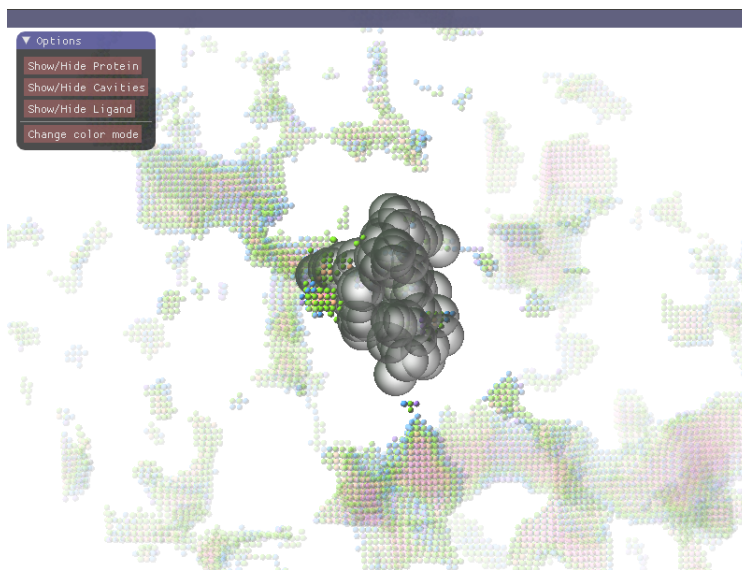
Darüber hinaus zeigt dies auch, dass die Visualisierung der Kavitätenspunkte durchaus korrekt ist, da der Ligand in dieser Situation in seiner natürlichen Bindungstasche liegt und dies auch durch die Hohlräume und ihre grüne Farbe so angezeigt wird. Abbildung reffig:abb57 zeigt dieses Ergebnis noch einmal nach Anpassung der Farben.

In einer abschließenden Frage wurde die Visualisierung der Kavitäten auf einer Skala von „nicht hilfreich“ über „wenig hilfreich“ und „hilfreich“ bis „sehr hilfreich“ einstimmig als hilfreich für das Einschätzen der Bindungssituationen bewertet.

### 6.7.3 Fazit

Die technische Evaluation sowie die Bewertung durch die pharmazeutischen Experten haben ergeben, dass die Idee hinter CavityDetector durchaus Potential hat, die Forschung von Protein-Ligand-Interaktionen zu unterstützen. Neben einer besseren Einschätzung der Oberfläche eines Proteins und der Position eines Liganden, erhält der Benutzer Informationen über noch vorhandene Hohlräume und somit über Abstände zwischen Protein und Ligand und kann daraus weitere Erkenntnisse ableiten.

Die Berechnung der nötigen Daten kann einige Zeit dauern, muss jedoch nur einmal durchgeführt werden. Darüber hinaus ist eine Interaktion in Echtzeit möglich, sodass ein Docking selbst simuliert werden kann. Des Weiteren bietet der Algorithmus Potential zur Parallelisierung oder Multithreading, sodass auch große Proteine in angemessener Zeit geladen und berechnet werden könnten.



**Abbildung 57:** Ligand in korrekter Bindungstasche

Wichtig im Umgang mit CavityDetector ist zu wissen, wie die entstehenden Bilder korrekt interpretiert werden, um Missverständnisse zu vermeiden. Des Weiteren liefern die Informationen zu Protein-Ligand-Abständen nicht zwangsläufig eine Aussage darüber, ob ein Docking erfolgreich ist oder nicht, sondern liefern lediglich eine hilfreiche Information zur Einschätzung der Position des Liganden in seiner Bindungstasche und damit eine Tendenz. Dies zeigt, dass CavityDetector mittels bestehender Kavitäten-Algorithmen, Protein-Ligand-Komplexe räumlich verständlicher macht und Pharmazeuten dadurch in ihrer Forschung gut unterstützen könnte. Dennoch stehen weiterhin Fragen offen, die die pharmazeutischen Forscher versuchen zu beantworten. Dazu erhoffen sie sich weiterhin Unterstützung der Computergrafik, indem sie diese Problemstellungen durch Visualisierungen lösbar macht. Welche Probleme es aktuell zu lösen gilt und inwiefern CavityDetector eine Basis für weitere Entwicklungen bietet, beschreibt der folgende und letzte Abschnitt.

## 7 Ausblick

CavityDetector ermöglicht es einen Protein-Ligand-Komplex zu einem Zeitpunkt zu untersuchen und hilfreiche Informationen über Oberflächenstrukturen, sowie Abstände zwischen Protein und Ligand zu erhalten, um so Hinweise darauf zu bekommen, ob ein Ligand gut in seiner Bindungstasche liegt.

Proteine sind jedoch dynamische Gebilde, die ihre Konformation abhängig von äußeren Einflüssen ändern können. Nähert sich beispielsweise ein

Ligand an ein Protein, so wirken Kräfte, die eine Konformationsänderung des Proteins auslösen können.

Genau dieses Phänomen gilt es in der pharmazeutischen Forschung zu untersuchen, da erst so herausgefunden werden kann, wieso beispielsweise Liganden, die in eine Bindungstasche zu passen schienen, nicht mehr andocken können, sobald sie sich dem Protein nähern.

Um dieses Verhalten veranschaulichen zu können, müsste es möglich sein, die Berechnung und Visualisierung von CavityDetector für eine Simulation eines dynamischen Protein-Ligand-Dockings anzuwenden. Ein Ansatz dazu wäre, die Berechnung für jeden Zeitschritt einer Simulation durchzuführen und die Ergebnisse schließlich miteinander zu vergleichen. Dieses Prinzip bringt jedoch einen hohen Rechenaufwand mit sich, der durch Parallelisierung jedoch etwas minimiert werden könnte. Des Weiteren wäre es möglich nur bestimmte Zeitschritte zu untersuchen und Zwischenergebnisse zu schätzen oder zu interpolieren.

CavityDetector liefert, neben Informationen zur Struktur eines Proteins, Informationen zu Abständen zwischen Atomen. Diese allein haben jedoch nicht ausreichend Aussagekraft darüber, ob ein Ligand an ein Protein andocken kann oder nicht. Was zählt, sind die Atome, die sich einander annähern und die vorherrschenden Wechselwirkungen, die dafür sorgen, dass bestimmte Reste eines Liganden stärker angezogen werden als andere. Um diese zu visualisieren, wäre es möglich Kavitätenpunkte durch Glyphen auszutauschen und Informationen über die Anziehungskräfte in die Berechnung mit einfließen zu lassen, so dass eine Richtung der Kräfte zwischen Protein und Ligand erkennbar wird.

CavityDetector bildet somit eine gute Basis für ein besseres räumliches Verständnis während einer statischen Docking-Simulation und bietet noch viel Potential zur Erweiterung, um dynamische Wechselwirkungen nachvollziehen zu können und so bestehende Hürden der pharmazeutischen Forschung lösen zu können.

## CavityDetector - Evaluation

Vanessa Schüller

CavityDetector ist ein Tool, welches im Rahmen einer Bachelorarbeit entwickelt wurde, um Hohlräume eines Proteins zu visualisieren und zu beobachten, um Docking-Prozesse zwischen diesem und einem Liganden zu visualisieren und besser bewerten zu können.

Dazu werden die Kavitäten des Proteins mit Hilfe eines geometrischen Algorithmus berechnet und durch Punkte visualisiert, die durch ihre Farbgebung Aufschluss über den Abstand zum Protein liefern. Für verschiedene Bindungstypen gibt es jeweilige optimale Abstände, ab welchen eine solche Bindung zustande kommen kann. Diese werden durch den Benutzer definiert und durch grüne Punkte symbolisiert. Nicht-optimale Abstände wiederum werden durch rote Punkte dargestellt. Übergänge werden durch orangene Punkte visualisiert.

Durch verschiedene Tastaturbefehle kann ein Ligand hinzugefügt werden und beliebig verschoben werden. Betritt dieser eine Kavität, so wird der betretene Bereich ausgeschnitten.

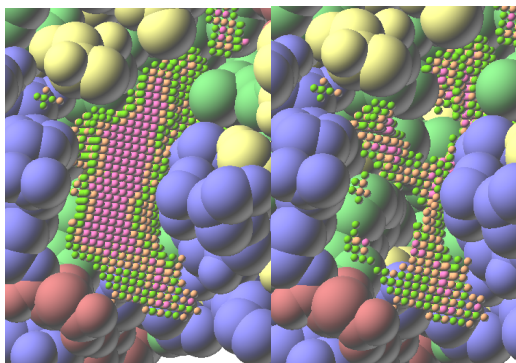


Abbildung 1: Kavität vor und nach Betreten eines Liganden

Hat ein Ligand, wie in der Abbildung, nun eine Position erreicht, in welcher er überwiegend an grüne Kavitätspunkte angrenzt, so kann davon ausgegangen werden, dass ein Docking erfolgreich sein könnte.

Im Folgenden wird Ihnen eine Bindungssituation aus verschiedenen Blickwinkeln auf verschiedene Weisen gezeigt. Sie werden gebeten, diese zu bewerten und eine Aussage darüber zu treffen, ob die dargestellte Konformation eine Bindung ermöglichen könnte oder nicht. Als optimal wurde in diesem Beispiel ein Abstand von 3.0 Å mit einer Toleranz von 0.2 Å angenommen, da dies ungefähr dem Abstand zwischen Atomen entspricht, die eine Wasserstoffbrücke bilden (2.8 - 3.2 Å)<sup>1</sup>.

Durch den definierten Abstand ergibt sich folgendes Bild einer Kavität:

Grüne Punkte symbolisieren optimale Abstände

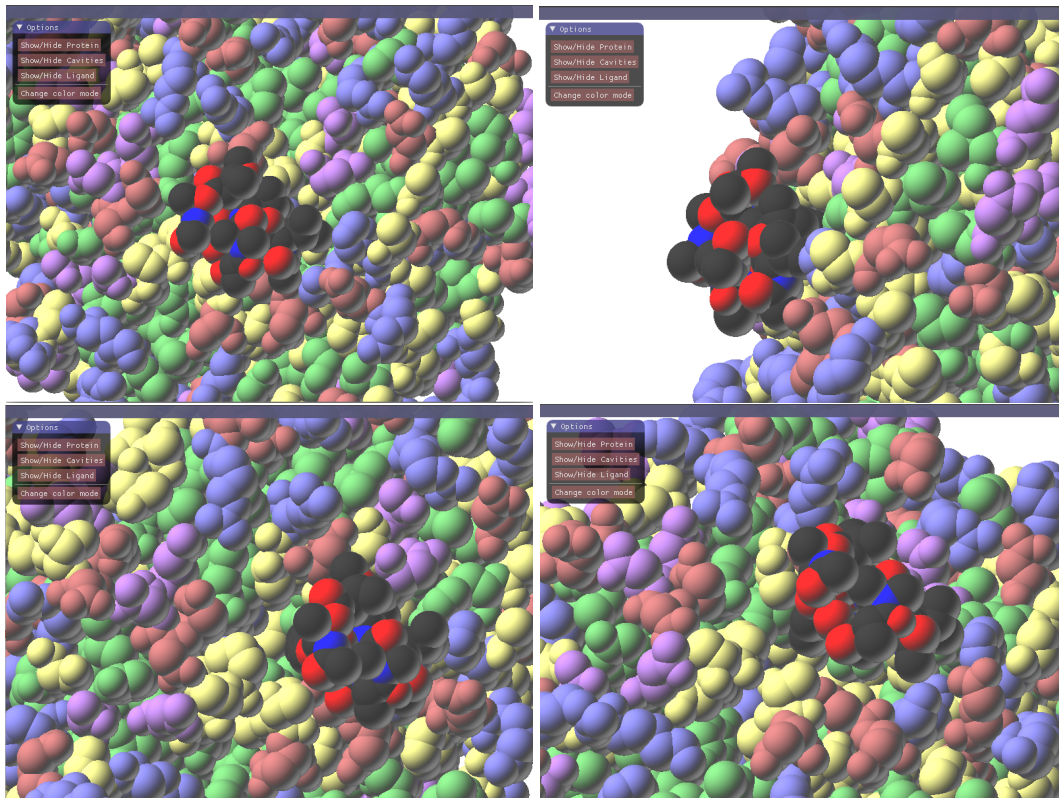
Orangene Punkte symbolisieren Abstände nahe dem optimalen Abstand

Rote Punkte symbolisieren nicht-optimale Abstände

<sup>1</sup>Gerhard Klebe - Wirkstoffdesign: Entwurf und Wirkung von Arzneistoffen

## 1

## 1.1 Protein und Ligand, einfach



## 1.1.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

## 1.1.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

## 1.1.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

## 1.1.4

Besitzt der Ligand einen optimalen Abstand zum Protein?

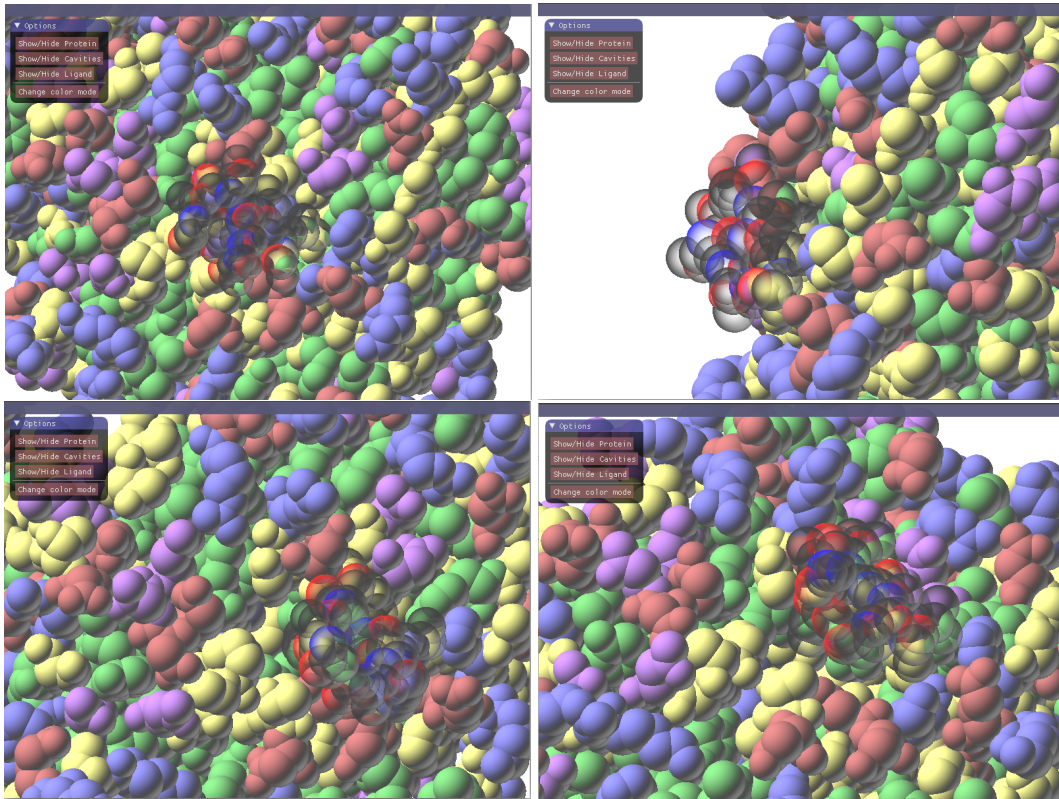
Nein	nicht ersichtlich	Ja

## 1.1.5

Persönliche Anmerkungen:



## 1.2 Protein und Ligand, Transparenz



### 1.2.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 1.2.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 1.2.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 1.2.4

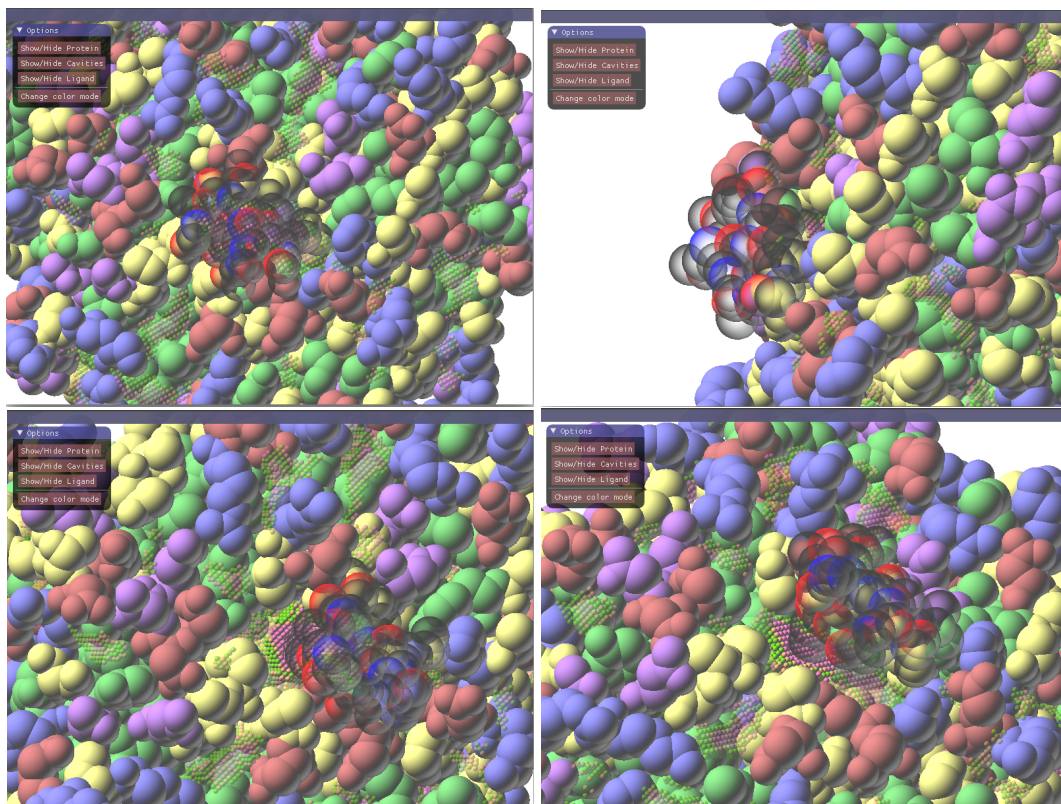
Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja

### 1.2.5

Persönliche Anmerkungen:

### 1.3 Protein und Ligand, Kavitätenpunkte



#### 1.3.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

#### 1.3.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

#### 1.3.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

#### 1.3.4

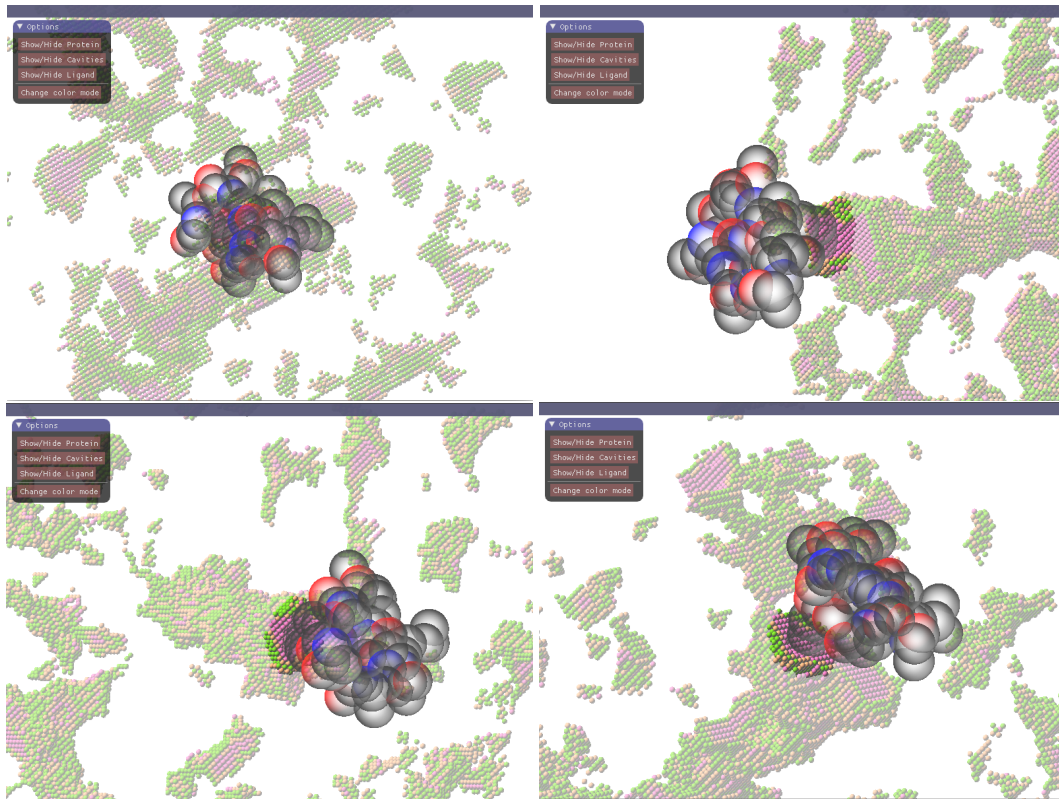
Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja

#### 1.3.5

Persönliche Anmerkungen:

## 1.4 Protein(ausgeblendet) und Ligand, Kavitätenpunkte



### 1.4.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 1.4.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 1.4.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 1.4.4

Besitzt der Ligand einen optimalen Abstand zum Protein?

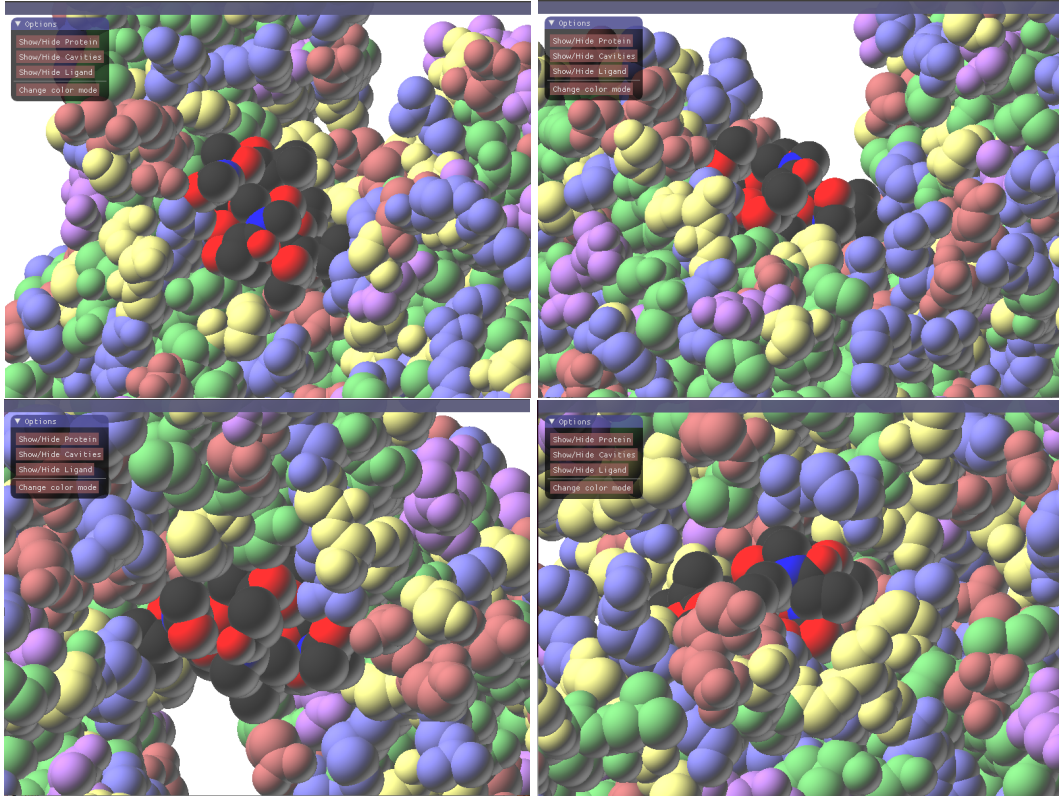
Nein	nicht ersichtlich	Ja

### 1.4.5

Persönliche Anmerkungen:

## 2

## 2.1 Protein und Ligand, einfach



## 2.1.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

## 2.1.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

## 2.1.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

## 2.1.4

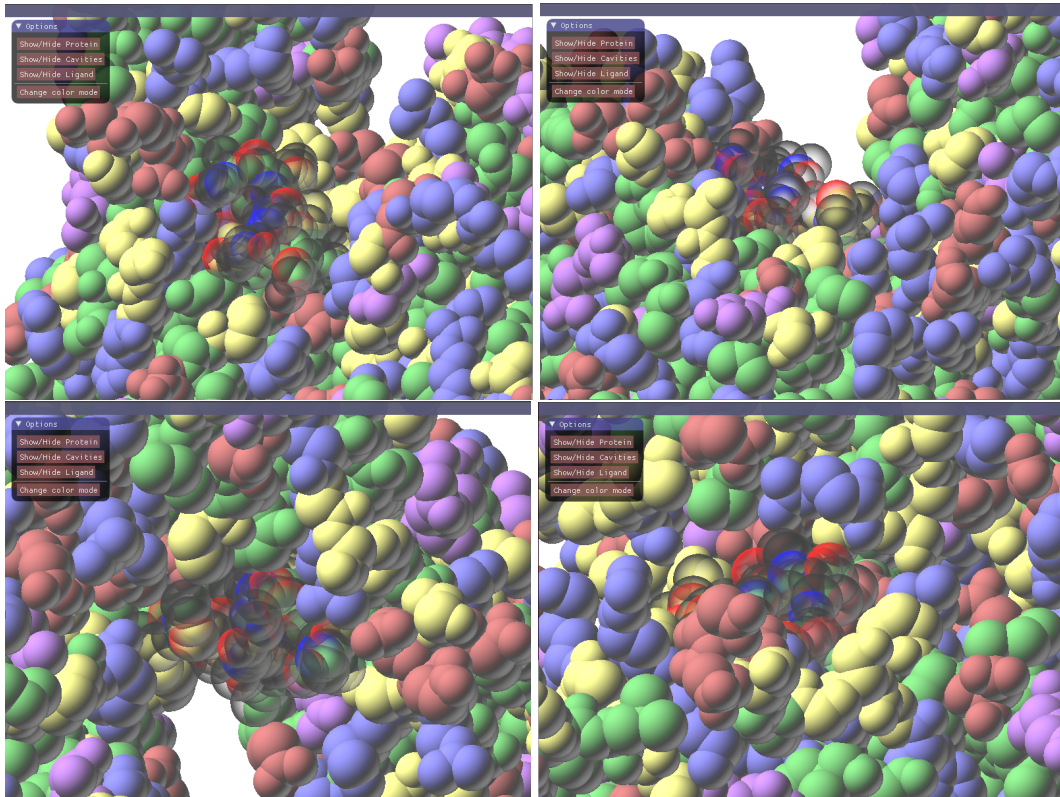
Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja

## 2.1.5

Persönliche Anmerkungen:

## 2.2 Protein und Ligand, Transparenz



### 2.2.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.2.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.2.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.2.4

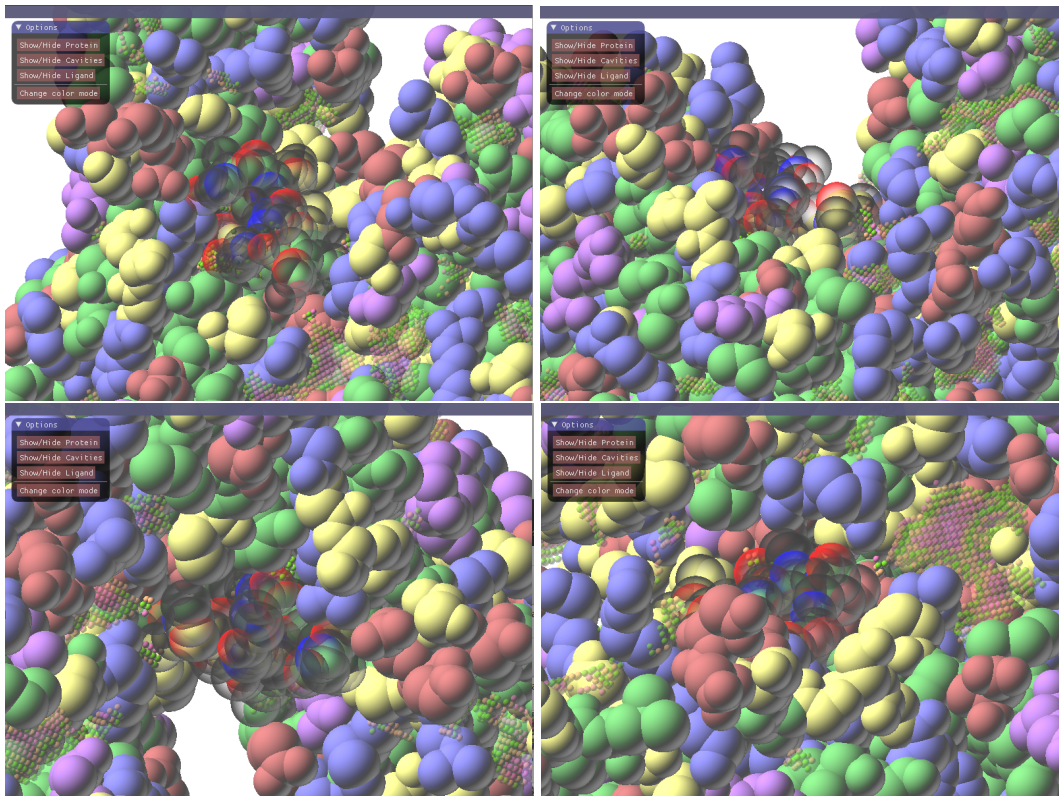
Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja

### 2.2.5

Persönliche Anmerkungen:

## 2.3 Protein und Ligand, Kavitätenpunkte



### 2.3.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.3.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.3.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.3.4

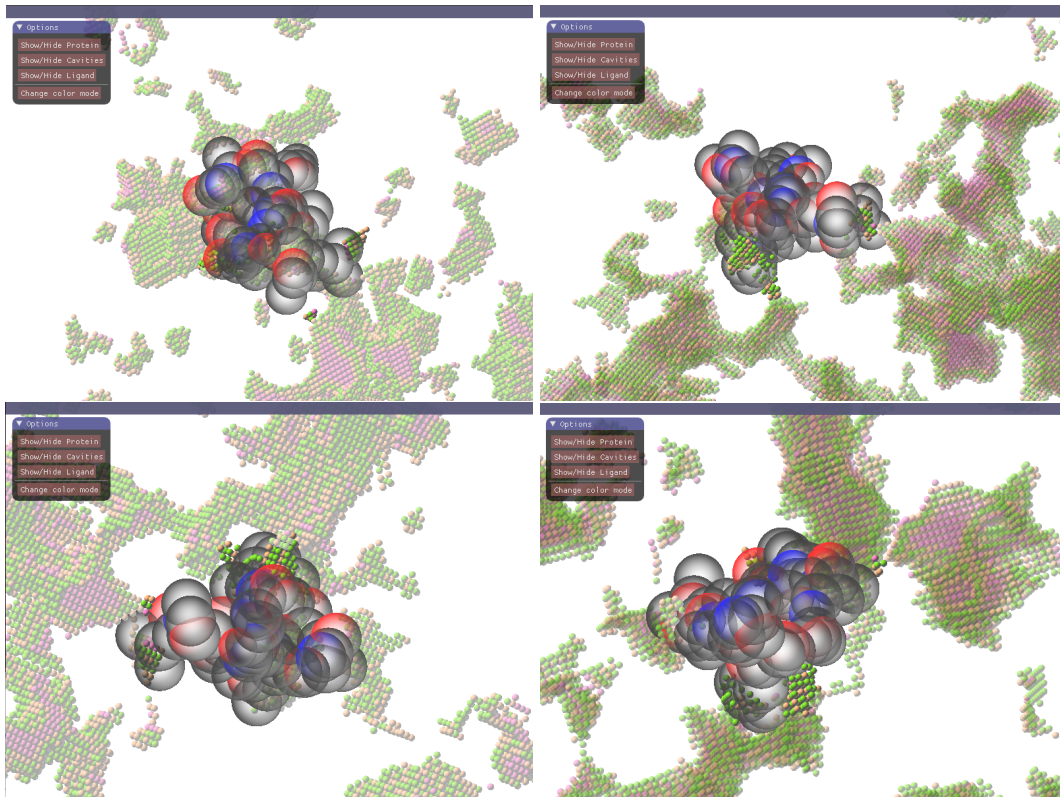
Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja

### 2.3.5

Persönliche Anmerkungen:

## 2.4 Protein(ausgeblendet) und Ligand, Kavitätenpunkte



### 2.4.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.4.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.4.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut

### 2.4.4

Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja

### 2.4.5

Persönliche Anmerkungen:

**3****3.0.1**

Wie hilfreich empfanden sie die Kavitätenvisualisierung für das Einschätzen der Bindungssituation?

sehr hilfreich	hilfreich	wenig hilfreich	gar nicht hilfreich

**3.0.2**

Option zur persönlichen Stellungnahme zum Grundgedanken und der Umsetzung von CavityDetector:



## CavityDetector - Evaluation

Vanessa Schüller

CavityDetector ist ein Tool, welches im Rahmen einer Bachelorarbeit entwickelt wurde, um Hohlräume eines Proteins zu visualisieren und zu beobachten, um Docking-Prozesse zwischen diesem und einem Liganden zu visualisieren und besser bewerten zu können.

Dazu werden die Kavitäten des Proteins mit Hilfe eines geometrischen Algorithmus berechnet und durch Punkte visualisiert, die durch ihre Farbgebung Aufschluss über den Abstand zum Protein liefern. Für verschiedene Bindungstypen gibt es jeweilige optimale Abstände, ab welchen eine solche Bindung zustande kommen kann. Diese werden durch den Benutzer definiert und durch grüne Punkte symbolisiert. Nicht-optimale Abstände wiederum werden durch rote Punkte dargestellt. Übergänge werden durch orangene Punkte visualisiert.

Durch verschiedene Tastaturbefehle kann ein Ligand hinzugefügt werden und beliebig verschoben werden. Betritt dieser eine Kavität, so wird der betretene Bereich ausgeschnitten.

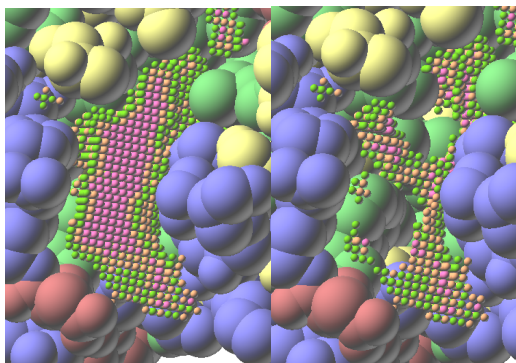


Abbildung 1: Kavität vor und nach Betreten eines Liganden

Hat ein Ligand, wie in der Abbildung, nun eine Position erreicht, in welcher er überwiegend an grüne Kavitätspunkte angrenzt, so kann davon ausgegangen werden, dass ein Docking erfolgreich sein könnte.

Im Folgenden wird Ihnen eine Bindungssituation aus verschiedenen Blickwinkeln auf verschiedene Weisen gezeigt. Sie werden gebeten, diese zu bewerten und eine Aussage darüber zu treffen, ob die dargestellte Konformation eine Bindung ermöglichen könnte oder nicht. Als optimal wurde in diesem Beispiel ein Abstand von 3.0 Å mit einer Toleranz von 0.2 Å angenommen, da dies ungefähr dem Abstand zwischen Atomen entspricht, die eine Wasserstoffbrücke bilden (2.8 - 3.2 Å)<sup>1</sup>.

Durch den definierten Abstand ergibt sich folgendes Bild einer Kavität:

Grüne Punkte symbolisieren optimale Abstände

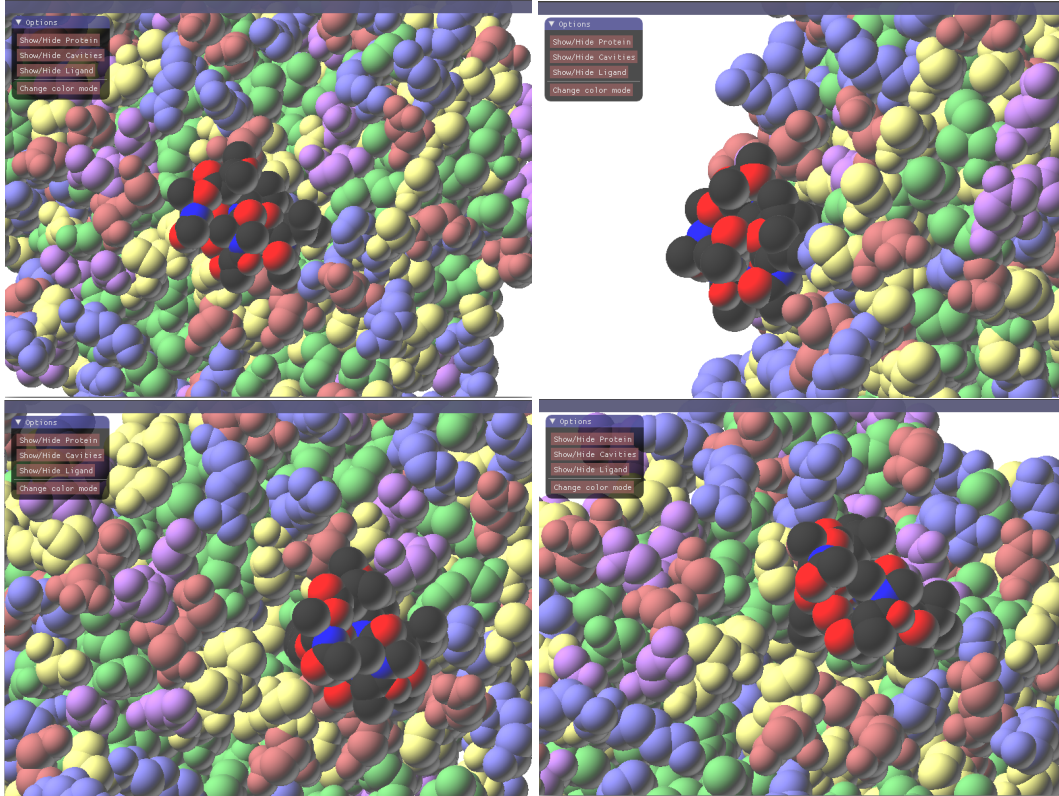
Orangene Punkte symbolisieren Abstände nahe dem optimalen Abstand

Rote Punkte symbolisieren nicht-optimale Abstände

<sup>1</sup>Gerhard Klebe - Wirkstoffdesign: Entwurf und Wirkung von Arzneistoffen

## 1

## 1.1 Protein und Ligand, einfach



## 1.1.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	2	1	0	0

## 1.1.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	3	0	0

## 1.1.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	3	0	0

## 1.1.4

Besitzt der Ligand einen optimalen Abstand zum Protein?

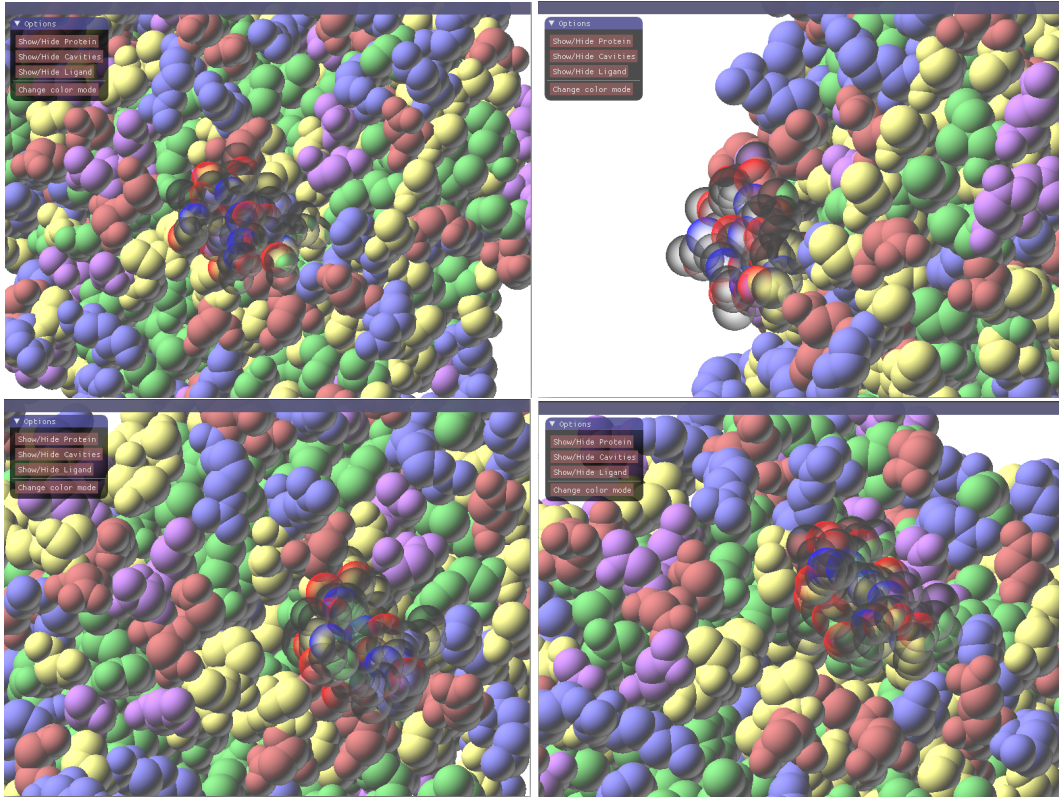
Nein	nicht ersichtlich	Ja
0	3	0

## 1.1.5

Persönliche Anmerkungen:

use contrasting single colour scheme for better visual impact

## 1.2 Protein und Ligand, Transparenz



### 1.2.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	2	1	0

### 1.2.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

### 1.2.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

### 1.2.4

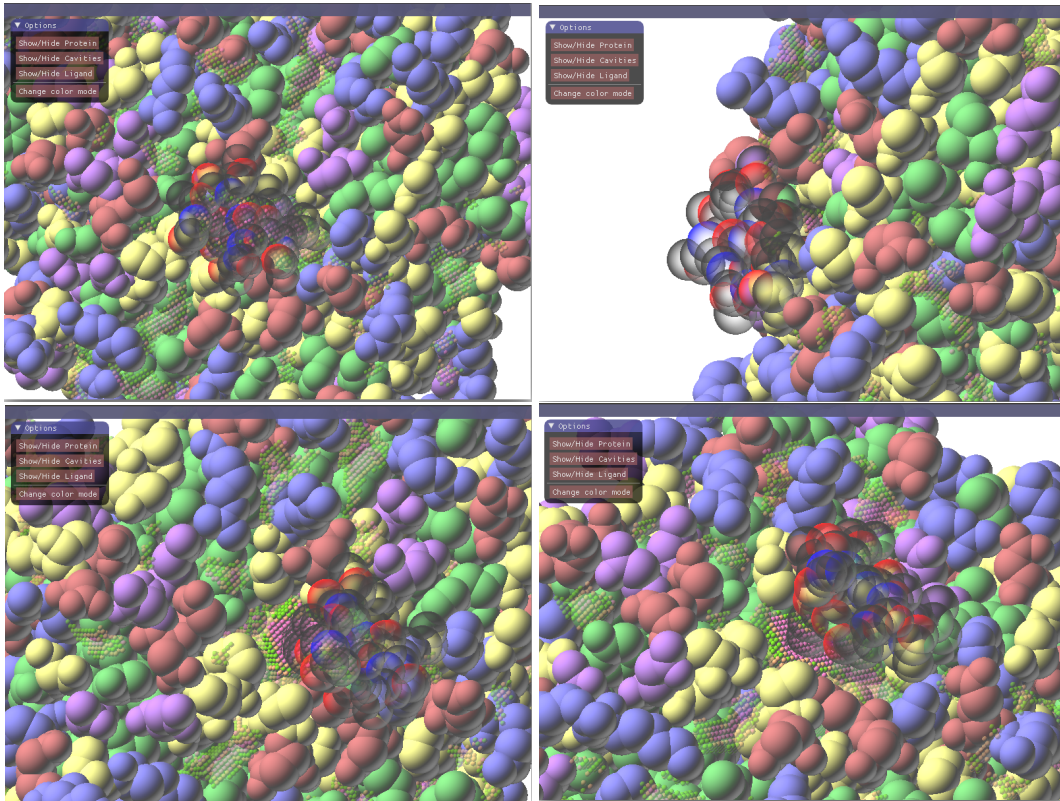
Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja
0	3	0

### 1.2.5

Persönliche Anmerkungen:

### 1.3 Protein und Ligand, Kavitätenpunkte



#### 1.3.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

#### 1.3.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	0	3	0

#### 1.3.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	0	3	0

#### 1.3.4

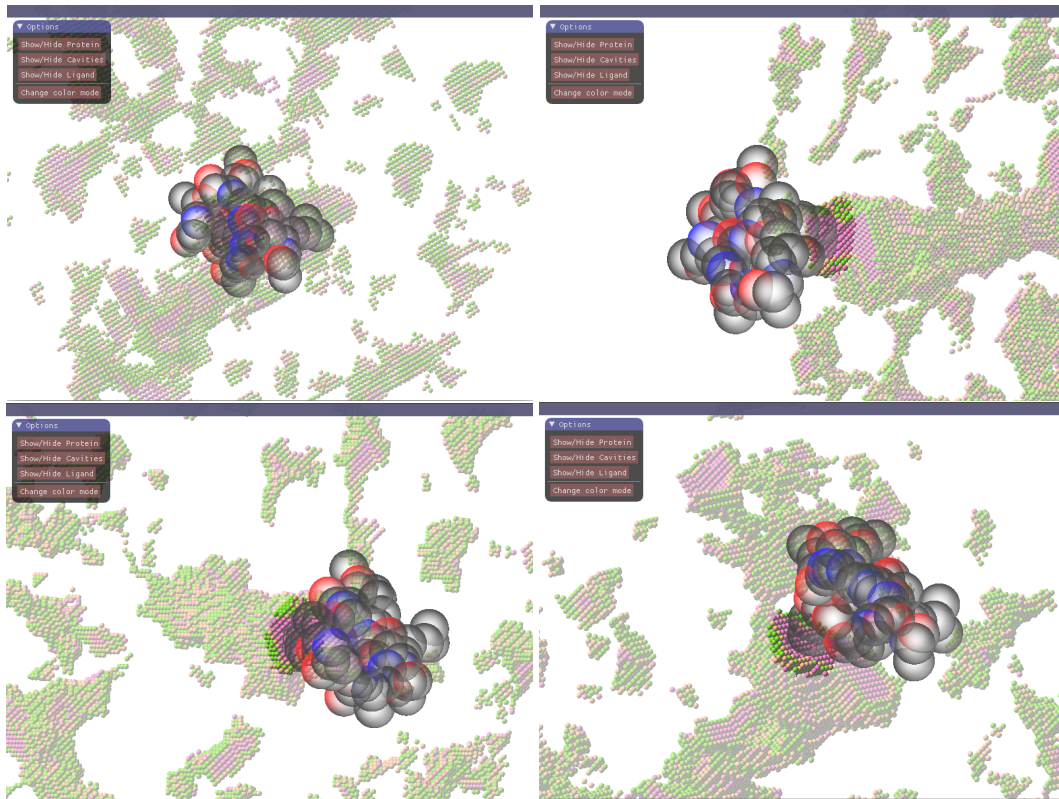
Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja
1	1	1

#### 1.3.5

Persönliche Anmerkungen:

## 1.4 Protein(ausgeblendet) und Ligand, Kavitätenpunkte



### 1.4.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	0	3	0

### 1.4.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	0	2	1

### 1.4.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	0	2	1

### 1.4.4

Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja
1	1	1

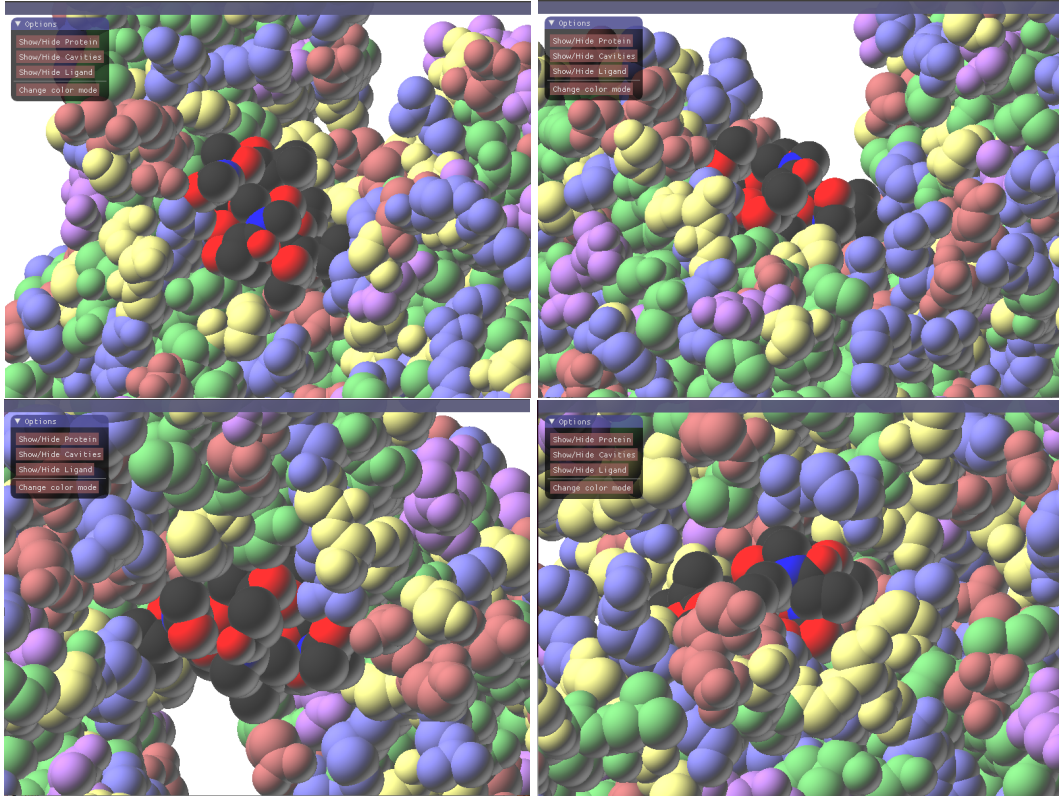
### 1.4.5

Persönliche Anmerkungen:

andere Kavitäten ausblenden

## 2

## 2.1 Protein und Ligand, einfach



## 2.1.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

## 2.1.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

## 2.1.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

## 2.1.4

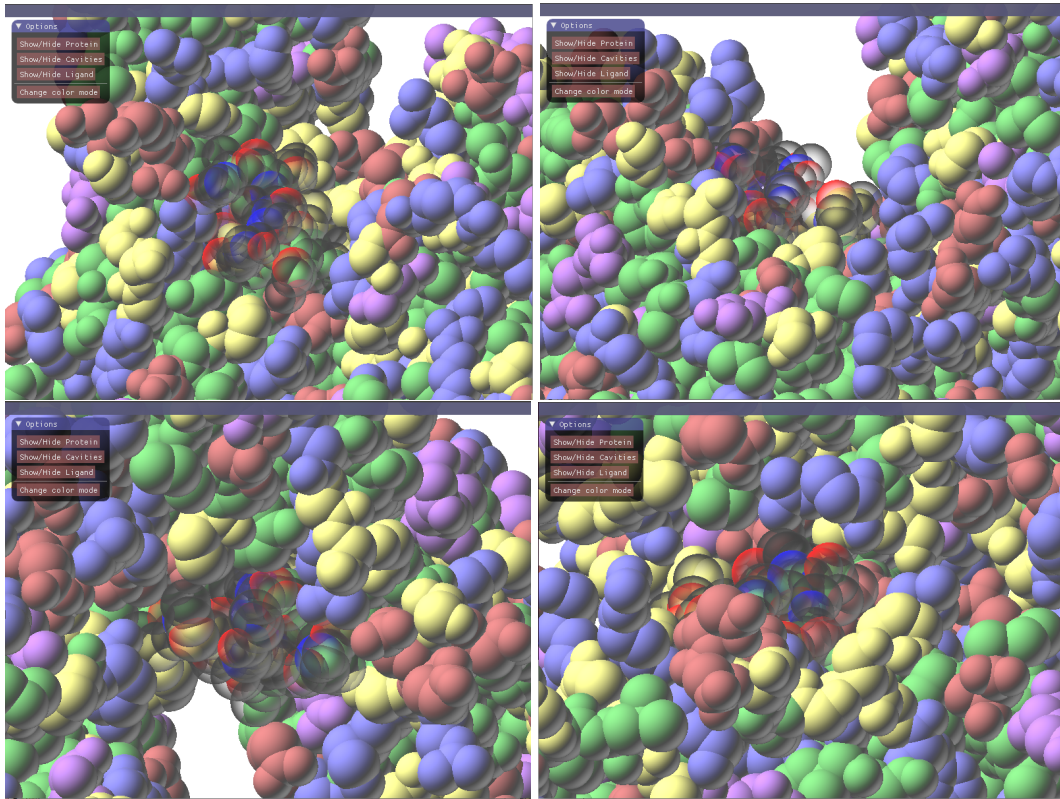
Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja
0	2	1

## 2.1.5

Persönliche Anmerkungen:

## 2.2 Protein und Ligand, Transparenz



### 2.2.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

### 2.2.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

### 2.2.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

### 2.2.4

Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja
0	2	1

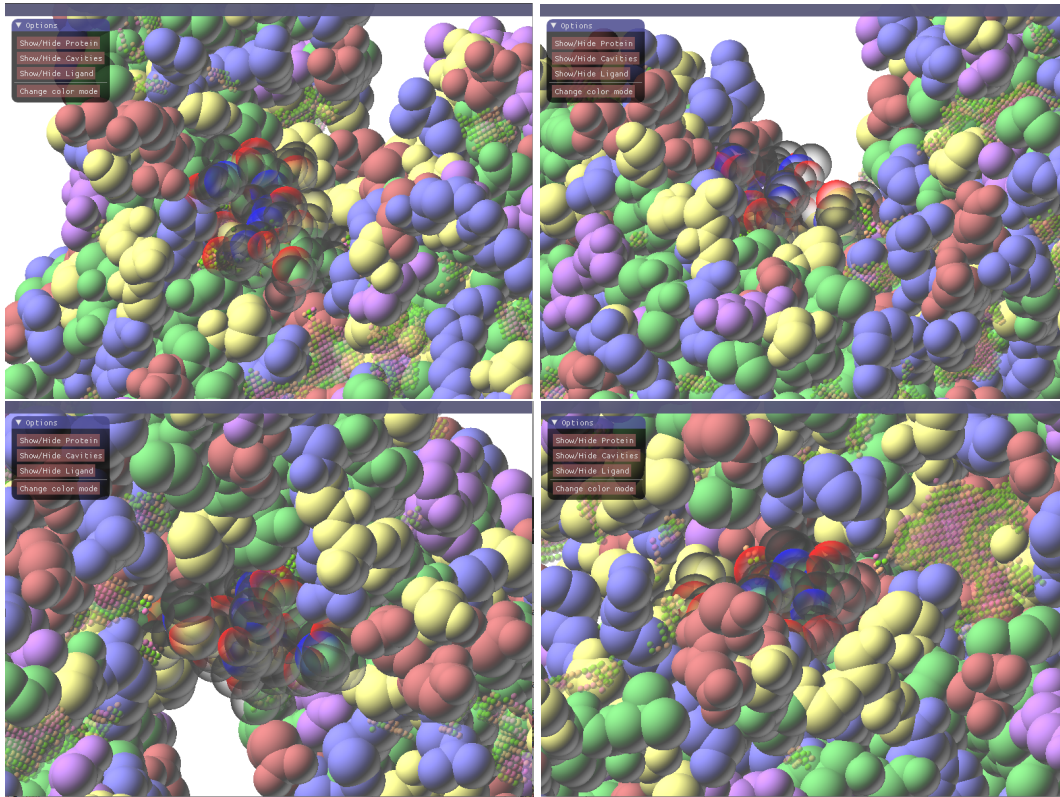
### 2.2.5

Persönliche Anmerkungen:

kritisch: optimaler Abstand zum Protein ist nicht gleich optimaler Aktivität

mono colored for protein/ligand

## 2.3 Protein und Ligand, Kavitätenpunkte



### 2.3.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

### 2.3.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

### 2.3.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	1	2	0

### 2.3.4

Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja
0	2	1

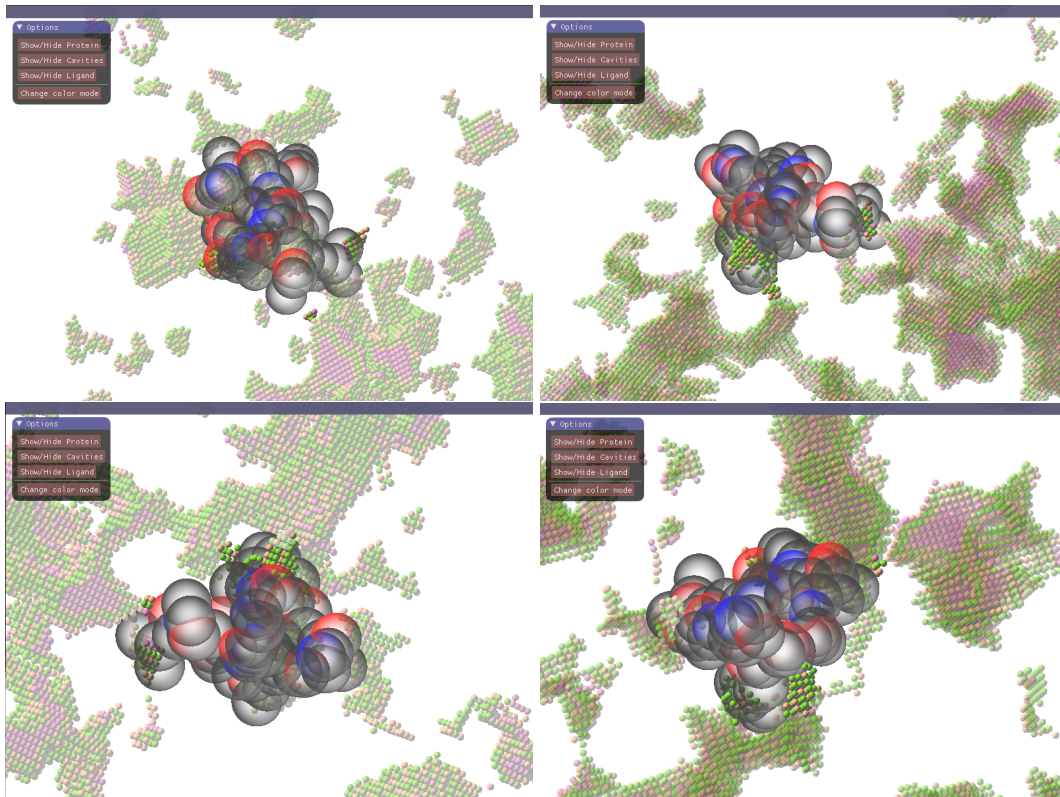
### 2.3.5

Persönliche Anmerkungen:

zu voll



## 2.4 Protein(ausgeblendet) und Ligand, Kavitätenpunkte



### 2.4.1

Wie gut können Sie die Bindungssituation zwischen Protein und Ligand räumlich einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	0	1	2

### 2.4.2

Wie gut ist ersichtlich, ob der Ligand in der Bindungstasche liegt??

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	0	1	2

### 2.4.3

Wie gut können Sie den Abstand zwischen Protein und Ligand einschätzen?

sehr schlecht	schlecht	mittelmäßig	gut	sehr gut
0	0	0	1	2

### 2.4.4

Besitzt der Ligand einen optimalen Abstand zum Protein?

Nein	nicht ersichtlich	Ja
0	0	3

### 2.4.5

Persönliche Anmerkungen:

andere Kavitäten transparent, Farbgradient nicht rot-grün-rot, eher kompletter Gradient zu nah-optimal-zuweit

Hintergrund blasser, direkte Interaktionen hervorheben, direkter Kontakt in Vergleich zu MD-Screenshots, Label AA in Protein

### 3

#### 3.1

Wie hilfreich empfanden sie die Kavitätenvisualisierung für das Einschätzen der Bindungssituation?

sehr hilfreich	hilfreich	wenig hilfreich	gar nicht hilfreich
0	3	0	0

#### 3.2

Option zur persönlichen Stellungnahme zum Grundgedanken und der Umsetzung von CavityDetector:

Vergleich zu MD wäre gut, da sonst Orientierung von Protein und Ligand nicht erkennbar

## Glossar

### Å

Ångström - Etablierte und typische Größenordnung für Abstände in atomaren Größenordnungen - entspricht 10 pm

### Bounding Box

Box, die ein Objekt vollständig umschließt und dabei minimal ist

### Fragment Shader

Shaderstufe, die einzelne Fragmente behandelt und Farben zuweist

### Geometry Shader

Shaderstufe, die einzelne Primitive (Dreiecke, Linien, Quads) behandelt

### Multiprocessing

Ausführen einzelner Prozesse auf verschiedenen Prozessorkernen

### Multithreading

Paralleles Abarbeiten mehrerer Threads innerhalb eines Programmes

### Out-of-Range-Exceptions

Fehler, der vom Compiler geworfen wird, wenn ein Index auf einen nicht definierten oder ungültigen Bereich zeigt

### Render-To-Texture

Szene wird nicht in den Bildschirm, sondern in eine Textur gerendert, sodass diese zur Weiterverarbeitung genutzt werden kann

### Shader Storage Buffer Object

Buffer Objekte, um große Datenmengen zu speichern und an Shader weiterzugeben - SSBOs sind beschreibbar und können eine variable Speicher Größe haben

### std-Vector

Container, der dynamische Arrays kapselt

### Tiefensuche

Verfahren zur Suche von Knoten in Graphen, wobei Pfade vollständig in die Tiefe abgearbeitet werden, bevor ein neuer beschritten wird

### Vertex Shader

Shaderstufe, die einzelne Vertices behandelt

## Literatur

- [1] An & Totrov & Abagyan. *Pocketome via comprehensive identification and classification of ligand binding envelopes*. 2005. URL: <https://www.ncbi.nlm.nih.gov/pubmed/15757999?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).
- [2] Hendlich & Rippmann & Barnickel. *LIGSITE: automatic and efficient detection of potential small molecule-binding sites in proteins*. 1997. URL: <https://www.ncbi.nlm.nih.gov/pubmed/9704298?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).
- [3] Armon & Grar & Ben-Tal. *ConSurf: an algorithmic tool for the identification of functional regions in proteins by surface mapping of phylogenetic information*. 2001. URL: <https://www.ncbi.nlm.nih.gov/pubmed/11243830?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).
- [4] accelero bioanalytics. *The Ligand Binding Assay*. 2011-2017. URL: <http://accelero-bioanalytics.com/corporate/science/ligand-binding-assays>. (Aufgerufen am 3. Feb. 2017).
- [5] Biokurs. *Bau von Proteinen - Peptidbindung und Raumstrukturen Teil III*. 2007. URL: <http://www.biokurs.de/skripten/bs11-10.htm>. (Aufgerufen am 3. Feb. 2017).
- [6] Uni Bochum. *Detektion von Protein/Protein-Interaktionen mit Hilfe des Yeast Two-Hybrid Systems*. 2013. URL: [http://www.ruhr-uni-bochum.de/ag-hovemann/mam/content/lehre/bachelor/yeast\\_two-hybrid\\_system\\_\\_2013.pdf](http://www.ruhr-uni-bochum.de/ag-hovemann/mam/content/lehre/bachelor/yeast_two-hybrid_system__2013.pdf). (Aufgerufen am 3. Feb. 2017).
- [7] Wikipedia Die freie Enzyklopädie. *Accessible surface area*. 2016. URL: [https://en.wikipedia.org/wiki/Accessible\\_surface\\_area](https://en.wikipedia.org/wiki/Accessible_surface_area). (Aufgerufen am 3. Feb. 2017).
- [8] Wikipedia Die freie Enzyklopädie. *Docking*. 2013. URL: [https://de.wikipedia.org/wiki/Docking\\_\(Chemie\)](https://de.wikipedia.org/wiki/Docking_(Chemie)). (Aufgerufen am 20. Feb. 2017).
- [9] Wikipedia Die freie Enzyklopädie. *Line-sphere intersection*. 2015. URL: [https://en.wikipedia.org/wiki/Line%E2%80%93sphere\\_intersection](https://en.wikipedia.org/wiki/Line%E2%80%93sphere_intersection). (Aufgerufen am 3. Feb. 2017).
- [10] Wikipedia Die freie Enzyklopädie. *OpenGL*. 2017. URL: <https://de.wikipedia.org/wiki/OpenGL>. (Aufgerufen am 3. Feb. 2017).
- [11] Wikipedia Die freie Enzyklopädie. *OpenGL Shading Language*. 2016. URL: [https://de.wikipedia.org/wiki/OpenGL\\_Shading\\_Language](https://de.wikipedia.org/wiki/OpenGL_Shading_Language). (Aufgerufen am 3. Feb. 2017).
- [12] Wikipedia Die freie Enzyklopädie. *Protein*. 2017. URL: <https://de.wikipedia.org/wiki/Protein>. (Aufgerufen am 1. Feb. 2017).

- [13] Wikipedia Die freie Enzyklopädie. *Protein-Protein-Interaktion*. 2016. URL: <https://de.wikipedia.org/wiki/Protein-Protein-Interaktion>. (Aufgerufen am 20. Feb. 2017).
- [14] Wikipedia Die freie Enzyklopädie. *Proteinstruktur*. 2016. URL: <https://de.wikipedia.org/wiki/Proteinstruktur>. (Aufgerufen am 1. Feb. 2017).
- [15] Uni Erlangen. *Van der Waals-Oberfläche*. 2003. URL: [http://www2.chemie.uni-erlangen.de/projects/vsc/chemoinformatik/erlangen/struktur\\_rep/oberflaeche/vanderwaals.html](http://www2.chemie.uni-erlangen.de/projects/vsc/chemoinformatik/erlangen/struktur_rep/oberflaeche/vanderwaals.html). (Aufgerufen am 3. Feb. 2017).
- [16] DocCheck Flexikon. *Titin*. 2017. URL: <http://flexikon.doccheck.com/de/Titin>. (Aufgerufen am 3. Feb. 2017).
- [17] Huang. *MetaPocket: a meta approach to improve protein ligand binding site prediction*. 2009. URL: <https://www.ncbi.nlm.nih.gov/pubmed/19645590?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).
- [18] Cosmos Indirekt. *Oberflächenplasmonenresonanzspektroskopie*. 2016. URL: <http://physik.cosmos-indirekt.de/Physik-Schule/Oberflächplasmonenresonanzspektroskopie>. (Aufgerufen am 3. Feb. 2017).
- [19] Lorient S & Cazals F & Bernauer J. *ESBTL: efficient PDB parser and data structure for the structural and geometric analysis of biological macromolecules*. 2010. URL: <https://academic.oup.com/bioinformatics/article/26/8/1127/206699/ESBTL-efficient-PDB-parser-and-data-structure-for>. (Aufgerufen am 3. Feb. 2017).
- [20] Laurie & Jackson. *Q-SiteFinder: an energy-based method for the prediction of protein-ligand binding sites*. 2005. URL: <https://www.ncbi.nlm.nih.gov/pubmed/15701681?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).
- [21] Kleywegt & Jones. *Detection, delineation, measurement and display of cavities in macromolecular structures*. 1994. URL: <http://scripts.iucr.org/cgi-bin/paper?pii=S0907444993011333&sentby=wiley>. (Aufgerufen am 3. Feb. 2017).
- [22] Gerhard Klebe. *4.4 Wichtige Typen von Protein-Ligand-Wechselwirkungen*. 2009.
- [23] Structural bioinformatics protein crystallography sequence analysis & homolog modeling. *The Protein Databank (PDB): File Format and Content*. 2010-2016. URL: <http://www.proteinstructures.com/Structure/Structure/proteinstructure-databases2.html>. (Aufgerufen am 3. Feb. 2017).

- [24] Moloc. *Molecular Surfaces*. URL: <http://www.msg.ucsf.edu/local/programs/moloc/surface.html>. (Aufgerufen am 3. Feb. 2017).
- [25] Technische Universität München. *Die "dunkle Materie" im Protein-Universum*. 2015. URL: <http://www.tum.de/die-tum/aktuelles/pressemitteilungen/lang/article/32762/>. (Aufgerufen am 1. Feb. 2017).
- [26] Volkamer & Griewel & Grombacher & Rarey. *Analyzing the topology of active sites: on the prediction of pockets and subpockets*. 2010. URL: <https://www.ncbi.nlm.nih.gov/pubmed/20945875?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).
- [27] Thomas Lundbäck Pär Nordlund Daniel Martinez Molina Rozbeh Jafarim Helena Almqvist Hanna Axelsson Marina Ignatushchenko. *The cellular thermal shift assay for evaluating drug target interactions in cells*. 2014. URL: <http://www.nature.com/nprot/journal/v9/n9/full/nprot.2014.138.html>. (Aufgerufen am 3. Feb. 2017).
- [28] Weisel & Proschka & Schneider. *PocketPicker: analysis of ligand binding-sites with shape descriptors*. 2007. URL: <https://www.ncbi.nlm.nih.gov/pubmed/17880740?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).
- [29] Oliveira & Ferraz & Honorato & Xavier-Neto & Sobreira. *KVFinder: steered identification of protein cavities as a PyMOL plugin*. 2014. URL: <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-15-197>. (Aufgerufen am 3. Feb. 2017).
- [30] Spektrum. *Künstliches Protein hält HIV bei Affen in Schach*. 2015. URL: <http://www.spektrum.de/news/kuenstliches-protein-haelt-hiv-bei-affen-in-schach/1333183>. (Aufgerufen am 1. Feb. 2017).
- [31] Spektrum. *Protein-Liganden-Komplex*. 1999. URL: <http://www.spektrum.de/lexikon/biologie/protein-liganden-komplex/54156>. (Aufgerufen am 3. Feb. 2017).
- [32] Guilloux & Schmidtke & Tuffery. *Fpocket: an open source platform for ligand pocket detection*. 2009. URL: <https://www.ncbi.nlm.nih.gov/pubmed/19486540?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).
- [33] UbuntuUsers. *CMake*. 2013. URL: <https://wiki.ubuntuusers.de/CMake/>. (Aufgerufen am 3. Feb. 2017).
- [34] UbuntuUsers. *QT Creator*. 2016. URL: [https://wiki.ubuntuusers.de/Qt\\_Creator/](https://wiki.ubuntuusers.de/Qt_Creator/). (Aufgerufen am 3. Feb. 2017).
- [35] PyMol Wiki. *Plugins*. 2015. URL: <https://pymolwiki.org/index.php/Category:Plugins>. (Aufgerufen am 3. Feb. 2017).

- [36] Wikipedia. *Molekulares Display*. 2016. URL: [https://de.wikipedia.org/wiki/Molekulares\\_Display](https://de.wikipedia.org/wiki/Molekulares_Display). (Aufgerufen am 3. Feb. 2017).
- [37] Yu & Zhou & Tanaka & Yao. *Roll: a new algorithm for the detection of protein pockets and cavities with a rolling probe sphere*. 2010. URL: <https://www.ncbi.nlm.nih.gov/pubmed/19846440?dopt=Abstract>. (Aufgerufen am 3. Feb. 2017).